



Universitatea Tehnică Gheorghe Asachi  
Facultatea de Automatică și Calculatoare  
Departamentul de Automatică și Informatică Aplicată  
Specializarea: Ingineria Sistemelor

## Lucrare de licență

### Metodă pentru netezirea traietoriei unei drone

Student  
Mereuță George-Emil

Coordonator  
Asist. drd. ing. Huștiu Sofia

Iași, 2024

# Cuprins

<b>Listă de figuri</b>	<b>vi</b>
<b>1 Introducere</b>	<b>2</b>
1.1 Motivația . . . . .	2
1.2 Stadiul actual al literaturii . . . . .	2
1.3 Formularea problemei . . . . .	3
1.4 Structura lucrării . . . . .	4
<b>2 Resurse hardware și software</b>	<b>5</b>
2.1 Ce este o dronă? . . . . .	5
2.2 Resurse hardware . . . . .	7
2.2.1 Drona Crazyflie 2.0 . . . . .	7
2.2.2 Flow deck v2 . . . . .	7
2.2.3 Crazyradio PA . . . . .	8
2.2.4 Mediu de lucru . . . . .	8
2.2.5 Sistemul Loco Positioning . . . . .	9
2.2.6 Explicarea modului TDoA . . . . .	10
2.2.7 Inițializarea modului TDoA . . . . .	12
2.3 Resurse software . . . . .	13
2.3.1 Matlab . . . . .	13
2.3.2 Python . . . . .	13
2.4 Modalități de conectare la drona Crazyflie . . . . .	14
<b>3 Netezire prin curbe Bézier</b>	<b>16</b>
3.1 Scurt istoric . . . . .	16
3.2 Polinoame Bernstein . . . . .	16
3.3 Aspectul matematic al curbelor Bézier . . . . .	17
Curbe Bézier liniare . . . . .	18
Curbe Bézier quadratice . . . . .	18
Curbe Bézier cubice . . . . .	19
3.4 Exemplificarea curbelor Bézier . . . . .	19
3.5 Proprietăți ale curbelor Bézier . . . . .	21
3.6 Algoritmi . . . . .	21
3.6.1 Algoritmul Bézier clasic . . . . .	22
Explicarea algoritmului Bézier clasic . . . . .	22
3.6.2 Algoritmul Bézier extins . . . . .	24
Explicarea algoritmului Bézier extins . . . . .	24
<b>4 Rezultate experimentale</b>	<b>26</b>
4.1 Traiectorie formată din 7 puncte unghiulare . . . . .	26
4.1.1 Spațiu de lucru fără obstacole . . . . .	26
a) Traiectorie cu puncte unghiulare . . . . .	26
b) Traiectorie netezită prin curbe Bézier - metoda clasică . . . . .	27
c) Traiectorie netezită prin curbe Bézier - metoda extinsă . . . . .	27
d) Traiectorie în mediul real . . . . .	28

4.1.2	Spațiu de lucru cu obstacole . . . . .	30
a)	Traекторie cu puncte unghiulare . . . . .	30
b)	Traекторie netezită prin curbe Bézier - metoda clasică . . . . .	30
c)	Traекторie netezită prin curbe Bézier - metoda extinsă . . . . .	31
d)	Traекторie în mediul real . . . . .	32
4.2	Traекторie formată din 10 puncte unghiulare . . . . .	34
4.2.1	Spațiu de lucru fără obstacole . . . . .	34
a)	Traекторie cu puncte unghiulare . . . . .	34
b)	Traекторie netezită prin curbe Bézier - metoda clasică . . . . .	34
c)	Traectorie netezită prin curbe Bézier - metoda extinsă . . . . .	35
d)	Traectorie în mediul real . . . . .	36
4.2.2	Spațiu de lucru cu obstacole . . . . .	38
a)	Traectorie cu puncte unghiulare . . . . .	38
b)	Traectorie netezită prin curbe Bézier - metoda clasică . . . . .	38
c)	Traectorie netezită prin curbe Bézier - metoda extinsă . . . . .	39
d)	Traectorie în mediul real . . . . .	40
<b>5</b>	<b>Concluzii și Îmbunătățiri viitoare</b>	<b>42</b>
<b>6</b>	<b>Anexe</b>	<b>45</b>
6.1	Anexa 1. Cod în MATLAB. Algoritmul Bézier clasic . . . . .	45
6.2	Anexa 2. Cod în MATLAB. Algoritmul Bézier extins . . . . .	48
6.3	Anexa 3. Cod in Python . . . . .	53

## DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ

Subsemnatul MEREUȚĂ GEORGE-EMIL,  
legitimat cu CJ seria 02 nr. 250412, CNP 6010424 170020  
autorul lucrării METODĂ PENTRU NEȚEZRIREA TRAJECTOPIEI UNEI  
DRONE

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul de studii AUTOMATICĂ și INFORMATICĂ APLICATĂ organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea iULIE 2024 a anului universitar 2023-2024, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data  
30.06.2024

Semnătura  
Elieută

# Listă de figuri

2.1	Sistemul de referință inerțial . . . . .	5
2.2	Sistemul de referință Body-fixed . . . . .	5
2.3	Configurația "+" . . . . .	6
2.4	Configurația "X" . . . . .	6
2.5	Unghiurile de rotație ale dronei Crazyflie 2.0 . . . . .	6
2.6	Drona Crazyflie 2.0 . . . . .	7
2.7	Flow deck v2 . . . . .	8
2.8	Dongle radio USB Crazyradio PA . . . . .	8
2.9	Mediul de lucru . . . . .	9
2.10	Ancora . . . . .	10
2.11	Tag . . . . .	10
2.12	Localizarea unei drone folosind 3 ancore pentru modul TDoA . . . . .	11
2.13	Localizarea unei drone folosind 3 ancore pentru modul TDoA în exteriorul în-fășurării convexe . . . . .	11
2.14	Ancora în spațiul nostru de lucru . . . . .	12
2.15	Sistemul Loco Positioning cu 8 Ancore . . . . .	12
2.16	Poziționarea ancorelor în modul TDoA 2 în spațiul nostru de lucru . . . . .	13
2.17	Interfața CFClient . . . . .	14
2.18	CFClient din mașina virtuală . . . . .	15
3.1	Polinoamele de bază Bernstein de gradul $n = 4$ . . . . .	17
3.2	Curba Bézier liniară . . . . .	18
3.3	Curba Bézier quadratică . . . . .	19
3.4	Curba Bézier cubică . . . . .	19
3.5	Definirea celor 3 puncte . . . . .	20
3.6	Formarea poligonului de control . . . . .	20
3.7	Foarmarea nivelului de interpolare . . . . .	20
3.8	Definirea punctului pentru trasarea curbei . . . . .	20
3.9	Trasarea curvei Bezier . . . . .	20
4.1	Traекторie formată din 7 puncte unghiulare privită de sus în simulare într-un spațiu fără obstacole . . . . .	26
4.2	Traекторie formată din 7 puncte unghiulare privită dintr-o parte în simulare într-un spațiu fără obstacole . . . . .	26
4.3	Traекторie parțial netezită privită de sus în simulare într-un spațiu fără obstacole . . . . .	27
4.4	Traекторie parțial netezită privită dintr-o parte în simulare într-un spațiu fără obstacole . . . . .	27
4.5	Traекторie totală netezită privită de sus în simulare într-un spațiu fără obstacole . . . . .	28
4.6	Traекторie totală netezită privită dintr-o parte în simulare într-un spațiu fără obstacole . . . . .	28
4.7	Traекторie parțial netezită privită de sus în mediul real într-un spațiu fără obstacole . . . . .	29
4.8	Traекторie parțial netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole . . . . .	29
4.9	Traекторie totală netezită privită de sus în mediul real într-un spațiu fără obstacole . . . . .	29
4.10	Traекторie totală netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole . . . . .	29

4.11 Traiectorie formată din 7 puncte unghiulare privită de sus în simulare într-un spațiu cu obstacole . . . . .	30
4.12 Traiectorie formată din 7 puncte unghiulare privită dintr-o parte în simulare într-un spațiu cu obstacole . . . . .	30
4.13 Traiectorie parțial netezită privită de sus în simulare într-un spațiu cu obstacole . . . . .	31
4.14 Traiectorie parțial netezită privită dintr-o parte în simulare într-un spațiu cu obstacole . . . . .	31
4.15 Traiectorie totală netezită privită de sus în simulare într-un spațiu cu obstacole . . . . .	32
4.16 Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu cu obstacole . . . . .	32
4.17 Traiectorie parțial netezită privită de sus în mediul real într-un spațiu cu obstacole . . . . .	33
4.18 Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole . . . . .	33
4.19 Traiectorie totală netezită privită de sus în mediul real într-un spațiu cu obstacole . . . . .	33
4.20 Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole . . . . .	33
4.21 Traiectorie formată din 10 puncte unghiulare privită de sus în simulare într-un spațiu fără obstacole . . . . .	34
4.22 Traiectorie formată din 10 puncte unghiulare privită dintr-o parte în simulare într-un spațiu fără obstacole . . . . .	34
4.23 Traiectorie parțial netezită privită de sus în simulare într-un spațiu fără obstacole . . . . .	35
4.24 Traiectorie parțial netezită privită dintr-o parte în simulare într-un spațiu fără obstacole . . . . .	35
4.25 Traiectorie totală netezită privită de sus în simulare într-un spațiu fără obstacole . . . . .	36
4.26 Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu fără obstacole . . . . .	36
4.27 Traiectorie parțial netezită privită de sus în mediul real într-un spațiu fără obstacole . . . . .	37
4.28 Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole . . . . .	37
4.29 Traiectorie totală netezită privită de sus în mediul real într-un spațiu fără obstacole . . . . .	37
4.30 Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole . . . . .	37
4.31 Traiectorie formată din 10 puncte unghiulare privită de sus în simulare într-un spațiu cu obstacole . . . . .	38
4.32 Traiectorie formată din 10 puncte unghiulare privită dintr-o parte în simulare într-un spațiu cu obstacole . . . . .	38
4.33 Traiectorie parțial netezită privită de sus în simulare într-un spațiu cu obstacole . . . . .	39
4.34 Traiectorie parțial netezită privită dintr-o parte în simulare într-un spațiu cu obstacole . . . . .	39
4.35 Traiectorie totală netezită privită de sus în simulare într-un spațiu cu obstacole . . . . .	40
4.36 Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu cu obstacole . . . . .	40
4.37 Traiectorie parțial netezită privită de sus în mediul real într-un spațiu cu obstacole . . . . .	41
4.38 Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole . . . . .	41
4.39 Traiectorie totală netezită privită de sus în mediul real într-un spațiu cu obstacole . . . . .	41
4.40 Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole . . . . .	41

# Metodă pentru netezirea traectoriei unei drone

## Rezumat

Industria dronelor a primit o atenție considerabilă în ultimii ani, extinzându-se rapid la nivel global și devenind tot mai utilizată pentru o varietate de sarcini. Dronele Crazyflie sunt în prezent printre cele mai populare și recunoscute drone, cunoscute pentru fiabilitatea lor și pentru îmbunătățirile continue care le sporesc performanța și funcționalitatea.

Acest studiu se concentrează pe implementarea unei metode eficiente de netezire a traectoriei unui nanoquadcopter Crazyflie 2.0, cu scopul de a optimiza deplasarea acestuia prin eliminarea punctelor unghiulare care nu permit ca acceleratia să fie continuă, garantând astfel o navigare fluidă și sigură. În cadrul acestui proiect, s-a ales să se utilizeze metoda Bézier Curve pentru îmbunătățirea comportamentului dronei Crazyflie 2.0 prin crearea unei traectorii fluide care trece prin toate punctele de interes. Pentru determinarea poziției reale a quadcopterului, s-a folosit Loco Positioning System comparând datele din mediul real cu cele din simulare. Simulațiile s-au obținut în mediul MATLAB, iar transmiterea coordonatelor a fost făcută prin Python, facilitând astfel comunicarea directă între o unitate de calcul și Crazyflie. Evaluarea rezultatelor a fost realizată prin comparații între traectoria exactă și cea netezită, atât în simulări, cât și în mediul real.

# 1 Introducere

## 1.1 Motivația

De-a lungul timpului, numeroși algoritmi de planificare a traectoriei au fost dezvoltăți pentru roboții mobili, având scopul de a-i ghida de la o poziție de start la o destinație pe o hartă dată. Acești algoritmi generează traectorii care mențin o distanță sigură față de obstacole. Totuși, majoritatea acestor planificatori produc căi cu viraje ascuțite și unghiulare, ceea ce poate cauza încetiniri sau opriri neașteptate, afectând fluiditatea mișcării robotului. Din acest motiv, este preferabilă o traectorie netedă, care permite robotului să se deplaseze cu o viteză aproape constantă, asigurând astfel o operare mai eficientă și mai sigură. Acest aspect este deosebit de important în situațiile în care robotul transportă obiecte fragile, periculoase sau valoroase, deoarece virajele ascuțite pot prezenta riscuri semnificative pentru aceste încărcături. Generarea unor astfel de traectorii netede necesită luarea în considerare a obstacolelor statice și dinamice, precum și a altor constrângeri, cum ar fi curbele fezabile, dimensiunile robotului și ale benzii și viteza de deplasare. De asemenea, algoritmii trebuie să fie capabili să se adapteze la schimbările neașteptate din mediul înconjurător și să optimizeze continuu traectoria pentru a menține performanța și siguranța operațiunii [1], [2].

## 1.2 Stadiul actual al literaturii

Un UAV este adesea definit ca o aeronavă autonomă, capabilă să zboare și să rămână în aer fără necesitatea unui pilot uman la bord. Aceste vehicule oferă o eficiență superioară în operațiuni, comparativ cu sistemele echivalente cu echipaj uman, și permit desfășurarea de misiuni critice fără riscuri pentru viața umană. UAV-urile pot fi controlate de la distanță, primind comenzi de la o stație de bază terestră (BS) prin telecomandă. În plus, aceste vehicule sunt echipate pentru a executa operațiuni de control autonom, folosind sisteme de autopilot și o varietate de senzori, inclusiv sisteme de poziționare globală (GPS) și unități de măsurare inerțială (IMU) [3].

În ultimele decenii, vehiculele aeriene fără pilot (UAV), cunoscute și sub denumirea de drone, au avansat considerabil și au câștigat o popularitate crescândă la nivel mondial. Inițial utilizate în principal în sectorul militar pentru misiuni de recunoaștere și supraveghere [4], aceste tehnologii s-au extins rapid în diverse alte domenii. În continuare, voi prezenta câteva exemple concrete ale utilizării dronei în diverse contexte actuale.

În cazul dezastrelor naturale precum uraganele, tornadele, cutremurele sau inundațiile, dronele sunt utilizate pentru căutarea supraviețuitorilor. Capacitatea lor de a oferi o perspectivă generală asupra situației și de a localiza persoanele aflate în pericol este crucială pentru dezvoltarea unui plan de salvare și pentru identificarea celor care au nevoie urgentă de ajutor. Cercetătorii de la Universitatea din Zurich au dezvoltat un model special de dronă pliabilă, conceput pentru a opera în zonele afectate de dezastre. Aceasta poate schimba formă pentru a se strecu prin crăpături și spații înguste, ceea ce poate îmbunătăți semnificativ eficiența operațiunilor de salvare [5]. În agricultură, dronele facilitează monitorizarea și colectarea rapidă a datelor pe suprafețe mari de teren agricol. De la identificarea bolilor la gestionarea buruienilor,

capacitatea de a obține și analiza rapid informațiile poate contribui semnificativ la creșterea productivității fermelor. Această abilitate devine din ce în ce mai crucială în contextul presiunii crescute de a produce mai multă hrană cu resurse limitate [6] [7]. Dronelor le revine și un rol important în conservarea faunei sălbaticice. Monitorizarea populațiilor de animale sălbaticice este dificilă pentru oamenii pe teren, dar dronele pot urmări grupuri de animale în mișcare, de la orangutani în Borneo până la bizoni în Câmpurile Mari. Aceasta permite conservaționistilor să obțină o înțelegere mai profundă a sănătății speciilor și a ecosistemelor lor. De asemenea, dronele sunt extrem de utile în combaterea braconajului în Asia și Africa [8] [9]. În plus, dronele sunt utilizate în proiecte globale de silvicultură, scanând solul pădurilor arse și dispersând capsule de semințe ce conțin semințe, îngrășăminte și nutrienți pentru a ajuta la regenerarea copacilor din cenușă. Începând cu anii '90, aproximativ 300 de milioane de acri de păduri au fost defrișate. Ceea ce ar dura oamenilor aproximativ 300 de ani pentru a reîmpăduri poate fi realizat mult mai eficient cu tehnologia dronelor pentru plantarea semințelor [10].

Astfel, interesul academic pentru drone a crescut exponential, reflectând potențialul lor vast și promițător. Cercetătorii au dezvoltat o multitudine de algoritmi și metode pentru a optimiza capacitatele de navigație, control și ghidare ale UAV-urilor, cu scopul de a le îmbunătăți eficiența și fiabilitatea în diverse aplicații. Această dezvoltare continuă subliniază importanța și versatilitatea UAV-urilor în diverse domenii, demonstrând impactul lor semnificativ asupra tehnologiei moderne și societății.

Există o serie de articole și lucrări științifice care explorează diverse metode de netezire a traectoriei dronei în diferite medii de lucru. În continuare, voi oferi o scurtă prezentare a câtorva dintre aceste lucrări relevante pe care le-am studiat în legătură cu acest subiect.

Această teză [11], propune o nouă metodă de planificare a traectoriei netezite folosind curbele geometrice NURBS (Non Uniform Rational Basis-Splines), concentrându-se pe explorarea suprafețelor planetare. Această abordare este inspirată de algoritmi de navigație și planificare utilizati de roverele spațiale NASA și ESA. În următoarea lucrare [12], realizată în strânsă colaborare cu Politehnica din Torino, se dorește dezvoltarea unei noi metodologii pentru optimizarea planificării traseului UAV-urilor. Accentul principal este pe minimizarea razei de viraj și maximizarea ratei de urcare, pentru a crea trasee fezabile, adecvate pentru diverse tipuri de aeronave cu aripă fixă. S-au investigat două abordări principale: curbele Bezier și curbele Dubin. După o analiză detaliată, s-a optat pentru curbele Dubin ca fiind cele mai potrivite pentru implementarea noastră. De asemenea, s-a propus un model pentru optimizarea consumului de energie al dronelor.

### 1.3 Formularea problemei

Această lucrare se concentrează pe implementarea unei metode eficiente de netezire a traectoriei pentru drona Crazyflie 2.0, având ca scop optimizarea deplasării prin eliminarea punctelor unghiulare care afectează continuitatea accelerării, asigurând astfel o navigare fluidă și sigură într-un spațiu de lucru cunoscut. Acest spațiu de lucru va conține obstacole, iar drona trebuie să urmeze cu precizie traectoria netezită impusă, evitând în același timp obstacolele. Vor fi testați algoritmi pentru generarea traectoriilor netezite, iar aceste traectorii vor fi evaluate prin parcurgerea lor de către drona Crazyflie în mediile de lucru menționate anterior.

## 1.4 Structura lucrării

Această lucrare este structurată în cinci capituloare. Capitolul 2 oferă detalii despre drona Crazyflie 2.0 și echipamentele auxiliare atașate acesteia, descriind, de asemenea, sistemul Loco Positioning, cu o explicare detaliată a celor trei moduri de operare, în special modul TDoA, și prezentând cele două metode de conectare între PC și dronă. Capitolul 3 discută metoda aleasă pentru netezirea traекторiei dronei, și anume curbele Bezier, explicând ce sunt acestea și conceptele care stau la baza lor. La finalul capitolului 3 sunt prezentate algoritmii care vor fi utilizati în partea de testare, fiecare algoritm fiind explicat în detaliu pe baza unui pseudocod propriu. Capitolul 4 prezintă rezultatele experimentale atât din simulare, cât și din mediul real, explicate amănunțit. Lucrarea se încheie cu capitolul 5, unde sunt prezentate concluziile și îmbunătățirile posibile pentru cercetările viitoare.

## 2 Resurse hardware și software

### 2.1 Ce este o dronă?

Dronele sunt aparate zburătoare automate care cuprind o gamă largă de vehicule aeriene fără pilot (UAV-uri), capabile să parcurgă distanțe mari, de la mii de kilometri pentru UAV-uri mari, până la zboruri în spații restrânse pentru dronele mici. Aceste vehicule aeriene operează fără intervenția directă a unui pilot uman, putând efectua zboruri controlate de la distanță sau autonome și transportând diverse tipuri de sarcini, fie letale, fie non-letale.

Nu sunt considerate drone vehiculele balistice sau semi-balistică, rachetele de croazieră, proiectile de artilerie, torpilele, minele și sateliții. Dimensiunile și echipamentele instalate pe drone variază în funcție de tipul și scopul misiunilor lor de zbor. Avantajele semnificative ale dronelor au stimulat numeroase studii concentrate pe optimizarea și îmbunătățirea performanțelor acestora.

Un quadcopter, numit și quadrotor, este un vehicul aerian fără pilot (UAV) cu patru rotoare. Aceste UAV-uri fac parte din categoria vehiculelor cu decolare și aterizare verticală (VTOL). Dispunerea rotoarelor este în formă de pătrat, la distanțe egale față de centrul de masă al aparatului. Controlul vitezei rotoarelor permite quadcopterului să execute diverse manevre, să se mențină în aer, să decoleze și să aterizeze [13].

Pozitia quadcopterului poate fi exprimată prin coordonatele  $x$ ,  $y$  și  $z$ , care reprezintă distanța față de un punct fixat,  $O_i$ , pe Pământ, cunoscut și ca **sistem de referință inertial** sau sistem global, figura 2.1. Orientarea quadcopterului este determinată de unghiurile prin axele  $x'$ ,  $y'$  și  $z'$  având originea în centrul de greutate al dronei. Aceasta este denumit **sistemul de referință body-fixed**, figura 2.2.

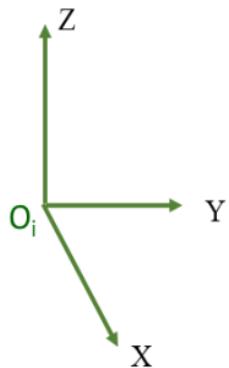


Figure 2.1: Sistemul de referință inertial

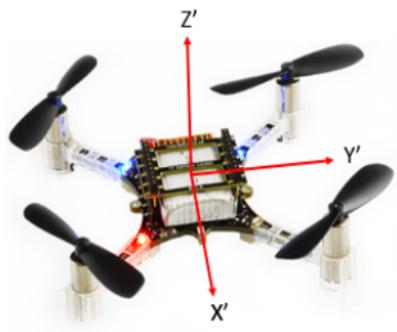


Figure 2.2: Sistemul de referință Body-fixed

În domeniul aeronautic, există o convenție comună conform căreia altitudinea pozitivă este orientată în jos, cu axa Y orientată spre est și axa X orientată spre nord, cunoscută sub denumirea de sistem NED (North, East, Down). În contrast, drona Crazyflie 2.0 urmează o convenție în care altitudinea pozitivă este orientată în sus, definind astfel sistemul ENU (East, North, Up).

Un alt aspect important este cunoașterea configurației de zbor a quadcopterului. Există două tipuri de configurații: configurația "+" din figura 2.3 și configurația "X" din figura 2.4. Diferența dintre ele constă în orientarea axelor X-Y în raport cu brațele quadcopterului.

În contextul actual al quadcopterelor, configurația "X" este preferată în locul configurației "+", iar Crazyflie 2.0 este setat implicit în modul "X" [14].

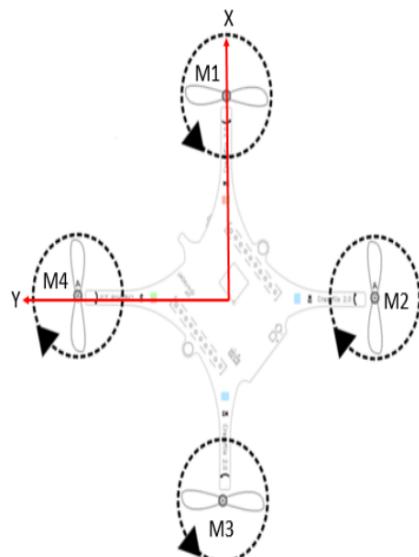


Figure 2.3: Configurația "+"

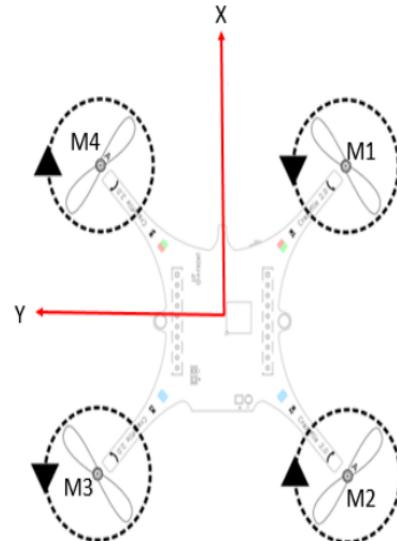


Figure 2.4: Configurația "X"

În timpul zborului unui quadcopter, sunt patru dimensiuni principale de control: roll (înclinare laterală), pitch (înclinare longitudinală), yaw (rotație în jurul axei verticală) și thrust (propulsie), figura 2.5.

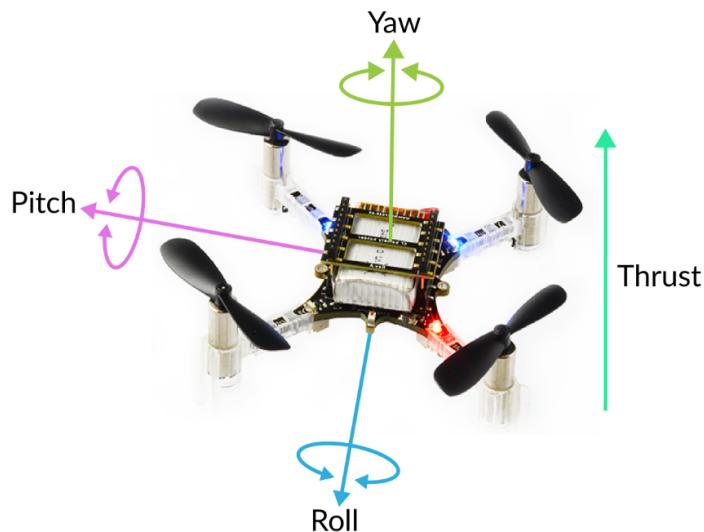


Figure 2.5: Unghiurile de rotație ale dronei Crazyflie 2.0

**Roll** reprezintă rotația în jurul axei x care trece prin quadcopter de la partea din spate spre partea din față. Această mișcare încină corpul dronei și o deplasează la stânga sau la dreapta. **Pitch** semnifică rotația în jurul axei Y care traversează quadcopterul de la stânga la dreapta. Aceasta încină Crazyflie și îl mișcă în direcția înainte sau înapoi. **Yaw** reprezintă rotația în jurul axei Z. Această mișcare rotește quadcopterul spre stânga sau spre dreapta și

este utilizată pentru schimbarea direcției de zbor prin orientarea frontală a Crazyflie în diferite direcții. **Thrust** controlează înălțimea sau altitudinea la care se află Crazyflie-ul [15].

## 2.2 Resurse hardware

### 2.2.1 Drona Crazyflie 2.0

Drona Crazyflie 2.0, figura 2.6, este un nano quadcopter produs de către compania Bitcraze. Este echipată cu 4 motoare de curent continuu fără miez și 4 elice din plastic de 45 mm. Unitătile de propulie sunt atașate de cadru placii de circuite al UAV. Acest tip de dronă are dimensiunea de 92 milimetri de la motor la motor cu o înălțime de 29 milimetri și o greutate totală de 27 de grame. Drona Crazyflie este prevăzută și cu o baterie LiPo de 240 mAh care poate furniza energie pentru 7 minute de zbor continuu.



Figure 2.6: Drona Crazyflie 2.0

Drona Crazyflie este prevăzută cu 2 microcontrolere: unul principal (ARM STM32F405 Cortex-M4 pe 32 de biti care are o viteză de 168 MHz) și unul adițional (nRF51822 ARM Cortex-M0 de 32 MHz pentru gestionarea eficientă a comunicațiilor radio și a nivelului de putere folosit). Microcontrolerul principal comunică cu stația de la sol (calculator PC cu USB dongle) și este controlat prin Crazyradio PA într-o rază de acțiune de până la 1 km. Crazyflie 2.0 poate fi controlată și utilizând un smartphone sau o tabletă prin intermediul Bluetooth.

Echipamentul de la bordul dronei include un senzor giroscopic, accelerometru, magnetometru și un senzor de presiune cu precizie ridicată. Firmware-ul acestei drone este bazat pe sistemul de operare open-source FreeRTOS. Software-ul implicit este Crazyflie Python Client (CFClient) scris în limbajul de programare Python și oferă o vizualizare în timp real la cei mai importanți parametri de zbor și a indicatorilor de altitudine [16].

### 2.2.2 Flow deck v2

Flow deck-ul v2, figura 2.7, este un echipament de expansiune ce poate fi montat pe drona Crazyflie 2.0 pentru a îmbunătăți performanțele de zbor. Acest echipament oferă dronei abilitatea de a sesiza mișcarea dronei, indiferent de direcție. Este alcătuit dintr-un senzor VL53L1x ToF și un senzor de flux optic PMW3901. Senzorul VL53L1x ToF măsoară distanța dintre dronă

și sol cu precizie ridicată, reușind să măsoare distanțe de până la 4 metri și câțiva milimetri, în funcție de suprafața solului și condițiile de lumină. Senzorul de flux optic PMW3901 măsoară mișcările dronei în relație cu suprafețele mate [17].



Figure 2.7: Flow deck v2

### 2.2.3 Crazyradio PA

Crazyradio PA, figura 2.8, este un dongle radio USB cu o rază lungă de acțiune bazat pe un chip, nRF24L01+, furnizat de compania Nordic Semiconductor ce conferă posibilitatea conexiuni la distanță dintre drona Crazyflie 2.0 și calculator. Chip-ul nRF24L01+ furnizează 125 de canale radio prin care poate trimite și primi de la drona pachete de date de până la 32 bytes, rata de transmisie putând avea urmatoarele valori, în funcție de setările făcute în CFClient de către utilizator: 2Mbps, 1Mbps sau 250 Kps [18].



Figure 2.8: Dongle radio USB Crazyradio PA

### 2.2.4 Mediu de lucru

Planificarea traiectoriilor dronelor necesită o cunoaștere detaliată a mediului în care acestea își desfășoară zborurile. Spațiul de lucru utilizat în experimentele prezentate în această lucrare, figura 2.9, este un spațiu închis, având forma unui dreptunghi cu dimensiunile de 4.05 metri lungime(axa X) și 3.54 metri lățime(axa Y). Înălțimea maximă(axa Z) la care au fost efectuate experimentele este de 3 metri. Acum spațiu conține de asemenea cele 8 ancore ale

sistemului Loco Positioning, utilizate pentru extragerea poziției în timp real a dronei. În continuare, vom oferi detalii despre funcționarea acestui sistem de poziționare.



Figure 2.9: Mediul de lucru

### 2.2.5 Sistemul Loco Positioning

Sistemul Loco Positioning este un sistem ce se bazează pe unde radio de bandă largă, utilizate pentru determinarea poziției 3D absolute a obiectelor într-un spațiu dat. Acest sistem poate fi comparat cu un sistem GPS într-o formă miniaturală.

Structura de bază a sistemului Loco Positioning este compusă dintr-un set de ancore (Anchors), figura 2.10, amplasate în spațiu (similar cu poziționarea sateliștilor în cazul GPS-ului), care servesc drept puncte de referință. În același mod, componenta sistemului constă din unul sau mai multe Tag-uri (asemănătoare receptorilor GPS), figura 2.11, atașate obiectului ce urmează să fie urmărit. Prin transmiterea undelor radio scurte la frecvențe înalte între Anchors și Tags, sistemul Loco Positioning măsoară distanța de la fiecare Anchor la Tag și calculează poziția Tag-ului. Toate datele necesare pentru calcularea poziției sunt integrate în Tag, permitând estimarea poziției chiar pe drona Crazyflie, fără necesitatea unui calculator extern.

Sistemul Loco Positioning conține 3 moduri diferite de poziționare: Two Way Ranging (TWR), Time Difference of Arrival 2 (TDoA 2) and Time Difference of Arrival 3 (TDoA). În aceasta lucrare am folosit TDoA 2.

În **modul TWR**, tag-ul trimite semnale către ancore secvențial, permitând măsurarea distanței dintre tag-uri și ancore. Teoretic, sunt necesare minim 4 ancore pentru a calcula poziția 3D a unui tag, dar în practică sunt recomandate 6 pentru redundanță și acuratețe. Acest mod este cel mai precis și funcționează chiar și atunci când tag-ul sau Crazyflie părăsește spațiul delimitat de ancore. Tag-ul comunică activ cu ancorele într-un mod cronometrat, permitând poziționarea unui singur tag sau Crazyflie cu un maxim de 8 ancore.

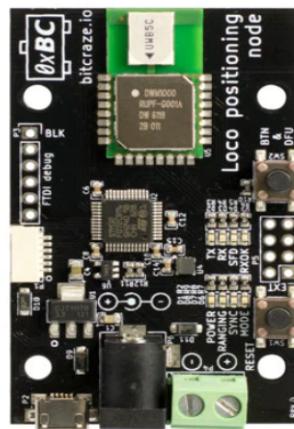


Figure 2.10: Ancora



Figure 2.11: Tag

În **modul TDoA 2**, sistemul de ancore transmite continuu pachete de sincronizare. Tagurile, ascultând aceste pachete, pot calcula distanță relativă față de două ancore prin măsurarea diferenței de timp a sosirii pachetelor. Aceste informații permit determinarea poziției 3D a dronei în spațiu. În acest mod, tag-urile ascultă pasiv, permitând pozitionarea simultană a unui număr nelimitat de tag-uri sau drone Crazyflie, fiind ideal pentru formarea plutoanelor. În acest mod, sistemul de ancore este sincronizat și împărțit în sloturi de timp, fiind limitat la 8 ancore.

Comparativ cu modul TWR, TD0A 2 necesită ca tag-urile să se afle în interiorul sau foarte aproape de spațiul delimitat de ancore pentru a funcționa corect. Aceasta înseamnă că TD0A 2 oferă cea mai bună performanță atunci când sunt utilizate 8 ancore amplasate în colțurile spațiului de zbor, asigurând astfel o acuratețe și precizie comparabilă cu TWR.

**Modul TD0A 3** reprezintă o extensie a capabilităților modului TD0A 2, permitând utilizarea unui număr nelimitat de taguri și ancore. Spre deosebire de TD0A 2, unde se folosea o schemă de sloturi de timp, TD0A 3 utilizează un program de transmisie randomizat. Aceasta schimbare permite adăugarea unui număr mai mare de ancore, extinzând sistemul pentru a acoperi spații mai mari sau multiple încăperi fără a necesita linie de vedere directă între ancore. Acest lucru îmbunătățește robustețea sistemului și permite gestionarea dinamică a pierderii sau adăugării de ancore [19].

## 2.2.6 Explicarea modului TD0A

TD0A, sau Diferența de Timp de Sosire, se bazează pe ideea măsurării diferenței de distanță dintre două ancore. Modul TD0A are caracteristici geometrice care restricționează spațiul în care poziția poate fi determinată cu o precizie rezonabilă. Următorul exemplu demonstrează cum, utilizând modul TD0A, putem estimă poziția unei drone folosind 3 ancore.

Vom considera un exemplu în 2D pentru o vizualizare mai clară, deși aceleași principii se aplică și în 3D. Avem trei ancore, A0, A1 și A2, și drona Crazyflie, figura 2.12. Drona este localizată la coordonatele (0.0, -0.5). Liniile albastre, portocalii și verzi reprezintă locurile unde diferența de timp de sosire (TD0A) este constantă, iar punctul de intersecție al acestor linii indică poziția dronei. De exemplu, linia albastră reprezintă diferența de timp constantă între ancorele A0 și A1, ceea ce înseamnă că drona poate fi oriunde pe această linie. Linia portocalie arată diferența de timp între A0 și A2, iar linia verde între A1 și A2.

Liniile roșii reprezintă înfășurarea convexă a sistemului, definind spațiul delimitat de cele trei ancore.

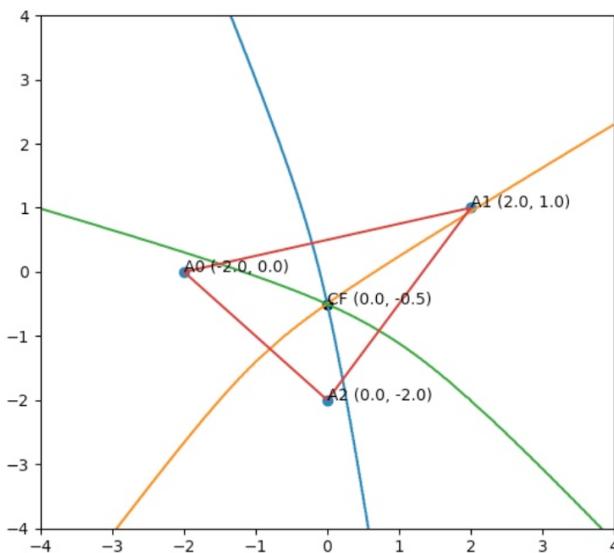


Figure 2.12: Localizarea unei drone folosind 3 ancore pentru modul TDoA

Pentru a calcula diferența de timp  $d_{i,j}$  dintre dronă și ancorele  $i$ , respectiv  $j$ , avem următoarea formulă:

$$d_{i,j} = \frac{\|\mathbf{p} - \mathbf{p}_i\| - \|\mathbf{p} - \mathbf{p}_j\|}{c} + n_j - n_i. \quad (3)$$

unde  $c$  - viteza luminii,  $\mathbf{p}$  - poziția dronăi,  $\mathbf{p}_i$ ,  $\mathbf{p}_j$  - poziția ancorei care transmite,  $n_j$  - zgomotele de transmisie și măsurare a celor două ancore  $i$  și  $j$  [20].

În a doua imagine, figura 2.13, Tag-ul dronăi se află în afara spațiului delimitat de cele trei linii ale modulului TDoA, la coordonatele  $(3.5, -0.3)$ . Se observă că liniile care indică posibilele poziții ale dronăi sunt paralele în punctul de intersecție, ceea ce face estimarea poziției dronăi mult mai dificilă.

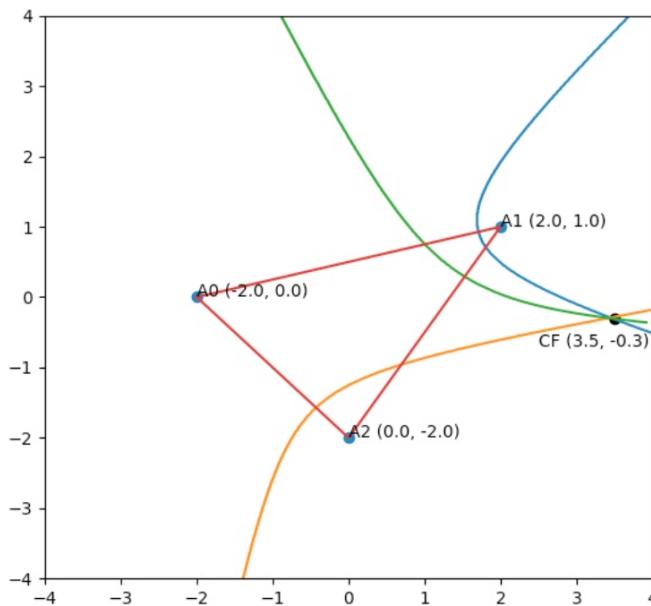


Figure 2.13: Localizarea unei drone folosind 3 ancore pentru modul TDoA în exteriorul înfășurării convexe

Acest exemplu demonstrează că, pentru a estima poziția unei dronăi cu o acuratețe ridicată, aceasta trebuie să se afle în înfășurarea convexă determinată de ancore. S-a observat că, pe

măsură ce drona se îndepărtează de această înfășurare, acuratețea estimării scade.

Prin urmare, în acest studiu, vom utiliza modul TDoA 2, având în vedere că spațiul nostru de lucru este restrâns și dorim o estimare cât mai precisă a poziției dronei, ceea ce nu poate fi obținut folosind modul TDoA 3. În continuare, voi explica cum se inițializează acest mod folosind interfața CFClient [21].

### 2.2.7 Inițializarea modului TDoA 2

Modul TDoA 2 necesită 8 ancore pentru a funcționa corect, fiind poziționate conform ilustrației 2.14. În configurația noastră cu 8 ancore, am plasat nodurile în colțurile unei cutii, deoarece estimarea poziției pentru TDoA funcționează optim în interiorul înfășurării convexe. Ordinea ID-urilor este crucială pentru TDoA 2, însă poate fi neglijată pentru TDoA 3. Ancorele sunt alimentate prin două fire, conform ilustrării din figura 2.15, cu o tensiune de 5-12V și un curent minim de 150mA.

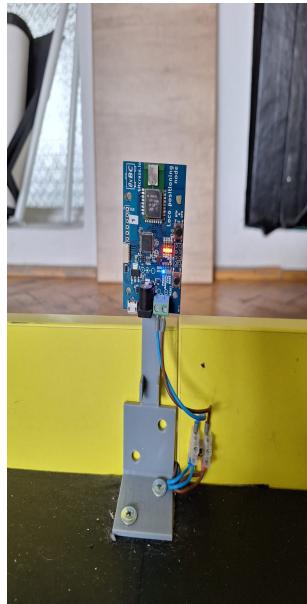


Figure 2.14: Ancora în spațiul nostru de lucru

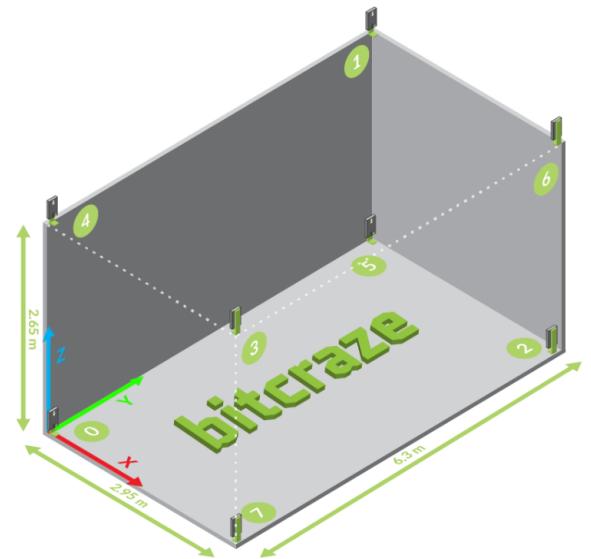


Figure 2.15: Sistemul Loco Positioning cu 8 Ancore

Pentru a verifica funcționarea sistemului Loco Positioning, vom folosi interfața CFClient. Vom plasa drona Crazyflie 2.0 în interiorul spațiului de lucru delimitat de ancorele sistemului. Deschidem interfața CFClient și realizăm conexiunea între PC și dronă. Pentru a evita interferențele cauzate de ancore, vom configura drona să comunice cu PC-ul în modul radio de 2 Mbit. Verificăm statusul ancorelor în tab-ul Loco Positioning. În secțiunea dedicată ancorelor, ar trebui să vedem tot atâtea casete verzi câte ancore avem. Dacă o casetă este roșie, înseamnă că drona Crazyflie nu poate comunica cu ancore respectivă. În acest caz, trebuie să verificăm dacă ancore este configurată corect sau dacă are probleme de alimentare. În exemplul nostru, figura 2.16, toate ancorele funcționează, fiind numerotate de la 0 la 7. Se poate observa că folosim modul TDoA 2, bifat în figură. Cerculețul albastru din imagine reprezintă drona Crazyflie, iar poziția sa în timp real poate fi văzută în secțiunea Configuration, având coordonatele (0.39, 0.50, 0.00) în acest exemplu [22].

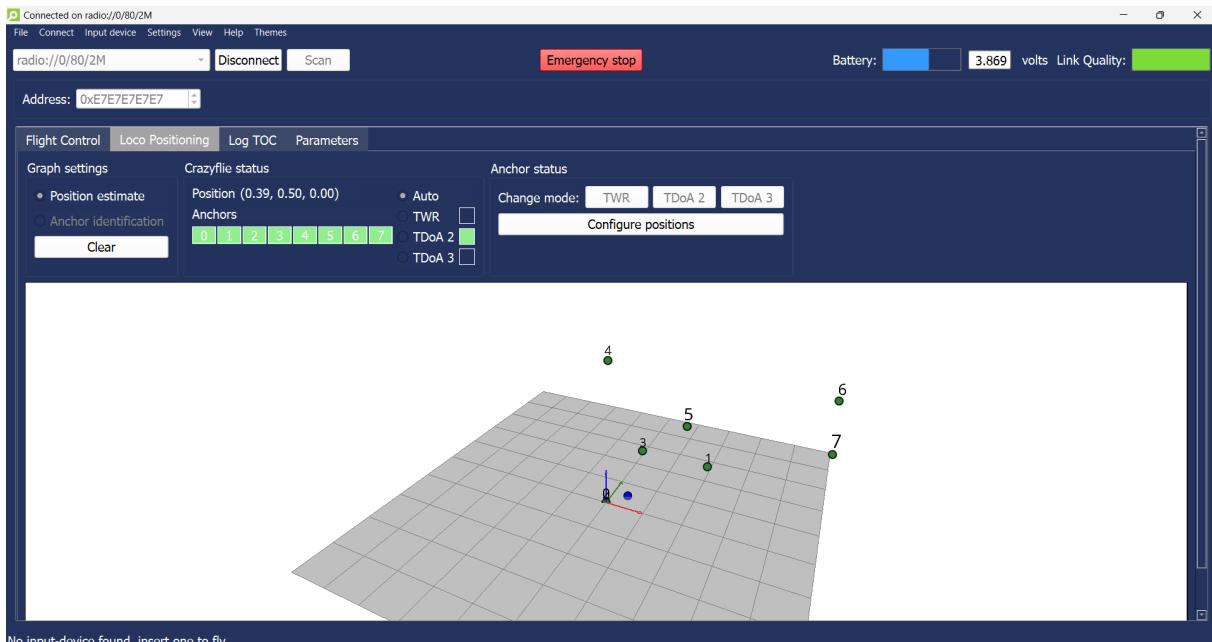


Figure 2.16: Poziționarea ancorelor în modul TDoa 2 în spațiul nostru de lucru

## 2.3 Resurse software

### 2.3.1 Matlab

MATLAB, dezvoltat de MathWorks, este o unealtă esențială în domeniul calculului științific și tehnic. Înființat în anii 1970 ca o interfață interactivă pentru EISPACK [41] și LINPACK [19], furnizează rutine pentru soluții ale valorilor proprii și sistemelor liniare. De atunci, s-a extins considerabil, integrând biblioteci numerice moderne precum ATLAS [46] și FFTW [23], și oferind kituri de instrumente pentru diverse aplicații, inclusiv matematică financiară, rețelele neuronale și teoria controlului. MATLAB include un limbaj interpretat încorporat, similar cu Fortran 90, și dispune de o indexare flexibilă a matricelor, ideală pentru rezolvarea problemelor matriceale. De asemenea, susține integrarea cu clase Java și biblioteci cu legături dinamice.

Popularitatea MATLAB se datorează ușurinței sale de utilizare și puterii sale grafice avansate, făcându-l un instrument de neprețuit pentru analiza datelor. Este larg utilizat în sălile de clasă pentru educație și este preferat de cercetători pentru dezvoltarea de sisteme complexe folosind scripturile și kiturile de instrumente MATLAB. MATLAB integrează capacitați numerice, simbolice și de vizualizare grafică de vârf într-un mediu de programare intuitiv, ceea ce îl face popular în rândul informaticienilor, oamenilor de știință din diverse domenii și inginerilor, în special în rândul specialiștilor în matematică computațională [23], [24].

### 2.3.2 Python

Python, dezvoltat de Guido Van Rossum la sfârșitul anilor 1980, este un limbaj de scripting puternic și interactiv, orientat pe obiecte, ideal pentru cei care încep să învețe programarea. Cu o sintaxă simplă, Python permite dezvoltarea unei game variate de aplicații, incluzând procesarea de text, crearea de site-uri web și jocuri video [25]. Este cunoscut pentru versatilitatea sa, acoperind domenii precum administrarea sistemelor și designul web, beneficiind de o gamă largă de biblioteci pentru cercetare științifică. Python combină modele de design orientate pe obiecte și funcționale, oferind o sintaxă intuitivă și facilitând crearea de cod reutilizabil. Popularitatea sa în rândul dezvoltatorilor se datorează capacitații de a rezolva probleme complexe cu mai

puțin cod decât alte limbaje precum C sau C++. Python devine rapid dominant în programare datorită capacitatea sale de dezvoltare rapidă și eficienței în transformarea codului lung în soluții concise [26].

## 2.4 Modalități de conectare la drona Crazyflie

Conecțarea la drona Crazyflie se poate realiza în două moduri:

- Direct din linia de comandă, accesând folderul **crazyflie-lib-python** și deschizând interfața **CFClient**.
- Prin intermediul mașinii virtuale create de compania Bitcraze, producătorul dronei Crazyflie, și accesând de acolo interfața **CFClient**.

Indiferent de metoda utilizată, pentru a stabili conexiunea dintre drona Crazyflie și PC, este necesar dispozitivul Crazyradio PA. Acesta se conectează la PC și permite comunicarea cu drona Crazyflie, funcționând ca o antenă care transmite date între PC și dronă în ambele sensuri.

Pe drona Crazyflie se pot rula doar programe scrise în limbajul de programare Python. Alte limbaje de programare nu sunt suportate de această dronă. Astfel, MATLAB este folosit pentru a realiza simulări ale traiectoriei dronei, iar Python este utilizat pentru a implementa traiectoriile în mediul real.

**Interfața CFClient**, figura 2.17 este folosită pentru controlul dronei Crazyflie, programarea firmware-ului, setarea parametrilor și înregistrarea datelor. Interfața principală este structurată în mai multe tab-uri, fiecare având un scop specific.

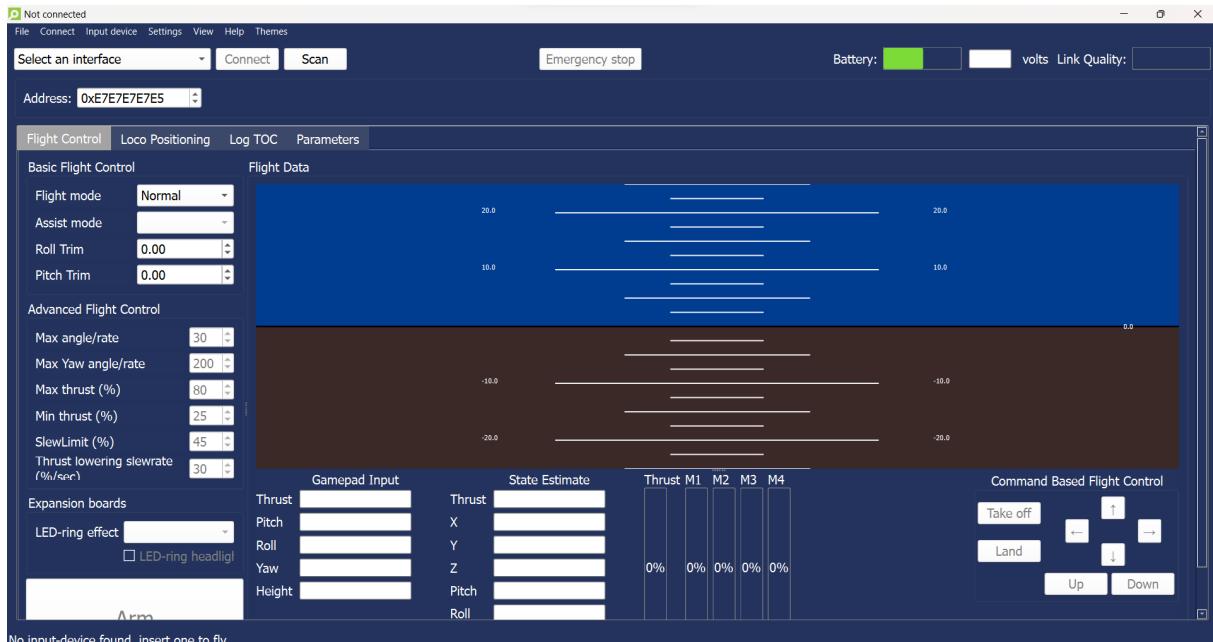


Figure 2.17: Interfața CFClient

**Mașina virtuală Bitcraze**, figura 2.18, conține toate cerințele necesare pentru rularea proiectelor Bitcraze, dezvoltarea lor și controlul dronei Crazyflie. Aceasta reprezintă o soluție mult mai ușoară și rapidă pentru a începe un proiect legat de dronele Crazyflie, deoarece elimină necesitatea instalării dependențelor și configurării mediului de dezvoltare.

## 2. Resurse hardware și software

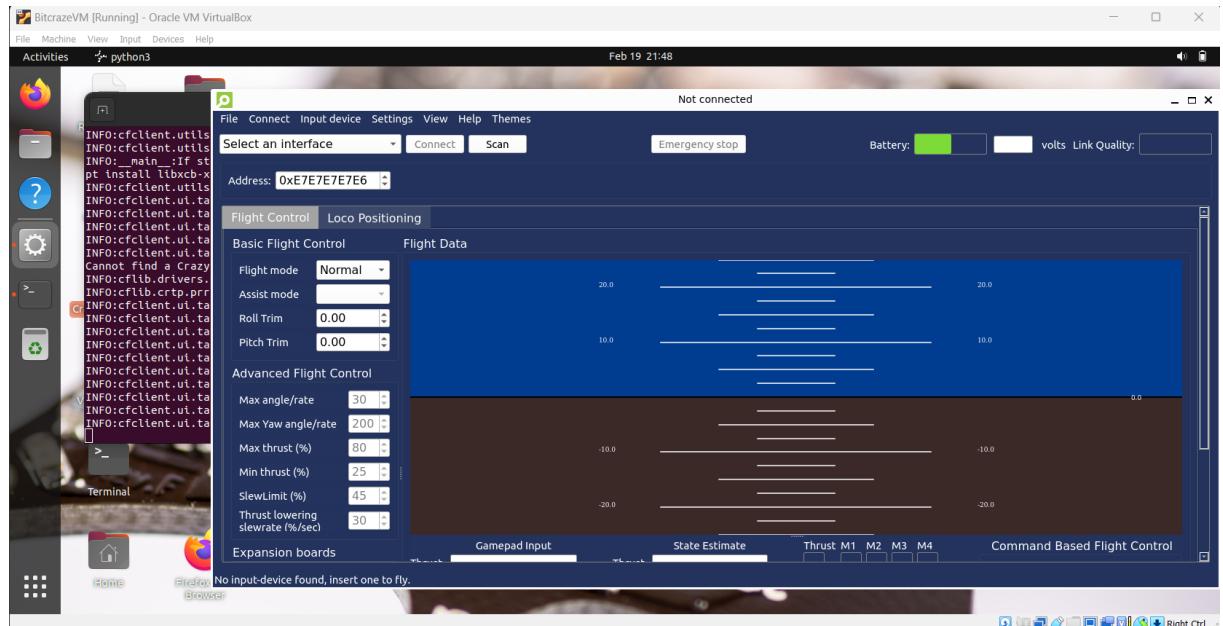


Figure 2.18: CFClient din mașina virtuală

În această lucrare, au fost testate ambele metode de conectare la drona Crazyflie. Metoda utilizată în capituloarele următoare este interfața CFClient direct din linia de comandă. Aceasta a fost aleasă deoarece interfața CFClient din mașina virtuală funcționează foarte greu, având dificultăți în menținerea unei conexiuni stabile, blocându-se frecvent. În schimb, metoda directă din linia de comandă nu a avut întârzieri, realizând conexiunea în 15 secunde, comparativ cu 1-3 minute în cazul mașinii virtuale. Comenzi trimise din interfața grafică sunt executate aproape instantaneu, spre deosebire de întârzierea de 1-2 minute din mașina virtuală. Astfel, s-a decis utilizarea interfeței CFClient din linia de comandă pentru a verifica buna funcționare a dronei pentru execuția codului Python.

## 3 Netezire prin curbe Bézier

Curbele Bézier reprezintă un concept esențial în grafica computerizată și în designul asistat de calculator. Ele sunt curbe parametrice polinomiale folosite pentru a genera traiectorii și forme netede în spațiu bidimensional sau tridimensional. Datorită versatilității lor remarcabile, aceste curbe sunt utilizate în diverse aplicații, cum ar fi designul auto, animația, modelarea 3D și grafica vectorială.

### 3.1 Scurt istoric

Polinoamele Bernstein, numite după matematicianul rus Sergei Natanovici Bernstein, formează baza matematică a curbelor Bézier. Introduse în 1912 pentru a demonstra teorema lui Weierstrass privind aproximarea funcțiilor continue, aceste polinoame au fost redescoperite în anii '60, când doi ingineri francezi au căutat modalități de reprezentare digitală a formelor complexe, din cauza costurilor ridicate ale modelării cu argilă. Paul de Faget de Casteljau, de la Citroën, a dezvoltat în 1959 un algoritm numeric stabil pentru curbele Bézier, cunoscut doar intern pentru o vreme. Pierre Ètienne Bézier de la Renault, care a publicat extensiv în anii '60 și '70, este responsabil pentru popularizarea curbelor Bézier. Deși inițial nu au menționat explicit baza Bernstein, aceasta este acum esențială în formularea curbelor Bézier [27].

### 3.2 Polinoame Bernstein

Polinoamele Bernstein sunt o familie de polinoame esențiale pentru teoria aproximării funcțiilor și pentru reprezentarea curbelor și suprafețelor în grafică computerizată, fiind deosebit de importante pentru definirea curbelor Bézier. Aceste polinoame constituie baza graficiei vectoriale pe calculator.

Pentru  $i = 0, \dots, n$ , definim elementele de bază Bernstein astfel:

$$B_i^n(x) = \binom{n}{i} x^i (1-x)^{n-i} \quad (3.1)$$

unde elementul  $B_i^n(x)$  se referă la polinomul de bază Bernstein de ordin  $n$ . Combinându-se, acestea formează polinoame de forma:

$$P(x) = \sum_{i=0}^n c_i B_i^n(x) = \sum_{i=0}^n c_i \binom{n}{i} x^i (1-x)^{n-i} \quad (3.2)$$

unde  $c_i \in \mathbb{R}$  sunt constante.

În figura 3.1, avem funcțiile de bază pentru  $n = 4$ , unde  $n$  reprezintă gradul polinomului. Atracția polinoamelor Bernstein derivă din anumite proprietăți intrinseci ale acestor funcții de bază [28].

Polinoamele Bernstein sunt **pozitive** în intervalul  $(0, 1)$ . Pentru orice  $x \in (0, 1)$ :

$$B_n^i(x) > 0.$$

Polinoamele sunt **simetrice**. Polinomul Bernstein de ordinul  $i$  și gradul  $n$  este egal cu polinomul Bernstein de ordinul  $n - i$  și gradul  $n$ :

$$B_n^i(x) = B_n^{n-i}(x).$$

Ele formează o **partiție de unitate**. Pentru orice  $n$ , suma tuturor polinoamelor Bernstein de grad  $n$  este egală cu 1 pentru toate  $x \in \mathbb{R}$ :

$$\sum_{i=0}^n B_n^i(x) = \sum_{i=0}^n \binom{n}{i} x^i (1-x)^{n-i} = (x + (1-x))^n = 1^n = 1.$$

Ele satisfac o **formulă de recurență**. Notând identitatea  $\binom{n+1}{i} = \binom{n}{i-1} + \binom{n}{i}$ , putem scrie:

$$B_{n+1}^i(x) = x B_n^{i-1}(x) + (1-x) B_n^i(x)$$

unde  $B_n^1 = B_n^{n+1} = 0$  și  $B_0^0 = 1$ .

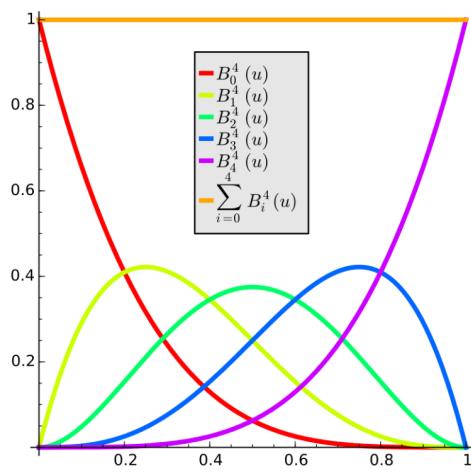


Figure 3.1: Polinoamele de bază Bernstein de gradul  $n = 4$

### 3.3 Aspectul matematic al curbelor Bézier

O curbă Bézier este determinată de un set de puncte de control  $P_0$  până la  $P_n$ , unde  $n$  indică gradul curbei. Pentru o curbă de gradul  $n$ , există  $n + 1$  puncte de control, iar primele și ultimele puncte sunt întotdeauna capetele curbei. Punctele intermediare nu se află neapărat pe curbă. Curba Bézier are urmatoare formă generală:

$$\mathbf{B}(t) = \sum_{i=0}^n b_{i,n}(t) \mathbf{P}_i, \quad 0 \leq t \leq 1 \tag{3.3}$$

, unde

- $P_i$  - punctele de control pentru curba Bézier

- $t$  - parametrul de interpolare care controlează modul în care punctele de control influențează forma și traiectoria curbei. Cu alte cuvinte, " $t$ " este un parametru care variază între 0 și 1 și controlează poziția de-a lungul curbei. Când " $t$ " este 0, punctul corespunde primului punct de control al curbei (de obicei numit  $P_0$ ), iar când " $t$ " este 1, punctul corespunde ultimului punct de control al curbei (de obicei numit  $P_n$ , unde "n" reprezintă gradul curbei minus 1). Pe măsură ce " $t$ " variază între 0 și 1, curba Bézier se interpolează între aceste două puncte de control, definind astfel traiectoria curbei.

Polinoamele Bernstein, care fac parte din formula curbelor Bezier, poate fi definită astfel:

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \dots, n \quad (3.4)$$

unde  $n$  reprezintă ordinul curbei

Coefficientul binomial se calculează folosind formula:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \quad (3.5)$$

Există 3 cazuri specifice de curbe Bézier, fiecare folosind un număr specific de puncte de control.

### Curbe Bézier liniare

O curbă Bézier liniară, figura 3.2, este o simplă linie între două puncte distincte,  $P_0$  și  $P_1$ :

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1, \quad 0 \leq t \leq 1 \quad (3.6)$$

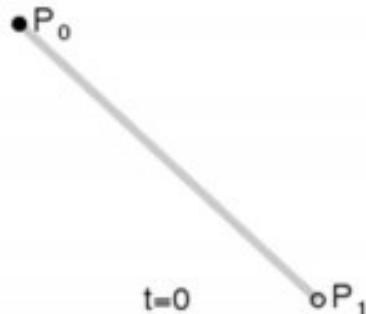


Figure 3.2: Curba Bézier liniară

### Curbe Bézier quadratiche

O curba Bézier quadratică, figura 3.3, este o curba definită de trei puncte de control,  $P_0$ ,  $P_1$ ,  $P_2$ , și prezintă o formă parabolică între aceste puncte:

$$\mathbf{B}(t) = (1-t)^2 \mathbf{P}_0 + 2(1-t)t \mathbf{P}_1 + t^2 \mathbf{P}_2, \quad 0 \leq t \leq 1 \quad (3.7)$$

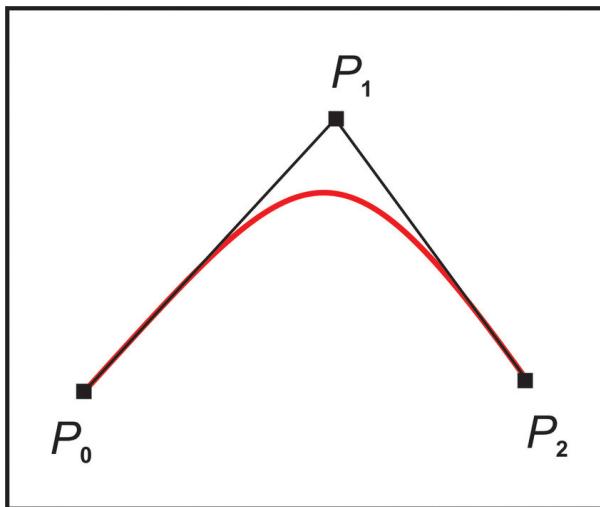


Figure 3.3: Curba Bézier quadratică

### Curbe Bézier cubice

O curba Bézier cubică, figura 3.4 este cel mai comun tip de curbe Bézier și este definit de patru puncte de control. Curba cubică poate descrie o varietate mai mare de forme și traекторii decât curbele liniare și pătratice:

$$\mathbf{B}(t) = (1-t)^3 \mathbf{P}0 + 3(1-t)^2 t \mathbf{P}1 + 3(1-t)t^2 \mathbf{P}2 + t^3 \mathbf{P}3, \quad 0 \leq t \leq 1 \quad (3.8)$$

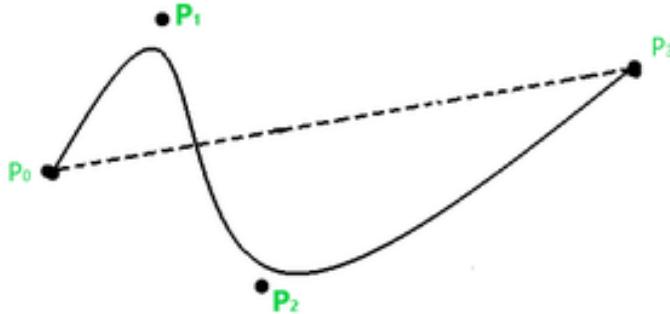


Figure 3.4: Curba Bézier cubică

### 3.4 Exemplificarea curbelor Bézier

Luăm în considerare trei puncte:  $P_0$ ,  $P_1$  și  $P_2$ , figura 3.5. Aceste puncte sunt folosite pentru a genera un poligon de control. **Un poligon de control** este un poligon care influențează forma sau traectoria unei curbe sau a unei suprafețe, figura 3.6. După ce am determinat poligonul de control, considerăm puncte de interpolare de-a lungul fiecărei laturi a poligonului format de cele trei puncte 3.7. Astfel, obținem două puncte,  $Q_0$  și  $Q_1$ , de interpolare, conectate printr-o dreaptă. Pe această nouă dreaptă, considerăm un punct suplimentar,  $C$ , care va defini curba, figura 3.8. Astfel, în același timp, punctul  $Q_0$  va interpola între  $P_0$  și  $P_1$  de-a lungul laturii, punctul  $Q_1$  va interpola între  $P_1$  și  $P_2$ , iar punctul  $C$  va interpola între  $Q_0$  și  $Q_1$ , determinând astfel traseul curbei, figura 3.9.

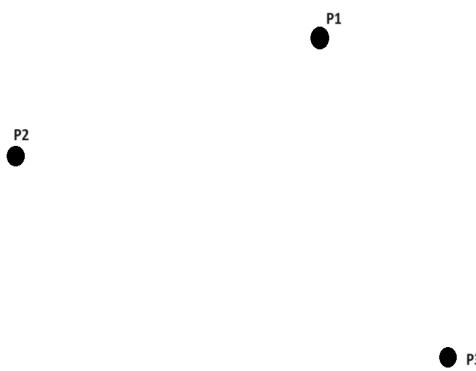


Figure 3.5: Definirea celor 3 puncte

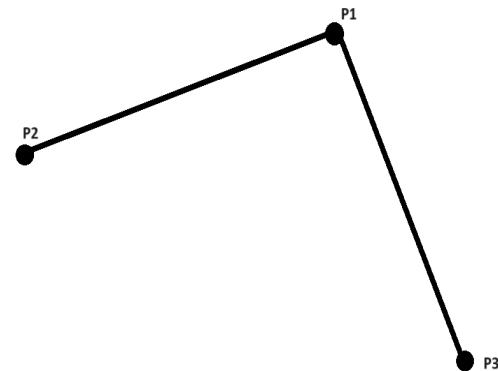


Figure 3.6: Formarea poligonului de control

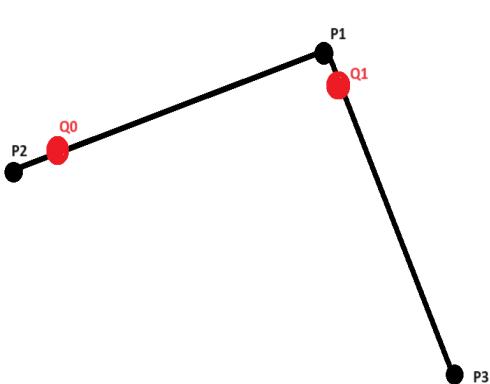


Figure 3.7: Foarmarea nivelului de interpolare

Figure 3.8: Definirea punctului pentru trasarea curbei

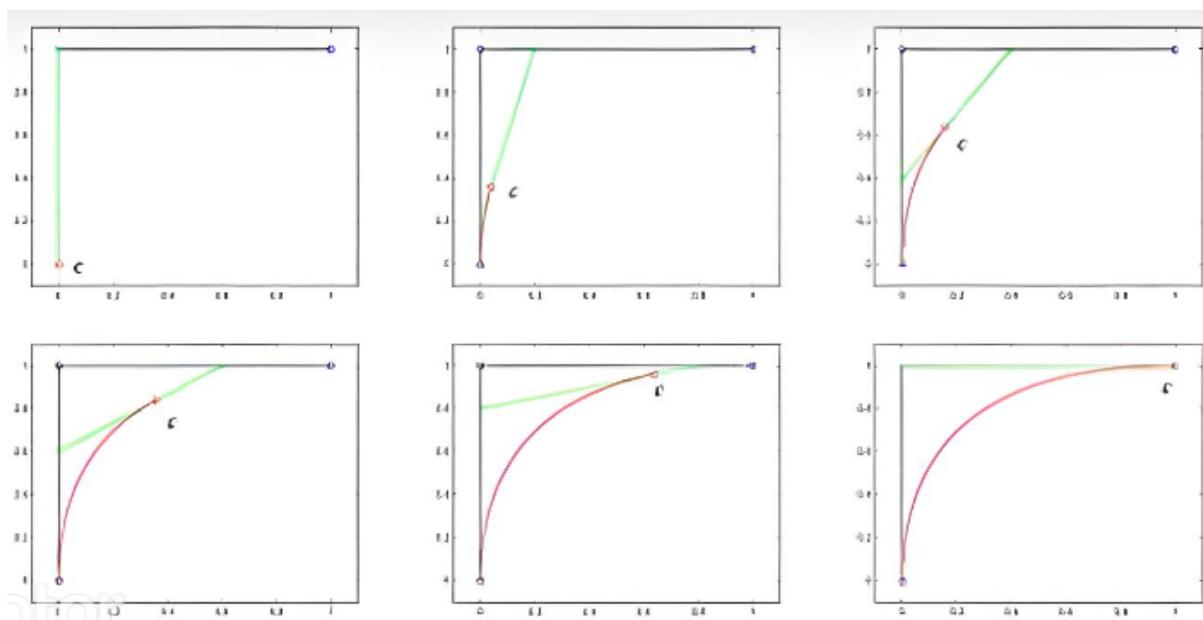


Figure 3.9: Trasarea curvei Bezier

### 3.5 Proprietăți ale curbelor Bézier

În paragrafele următoare, vom explora câteva dintre cele mai importante proprietăți ale curbelor Bézier [29]:

- **Proprietatea de interpolare a punctelor de capăt**

O curbă Bézier trebuie să atingă direct punctele sale de control de început și de sfârșit la capetele respective.

- **Simetria**

Curba Bézier definită de un set de puncte de control  $P_0, P_1, \dots, P_n$  are aceeași formă ca și curba Bézier definită de aceleași puncte de control în ordine inversă  $P_n, P_{n-1}, \dots, P_0$ . Curba este desenată doar în sens invers.

- **Invarianță afină**

Dacă luăm o curbă Bézier și aplicăm o transformare afină, cum ar fi o translatare sau o rotație, vom obține o nouă curbă. Aceeași transformare aplicată punctelor de control ale curbei originale va determina punctele de control care vor genera curba transformată.

În matematică, invarianța afină se referă la proprietatea unui obiect matematic de a rămâne neschimbat sub operațiile de translatare, scalare și rotație (transformări affine). Aceasta înseamnă că obiectul păstrează aceeași formă generală și proprietăți esențiale, indiferent de cum este translatat, scalat sau rotit în spațiu.

- **Control pseudo-local**

În curbele Bézier de gradul  $n$ , nu există control local, ceea ce înseamnă că orice modificare a unui punct de control necesită recalculara întregii curbe și afectează aspectul ei în ansamblu. Cu toate acestea, efectul modificării este mai mic pe măsură ce ne îndepărțăm mai mult de punctul de control modificat.

- **Diminuarea variației**

Dacă o linie dreaptă intersectează un poligon Bézier planar de  $m$  ori, atunci numărul maxim de intersecții ale liniei cu curba este de  $m$ . Simplu spus, curba nu poate avea mai multe oscilații decât poligonul.

Diminuarea variației se referă la o proprietate a anumitor obiecte matematice în care modificarea lor implică o reducere a numărului de schimbări de semn sau a fluctuațiilor, cum ar fi trecerea de la valori pozitive la negative sau invers. Această proprietate este adesea dorită în contexte precum interpolarea sau aproximarea, unde se dorește să se minimizeze oscilațiile sau să se asigure o convergență uniformă a soluțiilor.

- **Precizie liniară**

Dacă punctele de control  $b_i$ , unde  $i = 1, \dots, n - 1$ , sunt situate pe linia dreaptă dintre  $b_0$  și  $b_n$ , atunci curba Bézier de gradul  $n$  reprezintă o interpolare liniară între  $b_0$  și  $b_n$ .

### 3.6 Algoritmi

În această lucrare, scopul nostru este să rezolvăm problema netezirii traiectoriei dronei Crazyflie 2.0 prin eliminarea punctelor unghiulare existente în traiectorie. Am dezvoltat doi algoritmi pentru a aborda această problemă progresiv. Primul algoritm, cunoscut sub numele de algoritmul Bézier clasic, încearcă să eliminate traiectoria unghiulară inițială prin împărțirea ei în mai multe curbe Bézier. Punctele unghiulare care marchează tranziția între aceste curbe Bézier sunt păstrate, restul fiind eliminate. Al doilea algoritm, denumit algoritmul Bézier extins, se

bazează pe algoritmul anterior, dar reușește să elimine complet punctele unghiulare de tranziție, transformând astfel traectoria inițială într-o traectorie complet netezită. Codurile sursă ale celor doi algoritmi, împreună cu codul de conectare la dronă, se găsesc la [30]. În continuare, voi detalia fiecare algoritm în parte.

### 3.6.1 Algoritmul Bézier clasic

Algoritmul Bézier clasic se bazează pe formula matematică a curbelor Bézier cubice, care implică patru puncte: un punct de start, un punct de stop și două puncte intermediare de control. Pornind de la această formulă, algoritmul propus transformă traectoria initială a dronei Crazyflie, care conține puncte unghiulare, într-o traectorie compusă din mai multe curbe Bézier. Aceste curbe sunt legate între ele prin intermediul punctelor unghiulare, astfel încât punctul de stop al unei curbe Bézier devine punctul de start al curbei Bézier următoare. În continuare, voi detalia pseudocodul asociat acestui algoritm propus.

#### Explicarea algoritmului Bézier clasic

Algoritmul începe cu o matrice numită `matrix_points`, care conține coordonatele punctelor ce definesc traectoria cu puncte unghiulare. Variabila `i`, inițializată cu 1 (linia 2), funcționează ca un contor pentru bucla `while` (linia 6), în timp ce `pts_final` va conține traectoria rezultată în urma algoritmului (linia 5). Procesul continuă atâta timp cât contorul `i` este mai mic decât lungimea inițială a matricei `matrix_points`.

În cadrul buclei `while`, se formează o nouă matrice numită `matrix_aux`, care extrage primele 4 puncte din `matrix_points` (linia 14). Variabila `var` servește drept index pentru această nouă matrice. După completarea matricii `matrix_aux`, ultimul punct din matrice este salvat în variabila `last_value` (linia 21), având inițial valoarea 0. Apoi se folosește funcția `kron` (linia 22).

Această funcție este folosită pentru a calcula produsul Kronecker între elementele din `matrix_aux`. Produsul Kronecker este o operație matriceală care combină elementele unei matrice cu elementele unei alte matrice într-un mod specific, generând o matrice rezultată mai mare. Deși inițial nu pare să aibă o legătură directă cu algoritmul, funcția `kron` reprezintă partea crucială a întregului algoritm. Folosim această funcție pentru a putea calcula rezultatul obținut în urma formulei matematice a curbelor Bézier cubice explicată în capitolul 3. Funcția ne permite să combinăm cele 4 puncte din matricea `matrix_aux` cu parametrul de interpolare `t` în cadrul formulei matematice. Astfel, generăm traectoria netezită a celor 4 puncte din matricea `matrix_aux`.

Astfel, obținem o traectorie netezită care conține 101 puncte, punctul de start fiind primul element din matricea `matrix_aux`, iar punctul de stop este ultimul element al aceleiași matrice. Numărul de 101 puncte derivă din faptul că parametrul `t`, care reprezintă parametrul de interpolare, variază de la 0 la 1 luând 101 puncte. Am ales această valoare pentru a asigura o curbă netezită perfect; încercări cu 5, 20 sau 50 de puncte au dus la o curbură nesatisfăcătoare, cu aspect zimțat.

După calcularea traectoriei pentru `matrix_aux`, aceasta este salvată în variabila `pts_final` (linia 24), iar bucla `while` este reluată. Matricea `matrix_aux` este reinicializată de fiecare dată când bucla `while` se reia. Există o instrucțiune `if` despre care nu am vorbit care se află la începutul buclei `while`. Acea instrucțiune (linia 8) se apelează după o iterare a buclei `while` deoarece inițial variabila `var` este 1. Astfel, prin acea instrucțiune `if` de la început, ultimul element al matricii `matrix_aux` de dinainte devine primul element al noii matrice. Acest proces se repetă până când toate punctele unghiulare inițiale au fost procesate, iar traectoria finală este stocată în variabila `pts_final`.

**Algorithm 1** Algoritmul Bézier clasic

---

```
1:  $t \leftarrow \text{linspace}(0, 1, 101);$ 
2:  $i \leftarrow 1$ 
3:  $var \leftarrow 1$ 
4:  $last\_value \leftarrow [0, 0, 0]$  {Presupunem că last_value inițial este un o matrice cu o coloană și 3 linii având elementele nule}
5:  $pts\_final \leftarrow 0$ 
6: while  $i \leq \text{lungime}(\text{matrix\_points})$  do
7:    $\text{matrix\_aux} \leftarrow \text{zeros}(3, 4)$ 
8:   if  $var \neq 1$  then
9:      $\text{matrix\_aux}[:, 1] \leftarrow last\_value$  {Coloana 1 a matricei matrix_aux primește last_value}
10:    end if
11:     $var \leftarrow 1$ 
12:    while  $var \leq 4$  do
13:      if  $i \leq \text{lungime}(\text{matrix\_points})$  then
14:         $\text{matrix\_aux}[:, var] \leftarrow \text{matrix\_points}[:, i]$  {Populăm matrix_aux cu valorile core-spunzătoare din matrix_points}
15:         $var \leftarrow var + 1$ 
16:         $i \leftarrow i + 1$ 
17:      else
18:        break
19:      end if
20:    end while
21:     $last\_value \leftarrow \text{matrix\_aux}[:, 4]$  {Ultima coloană a lui matrix_aux devine nouă last_value}
22:     $pts \leftarrow \text{kron}((1 - t)^3, \text{matrix\_aux}[:, 1]) + \text{kron}(3 * (1 - t)^2 * t, \text{matrix\_aux}[:, 2]) + \text{kron}(3 * (1 - t) * t^2, \text{matrix\_aux}[:, 3]) + \text{kron}(t^3, \text{matrix\_aux}[:, 4])$ 
23:    if  $pts\_final == 0$  then
24:       $pts\_final \leftarrow pts$ 
25:    else
26:       $pts\_final \leftarrow \text{concat}(pts\_final, pts)$ 
27:    end if
28:     $var \leftarrow 2$ 
29:  end while
```

---

### 3.6.2 Algoritmul Bézier extins

Algoritmul Bézier extins este o îmbunătățire a algoritmului Bézier clasic prezentat anterior. Această îmbunătățire constă în eliminarea punctelor unghiulare de legătură dintre curbele Bézier, netezind astfel întreaga traекторie. Algoritmul preia output-ul generat de algoritmul clasic și produce o traекторie complet netezită. În continuare, voi explica cum se realizează acest proces bazându-mă pe pseudocodul atașat.

#### Explicarea algoritmului Bézier extins

Algoritmul este destul de similar cu cel precedent. Formarea matricei `matrix_aux` (linia 7) se face la fel, iar după generarea primei traectorii apare primul element de noutate: un bloc condițional (linia 18). La prima iterare se urmează ramura `else` (linia 29) deoarece flagul `midlevel` are valoarea 0. În ramura `else`, se extrage punctul de mijloc al traectoriei din variabila `pts` (linia 32). Astfel, în variabila `pts_final`, care reprezintă traectoria completă, se stochează doar jumătate din traectoria obținută (linia 38), deoarece scopul acestui algoritm este să eliminăm toate punctele unghiulare. Luăm doar jumătate din traectorie deoarece punctul unghiular de legătură va constitui un punct intermediar de control.

Bucla `while` se reia, iar de data aceasta intrăm pe ramura `if` a blocului condițional (linia 18). Se extrage din nou punctul de mijloc al noii traectorii (linia 20) și se folosește formula matematică pentru curbe Bézier quadratice. Dorim ca traectoria să nu ajungă în punctul unghiular de legătură. Astfel, vom crea o nouă traectorie având ca punct de start mijlocul traectoriei calculate la iterarea anterioară, stocat în `midpoint_first`, punctul unghiular de legătură, `middle_points`, și mijlocul traectoriei recent calculate, `midpoint_last`. Astfel, obținem în variabila `pts_midpoint` o traectorie ce elimină punctul unghiular de legătură (linia 22). Această variabilă `pts_midpoint` se stochează în continuarea variabilei `pts_final` (linia 24). Astfel, cele două curbe Bézier din primul algoritm se combină într-o singură curbă Bézier. Acest proces se repetă până când toate punctele unghiulare initiale au fost procesate, iar traectoria finală este stocată în variabila `pts_final`.

**Algorithm 2** Algoritm Bézier Extins

---

```
1: i  $\leftarrow$  1
2: while i  $\leq$  length(matrix_points) do
3:   midlevel  $\leftarrow$  0
4:   var  $\leftarrow$  1
5:   while var  $\leq$  4 do
6:     if i  $\leq$  length(matrix_points) then
7:       matrix_aux(:, var)  $\leftarrow$  matrix_points(:, i)
8:       var  $\leftarrow$  var + 1
9:     i  $\leftarrow$  i + 1
10:   else
11:     break
12:   end if
13:   end while
14:   if i < length(matrix_points) then
15:     i  $\leftarrow$  i - 1
16:   end if
17:   pts  $\leftarrow$  kron((1 - t)3, matrix_aux[:, 1]) + kron(3 * (1 - t)2 * t, matrix_aux[:, 2]) + kron(3 * (1 - t) * t2, matrix_aux[:, 3]) + kron(t3, matrix_aux[:, 4])
18:   if midlevel = 1 then
19:     middle  $\leftarrow$  int16(length(pts)/2)
20:     midpoint_last  $\leftarrow$  pts(:, middle)
21:     middle_point  $\leftarrow$  pts(:, 1)
22:     pts_midpoint  $\leftarrow$  kron((1 - t)2, midpoint_first) + kron(2(1 - t) · t, middle_point) + kron(t2, midpoint_last)
23:     pts_aux  $\leftarrow$  pts_midpoint(:, 2 : end)
24:     pts_final  $\leftarrow$  horzcat(pts_final, pts_aux)
25:     if i  $\geq$  length(matrix_points) then
26:       pts_final  $\leftarrow$  horzcat(pts_final, pts(:, middle + 1 : end))
27:     end if
28:     midpoint_first  $\leftarrow$  midpoint_last
29:   else
30:     if midpoint_first = 0 then
31:       j  $\leftarrow$  int16(length(pts)/2)
32:       midpoint_first  $\leftarrow$  pts(:, j)
33:       midlevel  $\leftarrow$  1
34:     end if
35:     middle  $\leftarrow$  int16(length(pts)/2)
36:     pts_aux  $\leftarrow$  pts(:, 1 : middle - 1)
37:     if pts_final = 0 then
38:       pts_final  $\leftarrow$  pts_aux
39:     else
40:       pts_final  $\leftarrow$  horzcat(pts_final, pts_aux)
41:     end if
42:   end if
43: end while
```

---

# 4 Rezultate experimentale

În acest capitol, vom analiza rezultatele obținute în urma aplicării celor doi algoritmi prezenți anterior. Algoritmii au fost testați în două situații distincte: una fără obstacole în spațiul de lucru și alta cu obstacole. Pentru fiecare situație, am considerat două scenarii: unul cu 7 puncte de control și altul cu 10 puncte de control. Toate experimentele au fost realizate utilizând o singură dronă. Vom compara rezultatele din simulare cu cele obținute în mediul real pentru fiecare scenariu în parte.

## 4.1 Traекторie formată din 7 puncte unghiulare

### 4.1.1 Spațiu de lucru fără obstacole

#### a) Traectorie cu puncte unghiulare

Se consideră o traectorie unghiulară formată din 7 puncte într-un spațiu de lucru fără obstacole. Punctul de start are coordonate (0.4, 0.8, 0.4), iar punctul de oprire are aceleași coordonate x și y, fiind situat doar la o altă înălțime (0.4, 0.8, 0.8). Ideea acestei traectorii a fost inspirată de o situație reală, similară cu livrarea de colete la mai multe destinații și revenirea la punctul de start. În figurile 4.1 și 4.2 se poate observa cum arată această traectorie în simulare.

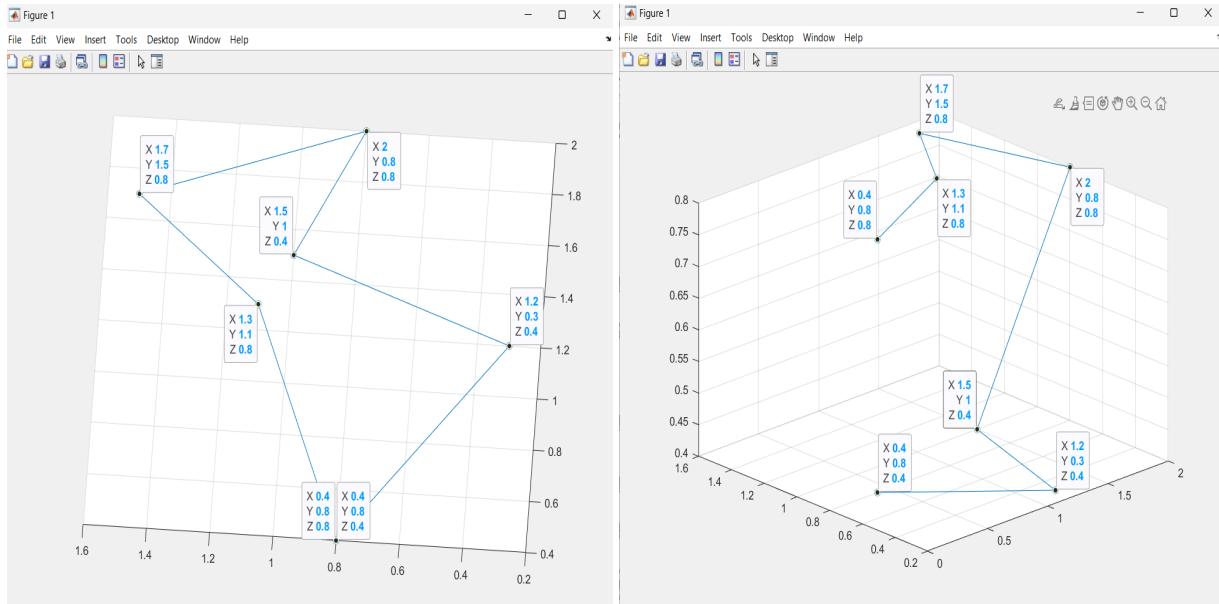


Figure 4.1: Traectorie formată din 7 puncte unghiulare privită de sus în simulare într-un spațiu fără obstacole

Figure 4.2: Traectorie formată din 7 puncte unghiulare privită dintr-o parte în simulare într-un spațiu fără obstacole

### b) Traекторie netezită prin curbe Bézier - metoda clasică

Pentru această traекторie inițială se aplică algoritmul Bézier clasic, care va reduce numărul punctelor unghiulare inițiale, rămânând doar un singur punct unghiular ce unește cele două curbe Bézier, exceptând punctele de start și stop. Algoritmul creează o curbă Bézier din primele 4 puncte, oprindu-se la punctul unghiular de legătură cu coordonatele  $(2, 0.8, 0.8)$ , care devine primul punct al celei de-a doua curbe Bézier, ce se oprește în punctul final. Astfel, traectoria inițială unghiulară este parțial netezită de algoritmul Bézier clasic, reprezentând o îmbunătățire parțială față de forma inițială a traectoriei, figurile 4.3 și 4.4.

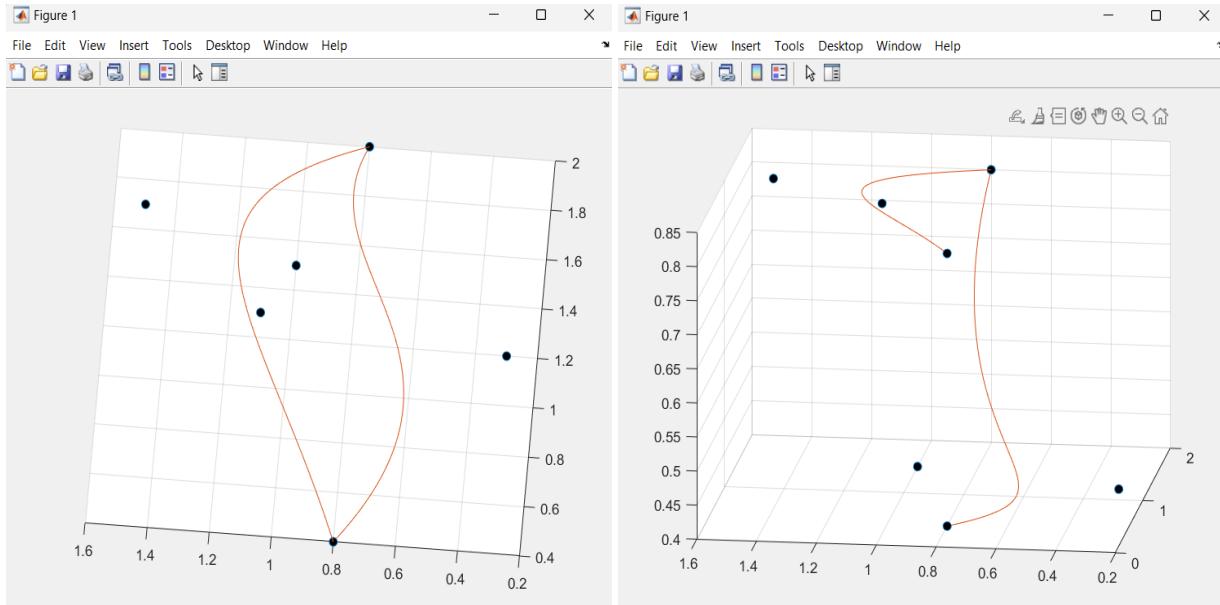


Figure 4.3: Traекторie parțial netezită privită de sus în simulare într-un spațiu fără obstacole

Figure 4.4: Traекторie parțial netezită privită dintr-o parte în simulare într-un spațiu fără obstacole

### c) Traекторie netezită prin curbe Bézier - metoda extinsă

Aplicând al doilea algoritm, Bézier extins, pe traectoria inițială, se observă în figurile de mai jos că punctul unghiular de legătură prezent în algoritmul anterior este eliminat. Această eliminare se realizează, aşa cum am explicat în capitolul 4, prin crearea unei curbe Bézier quadratic din 3 puncte. Punctul de start pentru această curbă este reprezentat de punctul de mijloc al primei curbe, punctul intermediar de control este reprezentat de coordonatele  $(2, 0.8, 0.8)$ , iar punctul de stop este reprezentat de punctul de mijloc al celei de-a doua curbe. Cu aceste 3 puncte se formează curba care elimină punctul unghiular de legătură. Astfel, traectoria rezultată în urma acestui algoritm este complet netezită, figurile 4.5 și 4.6.

În tabelul de mai jos, tabelul 4.1, se poate observa că lungimea traectoriei rezultate în urma aplicării algoritmului Bézier extins asupra traectoriei inițiale unghiulare este mult mai mică în comparație cu lungimea traectoriei rezultate în urma aplicării algoritmului Bézier clasic. Deși traectoria generată prin metoda Bézier extinsă conține un număr mai mare de puncte și necesită mai mult timp pentru generare față de traectoria generată prin Bézier clasic, traectoria Bézier extinsă este mult mai eficientă. Aceasta permite dronei să ajungă la punctul de stop mult mai rapid.

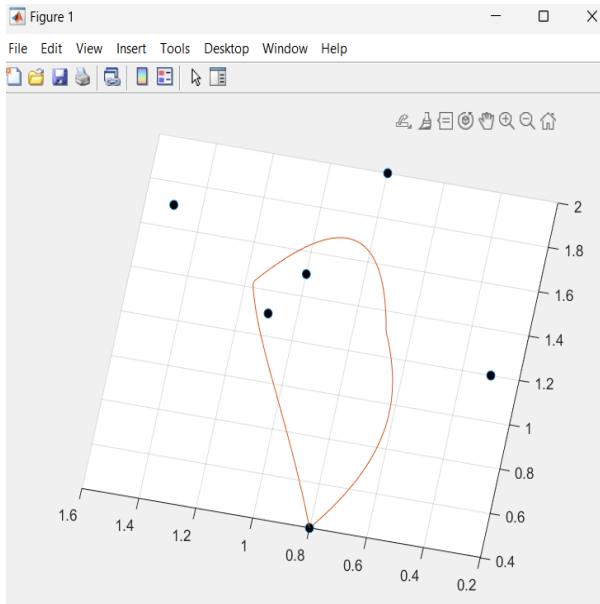


Figure 4.5: Traiectorie totală netezită privită de sus în simulare într-un spațiu fără obstacole

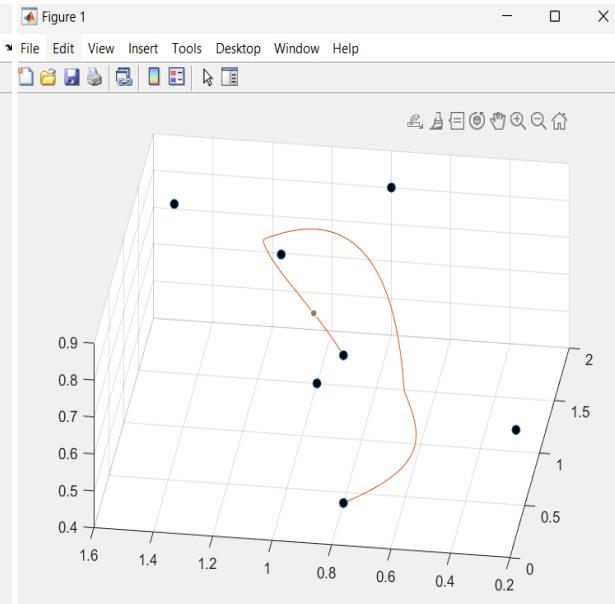


Figure 4.6: Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu fără obstacole

	Metoda Bézier clasica	Metoda Bézier extinsă
	Simulare	Simulare
Lungimea traiectoriei	3.6229 metri	2.9972 metri
Numărul de puncte din care este formată traiectoria	202	205
Timpul de execuție	0.33792 secunde	0.36115 secunde

Table 4.1: Rezultatele pentru o traiectorie formată din 7 puncte unghiulare în simulare într-un spațiu fără obstacole

#### d) Traiectorie în mediul real

În mediul real, am testat ambele traiectorii rezultate din cei doi algoritmi aşa cum se poate observa in figurile 4.7, 4.8, 4.9, 4.10. Linia albastră reprezintă traiectoria dronei în mediul real. După cum se poate observa în figurile de mai jos, drona încearcă să urmeze traiectoria furnizată ca input. Totuși, există câteva devieri de la traiectorie, datorate instabilității dronei Crazyflie. Cu toate acestea, drona reușește să urmeze în mare măsură traiectoria prestabilită. Ambele traiectorii rezultate pot fi urmărite la [31], respectiv [32].

În tabelul 4.2 este demonstrat ceea ce s-a menționat anterior în simulare, anume că algoritmul Bézier extins este mult mai eficient decât algoritmul Bézier clasic. Din tabel se observă că traiectoria rezultată de la algoritmul Bézier extins are o lungime mai mică, conține mai puține puncte și este parcursă mai rapid de drona Crazyflie în comparație cu traiectoria rezultată de la algoritmul Bézier clasic.

	Metoda Bézier clasica	Metoda Bézier extinsă
	Real	Real
Lungimea traiectoriei	3.90 metri	3.4330 metri
Numărul de puncte din care este formată traiectoria	58	54
Timpul în care drona execută traiectoria în mediul real	30 secunde	28 secunde

Table 4.2: Rezultatele pentru o traiectorie formată din 7 puncte unghiulare în mediul real într-un spațiu fără obstacole

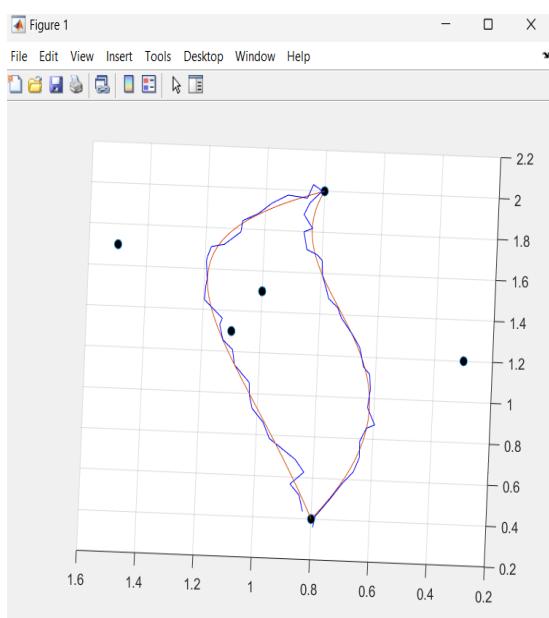


Figure 4.7: Traiectorie parțial netezită privită de sus în mediul real într-un spațiu fără obstacole

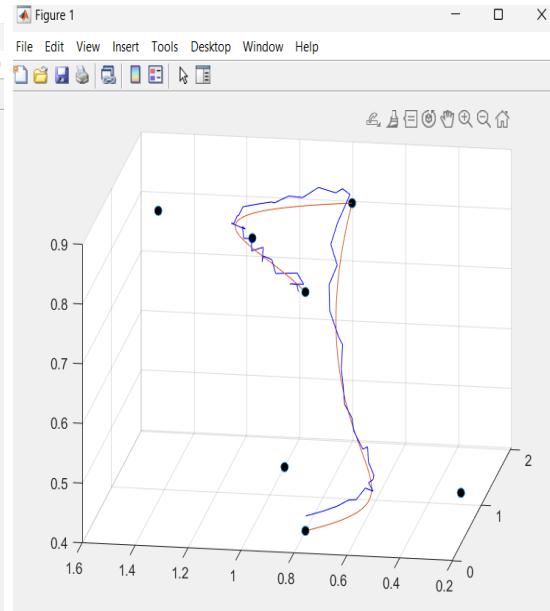


Figure 4.8: Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole

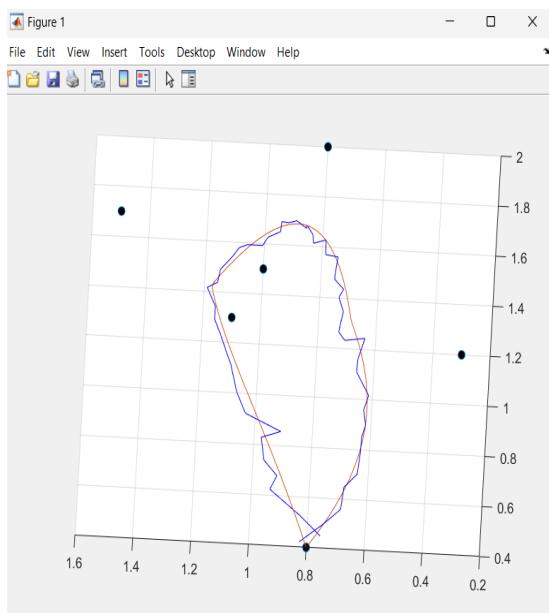


Figure 4.9: Traiectorie totală netezită privită de sus în mediul real într-un spațiu fără obstacole

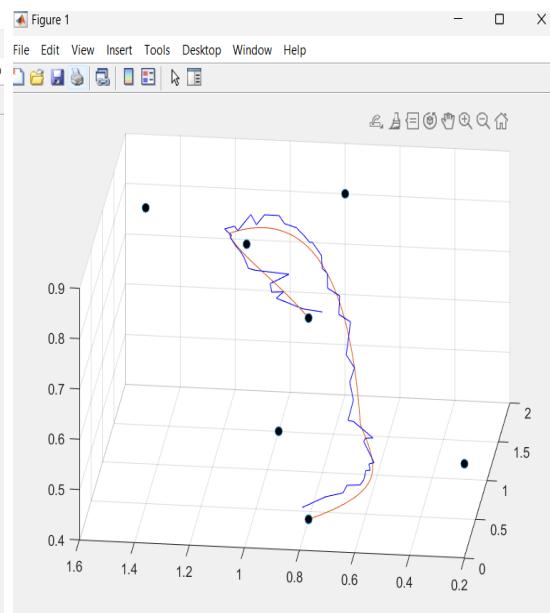


Figure 4.10: Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole

### 4.1.2 Spațiu de lucru cu obstacole

#### a) Traiectorie cu puncte unghiulare

Se consideră un spațiu de lucru cu obstacole. Obstacolele sunt două paralelipipede dreptunghice cu dimensiunile: 0.25 m lungime, 0.1 m lățime și 1 m înălțime, plasate la diferite coordonate în spațiul de lucru. Scopul dronei este să evite aceste obstacole. Scenariul este similar cu cel prezentat în subcapitolul anterior, referitor la livrarea de colete, dar de această dată cu obstacole. Drona pornește de la punctul de start cu coordonatele (0.3, 0.8, 0.4) și se deplasează către punctul de stop cu coordonatele (0.8, 1.3, 0.7). În figurile 4.11 și 4.12 se poate observa traiercia unghiulară a dronei printre obstacole, în simulare.

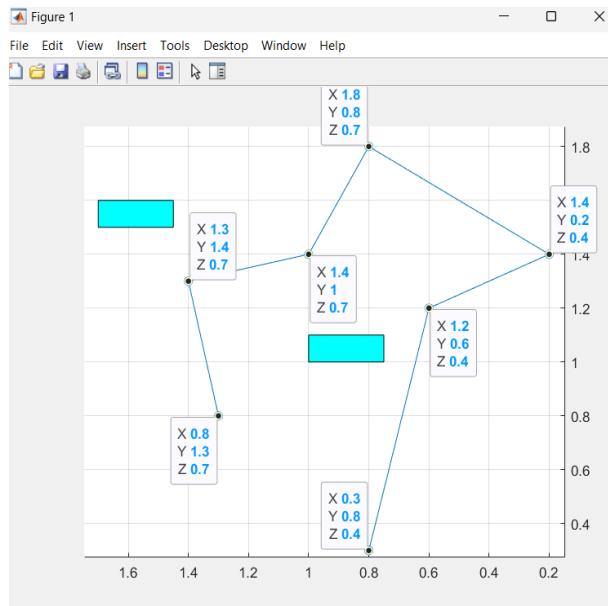


Figure 4.11: Traiectorie formată din 7 puncte unghiulare privită de sus în simulare într-un spațiu cu obstacole

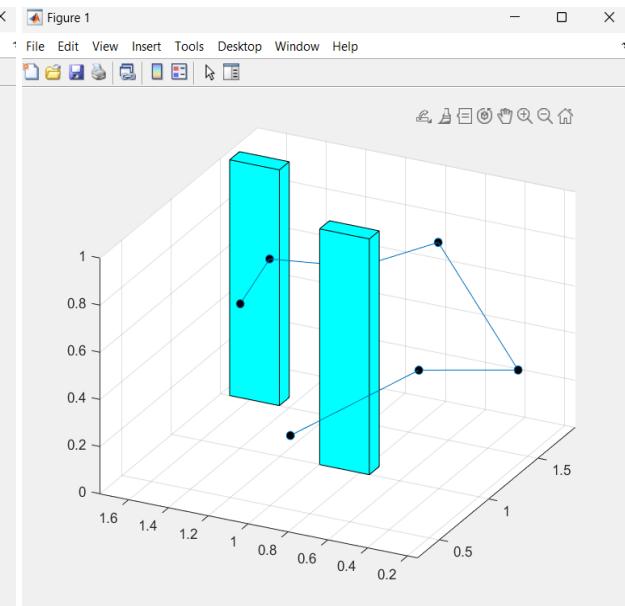


Figure 4.12: Traiectorie formată din 7 puncte unghiulare privită dintr-o parte în simulare într-un spațiu cu obstacole

#### b) Traiectorie netezită prin curbe Bézier - metoda clasică

Pentru traiercia unghiulară inițială descrisă anterior, s-a aplicat algoritmul Bézier clasic, rezultând într-o traiercie formată din două curbe Bézier unite printr-un punct unghiular de legătură situat la coordonatele (1.8, 0.8, 0.7). Astfel, din cele 7 puncte unghiulare inițiale rămân punctele de start și stop, precum și punctul de legătură. Traiercia rezultată în urma aplicării algoritmului poate fi văzută în figurile 4.13 și 4.14.

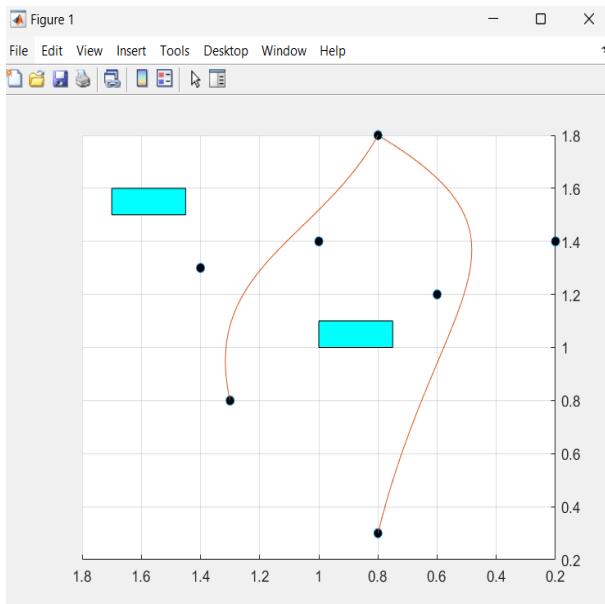


Figure 4.13: Traiectorie parțial netezită privită de sus în simulare într-un spațiu cu obstacole

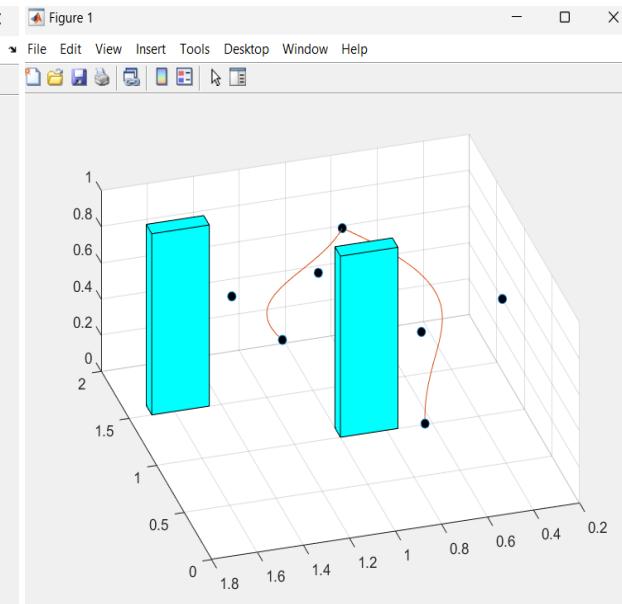


Figure 4.14: Traiectorie parțial netezită privită dintr-o parte în simulare într-un spațiu cu obstacole

### c) Traiectorie netezită prin curbe Bézier - metoda extinsă

Pentru aceeași traiectorie unghiulară inițială, s-a aplicat algoritmul Bézier extins, care elimină punctul unghiular de legătură, rezultând într-o traiectorie complet netezită, aşa cum se poate observa în figurile 4.15 și 4.16, fără a atinge niciunul dintre obstacole.

Din tabelul 4.3 se poate observa că traectoria creată de algoritmul Bézier extins este mai scurtă decât cea generată de algoritmul Bézier clasic, chiar dacă traectoria extinsă conține mai multe puncte și este generată într-un interval de timp mai mare.

	Metoda Bézier clasăcă	Metoda Bézier extinsă
	Simulare	Simulare
Lungimea traectoriei	2.8954 metri	2.489 metri
Numărul de puncte din care este formată traectoria	202	206
Timpul de execuție	0.12978 secunde	0.15003 secunde

Table 4.3: Rezultatele pentru o traiectorie formată din 7 puncte unghiulare în simulare într-un spațiu cu obstacole

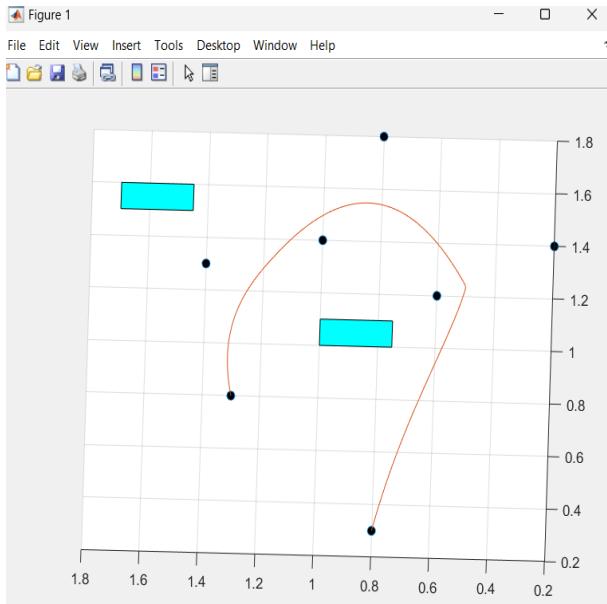


Figure 4.15: Traiectorie totală netezită privită de sus în simulare într-un spațiu cu obstacole

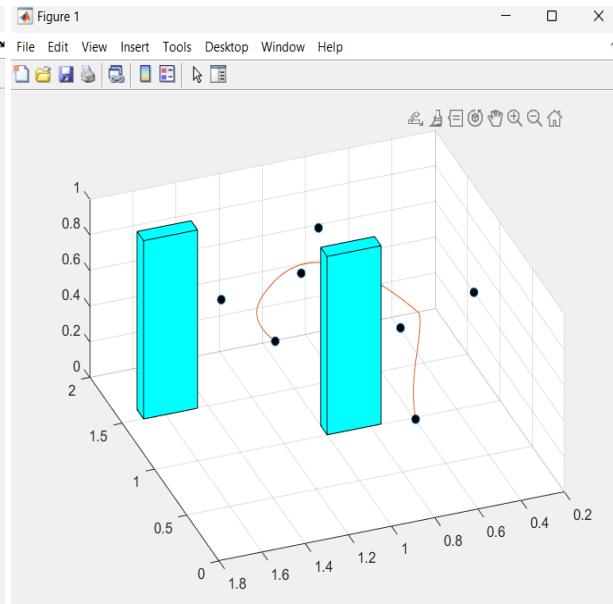


Figure 4.16: Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu cu obstacole

#### d) Traiectorie în mediul real

În mediul real, figurile 4.17, 4.18, 4.19 și 4.20, am utilizat două obstacole cu aceleași dimensiuni ca în simulare. Ambele traiectorii rezultate după aplicarea celor doi algoritmi au fost implementate pe drona Crazyflie. Se poate observa că drona reușește într-o mare măsură să păstreze traiectoria furnizată ca input, fără să intre în coliziune cu obstacolele. Ambele traiectorii rezultate pot fi urmărite la [33], respectiv [34].

Datele din tabelul 4.4 susțin ideea că, chiar și atunci când spațiul de lucru conține obstacole, traiectoria generată de algoritm Bézier extins este semnificativ mai scurtă, mai rapidă și conține mai puține puncte decât traiectoria generată de algoritm Bézier clasic.

	Metoda Bézier clasice	Metoda Bézier extinsă
	Real	Real
Lungimea traiectoriei	3.2284 metri	2.6276 metri
Numărul de puncte din care este formată traiectoria	53	52
Timpul în care drona execută traiectoria în mediul real	27 secunde	27 secunde

Table 4.4: Rezultatele pentru o traiectorie formată din 7 puncte unghiulare în mediul real într-un spațiu cu obstacole

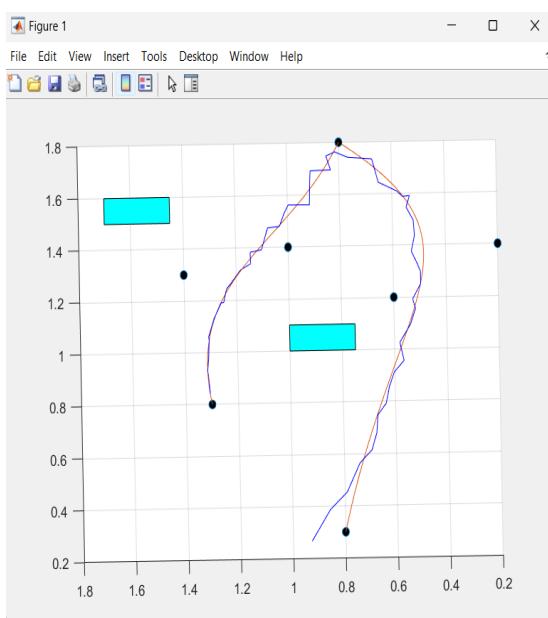


Figure 4.17: Traiectorie parțial netezită privită de sus în mediul real într-un spațiu cu obstacole

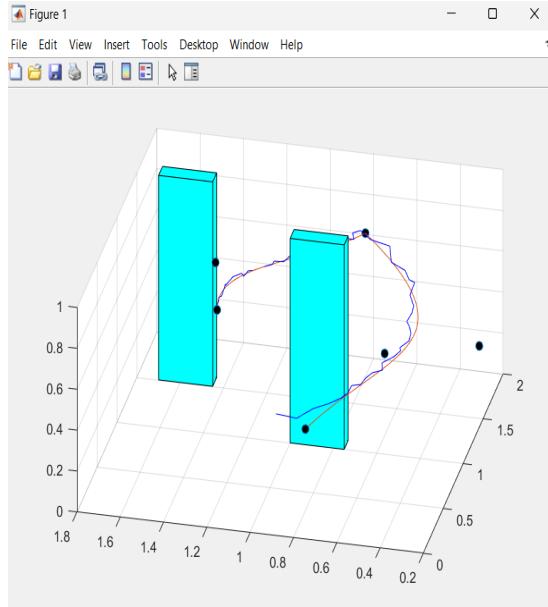


Figure 4.18: Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole

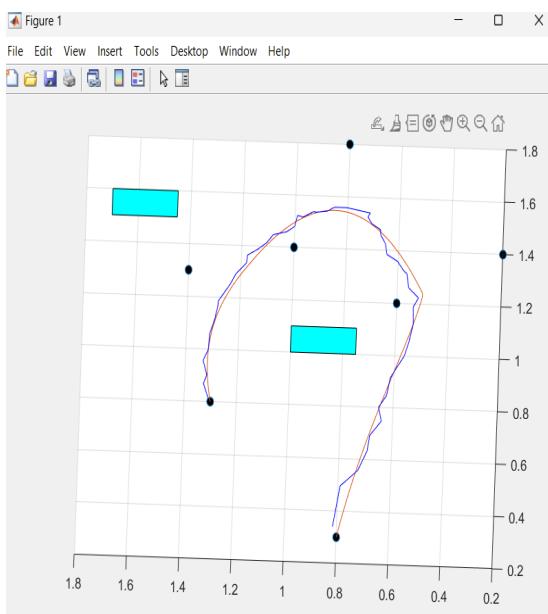


Figure 4.19: Traiectorie totală netezită privită de sus în mediul real într-un spațiu cu obstacole

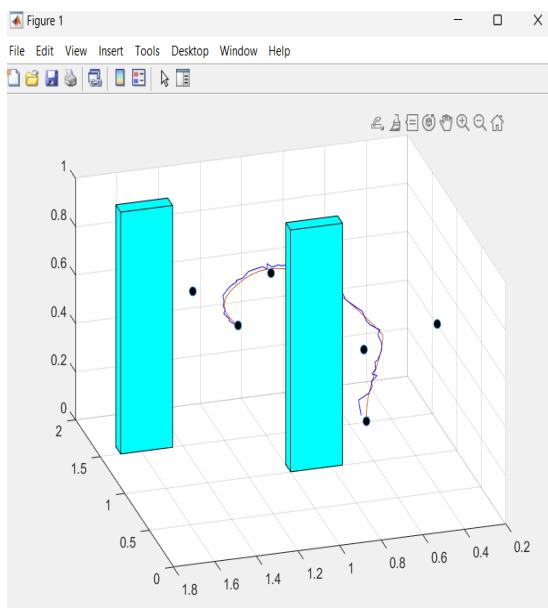


Figure 4.20: Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole

## 4.2 Traекторie formată din 10 puncte unghiulare

### 4.2.1 Spațiu de lucru fără obstacole

#### a) Traекторie cu puncte unghiulare

Într-un spațiu de lucru fără obstacole, avem o traectorie inițială formată din 10 puncte unghiulare. Punctul de start are coordonatele  $(0.5, 1, 0.2)$ , iar punctul de stop are coordonatele  $(1, 1, 1.1)$ . Scopul este să generăm o traectorie care să descrie o spirală elicoidală, adică o spirală care se deplasează în sus pe axa Z. În figurile 4.21 și 4.22 se poate vedea această traectorie inițială.

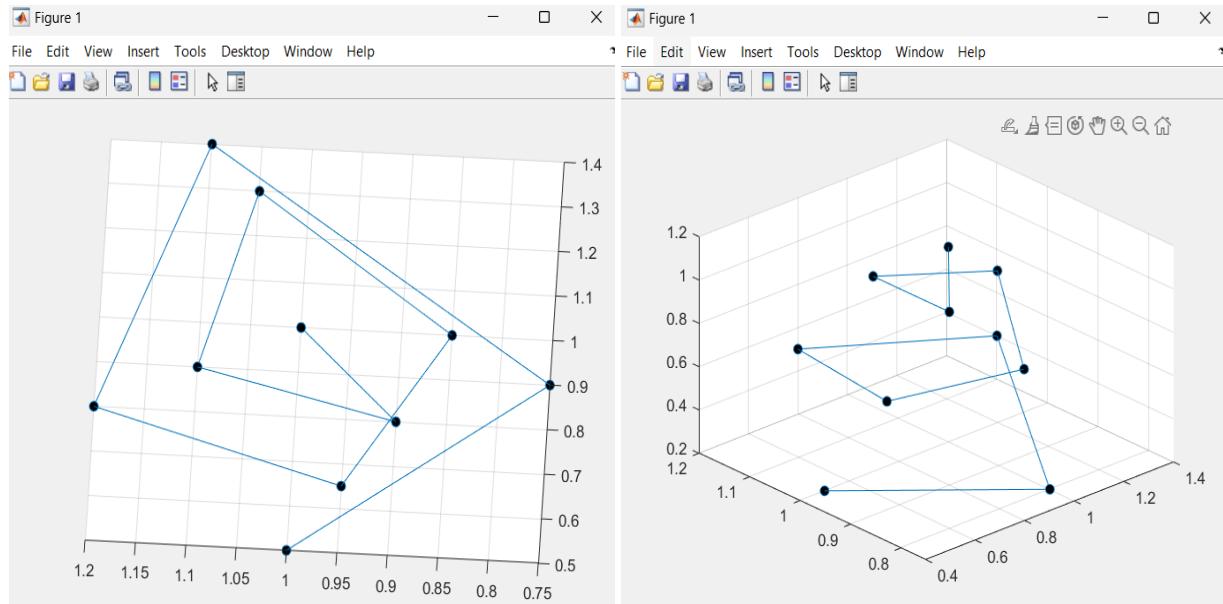


Figure 4.21: Traекторie formată din 10 puncte unghiulare privită de sus în simulare într-un spațiu fără obstacole

Figure 4.22: Traекторie formată din 10 puncte unghiulare privită dintr-o parte în simulare într-un spațiu fără obstacole

#### b) Traectorie netezită prin curbe Bézier - metoda clasică

Pentru traectoria inițială menționată, s-a folosit algoritmul Bézier clasic, obținând o traectorie parțial netezită formată din 3 curbe Bézier conectate printr-un total de două puncte de legătură. Primul punct de legătură se găsește la coordonatele  $(0.8, 1.2, 0.5)$ , iar al doilea punct de legătură este localizat la  $(1.29, 1.04, 0.79)$ . Rezultatul obținut în urma aplicării acestui algoritm poate fi observat în figurile 4.23 și 4.24.

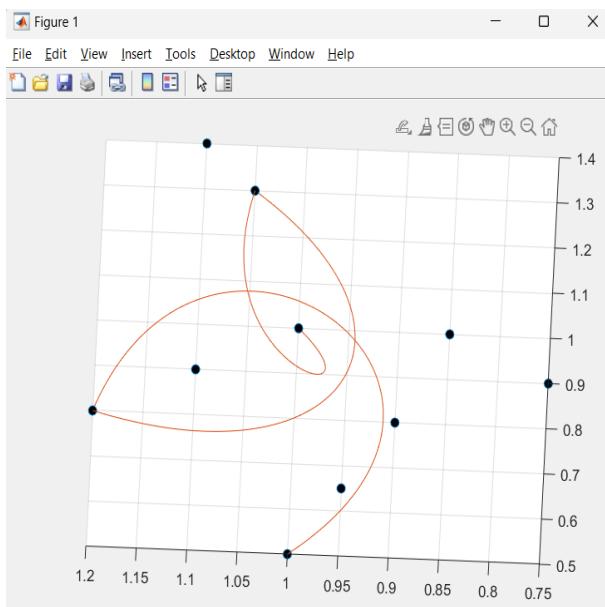


Figure 4.23: Traiectorie parțial netezită privită de sus în simulare într-un spațiu fără obstacole

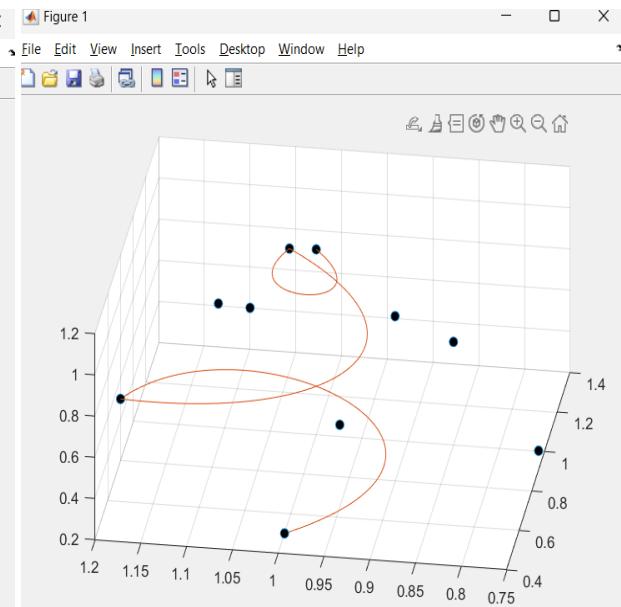


Figure 4.24: Traiectorie parțial netezită privită dintr-o parte în simulare într-un spațiu fără obstacole

### c) Traiectorie netezită prin curbe Bézier - metoda extinsă

Pentru trajectoria inițială formată din 10 puncte unghiulare, am aplicat algoritmul Bézier extins, eliminând toate punctele intermediare unghiulare și păstrând doar punctele de start și de stop. Astfel, trajectoria a fost complet netezită și este vizibilă în figurile 4.25 și 4.26.

Datele din tabelul 4.5, în cazul scenariului cu o traiectorie inițială formată din 10 puncte unghiulare, indică faptul că algoritmul Bézier extins generează o traiectorie mult mai scurtă, cu aproape 1 metru mai mică decât cea generată de algoritmul Bézier clasic. Cu toate acestea, trajectoria algoritmului Bézier extins este generată mai lent și conține mai multe puncte, ceea ce sugerează că algoritmul Bézier clasic are avantaje în această privință.

	Metoda Bézier clasăcă	Metoda Bézier extinsă
Simulare	2.4791 metri	1.7659 metri
Lungimea traectoriei	303	306
Numărul de puncte din care este formată traectoria	0.32871 secunde	0.3895 secunde
Timpul de execuție		

Table 4.5: Rezultatele pentru o traiectorie formată din 10 puncte unghiulare în simulare într-un spațiu fără obstacole

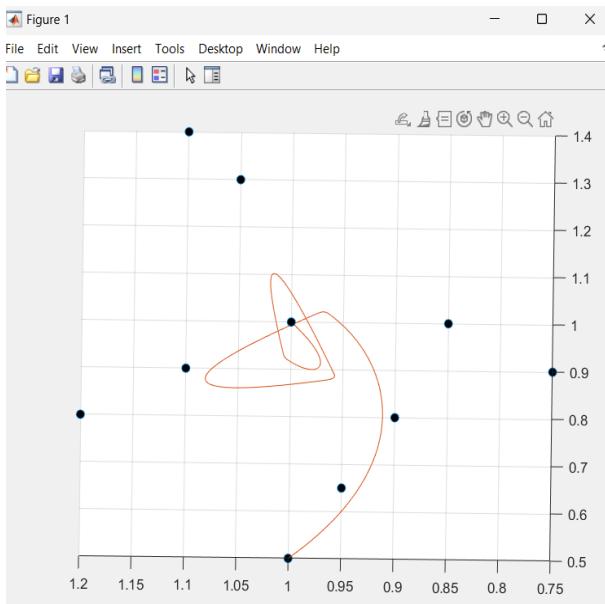


Figure 4.25: Traiectorie totală netezită privită de sus în simulare într-un spațiu fără obstacole

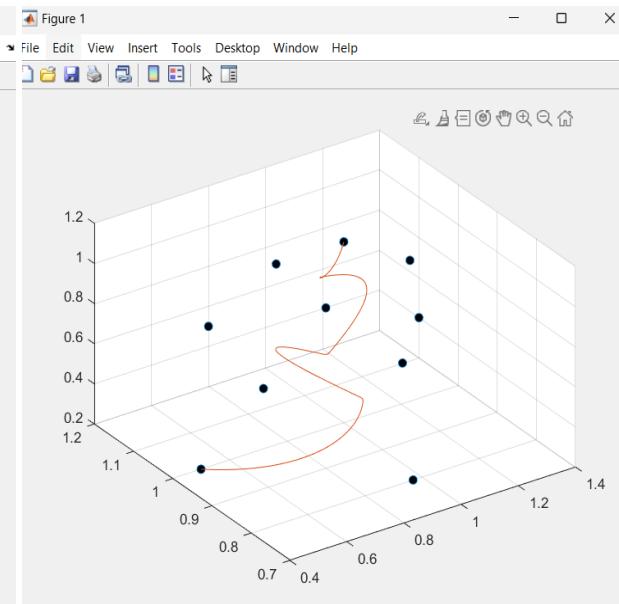


Figure 4.26: Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu fără obstacole

#### d) Traiectorie în mediul real

În mediul real, figurile 4.27, 4.28, 4.29 și 4.30, ambele traiectorii au fost testate pe drona Crazyflie. În comparație cu scenariul anterior, în care am încercat să ilustrăm analogia cu livrarea de colete la mai multe destinații, această traiectorie este mult mai complexă. Am dorit să observăm prin această încercare de a crea o spirală elicoidală dacă drona poate urmări o traiectorie în care se ridică continuu pe axa Z, încercând să atingă punctul de stop și să rămână aproape de diametrul spiralei. Din cele observate și din imaginile de mai jos, drona întâmpină dificultăți și are derapaje în urmărirea traiectoriei. Problemele par să apară pentru dronă în special când se apropie de al doilea punct de legătură, respectiv în ultima treime a traiectoriei. Drona înregistrează câteva mișcări necontrolate în încercarea de a urma traiectoria, însă instabilitatea și sensibilitatea dronei conduc la derapaje semnificative față de traiectorie. Cu toate acestea, drona reușește să atingă punctul de stop. Ambele traiectorii rezultate pot fi urmărite la [35], respectiv [36].

Tabelul 4.6 demonstrează că algoritmul Bézier extins este mult mai eficient în mediul real comparativ cu algoritmul Bézier clasic. Această concluzie se bazează pe lungimea mai scurtă a traiectoriei, numărul mai redus de puncte și rapiditatea cu care drona parcurge traiectoria.

	Metoda Bézier clasice	Metoda Bézier extinsă
Lungimea traiectoriei	Real	Real
Numărul de puncte din care este formată traiectoria	4.3784 metri	3.9542 metri
Timpul în care drona execută traiectoria în mediul real	72	68
	38 secunde	36 secunde

Table 4.6: Rezultatele pentru o traiectorie formată din 10 puncte unghiulare în mediul real într-un spațiu fără obstacole

#### 4. Rezultate experimentale

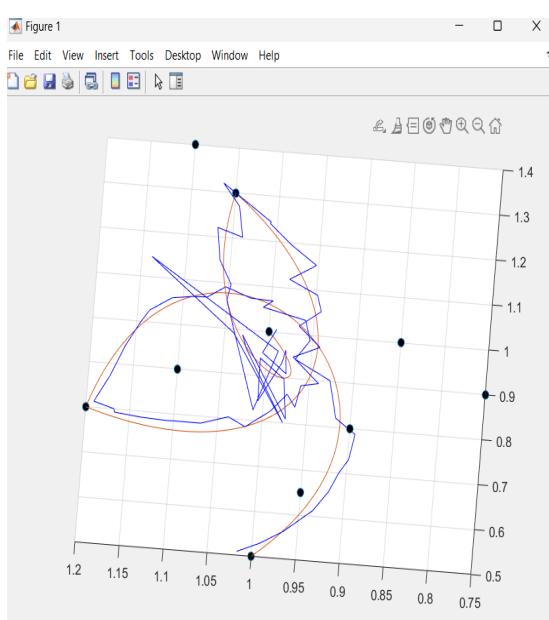


Figure 4.27: Traiectorie parțial netezită privită de sus în mediul real într-un spațiu fără obstacole

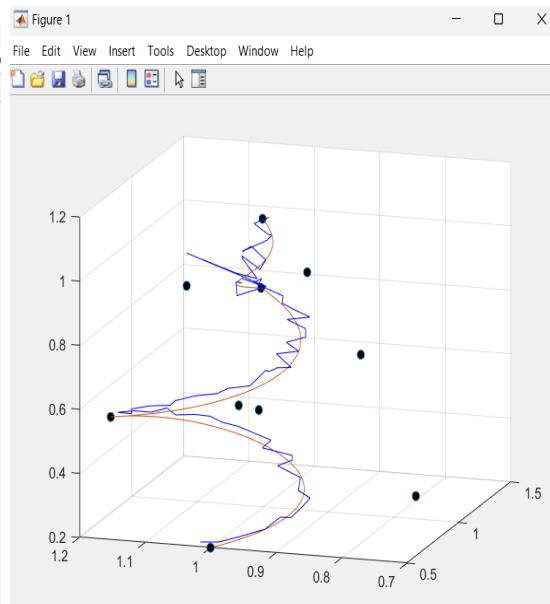


Figure 4.28: Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole

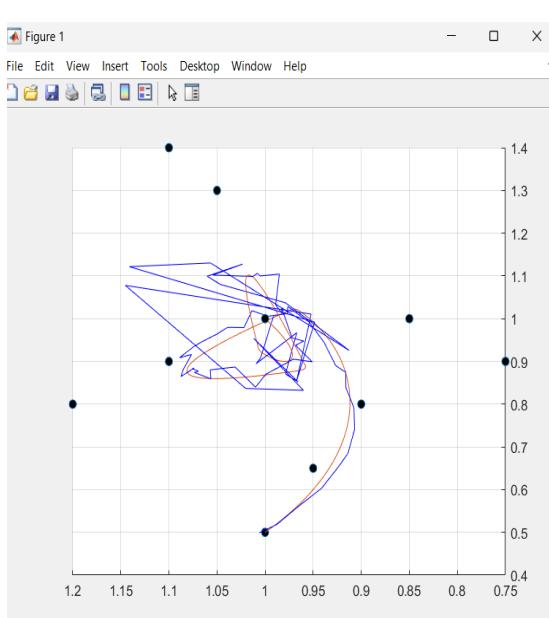


Figure 4.29: Traiectorie totală netezită privită de sus în mediul real într-un spațiu fără obstacole

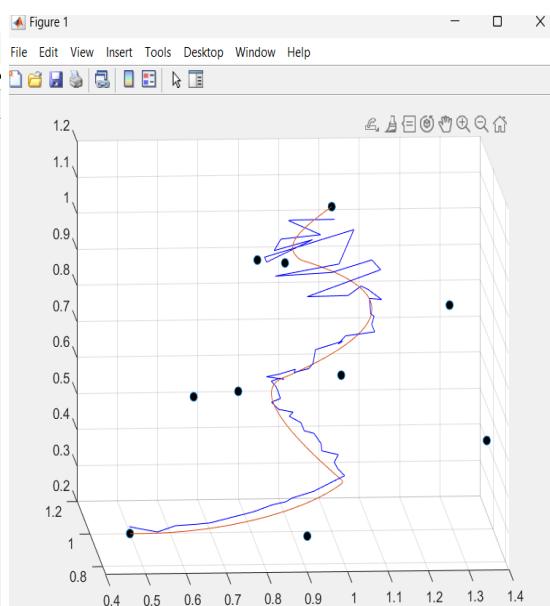


Figure 4.30: Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu fără obstacole

### 4.2.2 Spațiu de lucru cu obstacole

#### a) Traiectorie cu puncte unghiulare

Într-un spațiu de lucru cu obstacole, avem o traiectorie inițială compusă din 10 puncte unghiulare. Obstacolele constau în două paralelipipede dreptunghice cu dimensiunile: lungime 0.25 m, lățime 0.1 m și înălțime 1 m. Aceste două obstacole au fost plasate la coordonate diferite în spațiul de lucru. Scopul dronei este să urmărească traiectoria inițială evitând în același timp obstacolele. Punctul de start se află la coordonatele (0.4, 0.9, 0.4), iar punctul de stop la coordonatele (1.5, 2.1, 0.3). Traseul inițial unghiular poate fi observat în figurile 4.31 și 4.32.

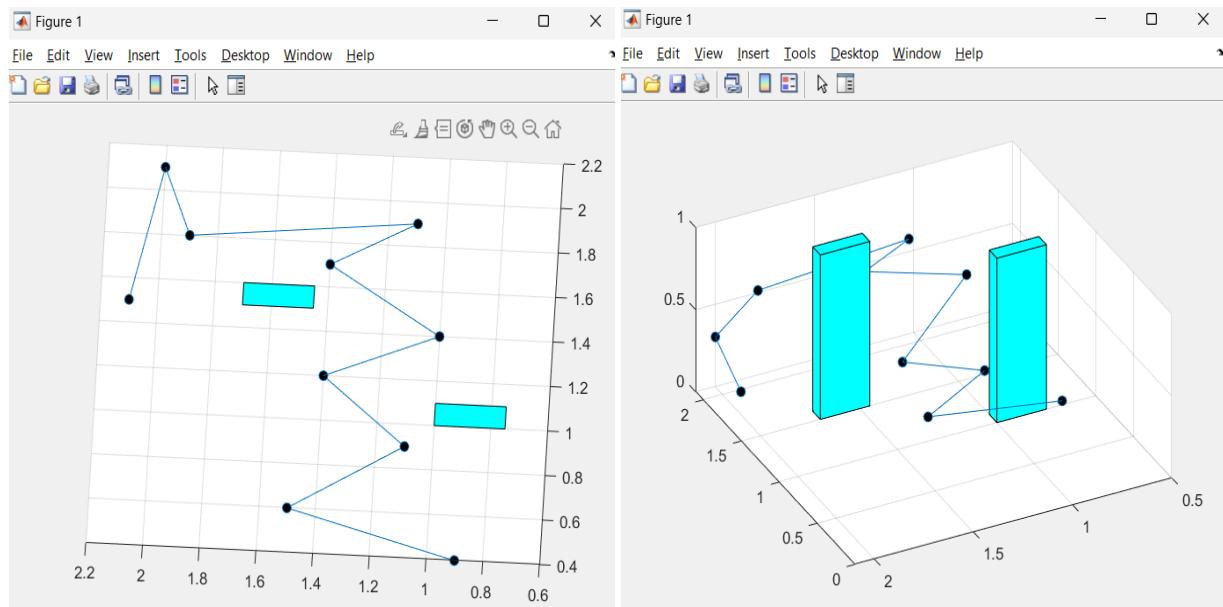


Figure 4.31: Traiectorie formată din 10 puncte unghiulare privită de sus în simulare într-un spațiu cu obstacole

Figure 4.32: Traiectorie formată din 10 puncte unghiulare privită dintr-o parte în simulare într-un spațiu cu obstacole

#### b) Traiectorie netezită prin curbe Bézier - metoda clasică

Pentru traiectoria inițială descrisă anterior, s-a aplicat algoritmul Bézier clasic. Astfel, a rezultat o traiectorie parțial netezită formată din 3 curbe Bézier unite prin două puncte de legătură. Primul punct de legătură este situat la coordonatele (1.20, 1.38, 0.4), iar al doilea punct de legătură se află la coordonatele (1.9, 1.1, 0.7). Din figurile 4.33 și 4.34 se poate observa cum traiectoria evită obstacolele și ajunge la punctul de stop.

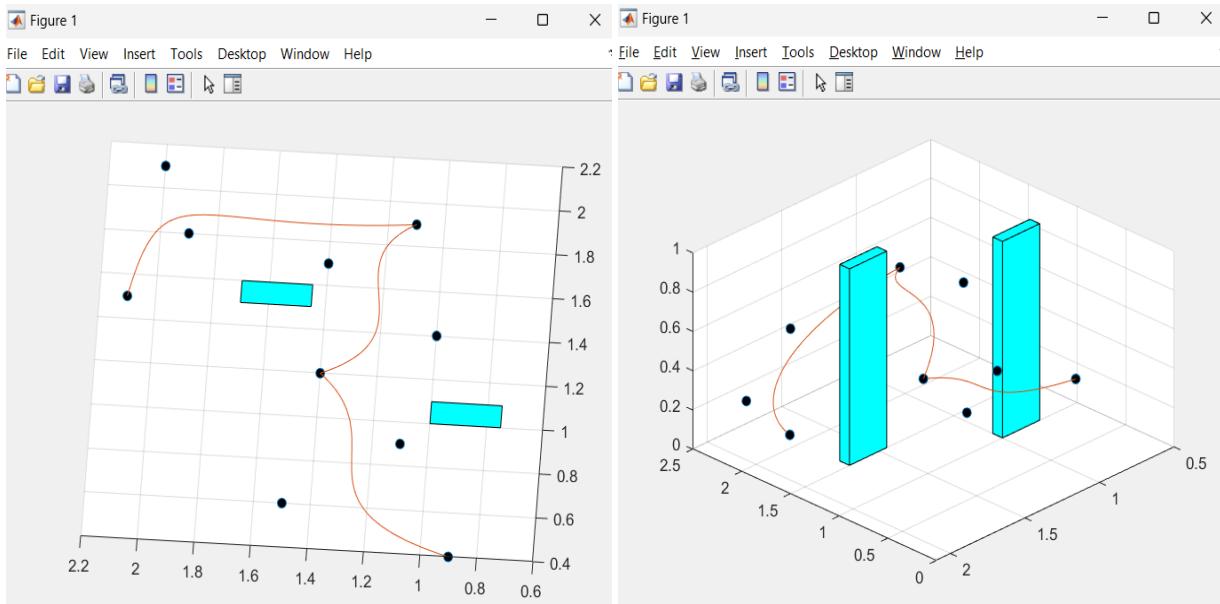


Figure 4.33: Traiectorie parțial netezită privită de sus în simulare într-un spațiu cu obstacole

Figure 4.34: Traiectorie parțial netezită privită dintr-o parte în simulare într-un spațiu cu obstacole

### c) Traiectorie netezită prin curbe Bézier - metoda extinsă

Pentru trajectoria inițială menționată anterior, s-a aplicat algoritmul Bézier extins, care a netezit întreaga traiectorie unghiulară. Astfel, traiectoria netezită trece între cele două obstacole și ocolește în spatele celui de-al doilea obstacol, aşa cum se poate observa în figurile 4.35 și 4.36.

În acest scenariu, tabelul 4.7, trajectoria generată de algoritmul Bézier clasic este mult mai lungă decât cea obținută prin algoritmul Bézier extins. Cu toate acestea, trajectoria clasă conține mai puține puncte și este generată mult mai rapid, oferindu-i un avantaj față de trajectoria generată de algoritmul Bézier extins.

	Metoda Bézier clasă	Metoda Bézier extinsă
	Simulare	Simulare
Lungimea traiectoriei	3.2843 metri	2.8545 metri
Numărul de puncte din care este formată traiectoria	303	310
Timpul de execuție	0.35798 secunde	0.38824 secunde

Table 4.7: Rezultatele pentru o traiectorie formată din 10 puncte unghiulare în simulare într-un spațiu cu obstacole

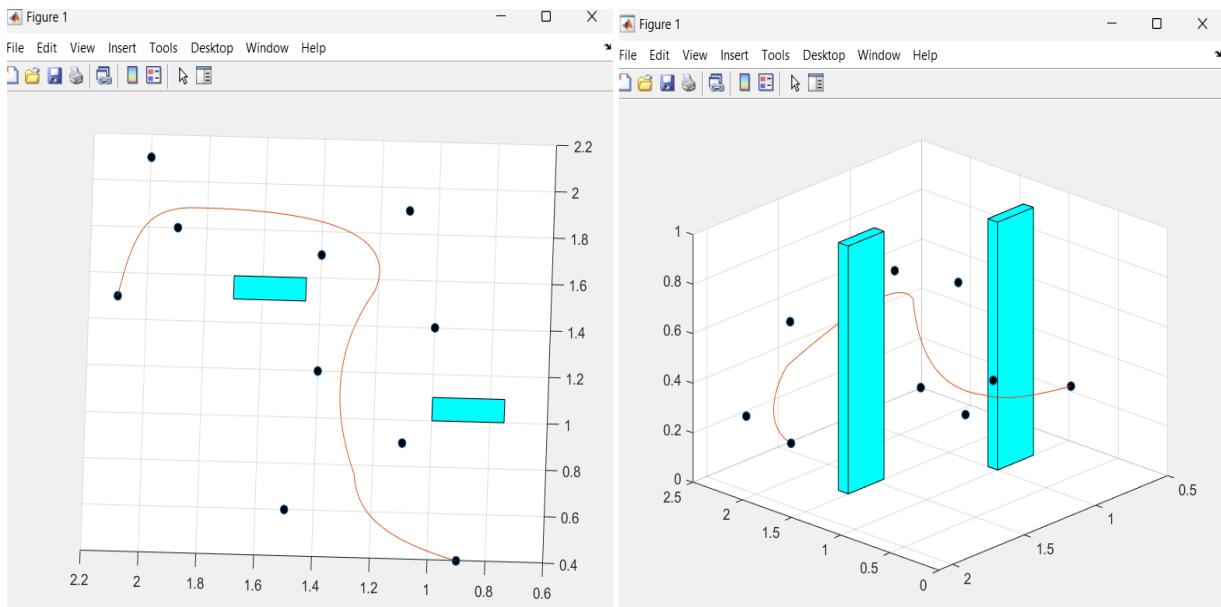


Figure 4.35: Traiectorie totală netezită privită de sus în simulare într-un spațiu cu obstacole

Figure 4.36: Traiectorie totală netezită privită dintr-o parte în simulare într-un spațiu cu obstacole

#### d) Traiectorie în mediul real

În mediul real, figurile 4.37, 4.38, 4.39 și 4.40, am amplasat obstacolele în spațiul de lucru. Ambele traiectorii rezultate au fost implementate pe drona. Din imaginile de mai jos se poate observa cum drona reușește să urmeze în întregime traiectoria impusă fără a se abate semnificativ. Aceasta poate fi considerat cel mai bun rezultat obținut în urma testelor realizate, deoarece este singurul în care traiectoria dronei este foarte apropiată de cea impusă. Ambele traiectorii rezultate pot fi urmărite la [37], respectiv [38]

Din tabelul 4.8, se poate observa clar că algoritmul Bézier extins este mult mai eficient decât algoritmul Bézier clasic. Această eficiență se datorează lungimii mai scurte a traiectoriei, rapiditatea cu care drona Crazyflie parcurge traiectoria și numărului redus de puncte al traiectoriei rezultate.

	Metoda Bézier clasăcă	Metoda Bézier extinsă
	Simulare	Simulare
Lungimea traiectoriei	3.5062 metri	3.0810 metri
Numărul de puncte din care este formată traiectoria	74	73
Timpul în care drona execută traiectoria în mediul real	39 secunde	37 secunde

Table 4.8: Rezultatele pentru o traiectorie formată din 10 puncte unghiulare în mediul real într-un spațiu cu obstacole

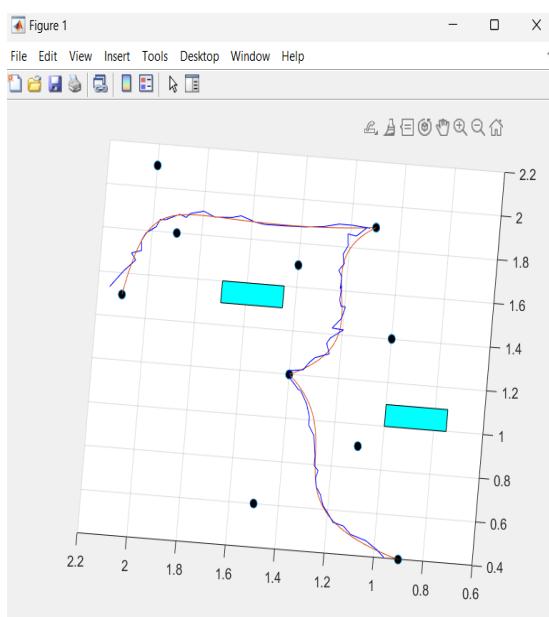


Figure 4.37: Traiectorie parțial netezită privită de sus în mediul real într-un spațiu cu obstacole

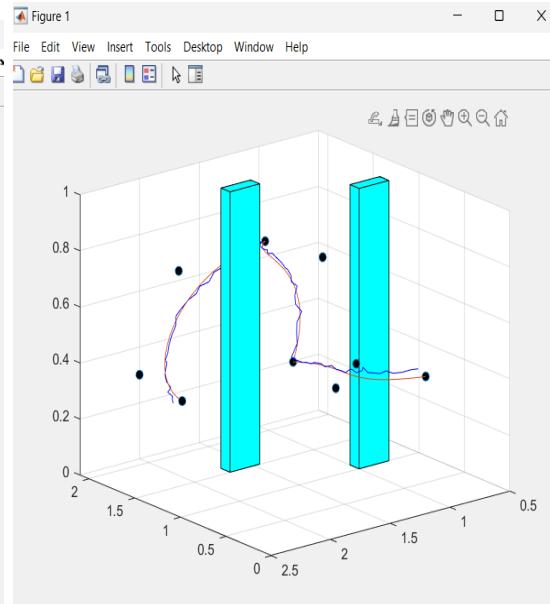


Figure 4.38: Traiectorie parțial netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole

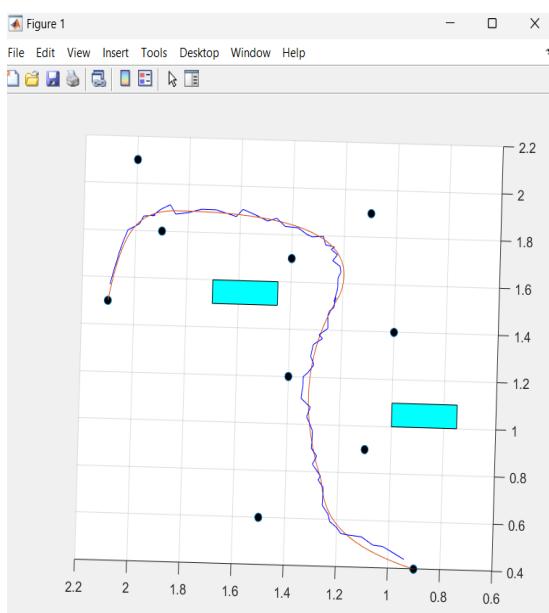


Figure 4.39: Traiectorie totală netezită privită de sus în mediul real într-un spațiu cu obstacole

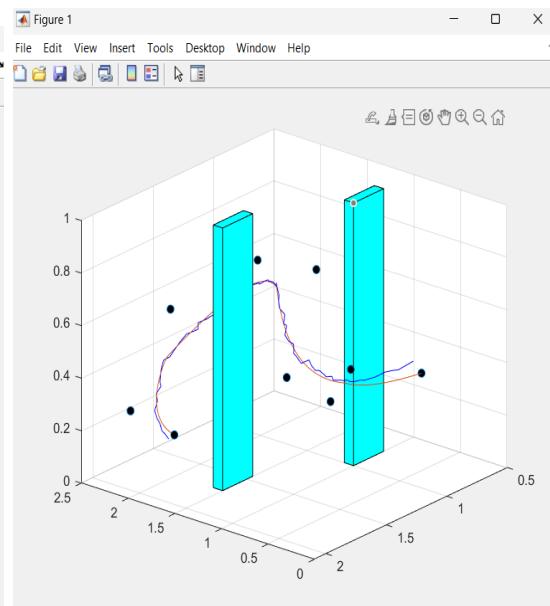


Figure 4.40: Traiectorie totală netezită privită dintr-o parte în mediul real într-un spațiu cu obstacole

## 5 Concluzii și Îmbunătățiri viitoare

În această lucrare, am urmărit să generez o traекторie netedă pentru o dronă Crazyflie 2.0, cu scopul de a îmbunătăți mișcarea acesteia prin eliminarea punctelor unghiulare care afectează accelerarea continuă. Am ales să folosesc curbele Bezier pentru netezirea traectoriei și am dezvoltat doi algoritmi bazați pe aceste curbe, obținând astfel rezultate din ce în ce mai precise și constante. Algoritmii au fost dezvoltăți în mediul MATLAB, unde au fost realizate și simulările prezentate în capitolul 4. De asemenea, am dezvoltat un cod în limbajul Python pentru a interacționa cu drona și pentru a încărca traectoriile generate de cei doi algoritmi, în vederea obținerii rezultatelor în mediul real. A fost analizată traectoria din simulare și cea din mediul real a dronei, constatăndu-se că aceasta reușește să urmeze traectoria din simulare în mediul real, cu mici abateri.

În privința direcțiilor viitoare de cercetare, o abordare interesantă ar fi investigarea influenței utilizării mai multor drone asupra performanței și fiabilității acestei metode. De asemenea, o altă direcție de cercetare ar putea fi explorarea controlului unghiului de rotație pe axa Z a dronei (yaw), pentru a permite dronelor să se rotească în conformitate cu traectoria pe care o urmează.

# Bibliografie

- [1] Abhijeet Ravankar et al. “Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges”. In: *Sensors* 18.9 (2018), p. 3170.
- [2] Abhijeet Ravankar et al. “Path smoothing extension for various robot path planners”. In: *2016 16th international conference on control, automation and systems (ICCAS)*. IEEE. 2016, pp. 263–268.
- [3] Syed Agha Hassnain Mohsan et al. “Towards the unmanned aerial vehicles (UAVs): A comprehensive review”. In: *Drones* 6.6 (2022), p. 147.
- [4] Hufang Wang et al. “The use of unmanned aerial vehicle in military operations”. In: *International Conference on Man-Machine-Environment System Engineering*. Springer. 2020, pp. 939–945.
- [5] Mario Arturo Ruiz Estrada et al. “The uses of unmanned aerial vehicles—UAV’s-(or drones) in social logistic: Natural disasters response and humanitarian relief aid”. In: *Procedia Computer Science* 149 (2019), pp. 375–383.
- [6] Jaime Del Cerro et al. “Unmanned aerial vehicles in agriculture: A survey”. In: *Agronomy* 11.2 (2021), p. 203.
- [7] Ludovica De Lucia et al. “UAV Main Applications: From Military to Agriculture Fields”. In: *Internet of Unmanned Things (IoUT) and Mission-based Networking*. Springer, 2023, pp. 1–23.
- [8] Evangeline Corcoran et al. “Automated detection of wildlife using drones: Synthesis, opportunities and constraints”. In: *Methods in Ecology and Evolution* 12.6 (2021), pp. 1103–1114.
- [9] Zeinab E Ahmed et al. “Monitoring of Wildlife Using Unmanned Aerial Vehicle (UAV) With Machine Learning”. In: *Applications of Machine Learning in UAV Networks*. IGI Global, 2024, pp. 97–120.
- [10] Chiara Torresan et al. “Forestry applications of UAVs in Europe: A review”. In: *International journal of remote sensing* 38.8-10 (2017), pp. 2427–2447.
- [11] Gabriel Lucian Palcau. “New Path Planner Methods for Planetary Explorations”. PhD thesis. Politecnico di Torino, 2022.
- [12] Bryan Gustavo Cabrera Ramirez. “Three-dimensional kinematic path planning constraining for fixedwing UAVs using dubins path”. B.S. thesis. Universitat Politècnica de Catalunya, 2021.
- [13] Muhammad Usman. “Quadcopter modelling and control with matlab/simulink implementation”. In: (2020).
- [14] Carlos Luis et al. “Design of a trajectory tracking controller for a nanoquadcopter”. In: *arXiv preprint arXiv:1608.05786* (2016).
- [15] Bitcraze AB. *Getting started with the Crazyflie 2.X*. <https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/>. 2023.
- [16] Wojciech Giernacki et al. “Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering”. In: *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE. 2017, pp. 37–42.

- [17] Bitcraze AB. *Flow deck V2*. <https://www.bitcraze.io/products/flow-deck-v2/>. 2023.
- [18] Bitcraze AB. *Crazyradio PA*. <https://www.bitcraze.io/products/crazyradio-pa/>. 2023.
- [19] Bitcraze AB. *Loco Positioning system*. <https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/>. 2023.
- [20] Anton Ledergerber et al. “A robot self-localization system using one-way ultra-wideband communication”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 3131–3137.
- [21] Bitcraze AB. *TDoA principles*. [https://www.bitcraze.io/documentation/repository/lps-node-firmware/master/functional-areas/tboa\\_principles/](https://www.bitcraze.io/documentation/repository/lps-node-firmware/master/functional-areas/tboa_principles/). 2023.
- [22] Bitcraze AB. *Getting started with the Loco Positioning system*. <https://www.bitcraze.io/documentation/tutorials/getting-started-with-loco-positioning-system/>. 2023.
- [23] Ron Choy et al. “Parallel MATLAB: Doing it right”. In: *Proceedings of the IEEE* 93.2 (2005), pp. 331–341.
- [24] M Abdul Majid et al. “MATLAB as a teaching and learning tool for mathematics: a literature review”. In: *International Journal of Arts & Sciences* 6.3 (2013), p. 23.
- [25] AS Saabith et al. “Python current trend applications-an overview”. In: *International Journal of Advance Engineering and Research Development* 6.10 (2019).
- [26] Arun Kumar et al. “A survey: How Python pitches in IT-world”. In: *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE. 2019, pp. 248–251.
- [27] Simen Vike Lande. “Path planning for marine vehicles using bézier curves”. MA thesis. NTNU, 2018.
- [28] Rik Ghosh. “Parametric Curve Fitting and Vector Graphics”. PhD thesis. Reed College, 2017.
- [29] Gerald Farin et al. *Handbook of computer aided geometric design*. Elsevier, 2002.
- [30] Codul sursă al algoritmilor. <https://github.com/MGEmil/Bachelor-Thesis.git>.
- [31] Filmare traекторie 7 puncte parțial netezită. <https://youtube.com/shorts/RqgYp4V4vFk>.
- [32] Filmare traекторie 7 puncte total netezită. [https://youtube.com/shorts/\\_4SfC9is0g8?feature=share](https://youtube.com/shorts/_4SfC9is0g8?feature=share).
- [33] Filmare traекторie 7 puncte parțial netezită în spațiu de lucru cu obstacole. <https://youtube.com/shorts/fqstDw5x-oQ?feature=share>.
- [34] Filmare traекторie 7 puncte total netezită în spațiu de lucru cu obstacole. <https://youtube.com/shorts/KJ3h99EjTW4?feature=share>.
- [35] Filmare traectorie 10 puncte parțial netezită. <https://youtu.be/s0rIlEt7jgo>.
- [36] Filmare traectorie 10 puncte totala netezită. <https://youtu.be/7tM1H-LDFI4>.
- [37] Filmare traectorie 10 puncte parțial netezită în spațiu de lucru cu obstacole. <https://youtu.be/Bj6DodukLoA>.
- [38] Filmare traectorie 10 puncte total netezită în spațiu de lucru cu obstacole. <https://youtu.be/aeKpzK-URW0>.

# 6 Anexe

## 6.1 Anexa 1. Cod în MATLAB. Algoritmul Bézier clasic

```
1 %% environment săfr obstacole -> Algoritmul Bézier clasic
2 tic;
3
4 xlim([0 3.5])
5 ylim([0 3])
6 zlim([0 2])
7 t = linspace(0,1,101);
8
9 % 7 puncte
10 % matrix_points = [0.4,1.2,1.5,2,1.7,1.3,0.4; 0.8,0.3,1,0.8,1.5,1.1,0.8;
11 % 0.4,0.4,0.4,0.8,0.8,0.8,0.8];
12
13 % 10 puncte
14 matrix_points = [0.5,0.9,1.4,0.8,0.65,1,1.3,0.9,0.8,1;
15 % 1,0.75,1.1,1.2,0.95,0.85,1.05,1.1,0.9,1;
16 % 0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1];
17
18 plot3(matrix_points(1,:),matrix_points(2,:),matrix_points(3,:),
19 'o','MarkerFaceColor','k');
20 hold on
21 grid on;
22
23 nr_puncte = 0;
24 i = 1;
25 j = 1;
26 last_value = zeros(3,1);
27 matrix_aux = zeros(3,4);
28 var = 1;
29 lung_traiectorie= 0;
30 pts_final = 0;
31
32 while i<=length(matrix_points)
33     matrix_aux = zeros(3,4);
34     if var ~= 1
35         matrix_aux(:,1) = last_value;
36     end
37     while var<=4
38         if(i<=length(matrix_points))
39             matrix_aux(:,var) = matrix_points(:,i);
40             var = var+1;
41             i = i+1;
42         else
43             break;
44         end
45     end
46     last_value = matrix_aux(:,4);
47     pts = kron((1-t).^3,matrix_aux(:,1)) +
48     kron(3*(1-t).^2.*t,matrix_aux(:,2)) +
49     kron((1-t).^1.*t.^2,matrix_aux(:,3)) +
50     t.*matrix_aux(:,4);
```

```

    kron(3*(1-t).*t.^2,matrix_aux(:,3)) + kron(t.^3,matrix_aux(:,4));
46 if(pts_final == 0)
47     pts_final = pts;
48 else
49     pts_final = horzcat(pts_final,pts);
50 end
51 var = 2;
52 end
53 plot3(pts_final(1,:),pts_final(2,:),pts_final(3,:));
54
55 size_pts = size(pts_final);
56 nr_puncte = size_pts(2);
57 while j < size_pts(2)
58     first_point = pts_final(:,j);
59     second_point = pts_final(:,j + 1);
60     lung_traiectorie = lung_traiectorie + sqrt((second_point(1,1) -
61         first_point(1,1))^2 + (second_point(2,1) - first_point(2,1))^2 +
62         (second_point(3,1) - first_point(3,1))^2);
63     j = j + 1;
64 end
65 elapsed_time = toc;
66 disp(['Lungimea traiectoriei: ' , num2str(lung_traiectorie) , ' metri']);
67 disp(['Durata de texecuie: ' , num2str(elapsed_time), ' secunde']);
68 disp(['Numarul de puncte utilizat:' , num2str(nr_puncte)]);
69
70 % csvwrite('data_normal_without_obst.csv', pts_final);
71 % Salvarea traiectorie intr-un sfier de tip csv
72
73 %% environment cu obstacole -> Algoritmul Bézier clasic
74
75 tic;
76 xlim([0 3.5])
77 ylim([0 3])
78 zlim([0 2])
79 t = linspace(0,1,101);
80
81 % 7 puncte
82 % matrix_points = [0.3,1.2,1.4,1.8,1.4,1.3,0.8; 0.8,0.6,0.2,0.8,1,1.4,1.3;
83 %                   0.4,0.4,0.4,0.7,0.7,0.7,0.7];
84
85 % 10 puncte
86 matrix_points = [0.4,0.6,0.9,1.2,1.4,1.7,1.9,1.8,2.1,1.5;
87 %                   0.9,1.5,1.1,1.4,1,1.4,1.1,1.9,2,2.1;
88 %                   0.4,0.4,0.4,0.4,0.7,0.7,0.7,0.7,0.3,0.3];
89
90 plot3(matrix_points(1,:),matrix_points(2,:),matrix_points(3,:),
91 'o','MarkerFaceColor','k');
92 hold on
93 grid on;
94
95 % Obstacol 1
96 length_obs = 1.1;
97 width_obs = 0.75;
98 height_obs = 1;
99
100 vertices = [
101     1 1 0;
102     length_obs 1 0;
103     length_obs width_obs 0;

```

```

101     1 width_obs 0;
102     1 1 height_obs;
103     length_obs 1 height_obs;
104     length_obs width_obs height_obs;
105     1 width_obs height_obs
106 ];
107
108
109 faces = [
110     1 2 6 5;
111     2 3 7 6;
112     3 4 8 7;
113     4 1 5 8;
114     1 2 3 4;
115     5 6 7 8
116 ];
117
118 patch('Vertices', vertices, 'Faces', faces, 'FaceColor', 'cyan',
119         'EdgeColor', 'black');
120 hold on
121
122 % Obstacol 2
123 length_obs = 1.6;
124 width_obs = 1.45;
125 height_obs = 1;
126
127 vertices = [
128     1.5 1.7 0;
129     length_obs 1.7 0;
130     length_obs width_obs 0;
131     1.5 width_obs 0;
132     1.5 1.7 height_obs;
133     length_obs 1.7 height_obs;
134     length_obs width_obs height_obs;
135     1.5 width_obs height_obs
136 ];
137
138 faces = [
139     1 2 6 5;
140     2 3 7 6;
141     3 4 8 7;
142     4 1 5 8;
143     1 2 3 4;
144     5 6 7 8
145 ];
146
147 patch('Vertices', vertices, 'Faces', faces, 'FaceColor', 'cyan',
148         'EdgeColor', 'black');
149 hold on
150
151 nr_puncte = 0;
152 i = 1;
153 j = 1;
154 last_value = zeros(3,1);
155 matrix_aux = zeros(3,4);
156 var = 1;
157 lung_traiectorie= 0;
158 pts_final = 0;
159

```

```

160 while i<=length(matrix_points)
161     matrix_aux = zeros(3,4);
162     if var ~= 1
163         matrix_aux(:,1) = last_value;
164     end
165     while var<=4
166         if(i<=length(matrix_points))
167             matrix_aux(:,var) = matrix_points(:,i);
168             var = var+1;
169             i = i+1;
170         else
171             break;
172         end
173     end
174     last_value = matrix_aux(:,4);
175     pts = kron((1-t).^3,matrix_aux(:,1)) +
176     kron(3*(1-t).^2.*t,matrix_aux(:,2)) +
177     kron(3*(1-t).*t.^2,matrix_aux(:,3)) + kron(t.^3,matrix_aux(:,4));
178     if(pts_final == 0)
179         pts_final = pts;
180     else
181         pts_final = horzcat(pts_final,pts);
182     end
183     var = 2;
184 end
185
186 size_pts = size(pts_final);
187 nr_puncte = size_pts(2);
188 while j < size_pts(2)
189     first_point = pts_final(:,j);
190     second_point = pts_final(:,j + 1);
191     lung_traiectorie = lung_traiectorie + sqrt((second_point(1,1) -
192         first_point(1,1))^2 + (second_point(2,1) - first_point(2,1))^2 +
193         (second_point(3,1) - first_point(3,1))^2);
194     j = j + 1;
195 end
196
197 elapsed_time = toc;
198 disp(['Lungimea traiectoriei: ' , num2str(lung_traiectorie) , ' metri']);
199 disp(['Durata de texecuie: ' , num2str(elapsed_time), ' secunde']);
200 disp(['Numarul de puncte utilizat:' , num2str(nr_puncte)]);
201
202 % csvwrite('data_normal_obst.csv', pts_final);
203 % Salvarea traiectorie într-un fișier de tip csv

```

## 6.2 Anexa 2. Cod în MATLAB. Algoritmul Bézier extins

```

1      %% environment să afișeze obstacole -> 7 puncte -> Algoritmul Bézier extins
2 tic;
3 xlim([0 3.5])
4 ylim([0 3])
5 zlim([0 2])
6 t = linspace(0,1,101);
7 t_aux = linspace(0,1,5);

```

```

8
9 % 7 puncte
10 % matrix_points = [0.4,1.2,1.5,2,1.7,1.3,0.4; 0.8,0.3,1,0.8,1.5,1.1,0.8;
11 % 0.4,0.4,0.4,0.8,0.8,0.8,0.8];
12 % 10 puncte
13 matrix_points = [0.5,0.9,1.4,0.8,0.65,1,1.3,0.9,0.8,1,
14 % 1,0.75,1.1,1.2,0.95,0.85,1.05,1.1,0.9,1;
15 % 0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1];
16 plot3(matrix_points(1,:),matrix_points(2,:),matrix_points(3,:),
17 'o','MarkerFaceColor','k');
18 hold on
19 grid on;
20
21 i = 1;
22 midlevel = 0;
23 midpoint_first = 0;
24 matrix_aux = zeros(3,4);
25 pts_final = 0;
26 pts_aux = 0;
27 nr_puncte = 0;
28 var = 1;
29 while i<=length(matrix_points)
30     var = 1;
31     while var<=4
32         if(i<=length(matrix_points))
33             matrix_aux(:,var) = matrix_points(:,i);
34             var = var+1;
35             i = i+1;
36         else
37             break;
38         end
39     end
40     if (i< length(matrix_points))
41         i = i-1;
42     end
43     pts = kron((1-t).^3,matrix_aux(:,1)) +
44     kron(3*(1-t).^2.*t,matrix_aux(:,2)) +
45     kron(3*(1-t).*t.^2,matrix_aux(:,3)) + kron(t.^3,matrix_aux(:,4));
46     if midlevel == 1
47         middle = int16(length(pts)/2);
48         midpoint_last= pts(:,middle);
49         middle_point = pts(:,1);
50         pts_midpoint = kron((1-t).^2,midpoint_first) +
51         kron(2*(1-t).*t,middle_point) + kron(t.^2,midpoint_last);
52
53         P1 = pts_aux(:,end);
54         P2 = pts_midpoint(:,2);
55         pts_second_curve = kron((1-t_aux).^2,P1) +
56         kron(2*(1-t_aux).*t_aux,midpoint_first) + kron(t_aux.^2,P2);
57         pts_final = horzcat(pts_final,pts_second_curve(:,2:end));
58
59         if (i >= length(matrix_points))
60             pts_aux = pts_midpoint(:,3:end-2);
61             pts_final = horzcat(pts_final,pts_aux);
62             P1 = pts_midpoint(:,end-1);
63             P2 = pts(:,middle+1);
64             midpoint_first = pts_midpoint(:,end);

```

```

61     pts_second_curve = kron((1-t_aux).^2,P1) +
62     kron(2*(1-t_aux).*t_aux,midpoint_first) + kron(t_aux.^2,P2);
63     pts_final = horzcat(pts_final,pts_second_curve);
64     pts_final = horzcat(pts_final,pts(:,middle+2:end));
65 else
66     pts_aux = pts_midpoint(:,3:end-2);
67     pts_final = horzcat(pts_final,pts_aux);
68 end
69 midpoint_first = midpoint_last;
70 aux = size(pts_final);
71 nr_puncte = nr_puncte + aux(2);
72 else
73     if midpoint_first == 0
74         j = int16(length(pts)/2);
75         midpoint_first= pts(:,j);
76         midlevel = 1;
77     end
78     middle = int16(length(pts)/2);
79     pts_aux = pts(:,1:middle-1);
80
81     if(pts_final == 0)
82         pts_final = pts_aux;
83     else
84         pts_final = horzcat(pts_final,pts_aux);
85     end
86 end
87
88
89 j = 1;
90 lung_traiectorie= 0 ;
91 size_pts = size(pts_final);
92 nr_puncte = size_pts(2);
93 while j < size_pts(2)
94     first_point = pts_final(:,j);
95     second_point = pts_final(:,j + 1);
96     lung_traiectorie = lung_traiectorie + sqrt((second_point(1,1) -
97     first_point(1,1))^2 + (second_point(2,1) - first_point(2,1))^2 +
98     (second_point(3,1) - first_point(3,1))^2);
99     j = j + 1;
100 end
101
102 elapsed_time = toc;
103 disp(['Lungimea traiectoriei: ' , num2str(lung_traiectorie) , ' metri']);
104 disp(['Durata de texecuie: ' , num2str(elapsed_time), ' secunde']);
105 disp(['Numarul de puncte utilizat:' , num2str(nr_puncte)]);
106
107 % csvwrite('data_modified_without_obst.csv', pts_final);
108 % Salvarea traiectorie intr-un sfier de tip csv
109
110 %% environment cu obstacole -> 7 puncte -> Algoritmul Bézier extins
111 tic;
112 xlim([0 3.5])
113 ylim([0 3])
114 zlim([0 2])
115 t = linspace(0,1,101);
116 t_aux = linspace(0,1,5);
117
118 % 7 puncte

```

```
119 % matrix_points = [0.3,1.2,1.4,1.8,1.4,1.3,0.8; 0.8,0.6,0.2,0.8,1,1.4,1.3;
120 % 0.4,0.4,0.4,0.7,0.7,0.7];
121 % 10 puncte
122 matrix_points = [0.4,0.6,0.9,1.2,1.4,1.7,1.9,1.8,2.1,1.5;
123 % 0.9,1.5,1.1,1.4,1,1.4,1.1,1.9,2,2.1;
124 % 0.4,0.4,0.4,0.7,0.7,0.7,0.7,0.3,0.3];
125 plot3(matrix_points(1,:),matrix_points(2,:),matrix_points(3,:),
126 'o','MarkerFaceColor','k');
127 hold on
128 grid on;
129 % Obstacol 1
130 length_obs = 1.1;
131 width_obs = 0.75;
132 height_obs = 1;
133
134 vertices = [
135     1 1 0;
136     length_obs 1 0;
137     length_obs width_obs 0;
138     1 width_obs 0;
139     1 1 height_obs;
140     length_obs 1 height_obs;
141     length_obs width_obs height_obs;
142     1 width_obs height_obs
143 ];
144
145
146 faces = [
147     1 2 6 5;
148     2 3 7 6;
149     3 4 8 7;
150     4 1 5 8;
151     1 2 3 4;
152     5 6 7 8
153 ];
154
155 patch('Vertices', vertices, 'Faces', faces, 'FaceColor', 'cyan',
156 'EdgeColor', 'black');
157 hold on
158 % Obstacol 2
159 length_obs = 1.6;
160 width_obs = 1.45;
161 height_obs = 1;
162
163 vertices = [
164     1.5 1.7 0;
165     length_obs 1.7 0;
166     length_obs width_obs 0;
167     1.5 width_obs 0;
168     1.5 1.7 height_obs;
169     length_obs 1.7 height_obs;
170     length_obs width_obs height_obs;
171     1.5 width_obs height_obs
172 ];
173
174
175 faces = [
```

```

176      1 2 6 5;
177      2 3 7 6;
178      3 4 8 7;
179      4 1 5 8;
180      1 2 3 4;
181      5 6 7 8
182 ];
183
184 patch('Vertices', vertices, 'Faces', faces, 'FaceColor', 'cyan',
        'EdgeColor', 'black');
185 hold on
186
187
188 i = 1;
189 midlevel = 0;
190 midpoint_first = 0;
191 matrix_aux = zeros(3,4);
192 pts_final = 0;
193 pts_aux = 0;
194 nr_puncte = 0;
195 var = 1;
196 while i<=length(matrix_points)
197     var = 1;
198     while var<=4
199         if(i<=length(matrix_points))
200             matrix_aux(:,var) = matrix_points(:,i);
201             var = var+1;
202             i = i+1;
203         else
204             break;
205         end
206     end
207     if (i< length(matrix_points))
208         i = i-1;
209     end
210     pts = kron((1-t).^3,matrix_aux(:,1)) +
kron(3*(1-t).^2.*t,matrix_aux(:,2)) +
kron(3*(1-t).*t.^2,matrix_aux(:,3)) + kron(t.^3,matrix_aux(:,4));
211     if midlevel == 1
212         middle = int16(length(pts)/2);
213         midpoint_last= pts(:,middle);
214         middle_point = pts(:,1);
215         pts_midpoint = kron((1-t).^2,midpoint_first) +
kron(2*(1-t).*t,middle_point) + kron(t.^2,midpoint_last);
216
217         P1 = pts_aux(:,end);
218         P2 = pts_midpoint(:,2);
219         pts_second_curve = kron((1-t_aux).^2,P1) +
kron(2*(1-t_aux).*t_aux,midpoint_first) + kron(t_aux.^2,P2);
220         pts_final = horzcat(pts_final,pts_second_curve(:,2:end));
221
222     if (i >= length(matrix_points))
223         pts_aux = pts_midpoint(:,2:end-2);
224         pts_final = horzcat(pts_final,pts_aux);
225         P1 = pts_midpoint(:,end-1);
226         P2 = pts(:,middle+1);
227         midpoint_first = pts_midpoint(:,end);
228         pts_second_curve = kron((1-t_aux).^2,P1) +
kron(2*(1-t_aux).*t_aux,midpoint_first) + kron(t_aux.^2,P2);
229         pts_final = horzcat(pts_final,pts_second_curve);
230         pts_final = horzcat(pts_final,pts(:,middle+2:end));

```

```

231     else
232         pts_aux = pts_midpoint(:,2:end);
233         pts_final = horzcat(pts_final,pts_aux);
234     end
235     midpoint_first = midpoint_last;
236     aux = size(pts_final);
237     nr_puncte = nr_puncte + aux(2);
238 else
239     if midpoint_first == 0
240         j = int16(length(pts)/2);
241         midpoint_first= pts(:,j);
242         midlevel = 1;
243     end
244     middle = int16(length(pts)/2);
245     pts_aux = pts(:,1:middle-1);
246
247     if(pts_final == 0)
248         pts_final = pts_aux;
249     else
250         pts_final = horzcat(pts_final,pts_aux);
251     end
252 end
253 % var = 2;
254 end
255
256
257 j = 1;
258 lung_traiectorie= 0 ;
259 size_pts = size(pts_final);
260 while j < size_pts(2)
261     first_point = pts_final(:,j);
262     second_point = pts_final(:,j + 1);
263     lung_traiectorie = lung_traiectorie + sqrt((second_point(1,1) -
264     first_point(1,1))^2 + (second_point(2,1) - first_point(2,1))^2 +
265     (second_point(3,1) - first_point(3,1))^2);
266     j = j + 1;
267 end
268
269 plot3(pts_final(1,:),pts_final(2,:),pts_final(3,:));
270
271 elapsed_time = toc;
272 disp(['Lungimea traiectoriei: ' , num2str(lung_traiectorie) , ' metri']);
273 disp(['Durata de texecuie: ' , num2str(elapsed_time), ' secunde']);
274 disp(['Numarul de puncte utilizat:' , num2str(nr_puncte)]);
275
276 % csvwrite('data_modified_obst.csv', pts_final);
277 % Salvarea traiectorie intr-un fiier de tip csv

```

### 6.3 Anexa 3. Cod in Python

```

1 import logging
2 import time
3
4 import cflib.crtp
5 from cflib.crazyflie import Crazyflie
6 from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
7 from cflib.utils import uri_helper
8 from cflib.positioning.position_hl_commander import PositionHlCommander

```

```

9 from examples.autonomy.autonomousSequence import start_position_printing,
10    display_coord_crazyflie
11 from examples.autonomy.autonomousSequence import reset_estimator
12
13 import pandas as pd
14 import numpy as np
15
16 # Calea către fișierul CSV
17 file_path = 'C:\\\\Users\\\\mereu\\\\Documents\\\\Licenta\\\\data_modified_obst.csv'
18
19 # Citirea datelor din fișierul CSV într-un DataFrame pandas
20 df = pd.read_csv(file_path)
21
22 matrix_coord = np.empty((3, df.shape[1]))
23
24 i = 0
25 for column in df.columns:
26
27     try:
28         matrix_coord[0][i] = float(column)
29     except Exception:
30         column_aux = column.split('.')
31         column_aux2 = column_aux[0] + '.' + column_aux[1]
32         matrix_coord[0][i] = float(column_aux2)
33
34     for index in range(len(df)):
35         if index == 0:
36             matrix_coord[1][i] = df[column][index]
37         elif index == 1:
38             matrix_coord[2][i] = df[column][index]
39         i += 1
40
41 # Verificare dacă coordonatele sunt citite corecte
42 # for row in matrix_coord:
43 #     for element in row:
44 #         print(element, end=' ', )
45 #     print()
46
47
48
49 URI = uri_helper.uri_from_env(default='radio://0/80/2M/E7E7E7E5')
50 # Only output errors from the logging framework
51 logging.basicConfig(level=logging.ERROR)
52
53 if __name__ == '__main__':
54     # Initialize the low-level drivers
55     cflib.crtp.init_drivers()
56
57     with SyncCrazyflie(URI, cf=Crazyflie(rw_cache='./cache')) as scf:
58         cf = scf.cf
59
60         print('Resetarea estimatorilor')
61         reset_estimator(scf)
62
63         print('Parametrii încep să fie setați')
64         start_position_printing(scf)
65         cf.param.set_value('loco.mode', 2) # 1=TWR, 2 = TDoA2, 3 = TDoA3
66         while not scf.cf.param.is_updated:
67             time.sleep(1.0)
68         print('Parametrii au fost setați')

```

```
69
70     with PositionHlCommander(cf,
71         controller=PositionHlCommander.CONTROLLER_PID) as pc:
72         for i in range(len(matrix_coord[0])):
73             x = matrix_coord[0][i]
74             y = matrix_coord[1][i]
75             z = matrix_coord[2][i]
76             pc.go_to(x, y, z)
77             time.sleep(0.1)
78             print('Going to {}, {}, {}'.format(x, y, z))
79
80     matrix_crazyflie = display_coord_crazyflie(scf)
81
82     cf.commander.send_stop_setpoint()
83     # Hand control over to the high level commander to avoid timeout
84     and locking of the Crazyflie
85     cf.commander.send_notify_setpoint_stop()
86
87     # Salvarea coordonatelor dronei într-un fișier extern
88     df_coord = pd.DataFrame(matrix_crazyflie)
89     file_path =
90     "C:\\\\Users\\\\mereu\\\\Documents\\\\Licenta\\\\crazyflie_coord.xlsx"
91     df_coord.to_excel(file_path, index=False,
92                       header=False) # index=False și header=False pentru a nu
93     include indicii sau antetele coloanelor
```