Raport științific – proiect PN-III-P1-1.1-TE-2016-0737, nr. 32/2018

Etapa 1 – Stabilirea claselor de specificații și proiectarea domeniului de evoluție

Activitatea 1.1. Clase de specificații pentru echipe de roboți mobili

***Obiectiv**: Identificarea și definirea specificațiilor complexe pentru roboți mobili. Analiza soluțiilor existente și identificarea unei clase de specificații fezabile pentru rezolvarea problemei propuse.*

Această secțiune a raportului este dedicată prezentării succinte a unor rezultate marcabile pentru domeniul proiectului. Pe viitor, intenționăm să ne inspirăm din unele din aceste rezultate pentru a extinde contribuțiile recente ale membrilor echipei, descrise în secțiunea 1.2.

Planificarea traiectoriei formațiilor de roboți mobili este un domeniu de cercetare care atrage din ce în ce mai multă atenție, rezultatele fiind utilizate în aplicații variate. În acest context, o problemă fundamentală de navigare o constituie cazul unui singur robot mobil care plecând dintr-o poziție inițială trebuie să ajungă într-o poziție finală evitând obstacolele din spațiul de lucru [1,2]. Pentru rezolvarea acestei probleme au fost propuse soluții bazate pe funcții potențial care să poată lucra direct în spațiul continuu sau abordări care să divizeze spațiul continuu într-o descriere finită cu stări (de exemplu descompuneri în celule) [2,3]. Discretizarea spațiului continuu este utilizată preponderent în aplicații complexe, cum ar fi îndeplinirea unor sarcini descrise de specificații boolene [4,5] sau prin formule de logică liniară temporală [6-8].

Descompunerea în celule reprezintă descrierea spațiului de lucru prin regiuni convexe [3]. Această descompunere permite rezolvarea problemelor de planificare prin abstractizarea roboților mobili la puncte [1].

După descompunerea spațiului de lucru se construiește un graf de conectivitate în care nodurile reprezintă celulele iar arcele dintre noduri definesc adiacența celulelor. Graful de conectivitate este parcurs de un algoritm de găsire a traiectoriei care va genera o secvență de celule a căror parcurgere asigură îndeplinirea scopului. În funcție de scop, secvența de celule poate să includă celule libere sau celule de interes în care se pot realiza acțiuni. După stabilirea secvenței de celule care îndeplinește misiunea, traiectoria parcursă de robot este constituită din mijloacele segmentelor care definesc adiacența dintre celule [1,2]. Avantajul acestei abordări este complexitatea computațională redusă deoarece problema identificării traiectoriei dorite se reduce la o căutare într-un graf. Totuși, există un dezavantaj major al acestei metode și anume faptul că pot fi generate soluții cu lungime mare în raport cu distanța necesară de parcurs. Având în vedere acest dezavantaj, distanța minimă poate fi considerată ca restricție dar efectul acestei specificații este formarea unei probleme neliniare de optimizare [9,10]. O soluție propusă [11] o reprezintă trei relaxări suboptime, fiecare necesitând rezolvarea unei probleme de programare liniară (LPP), având complexitate polinomială. Aceste relaxări substituie distanța totală Euclidiană prin suma normelor L_1, L_2 pătrat, L_∞ a segmentelor ce formează traiectoria.

Multitudinea de probleme care pot fi definite pentru sisteme cu roboți mobili pot fi reduse la trei categorii:

1. Se consideră un sistem de roboți mobili modelat dinamic căruia i se atașează un spațiu de lucru limitat cu obstacole. Se dorește proiectarea unor legi de mișcare folosind informații locale accesibile roboților vecini pentru a crea o structura predefinită a formației. În paralel se dorește ca inter-coliziunile dintre roboți precum și coliziunile cu obstacole să fie evitate.

2. Fie un sistem de roboți mobili modelat dinamic în sensul celui mai apropiat vecin. Prin definirea unui task de nivel înalt fiecărui robot se dorește proiectarea unor legi de mișcare descentralizate astfel încât fiecare robot să îndeplinească specificația dorită considerând restricții de timp.
3. Se consideră un sistem de roboți mobili supus perturbațiilor și având eventuale erori de modelare. Prin asignarea unor taskuri specifice fiecărui robot, care pot include și alți roboți din sistem, se dorește proiectarea unor legi de mișcare care să garanteze îndeplinirea sarcinilor.

Fiecare dintre aceste probleme poate fi descrisa prin diferite clase de specificații. În cele ce urmează vom prezenta cele mai utilizate formalisme pentru specificarea sarcinilor de mișcare ale roboților mobili, punctând expresivitatea limbajului si complexitatea problemei de generare a strategiilor de control care sa asigure îndeplinirea unei specificații date.

Formal, dat fiind un mediu de lucru și un set de regiuni de interes *Y*, sarcinile de mișcare ale robotului vor fi definite peste setul tuturor ieșirilor posibile iar problema de rezolvat consta in generarea automata (daca este posibil) a strategiilor de control care asigura satisfacerea unei specificații date. Unele simboluri din setul *Y* pot corespunde la acțiuni robotice ce se pot efectua in regiunile aferente.

1. Expresii regulate

Expresiile regulate [12,13] sunt descrieri scurte si convenabile ale limbajelor regulate, care sunt seturi de secvențe finite acceptate de automatele finite. Astfel, expresiile regulate definite peste $Y=\{y_1, y_2, ..., y_{|Y|}\}$ pot fi utilizate pentru a specifica comportamente de terminare ale roboților. Spre exemplu,

- *vizitarea unor regiuni într-o ordine dorita* ("vizitează y_1, apoi y_2" poate fi exprimat prin $(y_2 + y_{3+...+} y_{|Y|}))^* \cdot y_1 \cdot (y_1 + y_{3+...+} y_{|Y|}))^* \cdot y_2$, unde * denota închiderea Kleene a unui set, + denotă disjuncția (în acest caz uniunea de seturi), iar · denotă concatenarea.
- *urmărirea unui tipar* ("din y_1 mergi in y_2 si apoi in y_3, evitând toate celelalte regiuni" poate fi exprimat cu $y_1 \cdot y_2 \cdot y_3$)

Deși acest formalism este apropiat de limbajul natural, specificarea unor cerințe mai complicate poate devine foarte dificila. Problema generării strategiei de control plecând de la o specificație data sub forma unei expresii regulate peste setul de intrări (sau etichete asociate tranzițiilor) este bine înțeleasă pentru un sistem finit de tranziții [12, 13]. Ea poate fi rezolvata si pentru expresii definite peste setul de ieșiri atât timp cat automatul este deterministic.

2. Expresii ω-regulate

Pentru comportamente continue ale roboților (de exemplu in sarcini de supraveghere), expresiile ω-regulate peste *Y* pot fi o opțiune potrivita. Astfel de expresii codifica seturi de secvențe de lungime infinita ce pot fi acceptate de automate Büchi. Spre exemplu,

- *sarcini de supraveghere* ("vizitează y_1 si y_2 infinit de des" poate fi exprimata prin $((y_1 + y_{3+...+} y_{|Y|}))^* \cdot (y_2 + y_{3+...+} y_{|Y|}))^* \cdot y_2 \cdot (y_2 + y_{3+...+} y_{|Y|}))^*)^\omega$, unde simbolul $^\omega$ denota un număr infinit de repetiții).

Ca si in cazul expresiilor regulate, dificultatea consta in maparea specificațiilor date peste setul de stări sau de regiuni de interes, pentru sintetizarea unu controler.

3. Logici temporale

Logicile temporale au apărut ca framework-uri pentru specificarea si verificarea corectitudinii programelor pe calculator si au devenit utilizate in multe alte domenii datorita apropierii de limbajele naturale. Formulele logicii temporale reprezintă o opțiune buna pentru specificarea mișcării roboților pentru trei motive. In primul rând, sunt expresive si apropiate de limbajul natural. In al doilea rând, sunt deja disponibili algoritmi pentru evaluarea valorilor de adevăr ale expresiilor. In al treilea rând, formulele de logica temporala sunt

definite peste setul de stări sau peste setul de ieșiri ale unui sistem de tranziție, spre deosebire de expresiile regulate si ω-regulate.

Principalele limbaje din aceasta categorie sunt Logica Temporala Lineara (LTL) si Computation Tree Logic (CTL).

Formulele ***LTL*** [14, 15] sunt create folosind un set de propoziții atomice de interes (în cazul nostru, aceste propoziții vor fi regiunile de interes din setul *Y*), operatori logici ("negație" (eng. "not", notat cu ¬), "sau" (disjuncție, eng. "or", notat cu ∨), "și" (conjuncție, eng. and, notat cu ∧), "implicație" (notat ⇒), "echivalență" (notat ⇔)) și operatori temporali ("până când" (eng. "until", notat *U*), "cândva" (eng. "eventually", notat ◊), "mereu" (eng. "always", notat □)). Spre exemplu,

- "ajungi cândva în y_1, evitând tot timpul obstacolele y_2 și y_3" este exprimată prin formula LTL "$\Diamond y_1 \wedge \Box\neg(y_2 \vee y_3)$",
- vizitarea în viitor a oricărei din regiunile y_1 sau y_2 este exprimată prin "$\Diamond (y_1 \vee y_2)$",
- vizitarea în viitor a regiunii y_1 și rămânerea acolo mereu este sugerată de formula "$\Diamond\Box y_1$"

Formulele LTL sunt interpretate pe secvențe de lungime infinită generată de mișcarea robotului (chiar dacă evoluția robotului se oprește într-o anumită poziție, cuvântul generat va fi extins la infinit prin repetarea ultimei poziții). Acest aspect facilitează însă exprimarea prin specificații LTL a unor cerințe care implică o mișcare permanentă a robotului mobil (cum ar fi supravegherea unor regiuni).

In general, modelul robotului utilizat pentru abstractizarea discreta este bazat pe sisteme de tranziții sau procese Markov de decizie, adică pe un model bazat pe grafuri. Sarcina de nivel înalt data sub forma unei formule LTL este transformata într-un automat Büchi sau Rabin. Traiectoriile robotului pot fi calculate prin efectuarea produsului sincron ale modelului echipei cu automatul Büchi sau Rabin si utilizarea unui algoritm de găsire a celui mai scurt drum într-un graf (ce prezintă o complexitate timp polinomiala). Totuși, daca numărul de roboti din echipa creste, numărul de stări ale modelului echipei creste considerabil, fiind necesara efectuarea produsului sincron al diferitelor sisteme de tranziție ca in [16] sau duplicarea automatelor roboților ca in [17].

În [18], specificațiile au fost descrise prin entități LTL, fiind propusă o metodă de planificare în care modelarea spațiului de lucru este construită pe baza situațiilor de deadlock ce pot să apară. Aceste presupuneri pot fi interpretate ca restricții de mișcare a roboților care pot fi relaxate în momente de timp când nu prezintă interes pentru legea de control a sistemului. Aplicațiile acestei abordări au fost considerate în mai multe scenarii printre care și echipe de drone care pot fi utilizate în stingerea incendiilor [19].

Recent, pentru aplicațiile care necesită planificarea de mișcare a unei echipe de roboți precum și transportarea unor obiecte statice a fost propus un framework hibrid folosind specificații LTL [20, 21]. Frameworkul include legi de control pentru navigarea liberă de către roboți și preluarea obiectelor din regiunile specificate fără a exista coliziuni între roboți. Modelarea evoluției roboților și a stării obiectelor se face folosind un sistem de tranziții finite care este utilizat în proiectarea unei abordări de nivel înalt ce satisface specificații LTL.

CTL [14] poate fi de asemenea utilizat pentru specificarea sarcinilor roboților mobili. Adițional operatorilor temporali utilizați in LTL, acest limbaj permite cuantificarea de-a lungul drumurilor posibile dintr-o stare data prin utilizarea operatorilor A („pentru toate rulările") si E („pentru unele rulări"). Totuși, formulele CTL sunt restricționate astfel încât fiecare operator temporal trebuie imediat precedat de un cuantificator de drum. LTL si CTL sunt incomparabile, în sensul că există formule CTL ce nu pot fi exprimate în LTL și vice-versa. Un dezavantaj al CTL este acela că translarea din limbaj natural in formule CTL este

predispusă erorilor, pe când LTL este mai intuitiv. Mai mult, se pare ca există mai multe specificații interesante exprimabile in LTL si nu in CTL.

CTL* este un framework unificator pentru LTL si CTL [14]. Totuși, verificarea modelului pentru CTL* este costisitoare, iar expresivitatea acestui limbaj este prea complicata pentru robotica.

O altă abordare recenta, Metric Interval Temporal Logic (MITL) este descrisă în [22]. Autorii propun un framework de generare automată a secvențelor de control pentru sisteme multi-agent supuse la restricții de timp. Mișcarea agenților în spațiul de lucru este abstractizată în sisteme de tranziție individuale. Fiecărui agent îi este asignată o formulă în MITL, în paralel întregului sistem de agenți fiindu-i asignată o formulă colaborativă. Frameworkul este bazat pe metode de sinteză corecte prin construcție, astfel garantând că sistemul în buclă închisă va satisface specificația.

4. Logica Booleana

Planificarea acțiunilor unei echipe de roboți mobili poate proiectată astfel încât să poată îndeplini specificații Booleene atașate regiunilor de interes [4]. Detaliile de constructie ale unei astfel de specificaţii sunt prezentate in capitolul următor. Acestea sunt construite global pentru echipă fără a permite alocări de tipul robot singular – task. Acest obiectiv poate fi realizat prin modelarea sub formă de reţea Petri a mişcării echipei de roboţi mobili şi a proprietăţilor regiunilor de interes. Formula Booleana ataşată modelului discret este reprezentată printr-un set de inegalităţi liniare ale unor variabile binare, evaluarea acestor variabile fiind strâns legată de posibile secvenţe finite de tranziţii din reţeaua Petri. Se obţine astfel o problemă ce poate fi rezolvată printr-o abordare ILP (integer liniar programming). Soluţia obţinută este optimă în raport cu distanţa parcursă de echipă, evitându-se posibilele cazuri de congestie.

Concluzii A1.1*:*

În opinia noastră, un limbaj de specificare este „bun” dacă este *natural* (apropiat de limbajul uman), *expresiv* (permite specificarea unei clase bogate de sarcini) și algoritmii pentru analiza și sinteza controler-ului au *complexitate mică*.

În cele ce urmează vom adopta o soluție bazată pe logica booleana. Pentru mai mult de un robot aceasta specificație impune o cerință globala asupra vizitării sau evitării regiunilor, fără a permite impunerea de cerințe individuale precum vizitarea a doua regiuni disjuncte de către un același agent. Totuși, aceasta expresivitate mai mica, combinata cu un model bazat pe rețele Petri, permite obținerea de soluții ale căror complexitate este puțin influențată de numărul de roboți, topologia modelului rămânând neschimbată atunci când cardinalitatea echipei de roboți se modifică.

Activitatea 1.2. Algoritmi pentru planificarea mișcării

Obiectiv: *Dezvoltarea de algoritmi pentru planificarea mişcării echipei de roboti mobili pe baza specificaţiilor complexe.*

Secțiunea curentă descrie dezvoltările recente ale membrilor echipei proiectului privind planificarea automată a unei echipe de agenți mobili pe baza specificațiilor complexe. De asemenea, vor fi punctate aspecte pe care ne propunem să le investigăm și să le extindem în etapele următoare.

În cele ce urmează, propunem problema planificării unei echipe de roboti identici astfel încât sa fie îndeplinită o specificaţie Booleana definita peste un set de regiuni de interes. Mediul robotic este cunoscut si static. Specificaţia impune cerinţe Booleene asupra regiunilor vizitate in timpul mişcării echipei precum si asupra poziţiilor finale ale roboţilor. Specificaţia este furnizata global pentru echipa de roboti, fără a permite asignări specifice

robot-sarcina. Pentru a dezvolta o soluţie, propunem modelarea mişcării echipei si a satisfacţiei asociate regiunilor folosind un sistem cu evenimente discrete sub forma unei Reţele Petri cu ieşiri. Astfel, evităm problema dimensiunii spaţiului stărilor pentru modelul echipei de roboti, problema care apare în abordarea bazată pe sisteme de tranziții și modelarea echipei ca un produs al acestor sisteme (conducând la o creștere exponențială a numărului de stări) . Modelele bazate pe reţele Petri sunt scalabile in raport cu dimensiunea echipei de roboti, sub constrângerea ca aceştia sa fie identici. Adăugarea unui robot in echipa nu presupune modificarea structurii reţelei Petri, ci doar adăugarea unui nou jeton.

Astfel, presupunem o echipă de n roboți mobili identici care pot colabora, fiind controlați de o unitate centrală. Domeniul de lucru conține un set de regiuni de interes statice și este partiționat pe baza acestora, folosind algoritmi existenți, de exemplu bazați pe descompuneri în celule. Evoluția posibilă a echipei de roboți în domeniul de lucru este abstractizată folosind un model de tip rețea Petri cu ieșiri, notat în continuare Q. Pentru obținerea lui Q se construiește întâi o rețea Petri cu structura $N = <P, T, F>$, unde P și T sunt mulțimile finite de locații, respectiv tranziții, iar $F \subseteq (P \times T) \cup (T \times P)$ este setul arcelor. Ideea de bază în crearea rețelei N este de a asigna o locație fiecărui element al partiției mediului de evoluție (celulă), iar tranzițiile corespund mișcărilor posibile ale unui robot dintr-o celulă în alta adiacentă. Pentru simplitatea expunerii considerăm că toate arcele au pondere unitară, dar acest lucru este nerestrictiv, putându-se asigna ponderi bazate pe anumite funcții de cost dependente de exemplu de distanța medie parcursă sau de energia consumată pentru mișcarea unui robot din locația curentă într-una adiacentă a partiției, sau pentru efectuarea unei acțiuni.

Fiecare robot din echipă corespunde unui jeton ale rețelei Petri, astfel modelul Q având structura $Q = <N, \boldsymbol{m}_0, \Pi, h>$, unde $\boldsymbol{m}_0$ este marcajul inițial, adică vectorul cu valori întregi arătând în ce locații sunt plasați roboții la momentul inițial. $\Pi=\{\Pi_1, \Pi_2, \ldots, \Pi_{|\Pi|}\} \cup \varnothing$ este de ieșiri (observabile), unde regiunile de interes și eventual acțiunile ce pot fi efectuate în anumite locații sunt notate cu Π_i, iar $\varnothing$ notează observația vidă (o locație din domeniul de evoluție care nu aparține unei regiuni de interes și în care nu pot fi efectuate acțiuni). Funcția de observații $h : P \to 2^{\Pi}$ arată ieșirea fiecărei locații a lui Q, adică atunci când cel puțin un jeton este în locația p, $h(p)$ indică regiunea (sau regiunile) de interes în care se află roboții respectivi și/sau ce acțiuni pot efectua în poziția curentă. Pseudo-codul pentru construcția modelului Q este dat în Algoritmul 1, iar detalii suplimentare pot fi găsite în [4]. Algoritmul 1 este dedicat doar vizitării regiunilor de interes, dar precum a fost menționat unele simboluri Π_i pot referi acțiuni ce pot fi efectuate în anumite regiuni, de exemplu culegerea unor resurse din locații în care astfel de resurse există plasate.

Tranzițiile lui Q corespund mișcărilor robotice, iar starea m (poziția jetoanelor/roboților) obținută în urma execuției tranzițiilor dintr-un așa-numit vector de executare σ (care contorizează numărul de execuții ale fiecărei tranziție) este dată de ecuația de stare a rețelei Petri (1), unde $\boldsymbol{C}$ notează matricea de incidență a modelului:

$$\boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma} \tag{1}$$

De menționat că modelul Petri obținut din Algoritmul 1 este o mașină de stare, fiecare tranziție având o singură locație de intrare și o singură locație de ieșire, ceea ce garantează că orice soluție m, σ a ecuației (1) corespunde unei stări ce poate fi atinsă, vectorul σ corespunzând unei secvențe fezabile de tranziții.

Algorithm 1: Construct the PN system $\mathcal{Q}$.

Input: Environment, regions Π, initial team deployment
Output: Team model $\mathcal{Q}$

Construct a cell decomposition of the environment based on the polygonal regions of interest from Π;
Associate each cell from decomposition to a place from P; let $P = \{p_1, p_2, \ldots, p_{|P|}\}$;
Let $T = \emptyset$, $F = \emptyset$, $\boldsymbol{w} = \mathbf{0}$;
for $p_i \in P$ **do**
    for $p_j \in P$, $p_i \neq p_j$ **do**
        if cells p_i and p_j are adjacent **then**
            Add transition $t_{i,j}$ to T;
            $F := F \cup \{(p_i, t_{i,j}), (t_{i,j}, p_j)\}$;
            $\boldsymbol{w}[t_{i,j}]$ = average distance traveled by a robot that moves from cell p_i to p_j;
for $p_i \in P$ **do**
    $\boldsymbol{m}_0[p_i]$ = no. of robots initially deployed in cell p_i;
    $h(p_i) = \{\Pi_j \in \Pi | \text{cell } p_i \text{ is included in region } \Pi_j\}$;

În cele ce urmează vom considera o specificație pentru misiunea echipei dată printr-o *formulă bazată pe logica Booleană*, după cum urmează. Se impune o formulă Booleană pe setul $P = P_t \cup P_f$, unde $P_t = \Pi$, iar $P_f = \{\pi_1, \pi_2, \ldots, \pi_{|\Pi|}\}$. Operatorii Booleeni folosiți sunt $\neg$ (negație), $\wedge$ (conjuncție), $\vee$ (disjuncție) și se consideră că formulele sunt în forma canonică conjunctivă. Seturile P_t și P_f se referă la aceleași regiuni și acțiuni, dar P_t sugerează că regiunile/acțiunile sunt vizitate/executate de-a lungul traiectoriilor roboților, iar P_f include cerințe ce trebuie îndeplinite în starea finală a echipei (atunci când roboții se opresc). Formalismul privind interpretarea semantică a formulelor pe baza secvențelor de observabile ale modelului Q poate fi găsit în [4] și se bazează pe așa-numiții vectori caracteristici v_i ai observației Π_i, i=1,...,n. De exemplu, specificația $\varphi = (\Pi_1 \vee \Pi_2) \wedge \neg\pi_1 \wedge \neg\Pi_3$ impune ca de-a lungul traiectoriilor robotice regiunile sau acțiunile Π_1 sau Π_2 să fie vizitate/efectuate, Π_3 să nu fie evitată totdeauna, iar Π_1 să nu fie adevărată în starea finală (de exemplu niciun robot să nu se oprească în zona notată Π_1).

Lucrarea [4] dezvoltă o soluție algoritmică pentru planificarea echipei de roboți pe baza oricărei specificații Booleene construită având în vedere sintaxa și semantica de mai sus. Ca idee de bază, specificația este translată într-o serie de inegalități lineare pe baza unor variabile binare cuprinse într-un vector $\boldsymbol{x}$, iar pentru rețeaua Petri se consideră o serie de k marcaje (stări) intermediare, între două marcaje succesive un jeton putându-se mișca prin cel mult o tranziție. Problema este apoi translată într-o problemă de programare liniară mixtă cu numere întregi (eng. Mixed Integer Linear Programming - MILP), ecuația (2) formalizând această scriere. Pentru detalierea notațiilor adiționale din (2) cititorul este rugat să consulte referința [4].

$$
\begin{aligned}
& \min \lambda \cdot \boldsymbol{w}^T \cdot \sum_{i=1}^{k} \boldsymbol{\sigma}_i + \mu \cdot b \\
\text{s.t.} \quad & \boldsymbol{m}_i = \boldsymbol{m}_{i-1} + \boldsymbol{C} \cdot \boldsymbol{\sigma}_i, i = 1, \ldots, k \\
& \boldsymbol{m}_{i-1} - \boldsymbol{Pre} \cdot \boldsymbol{\sigma}_i \geq 0, i = 1, \ldots, k \\
& \sum_{\gamma \in \mathcal{P}} (\alpha_i(\gamma) \cdot x_\gamma) \geq 1 + \sum_{\gamma \in \mathcal{P}} \min(\alpha_i(\gamma), 0), \quad \forall \varphi_i \\
& N \cdot x_\gamma \geq \boldsymbol{v}_\gamma \cdot \boldsymbol{m}_k, \quad \forall \gamma \in \mathcal{P}_f \\
& x_\gamma \leq \boldsymbol{v}_\gamma \cdot \boldsymbol{m}_k, \quad \forall \gamma \in \mathcal{P}_f \\
& N \cdot (k+1) \cdot x_\gamma \geq \boldsymbol{v}_\gamma \cdot \left(\sum_{i=0}^{k} \boldsymbol{m}_i \right), \quad \forall \gamma \in \mathcal{P}_t \\
& x_\gamma \leq \boldsymbol{v}_\gamma \cdot \left(\sum_{i=0}^{k} \boldsymbol{m}_i \right), \quad \forall \gamma \in \mathcal{P}_t \\
& \left(\boldsymbol{Post} \cdot \sum_{i=1}^{k} \boldsymbol{\sigma}_i \right) \leq b \cdot \mathbf{1}^T \\
& \boldsymbol{m}_i \in \mathbb{N}_{\geq 0}^{|P|}, \boldsymbol{\sigma}_i \in \mathbb{N}_{\geq 0}^{|T|}, i = 1, \ldots, k \\
& \boldsymbol{x} \in \{0,1\}^{|\mathcal{P}|}, b \geq 0.
\end{aligned}
\tag{2}
$$

Problema de optimizare (2) este rezolvată folosind rutine software existente, obținându-se un vector de executare fezabil σ pentru modelul Q. Ulterior, σ este translat în planuri de mișcare și executare de acțiuni pentru echipa de roboți mobili. Drept exemplu, considerăm domeniul de evoluție din Figura 1, în care există 3 roboți și 5 regiuni de interes, și specificația $\varphi = \neg\Pi_2 \wedge \Pi_1 \wedge \neg\pi_1 \wedge \pi_3 \wedge \pi_4 \wedge \pi_5$. Specificația impune ca regiunea 2 să fie evitată, regiunea 1 să fie vizitată de-a lungul traiectoriilor, dar niciun robot să nu se oprească în ea, iar în ultimele 3 regiuni să se oprească câte un robot. Traiectoriile robotice obținute sunt reprezentate în Figura 1. Problema MILP (2) a inclus 1891 variabile (din care 1400 întregi și10 binare), 480 de restricții de egalitate și 554 inegalități liniare, fiind rezolvată în aproximativ 0,01 secunde.

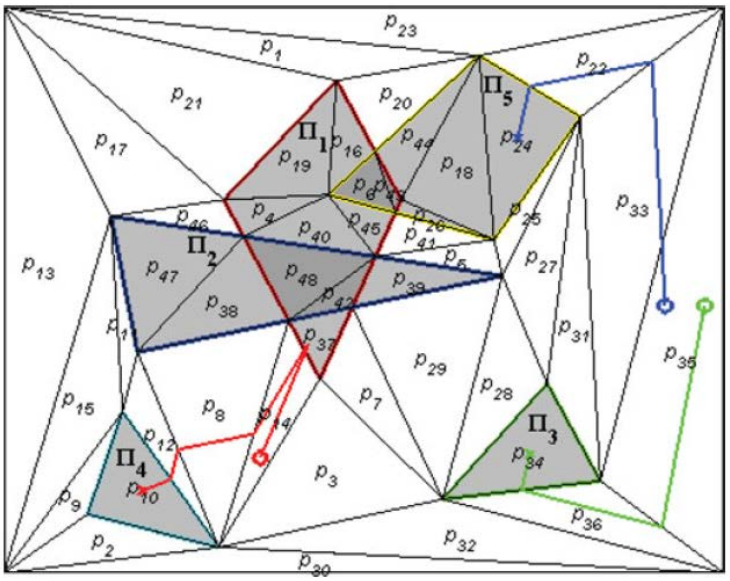

Figura 1. Domeniu de evoluție cu 5 regiuni și 3 roboți plasați inițial în pozițiile marcate cu "○". Modelul Q are 48 de locații și 140 de tranziții, iar traiectoriile obținute sunt reprezentate cu linii frânte, roboții oprindu-se în punctele marcate cu "x".

Concluzii A1.2:

Este clar că o clasă de astfel de specificații Booleene include multe specificații robotice utile, dar are și unele limitări. Pe de o parte, față de formule LTL, utilizatorul nu poate impune o ordine specifică în care regiunile sau acțiunile sunt satisfăcute de-a lungul traiectoriilor. Dar, formulele Booleene cu soluția bazată pe modele de tip rețea Petri aduc importante avantaje computaționale față de formulele LTL și algoritmii de rezolvare bazați pe sisteme de tranziții, formalism în care rezolvarea exemplului anterior ar fi putut dura până la câteva ore. În viitor, ne propunem tratarea formulelor LTL folosind tot modele de tip rețea Petri, în speranța de a obține soluții în timpi mult mai scurți. Pentru aceasta, ne vom baza pe pașii preliminari din [23], unde formulele sunt restricționate la subclasa co-safe LTL și nu includ acțiuni. Pe de altă parte, formulele pe care le considerăm deocamdată sunt globale, ceea ce înseamnă că echipa de roboți trebuie să îndeplinească misiunea, fără a putea impune anumite restricții unui anume robot. Acest fapt limitează deocamdată posibilitatea de a avea acțiuni de genul "ridică piesa A din regiunea 1 și depune-o în regiunea 2", deoarece specificația globală poate impune vizitarea regiunilor 1 și 2 de către roboți diferiți, de exemplu din cauza optimizării numărului total de tranziții. Pe viitor, vom încerca includerea în soluție de atare restricții specifice roboților, folosind pe de o parte modelele de tip rețea Petri și pe de altă parte inspirându-ne din metodele menționate în Secțiunea 1.1 a acestui raport. În plus, cercetările raportate în [24] au condus la soluții pentru probleme simple de planificare fără a utiliza modele de tip rețea Petri, dar transpunând problema într-un formalism apropiat de control predictiv, necesitând tot rezolvarea unor probleme de optimizare intermediare. Ne propunem și investigarea acestor idei pentru specificații mai complexe, în speranța că traiectoriile obținute vor fi mai mult îmbunătățite din punct de vedere al diverselor metrici ce pot fi integrate în funcțiile de cost ale problemelor de optimizare implicate.

Activitatea 1.3. Definirea unui domeniu de evoluție 3D

Obiectiv: *Proiectarea si dezvoltarea unui domeniu de evoluție 3D pentru roboți de tip dronă.*

Specificații generale

Quadcopters (Dronele) sunt sisteme robotice cu patru motoare atractive datorită simplității, agilității acestora cât și gamei largi de aplicații. Aceste platforme robotice sunt utilizate tot mai frecvent în aplicații precum servicii de monitorizare, mentenanță și cartografiere a unei zone de interes, servicii de livrare. Particularitatea quadcopterului este dată de faptul că două elice opuse se rotesc în sensul acelor de ceasornic, iar celelalte două se rotesc în sens contrar. Analiza și proiectarea unor algoritmi de reglare joacă un rol fundamental în aplicațiile cu drone, asigurând astfel stabilitatea, rejecția perturbațiilor, precum și funcționarea optimală cu obținerea performanțelor dorite. În acest context, în ultimii ani, s-a consemnat un interes deosebit pentru integrarea dronelor în aplicații atât pentru medii exterioare, cât și interioare.

Comunicația cu o dronă de mici dimensiuni, Crazyflie 2.0, proiectarea și dezvoltarea unui domeniu de evoluție 3D pentru drone și modelarea dinamicii acesteia [25], [26] precum și controlul și planificarea traiectoriilor cu vizitarea unor regiuni de interes reprezintă subiectele de interes ale acestui studiu. Algoritmul de reglare implică, în primul rând, găsirea unor modele matematice caracterizate printr-un grad ridicat de generalitate, în sensul că trebuie să fie valabile pentru o gamă largă de valori ale mărimilor de intrare, descriind comportarea dronei pe întreaga gamă de funcționare. Mai mult, modelul matematic adoptat trebuie să aibă o structură suficient de complexă pentru a surprinde dinamica, dar în același

timp simplă pentru a facilita proiectarea și implementarea în timp real al algoritmului de control.

Spațiul de lucru în care evoluează quadcopterul este ilustrat în Figura 2 și reprezintă o platformă utilizată pentru controlul roboților mobili înconjurată de o plasă de siguranță și un senzor Kinect amplasat în plafon in centrul platformei. Platforma are dimensiunea 4 x 2.5m, iar înălțimea de 2.8m.

Drona Crazyflie 2.0

Quadcopter-ul Crazyflie 2.0 reprezintă a doua generație a platformei de dezvoltare creată de Bitcraze. Proiectul a fost și este în continuare dezvoltat în paradigma „open source" (licențiere CC BY-SA 3.0) fiind puse la dispoziție atât informațiile despre structura hardware, cât și codul sursă pentru cele două microcontrollere de pe platformă, respectiv pentru aplicațiile de tip client. Caracteristicile generale ale dronei sunt illustrate în Tabel 1.

Tabel 1

Masă	27 g
Dimensiuni (LxÎxA, privire de sus)	92x92x29 mm (fără elicii, cu suporturile de plastic ale motoarelor)
Timp de zbor	Aprox. 7 minute
Timp de încărcare	Aprox. 40 minute

În determinarea ecuațiilor care descriu dinamica quadcopterului este necesară definirea unui sistem de referință neinerțial, în configurație "X", având originea în centrul dronei, și a unui sistem de referință global, sistemul inerțial raportat la centrul Pământului, ilustrate în Figura 2. Menționarea sistemului de referință precizează față de ce sistem se vor raporta măsurătorile preluate de la senzori. Convenția stabilită pentru sistemul de referință inerțial al dronei Crazyflie 2.0 presupune definirea axelor astfel: axa X îndreptată spre Est, axa Y spre Nord și valori pozitive ale altitudinii (axa Z) pentru orientare în sus.

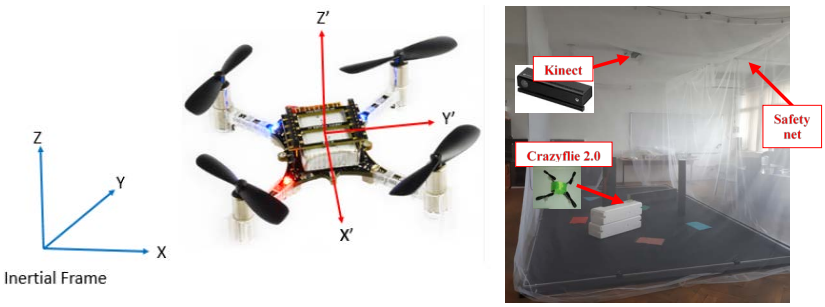

Figura 2. Sistem de referință inerțial CoG, sistem de referință atașat dronei (dreapta) și domeniul de evoluție

Structură hardware

Crazyflie 2.0 este un sistem multiprocesor cu o structură asimetrică ilustrată în Figura 3 și include:

- Un microcontroller cu un nucleu ARM Cortex-M4F (cu unitate de calcul în virgulă mobilă) care rulează algoritmii de control și aplicațiile utilizator;

- Un microcontroller cu un nucleu ARM Cortex-M0 ce include și o interfață radio în banda 2,4 GHz care implementează stiva de comunicație, respectiv mecanismul de control al alimentării dronei;
- O unitate inerțială ce include un accelerometru, un giroscop și un magnetometru. Valorile citite de la cei trei senzori sunt folosite pentru feedback în cadrul algoritmilor de control;
- Driver-ele pentru motoare sunt unidirecționale, fără feedback pentru turație;
- La dronă se pot atașa unul sau mai multe module de expansiune (eng. decks) ce implementează diferite funcționalități - de la afișaje cu LED-uri RGB la sisteme de poziționare relativă, respectiv de detecție a înălțimii de zbor sau a deplasării laterale.

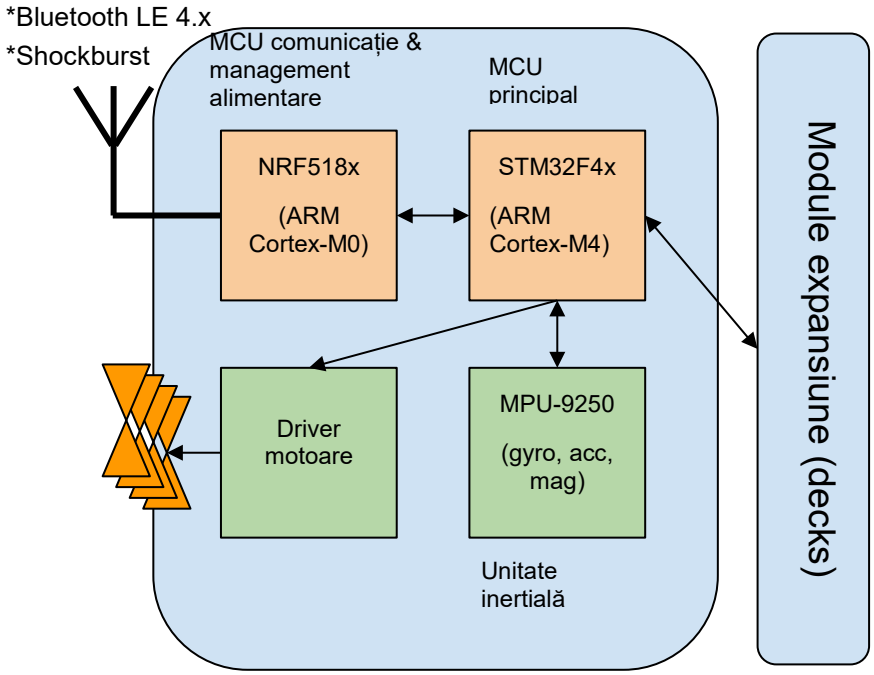

Figura 3 Structura hardware a dronei Crazyflie 2.0

La pornirea dronei, modulele instalate sunt autentificate printr-o interfață 1-wire și în consecință sunt activate modulele software corespunzătoare pentru inițializare și control.

Modelul matematic al dronei Crazyflie 2.0

Determinarea modelului matematic al dronei s-a realizat pe baza ecuațiilor Newton-Euler ce descriu dinamica quadcopterului. Modelul matematic stă la baza implementării unui simulator ce a fost mplementat în mediul Matlab. Parametrii constructivi și mărimile care intervin în modelul matematic sunt în concordanță cu drona Crazyflie 2.0 [26], [28]. Modelul obţinut poate oferi informaţii complete şi despre comportarea subsistemelor componente, în deplină concordanţă cu implementarea fizică a quadcopterului. Astfel, simulatorul construit pornind de la reprezentări matematice își găsește utilizare în testarea prin simulare a unor algoritmi de control pentru specificații de mișcare și efectuare de acțiuni. Limitările pe care acest model le impune rezultă din faptul că relaţiile matematice sunt adesea scrise considerând o serie de ipoteze de lucru și simplificări.

Modelul matematic a fost configurat conform informaţiilor tehnice şi parametrilor reali ai dronei Crazyflie 2.0, fiind validat experimental, iar performanţele sunt apreciate prin comparaţii cu datele reale oferite din funcționarea în buclă deschisă a quadcopterului. În acest scop, este propusă arhitectura din Figura 4, care este alcătuită din următoarele componente: subsistemul Crazyflie 2.0 conținând modelul matematic, blocul algoritmilor de control, un

subsistem pentru interfațarea utilizatorului cu drona și monitorizarea datelor preluate de la senzori și trimiterea de comenzi. Într-o primă etapă, poziționarea dronei și orientarea acesteia într-un domeniu de evoluție indoor, s-a realizat prin intermediul unui senzor extern ce folosește un sistem de evaluare vizuală de tip Kinect. În timpul experimentelor în timp real, s-a observat o întârziere în achiziția datelor prin Kinect și, prin urmare, se dorește utilizarea altor metode de interfațare și control pentru drona Crazyflie 2.0. Determinarea și implementarea unui model matematic facilitează proiectarea și implementarea în timp real a unor algoritmi de control pentru dronă.

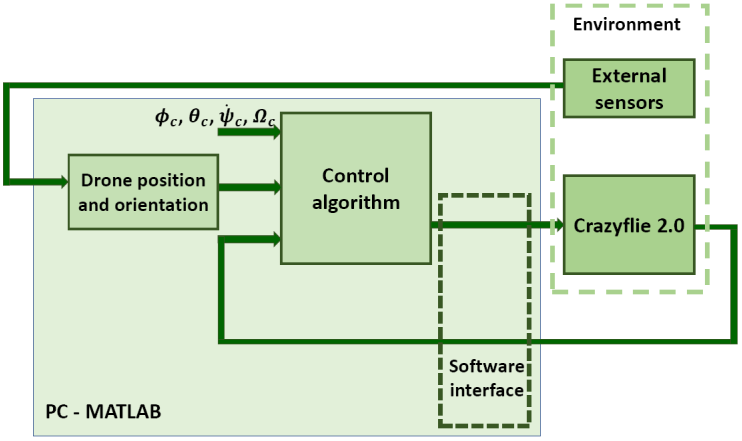

Figura 4 Arhitectura propusă pentru realizarea experimentelor cu drona Crazyflie 2.0

În implementarea modelului matematic s-au folosit 12 stări: pozițiile *x, y* și *z*, exprimate în sistemul de coordonate inerțial (CoG), vitezele liniare ale dronei *u,v,w* de-a lungul celor 3 axe, unghiurile roll (ϕ), pitch (θ) și yaw(ψ) și vitezele unghiulare ale dronei *p*, *q*, *r*. Modelul matematic neliniar este ilustrat în relația (1), în care toate mărimile și parametri constructivi sunt detaliați în articolul [25],[26].

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{r} \\ \dot{q} \\ \dot{p} \end{bmatrix} = \begin{bmatrix} w\left(s_\psi s_\phi + c_\psi c_\phi s_\theta\right) - v\left(s_\psi c_\phi - c_\psi s_\phi s_\theta\right) + u \times c_\psi c_\theta \\ v\left(c_\psi c_\phi + s_\psi s_\phi s_\theta\right) - w\left(c_\psi s_\phi - s_\psi c_\phi s_\theta\right) + u \times s_\psi c_\theta \\ w \times c_\psi c_\phi - u \times s_\theta + v \times s_\phi c_\theta \\ p + r \times c_\phi t_\theta + q \times s_\phi t_\theta \\ q \times c_\phi - r \times s_\phi \\ r\frac{c_\phi}{c_\theta} + q\frac{s_\phi}{c_\theta} \\ rv - qw + g \times s_\theta \\ pw - ru - g \times s_\phi c_\theta \\ qu - pv + \frac{U_1}{m} - g \times c_\theta c_\phi \\ \frac{U_2 + pq(I_{xx} - I_{yy})}{I_{zz}} \\ \frac{U_3 + pr(I_{zz} - I_{xx})}{I_{yy}} \\ \frac{U_4 + pq(I_{zz} - I_{yy})}{I_{xx}} \end{bmatrix} \quad (3)$$

Mărimile de intrare considerate pentru modelul matematic al dronei în formalismul intrare-stare-ieșire sunt exprimate în relația (4), pe baza celor 4 viteze unghiulare ale fiecărui motor al quadcopterului:

$$
\begin{aligned}
U_1 &= C_T\left(\omega_1^2+\omega_2^2+\omega_3^2+\omega_4^2\right)\\
U_2 &= dC_T\sqrt{2}\left(-\omega_1^2-\omega_2^2+\omega_3^2+\omega_4^2\right)\\
U_3 &= dC_T\sqrt{2}\left(-\omega_1^2+\omega_2^2+\omega_3^2-\omega_4^2\right)\\
U_4 &= C_D\left(-\omega_1^2+\omega_2^2-\omega_3^2+\omega_4^2\right)
\end{aligned}
\tag{4}
$$

În firmware-ul Crazyflie 2.0, mărimile de intrare sunt semnale PWM transmise fiecărui motor și sunt reprezentate pe 16 biți, în gama [0-65535]. Între viteza unghiulară a fiecărui motor și semnalul PWM există o relație liniară, dată de:

$$\mathrm{RPM} = 0.2685 \times \mathrm{PWM} + 4070.3 \tag{5}$$

În încercarea de a-si menține poziția de echilibru, într-un punct fix la o anumită altitudine (hover mode), forța generată de elicile dronei trebuie să compenseze greutatea dronei. Pornind de la ipoteza că drona este perfect simetrică, toate cele 4 motoare trebuie să se rotească cu aceeași viteză astfel încât să-și mențină poziția de echilibru într-un punct fix. Validarea modelului matematic dezvoltat s-a realizat prin experimente în timp real, folosind structura internă din Figura 5, având regulatoare încorporate pentru controlul dronei.

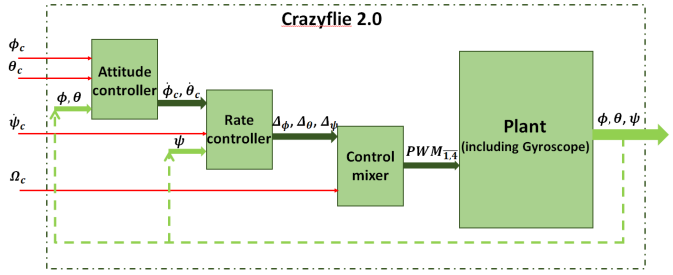

Figura 5 Structura internă cu algoritmi de reglare încorporați

Într-un prim pas se urmăreşte validarea modelului matematic pe setul de date oferit din funcţionarea dronei în circuit deschis, urmând ca apoi performanţele simulatorului să fie testate în circuit închis. În Figura 6 este ilustrată traiectoria dronei pe axa z până la o înălțime de 1.9m, date preluate din simularea numerică, comparativ cu setul de date preluat din funcţionarea în timp real a quadcopterului, rezultatele validând modelul matematic.

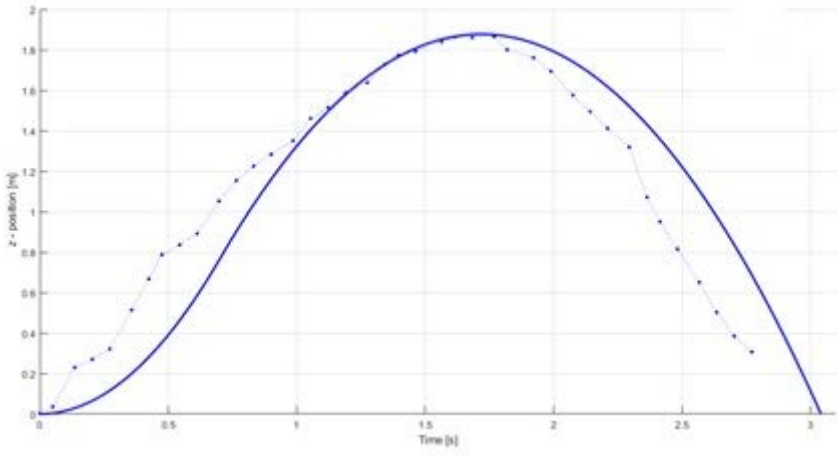

Figura 6 Studiu comparativ: evoluție traiectorie dronă axa z:
simulare numerică (linie continua) și testare în timp real (linie punctată)

În cadrul experimentelor, unghiurile de referință roll și pitch sunt setate la valoarea 0, în contextul în care informaţiile preluate de la sistemul Kinect sunt disponibile cu întârzieri de aproximativ 0.15 s, nefiind fezabile pentru controlul în timp real al dronei, sistemul avand o dinamică rapidă.

Mecanisme de comunicație și interacțiune cu drona

Microcontroller-ul dedicat pentru comunicație implementează o stivă duala bazată pe Bluetooth LE, respectiv Shockburst (protocol proprietar NordicSemi). Comunicația BLE este folosită pentru controlul din aplicațiile mobile (Android, iOS), aplicații ce implementează un control manual, iar comunicația proprietară este folosită pentru implementări mai complexe, prin intermediul unui PC (denumit în continuare client). Astfel, PC-ul trebuie echipat cu un adaptor USB-RF dedicat (CrazyRadio) ce permite comunicarea bidirecțională cu drona (Figura 7).

La nivel aplicație, pentru comunicațiile USB și radio există implementat protocolul CRTP (Crazy Real-Time Protocol), iar pentru comunicația între microcontrollere se folosește protocolul Syslink. Ambele protocoale sunt proprietare Bitcraze.

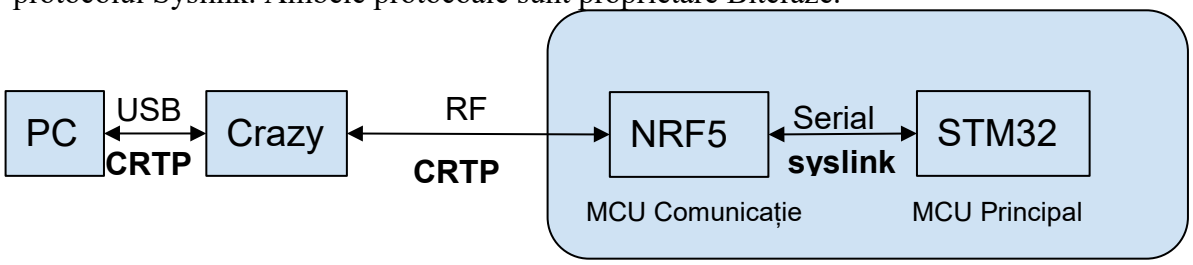

Figura 7 Infrastructura de comunicație între Crazyflie 2.0 și PC

Syslink este un protocol multifuncținal prin intermediul căruia cele două microcontrollere schimbă date, una dintre facilitățile oferite de acesta fiind împachetarea și transmiterea cadrelor CRTP. CRTP definește un mecanism de comunicație bazat pe sarcini (eng. tasks), respectiv funcționalități. Acestea sunt codificate prin intermediul unui antet de 1 octet, după cum urmează:

- Port (4 biți) - identifică sarcina sau funcționalitatea
- Link (2 biți) - nefolosit
- Channel (2 biți) - identifică sub-sarcina sau sub-funcționalitatea

În versiunea curenta, există definite 11 porturi:

Nume	**ID** **(4 biți)**	**Descriere**
Console	0	Port unidirecțional pentru transferul unor șiruri de caractere (stare, depanare etc.) de la dronă către client (similar cu o consolă text).
Parameters	2	Scriere/citire parametri dronă.
Commander	3	Trimitere puncte de control (set-points) pentru regulatoarele implementate pe dronă.
Memory access	4	Accesarea memoriei nonvolatile a dronei (1-Wire și I2C).
Data logging	5	Blocuri de variabile a căror valoare va fi trimisa periodic către PC. Parametrii sunt setați prin intermediul unei macrodefiniții în codul sursa de pe dronă.
Localization	6	Port folosit pentru citirea/scrierea informațiilor de localizare, atunci când există un astfel de sistem disponibil. Definește două

		moduri de localizare: externă și generică.
Generic setpoint	7	Trimitere puncte de control pentru regulatoarele implementate pe dronă (variante de comandă derivate din controlul de bază al port-ului 3)
Setpoint High Level	8	Interfață pentru controlul unor grupuri (eng. swarm) de drone.(variante de comandă derivate din controlul de bază al port-ului 3)
Platform	13	Comenzi diverse pentru dronă (ex. depanare, control alimentare)
Client-side debugging	14	Depanare client - nu se comunică efectiv cu drona.
Link layer	15	Testare legatura radio.

Detalii de implementare

Porturile necesare pentru a controla și pentru a citi informațiile de stare de la dronă sunt *Commander, Parameters* și *Data logging*. Ca interfață, port-urile *Parameters* și *Data logging* sunt similare, însă din punct de vedere funcțional există următoarele diferențe:

- Port-ul *Parameters* este folosit pentru scriere/citirea individuală a variabilelor interne ale dronei. Astfel, pot fi modificate constantele algoritmilor de reglare, pot fi resetați algoritmii, pot fi modificați parametrii de zbor etc.
- Port-ul *Data logging* este folosit pentru citirea uneia sau mai multor variabile în cazul în care frecvența cu care se modifică valorile lor este ridicată. Astfel, poate fi configurată rata de achiziție (cel puțin de ordinul zecilor de milisecunde) și valorile recepționate pot fi salvate la client.

Variabilele care se doresc expuse către interfața CRTP trebuie marcate prin intermediul unui set de macrodefiniții la finalul modulului sursă. Un exemplu din modulul care implementează regulatorul de stabilizare al zborului este prezentat mai jos.

```
PARAM_GROUP_START(pid_attitude)
PARAM_ADD(PARAM_FLOAT, roll_kp, &pidRoll.kp)
PARAM_ADD(PARAM_FLOAT, roll_ki, &pidRoll.ki)
PARAM_ADD(PARAM_FLOAT, roll_kd, &pidRoll.kd)
PARAM_ADD(PARAM_FLOAT, pitch_kp, &pidPitch.kp)
PARAM_ADD(PARAM_FLOAT, pitch_ki, &pidPitch.ki)
PARAM_ADD(PARAM_FLOAT, pitch_kd, &pidPitch.kd)
PARAM_ADD(PARAM_FLOAT, yaw_kp, &pidYaw.kp)
PARAM_ADD(PARAM_FLOAT, yaw_ki, &pidYaw.ki)
PARAM_ADD(PARAM_FLOAT, yaw_kd, &pidYaw.kd)
PARAM_GROUP_STOP(pid_attitude)
```

Pe baza acestui set de macrodefiniții este creată o structură într-o secțiune distinctă (.param) din memoria flash a microcontrollerului, structură ce conține tipul, numele și adresele variabilelor. Accesul in scriere/citire prin intermediul CRTP se realizeaza pe baza numelor stocate in aceste structuri. Prin intermediul CRTP poate fi accesat un cuprins (TOC) al grupurilor de parametri, respectiv al parametrilor, deoarece prin modificarea firmware-ului e posibil sa se adauge sau sa fie eliminati parametri. Mai multe detalii pot fi găsite în referința [28]. Prin intermediul port-ului *Commander* poate fi trimisă o sigură comandă către dronă: *send_setpoint(roll, pitch, yaw_rate, thrust).*

Mărimile *roll* și *pitch* sunt exprimate în grade, *yaw_rate* este exprimat în grade/secundă, iar *thrus*t este exprimat în valoare absolută pe 16 biți - limitată între 1000 și 60000 la recepția comenzilor. În urma testelor efectuate s-a observat că acest mod de control, chiar dacă asigură stabilizarea dronei, suferă din lipsa unui mecanism de reacție pentru a

indica deplasarea dronei pe cele trei axe. Astfel, dacă se dorește zborul la punct fix sau zborul la o anumită înălțime, apare problema reglării (manuale, experimentale) a t*hrust*-ului deoarece drona are nevoie de portanță mai mare la decolare și în apropierea solului. Mai mult, datorită distribuirii inegale a greutății, respectiv a efectelor dinamice (vibrații induse de motoare), drona intră în derivă pe axele x și y, fiind necesar și de această dată un proces de calibrare manuală.

Concluzii A1.3:

Domeniul de lucru dezvoltat este pretabil experimentelor cu dronele de mici dimensiuni Crazyflie 2.0. A fost studiat modelul matematic și structura dronelor, aspecte care vor permite dezvoltarea și testarea ulterioară a diverselor strategii de control. S-au realizat rutine de comunicație cu drona Crazyflie 2.0, urmând extinderea acestora pentru interfațarea cu plăci auxiliare dronei. Pentru controlul în timp real al dronei trebuie asigurat feedback-ul pe poziție și orientare. O încercare în acest sens a fost utilizarea unui senzor Kinect, însă întârzierile aferente achiziției și procesării imaginilor sunt prea mari în raport cu dinamica rapidă a roboților în cauză. Studii viitoare vor fi direcționate spre utilizarea unor echipamente auxiliare pentru preluarea poziției unui quadcopter Crazyflie 2.0.

Bibliografie

[1] H. Choset, K.-M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.-E. Kavraki, and S. Thrun, “Principles of robot motion: Theory, algorithms, and implementations.” Boston, MIT Press, 2005

[2] S. M. LaValle, Planning Algorithms. Cambridge, 2006

[3] M. D. Berg, O. Cheong, and M. van Kreveld, Computational Geometry: Algorithms and Applications, 3rd ed. Springer, 2008

[4] C. Mahulea, M. Kloetzer, "Robot Planning Based on Boolean Specifications Using Petri Net Models", IEEE Transactions on Automatic Control (TAC), vol. 63 (7), pp. 2218 - 2225

[5] E. Vitolo, C. Mahulea, and M. Kloetzer, “A computationally efficient solution for path planning of mobile robots with boolean specifications,” in 21st International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2017, pp. 63–69.

[6] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, “Symbolic planning and control of robot motion [grand challenges of robotics],” IEEE Robotics & Automation Magazine, vol. 14, no. 1, pp. 61–70, 2007.

[7] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, “Temporal logic motion planning for dynamic robots,” Automatica, vol. 45, no. 2, pp. 343–352, 2009.

[8] M. Guo and D. V. Dimarogonas, “Multi-agent plan reconfiguration under local ltl specifications,” The International Journal of Robotics Research, vol. 34, no. 2, pp. 218–235, 2015.

[9] G. Xue and Y. Ye, “An efficient algorithm for minimizing a sum of Euclidean norms with applications,” SIAM Journal on Optimization, vol. 7, pp. 1017–1036, 1997.

[10] L. Qi, D. Sun, and G. Zhou, “A primal-dual algorithm for minimizing a sum of Euclidean norms,” Journal of Computational and Applied Mathematics, vol. 138, no. 1, pp. 127 – 150, 2002.

[11] R. Gonzalez, M. Kloetzer, and C. Mahulea, “Comparative study of trajectories resulted from cell decomposition path planning approaches,” in 21st International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2017, pp. 49–54.

[12] C. Cassandras and S. Lafortune. Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. second edition.

[13] R. Kumar and V. K. Garg. Modeling and Control of Logical Discrete Event Systems. Kluwer, Boston, MA, 1995.

[14] E. Clarke, D. Peled, and O. Grumberg. Model checking. MIT Press, 1999.

[15] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science: Formal Models and Semantics, volume B, pages 995–1072. MIT Press, 1990.

[16] M. Kloetzer and C. Mahulea, “LTL-based planning in environments with probabilistic observations,” IEEE Trans. Autom. Sci. Eng., vol. 12, no. 4, pp. 1407–1420, Oct. 2015.

[17] F. Imeson and S.-L. Smith, “Multi-robot task planning and sequencing using the SAT-TSP language,” in Proc. IEEE Conf. Robot. Autom., May 2015, pp. 5397–5402.

[18] J. Alonso-Mora, J. DeCastro, V. Raman, D. Rus, H. Kress-Gazit, “Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles”, Autonomous Robots, 42(4), pp.801-824, 2017.

[19] J. Shaffer, E. Carrillo, H. Xu, “Receding Horizon Synthesis and Dynamic Allocation of UAVs to Fight Fires”, IEEE Conference on Control Technology and Applications (CCTA), 21-24 August 2018.

[20] C. Verginis, D. Dimarogonas, “Multi-Agent Motion Planning and Object Transportation under High Level Goals”, IFAC-PapersOnLine, Vol. 50(1), July 2017, Pages 15816-15821.

[21] C. Verginis, D. Dimarogonas, “Motion and Cooperative Transportation Planning for Multi-Agent Systems under Temporal Logic Formulas”, arXiv:1803.01579, 2018.

[22] S. Andersson, A. Nikou, D. Dimarogonas, “Control Synthesis for Multi-Agent Systems under Metric Interval Temporal Logic Specifications”, IFAC-PapersOnLine, Elsevier, Vol. 50, p. 2397-2402, 2017.

[23] M. Kloetzer and C. Mahulea, "Multi-robot path planning for syntactically co-safe LTL specifications," *13th International Workshop on Discrete Event Systems (WODES)*, Xi'an, China, pp. 452-458, 2016.

[24] E. Vitolo, C. Mahulea, M. Kloetzer, "Path-planning in Discretized Environments with Optimized Waypoints Computation", IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Torino, Italy, 2018.

[25] S. Huștiu, M. Lupașcu, Ș. Popescu, A. Burlacu, M. Kloetzer, "Stable Hovering Architecture for Nanoquadcopter Applications in Indoor Environments", IEEE 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 2018.

[26] C. Luis and J. Le Ny. “Design of a trajectory tracking controller for a nanoquadcopter”, Technical report, Mobile Robotics and Autonomous Systems Laboratory, Polytechnique Montreal, 2016.

[27] A.A.J. Lefeber. “Controlling of a single drone. Hovering the drone during flight modes.” Technical Report TU/E Eindhoven, Department of Mechanical Engineering, 2015.

[28] Crazyflie 2.0 documentation, available at: https://wiki.bitcraze.io/projects:crazyflie2:index, 2018.