

SISTEM ROBOTIC AERIAN AUTONOM. ALGORITM DE URMĂRIRE
A UNUI OBIECT ȚINTĂ MOBIL

PROIECT DE DIPLOMĂ

Prezentat ca cerință parțială pentru obținerea titlului de
Inginer în domeniul Electronică și Telecomunicații
Programul de studii Microelectronică, Optoelectronică și Nanotehnologii

Coordonatori ai lucrării:

Ș.L. Dr.Ing. Horia CUCU

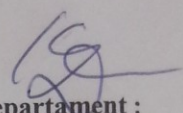
Prof. Dr.Ing. Corneliu BURILEANU

Student:

Iulian Florin BĂLAN

București
2016

Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației
Departamentul Dispozitive, Circuite și Arhitecturi Electronice


Aprobat Director de Departament :
Prof. Dr. Ing. Gheorghe Ștefan

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Bălan Florin Iulian, grupa 443E

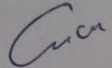
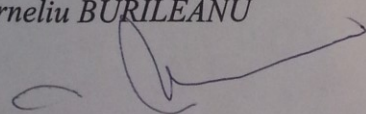
1. Titlul temei: *Sistem robotic aerian autonom. Algoritm de urmărire a unui obiect țintă mobil.*
2. Contribuția practică, originală a studentului va consta în (în afara părții de documentare):

Proiectul are ca scop programarea unui quadcopter care să urmărească o țintă mobilă.

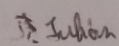
Etape ale proiectului:

 - Realizarea unei comunicații între microcontrolerul responsabil de menținerea stabilității și cel responsabil de autonomia robotului.
 - Simularea software a unei telecomenzi pentru rutina de calibrare, testele preliminare și controlul quadcopterului.
 - Dezvoltarea unui algoritm de urmărire a unui obiect țintă. Întoarcerea în punctul de plecare după atingerea obiectivului sau ca rutină de siguranță.
3. Proiectul se bazează pe cunoștințe dobândite în principal la următoarele 3-4 discipline: *Microcontrolere, Programarea Calculatoarelor, Prelucrarea digitală a semnalelor, Sisteme de comunicație.*
4. Proprietatea intelectuală asupra proiectului aparține: *Laborator de Cercetare Speech and Dialogue (Speed), studentului Bălan Florin Iulian.*
5. Locul de desfășurare a activității: *UPB-ETTI, Sala 3 de Calculatoare,*
6. Realizarea practică rămâne în proprietatea: *Laborator de Cercetare Speech and Dialogue (Speed), studentului Bălan Florin Iulian*
7. Data eliberării temei: *Septembrie 2015*

CONDUCĂTOR LUCRARE:

S.L. Dr. Ing. Horia CUCU 
Prof. Dr. Ing. Corneliu BURILEANU 

STUDENT:

BĂLAN Florin Iulian 

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul “*Sistem robotic aerian autonom. Algoritm de urmărire a unui obiect țintă mobil*”, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității “Politehnica” din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică, Telecomunicații și Tehnologia Informației* programul de studii *Microelectronica, optoelectronică și nanotehnologii* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurărilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 27.06.2016

Absolvent *Florin-Iulian BĂLAN*

(semnătura în original)

CUPRINS

Cuprins	7
Listă de figuri	9
Listă de acronime	11
Capitolul 1 Introducere	13
1.1 Motivația lucrării	13
1.2 Echipament utilizat	14
1.3 Planificare	15
1.4 Descriere succintă a etapelor parcurse	15
Capitolul 2 Noțiuni teoretice.....	17
2.1 Noțiuni hardware.....	17
2.1.1 Funcționarea quadcopterului.....	17
2.1.2 Motoare brushless	20
2.1.3 Electronic speed control.....	20
2.1.4 Microcontrollerul Hoverfly.....	22
2.1.5 Bateria LiPo	23
2.1.6 Placa de dezvoltare Arduino Uno	24
2.1.7 Placa de dezvoltare Raspberry Pi3.....	24
2.1.8 Modulele XBEE.....	26
2.2 Noțiuni software.....	26

2.2.1	Limbajul Arduino	26
2.2.2	XCTU și CoolTherm	27
2.2.3	Sistemul de operare Linux	28
2.2.4	Semnalele PWM și PPM	29
Capitolul 3	Design și dezvoltare.....	31
3.1	Calibrarea ESC-urilor	32
3.2	Simularea unei telecomenzi software	33
3.3	Simularea software a rutinei de armare a dronei	34
3.4	Teste incipiente efectuate în incinta laboratorului.....	34
3.5	Realizarea unei comunicații wireless între quadcopter și laptop.....	36
3.6	Pregatirea quadcopterului pentru testele de zbor.....	37
3.7	Testele de zbor efectuate în exterior	38
3.8	Schimbarea platformei Arduino cu cea Raspberry	39
3.9.	Algoritm de urmărire a unei ținte mobile	40
Capitolul 4	Concluzii și dezvoltări viitoare	45
4.1	Concluzii.....	45
4.2	Dezvoltări viitoare	46
Bibliografie		47
Anexe		49
Anexa 1		49
Anexa 2		50
Anexa 3		53
Anexa 4		53

LISTĂ DE FIGURI

Figura 1 Quadcopter ELEV8 v2	14
Figura 2 Traseul informației în cazul folosirii unui receiver și al unei telecomenzi	15
Figura 3 Sensul de rotație a celor 4 elice ale quadcopterului	18
Figura 4 Cele trei direcții și unghiuri după care se deplasează quadcopterul	18
Figura 5 Semnificația mișcărilor stickurilor de control	19
Figura 6 Motor brushed și brushless	20
Figura 7 Schema circuitului de alimentare	21
Figura 8 Electronic Speed Controller	22
Figura 9 Microcontroller de zbor HoverflyOpen	22
Figura 10 Conectarea canalelor de control la HoverflyOpen	23
Figura 11 Baterie Lipo3s	23
Figura 12 Placă Arduino Uno	24
Figura 13 Placă Raspberry Pi3	25
Figura 14 Scanare de IP-uri în Zenmap	25
Figura 15 Modul XBee s2	26
Figura 16 Parametrii XBee-ului citiți prin XCTU	28
Figura 17 Semnal PWM	29
Figura 18 Semnale PPM și PWM	30
Figura 19 Diagramă reprezentând ultima iterație a sistemului cu Arduino	32

Figura 20 Simularea stării de ThrottleLow_RudderLow.....	33
Figura 21 Semnale folosite în testare.....	35
Figura 22 Schema PCB în Proteus.....	37
Figura 23 Layoutul schemei PCB produs de Ares.....	38
Figura 24 Diagrama sistemului de test în aer liber folosind module XBee.....	39
Figura 25 Algoritmul de urmărire a obiectului.....	41
Figura 26 Identificarea colțului stânga sus al imaginii.....	42

LISTĂ DE ACRONIME

BEC = Battery Eliminator Circuit
CH = Channel
DH = Data High
DL = Data Low
ESC = Electronic Speed Controller
GND = Ground
GPIO = General Purpose Input Output
GPS = Global Positioning System
IDE = Integrated Development Environment
IP = Internet Protocol
LED = Light Emitting Diode
LiPo = Lithium Polymer
PAN ID = Personal Area Network Identifier
PCB = Printed Circuit Board
PPM = Pulse Position Modulation
PWM = Pulse Width Modulation
RISC = Reduced Instruction Set Machine
RPM = Revolutions per Minute
RSSI = Receiver Signal Strength Indicator
Rx = Receiver
SSID = Service Set Identifier
TDM = Time Division Multiplexing
Tx = Transmitter
uC = Microcontroller
VTOL = Vertical Take Off and Landing
USB = Universal Serial Bus

WiFi = Wireless Fidelity

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIA LUCRĂRII

Conceptul de quadcopter nu este unul nou, istoria acestora începând încă din anul 1907 când francezii Jacques și Louis Breguet au construit și testat Gyroplane No. 1, un quadcopter. Au reușit să îl facă să decoleze, însă designul s-a dovedit impractic, deci instabil în zbor. Motivul pentru care designerii timpurii au experimentat cu quadcoptere este acela că alternativa de a folosi un singur rotor principal și un rotor de coadă pentru contrabalansarea cuplului creat de rotorul principal părea risipitoare, complexă și ineficientă. Acest rotor de coadă consumă între 10 și 15% din puterea motorului fără a avea rol în înălțarea propriu zisă. De asemenea, s-a constatat că o parte din aerul împins în jos de rotorul principal al elicopterelor ajunge pe fuselaj, reducând eficiența.

Multicopterele sunt aeronave ce au mai mult de 2 rotoare (rotor = parte a unei mașini care se rotește în jurul axului pe care este montată și care se folosește pentru reluarea și transmiterea unui cuplu motor). În ciuda avantajelor precizate anterior, quadcopterele au și dezavantaje, ce au făcut ca acestea să nu fie folosite la scară largă decât recent. Având patru rotoare pentru propulsie, un quadcopter nu poate fi făcut stabil în zbor doar prin rotirea elicelor la aceeași viteză. Din contră, acestea trebuie stabilizate permanent. Aceasta presupune o sarcină practic imposibilă pentru pilot, în absența unui computer de zbor.

Progresele relativ recente în domeniul motoarelor electrice și al microelectronicii au făcut ca dronele de tip multicopter să devină foarte populare. De asemenea, s-a început scalarea mărimii acestora pentru a permite transportul oamenilor.



Figura 1 Quadcopter ELEV8 v2

Prezenta lucrare de diplomă abordează robotica aeriană, un domeniu care m-a fascinat dintotdeauna. Aflând de posibilitatea de a studia acest domeniu în cadrul laboratorului Speed, am început să mă documentez despre quadcopterul ELEV-8 v2. După scurt timp, am ales și o direcție pentru demararea proiectului de licență. Tema aleasă este dezvoltarea unui algoritm de urmărire a unui obiect țintă mobil. Am ales această temă deoarece mi s-a părut deopotrivă provocatoare ca dificultate, cât și atrăgătoare. Spre deosebire de alte kituri robotice făcute pentru a fi programate să îndeplinească diverse sarcini, cel de quadcopter a fost făcut cu scopul de a fi telecomandat. De aceea, ideea de a folosi acest kit altfel decât a fost proiectat să fie folosit mi s-a părut cu atât mai provocatoare. De asemenea, aplicațiile și dezvoltările viitoare ale produsului final mi se par extrem de utile într-o mare diversitate de domenii, precum misiunile de recunoaștere, de cartografiere, sau de transport în medii periculoase pentru om. Acestea mi-au stârnit curiozitatea și au făcut ca atingerea fiecărui obiectiv să fie satisfăcătoare.

1.2 ECHIPAMENT UTILIZAT

Cel mai important dintre echipamentele folosite în această lucrare este quadcopterul ELEV-8 v2, un multicopter complet asamblat, dotat cu patru motoare de tip brushless, patru ESC-uri pentru controlul motoarelor, un microcontroller Hoverfly Open cu un algoritm de stabilizare în zbor HoverCore preinstalat.

Pentru a trimite comenzi quadcopterului, a fost folosit mai întâi un microcontroller Arduino Uno ce comunică cu Hoverfly Open prin fire sau, ulterior, prin module de comunicație wireless Xbee. Din cauza inconvenientelor folosirii modulelor Xbee, Arduino a fost înlocuit cu un microcontroller de tip Raspberry Pi 3, cu modul integrat de wireless și bluetooth. Pentru a alimenta sistemul a fost folosită o baterie de tip LiPo 3s. Detalii despre funcționarea acestor

echipamente și felul în care au fost utilizate în proiect sunt oferite în capitolul de noțiuni teoretice.

Deasemenea, pentru realizarea legăturilor la motoare, ESC-uri, dar și în cazul celor de la PCB-ul realizat pentru sporirea rezistenței mecanice a sistemului, a fost folosit ledconul, fludorul și perforatorul.

1.3 PLANIFICARE

Prezenta lucrare are un prim capitol introductiv ce prezintă un scurt istoric al quadcoptereleor, abordează motivația acestei lucrări și enumeră echipamentele de la care s-a pornit, asupra cărora nu am intervenit în niciun fel.

Al doilea capitol se constituie ca un breviar teoretic atât pentru partea hardware cât și cea software a proiectului. Aici este descrisă funcționarea tuturor elementelor constitutive ale quadcopterului și modul general de funcționare al acestuia. Tot în acest capitol sunt prezentate succint și programele folosite pentru configurarea modulelor de comunicație radiofrecvență (XBEE), particularitățile limbajului Arduino (o variantă de C++ creată pentru simplificarea folosirii resurselor hardware), și folosirea sistemului de operare linux de pe Raspberry Pi.

Al treilea capitol descrie demersurile făcute pentru a obține configurația finală propusă. Sunt enumerate toate etapele parcurse în ordine cronologică, precum și dificultățile ce au survenit pe parcurs și rezolvările găsite pentru acestea.

În al patrulea capitol sunt prezentate concluziile trase în urma realizării lucrării și sunt descriși eventuali pași către perfecționarea sau dezvoltarea proiectului.

1.4 DESCRIERE SUCCINTĂ A ETAPELOR PARCURSE

Prima etapă a proiectului presupune înlocuirea telecomenzii și a transmitterului cu microcontrollerul Arduino pe care rulează un program de simulare a funcționalității componentelor înlocuite. Acest program este modificat pentru a realiza sarcini a căror dificultate crește progresiv, sarcina finală fiind urmărirea unui obiect folosind informațiile primite de la o cameră video, cu privire la poziția unui obiect mobil în raport cu quadcopterul.

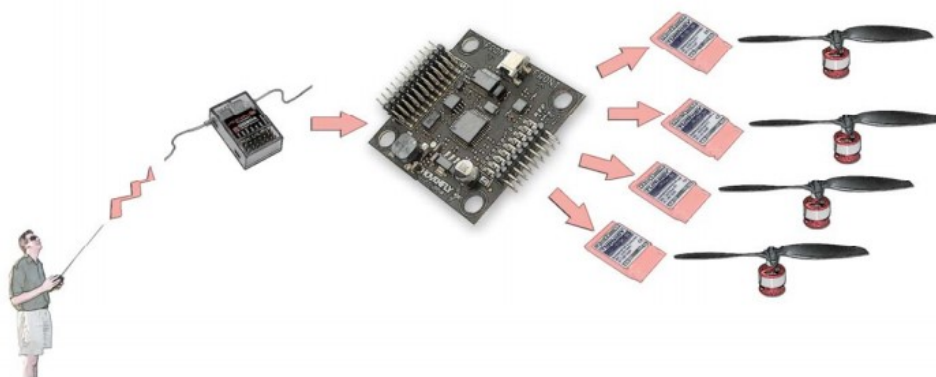


Figura 2 Traseul informației de control în cazul folosirii unui receiver și al unei telecomenzi

O etapă intermediară importantă a proiectului este realizarea comunicației fără fire între Arduino și HoverflyOpen (microcontrollerul responsabil de stabilizarea în zbor a quadcopterului). Codul de pe cel din urmă este indisponibil din motive de păstrare a proprietății intelectuale și se va trata în lucrare ca un „black box” ce are ca intrare semnale de control de la Arduino și ca ieșire un semnal în trei faze ce ajunge la ESC-uri. De la un anumit punct în lucrare, va fi folosit Raspberry Pi3 ca microcontroller de comandă în locul Arduino Uno.

În final, pe Raspberry Pi3 va rula codul de adaptare a vitezei motoarelor pentru urmărirea unui obiect detectat cu o cameră video.

CAPITOLUL 2

NOȚIUNI TEORETICE

2.1 NOȚIUNI HARDWARE

2.1.1 Funcționarea quadcopterului

Avantajul major al aeronavelor de tip VTOL (Vertical Take Off and Landing), dintre care face parte și quadcopterul, față de aeronavele obișnuite, este manevrabilitatea.

Quadcopterul este format din două tije perpendiculare la capătul cărora se află motoarele. În zona centrală sunt situate toate componentele electronice și bateria. Elicele sunt poziționate fiecare deasupra unui motor, rotirea acestora generând împingere. În același timp, aerul opune rezistență acestei mișcări de rotire, generând o forță în direcția opusă direcției de rotație a elicei. Motoarele aflate pe axe diferite se rotesc în direcții opuse pentru a contracara acest efect. Urmarea este că dacă vitezele de rotație ale celor patru elice sunt egale, forțele apărute în urma frecării cu aerul se anulează reciproc. Există două tipuri de configurație a quadcopterelor: X și +.

Quadcopterul din proiect este de tip X, avantajul acestuia față de cel de tip + fiind stabilitatea și accelerația mai mare, precum și o mai bună vizibilitate deoarece "fața" nu este obstrucționată de una din elice.

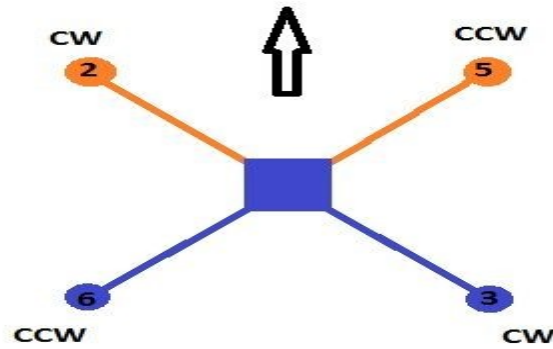


Figura 3 Sensul de rotație a celor 4 elice ale quadcopterului

Pentru a putea descrie mișcarea quadcopterului trebuie luată în considerare poziția sa absolută descrisă de un sistem de coordonate cu 3 axe (x, y, z), centrul fiind centrul de masă al dronei. Cele trei unghiuri ale lui Euler tangaj θ , girație ψ și ruliu ϕ (Pitch, Yaw și Roll așa cum vor fi numite în continuare) descriu rotația după fiecare dintre aceste trei axe. Deplasarea în jurul oricăreia dintre aceste axe sau mai multora concomitent schimbă „atitudinea” dronei. Rotația în jurul acestor axe poate fi controlată ajustând diferența de viteză între motoare în felul următor: pentru a schimba unghiul de Roll sau Pitch se variază viteza elicelor de-a lungul aceleiași axe, iar pentru a schimba unghiul de Yaw sunt folosite forțele generate în urma frecării cu aerul, menționate anterior.

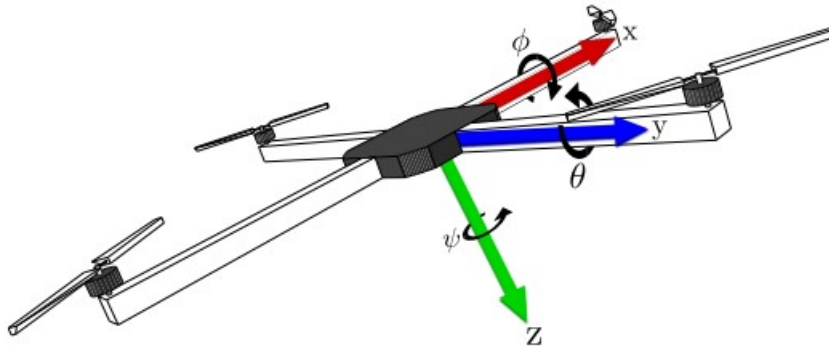


Figura 4 Cele trei direcții și unghiuri după care se deplasează quadcopterul

O dată ce atitudinea este menținută constantă, prin creșterea vitezei de rotație a tuturor elicelor poate fi crescută altitudinea dronei. În cazul controlului folosind o telecomandă, creșterea altitudinii și modificarea unghiului de Yaw se realizează mișcând joystickul stâng și modificarea unghiurilor de Roll sau Pitch se realizează mișcând joystickul drept. Aceste informații sunt necesare pentru simularea rutinei de armare a quadcopterului și pentru controlul acestuia în zbor.

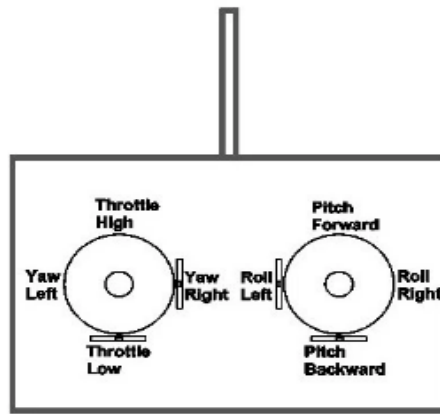


Figura 5 Semnificația mișcărilor stickurilor de control

Conform documentației quadcopterului ELEV-8, pentru a putea zbura cu acesta în siguranță, ESC-urile sale nu trimit la motoare semnale până nu este făcută procedura de armare. Există de asemenea și o procedură de dezarmare concepută cu același scop.

- Stickul de „Throttle” trebuie pus în poziția idle
- Trebuie conectată bateria
- În acest moment ESC-urile se vor inițializa în 5-10 secunde și vor da un semnal sonor
- Se va aprinde intermitent un LED verde pentru a semnaliza faptul că procedura de armare poate începe
- Trebuie mutat stickul de „Throttle” în colțul din stânga jos până se aude un beep
- Trebuie mutat stickul de „Throttle” în colțul din dreapta jos până se aud două beepuri.
- Ledul va fi apoi roșu intermitent timp de câteva secunde pentru a semnaliza faptul că microcontrollerul de zbor își calibrează senzorii interni (armarea trebuie făcută cât timp quadcopterul este nemișcat)
- Când armarea este completă, se va auzi un beep de trei ori, apoi LED-ul se va face verde

Procedura de dezarmare

- Stickul de „Throttle” trebuie pus în poziția idle
- Trebuie mutat stickul de „Throttle” în colțul din stânga jos până se aude un beep
- Trebuie mutat stickul de „Throttle” în colțul din dreapta jos până se aud două beepuri.
- Drona este dezarmată

Modul „Trouble”

Acest mod este activat atunci când unul dintre cele cinci canale (Throttle, Elevator, Aileron, Rudder, Gain) nu este detectat. Intrarea în acest mod este pusă în evidență de LED-ul ce se aprinde intermitent mov fără a începe procedura de armare.

Eșecul calibrării giroscopului

Așa cum am menționat la rutina de calibrare, după ce LED-ul verde se aprinde intermitent, drona își calibrează senzorii interni. În cazul eșecului acestor calibrări LED-ul se va aprinde mov intermitent. Diferența față de modul „Trouble” este aceea că în acest caz procedura de armare începe înaintea sesizării problemei.

2.1.2 Motoare brushless

Scopul motoarelor electrice este de a converti energia electrică în energie mecanică sub forma mișcării de rotație. Aceste sisteme electromecanice sunt caracterizate de anumiți parametri care ajută la precizarea performanțelor motorului. Viteza în absența sarcinii descrie viteza motorului când nu este cuplu aplicat, iar viteza de mers în gol descrie momentul de cuplu maxim al motorului când viteza acestuia este de 0 RPM (revolutions per minute). Alți parametri de interes sunt cuplul, curentul și tensiunea de alimentare a motorului. Modul de funcționare al acestor motoare este simplu, cuplul fiind creat de forța de magnetism apărută între magneți staționari și electromagneți alimentați astfel încât polaritatea dintre cele două componente să fie mereu inversă. Motoarele brushed, deși au avantajul unui preț mai mic, au electromagneții plasați pe rotor și magneții fixi pe stator, de aceea sunt predispuse supraîncălzirii, dar și uzurii din cauza comutatoarelor. La motoarele brushless, electromagneții sunt plasați pe stator și magneții staționari pe rotor. Polaritatea electromagneților este schimbată de data aceasta de un circuit de control care îmbunătățește precizia motorului.

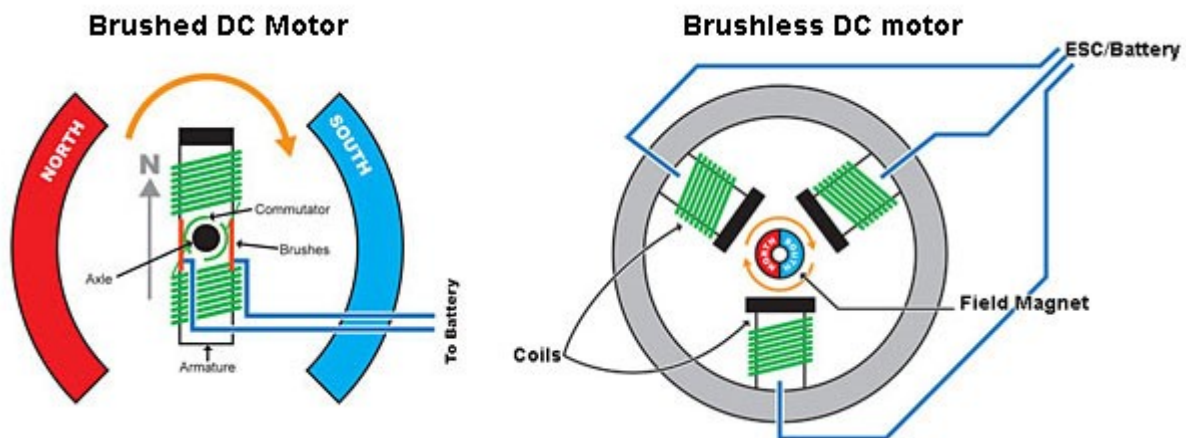


Figura 6 Motor brushed și brushless

$$\text{Rpm} = K_v * \text{volt}$$

Factorul K_v este cel mai folosit pentru a descrie capacitățile unui motor. După cum se observă, la aceeași tensiune este proporțional cu numărul de rotații pe minut (RPM). Tipul de motor trebuie adaptat tipului de quadcopter dorit. O dronă de mare performanță, acrobatică, va avea un motor de viteză mare. Pe de altă parte, un quadcopter folosit în zboruri lungi, trebuie să consume cât mai puțină energie, de aceea motoarele cu Rpm mic sunt de dorit în această situație.

2.1.3 Electronic speed control

Un Electronic Speed Control (ESC) este un circuit ce îndeplinește două funcții principale. Prima este de a acționa asemenea unui regulator de tensiune, reducând tensiunea furnizată de baterie până la valoarea necesară funcționării motoarelor și microcontrollerelor. A doua funcție este de a converti semnalul de la Hoverfly Open într-un semnal de control pentru motorul brushless.

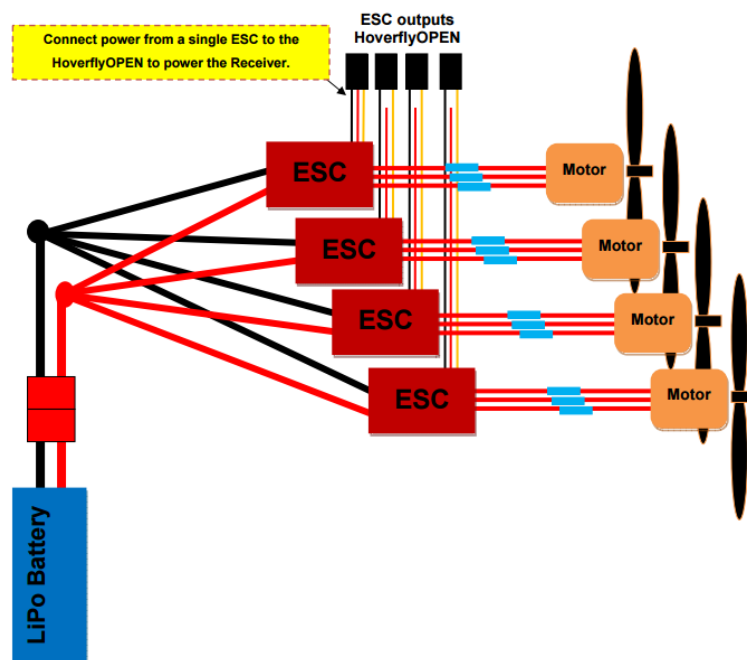


Figura 7 Schema circuitului de alimentare

Spre deosebire de speed controllerele bazate pe rezistori, ESC-urile nu funcționează aplicând motoarelor fracțiuni din tensiunea de alimentare proporționale cu viteza pe care o dorim, ci aplicând întreaga tensiune, dar oprind-o și pornind-o extrem de rapid. În cazul celor apărute primele, diferența de tensiune nefolosită pentru mișcarea motorului ar fi fost irosită, fiind disipată sub formă de căldură. Variind raportul dintre timpii de „ON” (tensiune maximă) și „OFF” (tensiune de 0V), ESC-ul variază de fapt tensiunea medie citită de motor. Cu alte cuvinte, la orice moment de timp, tensiunea de control este ori complet „ON” ori complet „OFF”. Aceasta face ca acest tip de control să fie, cel puțin teoretic, 100% eficient. În realitate totuși, ESC-urile sunt dispozitive imperfecte, eroarea acestora datorându-se în principal timpului de comutare al tranzistoarelor folosite în circuit și rezistențelor interne.

BEC (Battery Eliminator Circuit) este circuitul ce regulează tensiunea furnizată de baterie. Este extrem de folositor în aplicații de quadcopter, deoarece bateriile suplimentare ar crește greutatea dronei.

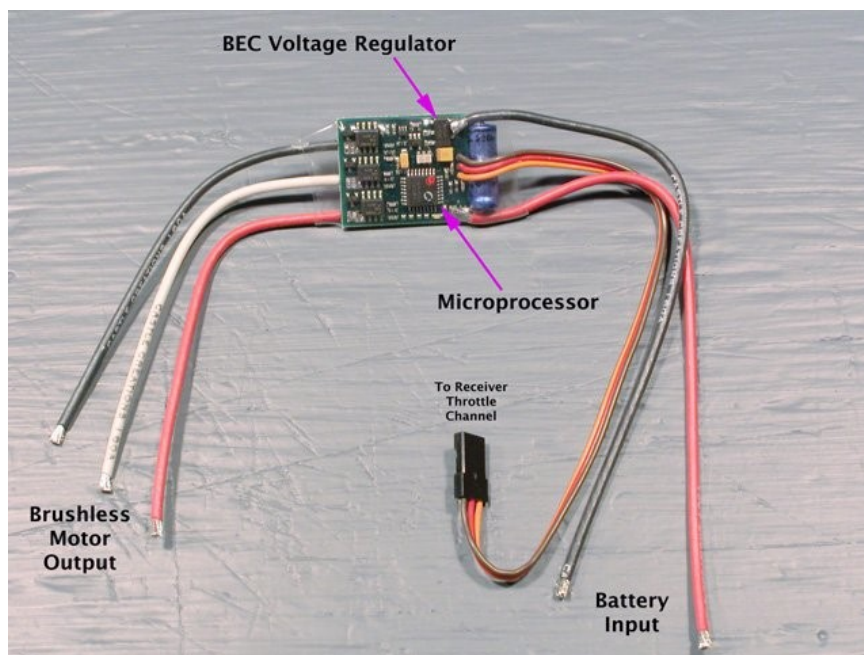


Figura 8 Electronic Speed Controller

2.1.4 Microcontrollerul Hoverfly

Acesta este microcontrollerul de zbor dezvoltat de firma Parallax, dotat cu accelerometru și giroscop pe trei axe. Așa cum am precizat anterior, pe acesta rulează algoritmul HoverCore ce utilizează o arhitectura de procesare în paralel pentru a colecta date de la senzori în mod independent. Acest algoritm generează apoi semnale Pulse Width Modulation (PWM) pentru a controla ESC-urile cu precizie.

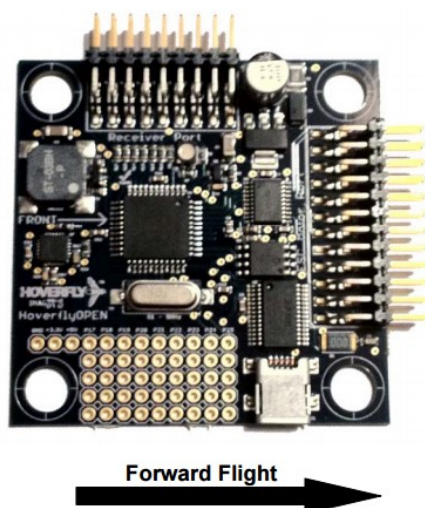


Figura 9 Microcontroller de zbor HoverflyOpen

Inputul acestui microcontroller este format din semnalele PWM primite pe cele cinci canale de control: Throttle, Rudder, Aileron, Elevator, Gear. Acestea corespund mișcărilor după anumite axe descrise în secțiunea despre funcționarea quadcopterului în felul următor: Throttle este

responsabil de modificarea altitudinii, Rudder modifică unghiul de Yaw, Aileron modifică unghiul de Roll și Elevator controlează Pitchingul.

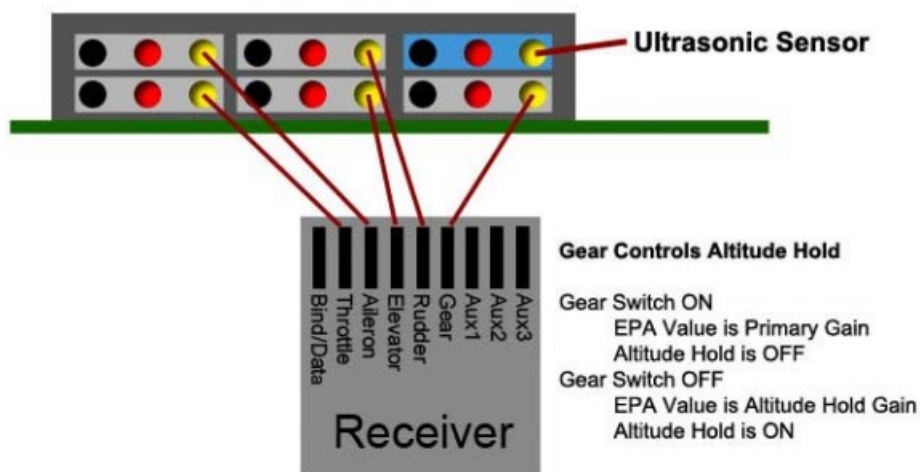


Figura 10 Conectarea canalelor de control la HoverflyOpen

2.1.5 Bateria LiPo

Bateriile LiPo (prescurtare de la Litiu-Polimer) sunt un tip de baterii reîncărcabile ce au permis echipamentelor RC (radio controlled) să prospere recent. Se poate considera că aceste baterii sunt motivul pentru care zborul aparatelor electrice este o opțiune viabilă față de modelele alimentate prin combustibili. Motivele pentru care aceste baterii sunt atât de folosite sunt :

- Greutatea mică și faptul pot avea orice formă sau mărime
- Posibilitatea de a stoca multă energie într-un pachet de mici dimensiuni
- Rată mare de descărcare pentru motoare electrice cu consum mare

Bateriile LiPo nu folosesc o soluție electrolitică lichidă ci un polimer electrolitic separator între anodul și catodul bateriei, ce permite schimbul ionilor de litiu.

Principalii parametri de interes ai acestei baterii sunt tensiunea și capacitatea. Bateriile LiPo au o tensiune nominală de 3.7 volți per celulă. Bateria folosită în proiect este de tip LiPo 3s, aceasta însemnând că are trei celule în serie, deci o tensiune totală de 11.1 volți. Capacitatea indică câtă energie poate fi stocată în baterie și este indicată în mAh (miliamperi * oră). Acesta este un fel de a indica cât de multă sarcină (măsurată în miliAmperi) poate fi conectată la baterie pentru a o descărca complet într-o ora. Bateria folosită în proiect consumă 3300 mAh.



Figura 11 Bateria Lipo3s

2.1.6 Placă de dezvoltare Arduino Uno

Uno este în esență o placă de dezvoltare bazată pe microcontrollerul Atmega 328P, un microcontroller construit cu arhitectura RISC, programabil și dispunând de 32 regiștri de uz general pe 8 biți. De asemenea, dispune de 32 kilo octeți de memorie Flash din care 0.5 KB sunt rezervați pentru Bootloader. De interes pentru proiect sunt pinii de intrare/ ieșire digitală ce sunt folosiți pentru generarea semnalelor PWM trimise la microcontrollerul HoverFly Open. Arduino Uno este alimentat cu 5 volți de la regulatorul de tensiune disponibil pe fiecare dintre cele patru ESC-uri. Arduino trebuie conectat la Laptop cu un cablu USB pentru a putea fi programat, iar mediul de programare se numește tot Arduino (IDE), fiind furnizat de aceiași producători.

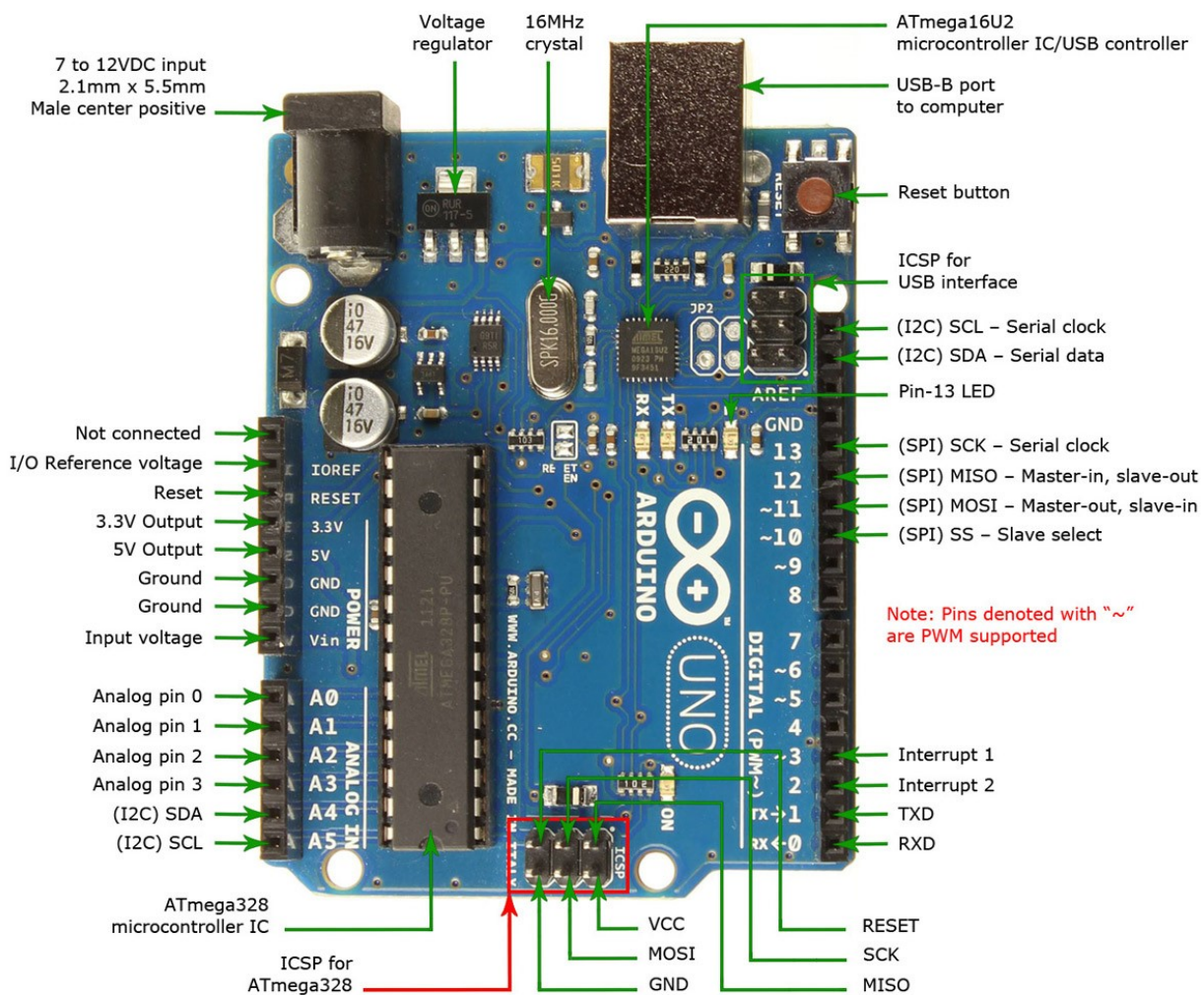


Figura 12 Placă Arduino Uno

2.1.7 Placă de dezvoltare Raspberry Pi3

Această placă de dezvoltare este similară cu cea Arduino Uno dar oferă funcționalități suplimentare precum conectivitate wireless, bluetooth, interfață pentru cameră video, slot micro SD, interfață Ethernet. Pe acest microcontroller se poate rula un sistem de operare asemănător cu

Linux, pe care pot fi instalate compilere de cod pentru diverse limbaje. Pentru a reduce diferența între codul de pe Arduino și cel de pe Raspberry, am ales un compiler pentru limbajul C++.

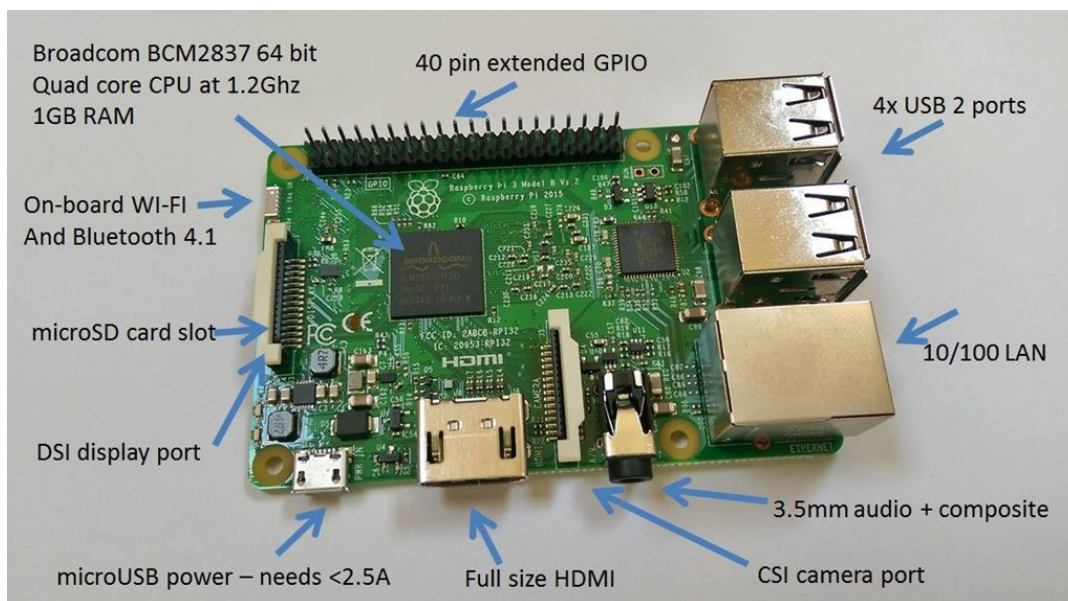


Figura 13 Placă Raspberry Pi3

Pentru a putea folosi plăcuța, mai întâi trebuie instalat un sistem de operare pe aceasta. Sistemul de operare ales se numește Raspbian Jessie. Sistemul, disponibil pe site-ul web al producătorilor Raspberry Pi3 este încărcat pe cardul SD. Apoi, cardul este introdus în placă și aceasta este alimentată prin portul microUSB. Pentru a putea folosi sistemul de operare trebuie să se facă conexiunea la Raspberry cu programul „PUTTY”. Acesta necesită adresa IP a Raspberry-ului și numărul portului.

Pentru a afla adresa IP s-a folosit software-ul Zenmap ce a scanat toate dispozitivele aflate în rețeaua respectivă.

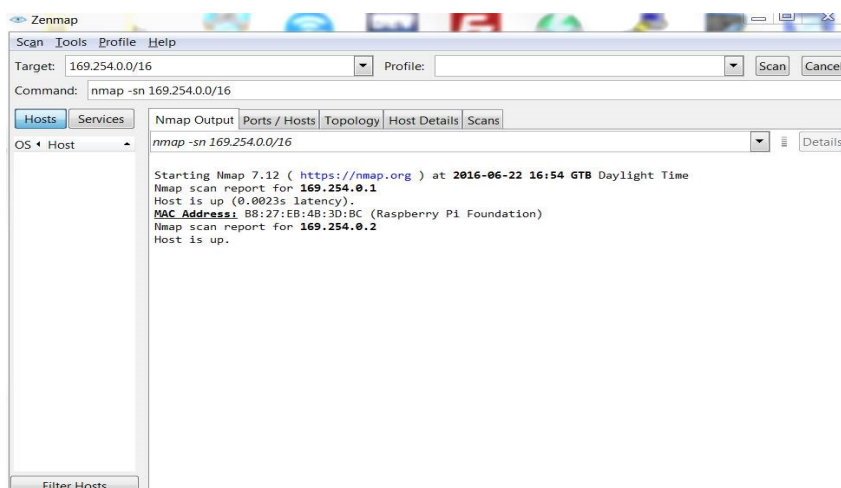


Figura 14 Scanare de IP-uri în Zenmap

După conectarea prin „PUTTY” se pot accesa resursele plăcii Raspberry Pi3. Următorul pas este configurarea unei conexiuni wireless pentru a putea descărca diverse resurse software necesare.

Cum nu dispunem de interfață grafică, comenzile pentru această configurare vor fi comenzi de linux și vor fi explicate în subcapitolul ce tratează această parte.

2.1.8 Modulele XBEE

Folosite pentru a realiza comunicația fără fire între laptop și microcontrollerul Arduino, modulele Xbee sunt alimentate cu 5 volți de la ESC, prin intermediul Arduino și au pini de RX (recepție date) și TX (transmisie date) conectați la pinii Tx, respectiv Rx ai Arduino. Configurarea acestora este realizată prin intermediul celor două programe CoolTherm și XCTU. Odată configurate pentru a comunica între ele, cele două module se conectează de îndată ce sunt alimentate. Existența unui Xbee cu care comunică este indicată prin aprinderea ledului de Asociate de pe dispozitivul asociat, ledurile de RX și TX se aprind în momentul recepționării/trimiterii de date, iar ledul RSSI (received signal strength indicator) indică prin intensitate nivelul de putere (în Dbi) al semnalului.



Figura 15 Modul XBee s2

2.2 NOȚIUNI SOFTWARE

2.2.1 Limbajul Arduino

C++ este un limbaj de programare orientată pe obiect, dezvoltat ca extensie a limbajului C. Limbajul Arduino, folosit pentru a programa microcontrollerul Atmega 328P este în esență C++ simplificat, dar ale cărui limitări nu sunt o problemă pentru codul necesar în proiect. De asemenea, mediul de programare Arduino oferă diverse funcții ce sunt folosite pentru controlul componentelor hardware, precum funcția „digitalWrite”, folosită la crearea semnalelor PWM. Software-ul de compilare, Arduino IDE (Integrated Development Environment) dispune de caracteristici folositoare precum colorarea părților de cod în funcție de categoria acestora (Syntax Highlighting) și indentarea automată. Programele scrise în acest mediu se numesc schițe. O astfel de schiță este compusă tipic din două funcții ce sunt compilate separat, dar pot fi asociate cu funcția „main()” din C++.

- `setup()` este o funcție ce rulează o singură dată la începutul programului și este folosită pentru inițializarea setărilor
- `loop()` este o funcție apelată repetat până când placa nu mai este alimentată

În continuare voi descrie funcțiile predefinite folosite în codul din Anexa 2, în ordinea în care sunt folosite:

- Funcția „`pinMode`” are rolul de a declara pinii folosiți ca fiind de tip „INPUT” sau „OUTPUT”. Toți pinii Arduino folosiți vor fi declarați de tip „OUTPUT” deoarece sunt necesari pentru generarea semnalului PWM ce este trimis la HoverFly Open.
- Funcția „`Serial.begin(9600)`” inițiază comunicația serială, argumentul acesteia fiind rata de transmisie a simbolurilor, în acest caz 9600 biți pe secundă.
- Funcția „`Serial.println(arg.)`” este folosită pentru a scrie în fereastra de comunicație serială argumentul funcției, ce poate fi o variabilă sau un text.
- Funcția „`switch`” permite testarea unei variabile pentru egalitate cu o listă de valori. Fiecare valoare este introdusă într-un „`case`”, iar variabila folosită ca argument al „`switch`”-ului este verificată pentru fiecare „`case`”. Când este găsită o egalitate, programul execută instrucțiunile din „`case`”-ul respectiv până când este întâlnită o funcție „`break()`”. Atunci „`switch`”-ul se încheie și programul trece la următoarea linie de cod de după această funcție.
- Funcția „`if`” testează valoarea booleană prezentă ca argument. Dacă valoarea acesteia este „TRUE”, atunci va fi executat codul din „`if`”, altfel va fi executat codul din „`else`”. În cazul lipsei acestuia, programul trece la următoarea linie de cod de după „`if`”.
- Funcția „`Serial.available()`” returnează numărul de octeți (sau caractere) disponibili pentru citire de la portul serial. În proiect funcția este folosită pentru a testa dacă au fost recepționate informații prin intermediul interfeței seriale.
- Funcția „`Serial.read()`” returnează informația citită de la portul serial. În program este folosită pentru a primi diverse comenzi de la laptopul responsabil de controlul dronei.

2.2.2 XCTU și CoolTherm

X-CTU este un software dezvoltat de creatorii modulelor XBEE și este folosit pentru a configura aceste module de comunicație și pentru a testa rețelele din care acestea fac parte. Pentru a le configura, XBEE-urile trebuie conectate la laptop cu un cablu USB, apoi din program trebuie selectat portul USB respectiv precum și anumiți parametri ai comunicării seriale (BaudRate, Data Bits, Flow Control). Apoi programul va descoperi modulul respectiv și îl va afișa în partea din stânga a ecranului. După ce se face click pe modulul XBEE, programul va citi diferiți parametri, precum ID-ul rețelei și adresele sursă și destinație.

Majoritatea setărilor de configurare controlează cu ce alte XBEE-uri poate comunica dispozitivul curent. Personal Area Network ID este necesar să fie același pentru ambele XBEE-uri pentru ca acestea să comunice. Același lucru este valabil și pentru Channel (canal) care controlează banda de frecvență în care dispozitivul comunică. Adresa MY este adresa sursă a XBEE-ului și adresa stocată în DH și DL este adresa destinație.

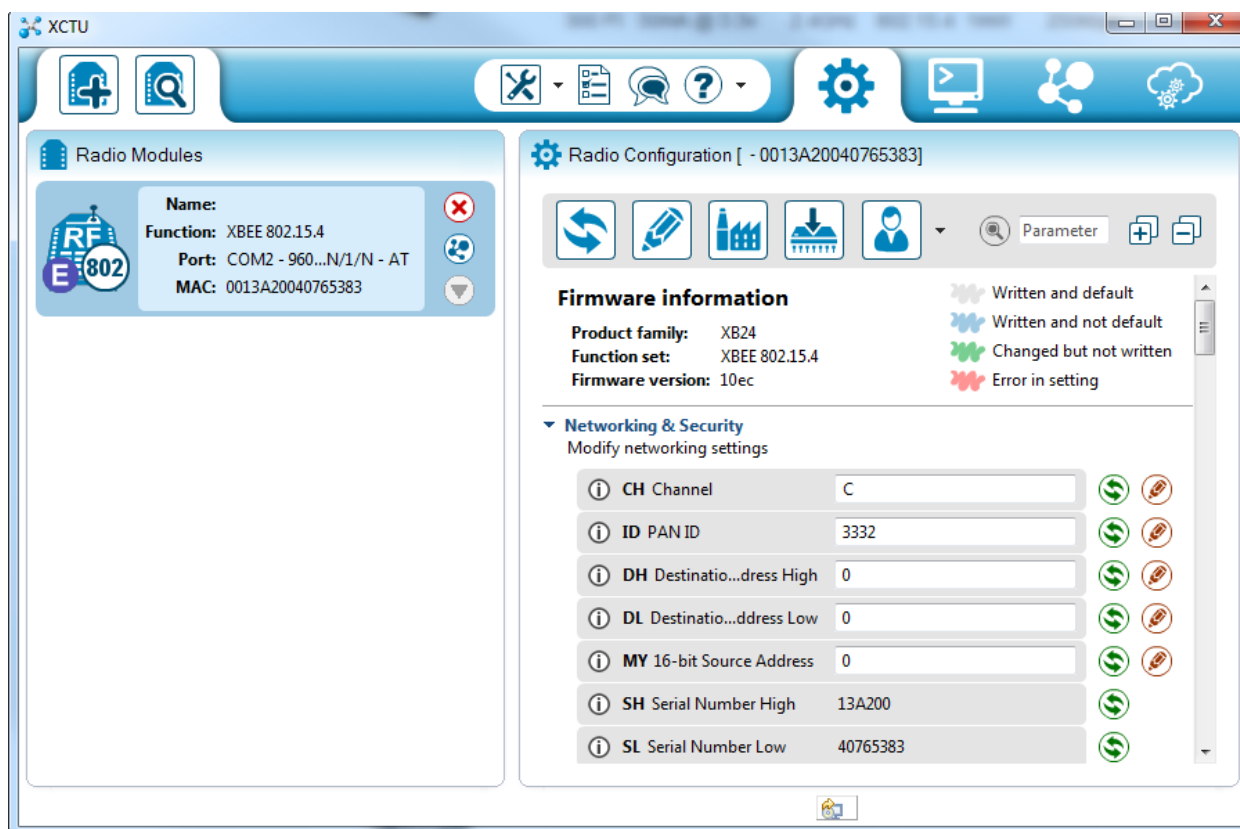


Figura 16 Parametrii XBee-ului citiți prin XCTU

Spre exemplu, un XBEE (a) cu adresa sursă 1 poate primi informații de la un XBEE (b) cu adresa destinație 1. Dar dacă adresa sursă a XBEE-ului (b) este diferită de cea destinație a XBEE-ului (a) atunci această comunicație va fi unidirecțională.

CoolTherm este un program similar, dar ce realizează aceste configurații folosind o linie de comandă. Modul de utilizare al acestuia, dar și alte detalii despre parametrii ce trebuie configurați sunt descrise în subcapitolul despre realizarea comunicației wireless.

2.2.3 Sistemul de operare Linux

La fel ca și Windows sau Mac, Linux este un software ce gestionează toate resursele hardware ale mașinii pe care rulează, în cazul de față Raspberry Pi 3. Motivul pentru care acest sistem de operare este atât de popular este acela că este de încredere (oferă protecție anti viruși, nu are „crash”-uri) și este și gratuit. De asemenea, Linux este OpenSource ceea ce înseamnă că a fost dezvoltat (și este în continuare îmbunătățit) de voluntari și membri ai anumitor organizații din toată lumea. Practic oricine poate contribui la dezvoltarea sa. Din acest motiv există o gamă foarte variată de versiuni de Linux pentru o multitudine de platforme hardware, nu doar calculatoare sau laptopuri. Sistemul de operare Raspbian este bazat pe sistemul Debian Linux. În continuare sunt descriși pașii de configurare a conexiunii wireless pe Raspberry Pi folosind comenzile de linux.

A doua platformă de dezvoltare folosită în proiect este Raspberry Pi3. După conectarea prin programul PUTTY, se pot da comenzi de linux pentru a configura placa. Mai întâi se dorește configurarea conexiunii wireless, deoarece sunt necesare și alte resurse software descărcabile de pe internet.

Comanda `sudo iwlist wlan0 scan` listează rețelele WiFi disponibile. Dintre informațiile afișate după primirea comenzii de interes sunt *ESSID*: „nume rețea” și *IE*: *IEEE 802.11i/WPA2 Version1* ce descrie metoda de autentificare folosită de rețea. Comanda `sudo nano /etc/wpa_supplicant/wpa_supplicant.conf` permite editarea fișierului de configurare „wpa_supplicant”. La capătul fișierului am adăugat:

```
network={  
  
    ssid=„nume rețea”  
  
    psk=„parola WiFi”  
  
}
```

După salvarea fișierului, Raspberry Pi detectează schimbarea făcută și încearcă să se conecteze la rețea.

În continuare se poate deconecta cablul de Ethernet prin intermediul căruia comunicam cu Raspberry și se poate afla IP-ul său din rețeaua WiFi folosind un software (cel folosit de mine se numește „Advanced Ip Scanner”). Apoi se poate folosi rețeaua wireless pentru a reiniția conexiunea cu Raspberry tot cu PUTTY. De fiecare dată când placa este alimentată, ea se va conecta automat la această rețea atâta timp cât există semnal și informațiile despre SSID și securitate nu sunt modificate.

2.2.4 Semnalele PWM și PPM

Acest tip de semnale, menționat adesea în lucrare, este folosit pentru a controla mișcarea quadcopterului. În mod uzual, ele sunt generate de transmițător în urma semnalelor primite de la telecomandă. În proiect, vor fi generate software de microcontrollerul Arduino. Semnalele PWM sunt semnale binare, tensiunea de 5 volți semnificând nivelul „HIGH”, iar cea de 0V „LOW”. Puterea transmisă motoarelor este proporțională cu durata în care semnalul este pe nivelul logic „HIGH”. Astfel, un semnal care într-o singură perioadă are timp de 1ms nivelul logic „HIGH”, apoi „LOW” restul perioadei va semnifica putere minimă pentru motoare. Pentru putere maximă, semnalul va fi timp de 2ms pe „HIGH” într-o perioadă. Perioada semnalului de PWM este de 20ms.

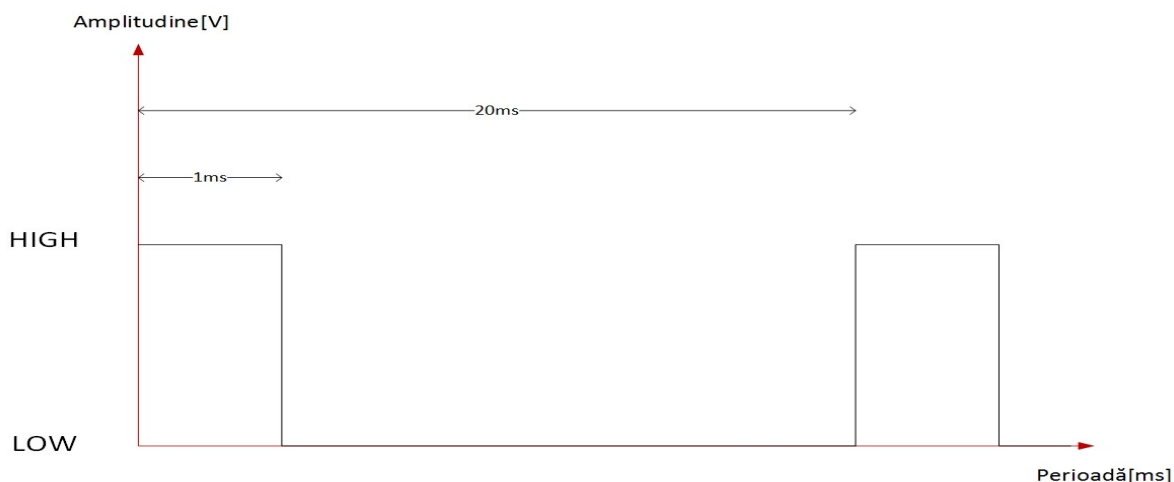


Figura 17 Semnal PWM

De la telecomandă la receiver semnalele PWM corespunzătoare canalelor de Rudder, Throttle, Aileron și Elevator sunt transmise într-un singur tren de undă numit PPM (Pulse Position Modulation) sau TDM (Time Division Multiplexing). În PPM, acestea sunt transmise unul după celălalt în cadrul aceleiași perioade. Perioada semnalului PPM fiind de 20 ms și fiecare semnal PWM având maxim 2 ms, putem concluziona că se pot trimite până la 10 pulsuri PWM într-o singură „anvelopă” PPM.

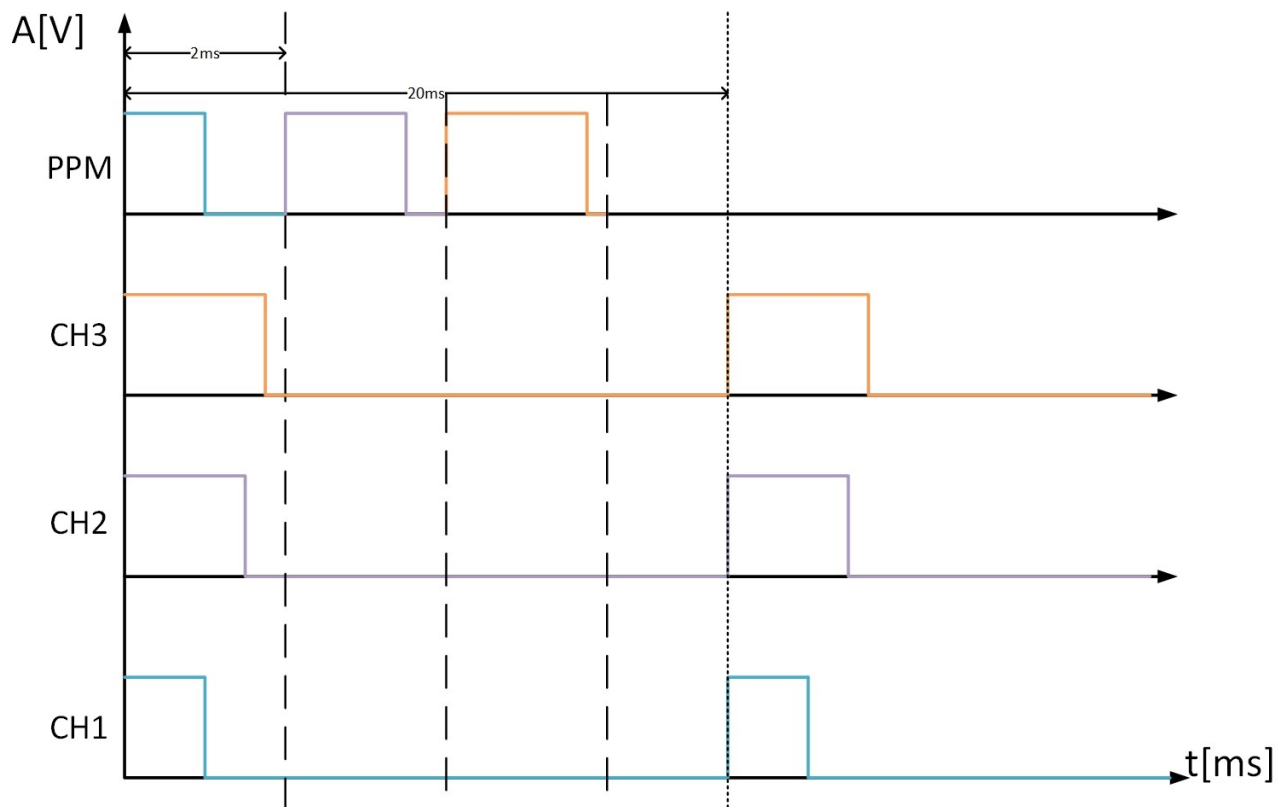


Figura 18 Semnale PPM și PWM

CAPITOLUL 3

DESIGN ȘI DEZVOLTARE

Scopul principal al lucrării este obținerea unui sistem robotic aerian autonom, folosind un kit de quadcopter deja asamblat, o placă de dezvoltare dotată cu un microcontroller pe care va rula codul responsabil de automatizarea dronei și module de comunicație wireless între cele două.

Autonomia robotului este dată de capacitatea acestuia de a se arma singur, de a se ridica de la sol și de a stabili o traiectorie bazându-se pe datele obținute în urma procesării video a poziției unui obiect. Schema quadcopterului este prezentată în subcapitolul teoretic ce descrie funcționarea ESC-urilor. Obținerea acestui sistem a presupus parcurgerea unor pași intermediari precum interconectarea tuturor dispozitivelor, simularea software a unei telecomenzi pentru rutina de armare, efectuarea unor teste preliminare și controlul quadcopterului, precum și numeroase etape de rezolvare a problemelor apărute pe parcurs.

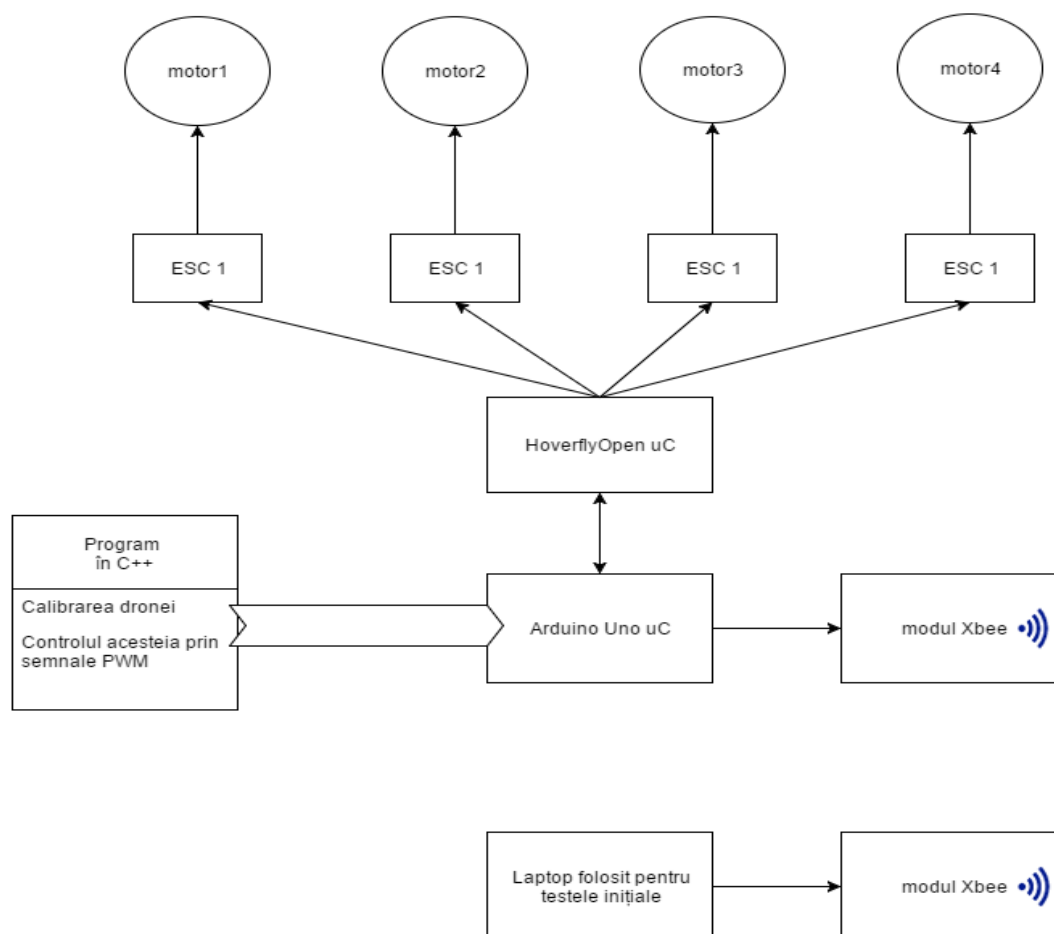


Figura 19 Diagramă reprezentând ultima iterație a sistemului cu Arduino

3.1 CALIBRAREA ESC-URILOR

După asamblarea quadcopterului, am conectat pinii de digital in/out ai Arduino la pinii de control ai ESC-urilor pentru a le calibra. Folosind codul din Anexa 1 am simulat software rutina de calibrare a ESC-urilor, descrisă mai jos:

1. Pornirea transmițătorului
2. Comandă de Throttle maxim
3. Conectarea bateriei la ESC
4. ESC-ul va da un semnal sonor pentru a semnaliza faptul ca este în modul de programare
5. Comandă de Throttle minim
6. O serie de beepuri de la ESC semnificând faptul ca modul de programare a fost decuplat
7. Deconectarea bateriei
8. Repetarea procedurii pentru fiecare ESC

Pentru a simula primii trei pași (fără a folosi un transmițător) am încărcat codul pe Arduino, folosind comunicația serială am apăsut tasta „a” corespunzătoare comenzii de Throttle maxim, și apoi am conectat bateria la ESC. După primirea confirmării sonore am dat comandă de Throttle minim apăsând tasta „d”.

Folosirea delayurilor de 1ms și 2ms pentru generarea pulsurilor PWM de minim și maxim este explicată în secțiunea ce descrie semnalele de tip PWM.

3.2 SIMULAREA UNEI TELECOMENZI SOFTWARE

Următoarea etapă a proiectului este realizarea unei telecomenzi software pentru quadcopter. În acest sens, am scris codul din Anexa 2. Acest cod a fost scris și ținând cont de nevoia unei rutine de calibrare a dronei.

a -Sunt folosiți pini 3,5,6,9,10 de pe Arduino Uno, ca pini digitali In/Out pentru generarea semnalelor PWM. Multiplicatorul este folosit pentru a face tranziția de la miliseconde la secunde. Această tranziție este necesară deoarece inițial, pentru a testa eficiența codului, acești pini au fost conectați la 5 LED-uri. Acestea se aprindeau și stingeau mai repede sau mai încet, proporțional cu variația pulsului PWM. Variabila x din cod este folosită pentru a incrementa sau decrementa durata pulsului PWM de throttle cu o valoare fixă, după cum se observă în funcția generateModifyPWM

b -Sunt declarate anumite stări ce corespund unor configurații ale telecomenzii. Spre exemplu STATE_THROTTLE_LOW_RUDDER_LOW simulează starea în care stickul din stânga este deplasat dreapta jos. (Rudder modifică unghiul de Yaw)

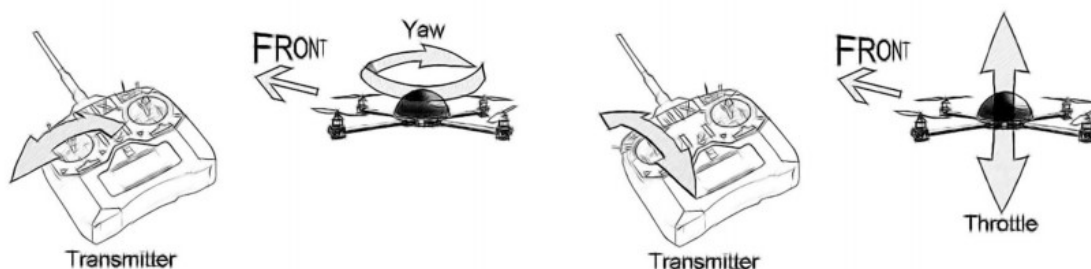


Figura 20 Simularea stării de ThrottleLow_RudderLow

c -Este declarată tasta de control folosită pentru a selecta sau modifica starea curentă

d -Funcțiile listate aici creează semnale PWM cu diferiți factori de umplere

e -Sunt setați pini Arduino ca pini de tip Output. Este pornită comunicația serială cu un BaudRate (rată de transmitere a datelor) de 9600. Starea curentă selectată este cea în care toate semnalele PWM au factorul de umplere de 7.5% (jumătate din viteză).

f -Instrucțiunea „switch” generează pulsurile PWM corespunzătoare fiecărei stări. Este importantă diferența între cele 5 canale folosite. Pe canalul de „Gear”, semnalul nu trebuie să varieze în cadrul unei perioade, deoarece butonul ce ar fi controlat acest canal din telecomandă ar fi avut două stări: una prin care ar fi trimis 0V constant în timp și alta prin care ar fi trimis 5V constant în timp. Un semnal de tip PWM, totuși, este interpretat de Hoverfly ca un semnal de „LOW” constant. Pe canalul de „Throttle” de la un semnal cu perioadă de „HIGH” de o milisecundă, până la unul cu maximum de 2 miliseconde, viteza de rotație a tuturor motoarelor crește pe tot intervalul. Canalul de „Rudder” lucrează asemănător, crescând viteza de rotație în

jurul unghiului de „yaw” proporțional cu creșterea factorului de umplere. Canalul „Aileron” mărește viteza de rotație a câte două seturi de motoare pentru a înclina drona și a o face să se deplaseze spre stânga sau dreapta. Semnalul PWM de pe acest canal determină deplasarea spre dreapta dacă perioada de „HIGH” este mai mare de 1.5 milisecunde și deplasarea spre stânga pentru perioade de „HIGH” mai mici de 1.5 milisecunde. Canalul de „Elevator” are un comportament similar, diferența fiind aceea că drona se va deplasa înainte sau înapoi pentru perioade de „HIGH” mai mari de 1.5 milisecunde, respectiv mai mici.

g -Instrucțiunea „if” testează apăsarea unei taste. În caz afirmativ, în funcție de tasta apăsată, este selectată starea dorită. În cazul stării de modificare a semnalului de Throttle, aici este selectată și valoarea (în milisecunde) cu care durata de „HIGH” a pulsului PWM este incrementată sau decrementată. Deasemenea, după fiecare schimbare a stării, aceasta este afișată în comunicația serială, alături de factorul de modificare „c”.

3.3 SIMULAREA SOFTWARE A RUTINEI DE ARMARE A DRONEI

Stările create software au ca scop principal simularea rutinei de armare. Deși folosind funcția „delayMicroseconds” s-ar fi putut integra într-o singură funcție toate stările în succesiunea în care sunt „așteptate” de către ESC-uri, am hotărât că este mai sigur să fac armarea și dezarmarea manual, de fiecare dată când ridic quadcopterul de la sol.

După cum se observă în segmentul de cod (f) se folosesc stări ce imită semnalele trimise în cazul folosirii unei telecomenzi. Procedura de armare folosind Arduino este următoarea

- Se deschide consola pentru comunicația serială, tasta „s” ce corespunde primei stări așteptate de ESC nu trebuie apăsată, deoarece algoritmul selectează această stare ca fiind cea de default
- După aprinderea LED-ului verde intermitent se apasă tasta „a”
- După un beep se apasă tasta „d”
- Drona este armată

În acest moment dacă apăsăm tasta „c” motoarele vor începe să se învârtă. Apăsarea repetată a acestei taste va mări viteza acestora. Apăsarea tastei „s” va opri complet rotația motoarelor. Pentru a scădea treptat această viteză, se apasă tasta „x”. Pentru a dezarma drona, trebuie repetată secvența de armare.

3.4 TESTE INCIPIENTE EFECTUATE ÎN INCINTA LABORATORULUI

Primele teste au fost efectuate în laborator, fără elice instalate pe motoare. Conectarea bateriei alimentează toate componentele sistemului. Odată ce este alimentat, Arduino trimite semnale PWM corespunzătoare stării ce începe procedura de armare. După ce aceasta a fost efectuată conform pașilor descriși în subcapitolul anterior, am crescut treptat factorul de umplere al semnalului de „Throttle” pentru a vedea cum viteza de rotație a motoarelor se mărește. Trei dintre motoare își modificau viteza sincron, dar al patrulea începea să se învârtă la factori de umplere mai mari. La un semnal PWM cu factor de umplere de 5.5% (perioadă „HIGH” de

1100 ms) începeau să se învârtă doar motoarele 1,2 și 3. Motorul 4 începea să se învârtă la un factor de umplere de 6% și nu ajungea din urmă celelalte motoare nici la factori de umplere mai mari (este acceptabil conform documentației ca la început motoarele să nu se învârtă exact cu aceeași viteză, dar acest lucru este corectat înainte ca acestea să ajungă la viteza de decolare).

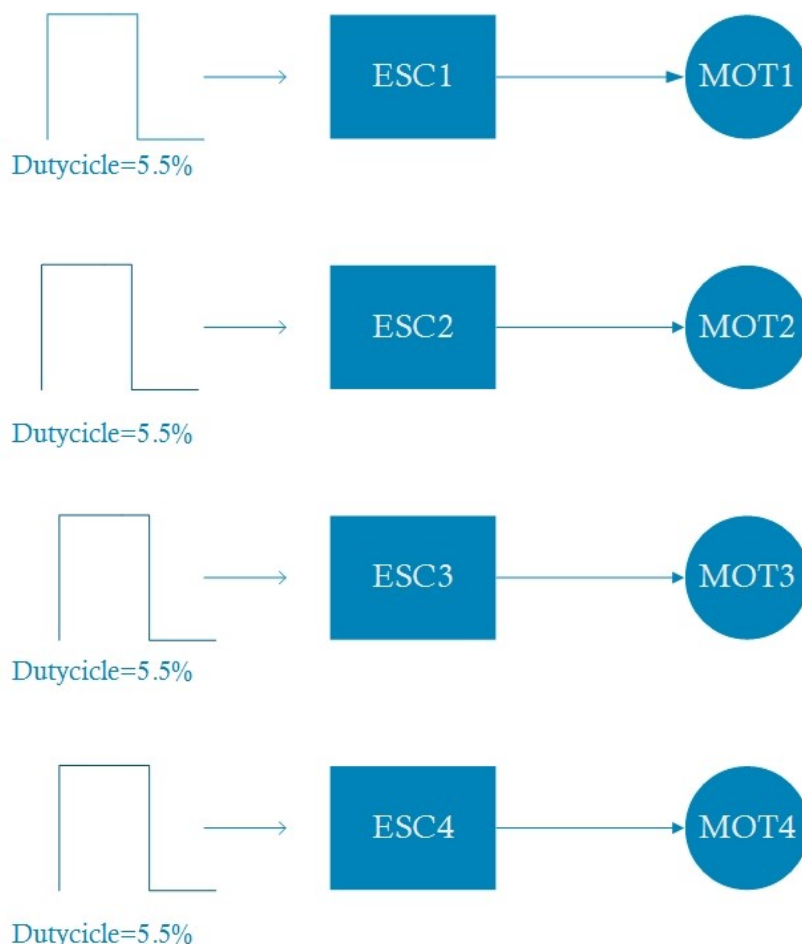


Figura 21 Semnale folosite în testare

În ideea că este o problema de hardware a motorului, l-am schimbat cu unul de rezervă, dar comportamentul a fost identic. Cand am interschimbat ESC-ul cu unul conectat la unul dintre motoarele ce funcționa corect, am constatat că motorul se învârtă sincron cu celelalte. Am concluzionat că problema era de la ESC și se datorează faptului că la un moment dat am schimbat unul dintre motoare împreună cu ESC-ul ce îl controla. Noul ESC era de pe un al doilea kit de quadcopter, ce avusese toate cele patru ESC-uri precalibrate împreună. Rezolvarea problemei apărute a constat în repetarea procedurii de calibrare a ESC-urilor în noua configurație și în testele ce au urmat, vitezele motoarelor s-au modificat corespunzător așteptărilor.

Pentru a verifica capacitatea dronei de a se ridica de la pământ, am creat un sistem cu scopul de a o menține relativ fixă și a nu îi permite să își crească altitudinea prea mult. Am legat strâns patru coarde de patru puncte diametral opuse de pe trenul de aterizare al dronei la un capăt, la celălalt capăt fiind obiecte fixe din cameră. În urma testelor, am constatat că drona se ridică și că viitoarele teste ar trebui efectuate într-un spațiu deschis.

3.5 REALIZAREA UNEI COMUNICAȚII WIRELESS ÎNTRE QUADCOPTER ȘI LAPTOP

Până în acest punct, conexiunea este realizată prin intermediul firelor de legătură. Pentru a putea testa drona în exterior trebuie păstrată o distanță relativ mare de aceasta pentru siguranță. Soluția cea mai la îndemână a fost folosirea modulelor Xbee disponibile.

Prima etapă a fost configurarea Xbee-urilor astfel încât acestea să comunice imediat după alimentare. Am folosit pentru aceasta software-ul CoolTerm. După conectarea primului Xbee la laptop folosind un cablu USB, am configurat setările portului serial: am ales BaudRate de 9600, am selectat 8 biți de date și un bit de stop. Din setările terminalului am activat opțiunea LocalEcho (ce permite vizualizarea în terminal a tastelor apăsate). Apoi, după apăsarea butonului „Connect” am scris „+++” în terminal pentru a intra în modul de comandă (confirmarea intrării în acest mod este dată de afișarea pe ecran a „OK”). Din modul de comandă, am setat anumiți parametri ai acestui modul Xbee pe care îl voi numi A.

- PAN ID a fost setat la o valoare între 0 și FFFE cu comanda „ATID”. Acest parametru (Personal Area Network ID) este un identificator unic pentru rețea. Xbee-urile cu același PAN ID pot comunica doar între ele. Aceasta permite existența a mai multor rețele în aceeași zonă.
- MYAddress a fost setat la valoarea 1 cu comanda „ATMY”. MYAddress reprezintă adresa sursă a acestui Xbee, este adresa folosită de alte dispozitive pentru a trimite informații.
- Destination Address High a fost setat la 0 cu comanda „ATDH”. Acesta reprezintă prima jumătate din adresa de 64 de biți către care vrem să trimitem mesaje. A fost setată la 0 deoarece avem nevoie doar de două adrese și putem folosi ultimii 32 de biți.
- Destination Address Low a fost setat la 1 cu comanda „ATDL”. Reprezintă a doua jumătate din adresa destinație.
- Verificarea comenzilor se face folosind aceleași comenzi, dar fără parametru.
- Comanda ATWR salvează configurația.

Apoi a fost conectat cel de-al doilea Xbee, pe care îl vom numi B. Acestuia i-a fost atribuit același PAN ID, iar MYAddress și Destination Address Low au fost interschimbate cu cele ale Xbee-ului A. Destination Address High a fost setată, desigur, la 0.

Din acest moment, cele două Xbee-uri comunică îndată ce sunt alimentate. Pentru a verifica, totuși acest lucru fără a adapta încă codul dronei, am folosit un cod simplu de aprindere a unui LED (Anexa 3). Am conectat Xbee-ul A la laptop prin USB și Xbee-ul B la Arduino în felul următor:

- Pinul 0 de la XBEE este pinul de GND
- Pinul 3 de la XBEE este pinul de VCC și se conectează la pinul Arduino 5V
- Pinul 4 de la XBEE este pinul Tx și se conectează la pinul digital 2 al Arduino
- Pinul 5 de la XBEE este pinul Rx și se conectează la pinul digital 4 al Arduino

Am modificat codul din Anexa 2, adăugând biblioteca SoftwareSerial și declarând pinii 2 și 4 ca fiind RX și TX folosiți în comunicația serială dintre Xbee-uri.

3.6 PREGĂTIREA QUADCOPTERULUI PENTRU TESTELE DE ZBOR

Sistemul obținut, format din microcontrollerul de zbor, placa de dezvoltare Arduino Uno și modulul Xbee, folosește în această etapă foarte multe fire de legătură între pinii diferitelor plăcuțe. Pentru a putea zbura cu quadcopterul, trebuie luată în calcul eventualitatea unor șocuri mecanice ce ar fi periculoase pentru toate componentele enumerate mai sus. De aceea am creat o placă în PCB care să ruteze toate legăturile reducând nevoia de fire la un minim necesar. De asemenea, kitul Arduino și modulul Xbee vor putea fi prins în șuruburi de această placuță.

Pentru crearea acestui „shield” de protecție, am folosit software-ul de design asistat de calculator Proteus.

Prima etapă a fost realizarea schemei electrice prin găsirea conectorilor potriviți și plasarea etichetelor pe fire.

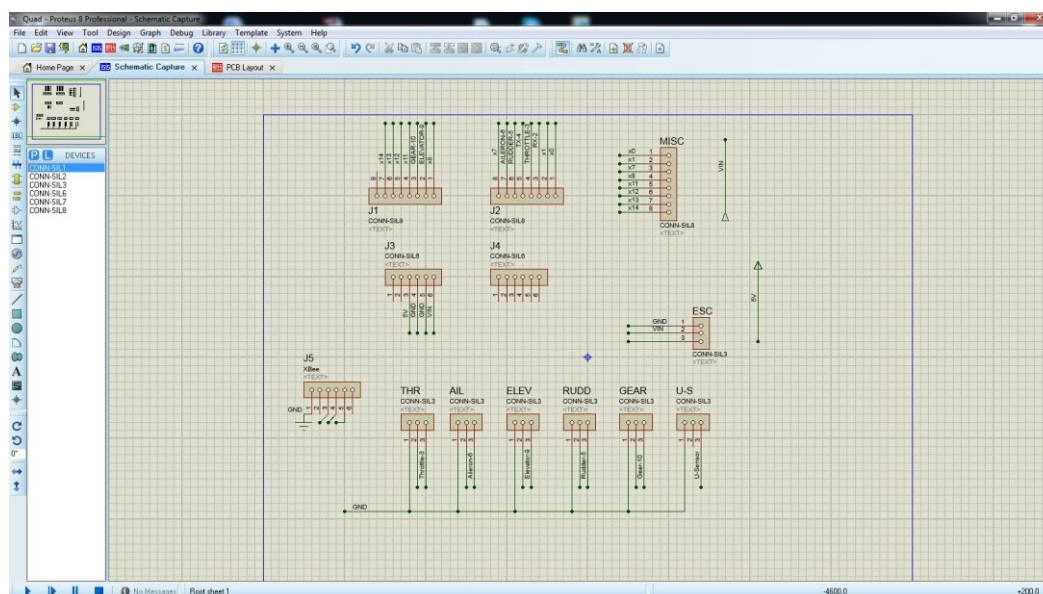


Figura 22 Schema PCB în Proteus

Apoi, folosind programul Ares, am rutat traseele electrice între conectorii din schemă. După generarea fișierului de layout ce conține informațiile necesare pentru imprimarea circuitului, am printat circuitul la laboratorul CETTI. Apoi am dat găurile necesare și am lipit barele de pini pentru conectarea cu componentele ce vor fi puse pe shield.

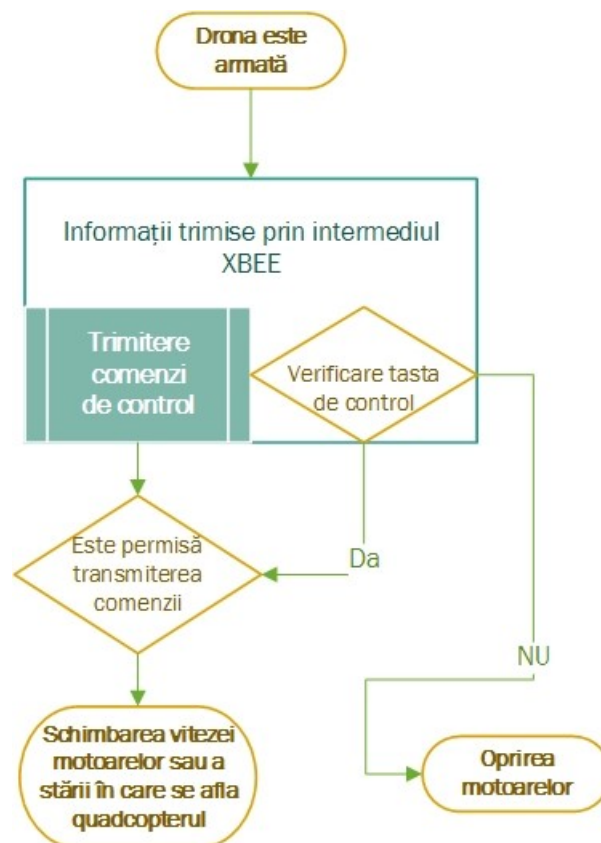


Figura 24 Diagrama sistemului de test în aer liber folosind module XBee

În urma unui set de teste s-a observat că XBEE-urile își pierdeau des conexiunea. Aceasta a fost pusă pe seama faptului că acest modul de XBEE folosește o antenă integrată de tip PCB ce este folosită de obicei în interior. De aceea, următorul pas a fost schimbarea Arduino cu un microcontroller cu modul de comunicație wireless integrat, Raspberry Pi3.

3.8 SCHIMBAREA PLATFORMEI ARDUINO CU CEA RASPBERRY

Prima etapă este updatarea sistemului de operare folosind cele două comenzi

sudo apt-get update

sudo apt-get upgrade

Pentru a scrie codul în Python, a fost necesară instalarea pachetelor și librăriilor specifice acestui limbaj, prin comenzile

sudo apt-get install python-pip

Pentru generarea semnalelor PWM trebuie folosiți pini GPIO ai Raspberry, deoarece sunt pini digitali de INPUT/OUTPUT. Pentru ca acești pini să poată fi folosiți, trebuie instalată biblioteca ce permite accesul la ei prin instrucțiuni de tipul „wiringpi.digitalWrite”. Configurarea pinilor s-a făcut prin instrucțiunea:

sudo apt-get install python-rpi.gpio

Importarea bibliotecii ce permite accesul la pini în C++ sau Python s-a făcut prin instrucțiunea:

```
git clone git://git.drogon.net/wiringPi
cd wiringPi
git pull origin
./build
```

Această librărie a fost în mod convenabil concepută pentru a se asemana sistemului Arduino de folosire a pinilor. Setul următor de comenzi descarcă și instalează această librărie:

Diferențele majore între cele două programe au constatat în

- Folosirea unei bucle infinite „while 1” în locul funcției „loop” specifică Arduino.
- Înlocuirea comunicației seriale cu o structură de python de scriere în linie de comandă ce modifică starea doar în cazul apăsării unei taste.
- Pentru a folosi un anumit pin ca output de semnal PWM se folosește instrucțiunea „wiringpi.pinMode(17, 2)”. Al doilea argument al funcției este cel ce identifică tipul de semnal ce va fi transmis prin pinul respectiv, 2 fiind echivalent cu PWM.

În final, se conectează laptopul la Raspberry Pi prin „PUTTY”, iar după rularea programului se pot da comenzile în consolă. Încheierea buclei infinite create nu se poate face decât întrerupând forțat execuția programului apăsând CTRL + Z.

3.9. ALGORITM DE URMĂRIRE A UNEI ȚINTE MOBILE

Coordonarea mișcărilor unui multicopter pentru a duce la îndeplinire un obiectiv, în cazul lucrării de față urmărirea unui obiect țintă mobil, poate folosi mai multe moduri de obținere a datelor cu privire la poziția dronei. Sistemul GPS este foarte popular în aeronautică, deoarece este ușor de folosit și de aceea avioanele îl utilizează în timpul zborului la mare altitudine. Cu toate acestea, acuratețea de ± 2.5 metri și delayul cauzat de transferul datelor la sateliți și înapoi fac GPS-ul o soluție mai puțin optimă în cazul dronelor de dimensiuni mici. Pentru a putea fi autonom, un quadcopter trebuie să aibă încorporate capacități de obținere a datelor video și de procesare a acestora. Camera video folosită este Rpi Vision2 și face parte din kitul Raspberry Pi.

Locul optim de plasare al camerei este în partea din față a quadcopterului, astfel se va cuprinde în raza vizuală a camerei o zonă relativ mare. Procedurile de armare și dezarmare a dronei se vor face software dar manual, apăsând taste ce corespund stărilor în care trebuie să intre quadcopterul. Tot manual se va realiza și pornirea/oprirea rutinei de urmărire a țintei mobile. Această rutină va începe cu ridicarea verticală de la sol a dronei, până la o înălțime de aproximativ 1.5m. Metoda de aflare a acestui punct este estimativă, în cod fiind folosită funcția „liftoff()” ce mărește factorul de umplere doar pe canalul de „Throttle” o anumită durată de timp după care este activat software switch-ul de la canalul Gear, ceea ce activează modul de „Altitude Hold” al dronei. Factorul de umplere folosit și timpul necesar vor fi aflate efectuând teste succesive cu diferiți factori de umplere pentru a vedea cum aceștia afectează viteza de ascensiune a dronei. Apoi, folosind un algoritm de captare și prelucrare a imaginilor, vor fi obținute informații cu privire la poziția obiectului. Pentru simplitate, acesta va fi de forma unui pătrat de culoare roșie. Părțile de transmisie și prelucrare video nu constituie obiectivul acestei lucrări, din acest algoritm fiind de interes doar 3 parametrii: coordonatele colțului din stânga sus al dreptunghiului, lungimea și lățimea acestuia. Astfel, în rutina de urmărire prima linie de cod după funcția de ridicare de la sol va verifica găsirea obiectului pe imagine. Dacă obiectul nu

apare pe imagine, quadcopterul va primi o comandă de modificare a unghiului de „yaw” cu 30°, iar aceasta în mod repetat până când obiectul este identificat în imagine. Când timp obiectul este identificat, se activează rutina „tracking()” ce dezactivează modul de „Altitude Hold”, modifică semnalele de „Throttle”, „Rudder” și „Aileron” astfel încât obiectul să fie mereu în centrul imaginii și să își păstreze dimensiunile. În cazul în care obiectul nu mai este găsit în imagine, se va activa din nou „Altitude Hold” și se va roti quadcopterul modificând succesiv unghiul de „yaw” până când este identificat din nou. Pentru a opri quadcopterul, se va reduce factorul de umplere al semnalului de pe canalul de „Throttle”, iar după ce drona a aterizat, se va opri rotația motoarelor și se va face rutina de dezarmare.

Acest algoritm este ilustrat în diagrama de pe pagina următoare.

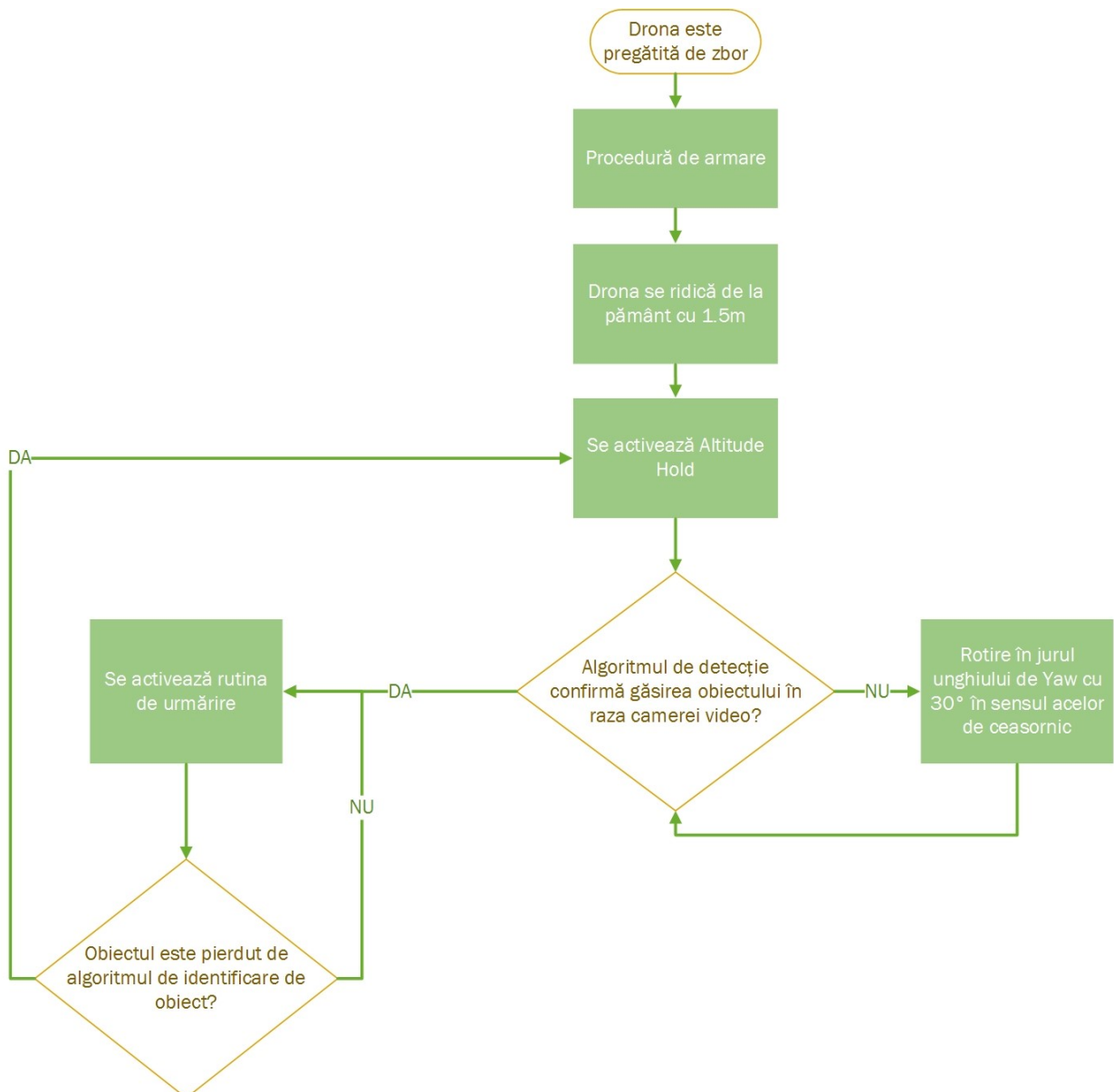


Figura 25 Algoritm de urmărire a obiectului

Pentru a urmări obiectul, acesta trebuie să aibă colțul din stânga sus încadrat într-o fereastră de mici dimensiuni de pe ecran. Această condiție verifică dacă quadcopterul este aliniat cu obiectul

căutat. De asemenea, trebuie ca dimensiunea obiectului, în cazul meu lungimea laturii, să se mențină constantă. Astfel distanța dintre obiect și quadcopter este și ea constantă.

Rutina de urmărire se bazează pe împărțirea imaginii într-o matrice cu 640 de coloane și 480 de rânduri. Primul test ce se va face pentru a stabili felul în care trebuie să se deplaseze quadcopterul se referă la colțul din stânga sus al imaginii. Se va împărți imaginea în 9 zone în funcție de fereastra din matrice în care trebuie „prins” colțul obiectului și pentru poziția colțului în fiecare dintre aceste 9 zone, se va executa o rutină ce va deplasa drona corespunzător. În cazul în care colțul este în fereastra dorită, se va verifica dacă pătratul este la distanța dorită de obiect. Aceasta se va realiza comparând dimensiunea laturii cu o dimensiune standard, ce s-a determinat prin calcul că este corespunzătoare unei anumite distanțe dorite între dronă și obiect.

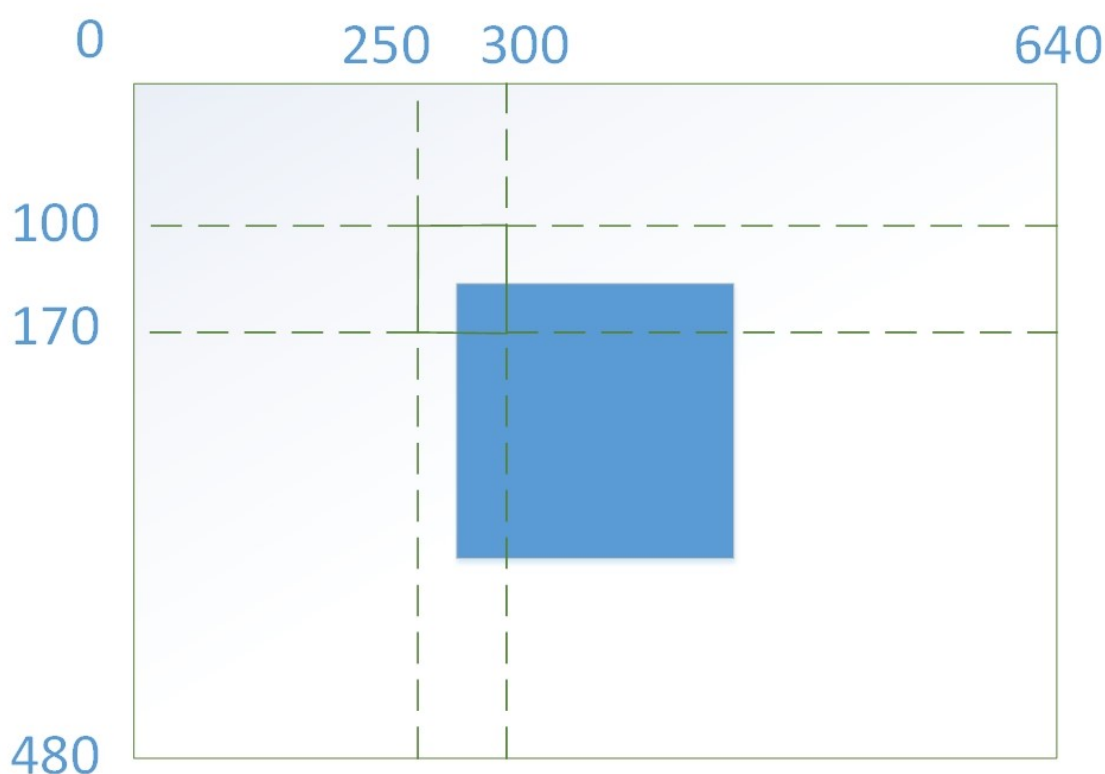


Figura 26 Identificarea colțului stânga sus al imaginii

În Anexa 4 este completarea codului din Anexa 2, ce permite quadcopterului să urmărească o țintă mobilă în mod autonom. Din cauza problemelor legate de funcționarea algoritmului de echilibrare în aer al microprocesorului Hoverfly Open, acest cod nu a fost testat pe dronă, dar în crearea lui s-a ținut cont de toate aspectele legate de funcționarea quadcopterului aflate până în acest punct prin intermediul diferitelor informații găsite online și testelor făcute cu quadcopterul.

Funcției „loop”, a cărei executare se repetă până când placa de dezvoltare pe care rulează este deconectată de la alimentare, îi sunt adăugate 2 comenzi noi de schimbare a stării în care se află quadcopterul și anume pornirea și oprirea rutinei de autonomie. În cazul în care aceasta este

pornită, prin intermediul funcției „startingpoint()”, quadcopterul se va ridica la o mică distanță de pământ, pentru a oferi o mai bună rază vizuală camerei video folosite. În continuare, funcția „identified” returnează o valoare booleană ce corespunde situației în care obiectul este sau nu găsit de algoritmul de detecție video. Dacă valoarea este „true”, atunci se execută funcția „tracking()”, iar în caz contrar se execută funcția „rotate30” (ce rotește quadcopterul în jurul unghiului de „yaw” timp de două secunde), apoi se execută din nou un salt la condiția de verificare a găsirii obiectului. În cazul în care obiectul nu poate fi găsit, această buclă se părăsește apăsând tasta „f”. Funcția „tracking()” modifică semnalele PWM pe toate canalele pentru a deplasa drona către obiect conform algoritmului descris mai sus. Astfel, după cum se observă în cod, pentru fiecare dintre cele 9 cazuri de poziție a colțului obiectului în imagine, există câte o rutină de mișcare a dronei în planul perpendicular pe Pământ. În cazul în care colțul respectiv este în locul dorit pe imagine, două rutine de mișcare vor deplasa robotul pentru a micșora/mări distanța între acesta și obiectul țintă. La începutul funcției „tracking()” este verificată mereu apăsarea tastei f pentru a ieși din acest mod de deplasare atunci când este necesar.

CAPITOLUL 4

CONCLUZII ȘI DEZVOLTĂRI VIITOARE

4.1 CONCLUZII

Așa cum se poate observa din conținutul acestei lucrări, scopurile propuse au fost atinse într-o anumită măsură. Acestea sunt integrarea mai multor dispozitive într-un sistem aerian de tip multicopter, permițând transferul informației destinate controlului acestuia, realizarea unei telecomenzi software și dezvoltarea unui program cu scopul de a face robotul autonom. Modul de control ales este generarea semnalelor PWM ce ar fi fost în mod obișnuit transmise de un receiver. Tipul de semnale necesar pentru a „convinge” quadcopterul că primește informații de la un receiver a fost aflat după numeroase încercări cu diverse tipuri de semnale PWM. După aflarea perioadelor exacte necesare, însă, progresul lucrării a avansat semnificativ și la scurt timp am reușit să implementez atât procedura de armare cât și cea de control. Comunicația wireless s-a dovedit necesară la scurt timp deoarece orice teste de interior cu elicele puse ar fi fost prea periculoase. De asemenea ar fi fost imposibil de evaluat eficiența controlului dacă robotului nu i-ar fi fost permis să zboare mai sus decât o putea face constrâns de niște legături pentru siguranță. După configurarea modulelor de comunicație fără fir, a devenit evidentă nevoia unei structuri care să înlocuiască toate firele ce legau numeroșii pini din sistem. Testele făcute în exterior după montarea tuturor componentelor pe „shieldul” pe care îl concepușem au evidențiat 2 probleme majore pentru proiect. Prima a fost aceea că modulul de XBee oferea o

conectivitate precară și nu putea fi folosit pentru etapa mai complexă a proiectului. A doua a fost aceea că algoritmul de stabilizare în aer pe care mă bazam nu era atât de precis pe cât sperasem. Deși prima problemă a fost depășită relativ ușor prin folosirea unui microcontroller de comandă cu modul WiFi integrat, cea de-a doua a însemnat imposibilitatea implementării pe quadcopter a algoritmului de automatizare. Cu toate acestea, nu consider nevoia de a schimba platforma o decepție, deoarece mi-a dat ocazia să învăț despre sistemul de operare linux, despre limbajul de programare „Python” și despre modul de lucru cu ambele platforme de dezvoltare Raspberry Pi și Arduino Uno. Testele implementării programului de simulare a telecomenzii software au reușit în toate formele în care au fost încercate: conectând platforma de dezvoltare la laptop prin cablu mini USB pentru trimiterea comenzilor pe portul serial și folosind modulele Xbee.

4.2 DEZVOLTĂRI VIITOARE

Deși multicopterele și controlul autonom al acestora au fost studiate extensiv de-a lungul timpului, se pot găsi mereu noi direcții promițătoare de studiu și dezvoltare. Pentru proiectul meu propun următoarele îmbunătățiri/direcții alternative:

- Crearea unui microcontroller de zbor propriu, folosind senzori precum giroscopae, accelerometre și senzori ultrasonici.
- Modificarea programului de urmărire a obiectului în așa fel încât acesta să aibă acces la informațiile primite de la senzori și să adapteze mișcările quadcopterului la aceste informații.
- Schimbarea programului pentru a permite urmărirea unor obiecte mai complexe. Spre exemplu, în lucrarea de față obiectul urmărit este un pătrat, iar acesta are mereu toate laturile egale, aceasta însemnând că obiectul se poate deplasa doar după anumite direcții și nu după anumite unghiuri. În realitate o astfel de deplasare este practic imposibilă.
- Adăugarea unui GPS pentru a avea o metodă alternativă de deplasare autonomă, folosind camera video doar pentru navigare în spații restrânse.
- Realizarea comunicației între mai mulți roboți.

BIBLIOGRAFIE

1. <http://www.krossblade.com/history-of-quadcopters-and-multirotors/> accesat la data 02.06.2016
2. <https://www.kth.se/en/ees/forskning/strategiska-forskningsomraden/intelligenta-transportssystem/smart-mobility-lab/projects/masters-theses/event-triggered-attitude-control-of-a-quadcopter-1.484267> accesat la data 19.04.2016
3. <https://www.parallax.com/sites/default/files/downloads/80000-Hoverfly-OPEN-Users-Guide-v1.0.pdf> accesat la data 12.10.2015
4. https://en.wikipedia.org/wiki/Euler_angles accesat la data 06.11.2015
5. https://support.dce.felk.cvut.cz/mediawiki/images/d/d4/Dp_2011_jirinec_tomas.pdf accesat la data 21.05.2016
6. <https://liu.diva-portal.org/smash/get/diva2:908582/FULLTEXT02.pdf> accesat la data 02.06.2016
7. <https://alselectro.wordpress.com/2014/10/13/quadcopter1-things-you-need-to-build-your-own-copter/> accesat la data 15.10.2015
8. <http://www.dronetrest.com/t/brushless-motors-how-they-work-and-what-the-numbers-mean/564> accesat la data 12.10.2015
9. <http://www.stefanv.com/electronics/escprimer.html> accesat la data 12.10.2015
10. <https://hoverflytech.zendesk.com/hc/en-us/categories/200136586-User-Guides-and-Software> accesat la data 20.11.2015
11. <http://www.rchelicopterfun.com/rc-lipo-batteries.html> accesat la data 02.01.2016
12. <https://en.wikipedia.org/wiki/Arduino> accesat la data 09.10.2015
13. https://en.wikipedia.org/wiki/Syntax_highlighting accesat la data 10.06.2016
14. <https://www.arduino.cc/en/Serial/Available> accesat la data 11.03.2016
15. <https://learn.sparkfun.com/tutorials/xbee-wifi-hookup-guide/using-x-ctu> accesat la data 20.04.2016
16. <https://learn.sparkfun.com/tutorials/exploring-xbees-and-xctu> accesat la data 20.04.2016
17. <https://www.linux.com/what-is-linux> accesat la data 02.06.2016
18. <http://www.raspberry-projects.com/pi/programming-in-c/getting-your-raspberry-pi-ready-for-c-programming> accesat la data 08.05.2016
19. <https://www.raspberrypi.org/documentation/remote-access/ip-address.md> accesat la data 09.05.2016
20. <https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md> accesat la data 09.05.2016
21. <http://wiringpi.com/> accesat la data 15.05.2016
22. http://robotics.asu.edu/ardrone2_ibvs/ accesat la data 10.06.2016
23. http://uknowledge.uky.edu/cgi/viewcontent.cgi?article=1097&context=gradschool_theses accesat la data 13.06.2016

ANEXE

ANEXA 1

```
#include<Servo.h>
int maxi=2000;
int mini=1000;
int pin=9;

Servo motor;
void setup() {
  Serial.begin(9600);
  Serial.println("Begin Calibration");
  motor.attach(pin);
}
void loop() {
  if (Serial.available() > 0){
    command=Serial.read();
    switch (command){
      case 97: //a
        motor.writeMicroseconds (maxi);
        break;
      case 100: //d
        motor.writeMicroseconds (mini);
        break;
```

```
} }
```

ANEXA 2

a

```
int throttlePin=3; //Throttle
int rudderPin=5; //Yaw
int aileronPin=6; //Roll
int elevatorPin=9; //Pitch
int gearPin=10; //Gear
int multiplier = 1; // factor de multiplificare
int x=0;
```

b

```
const int STATE_THROTTLE_LOW = 0;
const int STATE_THROTTLE_LOW_RUDDER_LOW = 1;
const int STATE_THROTTLE_LOW_RUDDER_HIGH = 2;
const int STATE_ALL_IDLE = 3;
const int STATE_lowThrottle_THROTTLE=4;
const int STATE_MODIFY_THROTTLE=5;
```

c

```
int command; // tasta de control
```

d

```
void generateDefaultPWM(int pin){
    digitalWrite(pin,HIGH);
    delayMicroseconds(1500*multiplier);
    digitalWrite(pin,LOW);
    delayMicroseconds(500*multiplier);
}

void generateLowDutyCyclePWM(int pin){
    digitalWrite(pin,HIGH);
    delayMicroseconds(1000*multiplier);
    digitalWrite(pin,LOW);
    delayMicroseconds(1000*multiplier);
}

void generateHighDutyCyclePWM(int pin){
    digitalWrite(pin,HIGH);
    delayMicroseconds(2000*multiplier);
    digitalWrite(pin,LOW);
    delayMicroseconds(0*multiplier);
}
```

```

void generatelowThrottlePWM(int pin) {
    digitalWrite(pin,HIGH);
    delayMicroseconds(1300*multiplier);
    digitalWrite(pin,LOW);
    delayMicroseconds(750*multiplier); }

```

```

void generateModifyPWM(int pin){
    digitalWrite(pin,HIGH);
    delayMicroseconds(1000+x);
    digitalWrite(pin,LOW);
    delayMicroseconds(1000-x);
}

```

e

```

void setup() {
    pinMode(throttlePin,OUTPUT);
    pinMode(rudderPin,OUTPUT);
    pinMode(aileronPin,OUTPUT);
    pinMode(elevatorPin,OUTPUT);
    pinMode(gearPin,OUTPUT);
    Serial.begin(9600);
    Serial.println("Apasa");
    state=STATE_THROTTLE_LOW;
    Serial.println(state);
}

```

f

```

void loop() {
    switch (state) {
    case STATE_THROTTLE_LOW:
        generateLowDutyCyclePWM(throttlePin);
        generateDefaultPWM(rudderPin);
        generateDefaultPWM(aileronPin);
        generateDefaultPWM(elevatorPin);
        generateDefaultPWM(gearPin);
        delayMicroseconds(10000);
        break;
    case STATE_THROTTLE_LOW_RUDDER_LOW:
        generateLowDutyCyclePWM(throttlePin);
        generateLowDutyCyclePWM(rudderPin);
        generateDefaultPWM(aileronPin);
        generateDefaultPWM(elevatorPin);
        generateDefaultPWM(gearPin);
        delayMicroseconds(10000);
        break;
    case STATE_THROTTLE_LOW_RUDDER_HIGH:
        generateLowDutyCyclePWM(throttlePin);
        generateHighDutyCyclePWM(rudderPin);
        generateDefaultPWM(aileronPin);
        generateDefaultPWM(elevatorPin);
        generateDefaultPWM(gearPin);
        delayMicroseconds(10000);
    }
}

```

```

    break;
case STATE_ALL_IDLE:
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
    break;

case STATE_MODIFY_THROTTLE:
    generateModifyPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);

    break;
}

```

g

```

if (Serial.available() > 0) {
    command=Serial.read();
    switch (command) {
    case 97: //a
        state = STATE_THROTTLE_LOW_RUDDER_LOW;
        break;
    case 100: //d
        state = STATE_THROTTLE_LOW_RUDDER_HIGH;
        break;
    case 115: //s
        state = STATE_THROTTLE_LOW;
        break;
    case 98: //b
        state = STATE_ALL_IDLE;
        break;
    case 99: //c = increment PWM
        x=x+25;
        state=STATE_MODIFY_THROTTLE;
        break;
    case 120: //x = decrement PWM
        x=x-25;
        state=STATE_MODIFY_THROTTLE;
        break;
    }
    Serial.println(state);
    Serial.println(x);
}

```

}

ANEXA 3

```
int myData = 0;
int const ledpin = 9;

void setup() {
    Serial.begin(9600);
    pinMode(ledpin, OUTPUT);
}

void loop() {

    if(Serial.available() > 0){

        myData = Serial.read();

        if(myData == '1'){
            digitalWrite(ledpin, HIGH);
            Serial.write(myData);
        }

        if(myData == '2'){
            digitalWrite(ledpin, LOW);
            Serial.write(myData);
        }
        else{
            Serial.write(myData);
        }

    }
}
```

ANEXA 4

```
int corner_x,corner_y,length; //variabile obtinute in urma identificarii
obiectului (pentru simplitate,am ales un pătrat)
int standard_length; //lungimea laturii pătratului ce corespunde unei anumite
distanțe fata de obiect, calculată apriori
void generatehigherPWM( int pin)
```

```

{
    digitalWrite(pin, HIGH);
    delayMicroseconds(1600);
    digitalWrite(pin, LOW);
    delayMicroseconds(400);
}

void generateDescentPWM(int pin)    //semnal PWM folosit pentru canalul de
Throttle in momentul in care se doreste scaderea altitudinii pentru a urmari
obiectul
{
    digitalWrite(pin, HIGH);
    delayMicroseconds(1100);
    digitalWrite(pin, LOW);
    delayMicroseconds(900);
}

void generatelowerPWM(int pin)
{
    digitalWrite(pin, HIGH);
    delayMicroseconds(1400);
    digitalWrite(pin, LOW);
    delayMicroseconds(600);
}

void liftoff()
{
    generatelowThrottlePWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

void AltitudeHold_off()
{
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateLowDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

void AltitudeHold_on()
{
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateHighDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

```

```

}

void move_up()
{
    generateLowThrottlePWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

void move_down()
{
    generateDescentPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

void move_up_right()
{
    generateLowThrottlePWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateHigherPWM(aileronPin); // deplasare dreapta
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

void move_up_left()
{
    generateLowThrottlePWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateLowerPWM(aileronPin); //deplasare stanga
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

void move_down_right()
{
    generateDescentPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateHigherPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

```

```

void move_down_left()
{
    generateDescentPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generatelowerPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateDefaultPWM(gearPin);
    delayMicroseconds(10000);
}

void move_left() // miscarea la stanga si la dreapta se fac cu AltitudeHold
activat -> Gearpin HIGH
{
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generatelowerPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateHighDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

void move_right() // miscarea la stanga si la dreapta se fac cu AltitudeHold
activat -> Gearpin HIGH
{
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generatehigherPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateHighDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

void move_forward()
{
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generatehigherPWM(elevatorPin);
    generateHighDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

void move_backward()
{
    generateDefaultPWM(throttlePin);
    generateDefaultPWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generatelowerPWM(elevatorPin);
    generateHighDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

```

```

}

void startingpoint() // functie folosita pentru a ajunge in punctul de
cautare al obiectului
{
    liftoff();
    delayMicroseconds(2000000);
    AltitudeHold();
}

void rotate30()
{
    generatelowThrottlePWM(throttlePin);
    generateInitialRotatePWM(rudderPin);
    generateDefaultPWM(aileronPin);
    generateDefaultPWM(elevatorPin);
    generateHighDutyCyclePWM(gearPin);
    delayMicroseconds(10000);
}

void tracking()
{
    if Serial.available()
        command=Serial.read();
    if (command==102) //f , comanda pentru iesirea manuala din modul de
tracking
        goto exit;
    AltitudeHold_off();
    if(corner_x>300)
        if(corner_y<100)
        {
            time_t start = time(NULL);
            time_t now = time(NULL);
            while ((now - start) <= 1)
            {
                move_up_right();
                time_t now = time(NULL);
            }
            goto label;
        }

    else if((corner_y>=100) && (corner_y<=170))
    {
        time_t start = time(NULL);
        time_t now = time(NULL);
        while ((now - start) <= 1)
        {
            move_right();
            time_t now = time(NULL);
        }
        goto label;
    }
}

```

```

    }
    else if(corner_y>170)
    {
        time_t start = time(NULL);
        time_t now = time(NULL);
        while ((now - start) <= 1)
        {
            move_down_right();
            time_t now = time(NULL);
        }
        goto label;
    }
    else if(corner_x<250)
        if(corner_y<100)
        {
            time_t start = time(NULL);
            time_t now = time(NULL);
            while ((now - start) <= 1)
            {
                move_up_left();
                time_t now = time(NULL);
            }
            goto label;
        }
    else if((corner_y>=100)&&(corner_y<=170))
    {
        time_t start = time(NULL);
        time_t now = time(NULL);
        while ((now - start) <= 1)
        {
            move_left();
            time_t now = time(NULL);
        }
        goto label;
    }

    else if(corner_y>170)
    {
        time_t start = time(NULL);
        time_t now = time(NULL);
        while ((now - start) <= 1)
        {
            move_down_left();
            time_t now = time(NULL);
        }
        goto label;
    }
    else if((corner_x<=300)&&(corner_x>=250))
        if(corner_y<100)
        {
            time_t start = time(NULL);

```

```

        time_t now = time(NULL);
        while ((now - start) <= 1)
        {
            move_up();
            time_t now = time(NULL);
        }
        goto label;
    }

    else if((corner_y>=100)&&(corner_y<=170))
    {
        if(length<standard_length)
        {
            time_t start = time(NULL);
            time_t now = time(NULL);
            while ((now - start) <= 1)
            {
                move_forward();
                time_t now = time(NULL);
            }
            goto label;
        }
        else if (length>standard_length)
        {
            time_t start = time(NULL);
            time_t now = time(NULL);
            while ((now - start) <= 1)
            {
                move_backward();
                time_t now = time(NULL);
            }
            goto label;
        }
        else if (length==standard_length)
            goto label;
    }

    else if(corner_y>170)
    {
        time_t start = time(NULL);
        time_t now = time(NULL);
        while ((now - start) <= 1)
        {
            move_down();
            time_t now = time(NULL);
        }

        goto label;
    }
}

```

}

```

void automate()
{
    startingpoint();
label: if(identified())
        tracking();
    else
    {
        time_t start = time(NULL);
        time_t now = time(NULL);
        while ((now - start) <= 2)
        {
            rotate30();
            time_t now = time(NULL);
        }
        if Serial.available()
            command=Serial.read();
        if (command==102) //f , comanda pentru iesirea manuala
        din modul de căutare al obiectului
            goto exit;

        goto label;
    }
}

void loop()
{
    command=Serial.read();
exit: switch(command)
    {
        case 101: // e
            automate();
        case 102: //f
            AltitudeHold_on();
    }
}

```