UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Integrarea tehnologiei ESP32 în construirea unei drone

propusă de

Iustin Bilan

Sesiunea: iunie, 2023

Coordonator științific

Conf. Vârlan Cosmin

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Integrarea tehnologiei ESP32 în construirea unei drone

Iustin Bilan

Sesiunea: iunie, 2023

Coordonator științific

Conf. Vârlan Cosmin

Cuprins

Introducere

Această lucrare de licență prezintă generalitățile, metodele utilizate și etapele urmate în construirea unei drone terestre utilizând tehnologia ESP32, controlată prin intermediul unei aplicații mobile dedicate.

Drona cu model cilindric cu 2 roți a fost anterior explorată în domeniul militar sub programul "Defense Advanced Research Projects Agency" (DARPA) "Tactical Mobile Robots" (TMR) în anul 1999, sub numele de "ThrowBot". Scopul acestui proiect a fost dezvoltarea unui dispozitiv mobil capabil să fie aruncat și să supravegheze în misiuni militare. Modelul "Two-Wheeled Cylinder" a fost abandonat din cauza capacității sale limitate de a naviga pe teren accidentat sau de a urca pante abrupte.

Cu toate acestea, acest model de dronă prezintă avantaje precum capacitatea de a fi aruncată pentru lansare și ușurința în a se reașeza în poziție. Numărul redus de roți și dimensiunile compacte duce la un *design* mai puțin greoi.

Inspirată de proiectul "ThrowBot", această teză are ca scop explorarea și implementarea unui sistem integrat utilizând tehnologia ESP32 pentru controlul unei drone prin intermediul unei aplicații mobile dedicate.

Capitolul 1, **Aspecte generale**, va explora aspecte fundamentale despre microprocesoare, plăcile de dezvoltare ESP32 și protocoalele de comunicare precum TCP/IP și HTTP; de asemenea, funcționarea și rolul motoarelor DC, al modulului driver în cadrul sistemului și concepte de alimentare, precum și bibliotecile folosite.

Capitolul 2, **Construirea sistemului integrat**, se va concentra pe procesul de implementare și construire a dronei. Se va discuta despre componentele specifice utilizate în construcția dronei. Se va explora, de asemenea, configurarea nivelului hardware și software pentru a asigura funcționarea corectă a sistemului. În final, vom aborda procesul de testare a sistemului pentru a verifica funcționalitatea și performanța acestuia.

Capitolul 1

Aspecte generale

În era tehnologiei avansate, sistemele integrate au devenit o componentă indispensabilă în numeroase aplicații și dispozitive, inclusiv în domeniul robotizării. Un sistem integrat reprezintă o combinație sinergică de hardware și software care funcționează ca o entitate autonomă într-un mediu specific. În acest capitol, ne vom concentra asupra aspectelor generale ale sistemelor încorporate și modul în care acestea funcționează în drona noastră.

1.1 Introducere în microcontrolere

Înainte de toate, la baza multor sisteme integrate stă un microprocesor sau microcontroler; este esențial de știut ce fel de tehnologii utiliează acestea și cum sunt structurate.

Un microcontroler este un procesor echipat cu memorie, temporizatoare, pini I/O și alte periferice integrate pe cip. Acesta oferă avantaje în ceea ce privește costurile datorită integrării tuturor elementelor pe un singur cip, economisind spațiu și ducând la costuri de producție mai mici și timpi de dezvoltare mai scurți. Microcontrolerele sunt utilizate în mod obișnuit în sistemele integrate datorită capacității lor de a fi actualizate ușor, consumului redus de energie și fiabilității ridicate.

1.1.1 Arhitectura microcontrolerelor

CPU, sau Unitatea Centrală de Procesare, este nucleul arhitecturii unui microcontroler. Aceasta constă din calea de date și unitatea de control. Calea de date execută instrucțiuni în timp ce unitatea de control îi spune ce să facă. Unitatea aritmetică logică

(ALU) se află la baza CPU și efectuează calcule precum AND, ADD, INC, etc. ALU primește două intrări și returnează rezultatul operației ca ieșire. Sursa și destinația sunt preluate din registre sau din memorie. În plus, ALU stochează informații despre natura rezultatului într-un registru de stare.

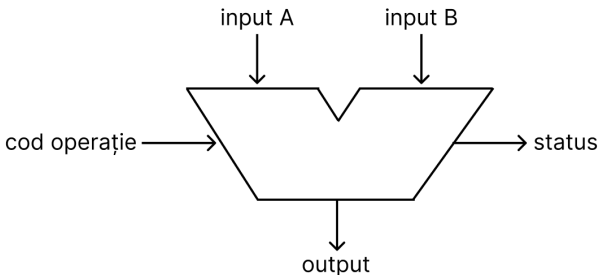

Figura 1.1: ALU

Componentele de memorie sunt o parte esențială a oricărui dispozitiv electronic. Există două tipuri principale de memorie: volatilă și nevolatilă.

Memoria volatilă își pierde conținutul de îndată ce sistemul este oprit, în timp ce memoria nevolatilă își păstrează conținutul chiar și atunci când alimentarea este oprită. În memoria volatilă, există două tipuri: statică și dinamică. *Dynamic Random Access Memory* (DRAM) este un tip de memorie volatilă care stochează informații în condensatoare, care își pot pierde încărcarea în timp din cauza curenților de scurgere. Pentru a preveni această pierdere de informații, DRAM-ul trebuie accesat la fiecare câteva milisecunde pentru a reîmprospăta încărcarea condensatorilor. În timp ce DRAM are o capacitate de stocare mai mare decât SRAM, este mult mai lentă și necesită mai mulți pini externi pentru adresare. Static Random Access Memory (SRAM) este un alt tip de memorie volatilă. Spre deosebire de DRAM, SRAM stochează informații în *flip-flops* formate din mai mulți tranzistori. Asta înseamnă că SRAM nu necesită reîmprospătare ca DRAM și poate accesa datele mult mai rapid. Cu toate acestea, SRAM este mai scump și are o capacitate de stocare mai mică decât DRAM. Un alt tip de memorie este PSRAM (Pseudo-Static RAM) ce este o formă de DRAM asemănătoare cu SRAM, celulele de memorie fiind dezvoltate la fel ca la DRAM, dar dispune de un circuit integrat ce face reîmprospătare automată a încărcării condensatorilor, deci, ca la SRAM, microprocesorul nu trebuie să reîmprospăteze manual memoria.

Memoria nevolatilă este un tip de memorie care își păstrează conținutul chiar și atunci când se oprește alimentarea de curent. Există mai multe tipuri de memorie volatilă, printre care:

- *Read-Only Memory* (ROM): Acest tip de memorie este programat în timpul producției și nu poate fi schimbat sau șters de către utilizator. Este folosit în mod obișnuit pentru a stoca *firmware* și alte date esențiale ale sistemului.
- *Programmable Read-Only Memory* (PROM): Poate fi programat de utilizator o singură dată, după care conținutul devine permanent și nu mai poate fi modificat.
- *Erasable Programmable Read-Only Memory* (EPROM): Conținutul poate fi șters folosind lumină ultravioletă și după reprogramat cu noi date.
- *Electrically Erasable Programmable Read-Only Memory* (EEPROM): Poate fi ștearsă folosind un impuls de tensiune.
- Memorie *flash*: Similară cu EEPROM dar cu capacitate mai mare și timpi de acces mai rapid

Componentele digitale I/O (*Input/Output*) reprezintă una din caracteristicile principale ale microprocesoarelor. Valoarea unui pin poate fi 1 sau 0, unde în general 1 corespunde unei stări "high" iar 0 unei stări "low". Trei registre controlează cum se comportă pinii digitali I/O: *Data Direction Register* (DDR), *Port Register* (PORT) și *Port Input Register* (PIN). Registrul DDR determină dacă un pin este configurat ca input sau output prin setarea sau ștergerea unui bit în DDR, registrul PORT controlează nivelul de voltaj al piniilor de output, iar registrul PIN este de tip *read-only* și conține stările (high sau low) tuturor pinilor.

Componenta analogă I/O a unui microprocesor se ocupă cu convertirea unui semnal analog în unul digital și invers. Microprocesorul funcționează în totalitate cu semnale digitale, deci aceasta permite evaluarea semnalelor analogice. O conversie din digital în analog (DAC) se poate face utilizând tehnica *Pulse Width Modulation* (PWM) în care se trimit semnale high numai într-un interval din ciclul de lucru.

Conversia analog/digital (ADC) este procesul convertirii unui input analog aflat într-o anumită gamă de voltaj în cuvinte de cod digital. Input-ul analog este divizat în $2r$ clase, unde r reprezintă numărul de biți folosiți în a reprezenta valoarea digitală. Fiecare clasă corespunde unui cuvânt de cod digital de la 0 la $2r - 1$. Rezoluția multor convertoare este de obicei 8 sau 10 biți. Cea mai mică diferență de voltaj ce poate fi distinsă în mod fiabil de un convertor este reprezentată de bitul cel mai puțin semnificativ a valorii digitale.

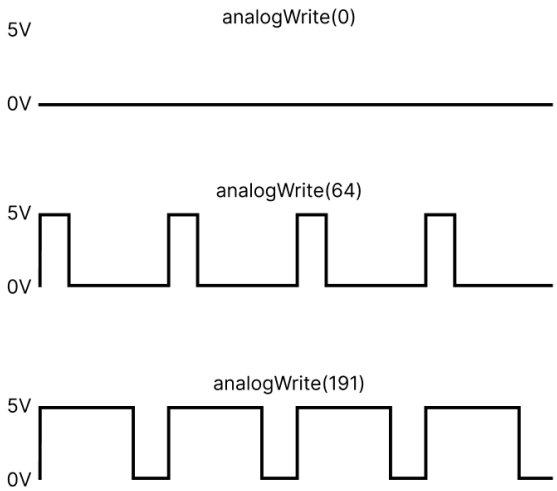

Figura 1.2: Pulse Width Modulation

1.1.2 Interfețe de comunicare

Interfețele de comunicare sunt metode de a transmite date între dizpozitive. Ele pot fi în mod sincron sau asincron, și pot folosi comunicare serială sau paralelă.

În comunicarea sincronă, datele sunt trimise într-o manieră sincronizată, expeditorul și receptorul au un semnal de ceas sincronizat pentru a coordona transferul de date. Datele sunt trimise la cicluri de ceas predefinite, operând pe baza aceluiași semnal de ceas, se asigură trimiterea și primirea datelor la aceași rată.

În comunicare asincronă, datele sunt transmise fără un ceas sincronizat, în loc, se folosesc la fiecare cadru de date biți de început și sfârșit pentru a stabili limitele frontierelor fiecărei transmisii. Expeditorul și receptorul nu trebuie să aibă un semnal de ceas sincronizat, datele se transmit în intervale neregulate. Oferă mai multă simplitate și flexibilitate, un exemplu de comunicare asincronă este protocolul UART (*Universal Asynchronous Receiver-Transmitter*).

Comunicarea serială este o metodă de transmitere de date într-o manieră secvențială, bit cu bit. Această abordare necesită numai o singurp linie de date, fiind eficient din punct de vedere al resurselor; un dezavantaj este că numărul de biți ce poate fi transmis este scăzut.

Comunicarea paralelă utilizează linii de date multiple pentru a transfera mai mult de un bit simultan. Numărul de biți ce pot fi trimiți poate varia, dar de obicei se folosesc lungimi de 4 sau 8 biți.

Protocolul UART este un protocol de comunicare serială asincronă unde se folosesc două fire, unul pentru transmisie (TX) și unul pentru receptare (RX). Din moment

ce comunicarea este asincronă, datele sunt transmise în pachete (de obicei un byte), fiecare pachet are un bit de start și unul sau mai mulți biți de sfârșit. Transmisia datelor se face cu un specific *baud rate*, ce determină rata în care datele sunt transmise și recepționate, măsurată în biți/secundă. Atât expeditorul cât și receptorul trebuie configurate similar în ceea ce privește *baud rate* și formatul de date (numărul de biți de date, biți de start/sfârșit) pentru a asigura un transfer corect de date.

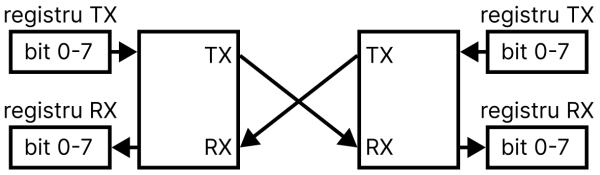

Figura 1.3: Comunicare UART

În protocolul UART mesajele sunt transmise utilizând nivelul de voltaj pentru a exprima valoarea unui bit, unde 1 reprezintă ”high”, iar 0 ”low”. Pe parcursul unei stări inactive, linia de transmisie va fi mereu ”high”, bitul de start al unui pachet este 0, deci când se detectează o cădere, atunci se va ști că începe să vină un pachet de date, iar bitul sau biții de sfârșit vor fi 1.

O metodă de a îmbunătăți acuratețea comunicării UART este supraeșantionarea (*oversampling*), această tehnică eșantionează datele primite la o frecvență sporită, cel mai uzual se eșantionează de 16 ori pe bit, această rată de eșantionarea ajută la determinarea sincronizării fiecărui bit și reduce șansele de eroare cauzate de zgomot, denaturări semnal sau variații de timp. În plus, receptorul își sincronizează ceasul o dată cu semnalele primite, astfel supraeșantionarea permite o sincronizare mai precisă.

Serial Peripheral Interface (SPI) este o comunicare serială sincronă ce funcționează pe baza relației dintre un *master* și un *slave* (sau mai mulți), în care dizpozitivul *master* inițiază și controlează transferul de date. Folosește linii separate pentru transfer, inclusiv:

- *Master Output Slave Input* (MOSI): *master* trimite date către *slave* prin această linie.
- *Master Input Slave Output* (MISO): *slave* trimite date înapoi către *master* prin această linie.
- *Serial Clock* (SCK): semnalul ceas care este generate de către *master* ce sincronizează transferul de date.

- *Slave Select* (SS): această linie este utilizată de *master* ca să selecteze un *slave*.

SPI funcționează pe baza a două registre, dispozitivul *master* și *slave* au un registru de fiecare, pe baza fiecărui ciclu de ceas al dispozitivului *master*, se transmite cel mai simnificativ bit al *master-ului* către *slave* prin linia MOSI și este șiftat în registrul *slave* ca cel mai puțin sesimnificativ bit, în același timp dispozitivul *slave* trimite prin linia MISO bitul cel mai semnificativ șiftat în registrul *master* ca cel mai puțin semnificativ; după 8 astfel de cicluri de ceas, s-a transmis întreg byte-ul.

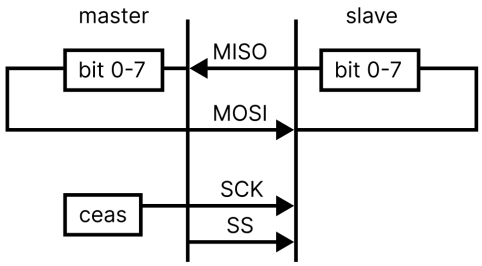

Figura 1.4: Comunicare SPI

Inter-Integrated Circuit (I2C) este un protocol de comunicare serială sincronă în care se folosește două linii de transimisie, o linie bidirecțională SDA (*Serial Data*) unde se transmite mesajele și o linie de transmitere semnal ceas SCL (*Serial Clock*). Funcționează tot ca SPI pe bază de *master* și *slave*, unde se pot conecta mai multe dispozitive *slave* pe magistrale. Acest protocol distinge mai multe moduri de viteză: mod implicit sau standard (100 kbps), *fast mode* (400 kbps), *high speed mode* (3.4 Mbps).

Linia SDA în mod inactiv este pe mod ”high” sau 1. Datele sunt transmise sub formă de mesaje, mesajele conțin biți de început și sfârșit (bitul de început este 0 pentru a semnifica începerea unui mesaj, iar cel de sfârșit 1), adresa dispozitivului *slave*, un bit de read/write (unde 0 înseamnă că se trimit date, iar 1 înseamnă că se cer date), biți de confirmare (unde se află după fiecare cadru al mesajului, unde se trimit înapoi către expeditor pentru a confirma că s-au primit cadrele respective), și datele în sine (bytes).

1.2 Microcontrolerul ESP32

Microcontrolerele au revoluționat industria electronică prin accesibilitatea și versatilitatea lor. Arduino, o arhitectură *open-source* care utilizează procesoare Atmel, a atins o popularitate imensă în rândul pasionaților datorită costurilor reduse și instrumentelor de dezvoltare ușor de utilizat. Raspberry Pi, pe de altă parte, oferă capabi-

lități mai puternice cu procesoarele sale ARM și sistemul de operare bazat pe Linux. Cu toate acestea, o limitare a acestor microcontrolere este lipsa capabilităților de conectare cu alte dispozitive fără fir.

În timp ce Arduino și Raspberry Pi răspund diferitelor nevoi, integrarea rețelelor fără fir a devenit o cerință de bază pentru numeroase aplicații. ESP32 este un microcontroler care oferă conectivitate *wireless* împreună cu capabilități puternice de procesare, suport GPIO, memorie RAM și memorie flash încorporată. ESP32 reprezintă o alternativă rentabilă la alte microcontrolere.

Odată cu integrarea de către ESP32 a protocoalelor WiFi și TCP/IP (*Transmission Control Protocol/Internet Protocol*), devine posibilă crearea de aplicații ce depășesc funcționalitățile Arduino și Raspberry Pi. ESP32 deschide multe posibilități pentru proiecte IoT și dispozitive inteligente.

Compania Espressif Systems au lansat mai multe modele de microcontrolere ESP32 și plăci de dezvoltare ce încorporează aceste microcontrolere, și din care vom discuta în particular mai departe.

1.2.1 Arhitectura ESP32-S

Un model important în acest proiect este microcontrolerul ESP32-S; la baza acestuia se află ESP32, având un design scalabil și adaptiv. ESP32-S este format două microprocesoare de 32 de biți, ce pot fi controlate sau alimentate separat, cu o frecvență de ceas ajustabilă între 80-240 MHz. Cipul integrează diverse periferice, inclusiv senzori tactili, senzori Hall, amplificatoare, interfață card SD, Ethernet, converteri ADC/DAC, PWM, și intefețe de comunicare precum SPI, UART, I2S și I2C.

Memoria internă cuprinde:

- 448 KB de memorie ROM pentru *bootare* și alte funcții de bază.
- 520 KB de memorie SRAM pentru date și instrucțiuni.
- 4 MB de memorie *flash* pentru stocarea programelor.

ESP32-S suportă conectivitatea WiFi (standardul 802.11) și Bluetooth (BLE 4.2) și dispune de o antenă PCB (*Printed Circuit Board*), dar se poate conecta și o antenă externă de 2.4 GHz, dispunând de un conector u.FL și un *jumper* pentru a fi setat când se utilizează o antenă externă.

ESP32-S dispunde de 32 de pini GPIO (*General Purpose Input/Output*).

1.2.2 Placa de dezvoltare ESP32-CAM

Microcontrolerul ESP32-S poate fi dificil de folosit, motiv pentru care au fost introduse plăci de dezvoltare care sunt *"breadboard friendly"*. Aceste plăci de dezvoltare integrează modulul ESP32-S și oferă periferice suplimentare pentru a facilita implementarea ușoară în aplicațiile IoT. Această compatibilitate permite interconexiuni rapide între componentele electronice fără a fi nevoie de lipire.

Pe lângă modulul ESP32-S, plăcile de dezvoltare oferă o gamă de periferice integrate. Aceste periferice pot include butoane, LED-uri, conectori pentru senzori. În plus, plăcile de dezvoltare oferă o accesibilitate convenabilă a piniilor, regulatoare la bord pentru alimentarea cu energie, rezistențe încorporate pentru configurații pull-up/pull-down și alte componente ce simplifică procesul de dezvoltare.

Lansată în 2019, placa ESP32-CAM integrează microprocesorul ESP32-S dar și multe alte periferice precum: 4 MB de PSRAM externi, modulul de cameră OV2640, slot și modul pentru card SD, *flash* pentru cameră (GPIO4). Are integrat un LED programabil pentru depanare (GPIO33) și dispune de 10 pini GPIO expuși (restul de pini GPIO sunt folosiți intern pentru cameră și PSRAM). ESP32-CAM are 10 canale de pini (toți pinii GPIO) PWM pentru a transmite semnale PWM.

ESP32-S dispune de două interfețe UART (U0 și U2), dar doar pinii interfeței UART0 sunt expuși în totalitate (GPIO1 și GPIO3); și de doi converteri ADC pe 12 biți (ADC1 și ADC2), însă doar canalele pentru ADC2 sunt expuse, ESP32-S mai dispue și de o interfață SPI, ce are pinii expuși.

Toți pinii GPIO sunt conectați la senzori tactili înafară de cei ce sunt folosiți pentru comunicarea UART.

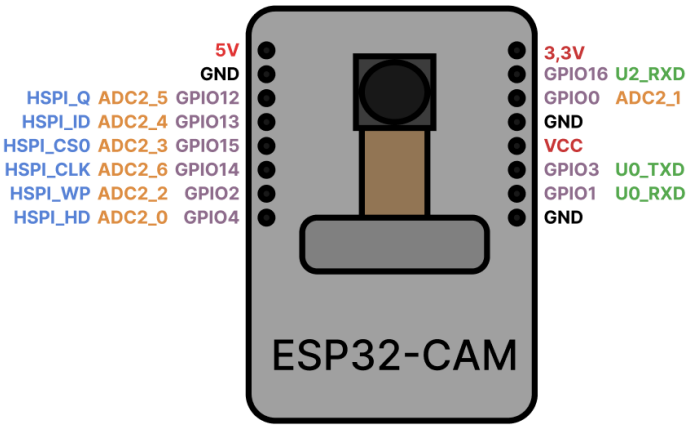

Figura 1.5: Pini ESP32-CAM

1.2.3 Modulul cameră OV2640

Modulul cameră integrat în ESP32-CAM este modelul OV2640. Este dotat cu o matrice de senzori de imagine cu o rezoluție de 1632x1232 (din care doar 1632x1220 sunt activi, restul sunt pentru calibrarea nivelului de negru și interpolare), filtrele de culori sunt aranjate în mod Bayer (BG/GR alternant), oferă rate variate de transfer de imagine ce pot prioritiza fie calitate sau eficiență. În plus, modulul este echipat cu mai multe funcții automate de control de imagine.

Transferul de imagine se face linie cu linie, fiecare linie trece pixel cu pixel printr-un amplificator analog, și după un ADC pe 10 biți. Datele de imagine trec la final printr-un DSP (*Digital Signal Processor*) unde se face interpolarea din date neprelucrate la RGB, plus alte funcții de control calitate; aceste date trec mai departe printr-un *formatter* de ieșire, un motor de compresie și la final ajung la un port digital video pe 10 biți (y9-0), aceste procese fiind controlate de un microcontroller de 8 biți. ESP32-CAM stabilește o conexiune de 8 biți în mod implicit (y9-2) cu portul digital video.

1.2.4 Modul card microSD

Modulul de card microSD utilizat de ESP32-CAM folosește o interfață de comunicare SPI pentru transfer de date, și își rezervă pini specifici pentru această comunicare (în cadrul utilizării bibliotecii *SD_MMC.h* dispusă de ESP32).

ESP32-S	Modul microSD
CLK	GPIO15
CMD	GPIO14
Data0	GPIO13
Data1	GPIO12
Data2	GPIO4
Data3	GPIO2

După cum se observă, acești pini sunt expuși, deci pentru o comunicare corespunzătoare trebuie lăsați liberi atunci când se utilizează modulul.

1.3 Standardul 802.11

Standardul IEEE 802.11 este standardul dominant pentru WLAN-uri (*Wireless Local Area Networks*). Acesta cuprinde stratul MAC (*Medium Access Control*) și stratul fizic

(PHY).

Standardul IEEE 802.11 cuprinde diverse tehnologii de nivel fizic, fiecare cu propriile metode de modulare. Acestea includ infraroșu, spectrul de răspândire cu secvență directă (DSSS), spectrul de răspândire cu salt de frecvență (FHSS) și multiplexarea prin divizare a frecvenței ortogonale (OFDM). Standardul a evoluat în timp și în prezent definește tehnologiile următoare:

Standard	Bandă	Transfer maxim
802.11	2.4 GHz	2 Mbps
802.11b	2.4 GHz	11 Mbps
802.11a	5 GHz	54 Mbps
802.11g	2.4 GHz	54 Mpbs
802.11n	2.4/5 GHz	600 Mbps
802.11ac	5 GHz	6.93 Gbps
802.11ax	2.4/5 GHz	14 Gbps

Arhitectura standardului este formată din mai multe componente de bază precum:

- Stație (STA): un dispozitiv echipat cu un MAC conform cu 802.11 și o interfață de nivel fizic pentru comunicații fără fir.
- Punct de Acces (AP): o entitate ce funcționează ca o stație și oferă acces la servicii distribuite la alte stații asociate.
- *Basic Service Set* (BSS): un grup constând dintr-un număr variabil de stații.
- *Basic Service Area* (BSA): zonă teoretică unde membrii dintr-un BSS comunică.

La baza comunicării fără fir în microprocesoarele ESP32 stă acest standard, mai exact, ESP32-S suportă standardele 802.11b/g/n, ce sunt standarde ce pot opera pe o bandă de 2.4 GHz (ESP32-S suportă doar banda de 2.4 GHz).

1.4 Protocolul TCP/IP

După ce microcontrolerul a fost deschis ca punct de acces sau s-a conectat însuși ca stație la o rețea, datele trebuie transmise cumva după un anumit standard.

Aici vine protocolul TCP/IP (*Transmission Control Protocol/Internet Protocol*) ce este utilizat de microprocesoarele ESP32 pentru transmiterea fiabilă a datelor prin rețea. TCP definește cum se pot crea canale de comunicații în rețea și cum trebuie spart mesajul în pachete și reconstruit, iar IP definește ruta pachetelor și unde trebuie să se ducă pentru a ajunge la destinație folosind adrese IP.

TCP/IP folosește modelul de server/client și se consideră ca fiind un model *stateless*, însemnând că fiecare cerere este individuală față de alta.

Datele, înainte de a fi trimise fizic, se encapsulează trecând prin cinci straturi:

- *Application*: datele mesajului din aplicație este encapsulat în protocoale precum: *Hypertext Transfer Protocol* (HTTP), *File Transfer Protocol* (FTP), *Simple Mail Transfer Protocol* (SMTP).
- *Transport*: se adaugă un antet de transport mesajului. La TCP, acesta poate include port-uri la sursă și destinație, număr de ordine, și alte informații de control. La UDP, antetul este mai simplu având la bază port-urile la sursă și destinație.
- *Network*: mesajul este encapsulat într-un pachet IP, adaugând adresele IP al sursei și destinației, și alte informații precum opțiuni, versiune, TTL (*Time to Live*), etc.
- *Data Link*: se adaugă adresele MAC la destinație și sursă, plus informații de detectare de erori, astfel completând cadrul de date.
- *Physical*: se ia cadrul de informații și se convertește într-un curent de biți pentru a fi transmiși fizic (ex. prin cabluri sau semnale *wireless*).

Diferența dintre TCP și UDP la nivelul de transport este că la TCP se stabilește o conexiune sigură între indivizi, la TCP când se trimite un pachet, se așteaptă un răspuns de confirmare, altfel se trimite dinnou, garantând trimiterea pachetului dar fiind mai lent. La UDP, nu este garantat că pachetul ajunge, nu se supraîncarcă rețeaua, deci e mai rapid.

TCP multiplexează o sursă IP, suportând până la 65536 de porturi. O aplicație ascultă la un anumit port pentru cereri. Porturile 0-1023 sunt atribuite protocoalelor uzuale precum: HTTP (80), FTP (21), SMTP (25).

1.5 Protocolul HTTP

La acest proiect, se va folosi protocolul HTTP pentru comunicarea server-client, deci este important de prezentat câteva aspecte generale și reguli pentru a fi folosit în mod corespunzător.

HTTP este un protocol asimetric pe bază de cereri și răspunsuri între clienți și servere. Clientul trimite o cerere către server și primește în schimb un răspuns. Este un protocol fără stare, deci fiecare cerere este independentă.

În cadrul unui *browser*, un URL (*Uniform Resource Locator*) este folosit pentru a identifica o resursă web, are forma "protocol://hostname:port/path" cu următoarele componente:

- *protocol*: protocolul de aplicație folosit (de exemplu: HTTP, FTP).
- *hostname*: numele DNS (*Domain Name System*) sau adresa IP.
- *port*: numarul port TCP unde ascultă serverul pentru cereri.
- *path*: numele resursei sau calea.

Un browser ar traduce un URL într-un mesaj de cerere de tip GET la adresa respectivă. Dacă nu se menționează portul, atunci acesta îl folosește pe cel implicit (80), chiar dacă protocolul HTTP are atribuit portul 80 în mod implicit, se pot implementa servere HTTP ce pot asculta la alte porturi atribuite de admini.

Un mesaj HTTP constă dintr-un antet și conținutul propriu-zis (ce poate fi opțional), cele două fiind separate printr-o linie goală.

În formatul mesajului de cerere, în antet, prima linia reprezintă linia de cerere, ce conține metoda de cerere, URL și versiunea HTTP (separate prin spații). Metodele de cerere pot fi de exemplu: GET, POST. Următoarele linii din antet sunt anteturi de cerere (sau *Request Headers*) ce au forma de perechi *name:value* și pot specifica informații adiționale precum tipul de date cerut sau trimis.

În formatul mesajului de răspuns, prima linie din antet reprezintă linia de status, pentru a indica clientului dacă cererea a fost validă; linia de status este formată din versiunea HTTP, codul unic de status (cod universal format din 3 cifre pentru a indica un status specific, de exemplu: 200, 404), și un cuvânt motiv asociat codului de status (de exemplu: 200 OK, 404 Not Found); restul de linii ale antetului reprezintă anteturi de răspuns (sau *Response Headers*).

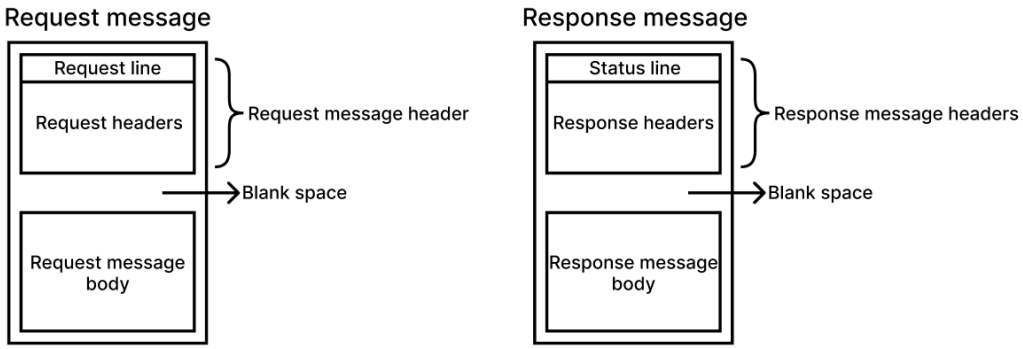

Figura 1.6: Mesaje HTTP

Protocolul HTTP definește o multitudine de metode de cerere, printre care:

- GET: pentru a extrage informații de la server.
- POST: pentru a posta date la server.
- PUT: pentru a-i spune serverului să stocheze datele.
- DELETE: pentru a-i spune serverului să șteargă datele.
- OPTIONS: pentru a primi o listă de metode de cerere suportate de server.

1.6 Motoare

Motoarele electrice folosesc energie electrică pentru a atinge o mișcare de rotație. Există două principii ce stau la baza funcționării lor: forța Lorentz și atracția magnetică.

Când trece un flux de curent printr-un fir, se generează un câmp magnetic circular în jurul firului, cu direcția depinzând de sensul de flux al electronilor.

Dacă se așează un fir într-un câmp magnetic static și se va permite trecerea unui flux de curent, o forță va acționa asupra firului numită forța Lorentz.

Această forță poate fi exploatată să genereze mișcări rotative folosind un fir aranjat într-o buclă și un câmp magnetic static. O dată ce se permite trecerea unui flux de curent prin fir, acesta se va roti până ajunge la planul ce coincide cu câmpul magnetic, după care se oprește. Dacă se oprește curentul înainte de a ajunge la un stop, lăsând mișcarea să treacă de acel punct, după inversând fluxul de curent, va cauza acel fir să facă o altă rotație mai departe; repetând această procedură, se poate menține o rotire continuă.

Utilizând atracția magnetică, se poate folosi o bobină; o bobină generează un câmp electromagnetic cu o polaritate ce depinde de sensul fluxului de curent. Folosind un magnet permanent și o bobină (sau mai multe) se pot atinge mișcări de rotație folosind același principu ca mai sus.

Motoarele electrice sunt formate din două părți principale: un *rotor* și un *stator*. Depinzând de tipul de motor, rotorul poate fi un set de bobine (sau bucle) și statorul un magnet permanent; sau rotorul poate fi un magnet permanent, și statorul format din perechi de bobine.

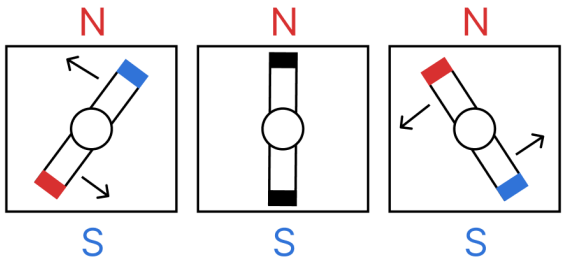

Figura 1.7: Rotire pe baza atracției magnetice

1.6.1 Motoare DC

Motoarele DC (*Direct Current*) funcționează pe baza unui voltaj non-alternant pentru a atinge mișcările de rotire. Acestea au doi pini pentru a controla viteza și direcția motorului.

Există în prezent două tipuri de motoare DC: cu sau fără perii.

La motoarele cu perii, un stator generează un câmp magnetic constant, iar rotorul constă dintr-o serie de bucle sau bobine; pentru a se inversa fluxul de curent se folosește un comutator (cu contacte separate atașate la capetele bobinelor), iar pentru a transmite curentul se folosesc perii ce ating contactele, după o rotație de 180°periile ajung în decalajele comutatorului, unde rotorul se va mișca din inerție după care se ating următoarele contacte ale comutatorului, creând și menținând astfel mișcarea de rotație.

Un dezavantaj al motoarelor cu perii îl prezintă uzarea periilor ce poate cauza un contact inadecvat. Motoarele fără perii în comparație folosește ca rotor un magnet permanent și ca stator perechi de bobine ce sunt excitate alternat, neavând perii acest tip de motoare au o durată de viață ridicată dar sunt mai greu de implementat, deci mai scumpe.

1.6.2 Modul driver

Din moment ce motoarele DC trag mult curent (sute de mA) și nu folosesc întodeauna același nivel de voltaj ca și microcontrollerul, nu putem să le conectăm direct.

Pentru a controla motoarele, se pot folosi module *driver*, ce implementază în general una sau mai multe punți H. Circuitul angajat într-o singură punte H implică utilizarea a doi tranzistori PNP la comutatorii superiori și doi tranzistori NPN la cei inferiori. Acest circuit este capabil să controleze direcția motorului prin activarea a unui comutator superior și prin activarea a comutatorului inferior opus creând astfel un flux de curent ce trece prin motor.

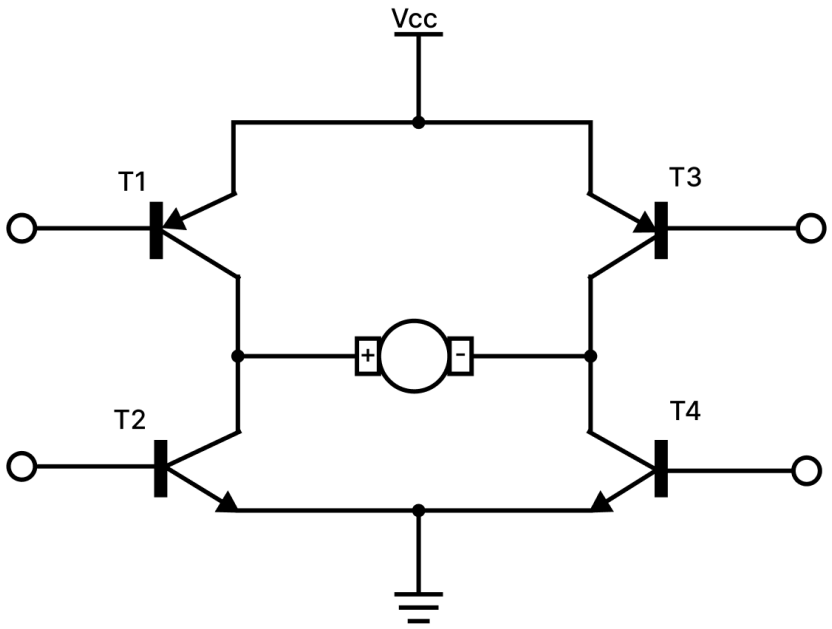

Figura 1.8: Circuit punte H simplificat

Puntea H permite controlarea motorului folosind două terminale de intrare, A și B, fiecare controlând o parte din punte. Atunci când se trimite un semnal HIGH terminalului de intrare A, tranzistorul superior T1 este activat și tranzistorul inferior T2 este dezactivat, și invers pentru un semnal LOW. Deci, pentru a atinge un circuit închis, se va trimite semnalele HIGH-LOW sau LOW-HIGH terminalelor de intrare A, și respectiv B. Pentru controlare de viteză se pot transmite și semnale PWM.

1.7 Alimentare

Este important de cunoscut tipurile diferite de baterii și cum trebuie controlat curentul pentru a alimenta microcontrollerele corespunzător.

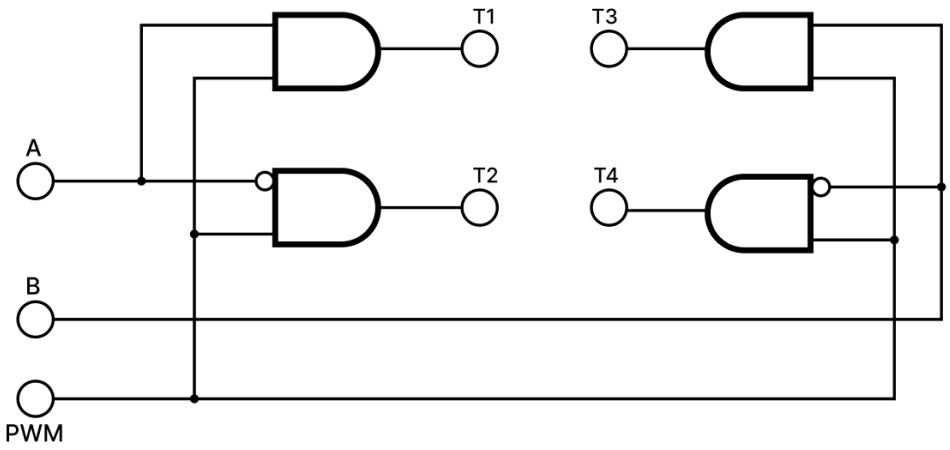

Figura 1.9: Controlare punte H

Există două clasificări de baterii: primare și secundare.

Bateriile primare sunt de unică folosință, ușor de folosit, fiind o alegere convenabilă, acestea sunt compuse din celule electrochimice în care reacțiile electrochimice devin ireversibile după utilizare.

Bateriile secundare folosesc reacții ce pot fi reversabile prin aplicarea unui anumit voltaj bateriei, fiind astfel reîncărcabile.

Microprocesorul ESP32 poate funcționa alimentat cu un curent de 3,3V sau 5V. În cazul ESP32-CAM, există multe recomandări de alimentare exclusiv la 5V.

ESP32 poate trage mult curent atunci când transmite date prin *wireless*, sau când se folosesc alte periferice precum camera în cadrul plăcii ESP32-CAM; asta poate duce la fluctuații de voltaj, și este recomandat (din mai multe surse) conectarea unui condensator între 10-100 μF cât mai aproape de modul.

Alte recomandări din fișe tehnice este ca curentul furnizat să rămână constant între 1-2A. ESP32-CAM poate consuma de la 6mA (în modul *deep sleep*) până la 310mA (cu *flash-ul* pornit).

În cazul folosirii bateriilor reîncărcabile, este recomandat de folosit module de reîncărcare conforme tipului lor, pentru a evita scenarii de supraîncărcare.

1.8 Biblioteci ESP32

În partea programării plăcii ESP32 stau la bază bibliotecile ESP32 puse la dispoziție de Esspresif Systems, ce fac programarea și comunicarea cu perifericele microcontrollerului mai ușoară și sigură.

În acest proiect, s-au utilizat trei astfel de biblioteci ce vor fi explorate și explicate în subcapitolele ce urmează.

1.8.1 WiFi

Biblioteca ce face posibilă utilizarea funcționalităților Wi-Fi peste TCP/IP este 'WiFi.h'. ESP32 poate acționa ca o stație Wi-Fi, punct de acces sau amândouă. Pentru a seta modul cum funcționează microcontrollerul, se va folosi funcția 'WiFi.mode()' cu unul din următoarii parametrii:

- WIFI_OFF: pentru a dezactiva funcționalitățile Wi-Fi.
- WIFI_STA: pentru a configura ESP32 ca stație.
- WIFI_AP: pentru a configura ESP32 ca punct de access.
- WIFI_AP_STA: pentru a configura ESP32 atât ca stație cât și punct de access.

Atunci când placa este configurată ca o stație Wi-Fi, aceasta se poate conecta la alte rețele, unde îi va fi atribuit o adresă IP unică. După care se pot folosi alte dizpozitive conectate în aceași rețea pentru a comunica cu placa folosind adresa IP asociată.

Atunci când placa este configurată ca un punct de acces, își crează propria rețea Wi-Fi unde se poate conecta direct orice dizpozitiv din apropiere cu capabilități Wi-Fi. Din cauză că ESP32 nu are acces la internet mai departe, această metodă se numește *soft AP (soft Access Point)*.

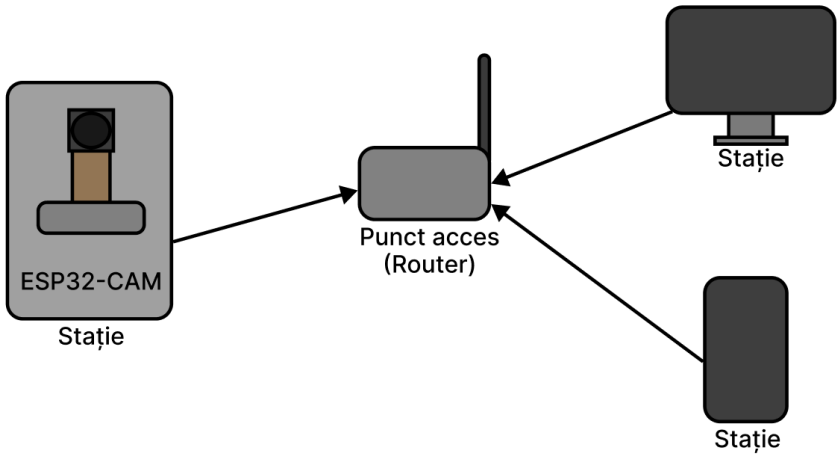

Figura 1.10: Stație Wi-Fi

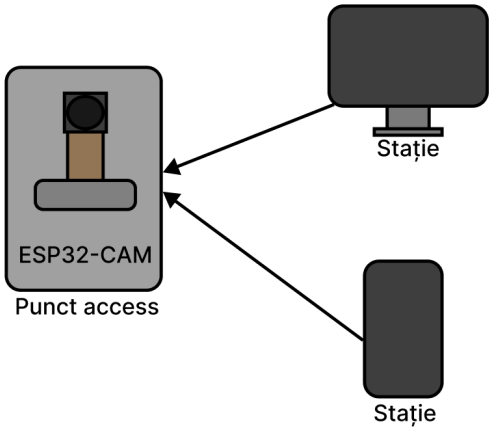

Figura 1.11: Punct de acces

Pentru a configura ESP32 ca stație se folosește funcția 'WiFi.begin' cu următorii parametrii:

- ssid: Numele rețelei Wi-Fi în care să se conecteze (maximum 63 de caractere).
- password (opțional): Parola pentru conectare, cu caractere ASCII între 32-126 (implicit NULL).
- bssid (opțional): un vector de 6 bytes, reprezentând adresa MAC al rețelei, în caz că sunt rețele cu același ssid.
- channel (opțional): Numărul canalului, pentru a identifica rețelele cu același ssid, între 1 și 13 (implicit 1).
- connect (opțional): valoare booleană pentru a indica dacă funcția se va conecta direct după inițializare, altfel se va conecta utilizând 'WiFi.connect()' (implicit true).

Pentru a configura ESP32 ca punct de acces se utilizează funcția 'WiFi.softAP()'. Această funcție primește următorii parametri:

- ssid: Numele punctului de acces (maximum 63 de caractere).
- password (opțional): Parola pentru punctul de acces, cu minimum 8 caractere (implicit NULL).
- channel (opțional): Numărul canalului Wi-Fi, între 1 și 13 (implicit 1).

- ssid_hidden (opțional): Valoare între 0 și 1, unde 1 indică faptul că SSID-ul este ascuns, iar 0 indică faptul că este public (implicit 0).
- max_connection (opțional): Numărul maxim de clienți conectați, cu valori între 1 și 4 (implicit 4).
- ftm_responder (opțional): O valoare booleană care indică dacă să se activeze funcționalitatea FTM (Fine Timing Measurement). FTM permite calcularea precisă a distanței utilizând măsurători de tip *time-of-flight* (implicit fals).

Este important de verificat statusul de conectare după ce au fost apelate una din funcțiile de mai sus utilizând funcția 'WiFi.status()', aceasta întoarce una din următoarele valori:

- 0 sau WL_IDLE_STATUS: indică faptul că modulul Wi-Fi se află într-o stare inactivă.
- 1 sau WL_NO_SSID_AVAIL: când nu sunt detectate rețele cu ssid-ul specificat
- 2 sau WL_SCAN_COMPLETED: scanarea de rețele s-a terminat.
- 3 sau WL_CONNECTED: s-a conectat la o rețea cu succes.
- 4 sau WL_CONNECT_FAILED: conexiune eșuată.
- 5 sau WL_CONNECTION_LOST: s-a pierdut conexiunea.
- 6 sau WL_DISCONNECTED: s-a deconectat de la o rețea.

Pentru a scana rețelele din apropiere, se poate utiliza funcția 'WiFi.scanNetworks()', aceasta întoarce numărul de rețele găsite.

După scanare, se pot accesa parametrii fiecărei rețele, folosind funcția 'WiFi.SSID(i)' se poate obține ssid-ul rețelei i din lista celor scanate, se poate folosi și funcția 'WiFi.RSSI(i)' pentru a obține un estimat al puterii semnalului acelei rețele.

Pentru a obține mai multe informații se poate folosi funcția 'WiFi.getNetworkInfo()' ce primește ca prim parametru numărul i al rețelei din listă, iar restul de parametrii vor fi adresele unor variabile pentru a încărca informațiile de rețea în acestea; în ordinea specificată se pot obține următoarele informații: ssid, encryptionType, RSSI, BSSID, channel. Această funcție va întoarce 'true' dacă a rulat cu succes.

Pentru a șterge lista scanată din RAM, se folosește funcția 'WiFi.scanDelete()'.

Pentru a deconecta ESP32 de la o rețea sau reconecta la ultima rețea, se pot folosi funcțiile 'WiFi.disconnect()' și 'WiFi.reconnect()'.

1.8.2 WebServer

Pentru a seta un server web, se va utiliza biblioteca 'WebServer.h'. Pentru a crea un server web, trebuie instanțiat un obiect WebServer. Constructorul clasei WebServer acceptă ca parametrii fie o adresă IP și port sau doar un port pe care va asculta cererile clientului:

```
WebServer server(80);
```

Funcția principală pentru a defini un *handler* de cereri este 'on()', ce poate primii următorii parametrii:

- URI (*Uniform Resource Identifier*): calea specifică unde este asociat acel *handler*.
- metoda HTTP (opțional): poate fi HTTP_GET sau HTTP_POST (implicit HTTP_ANY).
- funcția *handler*: funcție definită de utilizator pentru a procesa o cerere.

```
server.on("/test", HTTP_GET, handler_function);
```

După ce a fost configurat un server web, acesta poate fi pornit cu funcția 'begin()'. Pentru a opri un server complet se utilizează funcția 'stop()'. Mai există și funcția 'close()' ce închide conexiunea cu clientul, dar nu oprește serverul propriu-zis.

În protocolul HTTP, pentru a completa o cerere a unui client, serverul trebuie să trimită un mesaj de răspuns. O funcție ce poate crea și trimie mesaje de răspuns ușor de utilizat este 'send()' ce primește ca parametrii: codul de status (200/404/etc.), tipul de conținut (text/plain, image/jpeg, etc.) și conținutul propriu zis.

Mesajul de cerere poate fi trimis și manual, adică construit de user, într-un String, utilizând funcția 'sendContent()', de exemplu:

```
String response =
    "HTTP/1.1 200 OK\r\n"
    "Content-Type: text/plain\r\n\r\n"
    "Request has been handled.";
server.sendContent(response);
```

1.8.3 esp_camera

ESP32-CAM dispune de modulul OV2640, pentru a configura convenabil acest modul, vom utiliza bilbioteca 'esp_camera.h'. Microcontrolerul ESP32-S în cadrul plăcii ESP32-CAM își rezervă pini exclusivi pentru funcții legate de cameră. Prin urmare, este esențial să fie configurate corect pentru funcționarea corespunzătoare a camerei.

ESP32-S	OV2640
GPIO0	XCLK
GPIO22	PCLK
GPIO25	VSYNC
GPIO23	HREF
GPIO26	SDA
GPIO27	SCL
GPIO32	POWER
GPIO5	D0
GPIO18	D1
GPIO19	D2
GPIO21	D3
GPIO36	D4
GPIO39	D5
GPIO34	D6
GPIO35	D7

Biblioteca dispune de o structură de configurare (camera_config_t) pentru inițializarea camereri în care se pot seta pinii și alte setări de imagine:

- 'pixel_format' - pentru a specifica formatul pixel al camerei, PIXFORMAT_JPEG pentru o compresie JPEG.
- 'frame_size' - pentru a specifica rezoluția camerei, există multe opțiuni de ales printre care: FRAMESIZE_UXGA (1600x1200), FRAMESIZE_SVGA (800x600), FRAMESIZE_VGA (640x480), FRAMESIZE_CIF (352x288).
- 'jpeg_quality' - calitate jpeg, poate fi număr între 0 și 63, 0 reprezentând cea mai mică calitate dar compresie maximă, iar 63 reprezentând calitate maximă dar compresie minimă.

- 'fb_count' - numărul de *frame buffers* alocate.

Cu structura de configurare creată, se poate inițializa camera utilizând funcția 'esp_camera_init()', aceasta pregătește și configurează camera și va întoarce ESP_OK dacă inițializarea a avut succes.

Funcția de captură de imagine este 'esp_camera_fb_get()' ce intoarce un *pointer* către un *frame buffer*, aceasta pregătește un *frame buffer* întors de funcția 'cam_take()', unde se folosește 'xQueueReceive()' pentru a extrage un *frame buffer* dintr-o coadă de capturi de imagine, 'cam_take()' verifică dacă acel *buffer* este valid, și își alocă memorie depinzând de modul de compresie/imagine ales.

Este esențial de a elibera *frame buffer-ul* după ce a fost utilizat pentru a fi refolosit utilizând funcția 'esp_camera_fb_return(camera_fb_t *fb)', ce primește ca argument *pointer* către acel *buffer*.

Se pot efectua și setări de imagine precum luminozitate, constrast, saturație folosind funcția 'esp_camera_sensor_get()' ce întoarce un *pointer* către structura de control al senzorului.

```
sensor_t * s = esp_camera_sensor_get();
```

Se pot folosi următoarele funcții pentru aplicare de setări la nivel de imagine:

- s->set_brightness(s, ?): setare de luminozitate (valori între -2 și 2).
- s->set_contrast(s, ?): setare contrast (valori între -2 și 2).
- s->set_saturation(s, ?): setare saturație (valori între -2 și 2).
- s->set_special_effect(s, ?): pentru efecte speciale (0 - fara efect, 1 - negativ, 2 - nuanță gri, 3 - nuanță roșie, 4 - nuanță verde, 5 - nuanță albastră, 6 - sepia).

Capitolul 2

Construirea sistemului integrat

2.1 Pregătire componente

Așa cum a fost discutat în primul capitol, modulul de dezovoltare ales este ESP32-CAM din motivul integrării modulului camerei și a funcționalităților Wi-Fi într-o singură placă de dezvoltare.

Placa de dezvoltare ESP32-CAM, ca și multe alte plăci ESP32, nu vine cu programator integrat, microcontrolerul utilizează interfața de comunicare UART pentru programare și depanare. Deci o interfață externă de convertire din USB în UART face posibilă comunicarea prin stabiliare unei conexiuni seriale între cele două. Microcontrollerul folosește un *baud rate* implicit de 115200 bps. Orice punte *USB to UART* ce suportă acest *baud rate* este suficientă. Pentru acest proiect, am ales modulul FTDI232.

Motoarele utilizate în proiect sunt de tip DC cu perii, cu o raportare a angrenajului de 90:1, model JGB37-520, ce funcționează la o tensiune de până la 12V, având o viteză de rotație de 107 RPM (*Revolutions per Minute*). Aceste motoare reprezintă o alegere bună din punct de vedere al bugetului, oferind o combinație între putere și calitate pentru dronă. De asemenea, ele furnizează un cuplu mare, având un cuplu maxim de blocare de 3kg.cm, ceea ce este mai mult decât suficient având în vedere greutatea aproximativă a dronei de 1-2kg.

Modulul *driver* utilizat pentru controlarea motoarelor este puntea H dublă L298N, deseori folosită pentru motoare DC de până la 35V.

Ca sursă de alimentare, am folosit trei baterii cu litiu ionizat (Li-ion) de 3,7V conectate în serie. Recomand baterii de litiu (Li-Ion sau Li-Po) din motivul tensiunii lor stabile, ratele mari de descărcare și posibilitatea lor de re-încărcare. Am efectuat teste

cu diferite baterii alcaline, cum ar fi cele de 9V și 6V (patru baterii de 1,5V conectate în serie), cu și fără un regulator de tensiune, și am constatat că camera prezintă mici fluctuații sau linii distorsionate în imagine și nu răspunde la fel de bine. Aceste baterii vor alimenta puntea de motoare cu aproximativ 12V și folosind regulatorul intern de tensiune de 5V al modulului se va alimenta placa ESP32.

2.2 Programare ESP32

Înainte de a programa placa ESP32, se va seta puntea de comunicare USB/UART corespunzător. Modulul FTDI232 dispune de un *jumper* ce poate fi setat să furnizeze 5V sau 3,3V prin pinul VCC, o recomandare este ca puntea să alimenteze 5V în pinul de alimentare de 5V al plăcii; alimentarea plăcii la pinul de 3,3V cu 3,3V poate produce erori atunci când se încarcă cod. Pentru a încărca cod, înainte de a conecta placa ESP32 la punte, este esențial ca placa să fie setată în 'mod de programare', acest lucru se face prin conectarea pinilor GND și GPIO0 folosind un *jumper*. FTDI-ul va comunica cu microcontrolerul ESP32-S folosind interfața UART0 al acestuia (ce este singura intefață UART expusă din microcontroler la placă).

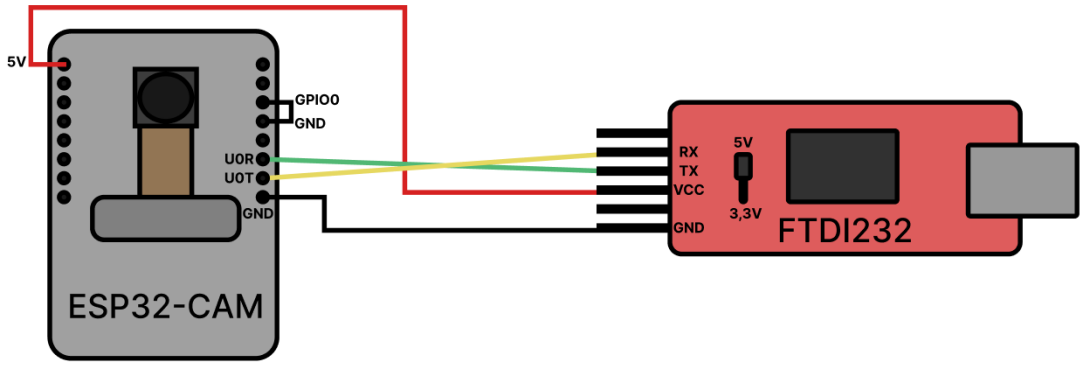

Figura 2.1: Conectare FTDI232 la ESP32-CAM

2.2.1 Mediu de dezvoltare

Programarea microcontrolerului ESP32 implică compilare de cod ESP32 și construirea aplicației pentru încărcare.

Un mediu de dezvoltare foarte utilizat în programarea microcontrolerelor este **Arduino IDE**, care suportă și programarea microcontrolerelor ESP32. Cu toate acestea,

pot exista probleme de compatibilitate cu versiunile mai noi ale IDE-ului. Versiunea recomandată pentru acest proiect este 1.8 (versiunea Legacy).

Înainte de programare, este esențial să se facă o configurare, mai exact, să se instaleze dependențele ESP32, folosind link-ul de mai jos adăugat la **Preferences**, sub **Additional Boards Manager URLs**:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-
pages/package_esp32_index.json
```

După de instalat *framework-ul* pentru plăcile ESP32 din *tab-ul* **Boards Manager**. *Board-ul* specific plăcii ESP32-CAM este 'AI Thinker ESP32-CAM'.

2.2.2 Configurare module

În această implementare, vom configura ESP32-CAM ca punct de acces unde se pot conecta alte dispozitive, cum ar fi telefonul mobil.

```
WiFi.mode(WIFI_AP);
WiFi.softAP("MyESP32AP", "password");
```

Configurarea modulului cameră este extrem de importantă, cum a fost discutat și despre biblioteca camerei, ESP32-S își rezervă unii pini pentru comunicarea modulului; un exemplu de configurare se află la anexa 1.

2.2.3 Controlare motoare

Modulul *driver* pentru motoare L298N utilizat în acest proiect primește trei semnale de intrare pentru fiecare punte, constând din două semnale digitale pentru controlarea direcției motorului și un semnal PWM pentru controlarea vitezei, iar la ieșire se află două terminale pentru a fi conectate la motor. Modulul dispune și de *jumpere* pentru a seta EnA și EnB la o viteză constantă (5V) dacă este cazul. În proiect s-au utilizat însă viteze variate.

EnA	semnal PWM Motor A
In1	semnal Motor A
In2	semnal Motor A
In3	semnal Motor B
In4	semnal Motor B
EnB	semnal PWM Motor B
Out1	terminal ieșire Motor A
Out2	terminal ieșire Motor A
Out3	terminal ieșire Motor B
Out4	terminal ieșire Motor B

Modulul oferă și alte funcționalități precum protecții termice folosind un radiator, protecții asupra circuitelor folosind diode în sensul opus tranzistorilor unei punți pentru a elimina vârfuri de tensiune după dezactivarea motoarelor, regulatori de tensiune și alte protecții de supracurent folosind rezistori.

Înainte de a discuta implementarea codului de controlare motoare, am pregătit o mapă ce va ajuta cu vizualizarea ideii generale:

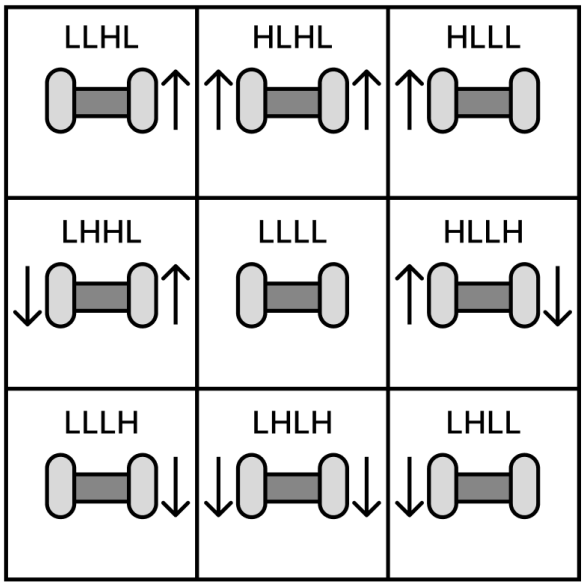

Figura 2.2: Mapă direcții

În mapă sunt prezentate 9 stări diferite de rotație a roților dronei, ce ating toate direcțiile posibile cu două roți motor. O stare este dată de 4 valori, ce pot fi H (High) sau L (Low), reprezentând semnalele ce vor fi trimise terminalelor de intrare In1, In2, In3 și In4. Reamintesc că un motor este pus în funcțiune dacă primește semnale diferite

(HL sau LH).

Pentru a atinge toate cele 9 stări, este suficient de lucrat cu 2 parametrii, x și y, unde x poate fi o valoare reprezentând *stânga* sau *dreapta*, iar y *înainte* sau *înapoi*; fie valori absolute pentru viteză maximă, fie valori parțiale pentru viteze variate.

În proiectul de față, valorile negative pentru x, respectiv y, reprezintă *stânga* și *înainte*; iar cele pozitive *dreapta*, respectiv *înapoi*; valoarea 0 reprezentând stare de repaus.

2.2.4 Server web

Aplicația de mobil va comunica cu placa prin cereri HTTP (*Hypertext Transfer Protocol*) peste protocolul TCP/IP.

Odată ce placa este setată ca punct de acces și o stație s-a conectat, putem trimite cereri către adresa IP implicită de *router* (192.168.4.1) peste diferite porturi.

Obiectivul este să se realizeze două funcționalități principale cu drona: transmiterea în flux a imaginilor de la cameră către client și controlul mișcării. Pentru transmiterea în flux, atunci când este primită o cerere de la client, serverul va trimite continuu mesaje de răspuns cu capturi de imagine pe parcusul cererii. Din cauza limitărilor serverului web care, din documentație, suportă doar un singur client simultan, nu se va putea trimite alte cereri aceluiași server pentru a controla drona. Pentru a rezolva problema, se va configura două servere separate pe porturi diferite (80 și 81 de exemplu), dedicate în mod specific *streaming-ului* camerei și controlului dronei.

Serverul ce ascultă portul 80 va fi pentru *streaming*, va avea un singur *handler* pentru a începe fluxul ce va răspunde la cereri de tip GET către *root*. O dată ce se trimite o cerere de tip GET către root. Serverul va trimite ca răspuns o linie de status de confirmare și antetul *'Content-Type: multipart/x-mixed-replace; boundary=frame'*, ce indică faptul că serverul va trimite date în mod continuu către client în bucăți delimitate de șirul dat de parametrul 'boundary'. Astfel am creat un *stream* pentru cameră. A se vedea anexa 2 pentru un exemplu de *handler*.

Serverul ce ascultă portul 81 va fi pentru alte butoane și funcționalități, acesta va fi pus într-un *thread* separat folosind 'xTaskCreatePinnedToCore()'. Serverul va avea un *handler* pentru a controla motoarele ce răspunde la cereri de tip POST către calea '/set', un *handler* pentru a opri fluxul camerei curente ce răspunde la cereri de tip GET la calea '/exit', și un *handler* pentru a porni/opri lumina *flash* al modulului ce răspunde

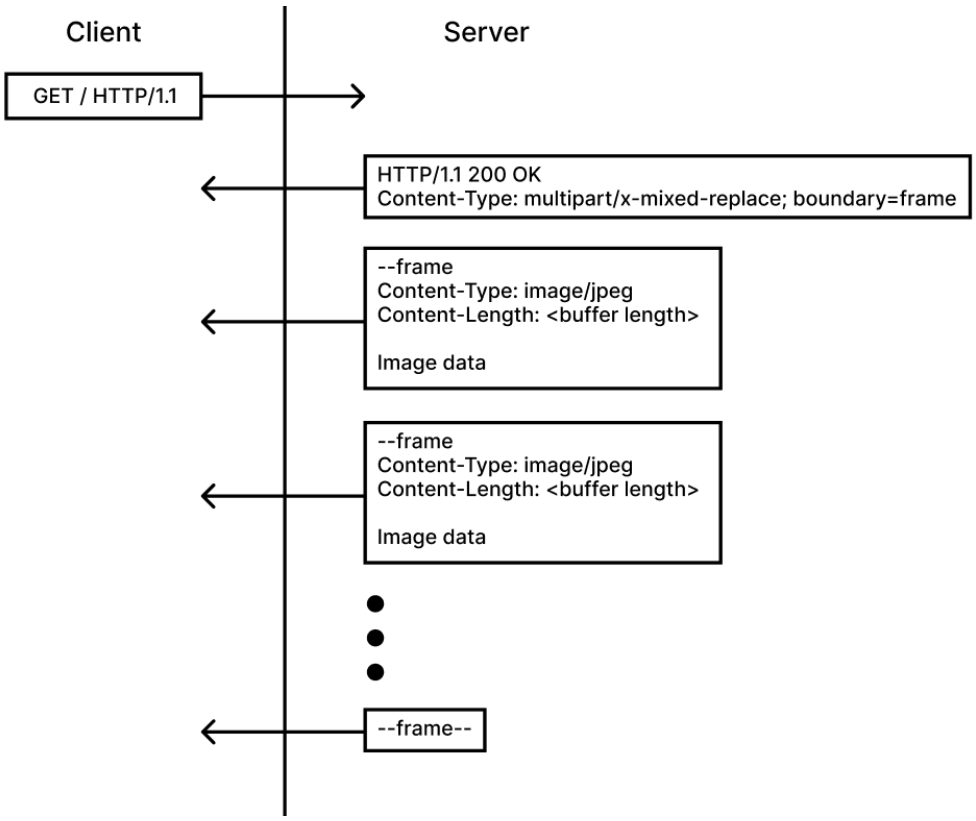

Figura 2.3: Camera stream

la cereri de tip GET către calea '/flash'. În cazul fiecărei cereri, serverul va trimite un mesaj de răspuns de cod 200 dacă cererea a avut succes.

În cadrul controlului motoarelor, serverul va trebuie să primească cele două valori, x și y, pentru a mapa direcția de control.

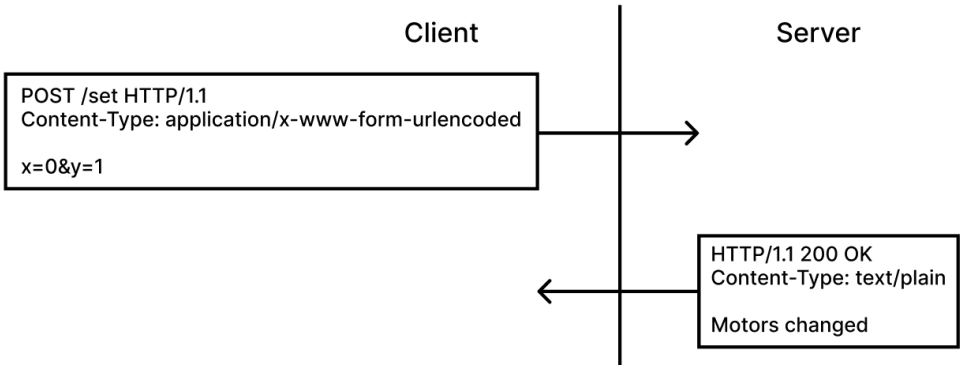

Figura 2.4: Control motoare

În cazul primirii cererei de aprindere/stingere a luminii *flash* al modulului, se poate folosi pinul GPIO4 ce este implicit conectat la acest modul; se pot trimite semnale PWM pentru a controla intensitatea luminii.

2.3 Conectare periferice

Placa ESP32-CAM dispune de 10 pini GPIO expuși, din care GPIO0 trebuie lăsat liber atunci când se folosește camera, iar pinul GPIO4 controlează modului *flash* (însă se poate folosi pentru a conecta periferice externe în același timp).

În cadrul acestui proiect, s-au folosit 6 pini GPIO pentru comunicarea cu modulul *driver*.

Cât mai aproape de placa ESP32-CAM, s-a conectat un condensator de 100 μF pentru a evita fluctuațiile de voltaj.

Modulul L298N folosește 5V pentru ași alimenta circuitele interne (chiar și când e conectat la o sursă de 12V) folosind un regulator intern; acesta dispune de două terminale de alimentare, cel de 12V, unde vom conecta sursa de alimentare, și cel de 5V, unde îl putem folosi pentru a conduce curent regulat de 5V la placa ESP32.

Sursa de alimentare reprezintă trei baterii de litiu ionizat de 3,7V conectate în serie, ce va rezulta un voltaj de 11.1V, la acestea s-a conectat un întrerupător și un conector de curent continuu (prin care va fi alimentată).

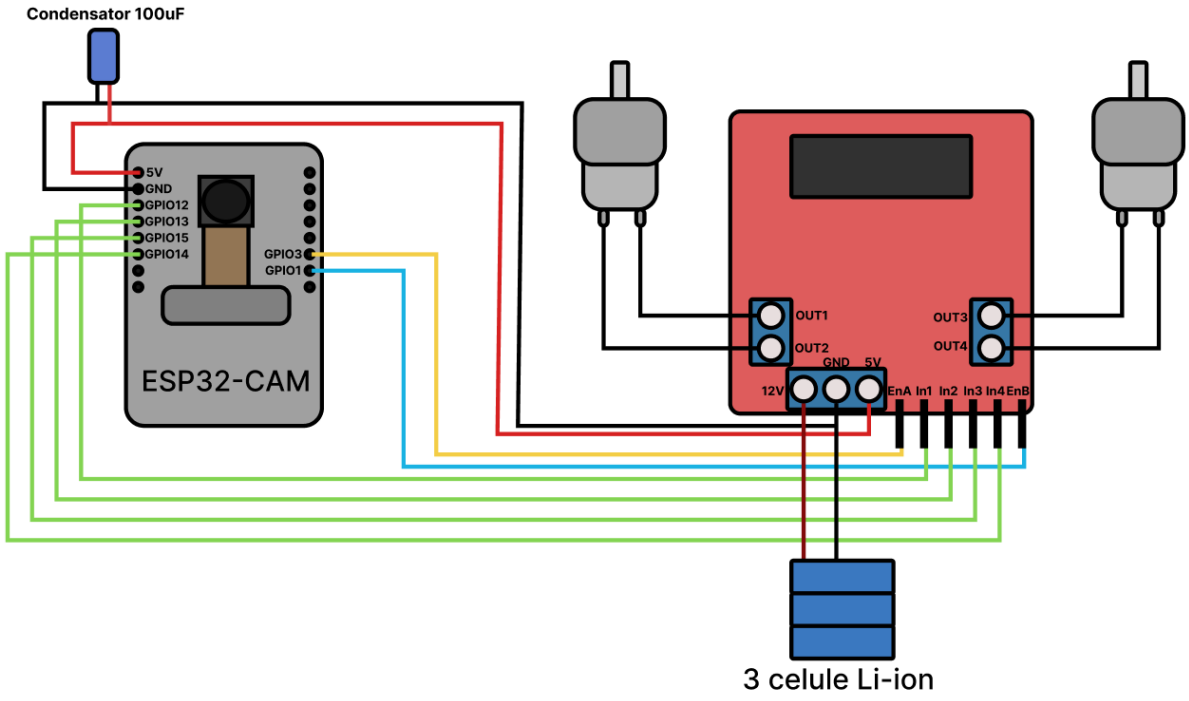

Figura 2.5: Circuit ESP32 și driver

2.4 Implementare aplicație

Pentru implementarea aplicației mobile, s-a folosit platforma **Kodular**, care dispune de numeroase module ușor de utilizat. Aceasta oferă un mediu de dezvoltare vizual, ce permite crearea rapidă și ușoară a interfețelor grafice și integrarea de funcționalități în aplicație. Platforma pune la dispoziție o varietate de componente și blocuri de construcție, ceea ce face adăugarea de funcționalități complexe în aplicație mai ușoară, fără prea multă configurare.

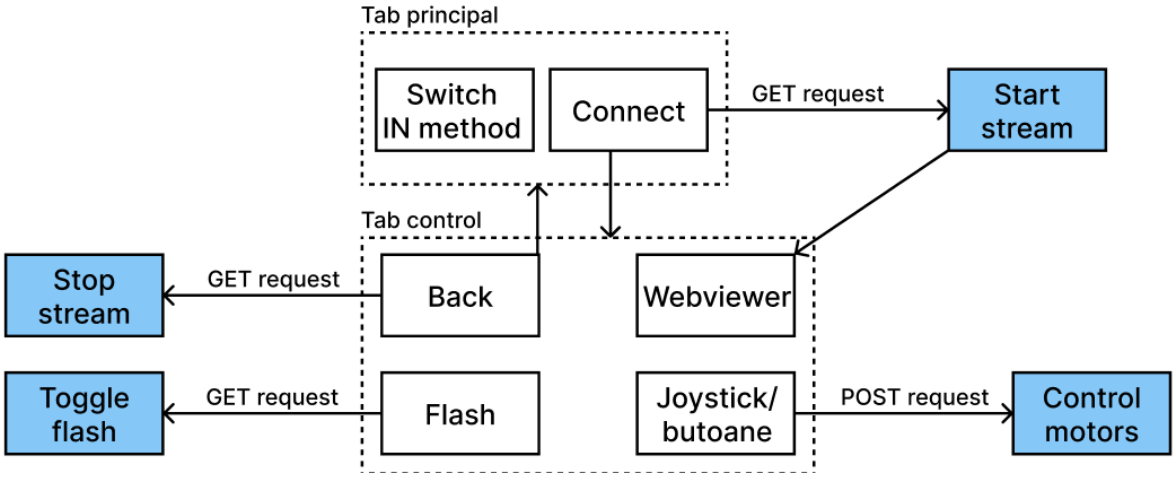

Figura 2.6: Diagramă Use-Case

Aplicația va fi formată din două *tab-uri*, *tab* principal va conține un buton ce va redirecționa utilizatorul în *tab-ul* de control și va iniția *stream-ul* camerei.

Tab-ul de control conține ecranul pentru *streaming*, un *joystick* (sau butoane) pentru controlul dronei, un buton de oprire ce încetează fluxului și redirecționează utilizatorul la *tab-ul* principal, un buton pentru aprinderea luminei *flash* al modulului, și un buton pentru rotirea camerei (în cazul în care camera trimite fluxul de imagini într-o rotație de sens nedorită).

Este implementat și un meniu de alegere a metodei de *input* în *tab-ul* principal; ce dispune de două metode: *joystick* sau butoane. Ambele metode comunică similar cu serverul când vine vorba de trimis date.

Butonul de rotire a camerei aplică un script JS (JavaScript) site-ului din *Web Viewer* (utilizând o variabilă globală pentru unghi):

```
var cssStyle = document.createElement("style");
cssStyle.innerHTML = "img { rotate: " + global_deg + "deg; }";
document.head.appendChild(cssStyle);
```

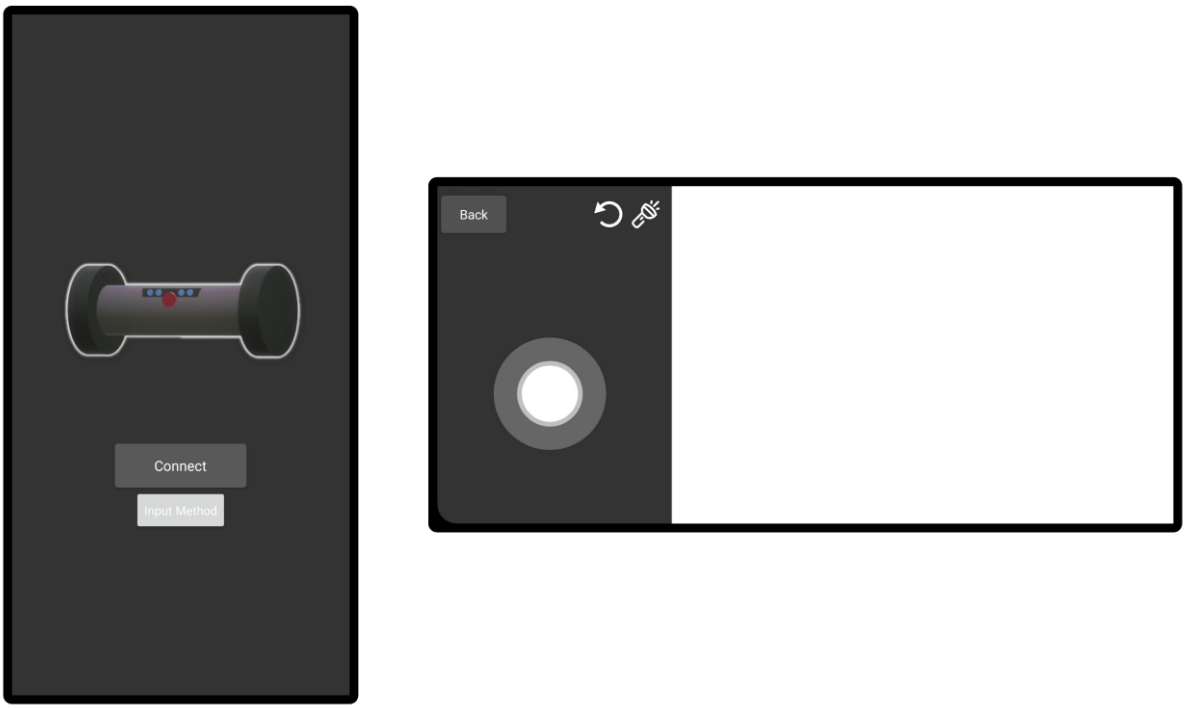

Figura 2.7: Aplicație mobil

Pentru a porni fluxul camerei, s-a folosit un **Web Viewer**, componentă folosită pentru a vizualiza pagini *web*, setat la adresa 'http://192.168.4.1' (ce este adresa IP a punctului de acces în mod implicit) ce va trimite implicit o cerere GET la portul 80.

Joystick-ul va trimite cererile de tip POST cu coordonatele x și y de fiecare dată când este mișcat; însă, o problemă ar fi următoarea: trimițând coordonatele la fiecare mică mișcare ar fi prea mult pentru server să se ocupe de toate cererile primite, creând astfel probleme de latență. Soluția este trimiterea coordonatelor după ce *joystick-ul* a fost mișcat după un anume prag; de exemplu, trimiterea mesajului dacă una din coordonate se schimbă cu zece unități în loc de una. Din moment ce *joystick-ul* în acest proiect poate lua coordonate de la -60 până la 60, putem lucra cu și trimite coordonate multiplu de 10 (de la -6 la 6), deci vitezele pot fi mapate între 0 și 6 (0 fiind viteză minimă iar 6 viteză maximă).

Butoanele trimit mesaje serverului de fiecare dată când se detectează o interacțiune cu unul din ele (*press/release*). Butoanele trimit întodeauna viteza maximă (deci variabile cu valorile -6, 6, sau 0 în cazul de față).

Ambele metode trimit datele exact la fel, printr-o metodă POST, formatate utilizând structura de cheie-valoare, specific antetului 'application/x-www-form-urlencoded', la adresa 'http://192.168.4.1:81/set' folosind componenta **Web**. Serverul primește aceste coordonate și le utilizează conform mapei de direcții.

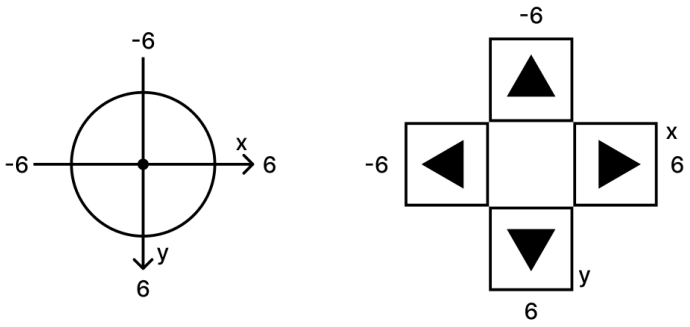

Figura 2.8: Exemplu valori control

2.5 Teste și probleme

După ce se pornește întrerupătorul de curent, după câteva secunde, modulul ar trebui să-și deschidă punctul de acces, acesta ar trebui să apară în lista de rețele Wi-Fi al dispozitivului ales sub numele 'MyESP32CAM'. După ce acesta s-a conectat, este gata să deschidă aplicația dedicată și să se conecteze la fluxul camerei.

Depinzând de distanța celor două dispozitive conectate, pot exista probleme de latență când vine vorba de fluxul camerei sau controalele dronei, nu există latență majoră la distanțe neobstrucționate de mai puțin de 5m. Mai mult de 10m se pot observa latențe sau chiar răspunderi neașteptate, chiar mai puțin dacă se află orice fel de zid obstructiv între cele două dispozitive.

Folosirea flash-ului în timpul controlului produce multă latență, posibil din motivul consumării mari de energie.

Probleme ce pot fi mitigate fie prin folosirea unei antene externe sau reducerea rezoluției camerei în partea de configurare.

O posibilă eroare întâlnită este *'Brownout detector was triggered'*, ce poate apărea din motivul alimentării incorecte sau insuficiente, cablu prea lung, conectare la 3,3V. Placa se resetează în astfel de situații pentru a evita posibile daune. Este recomandat de verificat sursa de alimentare, însă eroarea poate apărea și din defecte tehnice; dacă cauza nu se poate remedia, în astfel de situații se poate ignora eroarea (însă nu este recomandat) utilizând funcția 'WRITE_PERI_REG()' (din biblioteca 'soc/soc.h'), ce scrie direct pe adresa unui registru. Registrul pentru verificare de alimentare este dat de 'RTC_CNTL_BROWN_OUT_REG' (din biblioteca 'soc/rtc_cntl_reg.h').

```
WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
```

Concluzii

Rezultatul final al acestui proiect prezintă un prototip funcțional ce poate fi controlat de la distanță printr-o aplicație dedicată pe mobil. Pe parcursul acestui proiect, a fost prezentată documentația și folosirea microprocesoarelor ESP32, și o metodă de comunicare cu telefonul, facilitate de biblioteci și aplicații specializate, făcând partea de *setup* mai ușoară.

Deși proiectul demonstrează o implementare de bază, există potențiale îmbunătățiri ce pot fi explorate. La nivel de performanță, camera poate avea probleme de latență atunci când se folosesc alte funcționalități în mod repetat, modulul este conceput pentru proiecte de tip IoT simple, în care nu necesită un *stream* sau schimburi foarte volatile de valori.

ESP32-CAM, la momentul scrierii, are un succesor, ESP32-WROVER-E, lansată în 2020. Versiunea aceasta oferă pini adiționali, regulator de tensiune, un programator integrat, memorie RAM crescută (4MB de la 520KB) ce ar ajuta enorm cu problemele de întârziere. Tranziția celor două module se face ușor, acestea operând relativ la fel. Alte îmbunătățiri la nivel de design ar fi: o carcasă printată 3D, motoare *brushless*, microfon.

Bibliografie

- Neil Kolban, *Kolban's Book on ESP32*, 2017
- Rui Santos & Sara Santos, *ESP32 Web Server with Arduino IDE*
- Mitch Barnes & H.R. Evereft & Pavlo Rudakevych, ThrowBot: Design Considerations for a Man-Portable Throwable Robot
- Gunther Gridling & Bettina Weiss, *Introduction to Microcontrollers*, 2007
- ESP32-S fișă tehnică, `https://www.es.co.th/Schemetic/PDF/ESP32.PDF`
- Modul cameră OV2640 fișă tehnică, `https://docs.ai-thinker.com/_media/esp32/docs/ov2640_ds_1.8_.pdf`
- Protocol HTTP, `https://gndec.ac.in/~inderjeetsinghit/im_notes/im_theory/second_sessional/http_request_response.pdf`
- Franklin Lopez, Joystick Tutorial, `https://community.kodular.io/t/how-to-creat 58129`
- Modul L298N fișă tehnică, `https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf`
- Fisă tehnică motoare JGB37-520, `https://media.digikey.com/pdf/Data%20Sheets/Seeed%20Technology/108990017_Web.pdf`
- ESP32 documentare biblioteci, `https://github.com/espressif/arduino-esp32/tree/master`
- ESP32-CAM fișă tehnică, `https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf`
- Types of Batteries, `https://drive.google.com/file/d/1qDMY-5gRqfD1IOEmOs67Y view`

- Protocol TCP/IP, https://users.exa.unicen.edu.ar/catedras/comdat1/material/TP1-Ejercicio5-ingles.pdf
- Yash Sanghvi, How to disable brownout detector in ESP32 in Arduino, 2021, https://iotespresso.com/how-to-disable-brownout-detector-in-esp32-i

Anexe

Anexa 1

Exemplu configurare cameră

```
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = 5;
config.pin_d1 = 18;
config.pin_d2 = 19;
config.pin_d3 = 21;
config.pin_d4 = 36;
config.pin_d5 = 39;
config.pin_d6 = 34;
config.pin_d7 = 35;
config.pin_xclk = 0;
config.pin_pclk = 22;
config.pin_vsync = 25;
config.pin_href = 23;
config.pin_sscb_sda = 26;
config.pin_sscb_scl = 27;
config.pin_pwdn = 32;
config.pin_reset = -1;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size = FRAMESIZE_VGA;
config.jpeg_quality = 12;
```

```
config.fb_count = 1;
if (esp_camera_init(&config) != ESP_OK) {
    return;
}
```

Anexa 2

Camera stream handler

```
void handleStream(){
  watch = 1;
  String response =
    "HTTP/1.1 200 OK\r\n"
    "Content-Type: multipart/x-mixed-replace; boundary=frame\r\n\r\n";
  streamServer.sendContent(response);
  while (watch == 1) {
    camera_fb_t * fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      break;
    }
    String header =
      "--frame\r\n"
      "Content-Type: image/jpeg\r\n"
      "Content-Length: " + String(fb->len) + "\r\n\r\n";
    streamServer.sendContent(header);
    streamServer.sendContent((const char*)fb->buf, fb->len);
    streamServer.sendContent("\r\n");
    esp_camera_fb_return(fb);
  }
  streamServer.sendContent("--frame--\r\n");
}
```