Anexa 5 la Procedura de înscriere on-line la examenele de finalizare a studiilor

PROIECT DE DIPLOMĂ

Îndrumător proiect/Coordonator ştiinţific,
Conf. Dr. Ing. Răzvan Constantin ŞOLEA

Absolvent,
Andrei-Petrişor LĂBUŞ

Galaţi
Anul 2024

PROGRAMUL DE STUDII: Ingineria Sistemelor
SPECIALIZAREA: Automatică şi Informatică Aplicată

Realizarea și implementarea unui algoritm pentru controlul unei drone aeriene

Îndrumător proiect/Coordonator ştiinţific,
Conf. Dr. Ing. Răzvan Constantin ȘOLEA

Absolvent,
Andrei-Petrişor LĂBUŞ

Galaţi
Anul 2024

Rezumat

Industria dronelor a crescut semnificativ în ultimii ani, devenind esenţială în diverse domenii precum agricultură, securitate, logistică şi filmare. Această creştere este susţinută de avansurile tehnologice care au permis dezvoltarea de drone mai eficiente şi mai accesibile. Un aspect crucial al funcţionării dronelor este sistemul de control al altitudinii, esenţial pentru stabilitatea şi precizia zborului. Aceste sisteme, care utilizează senzori barometrici, GPS şi algoritmi avansaţi, sunt extrem de precise şi pot recunoaşte persoane, zburând cu o precizie incredibilă chiar şi în cazul dronelor mici, de doar 5 cm. În plus, aceste sisteme sunt integrate cu inteligenţa artificială, îmbunătăţind performanţa şi adaptabilitatea în diverse condiţii de operare. Această evoluţie continuă stimulează inovaţia şi extinde aplicabilitatea dronelor în noi domenii, evidenţiind potenţialul imens al acestor tehnologii în viitorul apropiat.

Această lucrare de licenţă se va concentra pe analiza detaliată a componentelor utilizate în construcţia unei drone, oferind specificaţii tehnice şi funcţionale pentru fiecare piesă. Printre componentele discutate se vor număra motoarele, elicele, controlerul de zbor, senzorii şi sistemele de comunicaţie. Pe lângă descrierea fizică, vor fi examinate şi motivele pentru care au fost alese aceste componente specifice, inclusiv costul, performanţa şi compatibilitatea.

Un capitol important al lucrării va fi dedicat testării performanţelor sistemului video al dronei. Aceste teste vor fi efectuate pentru a evalua calitatea imaginii, stabilitatea transmisiei şi capacitatea de recunoaştere a obiectelor, utilizând metode avansate precum filtrul Kalman, care va contribui la îmbunătăţirea preciziei şi fiabilităţii datelor video. Aceste teste vor fi esenţiale pentru validarea eficienţei sistemului video în diferite condiţii de zbor.

În plus, va fi dezvoltat un program în limbajul Python, capabil să preia datele de la telecomandă, să le proceseze şi să utilizeze un algoritm PID (Proporţional-Integral-Derivativ) pentru a controla puterea motoarelor. Acest sistem PID va calcula ajustările necesare pentru menţinerea altitudinii dorite, asigurând stabilitatea dronei în zbor. Programul va integra datele de la senzori precum accelerometre, giroscoape şi barometre pentru a realiza o calibrare precisă şi rapidă a altitudinii, răspunzând eficient la modificările de mediu şi la comenzi.

De asemenea, se vor discuta alte metode de programare şi algoritmi utilizaţi în dezvoltarea acestui sistem, explicând principiile teoretice şi implementările practice. Această abordare cuprinzătoare va demonstra cum tehnologiile moderne pot fi aplicate pentru a crea drone eficiente şi autonome, capabile să îndeplinească sarcini complexe cu un grad ridicat de precizie şi fiabilitate. În concluzie, lucrarea va evidenţia impactul acestor tehnologii asupra viitorului dronelor şi va propune direcţii viitoare pentru cercetare şi dezvoltare în acest domeniu.

CUPRINS

LISTA FIGURILOR

LISTA TABELELOR

INTRODUCERE

Conceptul de dronă reprezintă un vehicul aerian, care poate zbura fără a avea un pilot uman la bord. Acestea pot fi controlate de un operator uman, de la distanță, prin diverse mijloace sau prin funcționarea autonomă pe baza unor programe sau planuri de zbor, utilizând anumiți algoritmi atât matematici cât și pe baza inteligenței artificiale.

Prima "dronă" din istorie a fost construită la începutul secolului al XX-lea, de un fotograf, care a echipat mai multe zmeie cu o cameră ce cântarea 22 de kilograme pentru a face o poză de la câteva sute de metrii a orașelor San Francisco, Chicago și New York City.[1]

Conceptul de dronă modernă, avion-țintă controlat cu telecomandă, a fost folosit pentru prima dată în al Doilea Război Mondial și a avut ca rol antrenarea piloților militari. Abia în anii 1960 si 1970, cu ajutorul evoluției tehnologiei, dronele au putut fi folosite pentru misiuni de recunoaștere și supraveghere [2].

În zilele noastre, există o multitudine de senzori folosiți pentru a spori senzitivitatea vehiculelor aeronautice fără pilot [3] precum:

- IMU (unități de măsură inerțiale) este un dispozitiv electronic, capabil să măsoare accelerația si rotația. Acesta este folosit în special pentru a calcula poziția și viteza, fiind un ansamblu de senzori conceput, de regulă, dintr-un accelerometru, un giroscop și un magnetometru.
- GPS, având rolul de a primi date de la sateliți, date determinate prin triangulare. Acest tip de senzori poate citi atât locația și viteza, cât și altitudinea. Este folosit, de cele mai multe ori, la sistemele mai complexe, pentru a crește acuratețea poziționării.
- Senzorii ultrasonici, folosiți în general la jucării, au rolul de a detecta altitudinea relativă a aeronavei, dar pot fi folosiți și la detectarea obstacolelor. Aceștia sunt o alternativă ieftină în locul senzorilor GPS.
- Senzorii laser, exact ca în cazul celor ultrasonici, sunt folosiți pentru a detecta altitudinea sau obstacolele.
- Camera video, utilizată în special pentru supravegherea zborului, este un punct cheie în multe industrii, începând de la posibilitățile de cinematică la detectarea și recunoașterea obiectelor, până la crearea sistemelor de detectare a obstacolelor.

Conceptul de vehicul aeronautic fără pilot constituie o temă de mare actualitate, având în vedere dezvoltarea rapidă a acestei industrii și potențialul considerabil de inovare pe care îl prezintă. Dronelor li se descoperă constant noi aplicații în diverse domenii, de la logistică și cartografiere, la agricultură de precizie și securitate.

În acest context, ideile inovatoare pot avea un impact semnificativ asupra viitorului industriei. Tinerii inovatori, datorită perspectivelor lor unice și neconvenționale, pot aduce contribuții esențiale în ceea ce privește îmbunătățirea autonomiei, navigației și eficienței energetice a dronelor. Încurajarea creativității și susținerea cercetării în rândul acestora sunt vitale pentru progresul continuu al tehnologiei și pentru formarea unei generații de profesioniști capabili să modeleze viitorul acestei industrii.

Conform unui studiu realizat de Vision REASEARCH REPORTS [4] reiese faptul că în industria dronelor, la nivel global, este prevăzută o creştere anuală cu până la 39.02%.

Expansiunea rapidă a pieţei serviciilor de date pentru drone poate fi atribuită mai multor factori cheie de creştere. În primul rând, adoptarea tot mai mare a dronelor în diverse industrii, cum ar fi agricultura, construcţiile şi monitorizarea mediului, stimulează cererea pentru servicii de date.

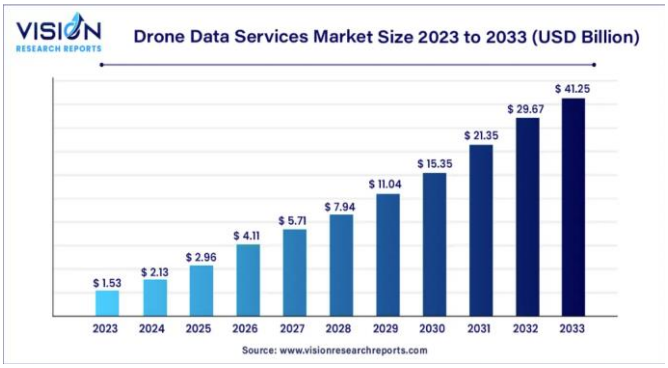

Figura 0.1: Prezicerea pieţei de date a dronelor din 2023 până în 2033[4]

Scopul prezentei lucrări constă în dezvoltarea unui sistem avansat de control al altitudinii pentru un vehicul aeronautic fără pilot, realizat prin conceperea şi implementarea unui prototip. Acesta va integra o serie de senzori, care vor furniza date esenţiale privind parametrii de zbor. Pe baza acestor date, vor fi aplicate formule matematice şi algoritmi de control avansaţi, care vor asigura stabilitatea şi precizia necesare pentru menţinerea altitudinii dorite.

În mod specific, lucrarea va aborda selectarea şi calibrarea senzorilor adecvaţi, cum ar fi cei de presiune barometrică, accelerometrele şi giroscoapele, pentru a obţine măsurători precise şi fiabile ale altitudinii şi poziţiei vehiculului.

De asemenea, vor fi dezvoltaţi algoritmi de control, cum ar fi controlul proporţional-integral-derivativ (PID) şi algoritmi de control adaptiv, care să permită ajustarea continuă a parametrilor de zbor pentru a compensa variaţiile şi perturbările externe.

Există o multitudine de probleme întâlnite pe parcursul proiectului, pornind de la compatibilitatea pieselor, în cazul alegerii greşite ale acestora, continuând cu probleme de interpretare a datelor, formule matematice greşite si terminând cu problematica scrierii codului.

În această lucrare, au fost utilizaţi mai mulţi algoritmi avansaţi pentru a asigura controlul precis al altitudinii unui vehicul aeronautic fără pilot. Printre aceştia se numără algoritmii de control proporţional-integral-derivativ (PID), algoritmi de reducere a zgomotului, precum filtrul Kalman, şi algoritmi complementari. Algoritmul PID a fost ales pentru capacitatea sa de a ajusta continuu parametrii de zbor, asigurând stabilitatea şi reducerea erorilor. Filtrul Kalman a fost integrat pentru a îmbunătăţi acurateţea măsurătorilor prin estimări precise ale stării sistemului, reducând influenţele zgomotului şi incertitudinilor.

Algoritmii complementari au fost folosiţi pentru a combina datele provenite de la diferiţi senzori, precum accelerometrele şi giroscoapele, oferind astfel o estimare robustă şi coerentă a altitudinii şi orientării vehiculului.

Prin utilizarea acestor algoritmi, sistemul dezvoltat asigură un control stabil şi precis al altitudinii, demonstrând eficienţa şi robusteţea în urma testelor practice şi simulărilor.

CAPITOLUL 1. ARHITECTURA HARDWARE A DRONEI

1.1. Noţiuni introductive

Chiar dacă acest subiect poate fi destul de greu de abordat la o prima vedere, cu ajutorul unor informaţii bine structurate, de actualitate şi corecte, orice începător în domeniul construcţiei de drone poate fi capabil să realizeze un sistem funcţional şi capabil de zbor.

O întrebare bună poate fi "Nu e mai simplu să achiziţionezi direct o dronă?", dar asamblând un vehicul aeronautic fără pilot poate învăţa un utilizator cunoştinţe din diverse domenii, precum: compatibilitatea dintre fiecare piesă şi fiecare protocol, cum se poate repara un astfel de sistem în cazul unei defecţiuni , cum se calibrează un sistem de control PID şi multe altele.

Este adevărat că pentru o persoană nefamiliarizată cu acest subiect, construcţia primei drone este un proces anevoios şi poate dura de la câteva zile până la câteva luni, luând în calcul complexitatea proiectului şi posibilitatea de personalizare. [5]

1.2. Ustensile esenţiale

Pentru un astfel de proiect trebuie luată în considerare necesitatea diferitelor ustensile necesare în construirea unui vehicul aeronautic fără pilot. Probabil cea mai importantă unealtă este pistolul de lipit, sau staţia de lipit, deoarece cu excepţia componentelor costisitoare create special pentru a fi conectate prin mufe speciale, fiecare piesă ce trebuie încadrată într-un buget va avea nevoie de suduri. Pe lângă un pistol de lipit vor fi necesare diferite chei hexagonale şi şurubelniţe. Este recomandat totodată prezenţa benzii izolatoare, a unui multimetru, unei foarfeci sau a unui dispozitiv de tăiere si dezizolare a cablurilor.

1.3. Cadrul

Cadrul este reprezentat de o structură solidă pe care se montează toate componentele. Dimensiunea şi materialul din care sunt create impactează foarte tare durabilitatea, modul de zbor, viteza maximă şi acceleraţia maximă a dronei, dar şi timpul de zbor şi capabilitatea de manevrare a acesteia.

Mărimea cadrului este esenţială, deoarece aceasta limitează dimensiunile maxime ale elicelor, care mai târziu impactează alegerea motoarelor.

Cele mai des întâlnite materiale folosite în domeniul construcţiilor de vehicule aeronautice fără pilot sunt fibra de carbon, aluminiul, plasticul şi lemnul. Fibra de carbon este un material uşor, rigid, şi rezistent la coroziune, dar totodată scump şi casant. Aluminiul este ieftin, puternic, uşor de îndoit, dar mai greu şi predispus la vibraţii. Plasticul este flexibil, durabil şi ieftin, dar şi mai puţin rigid şi afectat de căldură. Lemnul este ieftin, si uşor de prelucrat, dar este de asemenea greu, slab din punct de vedere a rezistenţei şi neomogen. [6]

Considerând aceşti factori, în acest proiect a fost ales un cadru de fibră de carbon, pe care s-au montat un set de suporţi din plastic printaţi 3D pentru a atenua căderile de la distanţe mari sau aterizările forţate.

Figura 1.1: Cadrul din fibră de carbon "Mark 4"

1.4. Controlerul de zbor

Dacă cadrul dronei poate fi considerat drept „scheletul" sau structura de rezistenţă, atunci controlerul de zbor este mai mult ca sigur „creierul" sistemului, deoarece acesta are scopul atât de a da comenzi motoarelor, sau controlerelor electronice de viteză, cât şi a citi datele de la senzori, a le interpreta şi a le filtra cu ajutorul unor algoritmi complexi.

Toate controlerele de zbor sunt echipate cu senzori precum: giroscopul şi accelerometrul. Unele includ şi senzori adiţionali, cum ar fi cel de presiune barometrică sau magnetometrul, pentru a sprijini zborurile autonome. De asemenea, acestea pot fi extinse cu piese externe, cum ar fi GPS-ul, LED-urile, servo-motoarele şi sistemele de transmisie video.[7]

Există mai multe tipuri de unităţi de control şi se clasifică în 3 mari clase[8]:

a) Pentru începători, aceste controlere sunt pre-programate şi pot fi folosite imediat fără procese complicate de cablare sau calibrare. Sunt ideale pentru piloţii noi şi au un software prietenos şi opţiuni de reglaj de bază.

Exemplu: *Seria KK Flight Controller (KK2.0, KK2.1) şi Beta Flight: Open Pilot CC3D*

b) Pentru acrobaţii şi curse, fiind compacte, uşoare şi au procesoare rapide, fiind ideale pentru manevre rapide şi agile, precum şi pentru curse de drone.

Exemplu: *SP Racing F3 (versiunea ACRO) şi Naze32 6DOF Flight Control Board*

c) Pentru experţi, aceste controlere sunt dotate cu procesoare avansate, o gamă largă de senzori şi opţiuni extinse pentru adăugarea de noi instrumente şi senzori.

Exemplu: *Seria APM Flight Controller (exemplu: APM 2.8)*

Pentru acest proiect s-a optat alegerea controlerului SpeedyBee F405 V3, cea mai populară unitate de zbor a anului 2023. Cu un preţ accesibil de 350 de lei, este echipat cu 4 controlere electronice de viteză folosite la manevrarea motoarelor, fiecare având un curent maxim de 50 de amperi. Acesta are o cutie neagră pentru înregistrarea datelor de zbor, Bluetooth pentru a configura unitatea de control cu ajutorul telefonului, capabilităţi de interpretare video (pentru anumite protocoale) şi afişare în timp real a datelor în funcţie de preferinţele utilizatorului.

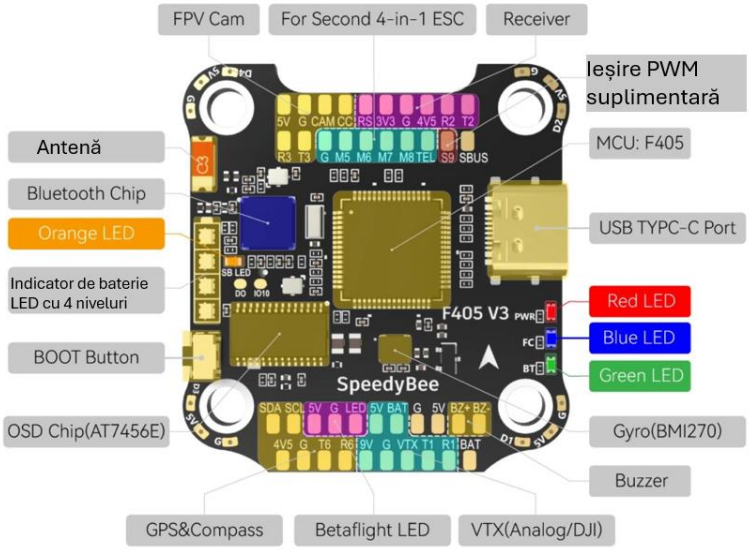

Figura 1.2: Schema tehnică a controlerului „SpeedyBee F405"[9]

1.5. Motoare + ESC

Pentru alegerea corectă a motoarelor este necesară cunoaşterea şi explorarea acestora atât la interior cât şi la exterior, este necesară cunoaşterea modului de construcţie, caracteristicile de proiectare şi factorii care influenţează performanţa şi eficienţa. Doar având un set solid de cunoştinţe şi o idee bine stabilită de proiectare a dronei se pot alege motoarele ideale.

În lumea vehiculelor aeronautice fără pilot există, de regulă, două tipuri de motoare folosite: cele cu perii şi cele fără perii. În general, industria are tendinţa de a alege motoarele fără perii, fiind mai rezistente, mai puternice şi mai eficiente din punct de vedere al consumului de energie electrică, pe când motoarele cu perii sunt folosite, de regulă, la jucăriile ieftine, deoarece sunt cele mai eficiente din punct de vedere a preţurilor.

Când se aproximează greutatea totală a unei drone, trebuie luat în calcul toate componentele acesteia: cadru, unitatea de control, motoare, elice, antene, receptori şi transmiţători, baterii, controlerele de motoare şi aşa mai departe. Chiar dacă estimarea nu trebuie sa fie 100% corectă, aceasta este esenţială, fiind mai bine să se supraestimeze şi să se aleagă nişte motoare cu mai multă putere .

Pentru a fi sigur că aeronava poate zbura, este necesară calcularea puterii minime pentru decolare, creată de motoare în combinaţie cu elicele. O regulă de bază este cea de a avea o putere de ridicare dublă faţă de cea minimă pentru a susţine

greutatea dronei. În cazul celor de curse sau de zboruri acrobatice, această raţie se poate ridica pană la 10 la 1 sau 16 la 1. În cazul în care forţa de ridicare a sistemului nu este suficientă, aceasta poate determina apariţia dificultăţilor ale vehiculului la decolarea de la sol şi a controlului întârziat.

De exemplu, pentru o dronă de 1.5 Kg, este recomandat ca forţa maximă generată de motoare să fie de măcar 3Kg, adică 750 de grame pentru fiecare motor, dar mai multă putere este de fiecare dată bine venită. Cu cât raţia de putere-greutate este mai mare, agilitatea şi acceleraţia dronei cresc, dar odată cu acestea creşte şi dificultatea de controlare a sistemului aeronautic.[10]

Pentru a conecta motoarele la o unitate de zbor, este necesar un ESC (electronic speed controller), adică un controler de motor fără perii. Spre deosebire de motoarele cu perii, care au doar două fire, cele fără perii au trei şi se conectează la un ESC în orice ordine. Pentru a inversa direcţia de rotaţie, trebuie doar să se inverseze două din cele trei fire şi uneori este posibil să se realizeze schimbarea direcţiei direct din componenta software.[11]

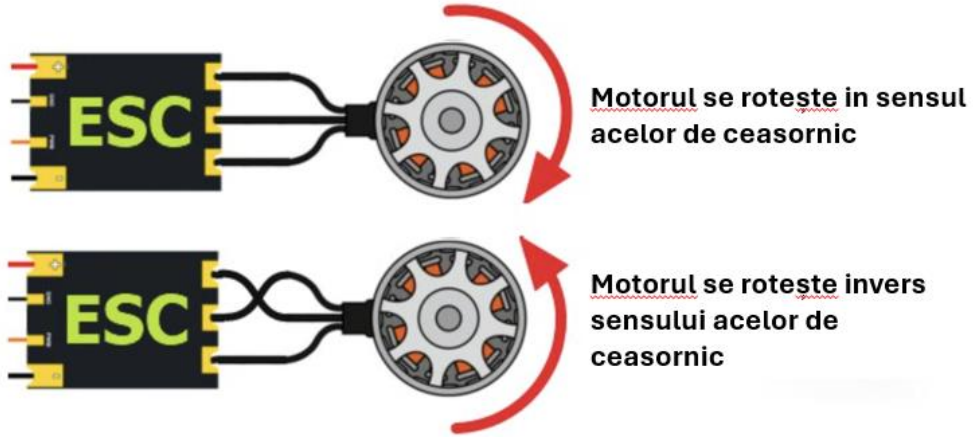

Figura 1.3: Schema de conectare a unui motor la ESC [10]

Înţelegerea si decodificarea datelor tehnice a unui motor fără perii necesită anumite cunoştinţe ce vor fi prezentate în următoarele rânduri, astfel făcând mai uşoară alegerea corecta a acestora.

Dimensiunile motorului sunt reprezentate de numărul format din patru cifre, ca de exemplu 2507, unde primele două numere reprezintă lăţimea statorului (diametrul), iar ultimele reprezintă înălţimea statorului, ambele fiind măsurate în milimetrii.

Statorul este partea staţionară a motorului constând în “polii” acestuia înfăşuraţi cu fir de cupru. Aceşti poli sunt realizaţi din mai multe plăci metalice subţiri lipite cu ajutorul unor straturi foarte subţiri de izolaţie.

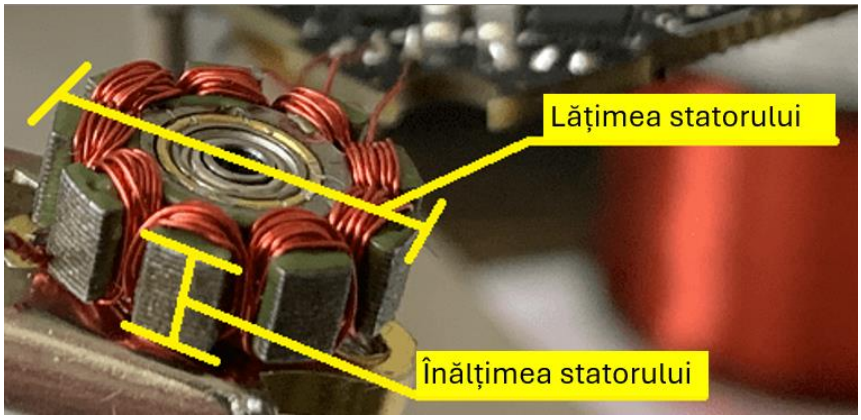

Figura 1.4: Statorul unui motor fără perii[12]

Motoarele mai late au mai multă inerţie când se învârt, deoarece masa motorului este mai departe de axul de rotaţie, necesitând mai multă energie pentru a schimba RPM-ul (numărul de rotaţii pe minut). În consecinţă, motoarele mai late şi mai scunde sunt de obicei mai puţin receptive decât cele mai înguste si mai înalte, chiar dacă statorul acestora are acelaşi volum si generează aceeaşi turaţie. Motoarele mai late şi mai scurte au, de asemenea, magneţi mai mici, ceea ce poate reduce puterea motorului.

Cu toate acestea, motoarele mai largi oferă o răcire mai bună datorită suprafeţei mai mari, temperatura fiind esenţială pentru performanţa motorului. Pe măsură ce un motor se încălzeşte, capacitatea sa de a genera flux magnetic scade, afectând eficienţa şi producţia de cuplu.

În esenţă, lăţimea şi înălţimea unui stator de motor reprezintă un echilibru între reacţie şi răcire. Decizia depinde de stilul de zbor.

Statoarele mai late permit, de asemenea, rulmenţi mai mari, ceea ce poate îmbunătăţi eficienţa, netezimea şi longevitatea.

Statoarele mai mari nu sunt întotdeauna mai bune. De exemplu, motoarele 2207 pot gestiona elice tipice de 5 inch, dar utilizarea motoarelor 2506 mult mai grele de acelaşi KV pot să nu ofere beneficii vizibile, deoarece ar produce aceeaşi forţă folosind aceleaşi elice sau chiar ar oferi o reacţie mai slabă din cauza greutăţii. Pentru a îmbunătăţi performanţa fără a adăuga greutate, luaţi în considerare motoarele KV mai mari. Cu toate acestea, motorul 2506 din acest exemplu ar funcţiona probabil mai bine pentru elicele de 6 inch decât 2207 din cauza cerinţelor crescute de cuplu.

Motoarele cu cuplu mare oferă schimbări rapide ale turaţiei şi timp de răspuns mai rapid, ceea ce duce la mai puţine oscilaţii şi răspunsuri mai rapide.

Cuplul motorului depinde de mai mulţi factori, printre care:

- Dimensiunea statorului (volum);
- Materiale: tipul magneţilor şi calitatea înfăşurărilor de cupru;
- Construcţia motorului: cum ar fi spaţiul de aer, numărul de poli etc.;
- Deoarece motoarele fără perii au specificaţii şi design similare, în ultimii ani, dimensiunea statorului este cea mai simplă modalitate de a cuantifica cuplul.

Dimensiunea statorului poate fi calculată folosind formula volumului unui cilindru:

$$volum = pi * raza^2 * înălțime \quad (1.1)$$

Cu cât volumul statorului este mai mare, cu atât un motor poate genera mai mult cuplu. Comparând un motor 2306 cu un volum de 2492.85, un motor 2207 are un cuplu mai mare.

Când se alege un motor, se compară volumul și greutatea statorului motorului. Este de preferat un motor mai ușor, cu același volum, presupunând că alți factori sunt egali. Deci, de ce nu se alege cel mai mare motor disponibil? Răspunsul constă în greutate. Motoarele cu volume mai mari ale statorului sunt mai grele, așa că depinde foarte mult de aplicație.

Singurul moment în care este preferat un motor mai puțin puternic (cu cuplu mai mic) este atunci când netezimea este prioritară față de reacție. Motoarele cu cuplu mare pot schimba RPM atât de repede încât se pot simți neuniforme și mai puțin netede. Ele pot crea, de asemenea, mai multe vârfuri de tensiune și zgomot electric în sistemul de alimentare, care pot afecta performanța giroscopului și performanța generală a zborului, dacă filtrarea zgomotului nu este optimă, ceea ce duce la oscilații.

„KV" indică numărul de rotații pe minut (rpm) atunci când se aplică 1 V (un volt) fără nicio sarcină sau elice atașată la motor. De exemplu, un motor de 2300KV alimentat de o baterie LiPo 3S (12.6V) se va învârti la aproximativ 28.980 RPM fără elice montate (2300 x 12.6). KV este de obicei o estimare aproximativă specificată de producătorul motorului.

Dacă un motor cu KV mare este asociat cu o elice excesiv de mare, motorul va încerca să se învârtească rapid, așa cum ar face cu o elice mai mică, necesitând un cuplu mai mare. Această cerere crescută de cuplu va duce la un consum mai mare de curent și la generarea de căldură. Supraîncălzirea poate duce la arderea motorului, deoarece învelișul bobinei se poate topi și poate provoca scurtcircuitări electrice în interiorul motorului. De aceea, un motor KV mai mare este probabil să funcționeze mai fierbinte decât un motor KV mai mic de aceeași dimensiune.

Motoarele KV mai mari au o constantă de cuplu mai mare, ceea ce înseamnă că au nevoie de mai mult curent pentru a genera aceeași cantitate de cuplu în comparație cu un motor KV mai scăzut. Pentru a genera aceeași cantitate de cuplu, motorul KV mai mare necesită mai mult curent, ceea ce duce la pierderi suplimentare în ESC, baterie și fire. În plus, se acumulează mai multă căldură în motor datorită curentului mai mare, este generat mai puțin flux magnetic. În general, un motor KV mai mare este mai puțin eficient dacă ar fi să se zboare cu aceeași viteză ca și motorul KV mai mic.

Prin urmare, este o idee bună să se evite exagerarea cu KV, fiind recomandat să fie menținut moderat. Acest lucru este deosebit de important atunci când se construiește un vehicul aeronautic fără pilot cu rază lungă de acțiune care prioritizează eficiența și durata timpului de zbor.

Când cineva caută motoare pentru dronă, poate întâlni specificaţii precum „12N14P” pe cutie. Aceste numere au o semnificaţie specifică: numărul care precedă litera „N” indică numărul de electromagneţi (poli) din stator, în timp ce numărul care precedă litera „P” reprezintă numărul de magneţi permanenţi din clopoţel.

Diferitele dimensiuni ale motoarelor au un număr variabil de poli; de exemplu, motoarele 22XX şi 23XX au în general 12 poli şi 14 magneţi.

Numărul de poli influenţează direct performanţa motorului. Dacă există mai puţini poli, se poate încorpora mai mult conţinut de fier în stator, ceea ce duce la o putere mai mare. Totuşi, un număr mai mare de poli asigură un câmp magnetic distribuit mai uniform. Acest lucru rezultă într-un motor care funcţionează mai lin şi permite un control mai fin al rotaţiei clopoţelului.

Pentru a determina dimensiunea ideală a motorului, se urmăreşte această secvenţă:

Dimensiune cadru => Dimensiune elice => Dimensiune motor (1.2)

Identificarea dimensiunii cadrului este primul pas. Dimensiunea cadrului constrânge dimensiunea elicei, iar fiecare dimensiune a elicei necesită o turaţie diferită a motorului pentru a genera forţă eficient. Aici intervine valoarea KV a motorului.

Tabelul de mai jos oferă un ghid general. Nu este o regulă rigidă, deoarece este posibil să găsiţi oameni care folosesc motoare KV puţin mai mari sau mai mici decât sugerează tabelul. Cu toate acestea, serveşte ca un bun punct de plecare. Acest tabel presupune alimentarea dronei cu baterii 4S LiPo, iar dimensiunea cadrului se referă la distanţa diagonală de la un motor la celălalt.

Tabelul 1.1. Alegerea motoarelor pentru o baterie de tip 4S

Dimensiune cadru	**Dimensiune elice**	**Dimensiune motor**	**KV motor**
180-220mm	4-5 inch	22XX	2300-2700
220-250mm	5-6 inch	23XX	2000-2500
250-300mm	6-7 inch	24XX	1700-2300
300-350mm	7-8 inch	25XX	1400-2000

Aceste valori sunt orientative şi pot varia în funcţie de preferinţele personale şi de specificaţiile tehnice exacte ale componentei folosite. Alegerea finală ar trebui să ţină cont de testările şi ajustările efectuate pentru a obţine performanţa dorită.

1.6. Elice

Elicele sunt „eroii” necunoscuţi ai unei drone. Ele joacă un rol crucial în generarea de forţă care ridică drona de la sol şi îi permite să se mişte în direcţii diferite. Cu toate acestea, mulţi piloţi de drone trec adesea cu vederea importanţa alegerii elicelor potrivite, ceea ce duce la probleme precum creşterea zgomotului, reducerea timpului de zbor sau chiar defectarea motorului.

Elicele sunt proiectate pentru scopuri diferite, pasul, forma şi materialul joacă toate un rol în performanţa şi caracteristicile de zbor.

S-a descoperit faptul că elicele HQ sunt unele dintre cele mai fine şi mai uşor de reglat. Ele par a fi mai echilibrate şi produc mai puţine vibraţii. De asemenea, triblades a fost un favorit faţă de cele twinblades când vine vorba de netezimea zborului.

Elicele sunt proiectate să se rotească fie în sensul acelor de ceasornic (CW) fie în sens invers acelor de ceasornic (CCW). Într-un vehicul de zbor, două motoare se rotesc în sens orar, iar celelalte două se învârt în sens invers, deci este important să potriviţi elicele cu motoarele în funcţie de direcţia de rotaţie dorită.

Pe o dronă, plasarea descentrată a elicelor produce atât tracţiune, cât şi rotaţie în jurul centrului dronei. Pentru a contracara această rotaţie, este necesară folosirea a două elemente de elice CW şi două CCW.

La achiziţionarea de elice, acestea vin de obicei în perechi de CW şi CCW.

Pentru a genera o tracţiune în jos, ca drona să decoleze, elicele ar trebui să se învârtească într-un mod care să permită marginii înainte să taie mai întâi aerul, aerul scăpând apoi prin marginea de fugă. Se poate determina cu uşurinţă direcţia unei elice identificând marginea sa anterioară, care este adesea etichetată ca CW sau CCW pe lamă.

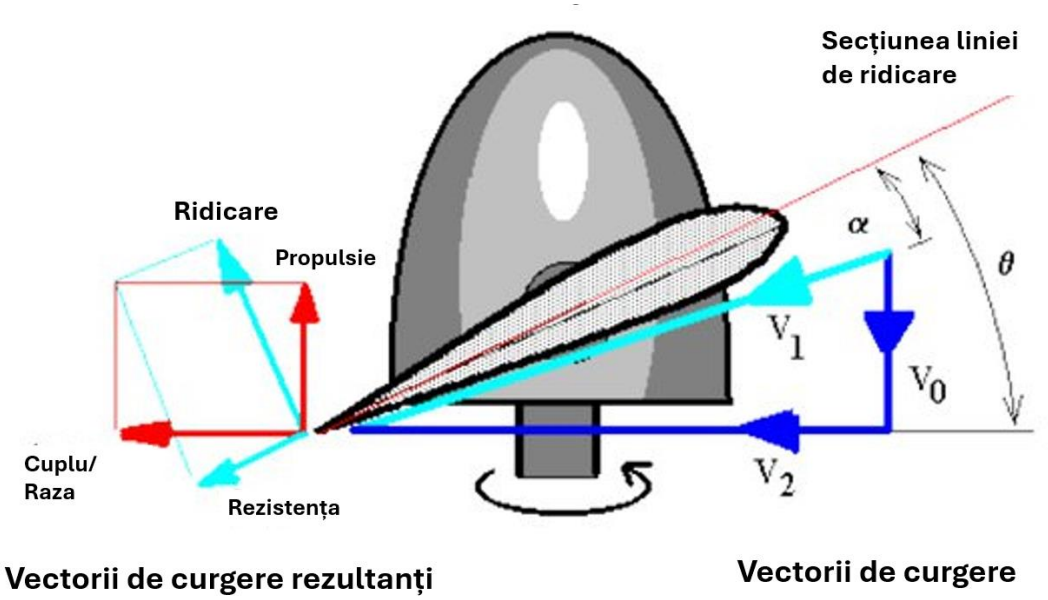

Figura 1.5: Imagine reprezentată a unei elice [13]

1.7. Bateriile

1.7.1. Înțelegerea bazelor bateriilor LiPo

Bateriile LiPo sunt preferate pentru drone datorită raportului lor excelent între putere și greutate. Selectarea bateriei potrivite necesită înțelegerea specificațiilor cheie și a terminologiei.

LiPo funcționează în siguranță într-un interval de tensiune specific, de obicei între 3.0V și 4.2V. Supraîncărcarea peste 4.2V poate fi periculoasă și poate duce la incendii, iar descărcarea sub 3V poate degrada iremediabil performanța bateriei. Este recomandat să se oprească descărcarea la 3.5V pe celulă pentru a prelungi durata de viață a bateriei.

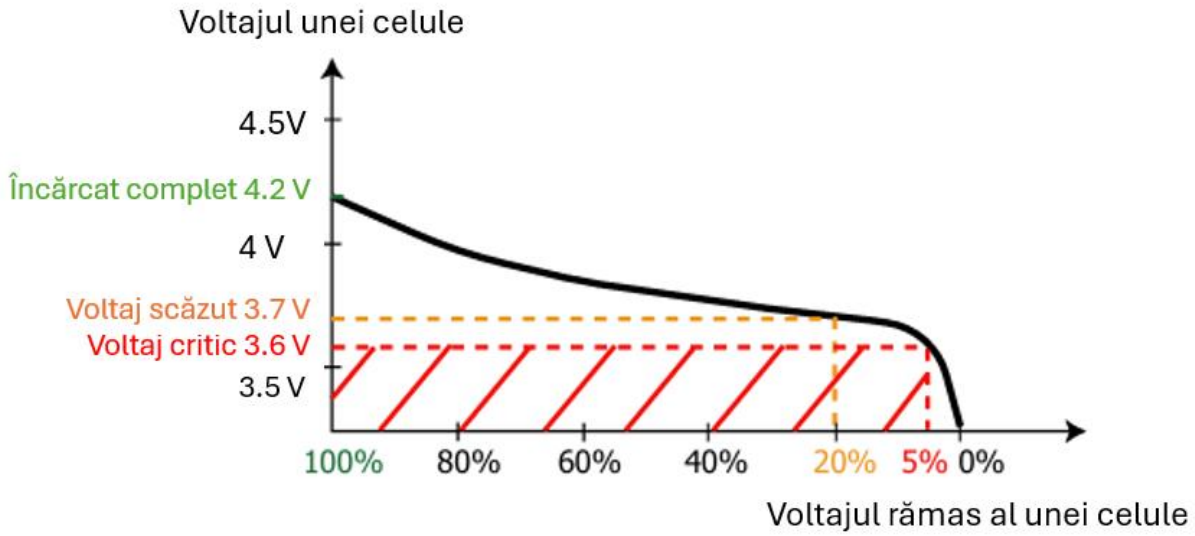

Figura 1.6: Graficul de funcționare a unei celule

Bateriile LiPo pot avea multiple indicate de ratingul „S”:
1S = 1 celulă = 3.7V
2S = 2 celule = 7.4V
3S = 3 celule = 11.1V
4S = 4 celule = 14.8V
5S = 5 celule = 18.5V
6S = 6 celule = 22.2V

Tensiunea bateriei afectează direct viteza motorului, astfel încât folosirea unei baterii cu un număr mai mare de celule poate crește puterea dronei, cu condiția ca drona să suporte tensiunea mai mare. Totuși, mai multe celule înseamnă și o baterie mai grea și mai scumpă.

Capacitatea unei baterii LiPo, măsurată în mAh (miliamperi-oră), arată cât de mult curent se poate extrage continuu din baterie timp de o oră până se golește. Creșterea

capacității bateriei poate oferi un timp de zbor mai lung, dar înseamnă și o baterie mai mare și mai grea.

Ratingul C reprezintă curentul maxim care se poate extrage în siguranță dintr-o baterie LiPo fără a o deteriora:

$$\textit{Curent Maxim de Extracție} = \textit{Capacitate} \times \textit{Ratingul C} \qquad (1.3)$$

Depășirea curentului specificat de ratingul C nu este recomandată deoarece bateria se poate supraîncălzi, poate crește rezistența internă în timp, poate scurta durata de viață a bateriei sau, în cazuri extreme, poate provoca un incendiu. Ratingurile C mai mari oferă o performanță mai bună pentru dronele care necesită multă putere, dar adaugă și greutate.

Rezistența internă (IR) a unei baterii sugerează cât de mult se opune curentului. O rezistență internă mai mică înseamnă o livrare mai eficientă a puterii către dronă. Monitorizarea în timp este utilă pentru a determina când o baterie LiPo trebuie retrasă din uz.

Bateriile LiPo au două seturi de fire/conectori: conectorul de descărcare (firul principal) și conectorul de echilibrare (firul de echilibrare). Conectorul de echilibrare permite monitorizarea și echilibrarea tensiunii fiecărei celule în timpul încărcării. Conectarea corectă a firului de echilibrare înainte de încărcare este esențială pentru siguranță.

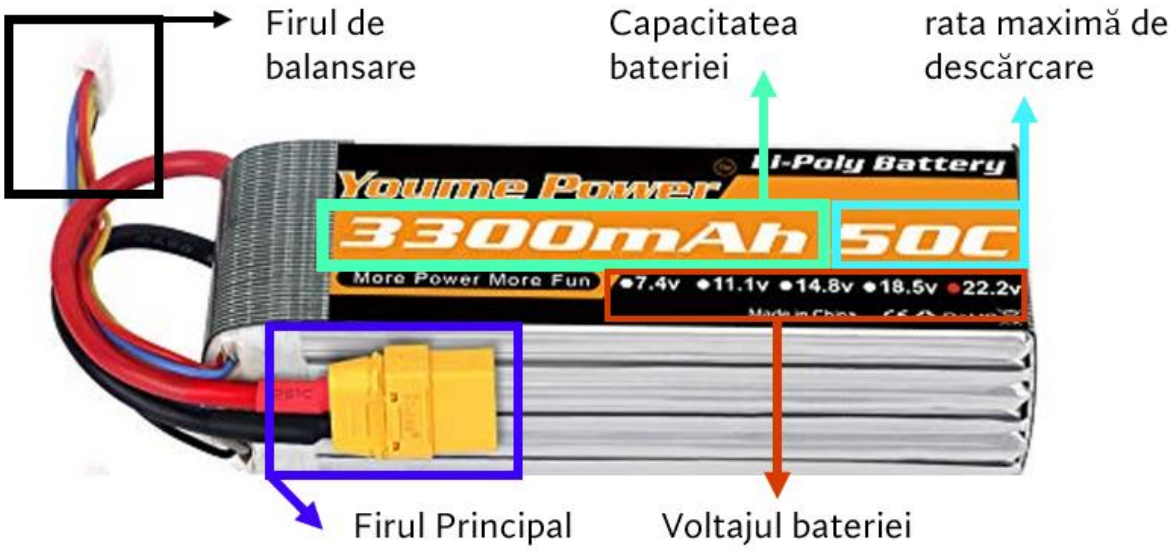

Figura 1.7: Înțelegerea datelor tehnice ale bateriei

Încărcarea bateriilor LiPo necesită un încărcător special proiectat pentru acestea, datorită cerinţelor stricte de încărcare. Încărcătoarele moderne facilitează procesul de încărcare prin setarea parametrilor necesari şi monitorizarea celulelor.

Modurile de încărcare ale încărcătoarelor

Încărcare echilibrată: Încărcătorul monitorizează şi echilibrează tensiunea fiecărei celule.

Încărcare de stocare: Încărcătorul aduce fiecare celulă la tensiunea de stocare (3.80V-3.85V).

Încărcarea fără utilizarea firului de echilibrare poate duce la diferenţe periculoase de tensiune între celule. Este recomandat să se utilizeze întotdeauna încărcarea echilibrată.

Încărcarea la un amper sau mai puţin este recomandată pentru a minimiza stresul asupra bateriei. Încărcarea la viteze mai mari, cum ar fi 3A sau 5A, trebuie realizată numai dacă specificaţiile bateriei permit acest lucru.

Figura 1.8: Încărcarea unei baterii LiPo S6

Încărcarea trebuie să se facă într-un loc fără materiale inflamabile. Este preferabil să se facă lângă o fereastră sau o uşă pentru a arunca rapid bateria în caz de incendiu.

Bateriile LiPo nu trebuie lăsate nesupravegheate în timpul încărcării. Este esenţial să se monitorizeze constant temperatura bateriilor.

Nu se vor folosi sau încărca baterii deteriorate sau umflate şi se va verifica setarea corectă a celulelor şi tipului de baterie pe încărcător înainte de încărcare.

CAPITOLUL 2. ARHITECTURA HARDWARE A SISTEMULUI DE CONTROL CREAT

2.1. Viziunea asupra proiectului

Iniţial, în acest proiect, se dorea crearea unui sistem de localizare a dronei bazat pe mai mulţi senzori, cum ar fi accelerometru, giroscop, magnetometru şi GPS. Din cauza semnalelor slabe de la sateliţi şi a funcţionării optime a acestora doar pe perioade scurte, s-a ajuns la folosirea unui senzor barometric şi unul ultrasonic pentru a compensa datele care trebuiau să fie receptate de GPS. Din cauza acestei lipse, crearea unui sistem de localizare pe toate cele trei axe a devenit foarte greu de implementat, erorile fiind mici pe perioade scurte, dar prin integrare, chiar şi cu ajutorul filtrelor sau programelor, devenind foarte mari.

2.2. Diagrama de conectare

Diagrama de conectare reprezintă schema sistemului de control al dronei. Aceasta ilustrează modul în care diferitele componente electronice sunt interconectate pentru a asigura funcţionarea corectă a dronei. Diagrama de mai jos arată legăturile între controlerul principal precum şi interfeţele de comunicaţie necesare. Scopul este de a clarifica fluxul de semnale şi alimentare în întregul sistem.

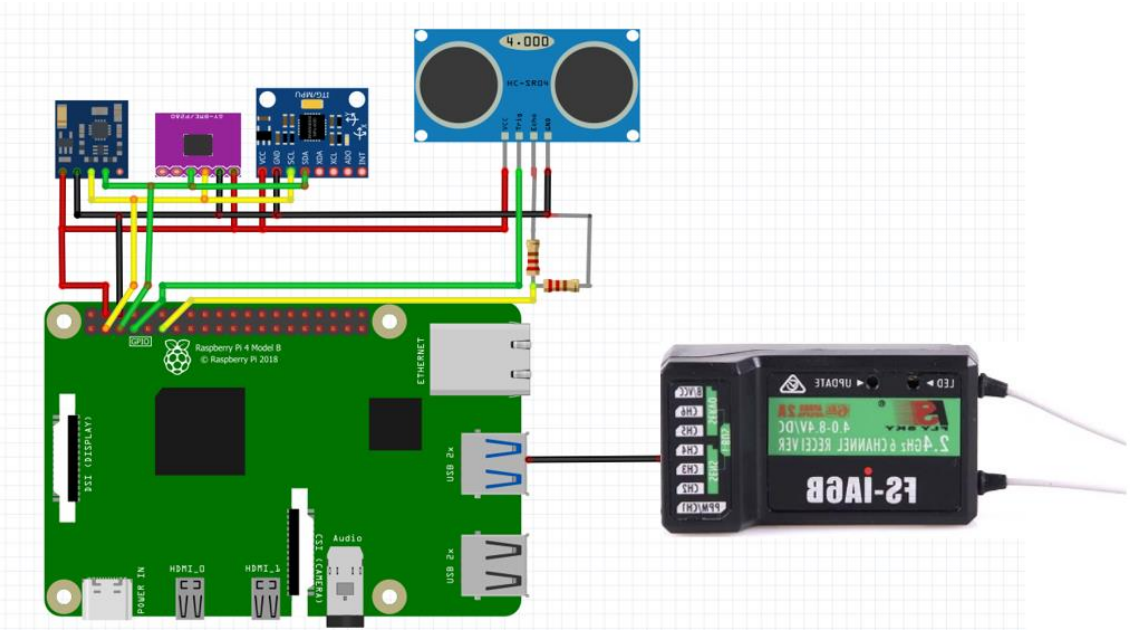

Figura 2.1: Diagrama de conectare a senzorilor realizată in Fritzing

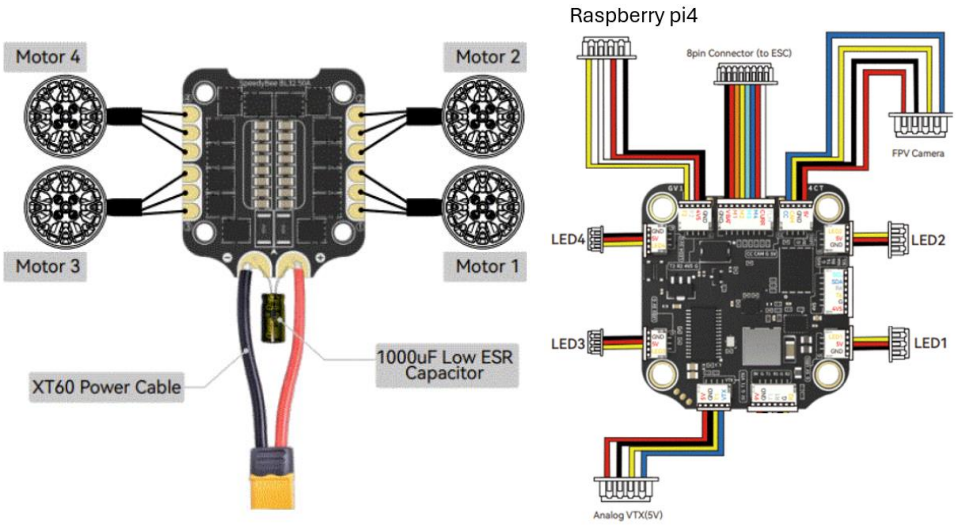

Figura 2.2: Diagrama de conectare a sistemului creat la o unitate de zbor [14]

2.3. Raspberry pi 4B

Raspberry Pi 4B este o placă de dezvoltare cu o putere de procesare impresionantă, fiind ideală pentru proiecte complexe și variate. Echipată cu un procesor quad-core de 1.5 GHz și o capacitate de memorie RAM de până la 8 GB, Raspberry Pi 4B poate gestiona fără dificultate aplicații intensive și multitasking-ul.

Unul dintre punctele forte ale Raspberry Pi 4B este versatilitatea sa în extinderea funcționalităților proiectelor. Dispune de porturi USB 2.0 și 3.0, port Ethernet pentru conectivitate rapidă la internet și două porturi micro-HDMI pentru conectarea a două monitoare simultan. Integrarea Wi-Fi și Bluetooth facilitează conexiunile wireless și comunicarea cu dispozitive externe. De asemenea, portul GPIO permite conectarea unei game variate de senzori și periferice.

Puterea de calcul a microcontrolerului este perfectă pentru aplicații ce necesită filtre și calcule matematice intensive, cum ar fi sistemele de menținere a înălțimii pentru drone. În plus, accesul la o varietate de soluții și biblioteci online este extrem de util pentru rezolvarea diverselor probleme și pentru implementarea tehnologiilor avansate, precum detectarea și recunoașterea obiectelor în timp real.

Raspberry Pi 4B este o opțiune accesibilă comparativ cu platforma Jetson, fiind o alternativă mai economică pentru proiecte care necesită putere de calcul considerabilă. Cu toate acestea, nu face compromisuri în ceea ce privește fiabilitatea și performanța. Este recunoscută pentru stabilitatea sa în funcționare și pentru suportul comunității extinse de dezvoltatori, ceea ce o face o alegere populară în rândul entuziaștilor și profesioniștilor în domeniul IT și al ingineriei.

2.4. Senzorul HC-SR04

Pentru calcularea distanței se utilizează senzorul HC-SR04 in cadrul vehiculelor aeronautice fără pilot în scopul obținerii unei distanțe optime a vehiculului față de sol.

Acesta folosește tehnica sonarului, similară cu cea utilizată de delfini și lilieci pentru orientare în spațiu permițând determinarea intervalului până la un obiect fără a necesita contact fizic, oferind o acuratețe ridicată și rezultate precise în anumite medii de utilizare. Senzorul poate măsura distanțe între 2 cm și 400 cm, cu o eroare de până la 3 cm.

Eroarea de măsurare poate fi influențată în principal de materialul din care este format obiectul, dar si de temperatura mediului în care este folosit, însă nu este afectată de lumina soarelui sau de culoarea obiectului detectat. Modulul HC-SR04 include unități de control, un transmițător și un receptor.

Figura 2.3: Senzorul HC-SR04 [15]

2.5. Senzorul MPU5060

Unitatea de măsurare inerţială este unul dintre cei mai utilizaţi senzori în navigaţie. MPU6050 IMU are atât accelerometru cu 3 axe, cât şi giroscop cu 3 axe integrate pe un singur cip.

Senzorul măsoară acceleraţia prin măsurarea schimbării capacitivităţii. Structura microscopică poate fi observata in figura 2.3. Are o masă ataşată la un arc care poate să se mişte doar într-o direcţie, între plăcile externe fixe. Astfel, atunci când este aplicată o acceleraţie în direcţia respectivă, masa se va mişca, iar capacitivitatea între plăci şi masa se va schimba. Această schimbare în capacitivitatea este măsurată, procesată şi corespunde unei valori specifice de acceleraţie.[15]

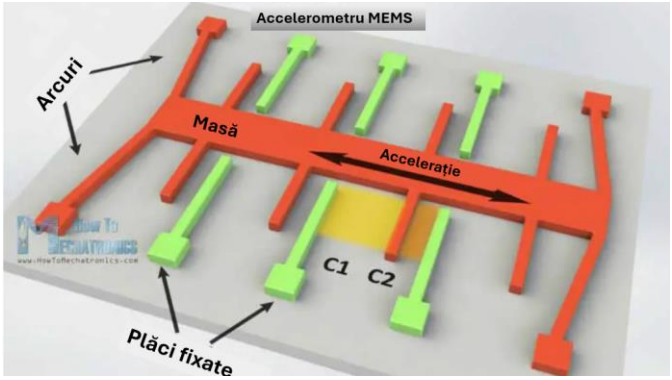

Figura 2.4: Modul de funcţionare a accelerometrului [16]

Giroscopul măsoară viteza unghiulară folosind efectul Coriolis. Atunci când o masă se mişcă într-o anumită direcţie cu o anumită viteză şi când este aplicată o rată unghiulară externă, aşa cum este arătat cu săgeata verde in figura 2.5., va apărea o forţă,

așa cum este arătat cu săgeata roșie și albastră, care va cauza o deplasare perpendiculară a masei. Similar cu accelerometrul, această deplasare va provoca o schimbare în capacitivitate care va fi măsurată, procesată și va corespunde unei anumite rate unghiulare.

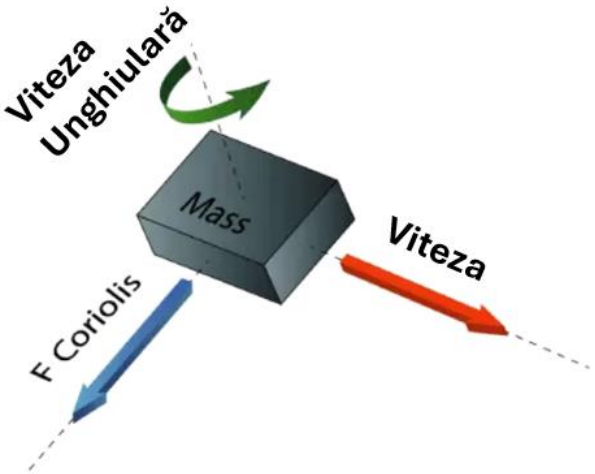

Figura 2.5: Modul de funcționare a giroscopului [16]

2.6. Senzorul HMC5883L

Senzorul măsoară câmpul magnetic al Pământului folosind efectul Hall împreună cu 90% dintre senzorii de pe piață. Dacă există o placă conductoare, cum este afișată în figura 2.6., și curge curent prin ea, electronii vor circula în linie dreaptă de la o parte la alta a plăcii. Însă în prezența unui câmp magnetic în apropierea plăcii, acesta va perturba fluxul drept și electronii se vor deplasa către o parte a plăcii, iar polii pozitivi spre cealaltă parte a plăcii. Înseamnând că, dacă există un voltmetru între cele două părți, se poate măsura o tensiune care depinde de intensitatea și direcția câmpului magnetic.[16]

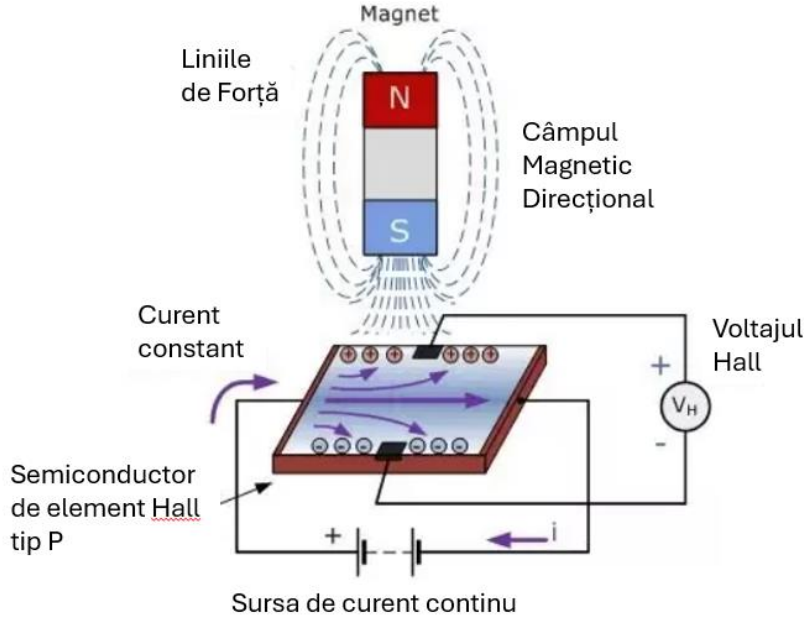

Figura 2.6: Modul de funcționare a magnetometrului [17]

2.7. Senzorul WH-611

În mare parte a istoriei noastre, barometrele depindeau de comportamentul mercurului sau a altor lichide în răspuns la schimbările presiunii atmosferice. Barometrul aneroid, numele său referindu-se la absenţa lichidului, a fost inventat în 1844, folosind deformarea unui metal în loc.

În acesta, o celulă metalică parţial evacuată este supusă presiunii atmosferice. Pe măsură ce presiunea variază, celula se contractă sau se dilată. Această mişcare este tradusă şi amplificată, printr-un arc de contracţie, un sistem de pârghii şi un indicator, pentru a înregistra o citire pe cadranul barometrului.

Senzorii moderni de presiune barometrică sunt, într-un sens, barometre aneroid, deoarece metoda lor de funcţionare nu implică lichid. În construcţie şi aspect, însă, aceştia sunt foarte diferiţi faţă de predecesorii lor - adesea folosind tehnologia modernă cu sisteme microelectromecanice (MEMS).

În comun cu barometrele aneroid originale, aceştia detectează presiunea atmosferică prin efectul său asupra unei structuri flexibile - în acest caz o membrană sau o diafragmă. Gradul de deformare al membranei este proporţional cu presiunea şi este tradus într-un semnal electric.

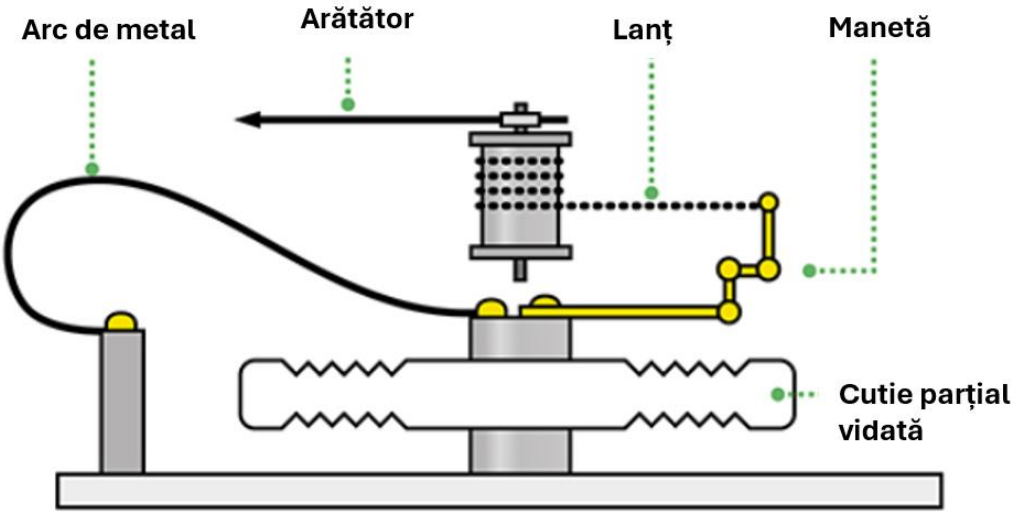

Figura 2.7: Un barometru aneroid clasic [18]

2.8. Transmiţătorul si receptorul FlySky

FS-i6 este un sistem digital de control radio care operează pe banda globală ISM de frecvenţă 2.4GHz, fiind potrivit pentru utilizare globală. Transmiţătorul este construit pe tehnologia AFHDS 2A (Automatic Frequency Hopping Digital System Second Generation), ceea ce asigură o comunicare sigură şi reduce consumul de energie al transmiţătorului. Aceste caracteristici, împreună cu preţul accesibil, l-au făcut alegerea preferată a entuziaştilor de drone din întreaga lume. Transmiţătorul şi receptorul FLYSKY FS-i6 vine cu o gamă variată de opţiuni care permit personalizarea completă a setărilor de zbor pentru diferite tipuri de drone, avioane, elicoptere şi drone de curse.

Pentru cei aflaţi în faza incipientă, numeroasele butoane, comutatoare, joystick-uri şi opţiuni de personalizare pot părea copleşitoare.

Principalul avantaj a acestui transmiţător este lipsa necesităţii asistenţei unui calculator sau laptop pentru a-l configura. Are un ecran LCD şi poate fi setat uşor folosind butoanele disponibile pe el.

Transmiţătorul poate acoperi o distanţă de până la 1500 de metri, însă performanţa sa poate fi influenţată de interferenţele magnetice. În locuri cu interferenţe magnetice mai mari, raza de acţiune a transmiţătorului va fi mai mică, în timp ce în zone cu interferenţe mai reduse, acesta poate opera la distanţe mai mari.[18]

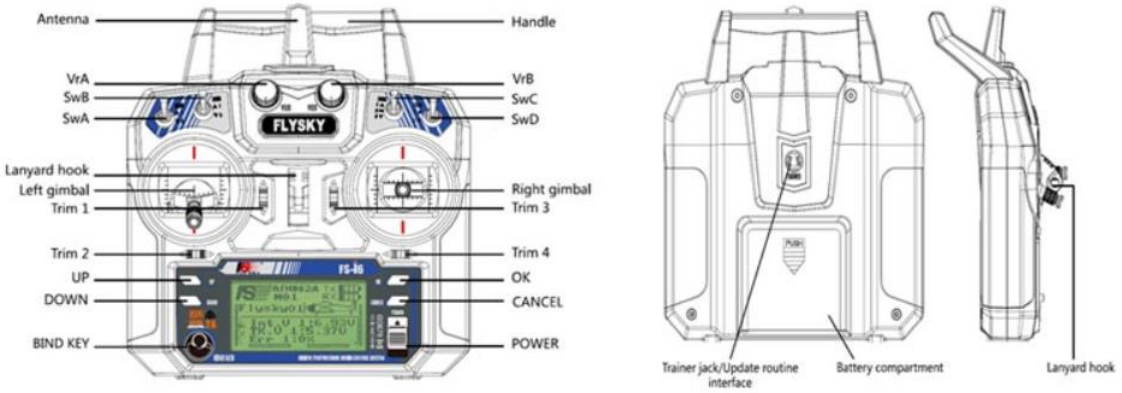

Figura 2.8: Transmiţătorul FlySky [19]

FS-i6B suportă mai multe protocoale de comunicaţie, inclusiv PWM şi IBUS, oferind utilizatorilor opţiuni flexibile în funcţie de nevoile lor. IBUS este un ideal pentru aplicaţii precum controlul dronelor. În schimb, PWM (Modulare pe Durată de Impuls) este un protocol analogic tradiţional folosit pentru controlul servo-motoarelor şi altor echipamente în aplicaţiile de radiocontrol.

Figura 2.9: Receptorul FS-IA6B FlySky [20]

CAPITOLUL 3. TESTAREA SISTEMELOR VIDEO ŞI DE TRANSMISIE IN CONDIŢII REALE

În era digitală modernă, sistemele video şi de transmisie joacă un rol crucial într-o varietate de aplicaţii, de la supravegherea de securitate la transmiterea de conţinut multimedia în timp real. Calitatea şi fiabilitatea acestor sisteme sunt esenţiale pentru asigurarea unei experienţe satisfăcătoare şi pentru prevenirea pierderilor de date critice. Acest capitol se concentrează pe evaluarea performanţelor sistemelor video şi de transmisie în condiţii reale, oferind o analiză detaliată a calităţii şi a erorilor de transmisie pe diferite distanţe.

3.1. Metodologia de testare

Metodologia noastră de testare a implicat aplicarea măsurătorilor la distanţe variate, beneficiind de condiţii meteorologice favorabile pentru a minimiza variabilele externe. Testele au fost concepute pentru a evalua eficienţa sistemului în diverse scenarii, excluzând interferenţele generate de structuri arhitecturale sau alte obiecte voluminoase care ar putea afecta semnalul.

Figura 3.1. Configuraţia de măsurare

3.2. Specificaţii Tehnice şi Echipamente Utilizate

Pentru evaluarea performanţei, am utilizat echipamente video avansate, inclusiv camere şi transmiţătoare specializate pentru sistemele FPV. Aceste echipamente au fost selectate pentru capacitatea lor de a transmite a semnale la distanţe mari cu o calitate ridicată, adaptându-se nevoilor specifice ale pilotajului dronelor.

3.3. Criterii de Evaluare

Criteriile noastre de evaluare au inclus calitatea imaginii transmise, stabilitatea și consistența semnalului în timp real, precum și rezistența la interferențe. S-a monitorizat și înregistrat performanța sistemului în diverse condiții de lumină, vreme, interferențe și în funcție de unghiul de transmisie.

În cadrul primului test, au fost colectate date ale sistemului video în condiții favorabile, fără vânt și fără prezența unor obiecte care să interfereze cu semnalul. Aceste condiții au permis obținerea următorului tabel de rezultate.

Tabelul 3.1. Măsurătorile video in condiții optime

Nr. test	Oră	Locație			Distanță	Eroare
		Latitudine	Longitudine	Altitudine		
1	10:58	46.041651	27.395793	68	0m	Calitatea video a fost foarte bună
2	11:02	46.041784	27.397882	72	162.4m	
3	11:05	46.041762	27.398515	75	211.1m	
4	11:08	46.041792	27.398876	77	239.2m	
5	11:12	46.041716	27.400674	103	380m	Calitate slabă

Pentru testul de radiocomandă, am efectuat măsurători în aceleași condiții favorabile, asigurându-ne că nu există interferențe semnificative din alte surse radio sau electromagnetice. Aceste condiții au permis evaluarea precisă a performanțelor radiocomenzii într-un mediu controlat, rezultând următorul tabel de rezultate.

Tabelul 3.2. Măsurătorile receptorului radio

Nr. test	Oră	Locație			Distanță	Eroare
		Latitudine	Longitudine	Altitudine		
1	11:24	46.041744	27.396756	69	75.3m	0%
2	11:27	46.041806	27.398555	67	230.1m	7%
3	11:28	46.041789	27.399084	81	270.9m	6%
4	11:29	46.041717	27.399904	90	334m	1%
5	11:31	46.041716	27.400674	103	393.6m	11%

În analiza datelor, se observă o tendință de scădere a erorii în funcție de altitudine în primele patru măsurători, chiar dacă distanța crește. Cu cât altitudinea crește, eroarea pare să fie mai mică, cu excepția ultimei măsurători. Acest fenomen sugerează că altitudinea înaltă facilitează captarea unui semnal mai bun și mai stabil. Totuși, la ultima măsurătoare, eroarea înregistrează o creștere semnificativă din cauza perturbațiilor create de copaci în apropiere.

Figura 3.2. Măsurarea vizuala a erorilor

3.4. Impactul Distanţei şi Factorii Ambientali

Rezultatele au evidenţiat impactul semnificativ al distanţei asupra calităţii video şi a stabilităţii semnalului. În condiţii ideale, sistemul a demonstrat o transmisie fiabilă şi clară a imaginii la distanţe mari, însă în condiţii mai puţin favorabile, cum ar fi vântul puternic sau condiţiile meteorologice inadecvate, performanţa sistemului a fost afectată în mod semnificativ.

Tabelul 3.3. Măsurătorile video in condiţii neprielnice

Nr. test	Oră	Locaţie			Distanţă	Eroare
		Latitudine	Longitudine	Altitudine		
1	18:24	46.041652	27.395590	69	0m	Calitatea video proastă, cu multe interferenţe
2	17:57	46.041594	27.395942	70	28m	
3	18:14	46.041604	27.396443	70	66.2m	
4	18:19	46.041627	27.398555	75	229.5m	

În cadrul acestui set de măsurători, datele au fost colectate în condiţii mai puţin favorabile, cu vânt puternic, ploaie şi întuneric, care au afectat semnificativ calitatea video. La prima măsurătoare, calitatea video a fost foarte slabă, cu multe interferenţe şi fără distanţă de transmisie efectivă. Pe măsură ce testele au continuat, s-a observat o îmbunătăţire a distanţei de transmisie, dar condiţiile meteo nefavorabile şi întunericul au continuat să degradeze calitatea imaginii. Deşi altitudinea a crescut uşor în timpul măsurătorilor, impactul condiţiilor meteo nefavorabile a rămas evident asupra performanţei sistemului.

Figura 3.3. Perturbațiile provocate de vreme si obstacole

Figura 3.4. Comportamentul sistemului video in relație cu interferențele si pierderea de semnal

Fluxul video primit de monitor este observat în secţiuni la distanţe diferite. Imaginea din dreapta sus arată un flux video clar şi stabil, cu detalii vizibile ale străzii, copacilor şi structurilor din jur. Cu toate acestea, secţiunea de jos a imaginii prezintă o degradare semnificativă a semnalului, cu pixelare şi pierdere a clarităţii vizuale, indicând interferenţe sau limitări în domeniul de transmisie.

Diferența de calitate dintre secțiunile din dreapta sus și de jos demonstrează modul în care distanța și obstacolele de mediu afectează fluxul video al dronei. Pe măsură ce drona se îndepărtează de receptor, calitatea video se înrăutățește, afișând mai multe artefacte și întreruperi ale semnalului. Mediile urbane, cu numeroasele lor clădiri și sursele potențiale de interferență electromagnetică, agravează această problemă.

Figura 3.4. ilustrează dificultățile de a menține o transmisie video stabilă în zonele urbane. Scăderea calității pe măsură ce distanța crește evidențiază importanța sistemelor de transmisie puternice și nevoia potențială de amplificatoare de semnal sau repetoare pentru a garanta performanțe consistente în aceste setări.

3.5. Concluzii și Recomandări

În concluzie, acest studiu nu doar că a evaluat performanțele curente ale sistemelor testate, dar și a identificat punctele forte și limitările acestora în diverse scenarii practice. Înțelegerea detaliată a acestor aspecte este esențială pentru optimizarea și îmbunătățirea sistemelor video și de transmisie în viitor. De asemenea, acest capitol servește ca un ghid valoros pentru dezvoltarea și implementarea de soluții mai eficiente în domeniul transmisiilor video și pentru pregătirea piloților în contextul utilizării sistemelor FPV în condiții variate.

Prin urmare, integrarea acestor cunoștințe în practica de pilotaj poate contribui semnificativ la îmbunătățirea siguranței și performanței în utilizarea dronelor în diverse aplicații, de la filmări aeriene profesionale la inspecții industriale și operațiuni de salvare.

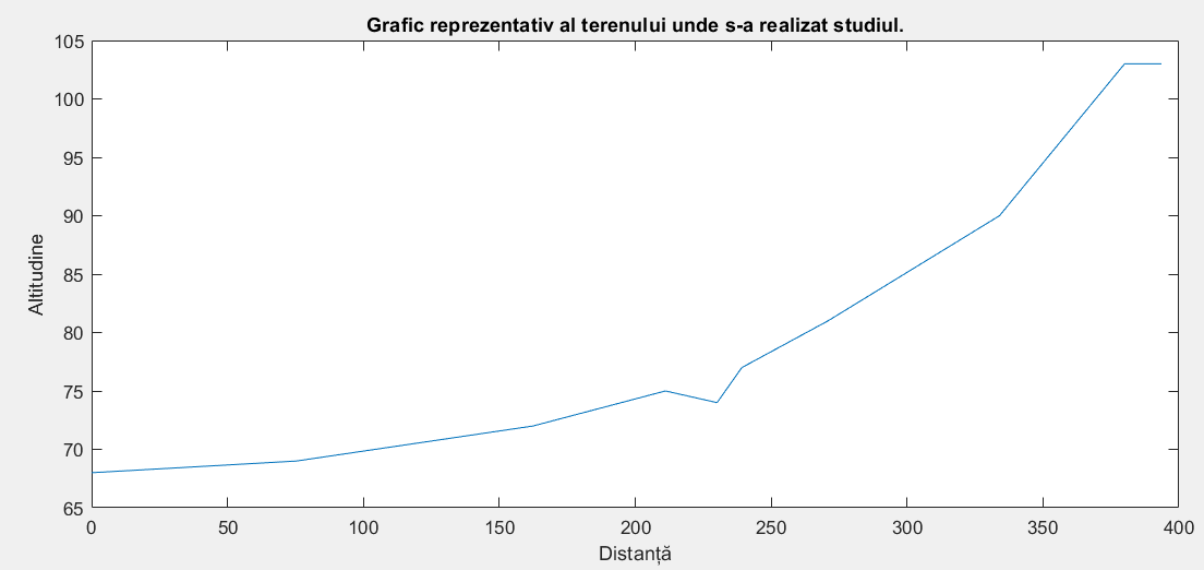

Figura 3.5. Graficul reprezentativ al terenului unde s-a realizat studiul

CAPITOLUL 4. PROIECTAREA SI IMPLEMENTAREA SOFTWARE

4.1. Sistemul de operare instalat pe Raspberry pi 4

Debian Bullseye este o opţiune excelentă pentru utilizatorii acestui mini-computer, oferind stabilitate şi performanţă ridicată. Debian Bullseye, versiunea 11 a sistemului de operare Debian, rulează eficient pe hardware-ul Raspberry Pi 4, care dispune de un procesor quad-core ARM Cortex-A72 şi opţiuni de memorie RAM de 2GB, 4GB şi 8GB. Sistemul include suport complet pentru USB 3.0, dual HDMI pentru ieşire 4K şi Ethernet Gigabit.

Pentru a rula Debian Bullseye pe un Raspberry Pi 4, se recomandă utilizarea unui card microSD de înaltă calitate, de cel puţin 16GB, preferabil UHS-I sau superior, pentru viteze de citire şi scriere mai mari. Carduri de 32GB sau 64GB sunt ideale pentru a asigura suficient spaţiu de stocare şi o experienţă de utilizare eficientă.

Debian Bullseye suportă complet toate componentele hardware ale Raspberry Pi 4, asigurând o funcţionare optimă şi o integrare fără probleme. Include o gamă largă de pachete software actualizate, cum ar fi LibreOffice 7.0 şi Python 3.9, facilitând dezvoltarea de aplicaţii, servere mici şi proiecte multimedia.

În concluzie, Debian Bullseye pe Raspberry Pi 4 oferă o platformă robustă şi versatilă, adecvată pentru o varietate largă de utilizări, de la educaţie şi proiecte DIY la servere mici şi media center.[21]

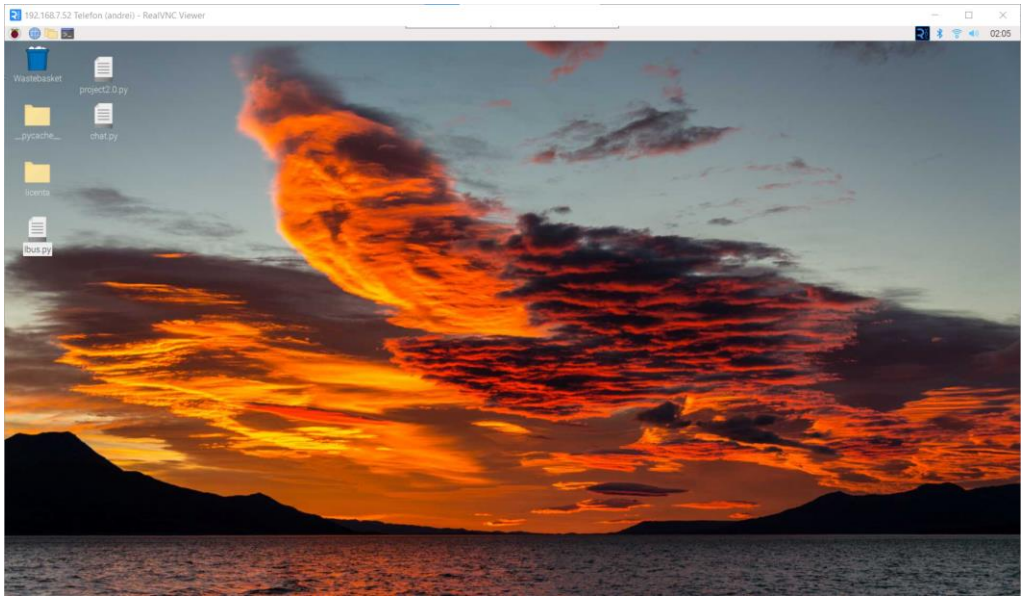

Figura 4.1. Debian Bullseye

4.2. Python 3.12.

Python este un limbaj de programare de nivel înalt care este interpretat, orientat pe obiecte şi are o semantică dinamică. Structurile de date încorporate, tastarea dinamică şi legarea dinamică îl fac ideal pentru dezvoltarea rapidă a aplicaţiilor şi ca limbaj de scripting pentru conectarea componentelor. Sintaxa Python este uşor de învăţat şi accentuează lizibilitatea, reducând costurile de întreţinere a programului. Acceptă module şi pachete pentru modularitatea programului şi reutilizarea codului.

Interpretul Python şi biblioteca standard sunt disponibile gratuit pe toate platformele majore în formă sursă sau binară.

Cea mai recentă versiune de Python, 3.12, aduce o multitudine de funcţii noi şi îmbunătăţiri care îmbunătăţesc gradul de utilizare, performanţa şi experienţa generală a dezvoltatorului limbajului. Cu îmbunătăţiri în sintaxa parametrilor de tip, mesaje de eroare şi diverse module, Python 3.12 îşi consolidează reputaţia ca limbaj de programare versatil şi robust.[22]

4.3. Măsurarea unghiului de rotaţie

4.3.1. Principiul de funcţionare al giroscopului

Pentru stabilizarea quadcopterului, este nevoie de măsurători ale orientării tridimensionale. Pentru modul de control, este suficient să se cunoască vitezele de rotaţie pentru ruliu (roll), tangaj (pitch) şi giraţie (yaw). Un senzor care poate înregistra aceste viteze de rotaţie se numeşte giroscop.

Giroscopul folosit este inclus în MPU-6050, un senzor de orientare de cost redus. Deşi nu este foarte precis, acurateţea sa este suficientă pentru a obţine rezultate bune în echilibrarea dronei. Acesta va măsura viteza de ruliu, tangaj şi giraţie, adică nu va măsura unghiuri absolute în grade (°), ci ratele unghiulare în grade pe secundă (°/s). De exemplu, o viteză unghiulară de 60°/s înseamnă că se roteşte cu 60° în fiecare secundă şi va efectua o rotaţie completă de 360° în 6 secunde (360° / 60°/s). Pentru a menţine vehiculul aeronautic fără pilot în echilibru, viteza unghiulară trebuie să fie 0°/s. [22]

- O rotaţie de ruliu înseamnă rotirea în sensul acelor de ceasornic în jurul axei X a giroscopului.
- O rotaţie de tangaj reprezintă rotirea în sensul acelor de ceasornic în jurul axei Y a giroscopului.
- O rotaţie de giraţie semnifică rotirea în sens invers acelor de ceasornic în jurul axei Z a giroscopului.

Trebuie să se observe că axele X şi Y şi direcţiile lor de rotaţie sunt scrise fizic pe senzorul MPU-6050. Atunci când se construieşte drona şi se fixează senzorul, trebuie să se asigure întotdeauna că axele scrise pe senzor sunt aliniate cu axele de ruliu, tangaj şi giraţie ale aeronavei.

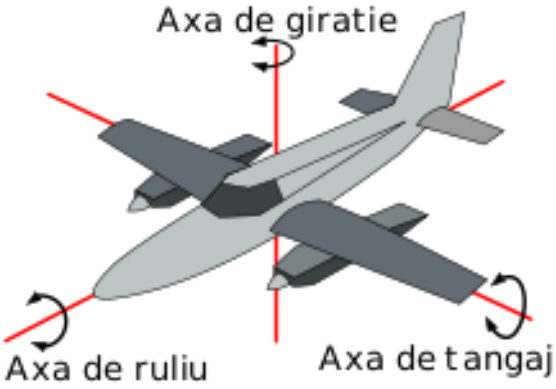

Figura 4.2. Axele de rotaţie a unei aeronave [23]

4.3.2. *Protocolul I2C*

Inter-Integrated Circuit (I2C) este un protocol de interfață de magistrală, fiind încorporat în dispozitive pentru comunicații seriale dezvoltat inițial de corporația Phillips pentru utilizarea în produse de consum. Este bidirecțional, ușor de implementat în orice proces IC (NMOS, CMOS, bipolar) și permite o comunicare simplă între circuitele integrate. Conexiunile sunt minimizate folosind o linie de date serială (SDA), o linie de ceas serială (SCL) și o masă comună pentru a transporta toate comunicațiile. I2C a câștigat o acceptare largă și a servit chiar ca prototip pentru System Management Bus (SMBus), care este un subset al acestuia.

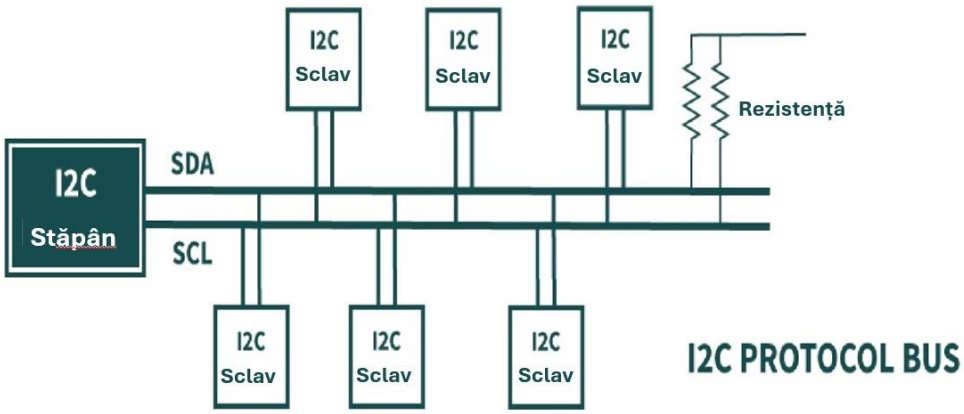

Figura 4.3. Modul de funcționare Master Slave al protocolului I2C [24]

Pentru a activa I2C pe un Raspberry Pi, utilizatorul trebuie să deschidă terminalul și să ruleze comanda „*sudo raspi-config*”.

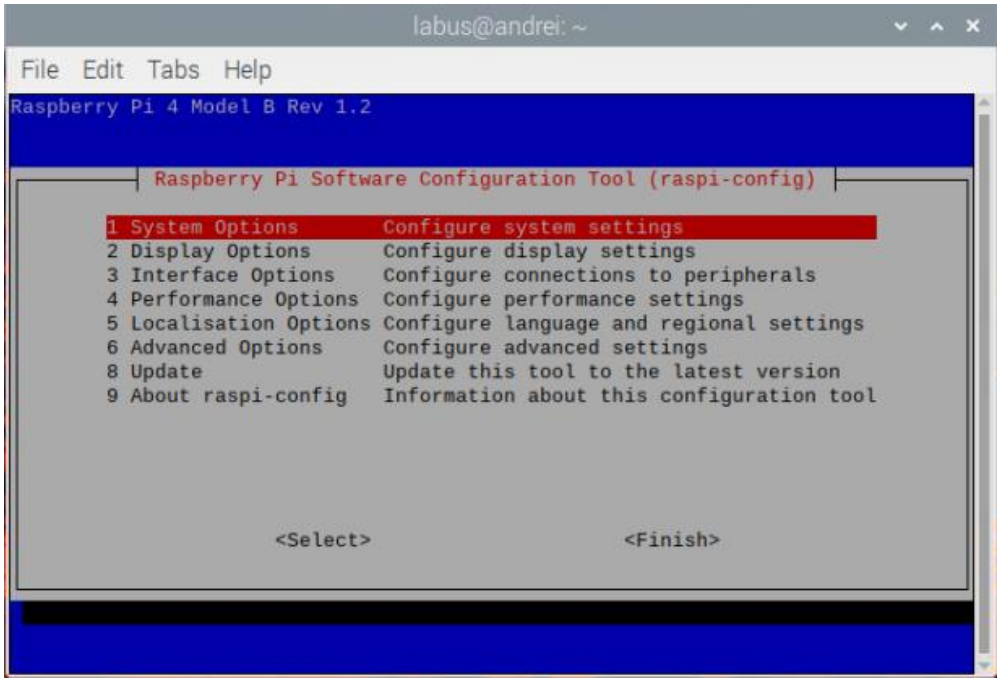

Figura 4.4. Ecranul de configurare al setărilor

Apoi, trebuie să navigheze la „*Interfacing Options*”, să selecteze „*I2C*” şi să aleagă „*Yes*” pentru a activa interfaţa.

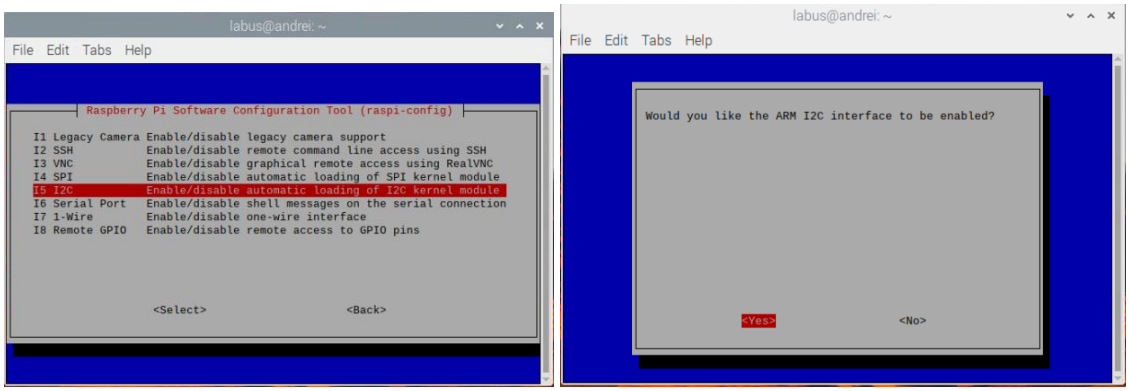

Figura 4.5. Paşii de activare a protocolului I2C

După confirmare şi ieşirea din configurare, plăcuţa trebuie repornită. După repornire, utilizatorul poate instala utilitarele I2C folosind comanda „*sudo apt-get install -y i2c-tools*”. Pentru a verifica dacă I2C este activ, se rulează comanda „*lsmod | grep i2c*”, iar modulele I2C încărcate vor fi afişate. [25]

Comanda „*i2cdetect -y 1*” este utilizată pe sistemele Linux pentru a scana şi detecta dispozitivele conectate pe magistrala I2C. Aceasta face parte din pachetul i2c-tools, argumentul -y opreşte solicitarea de confirmare interactivă permiţând comenzii să ruleze fără intervenţia utilizatorului, iar numărul 1 specifică magistrala I2C pe care se efectuează scanarea. Comanda returnează un tabel cu adresele dispozitivelor I2C detectate indicându-le prin adresele lor hexazecimale aceasta este utilă pentru diagnosticarea şi configurarea dispozitivelor I2C conectate.

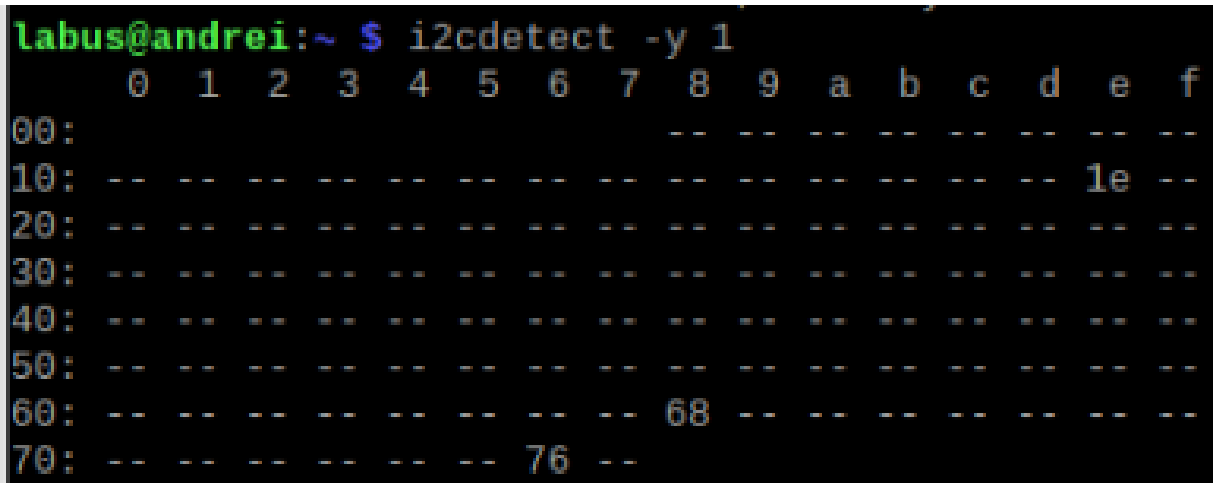

Figura 4.6. Lista senzorilor conectaţi prin protocolul I2C

4.3.3. Primul pas pentru măsurarea unghiurilor

Acest program foloseşte direct adresele de memorie ale registrelor MPU6050 pentru a citi şi scrie date, în locul unei biblioteci de nivel înalt. Această abordare permite un control precis asupra senzorului şi optimizează timpul de execuţie, reducând latenţa. Utilizarea directă a adreselor de memorie sporeşte eficienţa si rapiditatea codului, fiind esenţiale pentru aplicaţiile care necesită actualizări frecvente ale datelor.

Astfel, în program apar adresele mai multor senzori sub forma:

```
# Regiştrii MPU6050
PWR_MGMT_1 = 0x6B      # Registru pentru gestionarea puterii
SMPLRT_DIV = 0x19      # Registru pentru divizarea ratei de eşantionare
CONFIG = 0x1A          # Registru pentru configuraţia generală
GYRO_CONFIG = 0x1B     # Registru pentru configuraţia giroscopului
INT_ENABLE = 0x38      # Registru pentru activarea întreruperilor
ACCEL_XOUT_H = 0x3B    #  Citirea valorii axei X a accelerometrului
ACCEL_YOUT_H = 0x3D    # Citirea valorii axei Y a accelerometrului
ACCEL_ZOUT_H = 0x3F    # Citirea valorii axei Z a accelerometrului
GYRO_XOUT_H = 0x43
GYRO_YOUT_H = 0x45
GYRO_ZOUT_H = 0x47
#Adresa de memorie a senzorului
Device_Address = 0x68
```

Funcţia „*read_raw_data(addr)*” citeşte date brute de la senzorul MPU6050 prin magistrala I2C şi este esenţială pentru obţinerea valorilor necompletate ale accelerometrului şi giroscopului.

```
def read_raw_data(addr):
    high = bus.read_byte_data(Device_Address, addr)
    low = bus.read_byte_data(Device_Address, addr + 1)
    value = ((high << 8) | low)
    if value > 32768:
        value -= 65536
    return value
```

Această funcţie utilizează „*bus.read_byte_data*” pentru a citi doi biţi consecutivi de la adresa specificată addr. Apoi, valorile sunt combinate într-un singur întreg de 16 biţi (value). Dacă valoarea este negativă (adică depăşeşte 32768), este ajustată pentru a respecta formatul de complement la 2.

În acest mod se pot citi datele neprelucrate ale accelerometrului şi giroscopului.

Pentru a interpreta corect datele de la accelerometru şi giroscop, a fost necesară prelucrarea măsurătorilor iniţiale ale senzorilor prin împărţirea acestora la factorul de sensibilitate corespunzător fiecărei componente. Astfel, datele de la accelerometru au fost normalizate prin împărţirea lor cu 16384, în timp ce datele de la giroscop au fost ajustate prin împărţirea lor cu 131.

```
Ax = acc_x/16384.0  #Intervalul complet de +/- 250 de grade
Ay = acc_y/16384.0
Az = acc_z/16384.0
Gx = gyro_x/131.0
Gy = gyro_y/131.0
Gz = gyro_z/131.0
```

Pentru a progresa de la simpla măsurare a rotaţiilor sau acelerării pe secundă, este necesar să se măsoare şi să se integreze valorile în timpul dintre cicluri pentru a calcula unghiurile şi distanţele corespunzătoare.

```
Time = dateDateTimeNow - LastTime
  GPitch = GPitch + Time * Gy * 8.01
  GRoll  = GRoll  + Time * Gx * 8.01
```

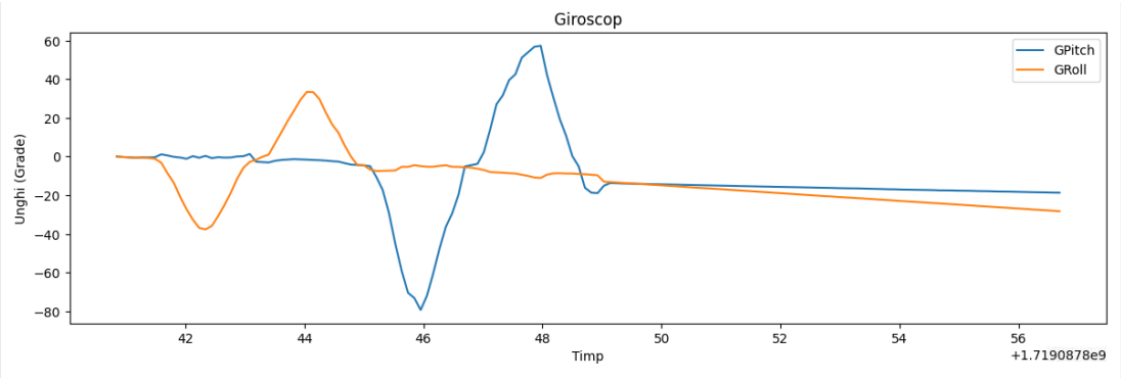

Figura 4.7. Citirea unghiurilor cu ajutorul giroscopului

Odată ce s-a reuşit calcularea unghiului sistemului, s-a decis să se adopte un filtru complementar pentru a îmbunătăţi precizia sistemului. Acest filtru complementar este conceput pentru a corecta erorile cumulative care apar din integrarea repetată a micilor erori de măsurare ale senzorului giroscopic. El combină avantajele filtrului trece-jos, care reduce zgomotul semnalului iniţial, cu capacitatea de a ajusta şi de a corecta erorile de lungă durată care pot afecta măsurătorile în timp. Astfel, utilizarea unui filtru complementar contribuie la asigurarea unei monitorizări mai precise şi mai stabile a acceleraţiilor şi rotaţiilor sistemului.

```
def Dist(a,b):
    return math.sqrt((a*a)+(b*b));
def Get_y_rotation(x,y,z):
    radians = math.atan2(y, Dist(x,z))
    return math.degrees(radians);
def Get_x_rotation(x,y,z):
    radians = math.atan2(x, Dist(y,z))
    return -math.degrees(radians);
APitch = Get_x_rotation( Ax, Ay, Az )
ARoll = Get_y_rotation( Ax, Ay, Az )

Pitch  = K * GPitch + K1 * APitch
Roll = K * GRoll  + K1 * ARoll
```

Unde K este o constată cu valoarea de K = 0.98, iar K1 = 0.02, de unde provine numele de filtru complementar.

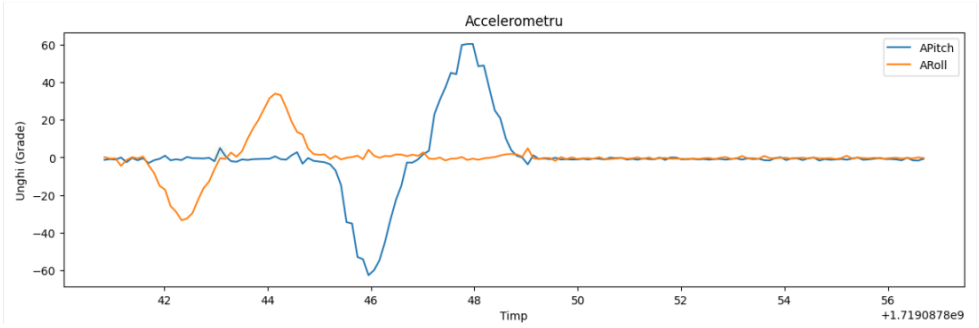

Figura 4.8. Citirea unghiurilor cu ajutorul accelerometrului

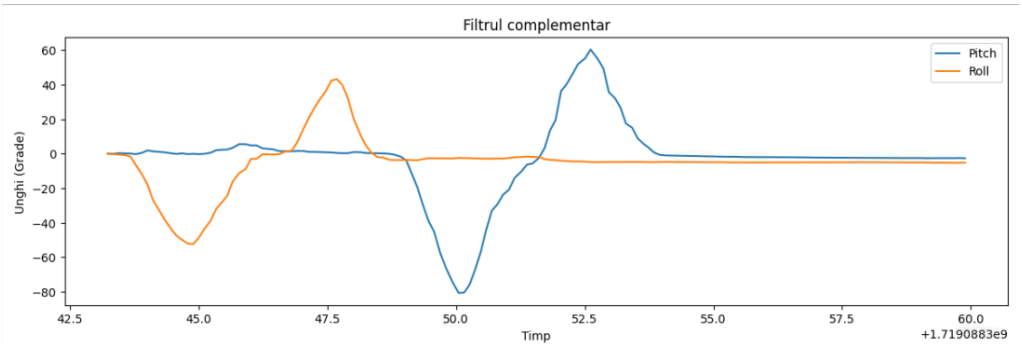

Figura 4.9. Filtrul complementar

Chiar dacă s-a observat o reducere semnificativă a zgomotului semnalelor, s-a observat o eroare considerabilă. Pentru a îmbunătăți precizia măsurătorilor, s-a decis implementarea unui filtru trece-jos. Acest filtru a fost folosit pentru a elimina zgomotul din semnalul senzorului MPU6050, asigurând o măsurare stabilă și exactă a accelerațiilor și rotațiilor, reducând fluctuațiile și interferențele din datele obținute.

LOWPASSFILTER= 26
bus.write_byte_data(Device_Address, LOWPASSFILTER, 5)

Giroscopul este un senzor foarte sensibil şi citirile sale sunt afectate drastic de vibraţiile cauzate de motoarele fără perii. Rata de eşantionare a giroscopului este egală cu 8 kHz, ceea ce corespunde unei măsurători la fiecare 1/8000 = 0.000125 secunde.

Vibraţiile de înaltă frecvenţă de la motoare provocă acceleraţii mici, dar foarte rapide ale cadrului quadcopterului, care sunt înregistrate de giroscop. Cu cât motoarele se rotesc mai repede, cu atât vibraţiile devin mai mari. În toate cazurile măsurate, quadcopterul a rămas staţionar pe sol, astfel încât rata de rotaţie ar trebui să fie egală cu 0°/s. Din figură observaţi că odată ce motoarele sunt pornite, valorile nefiltrate ale giroscopului încep să fluctueze foarte mult. Este clar că devine imposibil de stabilizat quadcopterul cu valori de măsurare care variază în timp ce el însuşi rămâne staţionar şi rata reală de rotaţie este egală cu 0 °/s.

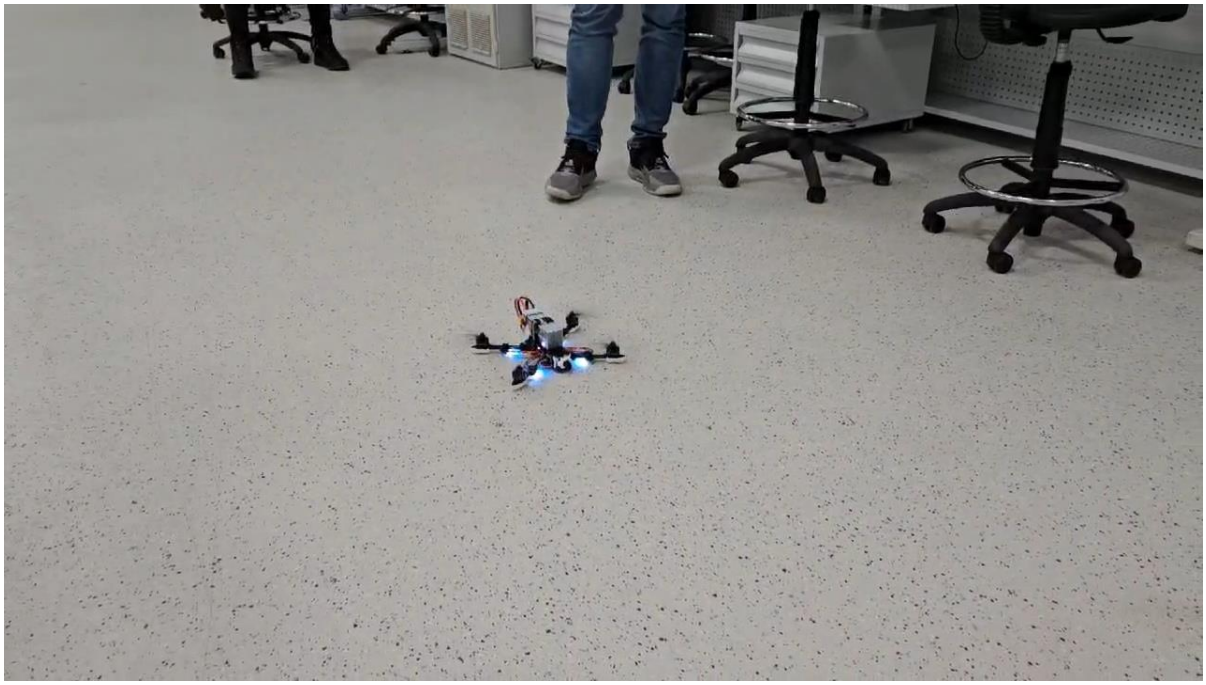

Figura 4.10. Drona in poziţie staţionară

Pentru a rezolva această problemă, s-a utilizat filtrul trece-jos cu o frecvenţă de tăiere de 10 Hz. Filtrul atenuează măsurătorile senzorului cu o frecvenţă de 10 Hz şi mai mare. Acest lucru înseamnă că variaţiile senzorului care au loc mai repede de 1/10=0.1 secunde vor avea doar un impact limitat asupra valorilor finale de măsurare. Valoarea de 10 Hz este aleasă prin încercări şi erori în timpul testării quadcopterului, iar frecvenţele de vibraţie ale motorului se schimbă cu fiecare motor fără perii şi amortizarea vibraţiei depinde de întregul cadru. Linia albastră din figura 4.11. arată valorile filtrate, acestea nu sunt afectate de vibraţiile cauzate de motoarele fără perii şi sunt astfel potrivite pentru controlerul de zbor.

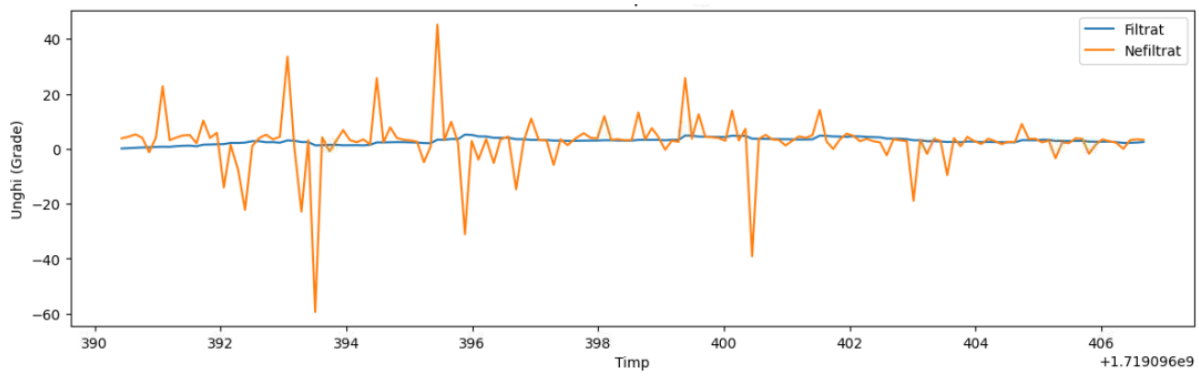

Figura 4.11. Citirea datelor cu ajutorul filtrului trece-jos

Deoarece existau erori chiar şi în cazul utilizării filtrului complementar, s-a decis dezvoltarea unui program suplimentar pentru calcularea offset-ului citirilor senzorului. Programul de calculare a offset-ului funcţionează prin efectuarea următoarelor etape:

Citirea valorilor de la accelerometru şi giroscop de 100 de ori pentru fiecare axă (X, Y şi Z). Aceasta asigură o colectare robustă a datelor, capturând variaţiile minore şi eliminând anomaliile. Fiecare citire este efectuată la un interval de timp constant, cunoscut sub numele de perioadă de eşantionare. Aceasta permite senzorului să stabilizeze şi să furnizeze date precise fără suprapunerea sau duplicarea citirilor.

După colectarea celor 100 de citiri pentru fiecare axă, programul calculează media aritmetică a acestor valori. Media aritmetică se obţine prin însumarea tuturor valorilor citite şi împărţirea rezultatului la numărul de citiri (100 în acest caz). Această valoare medie este apoi utilizată pentru a compensa citirile în programul principal, îmbunătăţind astfel calibrările şi precizia măsurătorilor efectuate de senzorul MPU6050.

Foarte important este ca în momentul rulării programului de stabilire a offset-ului să nu existe ventilatoare pornite, muzică sau orice alte obiecte ce ar putea interfera cu senzorii.

Axoffset = Axoffset/iterations
Ayoffset = Ayoffset/iterations
Azoffset = Azoffset/iterations
Gxoffset = Gxoffset/iterations
Gyoffset = Gyoffset/iterations
Gzoffset = Gzoffset/iterations

```
SYSTEM NOTIFICATION 10.0 seconds calibration
        Axoffset0.02 °/s        Ayoffset=-0.01 °/s      Azoffset=1.01 °/s
        Gxoffset=-0.22 g        Gyoffset=-0.08 g        Gzoffset=-0.15 g
>>>
```

Figura 4.12. Rezultatul calibrării senzorului MPU6050

4.4. Filtrul Kalman

4.4.1. Introducere

Filtrul Kalman şi filtrul complementar sunt două abordări diferite utilizate în fuziunea datelor senzorilor pentru a estima starea unui sistem. Deşi ambele metode sunt utilizate, în mod obişnuit, în aplicaţiile de navigaţie şi control, filtrul Kalman este cunoscut pentru capacitatea sa de a oferi estimări mai precise şi mai fiabile, chiar şi în prezenţa variabilităţii măsurătorilor şi a zgomotului.

Filtrul Kalman funcţionează folosind un model matematic al sistemului şi combinând predicţiile bazate pe acest model cu măsurători în timp real într-o manieră recursivă. Acest lucru permite ajustarea continuă a estimărilor bazate atât pe datele trecute, cât şi pe cele actuale, menţinând un echilibru optim între predicţii şi corecţii. Pe de altă parte, filtrul complementar combină direct măsurătorile senzorilor fără a utiliza un model matematic al sistemului. Acest lucru poate duce la estimări mai simple, care sunt mai predispuse la erori, mai ales atunci când există o variabilitate mare de măsurare sau schimbări bruşte în mediul de măsurare.

Pe scurt, filtrul Kalman excelează în integrarea eficientă a datelor senzorilor prin utilizarea unui model matematic al sistemului şi actualizarea continuă a estimărilor pe baza noilor măsurători. Acest lucru îl face alegerea preferată în aplicaţiile în care acurateţea şi robusteţea estimării sunt cruciale, cum ar fi navigarea vehiculelor autonome sau sistemele de control de înaltă precizie.

Filtrul Kalman este un instrument matematic versatil utilizat pentru filtrarea şi estimarea datelor în sistemele de navigaţie şi alte aplicaţii cu măsurători zgomotoase şi imprecise. Creat iniţial de Rudolf Kalman în anii 1960 pentru a aborda provocările de filtrare şi predicţie în sistemele de control automat, principiile sale sunt fundamentale în procesarea semnalului, inteligenţa artificială şi au aplicaţii ample în inginerie, informatică şi economie.

Funcţionând ca o metodă de filtrare recursivă, filtrul Kalman combină predicţii bazate pe modele şi măsurători în timp real pentru a genera estimări mai precise ale stării unui sistem. Funcţionează optim pentru sistemele modelate ca procese liniare afectate de zgomot aleatoriu în măsurători.

Structura filtrului constă din doi paşi principali: predicţie şi corecţie. În etapa de predicţie, sistemul foloseşte măsurătorile anterioare şi cunoştinţele despre comportamentul sistemului pentru a prognoza starea acestuia la următorul pas de timp. Această predicţie este apoi ajustată în etapa de corecţie folosind măsurători reale în timp real pentru a rafina estimările anterioare şi pentru a produce o estimare mai precisă a stării curente a sistemului.

O caracteristică cheie a filtrului Kalman este capacitatea sa de a cântări în mod corespunzător măsurătorile şi predicţiile pe baza incertitudinilor acestora

Filtrul Kalman derivat în proiect este adaptat pentru a prezice unghiul de ruliu sau de înclinare, făcându-l un filtru unidimensional cu starea sistemului reprezentată de o singură valoare. Acest concept poate fi extins pentru a gestiona stările multidimensionale folosind vectori şi matrici. Forma generală a filtrului Kalman este

furnizată mai jos şi va fi utilizată pentru estimarea altitudinii quadcopterului în etapele ulterioare. În plus, valorile pentru toţi vectorii şi matricele sunt incluse pentru comparaţie. [26]

4.4.2. Stările sistemului

Predicţia stării actuale a sistemului:

$$S(k)=F\cdot S(k-1)+G\cdot U(k)S(k) \quad (4.1)$$

- S = Vectorul de stare,
- F = matricea de tranziţie a stărilor,
- G = matricea de control,
- U = variabilele de intrare.

Calcularea incertitudinii predicţiei:

$$P(k)=F\cdot P(k-1)\cdot FT+QP(k) \quad (4.2)$$

- P = vectorul de predicţie a incertitudinii,
- Q = procesarea incertitudinii.

Calculaţi câştigul Kalman din incertitudini privind predicţiile şi măsurătorile:

$$L(k)=H\cdot P(k)\cdot H\text{^}T+R \quad (4.3)$$

K=P(k)·H^T/L(k) = P(k) ·H^T L(k)^{-1}

- L = matricea intermediară,
- K = câştigul Kalman,
- H = matricea de observaţie,
- R = măsurarea incertitudinii.

Actualizaţi starea prezisă a sistemului cu măsurarea stării prin câştigul Kalman:

$$S(k)=S(k)+K\cdot(M(k)-H\cdot S(k)) \quad (4.4)$$

- M = vectorul de măsurare

Actualizarea incertitudinii stării prezise:

$$P(k)=(I-K\cdot F)\cdot P(k) \quad (4.5)$$

- I = matricea unitate

Câştigul Kalman este o componentă crucială a filtrului Kalman. Determină semnificaţia predicţiei unghiului realizată prin integrarea giroscopului în raport cu unghiul măsurat obţinut de la accelerometru. Valoarea câştigului se încadrează întotdeauna între zero şi unu, deoarece acţionează ca un factor de ponderare. Un câştig

Kalman mai mare acordă o importanţă mai mare măsurării accelerometrului, în timp ce un câştig mai mic prioritizează predicţia realizată prin integrarea ratei de rotaţie.

În cazul filtrului Kalman unghiular, câştigul evoluează în timp, aşa cum este prezentat în figura 4.13. În esenţă, în timpul zborului, filtrul Kalman se bazează în principal pe predicţia integrării giroscopului. Unghiurile de pas ale accelerometrului sunt utilizate pentru a se asigura că integrarea giroscopului nu se abate excesiv de la unghiurile de pas ale accelerometrului, prevenind deriva. Această metodă combină eficient punctele forte ale ambelor măsurători. [27][28]

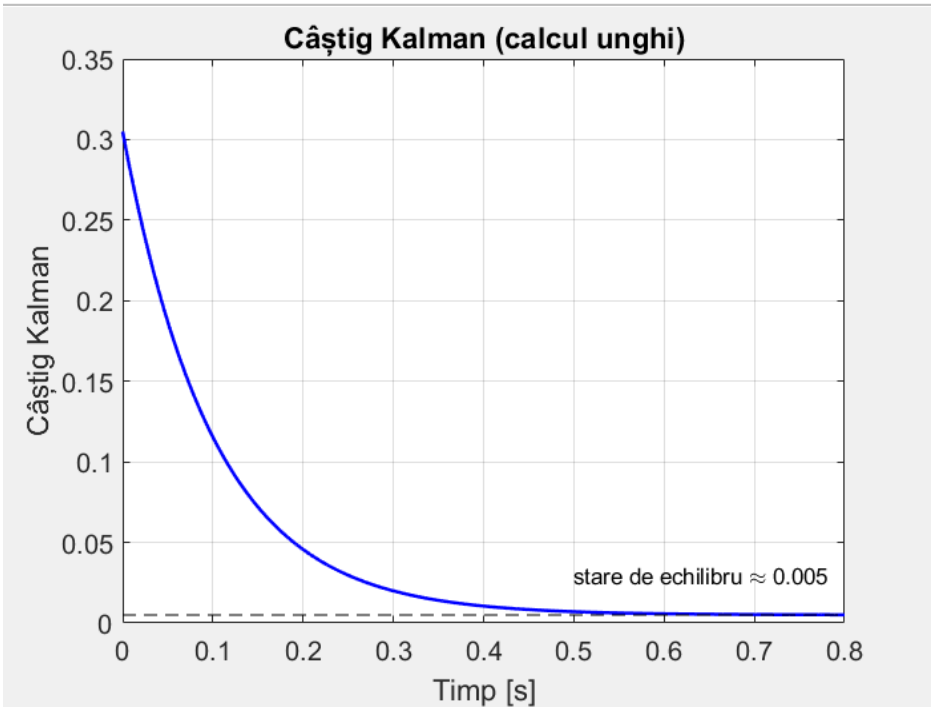

Figura 4.13. Câştigul Kalman in timp

La începutul graficului, câştigul Kalman are o valoare ridicată (aproximativ 0.3). Acest lucru indică faptul că, iniţial, filtrul Kalman acordă o pondere mare măsurătorilor noi pentru a corecta rapid orice erori iniţiale din predicţia stării sistemului.

Câştigul Kalman scade rapid la început şi continuă să scadă într-un mod exponenţial pe măsură ce timpul avansează. Aceasta este o caracteristică tipică a filtrului Kalman, unde importanţa măsurătorilor noi scade în timp, pe măsură ce estimarea stării devine mai precisă.

În jurul valorii de 0.6 secunde, câştigul Kalman atinge o stare de echilibru (steady state) în jurul valorii de 0.005. Aceasta înseamnă că, după o perioadă iniţială de ajustare, câştigul rămâne constant, indicând faptul că filtrul Kalman şi-a stabilizat estimarea şi nu mai este necesară o ajustare majoră pe baza măsurătorilor noi.

În starea de echilibru, câştigul Kalman mic indică faptul că modelul de predicţie al sistemului este de încredere, iar măsurătorile noi au o influenţă redusă asupra estimării stării. Aceasta reflectă o încredere crescută în predicţiile modelului datorită acumulării de informaţii şi ajustări repetate pe parcursul timpului.

4.4.3. Implementarea codului

Pasul de predicţie actualizează unghiul şi matricea de covarianţă a erorii de predicţie pe baza ratei giroscopului şi a incertitudinilor procesului.

```
unghi += dt * rata_gyro  # Actualizează unghiul estimat
P[0][0] += dt * (dt * P[1][1] - P[0][1] - P[1][0] + Q_unghi)
P[0][1] -= dt * P[1][1]
P[1][0] -= dt * P[1][1]
P[1][1] += Q_gyro * dt
# Se actualizează elementelor din matricea de covarianţă a erorii de predicţie
```

Pasul de actualizare calculează inovaţia şi câştigul Kalman.

```
y = unghi – unghi  # este zero pentru că măsurătoarea şi predicţia sunt aceleaşi
S = P[0][0] + R_unghi # suma incertitudinii predicţiei şi a incertitudinii măsurătorii
K[0] = P[0][0] / S
K[1] = P[1][0] / S
# Calculul câştigului Kalman
```

Pasul de corecţie ajustează unghiul şi rata giroscopului pe baza câştigului Kalman şi se actualizează matricea de covarianţă a erorii de predicţie.

```
rata_gyro -= unghi # Corecţia ratei giroscopului pe baza unghiului estimat
unghi += K[1] * rata_gyro     # Corecţia finală a unghiului estimat pe baza ratei giroscopului corectate şi a câştigului Kalman
```

```
P00_temp = P[0][0]
P01_temp = P[0][1]
# Stocarea temporară a valorilor din matricea de covarianţă a erorii de predicţie
```

```
P[0][0] -= K[0] * P00_temp
P[0][1] -= K[0] * P01_temp
P[1][0] -= K[1] * P00_temp
P[1][1] -= K[1] * P01_temp
# Actualizarea elementelor din matricea de covarianţă a erorii de predicţie pe baza câştigului Kalman şi a valorii temporare
```

Aceste linii sunt utilizate in programul funcţia *kalman_filter():* care este apelata de funcţia *get_pitch_roll():* pentru a putea calcula unghiurile cu ajutorul senzorilor.

```
angle_pitch = kalman_filter(pitch, P_pitch, K_pitch, Q_angle, Q_gyro, R_angle, gyro_x)
angle_roll = kalman_filter(roll, P_roll, K_roll, Q_angle, Q_gyro, R_angle, gyro_y)
```

4.5. Măsurarea înălţimii

4.5.1. Senzorul barometric

Senzorii barometrici, cunoscuţi şi ca barometre, joacă un rol vital în măsurarea presiunii atmosferice. Aceşti senzori sunt esenţiali în aplicaţii precum prognoza meteo, altimetria şi sistemele de control al zborului, datorită capacităţii lor de a detecta mici modificări ale presiunii.

Conceptul principal din spatele senzorilor barometrici este că pe măsură ce altitudinea creşte, presiunea atmosferică scade. Această relaţie permite ca citirile de presiune să fie utilizate pentru a estima altitudinea. Formula barometrică, care conectează presiunea atmosferică de altitudine, se bazează pe o temperatură standard de 15°C şi o presiune standard la nivelul mării de 1013.25 hPa. Această formulă este esenţială în determinarea altitudinii în diferite domenii, în special în aviaţie, unde citirile precise ale altitudinii sunt cruciale pentru zborul în siguranţă. Formula generală a presiunii atmosferice:

$$P = P_0 \cdot \left(1 - \frac{L \cdot h}{T_0}\right)^{\frac{g \cdot M}{R \cdot L}} \tag{4.6}$$

unde:

P - este presiunea atmosferică la altitudinea h,

P0 - este presiunea atmosferică standard la nivelul mării (1013.25 hPa),

L - este rata de scădere a temperaturii (aproximativ 0.0065 °C/m),

h - este altitudinea deasupra nivelului mării,

T0 - este temperatura standard la nivelul mării (288.15 K),

g - este acceleraţia datorată gravitaţiei (9.80665 m/s^2),

M - este masa molară a aerului Pământului (0.0289644 kg/mol),

R - este constanta universală a gazelor (8.3144598 J/(mol·K)).

Senzorii barometrici sunt extrem de valoroşi în controlerele de zbor pentru utilizarea lor practică. Au capacitatea remarcabilă de a detecta chiar şi cele mai mici variaţii ale presiunii atmosferice. Această sensibilitate este crucială, deoarece permite senzorilor să ofere citiri de altitudine foarte precise. Aceste citiri sunt esenţiale pentru gestionarea eficientă a traseelor de zbor şi pentru asigurarea stabilităţii aeronavei.

Înainte de a fi vândut, fiecare senzor este calibrat de producător. Valorile de calibrare sunt stocate în memoria senzorului ca doisprezece parametri de reglare, cu trei pentru temperatură şi nouă pentru presiune. Aceşti parametri sunt reprezentaţi ca numere întregi cu semn sau fără semn pe 16 biţi, aşa cum este specificat în fişa de date BMP-280.

O variabilă globală este definită pentru a reprezenta altitudinea măsurată de barometru, împreună cu altitudinea la pornire. Pentru a asigura o valoare stabilă pentru altitudine la pornire, este luată media unui număr mare de citiri de altitudine.

Adresa I2C pentru senzorul BMP-280 este setată la 0x76. Pentru a citi datele de presiune și temperatură, se inițiază o citire în rafală din registrele 0xF7 la 0xFC. Măsurătorile brute de temperatură și presiune sunt repartizate în trei registre fiecare. Sunt solicitați șase octeți pentru a citi aceste registre, iar datele sunt primite în format nesemnat pe 32 de biți.

Cele trei registre pentru temperatură și trei pentru presiune sunt combinate pentru a forma presiunea brută, necompensată și necalibrată și temperatura. Registrul de biți cel mai semnificativ conține biții de la 19 la 12, registrul de biți mai puțin semnificativ conține biții de la 11 la 4, iar registrul de biți de cel mai puțin semnificativ conține biții de la 3 la 0 ai măsurătorilor brute.

4.5.2. Implementarea software

Bibliotecile utilizate includ „*smbus2*" pentru comunicarea I2C, „*time*" pentru gestionarea temporizărilor și „*math*" pentru operațiuni matematice. Adresa I2C implicită și registrele specifice ale senzorului sunt definite pentru a facilita accesul la funcționalitățile senzorului.

```
# Adresa implicită I2C pentru BMP280
BMP280_I2C_ADDR = 0x76

# Regiștrii BMP280
BMP280_REG_DIG_T1 = 0x88
BMP280_REG_DIG_T2 = 0x8A
......
BMP280_REG_TEMPDATA = 0xFA

# Constantele senzorului
BMP280_CHIPID = 0x58
```

Aceste constante definesc adresa și registrele senzorului BMP280, necesare pentru comunicarea și citirea datelor de la senzor.

```
def write_register(reg, value):
    bus.write_byte_data(BMP280_I2C_ADDR, reg, value)
```

Funcția „*write_register*" este utilizată pentru a scrie un byte într-un registru specificat, configurând astfel senzorul.

Funcţiile „*read_unsigned_short*" şi „*read_signed_short*" sunt utilizate pentru a citi valori de 16 biţi din registrele senzorului. Aceste valori pot fi semnate sau nesemnate, în funcţie de registrele specifice.

```
def read_calibration_data():
    """Citește datele de calibrare de la senzorul BMP280."""
    cal = {}
    cal['dig_T1'] = read_unsigned_short(BMP280_REG_DIG_T1)
   ..............
    cal['dig_P9'] = read_signed_short(BMP280_REG_DIG_P9)
    return cal
```

Funcţia „*configure_sensor*" iniţializează şi configurează senzorul, verificând ID-ul cipului şi setând registrele de control şi configurare.

```
def read_raw_data():
    data = bus.read_i2c_block_data(BMP280_I2C_ADDR,
BMP280_REG_PRESSUREDATA, 6)
    raw_temp = (data[3] << 12) | (data[4] << 4) | (data[5] >> 4)
    raw_press = (data[0] << 12) | (data[1] << 4) | (data[2] >> 4)
    return raw_temp, raw_press
```

Această funcţie citeşte datele brute de temperatură şi presiune de la senzor.

Compensarea temperaturii şi presiunii implică utilizarea datelor brute şi a datelor de calibrare pentru a calcula valorile reale de temperatură şi presiune.

```
def compensate_temperature(raw_temp, cal):
    """Compensează temperatura brută folosind datele de calibrare."""
    var1 = (raw_temp / 16384.0 - cal['dig_T1'] / 1024.0) * cal['dig_T2']
    var2 = ((raw_temp / 131072.0 - cal['dig_T1'] / 8192.0) ** 2) * cal['dig_T3']
    t_fine = var1 + var2
    temp = var1 + var2
    return t_fine, temp / 5120.0

def compensate_pressure(raw_press, t_fine, cal):
    """Compensează presiunea brută folosind datele de calibrare."""
    var1 = (t_fine / 2.0) - 64000.0
    var2 = var1 * var1 * cal['dig_P6'] / 32768.0
    var2 = var2 + var1 * cal['dig_P5'] * 2.0
    var2 = (var2 / 4.0) + (cal['dig_P4'] * 65536.0)
    var1 = (cal['dig_P3'] * var1 * var1 / 524288.0 + cal['dig_P2'] * var1) / 524288.0
    var1 = (1.0 + var1 / 32768.0) * cal['dig_P1']
```

```
    if var1 == 0:
        return 0  # Evită diviziunea la zero
    pressure = 1048576.0 - raw_press
    pressure = (pressure - (var2 / 4096.0)) * 6250.0 / var1
    var1 = cal['dig_P9'] * pressure * pressure / 2147483648.0
    var2 = pressure * cal['dig_P8'] / 32768.0
    pressure = pressure + (var1 + var2 + cal['dig_P7']) / 16.0
    return pressure / 100.0
```

Funcţia „*calculate_altitude*” calculează altitudinea utilizând formula barometrică, pe baza presiunii atmosferice şi a presiunii la nivelul mării.

```
def calculate_altitude(pressure, sea_level_pressure=1013.25):
    altitude = 44330.0 * (1.0 - (pressure / sea_level_pressure) ** (1.0 / 5.255))
    return altitude
```

Acest program oferă rezultate precise pentru măsurarea altitudinilor mari, fiind util în aplicaţii de aviaţie şi navigaţie montană. Cu toate acestea, la altitudini mici, poate prezenta erori de aproximativ 50 de centimetri. Aceste erori vor fi compensate ulterior prin algoritmi de corecţie suplimentari, asigurând astfel acurateţea necesară în toate condiţiile.

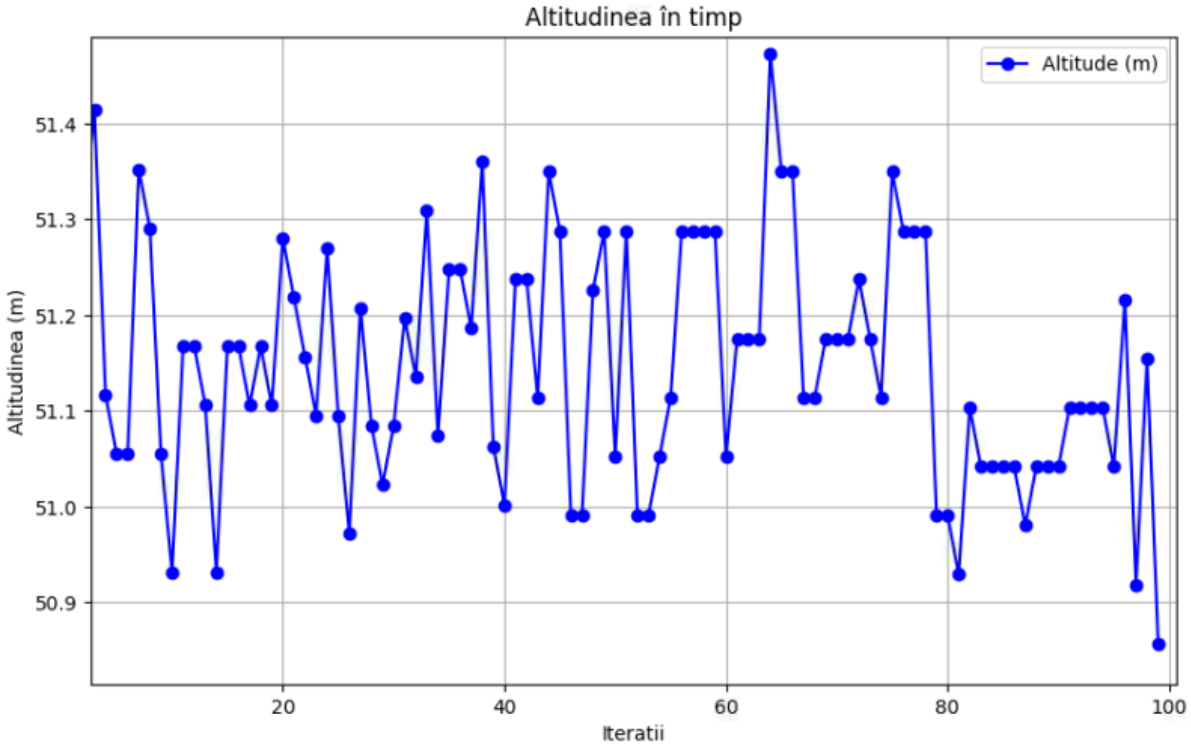

Figura 4.14. Graficul măsurătorilor altitudinii

4.5.3. *Accelerometrul*

Calcularea vitezei verticale a unui quadcopter se bazează în mare măsură pe accelerația de-a lungul axei Z inerțiale. Această accelerație, denumită AccZInertial, este determinată folosind citirile accelerometrului și orientările unghiulare.

$$AccZ_{Inertial} = -A_x \cdot \sin(\text{Pitch} \cdot \tfrac{\pi}{180}) + A_y \cdot \cos(\text{Pitch} \cdot \tfrac{\pi}{180}) \cdot \sin(\text{Roll} \cdot \tfrac{\pi}{180}) + A_z \cdot \cos(\text{Pitch} \cdot \tfrac{\pi}{180}) \cdot \cos(\text{Roll} \cdot \tfrac{\pi}{180}) \quad (4.7)$$

Unde Ax, Ay și Az sunt citirile accelerometrului ajustate pentru decalaje, iar Pitch și Roll sunt orientările unghiulare ale quadcopterului convertite în radiani.

După calcularea AccZInertial, acesta este ajustat prin scăderea a 1 g (echivalent cu 9.81 m/s^2) pentru a corecta accelerația gravitațională. Se utilizează și o scalare suplimentară cu 10, care transformă accelerația în m/s^2, care este apoi înmulțită cu 100 cm/m în scopuri practice, obținând accelerația în cm/s^2.

*AccZInertial = (AccZInertial - 1) * 9.81 * 10* (4.8.)

Pentru a determina viteza verticală VZ, accelerația AccZInertial este integrată pe intervale de timp (de obicei 0.005 secunde) pentru a actualiza continuu estimarea vitezei. Acest proces de integrare este crucial pentru obținerea de măsurători precise în timp real ale mișcării verticale a quadcopterului, asigurând control precis și stabilitate.

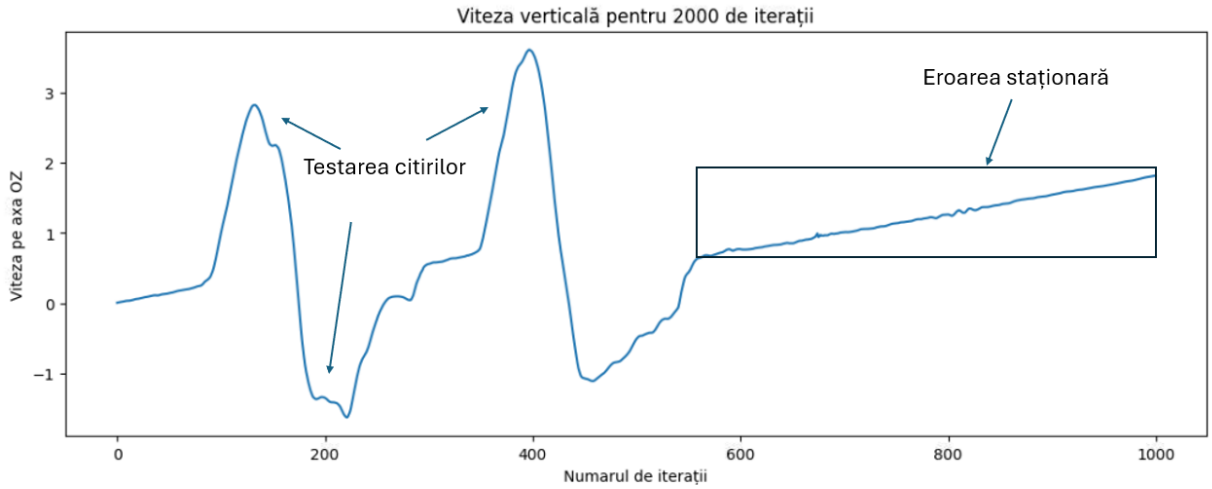

Figura 4.15. Graficul măsurătorilor altitudinii cu ajutorul accelerometrului

4.5.4. Senzorul ultrasonic

Implementarea unui sistem de senzori cu ultrasunete folosind un Raspberry Pi presupune configurarea hardware-ului, codificarea comportamentului senzorului și asigurarea unor măsurători precise ale distanței. Această secțiune elucidează codul și logica din spatele sistemului, subliniind configurarea, mecanismul de citire și procesarea datelor pentru a obține citiri de distanță fiabile.

Senzorul ultrasonic utilizat în această implementare este interfațat cu Raspberry Pi folosind doi pini GPIO: un pin de declanșare și un pin de echo. Senzorul funcționează prin emiterea de unde ultrasonice și măsurarea timpului necesar pentru ca ecoul să revină după ce sare de pe un obiect. Distanța până la obiect este apoi calculată pe baza vitezei sunetului.

Codul începe cu importarea modulelor necesare și definirea clasei UltrasonicSensor:

```
import RPi.GPIO as GPIO
import time
import statistics
```

Metoda *__init__* setează pinii GPIO și pregătește senzorul pentru funcționare:

```
class UltrasonicSensor:
  def __init__(self, trigger_pin, echo_pin):
    self.GPIO_TRIGGER = trigger_pin
    self.GPIO_ECHO = echo_pin

    # Setarea pinilor GPIO
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(self.GPIO_TRIGGER, GPIO.OUT)
    GPIO.setwarnings(False)
    GPIO.setup(self.GPIO_ECHO, GPIO.IN)
    GPIO.output(self.GPIO_TRIGGER, False)
    time.sleep(2)  # lasă senzorul sa se stabilizeze
```

Modul GPIO este setat la BCM, iar pinul de declanșare este configurat ca o ieșire, în timp ce pinul ecou este o intrare. O scurtă întârziere permite senzorului să se stabilizeze înainte de a efectua măsurători.

Metoda *read_distance* declanșează senzorul, așteaptă ecoul și calculează distanța. Senzorul este declanșat prin setarea știftului de declanșare ridicat pentru o perioadă scurtă (10 microsecunde). Codul așteaptă apoi ca pinul echo să primească semnalul, marcând ora de începere și, ulterior, să scadă, marcând timpul de oprire. Distanța este calculată folosind timpul scurs și viteza sunetului.

Metoda de citire colectează mai multe eşantioane la distanţă, filtrează valorile aberante şi calculează distanţa medie. Această metodă colectează un număr specificat de probe, calculează mediana şi filtrează citirile care se abat semnificativ de la mediană (mai mare de 10%). Media distanţelor filtrate este apoi returnată ca măsurătoare finală.

Metoda de curăţare asigură eliberarea corectă a resurselor GPIO:

```
def cleanup(self):
    GPIO.cleanup()
```

Următorul fragment de cod demonstrează cum se utilizează clasa UltrasonicSensor într-o buclă de citire continuă:

```
if __name__ == "__main__":
  try:
    sensor = UltrasonicSensor(trigger_pin=4, echo_pin=17)
    while True:
      distance = sensor.read()
      if distance is not None:
        print("Distance:", distance, "cm")
      else:
        print("Failed to read distance")
      time.sleep(0.2)  # Se reduce întârzierea dintre citiri la 0,2 secunde
  except KeyboardInterrupt:
    sensor.cleanup()
```

Acest cod oferă o metodă robustă şi fiabilă pentru măsurarea distanţelor folosind un senzor ultrasonic cu un Raspberry Pi. Prin declanşarea senzorului, captarea ecoului şi filtrarea datelor, sistemul asigură măsurători precise şi consistente ale distanţelor, potrivite pentru diverse aplicaţii în robotică şi automatizare.

4.6. Controlul sistemului

4.6.1. Radiocomanda

Scriptul Python a furnizat interfeţe cu un controler radio FS-iA6B printr-o conexiune serială folosind modulul serial. Scriptul citeşte datele canalului de la radio, le procesează, actualizează o altitudine de referinţă şi înregistrează informaţiile. Această secţiune include un fundal teoretic despre comunicarea în serie şi discută despre implementarea scriptului.

Comunicarea în serie implică transmiterea datelor câte un bit pe un canal de comunicaţie. Este folosit în mod obişnuit în telecomunicaţii şi conectarea perifericelor la computere. Parametrii importanţi în comunicarea în serie includ rata de transmisie, paritatea, biţii de oprire şi formatul cadrului de date. Scriptul setează aceşti parametri pentru a se potrivi cu cerinţele modulului FS-iA6B.

Scriptul începe prin configurarea modulului de logare în scopuri de depanare și apoi continuă cu definirea clasei RadioController, responsabilă cu gestionarea comunicației cu controlerul radio FS-iA6B.

```
import serial
import time
import logging

# Configurare logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')
```

Configurația de înregistrare este setată în prezent la nivelul *DEBUG*, permițând jurnalele complete ale operațiunilor scriptului.

La instanțiere, constructorul stabilește conexiunea serială și atribuie valori implicite datelor de canal și altitudinii de referință.

```
class RadioController:
  def __init__(self, serial_port="/dev/serial0", baudrate=115200):
    self.ser = serial.Serial(serial_port, baudrate)
    self.ser.parity = serial.PARITY_NONE
    self.ser.stopbits = serial.STOPBITS_ONE
    self.Reference_altitude = 0
    self.ch1, self.ch2, self.ch3, self.ch4, self.ch5, self.ch6 = 0, 0, 0, 0, 0, 0
```

În această setare, conexiunea serială este stabilită cu portul și rata de transmisie desemnate, iar paritatea și biții de oprire sunt ajustați în consecință. Datele canalului și altitudinea de referință sunt setate la valorile implicite.

Funcția *read_data* citește datele primite de la portul serial și le procesează atunci când este primit un cadru complet. Scriptul citește octeți din memoria tampon serial și îi adaugă într-un cadru. Dacă este detectat un cadru complet (cel puțin 32 de octeți, începând cu un anumit octet), acesta procesează cadrul pentru a extrage datele canalului și actualizează altitudinea de referință pe baza datelor canalului.

Funcția *_process_frame* extrage datele individuale ale canalului din cadru. Aceasta convertește secțiunile de octeți în numere întregi pentru fiecare canal și ajustează canalul 3 (ch3) după cum este necesar.

Funcția *_update_reference_altitude* actualizează altitudinea de referință pe baza valorii canalului 3, ajustând altitudinea de referință în mod incremental în funcție de intervalul în care se încadrează ch3.

Funcțiile *get_reference_altitude* și *get_channels_data* oferă acces la altitudinea de referință și, respectiv, datele de canal.

```
def get_reference_altitude(self):
    return self.Reference_altitude

def get_channels_data(self):
    return self.ch1, self.ch2, self.ch3, self.ch4, self.ch5, self.ch6
```

Metoda de curăţare închide conexiunea serială.

```
def cleanup(self):
  self.ser.close()
```

4.6.2. Multitasking sau Multithreading

Multitasking-ul în Python implică capacitatea unui sistem de operare de a gestiona mai multe sarcini simultan. Acest lucru poate fi realizat folosind modulul de multiprocesare, care permite crearea şi gestionarea proceselor separate care rulează concomitent. Aceste procese funcţionează independent pe diferite nuclee CPU, oferind un adevărat paralelism pe sistemele multi-core. Prin distribuirea sarcinilor legate de CPU pe mai multe procesoare, performanţa şi eficienţa pot fi îmbunătăţite. Mecanismele de comunicare între procese, cum ar fi cozile şi conductele, sunt utilizate pentru a permite comunicarea între aceste procese independente.

Multithreading implică utilizarea mai multor fire într-un singur proces pentru a obţine concurenţă. Thread-urile sunt uşoare şi partajează acelaşi spaţiu de memorie, făcându-le ideale pentru sarcini legate de I/O care implică aşteptarea resurselor externe, cum ar fi sistemele de fişiere sau conexiunile de reţea. Modulul de threading al lui Python permite crearea şi gestionarea thread-urilor. [29]

Cu toate acestea, este important să se ia în considerare Blocarea globală a interpretului (GIL) de la Python atunci când se lucrează cu multithreading. GIL asigură că doar un fir de execuţie poate executa bytecode Python la un moment dat, chiar şi pe sisteme cu mai multe nuclee. Această limitare înseamnă că multithreading-ul în Python nu oferă o execuţie paralelă reală pentru sarcinile legate de CPU. În ciuda acestui fapt, poate fi în continuare eficient pentru sarcini care implică aşteptarea finalizării operaţiunilor I/O.

Implementarea multitasking-ului cu multiprocesare în Python permite citirea şi procesarea simultană a datelor de la diverşi senzori. Sistemul constă din procese separate pentru citirea datelor de la un senzor ultrasonic, un IMU, un senzor BMP280 şi un controler radio. Datele fiecărui senzor sunt gestionate de propriul său proces.

```
if __name__ == "__main__":
    queue = mp.Queue()

    processes = [
        mp.Process(target=read_ultrasonic, args=(queue,)),
        mp.Process(target=read_imu_and_bmp, args=(queue,)),
```

```
        mp.Process(target=read_radio, args=(queue,)),
        mp.Process(target=process_data, args=(queue,))
    ]
    for p in processes:
        p.start()
    for p in processes:
        p.join()
```

Pentru a iniţializa procesele, este creată o coadă pentru comunicarea între acestea. Sunt create patru fire de execuţie: unul pentru fiecare funcţie de citire a senzorului şi unul pentru procesarea datelor colectate. Aceste procese sunt pornite şi unite pentru a se asigura că programul principal aşteaptă finalizarea lor.

Datele de la fiecare funcţie de citire a senzorului sunt plasate în coada partajată, iar funcţia *process_data* preia aceste date pentru o manipulare sincronizată şi eficientă. Datele sunt stocate într-un dicţionar pentru a se asigura că datele de la toţi senzorii sunt colectate înainte de procesare, permiţând o gestionare coerentă a datelor.

```
def process_data(queue):
    data_dict = {
        'ultrasonic': None,
        'imu_bmp': None,
        'radio': None
    }
    while True:
        try:
            sensor_type, data = queue.get(timeout=0.02)
            data_dict[sensor_type] = data
        except mp.queues.Empty:
            continue
        if all(value is not None for value in data_dict.values()):
            # Process and print the collected data
            ...
```

Prin utilizarea modulului de multiprocesare, scriptul citeşte eficient datele de la mai mulţi senzori în paralel, asigurând o procesare a datelor în timp util şi sincronizată.

4.6.3. PID

Controlul utilizând PID (Proportional-Integral-Derivative) este un mecanism de feedback utilizat în sistemele de control pentru a atinge și susține un punct de referință dorit. Funcționează prin utilizarea a trei componente principale:

1. Termen proporțional (*P*): Acest termen generează o ieșire care este direct proporțională cu eroarea curentă. Răspunde imediat la mărimea erorii curente și este controlată de câștigul proporțional (*Kp*). O valoare Kp mai mare are ca rezultat un răspuns mai pronunțat la erorile curente.

2. Termenul integral (*I*): Termenul integral integrează eroarea în timp și răspunde erorii acumulate într-o perioadă. Ajută la eliminarea oricărei erori de stare staționară care rămâne după ce termenul proporțional a ajustat ieșirea. Câștigul integral (*Ki*) determină nivelul de agresivitate cu care controlerul elimină aceste erori acumulate.

3. Termenul derivat (*D*): Termenul derivat prezice tendințele viitoare de eroare pe baza ratei actuale de modificare a erorii. Atenuează răspunsul pentru a reduce depășirea și oscilațiile. Câștigul derivat (*Kd*) controlează măsura în care controlerul reacționează la rata de modificare a erorii.

Prin combinarea acestor trei termeni, controlerul PID calculează ieșirea de control. Reglează continuu ieșirea pe baza semnalului de eroare (*error = setpoint - process_variable*) pentru a minimiza eroarea și a stabiliza sistemul în jurul valorii de referință dorite.

Clasa PID este definită pentru a inițializa și actualiza valorile PID. Constructorul *(__init__)* setează câștigurile proporțional *(kp)*, integral (*ki*), și derivat (*kd*), precum și punctul de referință (*setpoint*). Variabilele pentru eroarea anterioară și integrala erorii sunt, de asemenea, inițializate.

```
def __init__(self, kp, ki, kd, setpoint=0):
    self.kp = kp
    self.ki = ki
    self.kd = kd
    self.setpoint = setpoint
    self.previous_error = 0
    self.integral = 0
```

Metoda update primește valoarea curentă (*current_value*) și intervalul de timp (*dt*). Calculează eroarea ca diferență între punctul de referință și valoarea curentă, actualizează integralul erorii și derivata, și returnează ieșirea PID ca sumă a termenilor proporțional, integral și derivat.

```
def update(self, current_value, dt):
    error = self.setpoint - current_value
    self.integral += error * dt
    derivative = (error - self.previous_error) / dt
```

*output = self.kp * error + self.ki * self.integral + self.kd * derivative*
self.previous_error = error
return output

În funcţia *process_data*, PID-ul este iniţializat cu valori de câştiguri (kp=4, ki=0.01, kd=2) şi punctul de referinţă este actualizat conform altitudinii de referinţă (*reference_altitude*). În cadrul buclei de procesare, PID-ul este actualizat cu altitudinea finală (*final_altitude*) şi timpul (*Time*), producând o ajustare pentru controlul canalului 3 (*ch3*).

 # Actualizăm PID-ul cu noul feedback
pid.setpoint = reference_altitude
pid_output = pid.update(final_altitude, Time)

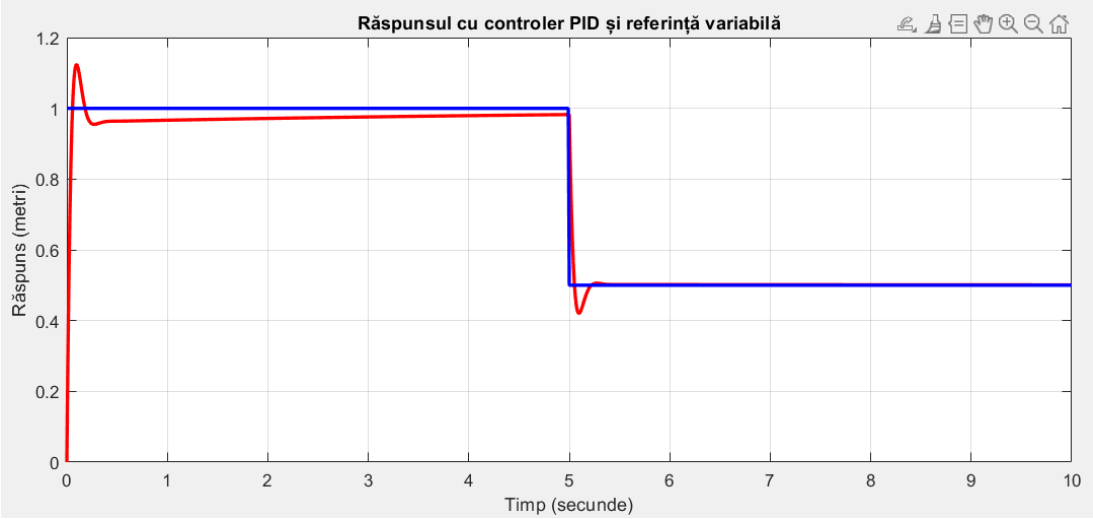

Figura 4.16. Graficul structurii de control PID

În rezumat, controlul PID asigură stabilitatea şi acurateţea în controlul altitudinii prin ajustarea dinamică a Canalului 3 pe baza măsurătorilor de altitudine în timp real şi al altitudinii de referinţă dorite. Pentru o stabilitate mai bună, se poate efectua tunning-ul parametrilor PID (Kp, Ki, Kd) prin metode cum ar fi metoda Ziegler-Nichols, metoda de răspuns la pas sau prin utilizarea algoritmilor de optimizare. Acest tunning ajută la optimizarea răspunsului sistemului, reducerea timpului de stabilizare, minimizarea depăşirii maxime şi eliminarea erorilor de regim permanent.

4.6.4. Trimiterea datelor către drona

Clasa IBUSController din acest proiect este responsabilă pentru gestionarea conversiei şi transmiterii datelor în format IBUS, utilizat în mod obişnuit în aplicaţiile de control de la distanţă pentru dispozitivele RC. Codul include următoarele componente cheie:

Constructorul clasei IBUSController stabileşte o conexiune serială folosind modulul serial al lui Python, configurând parametri precum portul şi rata de baud. Un LED este de asemenea iniţializat pentru a indica starea transmisiei.

Metoda *send_ibus_packet(radio,channels)* transmite un cadru IBUS prin conexiunea serială, cu gestionarea excepţiilor pentru a înregistra orice erori de transmisie.

Gestionarea excepţiilor şi curăţarea: Dacă programul este întrerupt de o apăsare a tastei sau întâlneşte excepţii neaşteptate, se efectuează înregistrarea corespunzătoare şi conexiunea serială este închisă pentru curăţare.

CAPITOLUL 5. COSTURI SI FIABILITATE

Când vine vorba de construirea unei drone, două aspecte esenţiale care influenţează semnificativ succesul proiectului sunt costurile şi fiabilitatea componentelor. Alegerea pieselor potrivite nu doar că afectează performanţa generală a dronei, dar şi determină cât de frecvent vor apărea defecţiuni şi necesitatea unor eventuale reparaţii sau înlocuiri. În analiza de faţă, ne vom concentra pe costurile şi fiabilitatea pieselor alese, comparându-le cu alte variante disponibile pe piaţă şi evaluând impactul unor posibile probleme asupra funcţionalităţii dronei.

Tabelul 5.1. Costurile pieselor dronei alese

Piese	Preţ
1 x Cadru dronă din fibră de carbon	138
1 x Cadrul dronei din plastic	134
4 x Motoare fără perii BrotherHobby Avenger 2507 V2 12000kV	500
4 x 40 Amperi Esc pentru motoare fără perii	128
2 x Elici 7 inch (17.78 cm)	54
2 x Baterii 6s 3300mAh	364.81
1 x Sistemul video	468
1 x Sistemul de radiocomandă	355
1 x Speedybee controller	348
1 x Încărcător baterii	268.81
Senzori, Cabluri, Piese	250
TOTAL	3008.62

În construcţia unei drone, investiţia în piese de calitate superioară joacă un rol crucial în asigurarea fiabilităţii şi durabilităţii aparatului. Componentele precum motoarele fără perii, cadrul din fibră de carbon, sistemele de control şi sistemele video de înaltă calitate sunt proiectate pentru a rezista la condiţii de operare variate şi solicitări intense. Aceste piese sunt concepute să gestioneze şocuri, vibraţii şi tensiuni mari care pot apărea în timpul zborului sau în condiţii meteorologice dificile.

De exemplu, motoarele fără perii de înaltă performanţă sunt construite pentru a oferi o putere mare şi o rezistenţă sporită la uzură, asigurând o funcţionare fiabilă în diverse condiţii de zbor. Cadrele din fibră de carbon sunt cunoscute pentru rezistenţa lor excelentă la impact şi greutatea redusă, fapt ce contribuie la manevrabilitatea şi stabilitatea dronei în aer. Sistemele de control avansate şi sistemele video de calitate

superioară sunt esențiale pentru a menține o legătură stabilă și sigură între operator și dronă, minimizând riscul de interferențe și pierderi de semnal.

În contrast, piesele mai ieftine pot să nu ofere aceeași rezistență și performanță, putând să cedeze sub presiunea utilizării intensive sau în condiții neprevăzute. Investiția inițială în piese de calitate superioară poate preveni costurile ulterioare asociate cu înlocuiri frecvente și reparări, asigurând un zbor stabil și sigur pe termen lung.

Astfel, alegerea componentelor care sunt capabile să reziste la solicitări ridicate și să funcționeze fără probleme în diverse condiții reprezintă un aspect esențial în construcția unei drone fiabile și performante.

Pentru construcția acestei drone, au fost selectate componente specifice, fiecare având un rol esențial în funcționarea corectă a aparatului. Se observă o diversitate în prețuri, de la cadrele din fibră de carbon și plastic, până la motoare și sisteme de control sofisticate. De exemplu, cadrul din fibră de carbon costă 138 lei, iar cel din plastic 134 lei. Motoarele BrotherHobby Avenger 2507 V2 12000kV sunt printre cele mai costisitoare piese, cu un preț de 500 lei pentru un set de patru, reflectând calitatea superioară și performanța ridicată.

Există, desigur, variante mai scumpe și mai ieftine pentru fiecare componentă. Variantele mai scumpe, cum ar fi motoarele de înaltă performanță sau sistemele video avansate, oferă de obicei o fiabilitate și o durabilitate superioară, în timp ce variantele mai ieftine pot economisi bani pe termen scurt, dar pot necesita înlocuiri mai frecvente și pot oferi o performanță inferioară.

Piesele selectate pentru această dronă sunt de o calitate bună, echilibrând eficient costul cu performanța. De exemplu, motoarele fără perii BrotherHobby sunt cunoscute pentru eficiența și durabilitatea lor, asigurând o putere ridicată și o întreținere minimă. La fel, sistemul video și cel de radiocomandă, deși costisitoare (468 lei și 355 lei), oferă o gamă extinsă și o transmisie clară, esențiale pentru navigarea și controlul precis al dronei.

În cazul unei probleme la sistemul de radiocomandă, fiabilitatea dronei poate fi serios afectată. Dacă sistemul de radiocomandă nu funcționează corect, controlul dronei poate fi pierdut complet, ducând la o prăbușire. Este crucial să se investească într-un sistem de radiocomandă de înaltă calitate și să se aibă măsuri de siguranță, cum ar fi moduri de failsafe care aduc drona înapoi la punctul de decolare în caz de pierdere a semnalului.

Problemele la sistemul video pot afecta abilitatea operatorului de a vedea în timp real ce filmează drona, ceea ce este esențial pentru aplicații de filmare și navigare precisă. Un sistem video de calitate, deși scump, reduce riscul de interferență și pierdere a semnalului, asigurând o transmisie stabilă și clară.

Problemele la sistemul de alimentare pot fi critice, deoarece bateriile și încărcătoarele de calitate slabă pot duce la întreruperea zborului sau chiar la incendii. Investiția în baterii de înaltă calitate, cum ar fi cele 6s 3300mAh, și încărcătoare fiabile, poate prelungi durata de viață a bateriilor și asigura un zbor sigur și stabil.

Sistemul de control, inclusiv controlerul Speedybee, este vital pentru stabilitatea și manevrabilitatea dronei. Un controler de înaltă calitate asigură integrarea precisă a senzorilor și răspunsuri rapide la comenzi, esențiale pentru un zbor lin și sigur. Problemele în acest sistem pot duce la instabilități în zbor și pierderea controlului, subliniind importanța investiției într-un controler robust și fiabil.

În concluzie, deși piesele de înaltă calitate pentru drone pot avea un cost inițial semnificativ, fiabilitatea, performanța și durabilitatea lor fac ca investiția să merite. Optarea pentru alternative mai ieftine poate economisi bani inițial, dar acestea conduc adesea la cheltuieli mai mari pe termen lung din cauza înlocuirilor frecvente, reparațiilor și potențialelor defecțiuni. În plus, compatibilitatea componentelor este un factor crucial care poate duce la costuri suplimentare. Asigurarea integrării fără întreruperi a tuturor părților poate preveni probleme precum incompatibilitatea și defecțiunile sistemului, care pot fi atât costisitoare, cât și consumatoare de timp de rezolvat. Prin urmare, prioritizarea componentelor de calitate și luarea în considerare a integrării globale a sistemului poate duce în cele din urmă la o dronă mai fiabilă, mai eficientă și mai rentabilă.

Pentru a atinge o durată de zbor de până la 40 de minute în zborurile de curse sau sportive, alegerea corectă a componentelor din lista menționată este esențială. Bateriile LiPo de 6s cu o capacitate mare, precum cele de 3300mAh, oferă suficientă energie pentru zboruri lungi și intense, asigurând o putere constantă și fiabilă. Acestea sunt încărcate și gestionate eficient de un încărcător dedicat, care contribuie la prelungirea duratei de viață a bateriilor și la o performanță stabilă în timpul zborurilor.

În ansamblu, integrarea și optimizarea corectă a acestor piese esențiale din lista de componente selectate sunt fundamentale pentru a asigura o durată de zbor lungă și fiabilă în cadrul competițiilor de drone. Alegerea acestor piese în conformitate cu cerințele specifice ale zborului de curse sau sportiv contribuie la maximizarea performanței și la minimizarea riscurilor în timpul zborurilor intensive și solicitante.

CONCLUZII

În concluzie, acest proiect s-a dovedit a fi extrem de provocator şi complex, necesitând o investiţie substanţială de timp şi efort. Au fost dedicate aproximativ 150 de ore doar pentru a identifica şi selecta componentele adecvate pentru dronă, cu toate că acestea nu au prezentat întotdeauna o compatibilitate perfectă. Pe lângă aceasta, au fost necesare aproximativ 200 de ore suplimentare pentru înţelegerea şi documentarea tehnologiilor utilizate, protocoalelor de comunicaţie, filtrelor şi metodelor de citire şi prelucrare a datelor de la senzori.

Factorul de influenţă în alegerea acestei teme a fost industria dronelor ce se află într-o continua dezvoltare atât din punct de vedere economic cat si tehnologică. Deşi provocările întâmpinate au fost numeroase, fiecare etapă a proiectului a fost abordată cu entuziasm şi curiozitate, transformând dificultăţile într-o experienţă plăcută şi educativă. Procesul de lucru a condus la acumularea unei vaste cunoştinţe, care va fi de mare folos pe termen lung, oferind o înţelegere solidă a modului de gestionare a proiectelor ambiţioase şi menţinerea motivaţiei chiar şi în faţa incertitudinii privind finalizarea la timp a acestora.

Se cuvine să fie felicitate persoanele care au parcurs această lucrare până la final, deoarece efortul şi dedicarea lor sunt demne de admirat. Sperăm că această lectură le-a oferit perspective valoroase şi inspiraţie. Din acest punct, sistemul dezvoltat poate fi extins nu doar pentru menţinerea altitudinii, ci şi pentru controlul autonom complet al dronei, deschizând noi direcţii pentru inovaţie şi aplicabilitate practică în diverse domenii. În plus, aceste tehnologii avansate pot contribui la dezvoltarea unor drone capabile de misiuni complexe şi precise, demonstrând astfel potenţialul imens al acestor dispozitive în viitorul apropiat.

BIBLIOGRAFIE

[1] https://www.descopera.ro/istorie/16596686-aceasta-este-prima-drona-din-istorie-imaginile-aeriene-prezinta-dezastrul-produs-de-cutremurul-din-1907-galerie-foto, 02-06-2024.

[2] https://idron.ro/blog/istoria-dronelor-de-la-concept-la-tehnologie-avansata-tipuri-de-drone-motoare-componente-si-evolutie/ 03-06-2024

[3] https://surveygyaan.medium.com/sensors-used-in-drones-e6f29be61fb4 06-06-2024

[4] https://www.visionresearchreports.com/drone-data-services-market/40866 08-06-2024

[5] https://oscarliang.com/how-to-build-fpv-drone/#Can-Beginner-Build-FPV-Drones 4-12-2023

[6] https://www.linkedin.com/advice/0/how-do-you-select-materials-components-drone-project-skills-drones 11-12-2023

[7] https://oscarliang.com/flight-controller/#Flight-Controller-What-it-is-and-How-it-Works 14-06-2024

[8] https://www.flyrobo.in/blog/flight-controller 10-06-2024

[9] https://hobbymania.com.ua/file/SpeedyBee_F405_V3_Stack.pdf 29-05-2024

[10] https://oscarliang.com/motors/#Brushless-vs-Brushed-Motors 13-06-2024

[11] https://rcdrone.top/blogs/articles/how-to-connect-esc-to-motor-a-step-by-step-guide-for-fpv-drones 10-11-2023

[12] https://dronenodes.com/brushless-motor-kv-rating-explained/ 24-04-2024

[13] https://www.quora.com/Which-way-does-a-propeller-turn-to-go-forward

[14] https://device.report/manual/9161621 12-11-2023

[15] https://www.sciencedirect.com/topics/engineering/ultrasonic-sensor 23-02-2024

[16] https://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/ 04-01-2024

[17] https://www.quora.com/In-what-ways-do-magnetic-sensors-work 06-01-2024

[18] https://my.avnet.com/abacus/solutions/technologies/sensors/pressure-sensors/media-types/barometric/ 20-12-2023

[19] https://circuitdigest.com/article/everything-you-need-to-know-about-the-flysky-fs-i6-transmitter-and-receiver-for-drone-control 20-02-2024

[20] https://files.banggood.com/2016/09/FS-i6X%20User%20manual.pdf 26-02-2024

[21] https://www.raspberrypi.com/news/raspberry-pi-os-debian-bullseye/ 09-06-2024

[22] https://docs.python.org/3/whatsnew/3.12.html 30-03-2024

[23] https://images.app.goo.gl/bs5hABZn5o3EGr4j6 15-06-2024

[24] https://images.app.goo.gl/c2LPfK2a9PTuKFc98 17-06-2024

[25] https://www.mathworks.com/help/matlab/supportpkg/enablei2c.html 04-04-2024

[26] Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME--Journal of Basic Engineering, 82(Series D), 35-45.

[27] Carbon_Aeronautics_Quadcopter_Manual 121-128. https://github.com/CarbonAeronautics/Manual-Quadcopter-Drone/blob/main/Carbon_Aeronautics_Quadcopter_Manual.pdf 21-10-2023

[28] Junjia Yang et al. "UAV Altitude Measurement Method Based on Data Fusion and Kalman Filter." *Journal of Physics: Conference Series*, vol. 1631, 2020, doi:10.1088/1742-6596/1631/1/012094.

[29] https://www.geeksforgeeks.org/difference-between-multithreading-vs-multiprocessing-in-python/ 13-05-2024

ANEXA 1. SENZORUL BAROMETRIC

```
import smbus2
import time
import math

class BMP280:
    def __init__(self, bus_number=1, device_address=0x76):
        # Inițializează bus-ul I2C și adresa dispozitivului
        self.bus = smbus2.SMBus(bus_number)
        self.device_address = device_address

        # Registrele BMP280
        self.BMP280_REG_DIG_T1 = 0x88
        self.BMP280_REG_DIG_T2 = 0x8A
        self.BMP280_REG_DIG_T3 = 0x8C
        self.BMP280_REG_DIG_P1 = 0x8E
        self.BMP280_REG_DIG_P2 = 0x90
        self.BMP280_REG_DIG_P3 = 0x92
        self.BMP280_REG_DIG_P4 = 0x94
        self.BMP280_REG_DIG_P5 = 0x96
        self.BMP280_REG_DIG_P6 = 0x98
        self.BMP280_REG_DIG_P7 = 0x9A
        self.BMP280_REG_DIG_P8 = 0x9C
        self.BMP280_REG_DIG_P9 = 0x9E
        self.BMP280_REG_CHIPID = 0xD0
        self.BMP280_REG_VERSION = 0xD1
        self.BMP280_REG_SOFTRESET = 0xE0
        self.BMP280_REG_CONTROL = 0xF4
        self.BMP280_REG_CONFIG = 0xF5
        self.BMP280_REG_PRESSUREDATA = 0xF7
        self.BMP280_REG_TEMPDATA = 0xFA

        # Constanta pentru chip ID-ul BMP280
        self.BMP280_CHIPID = 0x58

        # Inițializează datele de calibrare
        self.calibration_data = self.configure_sensor()

    def read_unsigned_short(self, reg):
        """Citește un short fără semn din registrul specificat."""
        data = self.bus.read_i2c_block_data(self.device_address, reg, 2)
        return data[0] + (data[1] << 8)
```

```
    def read_signed_short(self, reg):
        """Citește un short cu semn din registrul specificat."""
        data = self.bus.read_i2c_block_data(self.device_address, reg, 2)
        result = data[0] + (data[1] << 8)
        if result > 32767:
            result -= 65536
        return result

    def read_calibration_data(self):
        """Citește datele de calibrare de la senzorul BMP280."""
        cal = {}
        cal['dig_T1'] = self.read_unsigned_short(self.BMP280_REG_DIG_T1)
        cal['dig_T2'] = self.read_signed_short(self.BMP280_REG_DIG_T2)
        cal['dig_T3'] = self.read_signed_short(self.BMP280_REG_DIG_T3)
        cal['dig_P1'] = self.read_unsigned_short(self.BMP280_REG_DIG_P1)
        cal['dig_P2'] = self.read_signed_short(self.BMP280_REG_DIG_P2)
        cal['dig_P3'] = self.read_signed_short(self.BMP280_REG_DIG_P3)
        cal['dig_P4'] = self.read_signed_short(self.BMP280_REG_DIG_P4)
        cal['dig_P5'] = self.read_signed_short(self.BMP280_REG_DIG_P5)
        cal['dig_P6'] = self.read_signed_short(self.BMP280_REG_DIG_P6)
        cal['dig_P7'] = self.read_signed_short(self.BMP280_REG_DIG_P7)
        cal['dig_P8'] = self.read_signed_short(self.BMP280_REG_DIG_P8)
        cal['dig_P9'] = self.read_signed_short(self.BMP280_REG_DIG_P9)
        return cal

    def write_register(self, reg, value):
        """Scrie un byte în registrul specificat."""
        self.bus.write_byte_data(self.device_address, reg, value)

    def configure_sensor(self):
        """Configurează senzorul BMP280 cu filtrul activat."""
        # Resetează senzorul
        self.write_register(self.BMP280_REG_SOFTRESET, 0xB6)
        time.sleep(0.2)

        # Citește și verifică chip ID-ul
                    chip_id = self.bus.read_byte_data(self.device_address,
self.BMP280_REG_CHIPID)
        if chip_id != self.BMP280_CHIPID:
            raise RuntimeError('BMP280 chip ID mismatch')

        # Citește datele de calibrare
```

```
        calibration_data = self.read_calibration_data()

            # Configurează registrul de control: mod normal, oversampling x1 (temperatură și presiune)
        self.write_register(self.BMP280_REG_CONTROL, 0x27)

            # Configurează registrul de config: timp de standby 0.5ms, coeficient de filtrare 16 (0x14)
        self.write_register(self.BMP280_REG_CONFIG, 0x14)

        return calibration_data

    def read_raw_data(self):
        """Citește datele brute de temperatură și presiune de la senzorul BMP280."""
                        data = self.bus.read_i2c_block_data(self.device_address, self.BMP280_REG_PRESSUREDATA, 6)
        raw_temp = (data[3] << 12) | (data[4] << 4) | (data[5] >> 4)
        raw_press = (data[0] << 12) | (data[1] << 4) | (data[2] >> 4)
        return raw_temp, raw_press

    def compensate_temperature(self, raw_temp):
        """Compensează temperatura brută folosind datele de calibrare."""
        cal = self.calibration_data
        var1 = (raw_temp / 16384.0 - cal['dig_T1'] / 1024.0) * cal['dig_T2']
        var2 = ((raw_temp / 131072.0 - cal['dig_T1'] / 8192.0) ** 2) * cal['dig_T3']
        t_fine = var1 + var2
        temp = var1 + var2
        return t_fine, temp / 5120.0

    def compensate_pressure(self, raw_press, t_fine):
        """Compensează presiunea brută folosind datele de calibrare."""
        cal = self.calibration_data
        var1 = (t_fine / 2.0) - 64000.0
        var2 = var1 * var1 * cal['dig_P6'] / 32768.0
        var2 = var2 + var1 * cal['dig_P5'] * 2.0
        var2 = (var2 / 4.0) + (cal['dig_P4'] * 65536.0)
          var1 = (cal['dig_P3'] * var1 * var1 / 524288.0 + cal['dig_P2'] * var1) / 524288.0
        var1 = (1.0 + var1 / 32768.0) * cal['dig_P1']
        if var1 == 0:
          return 0  # evită împărțirea la zero
        pressure = 1048576.0 - raw_press
        pressure = (pressure - (var2 / 4096.0)) * 6250.0 / var1
```

```
        var1 = cal['dig_P9'] * pressure * pressure / 2147483648.0
        var2 = pressure * cal['dig_P8'] / 32768.0
        pressure = pressure + (var1 + var2 + cal['dig_P7']) / 16.0
        return pressure / 100

    def calculate_altitude(self, pressure, sea_level_pressure=1013.25):
        """Calculează altitudinea folosind presiunea atmosferică."""
        return 44330.0 * (1.0 - (pressure / sea_level_pressure) ** 0.1903)

    def read_sensor_data(self):
        """Citește și compensează datele de temperatură, presiune și altitudine."""
        raw_temp, raw_press = self.read_raw_data()
        t_fine, temperature = self.compensate_temperature(raw_temp)
        pressure = self.compensate_pressure(raw_press, t_fine)
        altitude = self.calculate_altitude(pressure)
        return temperature, pressure, altitude

class BMP280Calibrator:
    def __init__(self, bmp280):
        # Inițializează obiectul BMP280
        self.bmp280 = bmp280

    def calculate_altitude_offset(self):
        """Calculează offset-ul altitudinii."""
        readings = []
        print("Calibrating BMP280, please wait 13 seconds.")
        for _ in range(100):
            _, _, altitude = self.bmp280.read_sensor_data()
            readings.append(altitude)
            time.sleep(0.2)
        # Exclude primele 2 citiri
        return sum(readings[2:]) / len(readings[2:])

# Exemplu de utilizare
if __name__ == "__main__":
    bmp280 = BMP280()
    calibrator = BMP280Calibrator(bmp280)
    baseline_altitude = calibrator.calculate_altitude_offset()
    print(f"Baseline Altitude: {baseline_altitude:.2f} m")

    try:
        while True:
            temperature, pressure, altitude = bmp280.read_sensor_data()
```

```
            adjusted_altitude = altitude - baseline_altitude
                print(f"Temperature: {temperature:.2f} °C, Pressure: {pressure:.2f} hPa,
Altitude: {adjusted_altitude:.2f} m")
            time.sleep(0.2)  # Ajustare la intervalul de timp corect
        except KeyboardInterrupt:
            print("Iesire...")
```

ANEXA 2. SENZORUL ULTRASONIC

```
import RPi.GPIO as GPIO
import time
import statistics

class UltrasonicSensor:
  def __init__(self, trigger_pin, echo_pin):
    # Initializează pinii GPIO pentru senzorul ultrasonic
    self.GPIO_TRIGGER = trigger_pin
    self.GPIO_ECHO = echo_pin

    # Configurează pinii GPIO
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(self.GPIO_TRIGGER, GPIO.OUT)
    GPIO.setwarnings(False)
    GPIO.setup(self.GPIO_ECHO, GPIO.IN)
    GPIO.output(self.GPIO_TRIGGER, False)
    time.sleep(2)  # Permite senzorului să se stabilizeze

  def read_distance(self):
    try:
      speedSound = 34300  # Viteza sunetului în cm/s la temperatura camerei

      # Activează senzorul
      GPIO.output(self.GPIO_TRIGGER, True)
      time.sleep(0.00001)
      GPIO.output(self.GPIO_TRIGGER, False)

      start = time.time()
      stop = time.time()

      # Așteaptă începutul ecoului
      while GPIO.input(self.GPIO_ECHO) == 0:
        start = time.time()
        if time.time() - start > 0.03:  # Timeout pentru a preveni bucla infinită
          raise RuntimeError("Timeout la citirea senzorului ultrasonic (start)")

      # Așteaptă sfârșitul ecoului
      while GPIO.input(self.GPIO_ECHO) == 1:
        stop = time.time()
        if time.time() - start > 0.03:  # Timeout pentru a preveni bucla infinită
          raise RuntimeError("Timeout la citirea senzorului ultrasonic (stop)")
```

```
            elapsed = stop - start
            distance = (elapsed * speedSound) / 2  # Împarte la 2 pentru a ține cont de călătoria de întoarcere
            return distance

        except RuntimeError as e:
            print(e)
            return None  # Returnează None în caz de eroare

    def read(self, num_samples=3):
        distances = []

        for _ in range(num_samples):
            distance = self.read_distance()
            if distance is not None:
                distances.append(distance)
            time.sleep(0.02)  # Scurtă întârziere între mostre

        if len(distances) == 0:
            return None

        median_distance = statistics.median(distances)

        # Filtrează citirile care sunt prea departe de mediană
        filtered_distances = [d for d in distances if abs(d - median_distance) < median_distance * 0.1]

        if len(filtered_distances) == 0:
            return None

        average_distance = sum(filtered_distances) / len(filtered_distances)
        return average_distance

    def cleanup(self):
        # Curăță configurările GPIO
        GPIO.cleanup()

# Exemplu de utilizare dacă acest script este rulat direct
if __name__ == "__main__":
    try:
        sensor = UltrasonicSensor(trigger_pin=4, echo_pin=17)
        while True:
```

```
        distance = sensor.read()
        if distance is not None:
            print("Distance:", distance, "cm")
        else:
            print("Failed to read distance")
        time.sleep(0.2)  # Reduce întârzierea între citiri la 0.2 secunde
except KeyboardInterrupt:
    sensor.cleanup()
```

ANEXA 3. IMU

```
import smbus
import math
import time
import numpy as np

class MPU6050:
    def __init__(self, bus_number=1, device_address=0x68):
        self.bus = smbus.SMBus(bus_number)
        self.Device_Address = device_address
        self.ACCEL_SCALE_FACTOR = 16384.0
        self.GYRO_SCALE_FACTOR = 131.0
        self.Axoffset = 0.00498
        self.Ayoffset = 0.03055
        self.Azoffset = 0.0029
        self.Gx_offset = -0.19
        self.Gy_offset = -0.09
        self.Gz_offset = 0.15

        # Inițializarea matricilor și vectorilor pentru filtrul Kalman
        self.P_pitch = np.array([[0, 0], [0, 0]], dtype=float)
        self.K_pitch = np.array([0, 0], dtype=float)
        self.P_roll = np.array([[0, 0], [0, 0]], dtype=float)
        self.K_roll = np.array([0, 0], dtype=float)

        # Parametrii pentru filtrul Kalman
        self.Q_unghi_pitch = 0.001
        self.Q_gyro_pitch = 0.003
        self.R_unghi_pitch = 0.03
        self.Q_unghi_roll = 0.001
        self.Q_gyro_roll = 0.003
        self.R_unghi_roll = 0.03
        self.dt = 0.0065

        # Inițializează senzorul MPU6050
        self.MPU_Init()

    def MPU_Init(self):
        # Configurarea inițială a senzorului MPU6050
        self.bus.write_byte_data(self.Device_Address, 0x6B, 1)  # Scoate MPU6050 din
modul de așteptare
```

```
        self.bus.write_byte_data(self.Device_Address, 0x1A, 0)  # Setează banda de trecere a filtrelor DLPF
        self.bus.write_byte_data(self.Device_Address, 0x1B, 24)  # Setează scala giroscopului la ±2000°/s
        self.bus.write_byte_data(self.Device_Address, 0x38, 1)  # Activează întreruperile de date
        self.bus.write_byte_data(self.Device_Address, 26, 5)  # Setează frecvența de eșantionare

    def read_raw_data(self, addr):
        # Citește datele brute de la adresa specificată
        high = self.bus.read_byte_data(self.Device_Address, addr)
        low = self.bus.read_byte_data(self.Device_Address, addr + 1)
        value = ((high << 8) | low)
        if value > 32768:
            value -= 65536
        return value

    def kalman_filter(self, unghi, P, K, Q_unghi, Q_gyro, R_unghi, rata_gyro):
        # Aplică filtrul Kalman pentru estimarea unghiului
        unghi += self.dt * rata_gyro
        P[0][0] += self.dt * (self.dt * P[1][1] - P[0][1] - P[1][0] + Q_unghi)
        P[0][1] -= self.dt * P[1][1]
        P[1][0] -= self.dt * P[1][1]
        P[1][1] += Q_gyro * self.dt
        y = unghi - unghi
        S = P[0][0] + R_unghi
        K[0] = P[0][0] / S
        K[1] = P[1][0] / S
        unghi += K[0] * y
        rata_gyro -= unghi
        unghi += K[1] * rata_gyro
        P00_temp = P[0][0]
        P01_temp = P[0][1]
        P[0][0] -= K[0] * P00_temp
        P[0][1] -= K[0] * P01_temp
        P[1][0] -= K[1] * P00_temp
        P[1][1] -= K[1] * P01_temp
        return unghi

    def get_pitch_roll(self):
        # Citește valorile brute de la accelerometru și giroscop
        acc_x = (self.read_raw_data(0x3B) / self.ACCEL_SCALE_FACTOR) - self.Axoffset
```

```
            acc_y = (self.read_raw_data(0x3D) / self.ACCEL_SCALE_FACTOR) -
self.Ayoffset
        acc_z = (self.read_raw_data(0x3F) / self.ACCEL_SCALE_FACTOR) - self.Azoffset
            gyro_x = (self.read_raw_data(0x43) / self.GYRO_SCALE_FACTOR) -
self.Gx_offset
            gyro_y = (self.read_raw_data(0x45) / self.GYRO_SCALE_FACTOR) -
self.Gy_offset
            gyro_z = (self.read_raw_data(0x47) / self.GYRO_SCALE_FACTOR) -
self.Gz_offset
        # Calculează unghiurile de înclinare folosind datele accelerometrului
        pitch = math.degrees(math.atan2(acc_y, math.sqrt(acc_x ** 2 + acc_z ** 2)))
        roll = math.degrees(math.atan2(-acc_x, math.sqrt(acc_y ** 2 + acc_z ** 2)))
        # Aplică filtrul Kalman pentru a obține unghiurile precise
            unghi_pitch = self.kalman_filter(pitch, self.P_pitch, self.K_pitch,
self.Q_unghi_pitch, self.Q_gyro_pitch, self.R_unghi_pitch, gyro_x)
          unghi_roll = self.kalman_filter(roll, self.P_roll, self.K_roll, self.Q_unghi_roll,
self.Q_gyro_roll, self.R_unghi_roll, gyro_y)

        return unghi_pitch, unghi_roll, acc_x, acc_y, acc_z

    def vertical_velocity(self, Pitch, Roll, Ax, Ay, Az, ZVelocity, Time):
        # Calculează viteza verticală folosind unghiurile de înclinare și accelerațiile
         AccZInertial = -Ax * math.sin(Pitch * math.pi / 180) + Ay * math.cos(Pitch *
math.pi / 180) * math.sin(Roll * math.pi / 180) + Az * math.cos(Pitch * math.pi / 180) *
math.cos(Roll * math.pi / 180)
        AccZInertial = (AccZInertial - 1) * 9.81 * 10
        ZVelocity = ZVelocity + AccZInertial * Time
        return ZVelocity

# Exemplu de utilizare dacă acest script este rulat direct
if __name__ == "__main__":
    mpu = MPU6050()
    z_velocity = 0
    try:
        while True:
            pitch, roll, acc_x, acc_y, acc_z = mpu.get_pitch_roll()
             z_velocity = mpu.vertical_velocity(pitch, roll, acc_x, acc_y, acc_z, z_velocity,
mpu.dt)
                print(f"MPU6050 - Pitch: {pitch:.2f}, Roll: {roll:.2f}, Z Velocity:
{z_velocity:.2f}")
            time.sleep(0.2)  # Ajustează intervalul de timp între citiri
        except KeyboardInterrupt:
            print("Iesire...")
```

ANEXA 4. PROTOCOLUL IBUS

```
import serial
import time
import logging

# Configurare logging
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

class RadioController:
  def __init__(self, serial_port="/dev/serial0", baudrate=115200):
    self.ser = serial.Serial(serial_port, baudrate)
    self.ser.parity = serial.PARITY_NONE
    self.ser.stopbits = serial.STOPBITS_ONE
    self.Reference_altitude = 0
    self.ch1, self.ch2, self.ch3, self.ch4, self.ch5, self.ch6 = 0, 0, 0, 0, 0, 0

  def read_data(self):
    try:
      frame = bytearray()
      while self.ser.in_waiting > 0:
        received_data = self.ser.read()
        frame.extend(received_data)

      if len(frame) >= 32 and frame[0] == 32:
                self.ch1, self.ch2, self.ch3, self.ch4, self.ch5, self.ch6 = self._process_frame(frame)

      self._update_reference_altitude()

    except Exception as e:
      logging.error(f"Error in read_data: {e}")

  def _process_frame(self, frame):
    ch1 = int.from_bytes(frame[2:4], byteorder='little')
    ch2 = int.from_bytes(frame[4:6], byteorder='little')
    ch3 = int.from_bytes(frame[6:8], byteorder='little')
    ch4 = int.from_bytes(frame[8:10], byteorder='little')
    ch5 = int.from_bytes(frame[10:12], byteorder='little')
    ch6 = int.from_bytes(frame[12:14], byteorder='little')
    ch3 = (ch3 - 1000) / 10  # Ajustează ch3 conform aplicației
```

```
            return ch1, ch2, ch3, ch4, ch5, ch6

        def _update_reference_altitude(self):
            if self.ch3 < 10:
                self.Reference_altitude = max(0, self.Reference_altitude - 5)
            elif 10 < self.ch3 < 20:
                self.Reference_altitude = max(0, self.Reference_altitude - 0.5)
            elif 20 < self.ch3 < 40:
                self.Reference_altitude = max(0, self.Reference_altitude - 0.05)
            elif 60 < self.ch3 < 80:
                self.Reference_altitude += 0.05
            elif 80 < self.ch3 < 90:
                self.Reference_altitude += 0.5
            elif self.ch3 >= 90:
                self.Reference_altitude += 5

        def get_reference_altitude(self):
            return self.Reference_altitude

        def get_channels_data(self):
            return self.ch1, self.ch2, self.ch3, self.ch4, self.ch5, self.ch6

        def cleanup(self):
            self.ser.close()

    if __name__ == "__main__":
        try:
            radio = RadioController()
            while True:
                radio.read_data()
                reference_altitude = radio.get_reference_altitude()
                ch1, ch2, ch3, ch4, ch5, ch6 = radio.get_channels_data()
                    logging.debug(f"Channels: ch1={ch1}, ch2={ch2}, ch3={ch3}, ch4={ch4},
ch5={ch5}, ch6={ch6}")
                logging.debug(f"Reference Altitude: {reference_altitude}")
                time.sleep(0)  # Pauză mică pentru a nu supraîncărca procesorul
        except KeyboardInterrupt:
            radio.cleanup()
            logging.info("Exiting...")
        except Exception as e:
            logging.error(f"Unhandled exception: {e}")

            radio.cleanup()
```

ANEXA 5. PROGRAMUL PRINCIPAL

```
import multiprocessing as mp
import time
import struct
from gpiozero import DistanceSensor
from MPU6050 import MPU6050
from BMP280 import BMP280, BMP280Calibrator
from Ibus import RadioController
from Ultrasonic import UltrasonicSensor
from simple_pid import PID

def read_ultrasonic(queue):
    sensor = UltrasonicSensor(trigger_pin=4, echo_pin=17)
    while True:
        distance = sensor.read()
        queue.put(('ultrasonic', distance))
        time.sleep(0.02)  # Reducerea întârzierii pentru citiri mai rapide

def read_imu_and_bmp(queue):
    mpu = MPU6050()
    bmp280 = BMP280()
    calibrator = BMP280Calibrator(bmp280)
    baseline_altitude = calibrator.calculate_altitude_offset()
    z_velocity = 0
    while True:
        pitch, roll, acc_x, acc_y, acc_z = mpu.get_pitch_roll()
        z_velocity = mpu.vertical_velocity(pitch, roll, acc_x, acc_y, acc_z, z_velocity,
mpu.dt)
        temperature, pressure, altitude = bmp280.read_sensor_data()
        altitude -= baseline_altitude
        altitude = max(altitude, 0)
        queue.put(('imu_bmp', (pitch, roll, z_velocity, altitude)))
        time.sleep(0.02)  # Reducerea întârzierii pentru citiri mai rapide

def read_radio(queue):
    radio = RadioController()
    while True:
        radio.read_data()
        reference_altitude = radio.get_reference_altitude()
        ch1, ch2, ch3, ch4, ch5, ch6 = radio.get_channels_data()
        queue.put(('radio', ( ch1, ch2, ch3, ch4, ch5, ch6)))
        time.sleep(0.02)  # Reducerea întârzierii pentru citiri mai rapide
```

```
def send_ibus_packet(radio, channels):
    # Structura pachetului IBUS
    header = b'\x20\x40'
    channel_data = struct.pack('<14H', *channels)
    checksum = struct.pack('<H', 0xFFFF - (sum(channel_data) & 0xFFFF))

    packet = header + channel_data + checksum
    radio.send_packet(packet)

def process_data(queue):
    LastTime = time.time()
    data_dict = {
        'ultrasonic': None,
        'imu_bmp': None,
        'radio': None
    }

    # Inițializarea controlerului PID
    pid = PID(1.0, 0.1, 0.05, setpoint=0)  # Câștigurile PID trebuie ajustate
    pid.output_limits = (0, 100)  # Ieșirea trebuie să fie între 0 și 100

    # Inițializarea RadioController pentru comunicarea iBus
    radio = RadioController()

    while True:
        dateDateTimeNow = time.time()
        Time = dateDateTimeNow - LastTime

        try:
            sensor_type, data = queue.get(timeout=0.02)  # Get non-blocant cu timeout
            data_dict[sensor_type] = data
        except mp.queues.Empty:
            continue

        if all(value is not None for value in data_dict.values()):
            distance = data_dict['ultrasonic']
            pitch, roll, z_velocity, altitude = data_dict['imu_bmp']
            reference_altitude, ch1, ch2, ch3, ch4, ch5, ch6 = data_dict['radio']

            # Calcularea erorii între altitudinea curentă și altitudinea de referință
            altitude_error = reference_altitude - altitude
```

```
            # Folosirea controlerului PID pentru a calcula ajustarea pentru canalul 3
            ch3_adjustment = pid(altitude_error)

            # Actualizarea canalului 3 în intervalul 0-100
            ch3 = max(0, min(100, ch3_adjustment))

            # Crearea listei de canale, completând canalele neutilizate cu valori implicite
            channels = [ch1, ch2, int(ch3), ch4, ch5, ch6] + [1500] * 8

            # Trimiterea valorilor actualizate ale canalelor prin iBus
            send_ibus_packet(radio, channels)

            # Afișarea valorilor actualizate ale canalelor
              print(f"Channels: ch1={ch1}, ch2={ch2}, ch3={ch3}, ch4={ch4}, ch5={ch5},
ch6={ch6}")
             print(f"Reference Altitude: {reference_altitude}, Current Altitude: {altitude},
Altitude Error: {altitude_error}")

          LastTime = dateDateTimeNow

    if __name__ == "__main__":
        queue = mp.Queue()

        processes = [
            mp.Process(target=read_ultrasonic, args=(queue,)),
            mp.Process(target=read_imu_and_bmp, args=(queue,)),
            mp.Process(target=read_radio, args=(queue,)),
            mp.Process(target=process_data, args=(queue,))
        ]

        for p in processes:
            p.start()

        for p in processes:
            p.join()
```