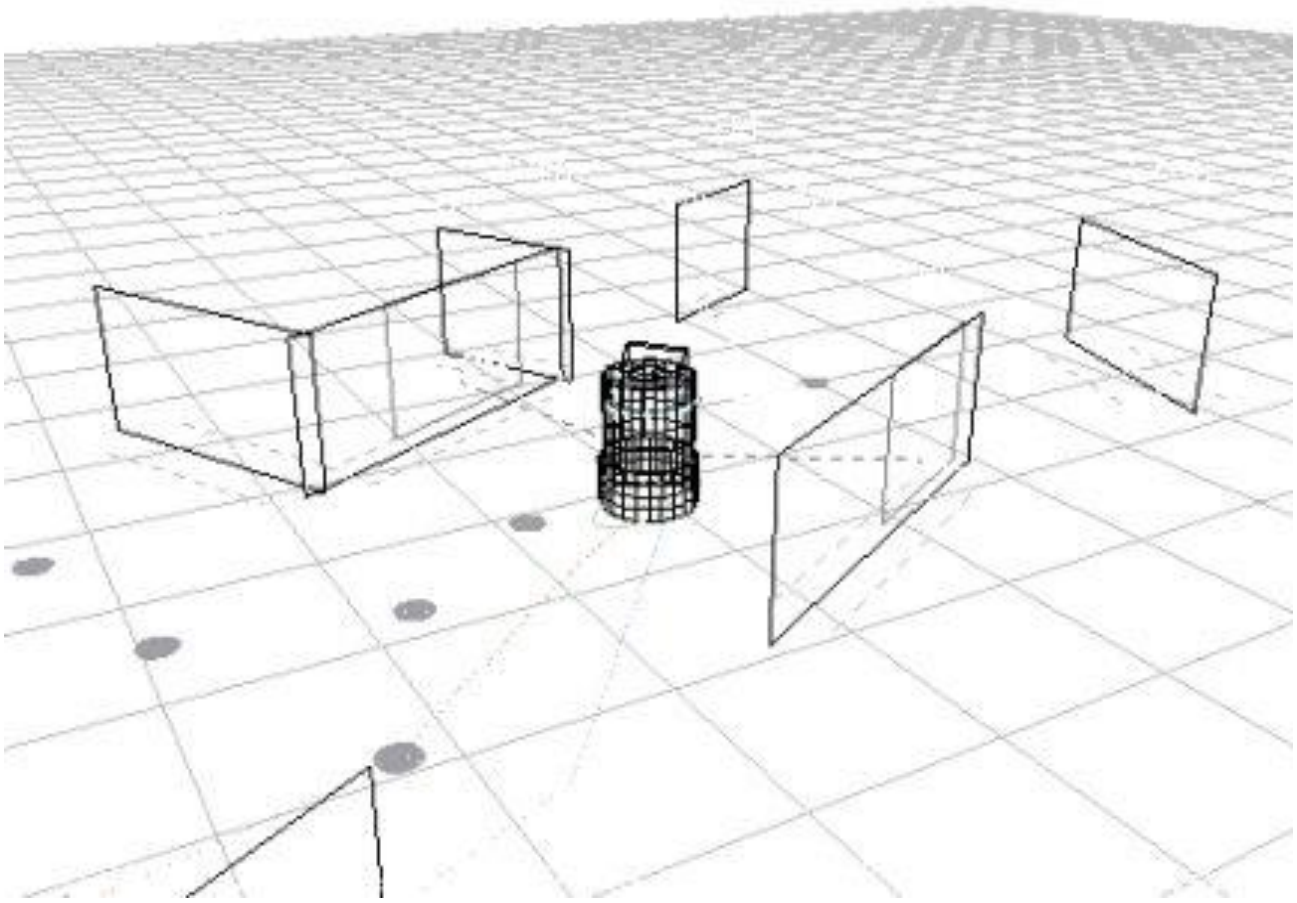


Maparea mediului pentru roboţi mobili cu structură senzorială limitată

Universitatea „Dunărea de Jos”, Galaţi
Facultatea de Automatică, Calculatoare,
Inginerie Electrică şi Electronică
Departament : **Automatică şi Inginerie Electrică**

Lucrare de dizertaţie

Temă : **Maparea mediului pentru roboţi mobili cu structură senzorială limitată**

Domeniu : **Ingineria Sistemelor, Robotică**

Student **: Axenie Cristian**
Specializare : **Informatică Aplicată în Conducere Avansată**
Grupa : **1**
Promoţia : **2009-2011**

Profesor coordonator **:**
Ş.l. Dr. Ing. Şolea Răzvan

Prefaţă

Lucrarea de faţă prezintă detaliile de implementare ale unei aplicaţii originale de mapare a mediului pentru roboţi mobili cu structură senzorială limitată. Obiectivul propus în aceasta teză s-a concretizat ca un nivel adiţional peste aplicaţia dezvoltată în cadrul lucrării de licenţă.

Concret, modulul de mapare proiectat se integrează funcţional cu structura dezvoltată anterior. Pornind de la proiectarea şi implementarea unei structuri minimale de robot mobil diferenţial, s-a dezvoltat apoi o aplicaţie de control în timp real, care se bazează pe sinteza unui controller Sliding Mode pentru operarea robotului în regim de trajectory tracking. Toleranţa la defecte a fost implementată sub forma unui banc de filtre Kalman extinse care prin propretăţile lor specifice au dat rezultate bune în ceea ce priveşte detecţia şi identificarea defectelor, marcate prin variaţia parametrilor sistemului. Pentru a spori autonomia, flexibilitatea şi aplicabilitatea aplicaţiei dezvoltate, aceasta a fost extinsă în lucrarea de faţă cu un nou modul, cel de mapare si localizare continuă, de tip SLAM.

Pe parcursul lucrării vor fi surprinse aspecte descriptive formale, aspecte de sinteză hardware şi software şi analiza rezultatelor obţinute în diferite contexte de operare ale robotului pentru maparea mediului.

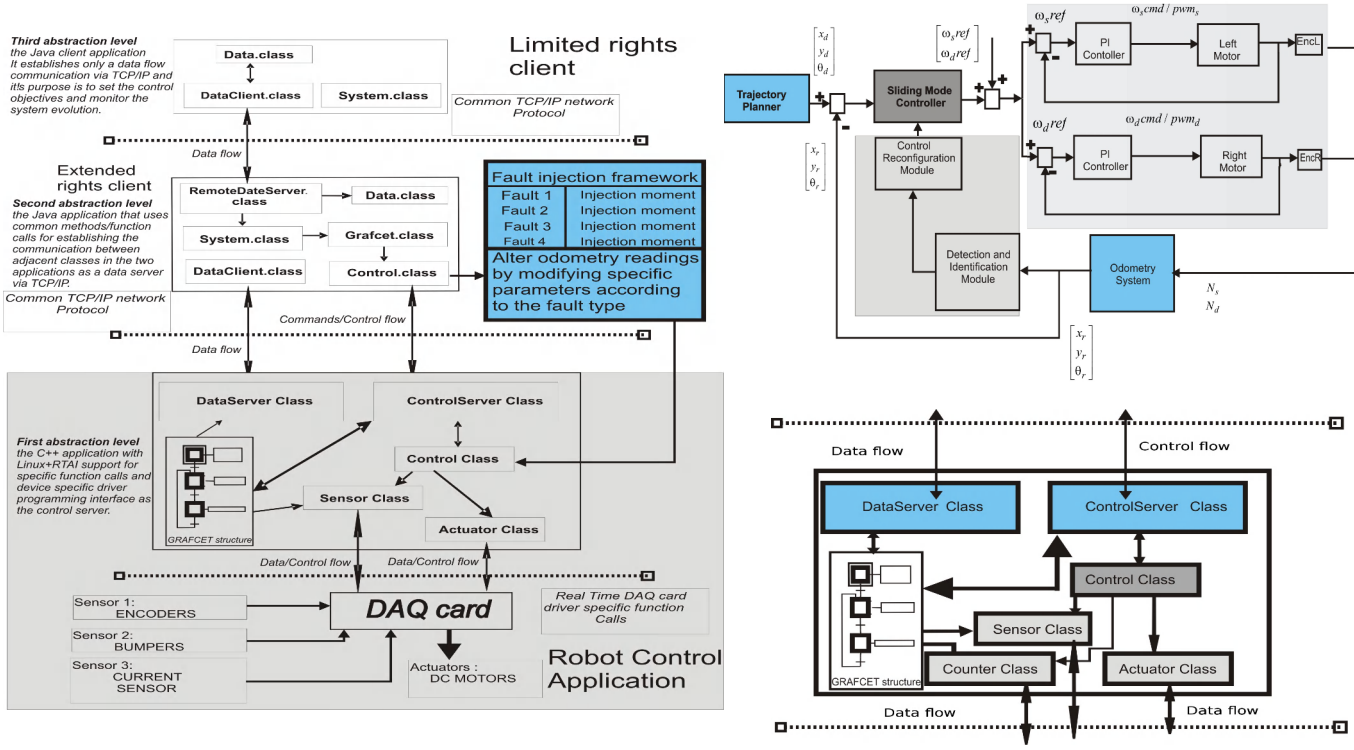
„Atunci când cea mai grea povară este sa trăieşti fără să exişti ...am văzut mai departe decât alţii pentru că m-am ridicat pe umeri de giganţi" şi pentru că am avut alături oameni extraordinari care să mă susţină şi să mă înţeleagă. Mulţumesc mult familiei mele, prietenilor şi colegilor mei şi nu în ultimul rând profesorilor mei.

Mai 2011

MSc. Thesis Report
Environment mapping for limited sensor mobile robots

The current report introduces a novel implementation for limited sensor mobile robots environment mapping. The developed algorithm is an offline SLAM implementation. It uses real time data acquired from the sonar ring and uses this information to feed the mapping module for offline mapping. The latter is running on top of a real time fault tolerant control application for mobile robot trajectory tracking operation. The designed application, that includes both hardware and software design, was developed to fulfill the MSc Thesis requirements.

As mentioned earlier the mapping module is based on a real time fault tolerant control application. This application was developed on a distributed pattern containing two main components, an embedded server running on the robot and a series of distributed clients. A synthetic description of the base application description is next given, focusing on the main software architecture, the distributed client architecture, the real time control loop and the interconnection logic.

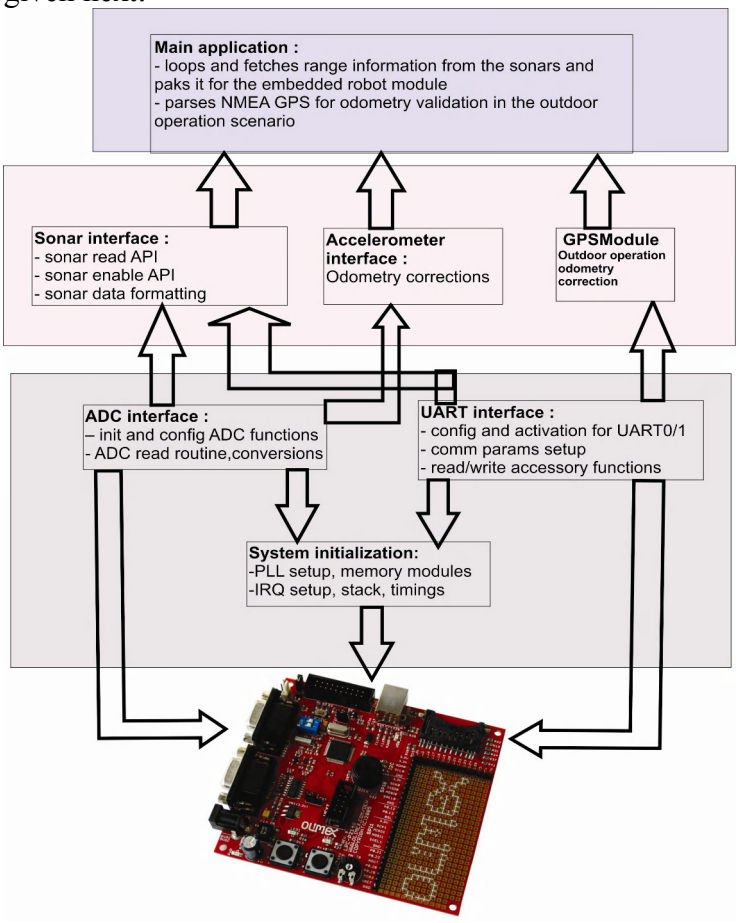


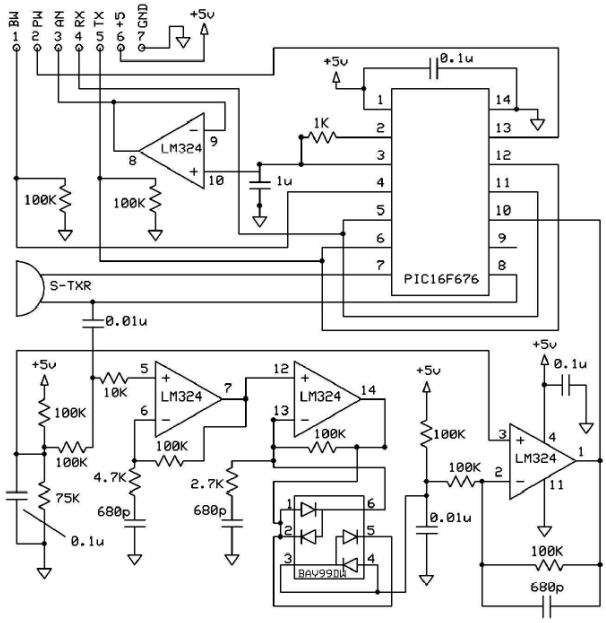
The server application implements a real time control loop to compute the control signal for a compound control loop (PI and Sliding Mode). The control task is executed concurrently with a monitoring and diagnosis task for fault detection and control reconfiguration. The monitoring and diagnosis task synthesizes five Kalman filters for residual computation and fault discrimination based on probability sequences. On the client side the main application enables only manual operation specific commands to access/set the control loop parameters and inject the faults for base platform testing procedure. The distributed architecture is using a typical TCP/IP network and is using message based upper level wrappers.

On top of this layered base level platform a mapping module was synthesized. The glue logic with the base level application is ensured by the main task concurrency mechanisms and by network file system distributed application support. The developed approach is a metric mapping algorithm, using Bayes rule and filter synthesis to implement an occupation matrix based offline SLAM algorithm. Starting from the Bayes rule a Bayes filter is synthesized to implement sensor fusion between odometry information and range information from the ultrasonic sensors. Some analysis details are next given to properly determine how the proposed implementation fits in the mention

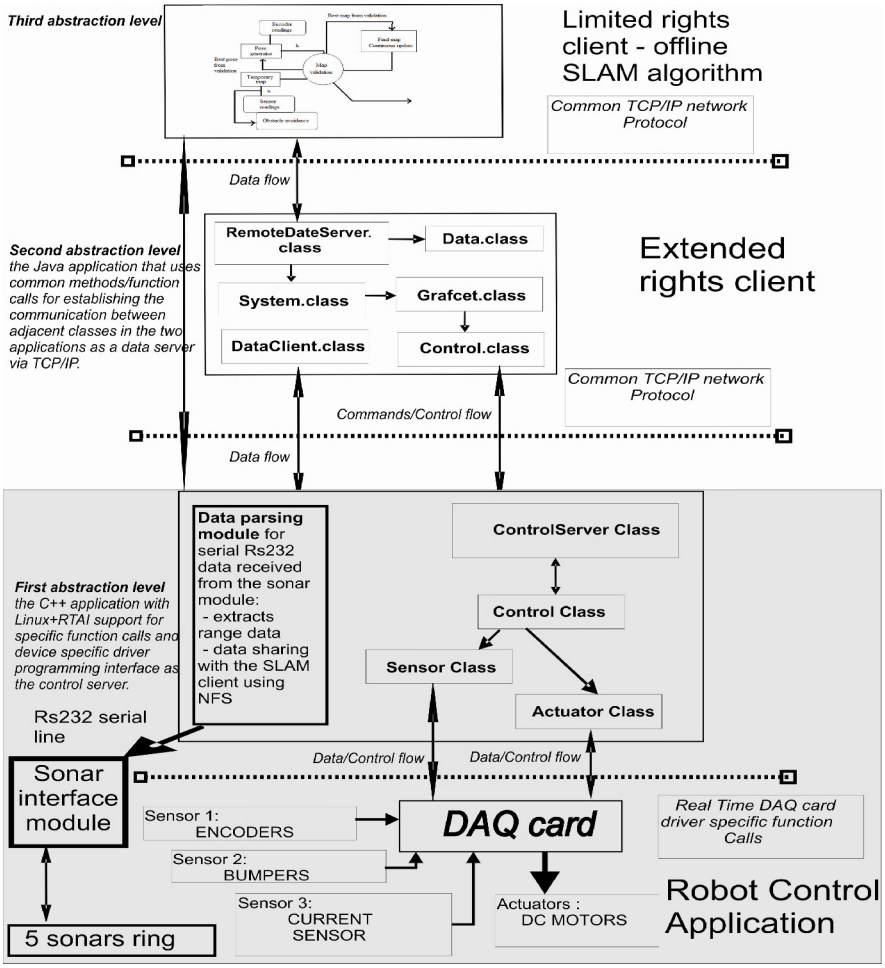
category of algorithms. By implementing an occupation matrix based algorithm uncertainty representation in the map is minimized by using continuously generated auxiliary temporary maps. This short term maps are containing a snapshot of the sensor readings at a certain moment in time. To minimize the error accumulation a specific time interval for data acquisition was defined. This interval will define in fact the amount of information that the temporary map will contain. A second important aspect is regarding the algorithm convergence that is good and it is ensured by the fact that the solution is based on incremental sensor fusion steps. The third aspect that was considered in the design stage was the dimension limits for the maps and the link between the environment features and the acquired data to give a valid solution in a static environment. Another important aspect was based on the quality of robot model, sensor model and environment model. The sensor model was useful to convert the raw measurements into a generic form to carry the conditional probability for each cell of the occupancy matrix given for a sensor reading.

To better emphasize the developed module specific a separate (hardware/software) description is given. So, starting with the low level sonar data acquisition level, an important aspect is to consider the definition of the embedded components. A new board to interface with the sonars was considered. This additional module reads voltage information from the sonar ring and packs it in specific RS-232 packets. The packaged range information is then sent using a serial line to the server application where an additional packet parsing and conversion and serial device initializing routines were added. A synthetic depiction of the range acquisition module along with a schematic of the sonar module is given next.

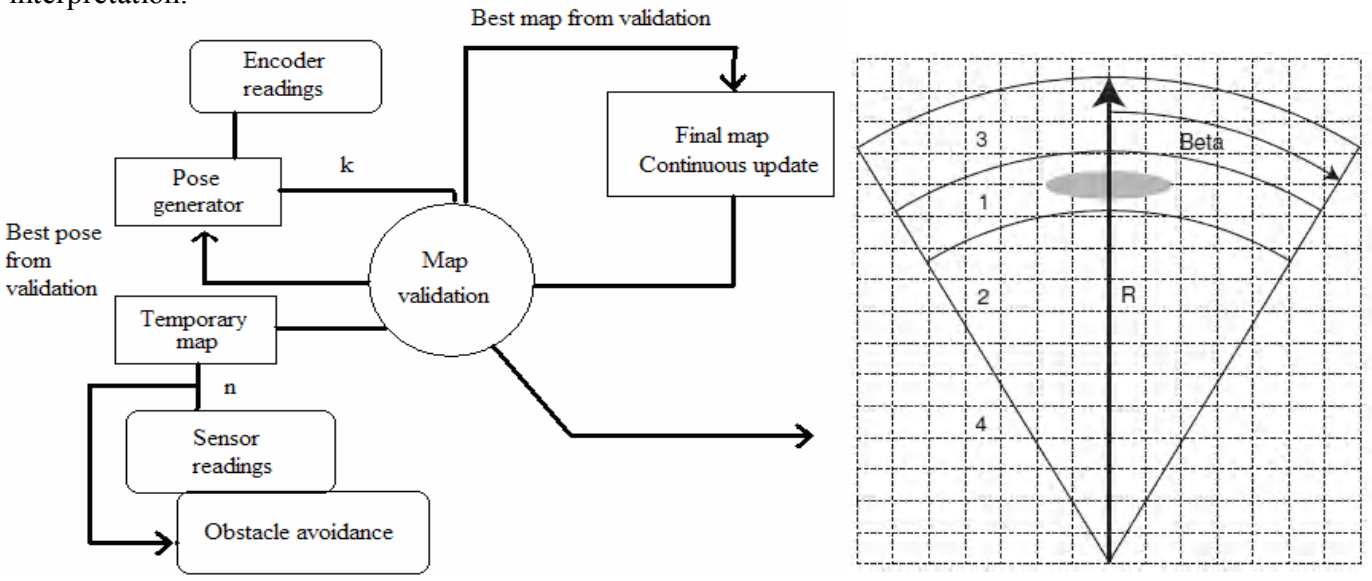




The main application running on the embedded linux machine on the robot includes an adapting layer to support the interface with the sonar interface module and the support for data acquisition used in the sensor fusion stage for the SLAM algorithm. The main architecture of the real time control application was enhanced to include the new functionality. The next image synthesizes the final application layers.



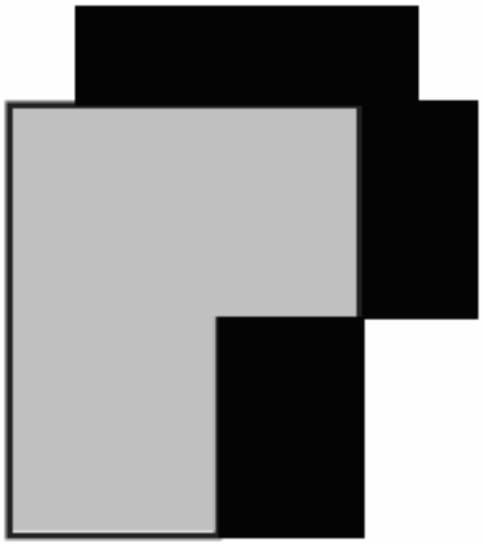
As mentioned earlier the SLAM implementation relies on multiple components. The first component is the sonar interface module found onboard the robot. This module gathers range data from the sonar modules and sends it to the real-time server application over the serial line. The server makes the sonar and odometry data available to the distributed client over a network file system. After the trajectory tracking operation is completed the acquired data is accessed by the client from the shared raw data files. The client then runs the offline SLAM algorithm. To synthesize the SLAM implementation a functional diagram is given along with auxiliary information about region separation used in data interpretation.

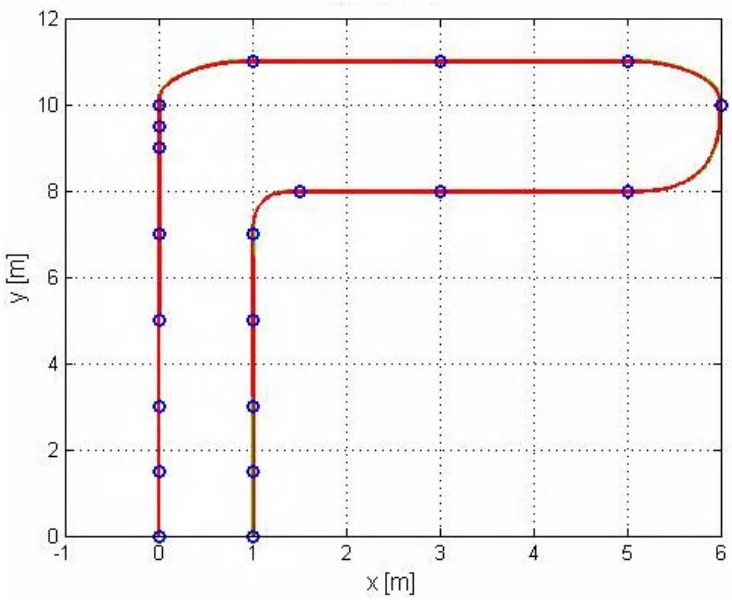


The defined regions are representing the way acquired data is interpreted. Hence, region 1 contains environment elements partially occupied, region 2 contains those environment elements that are probably empty, region 3 defines the uncertain elements (found behind the occupied elements) and finally region 4 defines the sonar out of range elements. In the first stage in map building will use the data acquired from the sonar ring. This data is later interpreted as probability profiles to define occupied and empty regions in the environment. Odometry data is integrated with the sonar data based map. Data integration is composed of multiple steps starting with the initialization of the temporary map to an empty state. Next, the superposition of the empty and occupied zones is done for the existing sensor readings. Following, the threshold computation is made by comparing relative power values for the empty/occupied matrix cells. The Bayes sensor fusion is used to complete the data integration stage by converting the measurements from the sensors into conditional probabilities used in the algorithm to make a generic representation in the map validation stage.

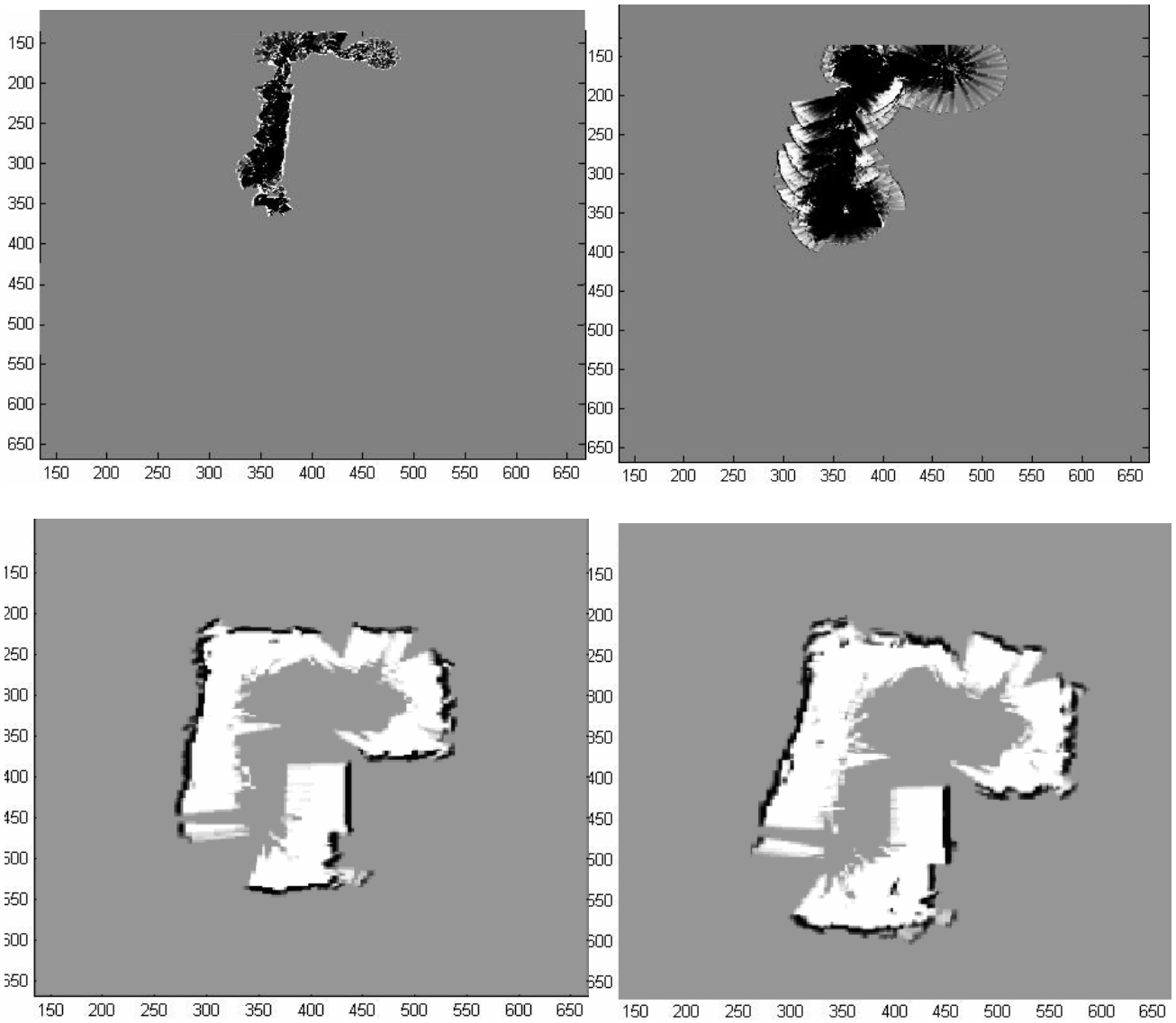
The offline SLAM application is built using multiple functional sub-modules. The first sub-module is responsible with acquiring data from the environment and the navigation logic. The robot shall navigate (trajectory tracking) in the environment and shall acquire data from the sonars and compute odometry data. The current pose of the robot is used also in the real time control algorithm. As one can see in the algorithm synthetic representation the best validated pose will be used to correct odometry data. The temporary map is incrementally built using sensor information until it becomes mature and then a validation stage is required. The validation stage assumes that the temporary map is compared on a per pose basis with the final map. Practically, the algorithm just compares the poses from the two maps and the difference between them is considered to be the odometry error. After updating the robot pose the temporary map is fused along with the final map resulting a new final map and a new iteration will start.

To validate the developed algorithm a test scenario was defined and the results are next presented. The environment configuration and the trajectory tracking profile are depicted next.

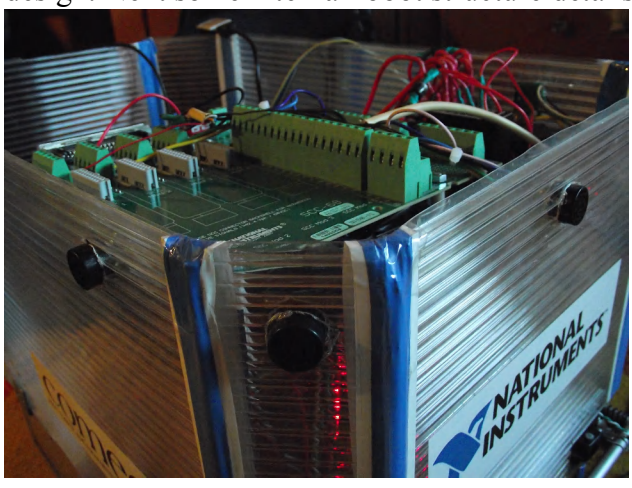


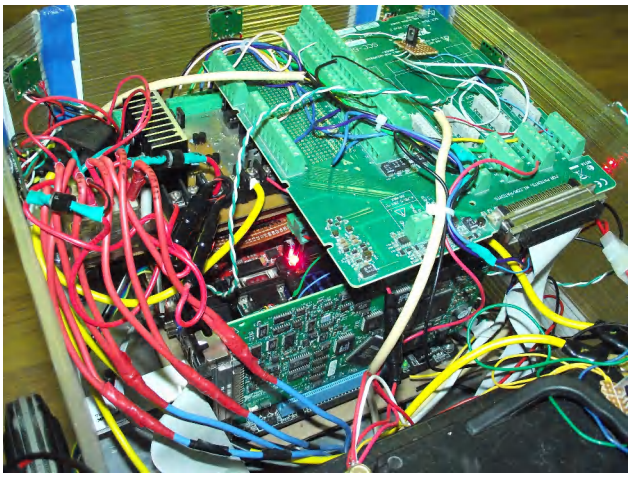


The developed algorithm refines the room representation incrementally, so by adjusting the algorithm parameters one can achieve good quality index in the context of a good and justifiable computational effort. For the given environment configuration the offline SLAM algorithm computed the next representation obtained for various parameters setup (continuous refinement). The upper images are depicting intermediary readings form the temporary maps and the lower image are introducing the final maps generated by the algorithm.



On can observe that the representation is proper and by adjusting the algorithm parameters can obtain a better performance index. The presented environment profile was defined in the real case of validation when the robot had to navigate to half the hallway in the faculty 5th storey. As future work the main focus will be on implementing the same concepts for online real time mapping and also to add an adaptive parametrization stage according to the input environment profile (empty areas, occupation matreix dimension, sensor number). The presented application was a minimal justifiable implementation of SLAM for limited sensor mobile robots that included hardware and software design. Next some internal robot structure details are depicted.





Cuprins

1 . Introducere în problematica mapării mediului **1.....6**

1.1 Prezentarea obiectivului propus pentru dezvoltare 2.....2

1.2 Un review al abordărilor existente asupra mapării mediului pentru roboţi mobili. Probleme specifice 2.....5

1.3 Scurtă prezentare a abordării propuse 5.....6

1.4 Privire de ansamblu asupra structurii lucrării 6.....6

2. Descrierea formală a conceptelor implicate în dezvoltare : Maparea şi localizarea în mediu necunoscut **7.....21**

2.1 Tehnici probabilistice şi modelarea senzorilor ultrasonici pentru maparea mediului 7.....13

2.2 Matrici de ocupare 13.....17

2.3 Fuziunea senzorilor pentru rafinarea matricii de ocupare 17.....19

2.4 Odometria şi localizarea continuă 19.....21

3. Descrierea platformei robotice şi a aplicaţiei de control distribuit ca suport pentru modulul de mapare a mediului **22.....43**

3.1 Descrierea structurii hardware a robotului mobil 22.....33

3.2 Descrierea aplicaţiei de control în timp real 33.....43

3.2.1 Nivelurile funcţionale ale aplicaţiei de conducere 33.....39

3.2.2 Arhitectura distribuită client-server 39.....43

4. Sinteza modulului de mapare a mediului **44.....52**

4.1 Implementarea offline SLAM 44.....50

4.1.1 Proiectarea modulului offline SLAM 45.....47

4.1.2 Parametrizări şi configuraţia senzorilor 47.....48

4.1.3 Configuraţii de medii, navigaţie şi localizare continuă 48.....50

4.2 Analiză, limitări în proiectare şi îmbunătăţiri 50.....52

5. Sinteza nivelului de integrare a modulului de mapare cu aplicaţia de control în timp real pentru platforma robotică **53.....69**

5.1 Suportul hardware pentru modulul de mapare a mediului 53.....63

5.1.1 Descrierea platformei cu sonare 53.....60

5.1.2 Descrierea aplicaţiei de interfaţare cu 60.....63

sonarele şi interacţiunea cu nivelul superior

5.2 Suportul de nivel superior pentru maparea mediului 63.....69

5.2.1 Operarea multitasking concurent pentru conducere şi mapare offline 63.....65

5.2.2 Analiza rezultatelor 65.....69

6. Concluzii 70.....70

7. Bibliografie 71.....73

8. Anexe (Listing, CD-ROM) 74.....95

8.1 Codul sursă C al modului de interfaţare cu senzorii ultrasonici bazat pe MCU LPC2148 (Listing, CD-ROM) 74.....92

8.2 Codul sursă Matlab pentru aplicaţia de mapare Offline (Listing, CD-ROM) 92.....95

Codul sursă C/Linux pentru aplicaţia de control în timp real a robotului mobil (CD-ROM)

Capitolul 1

Introducere în problematica mapării mediului

1. Introducere în problematica mapării mediului

În cursa continuă de a spori capacităţile si autonomia sistemelor robotice, în speţă roboţii mobili, au primit noi funcţionalităţi care să le confere reactivitate, luarea deciziilor acum bazându-se pe informaţia stocată apriori dar şi pe modificările dinamice ale mediului de operare. Astfel, putem considera că aplicaţiile cu roboţi mobili presupun pe lângă sinteza unui controller robust, precis şi eficient pentru componenta motorie şi existenţa unui modul de interpretare a informaţiei senzoriale ce este achiziţionată din mediul de operare şi sintetizată într-o reprezentare internă consistentă, [6]. Maparea mediului de către roboţi este un domeniu de cercetare activ în robotică şi inteligenţă artificială de peste 20 de ani. Aceasta componentă adresează problema achiziţiei modelelor spaţiale ale mediului fizic de operare de către roboţi, asigurându-le astfel sporierea autonomiei. S-au dezvoltat diverse metode de mapare a mediului, pornind de la abordări robuste şi statice, structurate, cu dimensiune finită până la mapare dinamică, nestructurată, la scară mare care ramâne încă o provocare, [5].

Lucrarea de faţă va prezenta o abordare originală în maparea mediului de operare in-door (interior) pentru un robot mobil diferenţial. În general toţi algoritmii de mapare a mediului sunt probabilistici. Unii algoritmi sunt incrementali, având posibilitatea execuţiei în timp real, în timp ce alte abordări necesită o parcurgere recursivă a informaţiei senzoriale şi impun o prelucrare offline, clasă de algoritmi în care se încadrează şi abordarea din lucrarea de faţă, [3]. Continuând, se poate aminti că există algoritmi care au nevoie de poziţia exactă pentru a construi harta mediului, în timp ce alţi algoritmi au nevoie de informaţia de odometrie; unii algoritmi menţin consistenţa între datele achiziţionate din mediu şi momentul achiziţiei în timp ce alte abordări presupun marcarea individuală şi distinctă a caracteristicilor mediului,[1].

Maparea mediului este o operaţie utilizată în mod comun pentru navigaţie (localizare) şi presupune achiziţia informaţiei din mediu utilizând senzori (sonar, laser, infraroşu, radar, interfeţe haptice, giroscop sau GPS). O problemă importantă care determină calitatea mapării rezidă în faptul că senzorii sunt afectaţi de zgomot de măsură şi au game de măsură limitate, [2]. Pornind de la aceste premise în continuare este descris obiectivul propus pentru dezvoltare în cadrul tezei realizând apoi o sinteză descriptivă a metodelor existente pentru maparea mediului în cazul roboţilor mobili.

1.1 Prezentarea obiectivului propus pentru dezvoltare

În lucrarea de faţă este prezentată sinteza unui modul de mapare şi localizare de tip SLAM(Simultaneous Localization and Mapping) utilizând o aplicaţie modulară cu o structură distribuită, eterogenă, pentru un robot mobil diferenţial cu 2 roţi motoare şi o roată directoare. Obiectivul propus s-a concretizat în dezvoltarea a două componente ce au implicat proiectare hardware şi software.

Prima componentă reprezintă modulul hardware şi aplicaţia de interfaţare cu senzorii ultrasonici. Acest modul embedded se bazează pe un MCU NXP ARMv7 LPC2148, iar aplicaţia ce rulează pe această platformă preia informaţia de la modulele cu senzor ultrasonic şi o împachetează pentru a fi transmisă pe o linie serială RS232 către platforma embedded server bazată pe MPU Intel ATOM D525 care execută taskurile de control şi monitorizare. Informaţia de la senzorii ultrasonici este utilizată pentru a asigura navigarea în mediul necunoscut (nu există informaţie apriori) şi este transmisă unei aplicaţii client care va realiza salvarea informaţiei pentru implementarea mapării offline.

A doua componentă este aplicaţia client Matlab care va prelua informaţia brută de la platforma embedded de pe robot şi va executa algoritmul de SLAM implementat, generând offline harta mediului în care robotul a operat. Această componentă va comunica cu aplicaţia server de pe robot prin intermediul unei interfeţe de funcţii API adaptoare. Nivelul adaptor va avea rolul de a exporta o interfaţă de funcţii apelabilă pentru obţinerea de informaţii de poziţie a robotului, emiterea de comenzi şi salvarea de informaţie de stare. Aplicaţia SLAM va fi considerată un framework de test pentru nivelul adaptor şi experimentele efectuate vor ilustra influenţa erorilor de odometrie în crearea hărţii mediului şi vor demonstra eficienţa localizării continue în tratarea acestei probleme. În continuare este prezentat un review asupra metodelor existente de mapare a mediului pentru roboţi mobili pentru a încadra obiectivul propus.

1.2 Un review al abordărilor existente asupra mapării mediului pentru roboţi mobili. Probleme specifice

În cele ce urmează se prezintă un scurt review al istoricului dezvoltării şi abordărilor existente în maparea mediului. S-au concretizat două mari direcţii în maparea mediului şi anume, maparea metrică şi maparea topologică. Hărţile metrice surprind proprietăţile

geometrice ale mediului, în timp ce maparea topologică descrie conectivitatea între diferite zone ale mediului. Una din primele abordări pentru maparea metrică, care este încă utilizată şi îmbunatăţită în aplicaţiile curente, este algoritmul de mapare cu matrice (grilă) de ocupare, dezvoltat de Elfes şi Moravec, care reprezenta hărţile prin celule care modelează spaţiul ocupat şi liber din mediu. O metodă alternativă de mapare metrică a fost propusă de Chatila şi Laumond şi utiliza seturi de poliedre pentru a descrie geometria mediului, [2]. Metode de mapare topologică, dezvoltate de Matarić, Kuipers, reprezentau mediul ca o listă de zone semnificative conectate prin arce care conţin informaţie privind sensul de navigaţie între zone. Deşi distincţia între cele două abordări este mică, se poate considera că hărţile obţinute prin mapare topologică sunt mai precise decât cele obţinute prin mapare metrică. Dacă se doreşte o rezoluţie mai ridicată se va atinge şi un efort computaţional major, care poate fi justificat doar de faptul ca acest lucru va determina rezolvarea problemelor de corespondenţă, probleme care vor fi prezentate în cele ce urmează.

O a doua clasificare a algoritmilor, sunt algoritmii centraţi (orientaţi) pe mediu şi algoritmi orientaţi pe robot, [5]. Hărţile orientate pe mediu sunt reprezentate într-un spaţiu global de coordonate şi entităţile din hartă nu conţin informaţii privind măsurătorile senzorilor care au determinat extragerea acelor elemente, în timp ce maparea orientată pe robot este descrisă în spaţiul măsurătorilor. Maparea orientată pe robot prezenta dezavantaje majore în faţa mapării orientate pe mediu (imposibilitatea extrapolării, imposibilitatea discriminării poziţiilor asemănătoare datorită lipsei geometriei în spaţiul de măsurare), fapt care a determinat utilizarea frecventă a mapării orientate pe mediu, [3]. În anii 90 maparea mediului de către roboţi a fost dominată de tehnicile probabilistice. S-au dezvoltat o serie de instrumente statistice şi algoritmi pentru rezolvarea simultană a problemei mapării şi a problemei induse de localizare a robotului relativ la harta construită incremental (Smith, Self, Cheeseman). Astfel s-a concretizat termenul de SLAM sau CML (Simultaneous Localization and Mapping sau Concurrent Mapping and Localization), [8].

O familie de abordări probabilistice utilizează filtrul Kalman pentru a estima harta şi locaţia robotului, harta rezultată descriind poziţiile unor puncte marcate (landmarks) sau caracteristici semnificative ale mediului, fie utilizând un număr mare de citiri brute de la senzori, [4]. O alternativă o reprezintă abordările bazate pe algoritmul de maximizare a aşteptării (expectation maximization) a lui Dempster, care adresează problema corespondenţei în mapare (corespondenţa între citirea de la senzor la un moment dat şi entitatea fizică reală din mediu), [9]. O a treia familie de tehnici probabilistice se bazează pe

identificarea obiectelor din mediu care corespund tavanului, pereţilor, uşilor sau altor obiecte care se pot mişca, însă sunt surclasate de metode de computer vision, [7].

În continuare sunt prezentate câteva probleme specifice mapării mediului, analizând câteva cazuri particulare de interes şi pentru aplicaţia dezvoltată. După cum menţionam anterior o provocare majoră în maparea mediului o constituie zgomotul de măsură. Problemele de modelare, cum este şi maparea mediului de către robot, sunt relativ uşor de rezolvat de cele mai multe ori atunci când zgomotul în măsurători diferite este statistic independent. În acest caz s-ar efectua din ce în ce mai multe citiri pentru a înlătura efectul zgomotului. În realitate erorile de măsură sunt dependente statistic, deoarece erorile în comanda/odometria robotului se acumulează în timp şi astfel influenţează modul în care sunt interpretate viitoarele măsurători. De exemplu, în cazul în care o eroare de rotaţie de amplitudine redusă apare la un capăt al unui coridor parcurs de robot (operând în mapare) poate determina erori foarte mari şi în capătul opus al coridorului. În continuare este prezentată o figură ce ilustrează şi susţine exemplul.

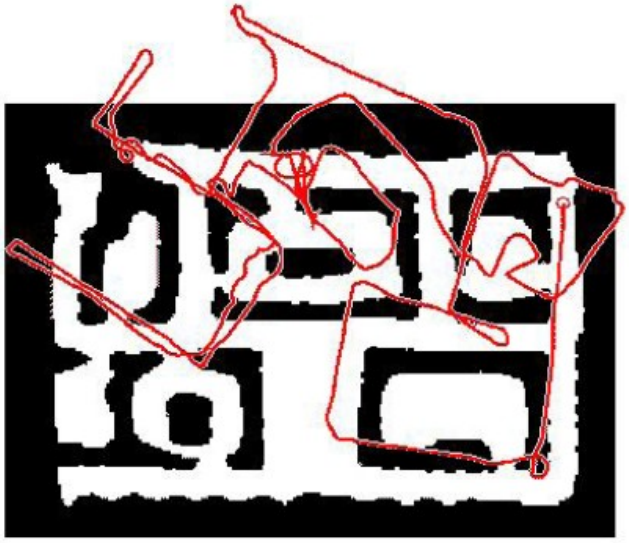


Fig.1 Manifestarea unei erori de odometrie şi influenţa asupra mapării

Figura prezintă eroarea de odometrie relativă la o hartă dată şi se poate trage concluzia că erori mici în comandă / odometrie (erori sistematice) pot determina erori mari în estimările ulterioare ale poziţiei. Încercarea de a compensa erorile sistematice de odometrie determină creşterea complexităţii algoritmului şi din punct de vedere matematic dar şi al implementării în timp real.

Un al doilea aspect de interes se referă la dimensiunile mari ale entităţilor mapate. În acest caz din punct de vedere statistic fiecare număr care reprezintă o entitate topologică majoră devine o dimensiune în problema de estimare.

A treia şi probabil cea mai dificilă problemă în maparea mediului, este problema corespondenţei, adresată şi anterior şi cunoscută şi ca problema asocierii datelor. Această problemă presupune determinarea / validarea corespondenţei între măsurători efectuate la anumite momente de timp şi obiectul fizic din mediu. Problema corespondenţei este una dificilă întrucât numărul de ipoteze posibile poate creşte exponenţial în timp, [6].

A patra provocare se referă la mediile dinamice, care suferă modificări structurale în timp. Unele modificări pot surveni relativ lent însă uneori starea / locaţia entităţilor mediului se poate schimba (uşi, persoane). În acest moment nu există niciun algoritm de mapare care să opereze eficient pentru medii dinamice, în cele mai multe cazuri algoritmii se bazează pe prezumţia unui mediu static în care doar robotul este o cantitate variantă în timp, celelalte entităţi care se mişcă fiind considerate zgomot, [4].

O ultimă problemă specifică mapării mediului se referă la modul în care robotul parcurge mediul în timpul mapării. Taskul generării traiectoriei robotului în cadrul mapării mediului se referă la explorarea mediului. Explorarea mediului este o problema de planificare care este rezolvată deseori sub-optimal, prin intermediul euristicilor. În acest context apar probleme şi se fac compromisuri între cantitatea de informaţie şi timpul de achiziţie a informaţiei. În continuare este redată o scurtă prezentare a abordării propuse.

1.3 Scurtă prezentare a abordării propuse

Aplicaţia dezvoltată îşi propune să realizeze maparea offline a mediului de operare a robotului mobil. Un scenariu demonstrativ presupune setarea robotului în parcurgerea mediului supus mapării în mod trajectory tracking, care presupune o parcurgere a unui traseu prestabilit cu restricţii temporale impuse. În timpul operării în regim trajectory tracking taskul concurent de preluare a informaţiei de la senzorii ultrasonici ce rulează pe modulul cu MCU LPC2148 prelucrează informaţia de distanţă facând-o disponibilă modulului server cu MPU ATOM D525 al robotului. Informaţia este apoi distribuită către clientul conectat la aplicaţia server ce rulează pe robot şi prin intermediul unui interfeţe de funcţii accesor preia informaţia utilă pentru construirea offline a hărţii mediului utilizând SLAM. Taskul de control în timp real calculează comanda emisă către actuatorii robotului la 50ms perioadă de eşantionare, odometria se bazează pe o perioadă de eşantionare de 50ms pentru citirea impulsurilor de la encoderele incrementale şi tot la 50 ms informaţia de distanţă de la sonare este primită pe linia serială RS232 asincron de la modulul de interfaţare cu LPC2148. Informaţia de distanţă este formatată pentru a fi transmisă clientului ce implementează

algoritmul SLAM însă este disponibilă şi în format brut pentru prelucrări locale la nivelul robotului. De asemenea pentru operarea out-door sistemul este prevăzut şi cu un modul GPS care oferă informaţie de poziţie pentru validarea odometriei.

1.4 Privire de ansamblu asupra structurii lucrării

Lucrarea de faţă este structurată în 8 capitole care introduc progresiv concepte şi detalii de proiectare şi implementare după cum urmează :

Capitolul 1. Stabileşte cadrul general de interes pentru problema abordată.

Capitolul 2. Descrie conceptele implicate în dezvoltare.

Capitolul 3. Descrie platforma robotică şi aplicaţia de control în timp real.

Capitolul 4. Descrie algoritmul SLAM implementat şi analizează performanţele.

Capitolul 5. Descrie arhitectura hardware şi software a modului de interfaţare cu sonarele şi implementarea nivelului adaptor pentru comunicarea cu aplicaţia de control în timp real.

Capitolul 6. Prezintă concluziile şi analiza rezultatelor obţinute.

Capitolul 7. Introduce bibliografia.

Capitolul 8. Anexe, cod sursă.

Capitolul 2

Descrierea formală a conceptelor implicate în dezvoltare: Maparea şi localizarea în mediu necunoscut

2. Descrierea formală a conceptelor implicate în dezvoltare: Maparea şi localizarea în mediu necunoscut

Capitolul curent al lucrării îşi propune să introducă acele aspecte teoretice, formale din spatele tehnicii de mapare a mediului pentru roboţi mobili şi a localizării. Un robot mobil care operează în modul de mapare explorând un mediu necunoscut generează hărţi prin acumularea de informaţie obţinută în timpul explorării de la senzori. O problemă care apare şi care a fost amintită şi anterior se referă la senzori. În acest context, din păcate, informaţia de la senzorii de distanţă (range sensors, ex.: sonar, laser, lidar, senzori optici) este utilă pentru mapare doar dacă poziţa senzorului la momentul citirii informaţiei este cunoscută. Encoderele sunt considerate cele mai uzuale unităţi senzoriale pentru roboţii mobili, utilizate pentru calculul de odometrie, dar în acelaşi timp sunt cele mai afectate de erorile sistematice (ex.: erori în calculul precis al dimensiunilor şi poziţionării roţilor reportat la şasiu) şi ne-sistematice (ex.: alunecarea roţilor funcţie de suprafaţă) care se acumulează în timp. Implicaţia directă a acestui aspect rezidă în faptul că în cazul în care robotul nu este localizat în mediu calitatea şi precizia hărţii va fi degradată. În continuare sunt prezentate acele aspecte formale implicate în problema mapării mediului şi se analizează problemele specifice şi elemetele care au fost prezente în analiza şi sinteza abordării propuse.

2.1 Tehnici probabilistice şi modelarea senzorilor ultrasonici pentru maparea mediului

În general toţi algoritmii dezvoltaţi pentru maparea mediului de către roboţi sunt bazaţi pe tehnici probabilistice. Aceşti algoritmi utilizează modelele probabilistice ale robotului şi a mediului său de operare şi se bazează pe inferenţe probabilistice pentru a transforma măsurătorile senzorilor în hărţi ale mediului.

Motivaţia utilizării tehnicilor porbabilistice rezidă în faptul că maparea mediului este caracterizată de incertitudine şi zgomot de măsură, care nu este o problemă trivială, ci una complexă, ce impune metode complexe de tratare. Algoritmii probabilistici abordează problema mapării prin modelarea explicită a diferitelor surse de zgomot şi efectul acestora în măsurătorile efectuate. Principiul de bază care stă în spatele majorităţii algoritmilor de mapare este regula lui Bayes :

$$p(x \mid d) = \eta p(d \mid x) p(x), \qquad (1)$$

Interpretarea acestei inferenţe probabilistice este redată în continuare. Se presupune că se porneşte de la determinarea informaţiei despre o cantitate (variabilă) x (ex.: harta mediului) bazându-ne pe informaţia de măsură, d (ex.: informaţia de la sonare). Regula lui Bayes oferă soluţia problemei prin multiplicarea termenilor p(d|x) si p(x), care redau probabiltatea de a observa măsurătoarea d în ipoteza x, respectiv presupunerea stării înaintea apariţiei oricărei informaţii de măsură. η este un factor de normalizare care asigură o distribuţie de probabilitate validă, [10].

În maparea mediului de către roboţi informaţia este disponibilă la anumite momente (perioada de eşantionare) şi putem astfel defini ca z (vectorul măsurătorilor) şi u (comenzile pentru actuatori). Se presupune că datele sunt preluate în timp, în alternanţă cu sinteza comenzilor pentru actuatori, după cum urmează,

$$z_1, u_1, z_2, u_2, ..., \qquad (2)$$

În particular, z_t este vectorul măsurătorilor la momentul de timp t şi u_t este comanda sintetizată pentru actuatori în intervalul [t-1, t). În maparea mediului abordarea dominantă pentru această integrare temporală a datelor este cunoascută sub denumirea de filtru Bayes, care are strânsă legătură cu filtrul Kalman, modelele Markov ascunse, reţelele Bayes dinamice şi procese de decizie Markov parţial observabile.

Filtrul Bayes are rolul de a extinde regula lui Bayes la probleme de estimare temporală, fiind un estimator recursiv pentru calculul unei secvenţe de distribuţii de probabilitate asupra unor cantităţi care nu sunt observate direct (ex.: harta mediului).
În continuare se introduce termenul de stare pentru a defini aceasta cantitate necunoscută, notată x_t, unde t este indexul temporal. Filtrul Bayes generic calculează probabilitatea (posterior) peste starea x_t prin următoarea ecuaţie recursivă :

$$p(x_t \mid z^t, u^t) = \eta p(z_t \mid x_t) \int p(x_t \mid u^t, x_{t-1}) p(x_t - 1 \mid z^{t-1}, u^{t-1}) dx_{t-1}, \qquad (3)$$

Variabilele cu index superior marcheaza datele până la momentul t, adică $z^t = \{z_1, z_2, z_3, ..., z_t\}, u^t = \{u_1, u_2, u_3, ..., u_t\}$. Filtrele Bayes sunt recursive şi se poate observa că probabilitatea $p(x_t \mid z^t, u^t)$ este calculată din aceeaşi probabilitate dar la un pas mai devreme. La limită, probabilitatea iniţială este $p(x_0 \mid z^0, u^0) = p(x_0)$. Putem constata astfel că timpul de înnoire a informaţiei este constant permiţând astfel filtrului Bayes să estimeze

informaţie continuu. Este important de menţionat că starea x_t trebuie să conţină toate cantităţile necunoscute care ar putea influenţa măsurătorile senzorilor la diferite momente în timp¸ în cazul nostru cele două cantităţi sunt harta şi poziţia robotului în mediu. Din moment ce ambele sunt influenţate de măsurătorile de la senzori de-a lungul timpului ambele trebuie estimate corelat. Putem astfel defini filtru Bayes pentru o hartă, m şi o poziţie, s a robotului :

$$p(s_t, m_t \mid z^t, u^t) = \eta p(z_t \mid s_t, m_t) \iint p(s_t, m_t \mid u_t, s_{t-1}, m_{t-1}) p(s_{t-1}, m_{t-1} \mid z^{t-1}, u^{t-1}) ds_{t-1} dm_{t-1}, \quad (4)$$

Majoritatea algoritmilor de mapare presupun că mediul este static şi astfel se poate omite indexul temporal pentru harta , m, deasemenea mai presupun şi că sinteza comenzii este independentă de hartă. Astfel, putem obţine o reprezentare simplificată a filtrului Bayes pentru maparea mediului, dată în continuare, [11],

$$p(s_t, m \mid z^t, u^t) = \eta p(z_t \mid s_t, m) \iint p(s_t \mid u_t, s_{t-1}) p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1}, \quad (5)$$

O observaţie bună, în contextul eficienţei mapării statice, se referă la faptul că acest estimator recursiv nu are nevoie de o integrare peste m hărţi, ceea ce redă simplificarea faţă de modelul iniţial, întrucât integrarea este dificilă peste un spaţiu multidimensional al tuturor hărţilor.

Operarea filtrului Bayes

O primă etapă în descrierea operării estimatorului este definirea a două probabilităţi generative $p(s_t \mid u^t, s_{t-1})$ şi $p(z_t \mid s_t, m)$ care sunt presupuse a fi invariante în timp şi care sunt reprezentate deseori sub forma $p(s \mid u, s')$, respectiv $p(z \mid s, m)$ reprezentând modelele generative ale robotului şi mediului. Probabilitatea $p(s \mid u, s')$, descrie efectul comenzii u asupra stării s, de fapt probabilitatea tranziţiei s' → s sub acţiunea comenzii u. Probabilitatea $p(z \mid s, m)$, este referită ca modelul perceptual, operarea senzorilor, descriind modul în care sunt generate măsuratorile z pentru diferite poziţii, s şi hărţi, m. Reprezentarea hărţii este condensată în reprezentarea teritoriului, care poate fi definit în detaliu în funcţie de algoritm.

În continuare sunt redate succint câteva aspecte privind performanţele şi funcţionalitatea metodei abordate în lucrare pentru maparea mediului la nivel formal al elementelor de analiza algoritmică. Abordarea aleasă se bazează pe matrici de ocupare, [10].

Primul aspect se referă la modul în care este reprezentată incertitudinea în hartă şi în acest caz se foloseşte o hartă auxiliară posterioară (relativ la momentele de timp considerate în reprezentare).

Al doilea aspect de interes referă convergenţa algoritmului care este puternică, minimul local nu este considerat, harta construindu-se incremental, fiind necesare informaţii de poziţie indiferent de zgomotul de măsură.

Al treilea aspect referă limitările dimensionale ale hărţii¸ în acest caz harta neavând limitări dimensionale, corespondenţa trăsăturilor e utilizată alături de date brute pentru a asigura o soluţie validă într-un mediu static sau un mediu dinamic limitat.

În continuare este realizată o descriere a elementelor specifice de modelare şi funcţionare pentru senzorii ultrasonici utilizaţi în aceasta abordare de mapare a mediului.

Senzorii ultrasonici (sonarele) sunt cel mai utilizat tip de senzori atât pentru domeniul comercial cât şi pentru mediul academic pentru roboţii cu operare indoor. Sonarul emite o undă la o frecvenţă ultrasonică şi determină timpul de la emisie până la revenirea undei la emiţător după reflexie. Ţinând cont de timpul de propagare a undei ultrasonice se poate determina distanţa la care se află obiectul faţă de sonar. Funcţional, un puls generat de traductorul ultrasonic determină o vibraţie a unei membrane care emite o undă considerată a avea o anvergură de 30^o. La emisie un timer este setat şi membrana devine staţionară. La recepţia unui ecou, de o amplitudine mai mare decât un prag, timerul este oprit şi timpul măsurat este timpul de propagare a undei de la emiţător la obiect şi retur.

Senzorii ultrasonici suferă însă şi de o serie de limtări. Prima problemă specifică a sonarelor este reflexia speculară, care se manifestă prin reflexii multiple în obiecte alăturate datorită unei diferenţe de unghi de reflexie, fapt ce determină creşterea timpului de propagare (time-of-flight) care implică şi detecţia unei distanţe eronate. O a doua problemă referă obţinerea unei distanţe mai mici la măsurătoare de la obiecte la senzor, care se manifestă în cazul în care la sonar ajunge o undă care se afla la extremitatea conului de activitate a undei şi nu fascicolul perpendicular pe suprafaţa de contact. Timpul de zbor va fi astfel mai scurt şi distanţa obţinută de sonar până la obiectul care reflectă va fi eronată.

Pentru a permite încorporarea informaţiei senzoriale de la diferite tipuri de senzori, în reprezentarea mediului de oprare prin hartă, se impune obţinerea unei reprezentări a senzorilor. Un model al senzorului este util pentru a realiza conversia măsurătorilor specifice senzorului într-o formă generică, care să ofere reprezentările probabilistice condiţionale pentru ficare locaţie din mediu, dată de o anumită citire a senzorului.

Robotul mobil utilizat în implementarea aplicaţiei de mapare prezintă 5 sonare dispuse într-o strctură de inel (ring) în partea frontală a robotului. Fiecare senzor ultrasonic este conectat într-un modul embedded cu MCU. În continuare sunt redate câteva aspecte privind modulele cu senzor ultrasonic. Senzorii ultrasonici utilizaţi în această aplicaţie sunt EZ0-LV-MaxSonar High Performance Sonar Range Finder de la MaxBotix. Fiecare modul cu sonar poate detecta obiecte aflate la distanţe între 0-inch şi 254-inch (0-6.45m) oferind o rezoluţie de 1-inch peste 6-inch până la 254-inch. Fiecare modul poate emite informaţia de distanţă în mai multe forme simultan. Astfel, modulul embedded conţine un MCU PIC 16F676 care preia semnalul de la senzorul ultrasonic şi dupa condiţionare cu un LM324 emite semnalul de tensiune corespunzător distanţei, sub forma unui semnal analogic proporţional cu distanţa ((Vcc/512)/inch) , un semnal PWM (147uS/inch) sau semnal codificat pe linia UART (0-Vcc, 9600-8N1). Citirile au loc la 50mS (20Hz frecvenţă) , iar senzorul operează la 42KHz realizând un control continuu al fascicolului şi suprimarea lobilor laterali. În continuare sunt redate caracteristicile conului de activitate a sonarului pentru diferite tensiuni de alimentare.

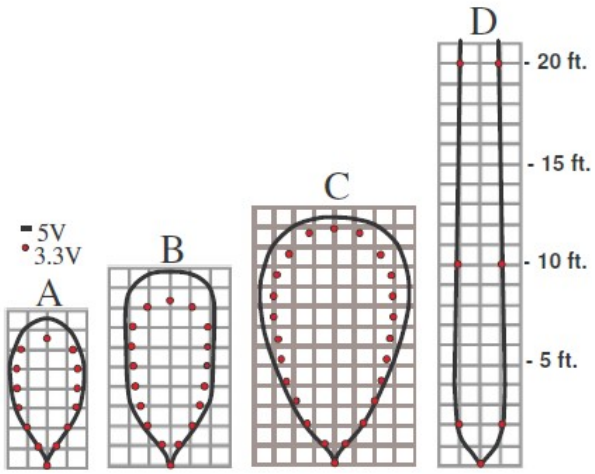


Figura 2 Caracteristicile conului de activitate a sonarului pentru diferite tensiuni de alimentare

Se poate observa că conul de activitate se poate modula funcţie de tensiunea de alimentare realizându-se apoi la o resetare de Vcc şi o recalculare a scarei de măsură pentru conversia din tensiune în distanţă. Schema modulului embedded cu senzor ultrasonic este redată în figura următoare.

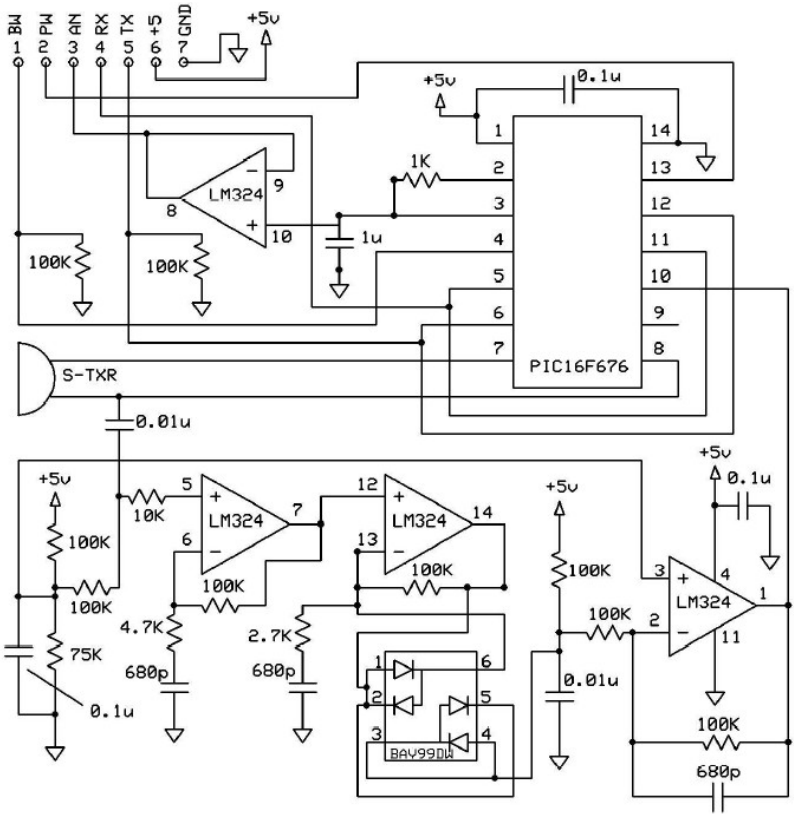


Figura 3 Schema modulului embedded cu senzor ultrasonic

La power-up fiecare modul are o perioadă de 250mS până când devine activ pentru acceptarea de comenzi de citire. Înainte de prima citire modulul realizează un ciclu de calibrare de 49mS şi apoi realizează citirile la fiecare 49mS. În continuare sunt redate aspecte privind modelarea senzorilor ultrasonici cu implicaţie directă în dezvoltarea algoritmului propus pentru maparea mediului.

Modelul sonarului a fost dezvoltat ca un model cu incertitudine şi s-au definit 4 regiuni pentru acesta (regiuni de probabilitate) care redau de fapt particularizarea funcţie de starea mediului. Astfel, primele 3 regiuni se află în conul de activitate a sonarului în timp ce a 4 a regiune este în afara ariei acoperite de sonar. Regiunile de operare ale sonarului sunt redate sintetic în următoarea figură în care R este distanţa maximă la care sonarul sesizează obiectul de reflexie şi Beta (β) este jumătate din unghiul de deschidere a conului sonarului.

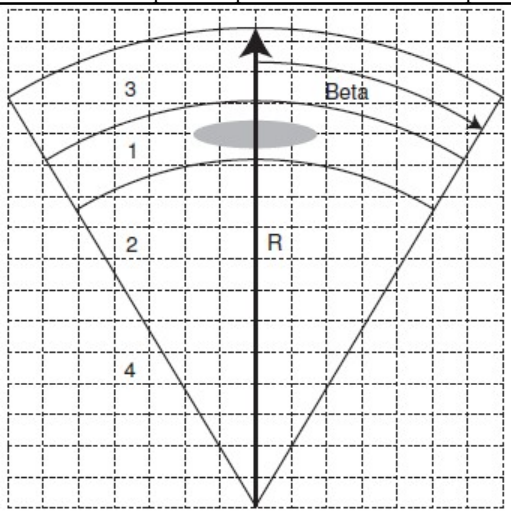


Figura 4 Regiunile de operare ale sonarului

- Regiunea 1 conţine acele elemente de mediu care sunt cel puţin parţial ocupate.
- Regiunea 2 conţine acele elemente de mediu care sunt probabil toate vide.
- Regiunea 3 conţine acele elemente din mediu cu stare incertă pentru că sunt aflate în spatele obiectelor care reflectă unda generată de sonar.
- Regiunea 4 conţine elemente despre care nu există informaţie din punctul de vedere al sistemului senzorial cu sonare.

Într-un scenariu tipic pentru o anumită citire de la sonar, elementele din regiunea 2 sunt mai probabil goale decât elementele din regiunea 1, citirile fiind mai corecte pentru elementele aproape de axa acustică faţă de elementele aflate la extremităţile conului sonarului. În continuare este introdus conceptul de matrice de ocupare pentru maparea mediului.

2.2 Matrici de ocupare

Matricile de ocupare sunt o abordare foarte utilizată în robotică. O matrice de ocupare este o reprezentare probabilistic finită a informaţiei spaţiale a robotului. Updateul matricii de ocupare se realizează cu informaţie (de corecţie) de la sonare şi astfel se creează permanent o reprezentare îmbunătăţită a mediului de operare a robotului. Un aspect interesant se referă la faptul că harta încorporează informaţie eterogenă de la diferite tipuri de senzori. Matricea finală va conţine reprezentarea regiunilor probabil ocupate, a regiunilor probabil goale şi a zonelor necunoscute. Această reprezentare poate fi utilizată şi pentru operare în mod de explorare, localizare, planificarea mişcării, identificarea landmark-urilor şi ocolirea obstacolelor.

Hărţile bazate pe matrici (grile) sunt formate dintr-un set de celule, a căror dimensiune este definită conform utilizării hărţii, de exemplu celule pătrate de latură l. În timpul construirii hărţii informaţia utilă este asociată cu fiecare celulă. În cadrul hărţilor bazate pe matrici de ocupare informaţia asociată cu fiecare celulă este corelată cu gradul de ocupare a regiunii din mediu corespunzătoare celulei, informaţia fiind obţinută din perspective diferite (de la senzori diferiţi), [12]. Nu se realizează nicio presupunere privind tipul trăsăturilor mediului. Abordarea prezentată în această lucrare va folosi matricile de ocupare pentru implementarea unei metode de mapare a mediului bazată pe sonare, care este o metodă promovată şi dezvoltată iniţial de Alberto Efes în anii 1990. Acesta a propus algoritmi de mapare şi navigaţie bazaţi pe informaţia de la sonare şi a utilizat matricile de ocupare pentru a obţine o reprezentare stochastică spaţială pentru percepţia activă a roboţilor mobili, [13].

În continuare sunt prezentate aspecte privind modul în care sunt utilizate matricile de ocupare (creare, update) pentru maparea mediului bazată pe sonare. Astfel pentru prima etapă, de creare a hărţii, se utilizează informaţia de la sonare. Această informaţie este interpretată utilizând profile de probabilitate pentru a determina zonele ocupate şi goale din mediu. Apoi aceasta informaţie este modelată de profilele de probabilitate care sunt proiectate într-o hartă bi-dimensională. Informaţia ce provine de la alte tipuri de senzori este integrată în harta generată cu ajutorul sonarelor. Important de menţionat este faptul că nu se utilizează o hartă cunoscută apriori, ci este construită incremental în contextul în care locaţia robotului este presupusă ca fiind cunoscută cu exactitate. În figura următoare este redat sintetic modul de funcţionare împreună cu procesarea specifică din spatele metodei.

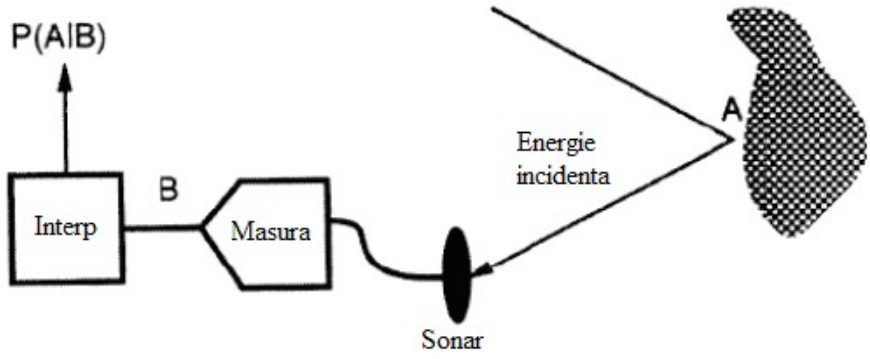


Figura 5 Modul de funcţionare împreună cu procesarea specifică din spatele metodei

După emiterea undei de o anumită frecvenţă de către sonar se recepţionează unda reflectată din obiect. Energia incidentă este preluată de membrana sonarului care determină o valoare de măsură (distanţă, timp) care prin interpolare oferă informaţie necesară pentru construirea

probabilităţii. Următoarea imagine ilustrează legătura între probabilităţile calculate şi operarea sonarului.

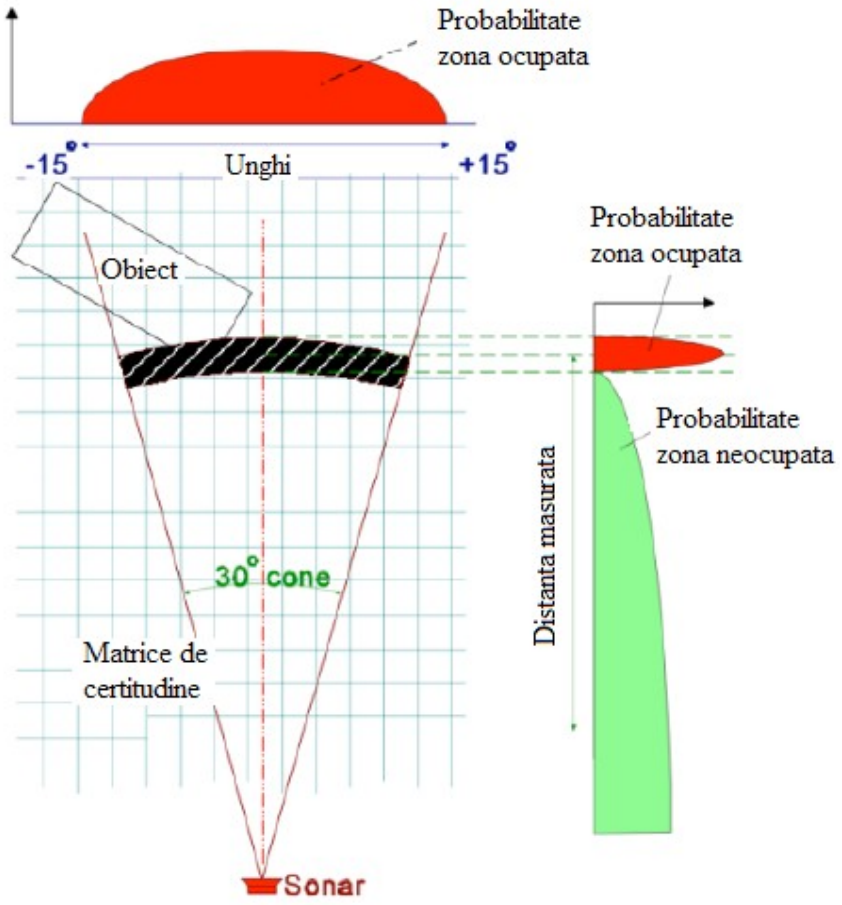


Figura 6 Legătura între probabilităţile calculate şi operarea sonarului

Se poate observa că obstacolul poate fi oriunde pe arcul delimitat la o anumită distanţă, în timp ce spaţiul mai mic decât distanţa respectivă este probabil neocupat. În matricea de ocupare finală fiecare celulă conţine statusul său de ocupare (ocupată, liberă, necunoscut) cu un factor de certitudine asociat, [11]. Harta finală va fi generată din două mulţimi separate de date derivate din probailităţile de distribuţie pentru celulele ocupate şi libere care sunt obţinute prin compararea puterii valorilor de liber/ocupat.

Compunerea informaţiei din mai multe citiri de la senzori presupune mai multe etape care sunt descrise în continuare, [10] :

Initializarea : harta este setată ca necunoscută

Suprapunerea zonelor libere : Se calculează probabilitatea pentru fiecare citire de la senzori,

$$p_{liber}(celula) = p_{liber}(celula) + p_{liber}(masura) - p_{liber}(celula) * p_{liber}(masura), \quad (6)$$

Suprapunerea zonelor ocupate : Se calculează probabilitatea pentru fiecare citire de la senzori şi se normalizează, $p_{ocupat}(masura') = p_{ocupat}(masura)(1 - p_{ocupat}(celula))$, (7) şi

$$p_{ocupat}(celula) = p_{ocupat}(celula) + p_{ocupat}(masura') - p_{ocupat}(celula) * p_{ocupat}(masura'), (8).$$

Calculul pragurilor : Se propagă două hărţi, cu celulele ocupate şi respectiv celulele libere, valoarea finală a unei celule fiind obţinută prin compararea puterii relative a valorilor pentru celula liberă sau ocupată. Citirile sunt proiectate pe o matrice bi-dimensională (grid).

În continuare sunt redate câteva elemente specifice procesării citirilor de la sonare ilustrate prin imagini edificatoare pentru cazurile care apar în realitate, înainte şi după terminarea execuţiei algoritmului (diferenţe consistente la ultima etapă de aplicare a pragurilor - thresholding).

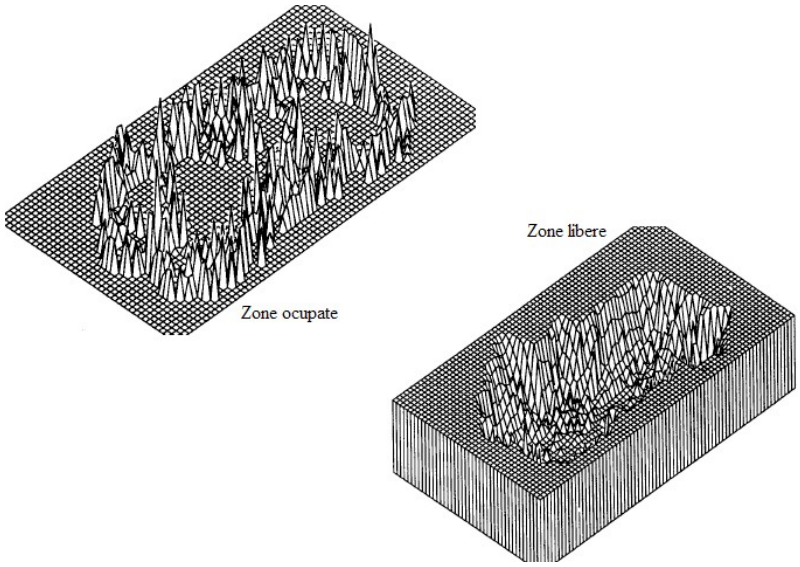


Figura 7 Densităţi de probabilitate înainte de execuţia algoritmului

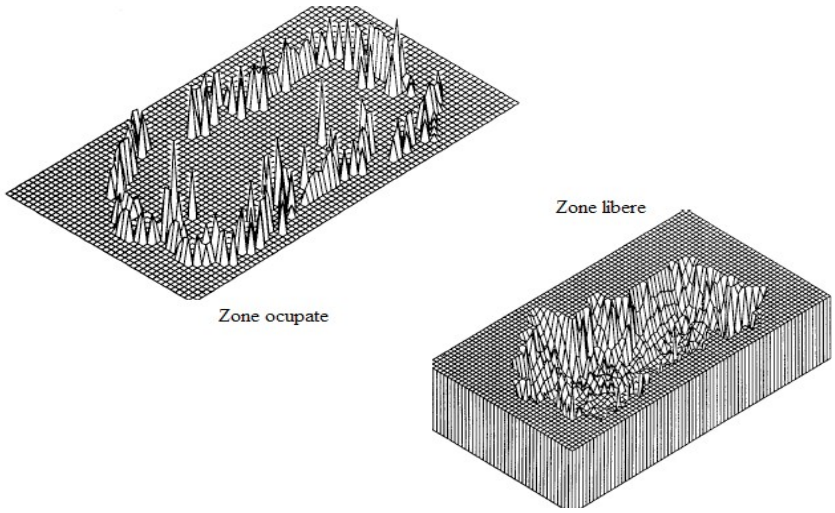


Figura 8 Densităţi de probabilitate după threasholding

Maparea cu matrici de ocupare este considerată o metodă extrem de robustă şi uşor de implementat, cel mai mare dezavantaj fiind lipsa unei metode pentru a trata incertitudinea din poziţia robotului. Un alt aspect se referă la influenţa zgomotului de măsură de la senzori care pot determina obţinerea de hărţi invalide, fapt care se poate realiza uşor prin filtrări şi desconsiderarea citirilor în timpul staţionării robotului, [13].

În continuare sunt redaţi paşii de particularizare a metodei generale pentru problema abordată în lucrare. Se vor surprinde aspecte precum updateul matricii de ocupare, fuziunea Bayes a senzorilor şi implicaţiile în localizare. După cum am menţionat anterior există multe metode de a converti informaţia de la senzori în valori numerice care să contribuie la updateul matricii de ocupare. Una din cele mai cunoscute metode, care a fost abordată şi în lucrarea de faţă, este fuziunea Bayes a senzorilor.

2.3 Fuziunea senzorilor pentru rafinarea matricii de ocupare

Fuziunea Bayes a senzorilor este o metodă care presupune conversia măsurătorilor de la senzori în probabilităţi condiţionate de forma P(s|H), probabilitatea ca o citire, s, a senzorului sa aibă loc, în contextul în care o celulă din matricea de ocupare este ocupată (ipoteza H). Modelul sonarului utilizat în această abordare porneşte de la modelul generic şi utilizează următoarele funcţii pentru a calcula probabilitatea pentru fiecare celulă a matricii care se află la o distanţă r şi un unghi α faţă de axa acustică a sonarului. Astfel, pentru fiecare regiune definită anterior se exprimă funcţiile de probabilitate, după cum urmează.

Pentru elementele din regiunea 1 :

$$P(ocupat) = \frac{(\frac{R-r}{R}) + (\frac{\beta - \alpha}{\beta})}{2} Max_{ocupat} \text{, (10)}$$

$$P(liber) = 1 - P(ocupat) \text{, (11)}$$

Pentru elementele din regiunea 2 :

$$P(ocupat) = 1 - P(liber) \text{ , (12)}$$

$$P(liber) = \frac{(\frac{R-r}{R}) + (\frac{\beta - \alpha}{\beta})}{2} \text{ , (13)}$$

Semnificaţia variabilelor este următoarea : R – distanţa maximă sesizabilă de către sonar, β - este jumătate din unghiul conului sonarului. Termenii $\frac{R-r}{R}$ şi $\frac{\beta - \alpha}{\beta}$ pun în evidenţă faptul

că elementele aflate mai aproape de axa acustică a sonarului sau mai aproape de originea pulsului sonarului au probabilităţi mai mari decât elementele de la extremităţi.

După cum am menţionat anterior aceste probabilităţi sunt proiectate în matricea de ocupare funcţie de poziţia şi orientarea sonarului şi apoi sunt combinate cu informaţia spaţială existentă a matricii utilizând fuziunea Bayes a senzorilor.

Modelul sonarului descrie lucrul cu probabilităţi condiţionate de forma P(s|H) însă probabilităţile P(H|s) (probabilitatea ca o celulă a matricii de ocupare să fie ocupată pentru o citire s a sonarului) sunt mai importante în problema abordată. Conform regulii lui Bayes se poate considera că :

$$P(H \mid s) = \frac{P(s \mid H)P(H)}{P(s \mid H)P(H) + P(s \mid \neg H)P(\neg H)}, \quad (14)$$

Utilizând apoi notaţiile din modelul senzorului definit anterior putem rescrie această relaţie după cum urmează.

$$P(ocupat \mid s) = \frac{P(s \mid ocupat)P(ocupat)}{P(s \mid ocupat)P(ocupat) + P(s \mid liber)P(liber)}, \quad (15)$$

În relaţia anterioară probabilităţile *P(ocupat)* şi *P(liber)* reprezintă ipoteza necondiţionată despre starea unei celule a matricii. În cazul în care există informaţie disponibilă despre mediul de operare se poate considera o valoare de iniţializare, de exemplu *P(ocupat)=P(liber)=0.5*. Pornind de la ecuaţia 14 şi considerând mai mule citiri aceasta devine

$$P(H \mid s_1, s_2, .., s_n) = \frac{P(s_1, s_2, .., s_n \mid H)P(H)}{P(s_1, s_2, .., s_n \mid H)P(H) + P(s_1, s_2, .., s_n \mid \neg H)P(\neg H)}, \quad (16).$$

Această reprezentare are însă o problemă în ceea ce priveşte monitorizarea tuturor schimbărilor de valoare pentru toţi senzorii¸ întrucât nu se cunoaşte numărul de update uri pentru fiecare celulă a matricii de ocupare. Soluţia este utilizarea relaţiei *P(H|s)P(s)=P(s|H)P(H)* şi astfel ecuaţia 16 devine

$$P(H \mid s_n) = \frac{P(s_n \mid H)P(H \mid s_{n-1})}{P(s_n \mid H)P(H \mid s_{n-1}) + P(s_n \mid \neg H)P(\neg H \mid s_{n-1})}, \quad (17)$$

Astfel, în noua reprezentare doar valoarea anterioară a probabilităţii pentru celula respectivă, $P(H \mid s_{n-1})$ va fi considerată şi va fi salvată pentru a asigura update-ul în iteraţiile următoare.

O reprezentare 3D a modelului sonarului pentru fuziunea Bayes a senzorilor este redată în continuare.

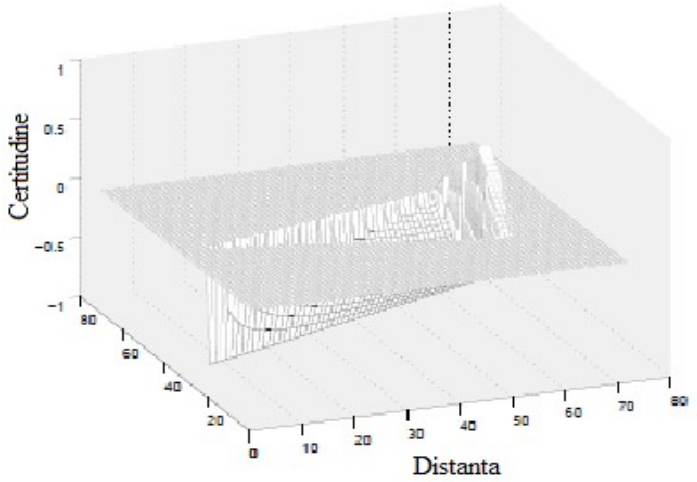


Figura 9 Reprezentare 3D a modelului sonarului pentru fuziunea Bayes a senzorilor

Se poate observa astfel că valorile certitudinii sunt mai mari aproape de poziţia sonarului şi mai aproape de axa acustică a sonarului. În continuare sunt redate aspecte privind localizare şi odometrie şi implicaţiile directe în maparea mediului.

2.4 Odometria şi localizarea continuă

Pentru a realiza majoritatea taskurilor un robot mobil trebuie să îşi cunoască poziţia în mediul de operare. Localizarea este procesul de update a poziţiei robotului utilizând informaţia de la sonare. Există mai multe tehnici de localizare, clasificate în 3 categorii, [12] :

- Localizarea bazată pe comportament : se bazează pe acţiunile robotului în mediu pentru a naviga (ex.: Follow-Right-Wall);
- Localizarea bazată pe trăsături : se bazează pe identificarea unor puncte din mediu, care au fost prestabilite, esenţa algoritmului constând în a extrage cât mai eficient trăsăturile mediului;
- Identificarea cu informaţie densă de la senzori : nu are nevoie de recunoaşterea obiectelor, utilizând orice informaţie senzorială pentru a o potrivi cu harta suprafeţei mediului (matricea de ocupare).

În continuare sunt redate aspecte privind odometria. Astfel, se consideră că majoritatea roboţilor mobili şi a vehiculelor autonome utilizează encoderele pentru a determina distanţa parcursă utilizând numărul de impulsuri obţinute prin rotaţia axului encoderului solidar cu roata robotului. O problemă majoră o reprezintă efectul erorilor asupra odometriei. Se

consideră ca există doua tipuri de erori, sistematice şi ne-sistematice. Erorile sistematice provin din măsurătoarea inexactă a dimensiunii roţilor sau plasarea nepotrivită a roţilor. Erorile ne-sistematice sunt generate de elemente externe structurii robotului, suprafaţa de contact, alunecările şi schimbările bruşte de direcţie determinând o acumulare a erorii.

Multe metode localizare încearcă să modeleze eroarea de odometrie sau să realizeze re-localizare doar după detecţia unei erori de odometrie majore. Pentru a elimina problemele de latenţă în algoritmi, a fost dezvoltată abordarea localizării continue (Schulz şi al.). Această tehnică utilizează faptul că erorile de poziţionare de la encodere se acumulează în timp. Localizarea continuă încearcă astfel să efectueze mici corecţii de poziţionare la intervale de timp mici în locul unor corecţii majore mai rare. La baza metodelor de localizare continuă se află crearea unor hărţi temporare ale mediului sub forma unor matrici de ocupare pe termen scurt care se bazează pe informaţie senzorială, [13]. Aceste hărţi pe termen scurt sunt construite la intervale scurte de timp şi astfel se poate considera că conţin un număr relativ mic de erori de odometrie. Procesul de localizare continuă poate fi sintetizat în următoare ilustraţie.

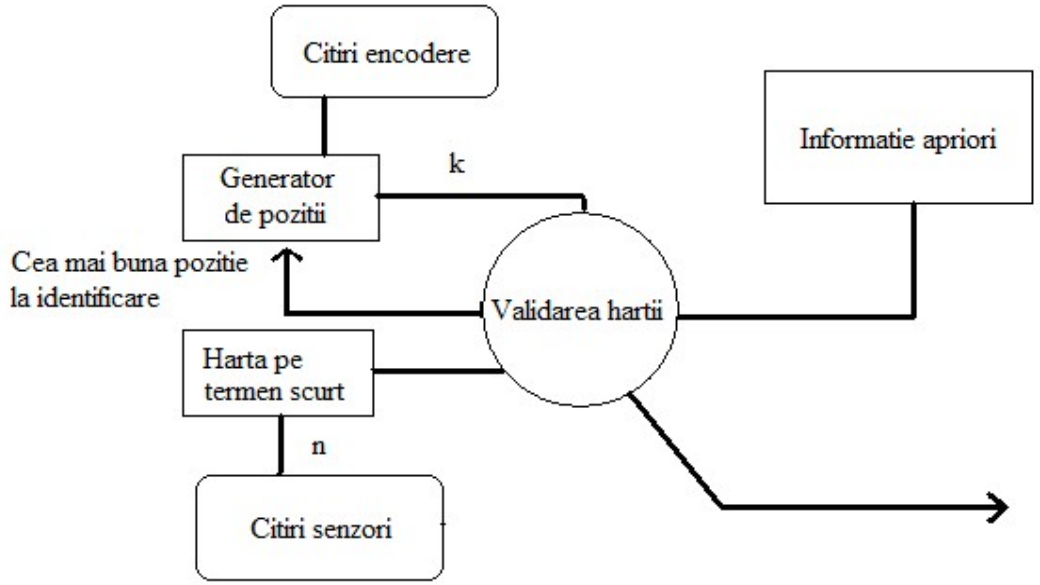


Figura 10 Metoda localizării continue

La fiecare localizare , n citiri ale senzorilor sunt fuzionate pentru a obţine harta temporară, pe termen scurt. Parametrul n defineşte o valoare de ponderare specifică echipamentului de achiziţie de date. În momentul în care o hartă parţială, pe termen scurt, conţine destulă informaţie despre mediu are loc validarea hărţii. În această etapă se compară harta pe termen scurt cu informaţia apriori disponibilă despre mediu, harta pe termen lung, astfel putem

consideră că performanţele şi calitatea acestui algoritm depind de completitudinea şi corectitudinea informaţiei apriori. Prin respectarea acestor condiţii se poate obţine o reducere a erorii de odometrie până la o valoare constantă, metoda aratându-şi eficenţa şi în cazul în care nu există informaţie spaţială iniţială.

Procesul de validare a hărţii este o etapă decisivă a algoritmului. Bazându-se pe informaţia din harta pe termen scurt aceasta poate determina suprapunerea multiplă cu harta pe termen lung. Pentru a limita posibilitatea apariţiei de identificări eronate se limitează căutarea la un număr de k poziţii probabile. Generatorul de poziţii are rolul de a implementa o funcţie ce generează un set de k poziţii posibile dată fiind poziţia posibilă a robotului şi navigaţie în mediu în durata de viaţă a unei hărţi pe termen scurt. Pentru fiecare din aceste k poziţii identificarea este realizată prin suprapunerea hărţii pe termen scurt centrată pe poziţia k peste harta pe termen lung. Identificarea este apoi apreciată printr-un scor, obţinut prin calculul sumei diferenţei între elementele comune celor două hărţi, după cum urmează

$$\sum | HG[i][j] - HTS[i][j] |, \quad (18)$$

unde HG – harta globală, pe termen lung şi HTS – harta pe termen scurt, parţială.
Poziţia robotului este updatată prin selecţia celui mai bun rezultat la identificare (matching) din poziţiile generate de modulul de generare a poziţiilor. Prin combinarea cu maparea mediului, localizarea continuă presupune o îmbinare completă a hărţii pe termen scurt cu cea pe termen lung pentru fiecare poziţie k.

În acest capitol au fost prezentate detalii privind conceptele implicate în dezvoltare, accentul cazând nu pe descriere formală ci pe descrierea problemelor specifice şi a modului în care acestea sunt rezolvate în abordarea propusă de lucrarea de faţă.

Capitolul 3

Descrierea platformei robotice şi a aplicaţiei de control distribuit ca suport pentru modulul de mapare a mediului

3. Descrierea platformei robotice şi a aplicaţiei de control distribuit ca suport pentru modulul de mapare a mediului

Sistemul robotic dezvoltat este o platformă mobilă cu două roţi motoare antrenate de 2 motoare de curent continuu, cu reductor şi o roată pasivă directoare de tip castor. Şasiul a fost construit din aluminiu pentru a reduce masa totală dar şi pentru a conferi robusteţe întregii structurii. În continuare sunt prezentate câteva imagini ale robotului mobil în diferite faze de construcţie.

3.1 Descrierea structurii hardware a robotului mobil

Prima fază : proiectarea CAD a şasiului şi caroseriei robotului mobil

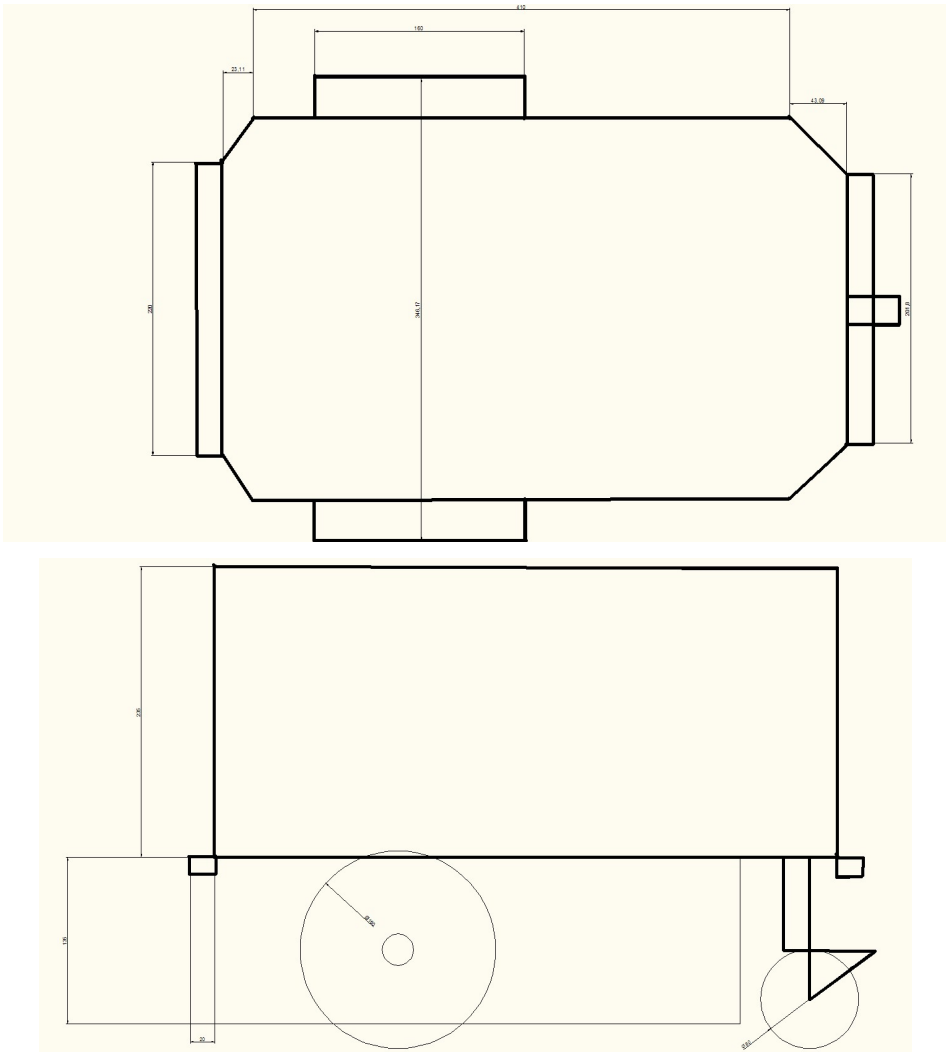
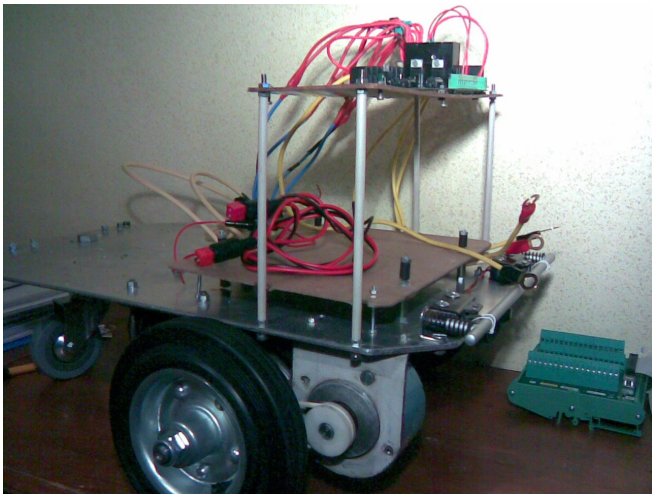


Figura 11 Proiectarea CAD a şasiului şi caroseriei robotului mobil

Faza a 2 a : şasiul robotului mobil şi subsistemul de locomoţie cu cele două motoare de curent continuu, encoderele şi bumperele



Faza a 3 a : subsistemul electronic, circuitele de electronică de putere şi unitatea de procesare

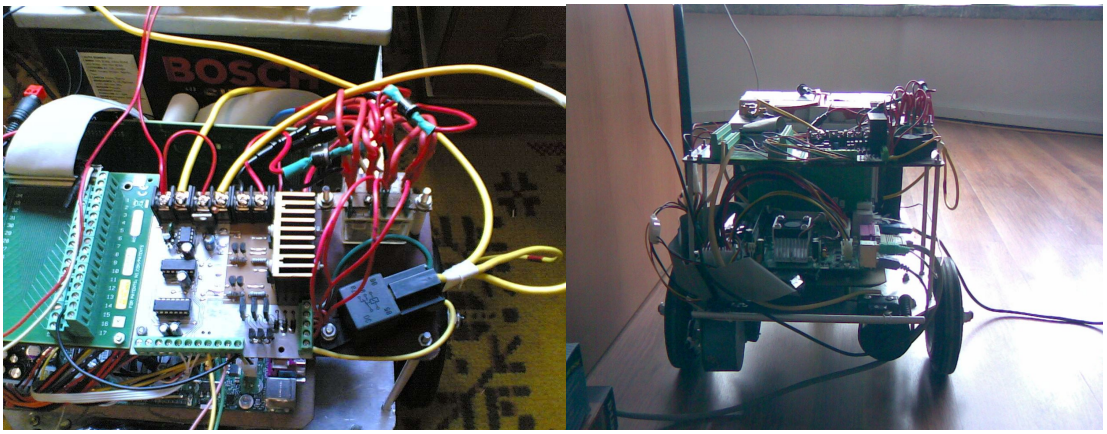
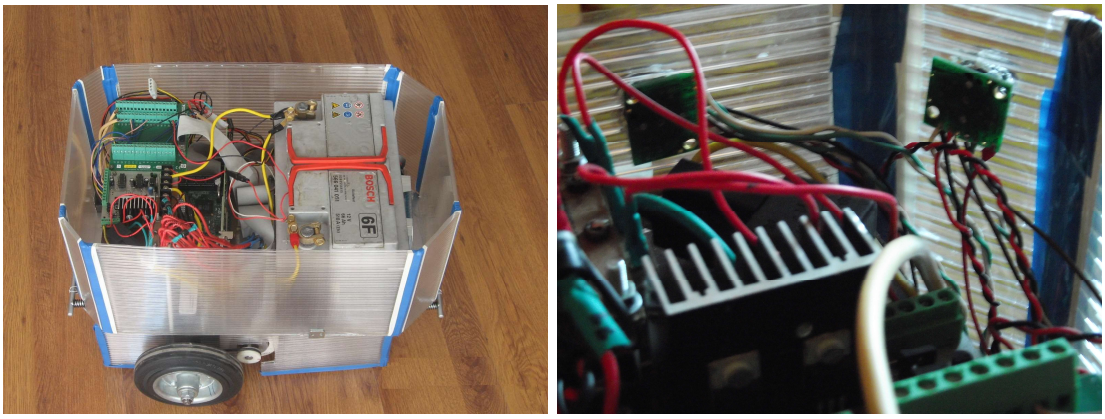


Figura 14 Etape intermediare în dezvoltarea platformei robotice

Faza a 5 a : structura finală



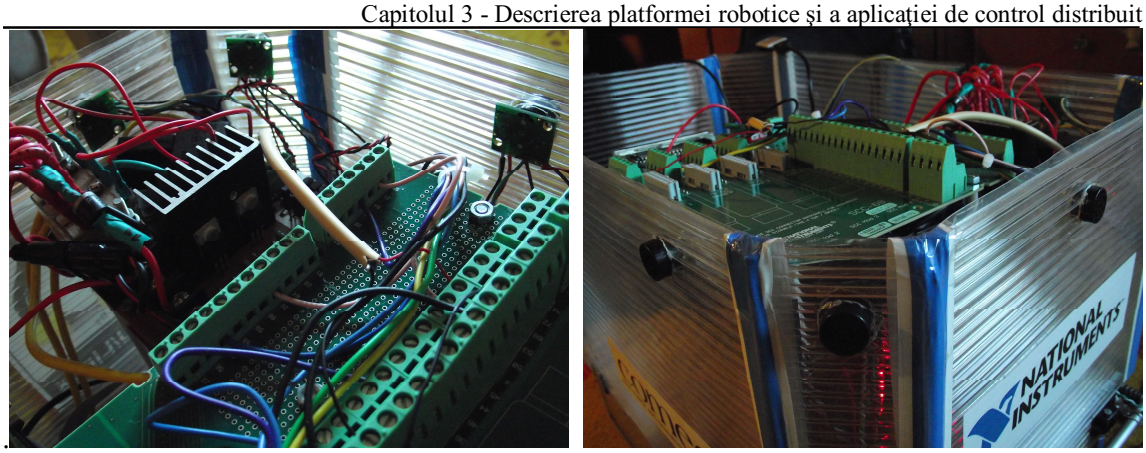


Figura 15 Robotul mobil

Motivul alegerii structurii de robot mobil diferenţial cu 2 roţi motoare şi una directoare provine din faptul că această structură de robot este simplă, o structură minimală uşor de proiectat şi implementat şi mai ales uşor de modelat, [15].

Nivelul actuatorilor

În continuare sunt prezentate câteva aspecte privind subsistemul de locomoţie a robotului. Astfel, în cazul locomoţiei cu ajutorul roţilor trebuie ţinut seama de anumite aspecte foarte importante, cum ar fi stabilitatea, manevrabilitatea şi controlabilitatea. Stabilitatea se referă la aspecte care privesc numărul şi geometria punctelor de contact cu suprafaţa de operare, localizarea centrului de greutate şi a centrului geometric, analiza stabilităţii în regim static şi dinamic, analiza gradului de înclinare, profilul ariei de contact, unghiului de contact şi tipul de frecare cu suprafaţa de operare a robotului, [16]. Primul nivel al sistemului de locomoţie îl reprezintă roţile. Cele două roţi motoare ale robotului sunt prevăzute cu jante din aluminiu şi suprafaţă de rulare de cauciuc pentru a diminua efectele alunecărilor. Diametrul roţilor este de 160 mm şi suportă sarcini de până la 140 de kg. Roata pasivă de tip castor are diametrul de 75mm şi poate suporta o sarcină de până la 55 de kg .

Figura 16 Tipurile de roţi utilizate la construcţia robotului

Următoarea componentă a subsistemului de locomoţie sunt cele două motoare de curent continuu cu reductor ce asigură un cuplu suficient pentru deplasarea robotului la viteze acceptabile, luând în considerare şi sarcina totală care se ridică la o valoare de 25 kg. Motoarele de curent continuu bi-direcţionale ale producătorului Bosch au tensiunea de alimentare de 12V, puterea nominală de 40W şi un curent maxim de 15A (în sarcina maximă). Viteza nominală este de 151 min^{-1} şi un cuplu continuu de 2,5Nm utilizând un raport de reducţie de 52:2, [17]. În figură sunt redate caracteristicile motoarelor.

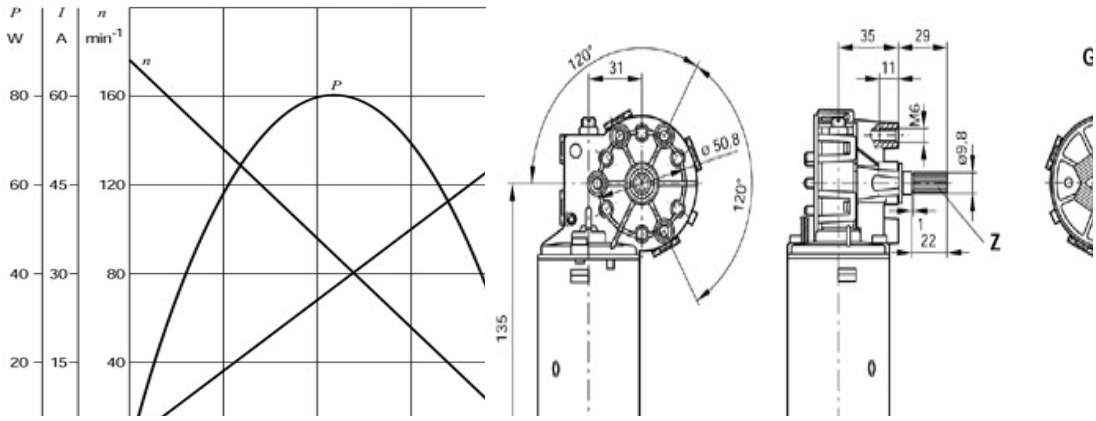


Figura 17 Caracteristicile motoarelor de curent continuu

Comanda motoarelor de curent continuu se realizează prin semnal PWM, generat de driverul de putere (punte H) care primeşte de la placa de achiziţie un semnal analogic (tensiune în intervalul 0-5V) şi generează la ieşire un semnal modulat cu factor de umplere variabil, [18].
Schema electronică şi conectica plăcii driverului de putere pentru comanda motoarelor de curent continuu.

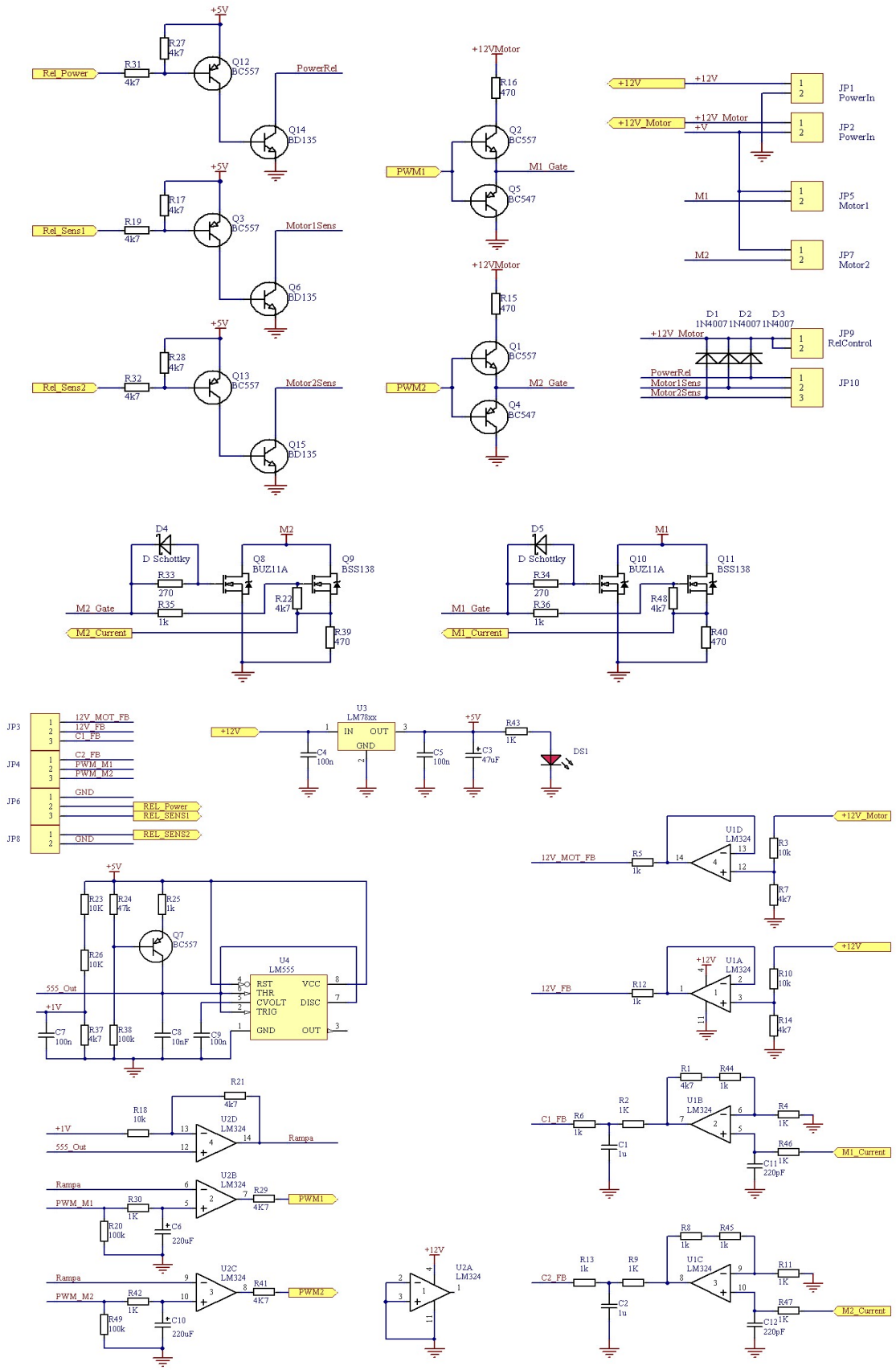


Figura 18 Schema driverului de putere pentru comanda motoarelor de curent continuu

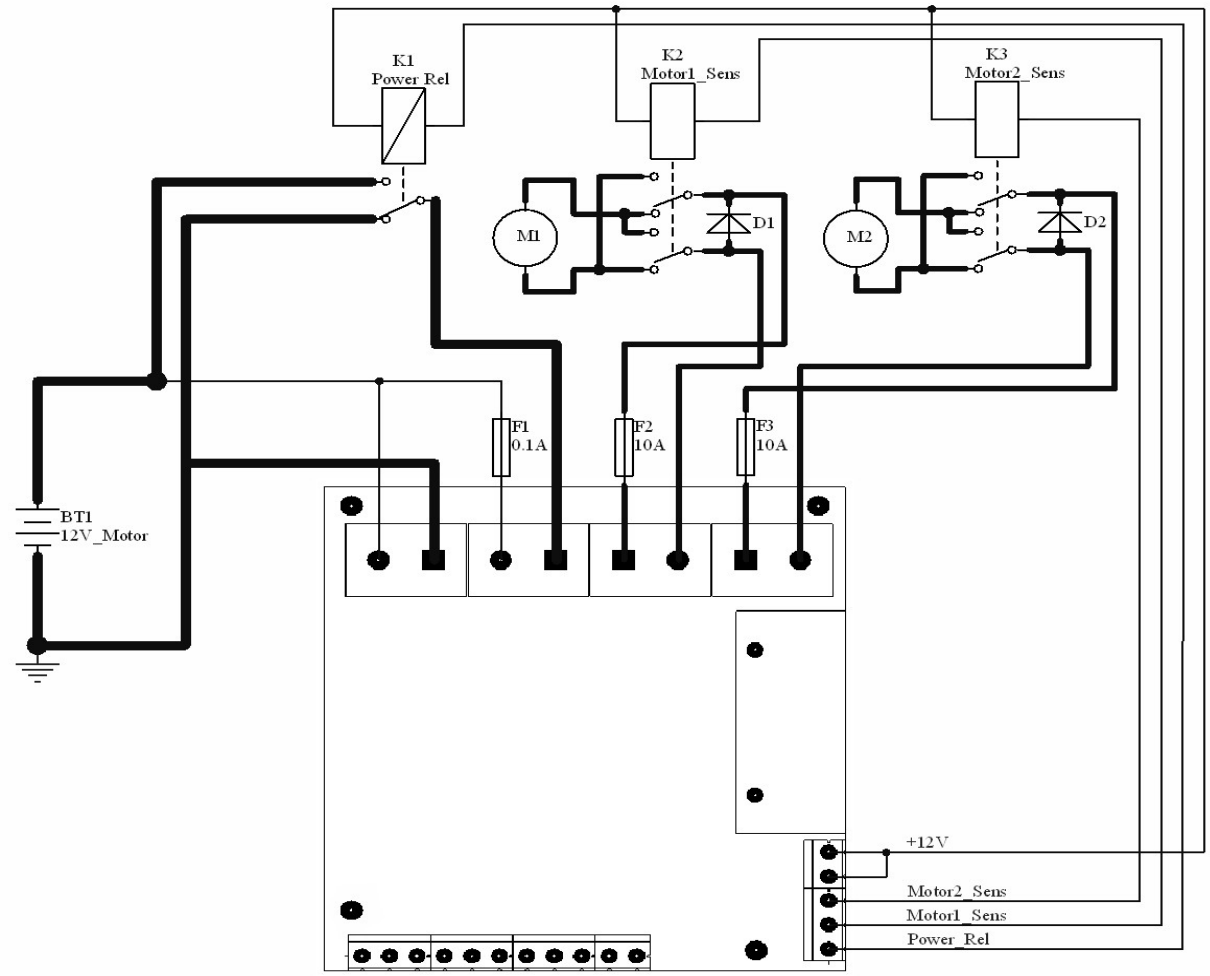


Figura 19 Conectica driverului de putere pentru comanda motoarelor de curent continuu

Nivelul senzorial

Percepţia în cazul roboţilor mobili este realizată de un sistem senzorial, format din mai mulţi senzori a căror informaţie este fuzionată şi utilizată la localizare şi control. Structura de robot mobil diferenţial dezvoltată este structura minimală care permite operarea robotului în regim de trajectory tracking (urmărirea unei curbe parametrizate cu restricţii de timp). Sistemul senzorial al robotului dezvoltat cuprinde două encodere, pentru poziţionare (odometrie) şi două bumpere. Bumperele sunt cel mai des întâlnit tip de senzori de contact, fiind reprezentate prin simple contacte care dau la ieşire o valoare binară corespunzătoare stării curente, determinând o acţiune de frânare de urgenţă la contact. Encoderele sunt senzori de stare internă utilizaţi pentru implementarea sistemului odometric, care returnează poziţia robotului, [16]. Encoderele incrementale Telemecanique (figura 10), dau la ieşire 500

impulsuri pe revoluţie şi utilizează o tensiune de alimentare de 5V semnalul de ieşire fiind direct preluat de intrarea în placa de achiziţie dotată cu rezistori de pull-up.

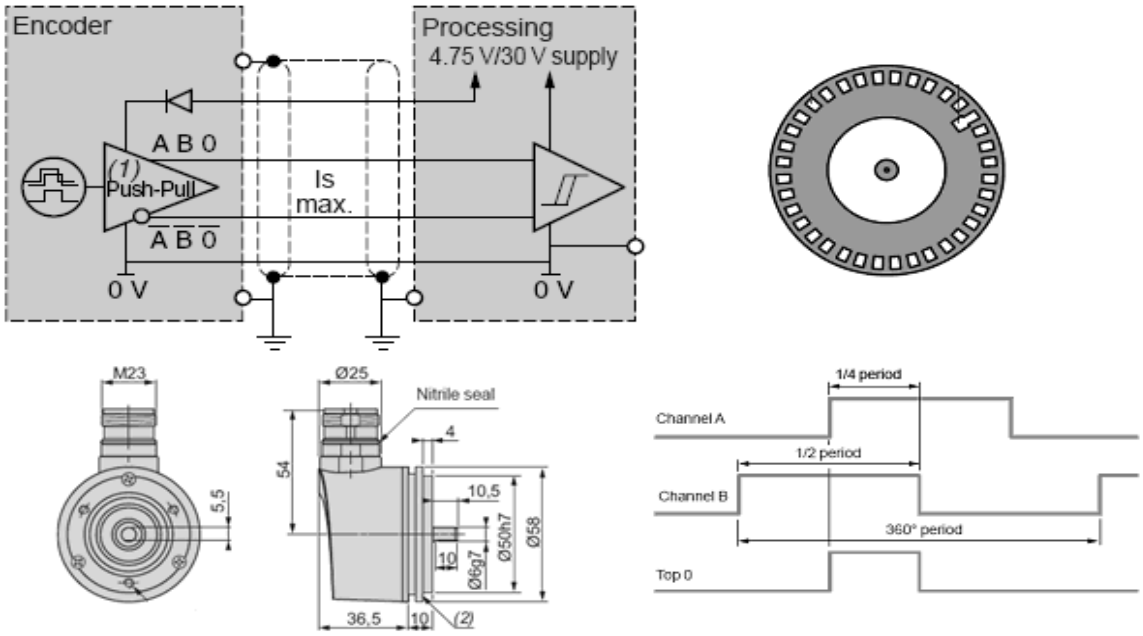


Figura 20 Prezentarea encoderelor şi modul de funcţionare

Robotul prezintă o platformă bazată pe un procesor Intel ATOM D525 Dual Core şi o placă de bază ASUS AT5NM10-I Mini ITX standard cu 1GB de memorie RAM şi capacitatea de stocare 160GB. Interfaţa cu hardwareul a aplicaţiei se realizează printr-o placă de achiziţie de date PCI NI 6024E care este o placă multifuncţională I/O. Din capacităţile sale de I/O amintim 16 intrări analogice single-ended (sau 8 diferenţiale), convertor ADC cu aproximări succesive de 12 biţi rata de eşantionare maximă garantată fiind de 220kS/s; două ieşiri analogice la o rezoluţie DAC (double-buffered, multiplicator) de 12 biţi şi o gamă de valori între +/- 10V pentru tensiunile de ieşire; 8 canale I/O digitale, programabile şi 2 timere/countere de uz general de 24 de biţi.

Nivelul de comunicaţie

Roboţii mobili trebuie să posede capacităţi de comunicare fie cu alţi roboţi din mediu fie cu un operator uman, pentru a raporta dacă o anumită sarcină dată a fost îndeplinită cu succes sau nu şi pentru a transmite informaţii de stare. Tehnologiile WiFi sunt deja răspândite în domeniul calculatoarelor personale şi oferă un suport complet de integrare al roboţilor în reţele de calculatoare, [17]. O problemă a acestor tehnologii ar fi faptul că sunt consumatoare

de energie, lucru care nu este prielnic robotului mobil. Eliminând problema energiei consumate, tehnologiile Bluetooth oferă servicii similare cu dispozitivele WiFi, însă distanţa maximă admisibilă între emiţător şi receptor este mult mai mică decât în cazurile celorlalte tehnologii. În momentul de faţă, tehnologiile Bluetooth se folosesc în interiorul clădirilor sau în laborator, unde distanţa maximă fără repetor este de maxim 20m. În cazul robotului dezvoltat am ales utilizarea unui modul de comunicaţie wireless pe portul USB care are încorporat un chipset (Ralink RT73) ce realizează preluarea pachetelor TCP/IP emise de aplicaţia de control utilizând facilităţile stivei implementate în sistemul de operare şi împachetează apoi informaţia (date şi comenzi) în pachete compatibile cu standardul 802.11b/g la o viteză maximă de 54Mbps pentru comunicarea cu clienţii aplicaţiei distribuite conectat la un router wireless.

Modelul cinematic al robotului

Modelul cinematic al robotului cuprinde analiza din punct de vedere geometric a robotului redând acele ecuaţii ce pun în evidenţă legătura între mărimile de control şi comportamentul robotului în spaţiu. Pentru a analiza cinematica robotului mobil trebuie studiate constrângerile impuse de roţile robotului. Astfel pe durata determinării modelului cinematic pentru un robot mobil se consideră robotul ca fiind un corp rigid cu roţi ce se deplasează în plan orizontal. Gradul maxim al modelului cinematic este astfel 3, două grade pentru poziţia în plan şi un grad pentru orientarea de-a lungul axei verticale care este ortogonală pe plan, [15]. Pentru a putea determina poziţia robotului mobil în plan trebuie să stabilim o relaţie între sistemul de referinţă global (sau inerţial) al planului şi sistemul de referinţă al robotului. Axele *Xi* şi *Yi* definesc sistemul de referinţă global, iar *{Xr, Yr}* este sistemul de referinţă al robotului care se alege într-un punct P de pe şasiu, de fapt fiind centrul de rotaţie al robotului. Poziţia punctului P în sistemul de referinţă global este dată de coordonatele *(x,y)* şi de diferenţa unghiulară dintre sistemul de coordonate global şi sistemul de coordonate a robotului, notat θ. Poziţia robotului se poate scrie ca un vector de trei elemente $\xi_I = [x\ y\ \theta]^T$ şi pentru a descrie mişcarea robotului este necesar să transformăm mişcarea din sistemul de referinţă global în mişcare de-a lungul sistemului de referinţă al robotului, [18]. În figură este redată reprezentarea robotului în plan pentru analiza cinematică.

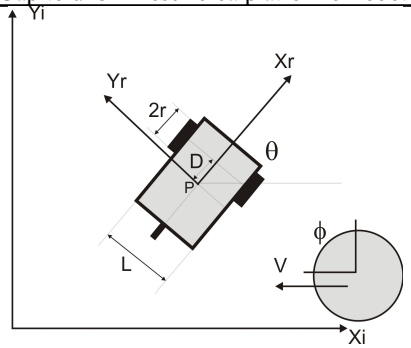


Figura 21 Reprezentarea robotului pentru analiza cinematică

Relaţia de transformare între cele două sisteme de referinţă se poate realiza utilizând ecuaţia prezentată în continuare,

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I = R(\theta)\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}, R = \begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}, \quad (19)$$

Cunoscând viteza robotului data de $\dot{\xi} = [\dot{x}\ \dot{y}\ \dot{\theta}]^T$, (20), care poate fi definită de viteza de rotaţie a roţii $\dot{\varphi}$, unghiul direcţiei β_i , viteza unghiulară $\dot{\beta}_i$ şi de parametrii geometrici ai robotului putem obţine descrierile pentru cinematica directă în ecuaţia următoare,

$$\dot{\xi} = \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = f(\dot{\varphi}_1,\ldots\dot{\varphi}_n,\beta_1,\ldots\beta_m,\dot{\beta}_1,\ldots\dot{\beta}_m) \quad , (21)$$

şi ecuaţiile pentru cinematica inversă sunt redate în ecuaţiile următoare,

$$\left[\dot{\varphi}_1 \ \cdots \ \dot{\varphi}_n \ \ \beta_1 \ \ldots \ \beta_m \ \ \dot{\beta}_1 \ \ldots \ \dot{\beta}_m\right]^T = f(\dot{x},\dot{y},\dot{\theta}), \quad (22)$$

Cunoscând raza unei roţi, *r*, lăţimea robotului (wheel base), θ orientarea (heading) şi vitezele de rotaţie ale roţilor $\dot{\varphi}_l$, $\dot{\varphi}_r$ se poate obţine modelul cinematic direct pentru robotul cu două roţi diferenţiale, descris de ecuaţia ce urmează,

$$\dot{\xi} = [\dot{x}\ \dot{y}\ \dot{\theta}]^T = f(L,r,\theta,\dot{\varphi}_l,\dot{\varphi}_r), \quad (23)$$

Putem determina mişcarea robotului în coordonatele sistemului de referinţă global funcţie de coordonatele sistemului de referinţă local, prin inversa matricii de transformare *R(θ)* . Se porneşte cu calculul mişcării fiecărei roţi în sistemul de coordonate local şi apoi translatăm ecuaţiile găsite în sistemul de coordonate global, [18]. Contribuţiile fiecărei roţi la viteza de translaţie a punctului P pe direcţia *Xr* sunt

$$\dot{x}_r = R\dot{\varphi}_r, \dot{x}l = R\dot{\varphi}_l, \quad (24)$$

iar viteza totală de translaţie a robotului este

$$v_R = \frac{v_l + v_r}{2}, \quad (25)$$

Contribuţiile fiecărei roţi la viteza unghiulară a robotului sunt descrise de ecuaţiile ce urmează,

$$\omega_l = 2r\dot{\varphi}_l / L, \omega_r = 2r\dot{\varphi}_r / L, \quad (26)$$

viteza unghiulară totală a robotului fiind dată de următoarele ecuaţii echivalente ,

$$\dot{\theta} = \omega_l + \omega_r, \omega_R = \frac{v_R - v_L}{L}, \quad (27)$$

În final modelul cinematic al robotului mobil cu locomoţie de tip diferenţial poate fi redat prin următorul set de ecuaţii echivalente,

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r\dot{\varphi}_r + r\dot{\varphi}_r \\ 0 \\ 2r\dot{\varphi}_l / L - 2r\dot{\varphi}_r / L \end{bmatrix}, sau \begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases}, \quad (28).$$

Cinematica directă ţine seama de modelul geometric al robotului şi de vitezele de rotaţie ale roţilor pe care le transformă în viteze globale. Înainte de trece la studiul cinematicii roţilor se impune prezentarea modificărilor aduse modelului cinematic generic întrucât în cazul robotului dezvoltat centrul de rotaţie se află translatat cu o distanţă D fată de poziţia clasică din model, [16]. Astfel în cazul de faţă robotul este descris de următorul model cinematic,

$$\begin{cases} \dot{x} = v_R\cos\theta - D\omega_R\sin\theta \\ \dot{y} = v_R\sin\theta + D\omega_R\cos\theta \\ \dot{\theta} = \omega \end{cases}, \quad (29).$$

Pentru implementare se discretizează modelul de mai sus obţinându-se următoarele ecuaţii, în care T_S este perioada de eşantionare,

$$\begin{cases} x(k+1) = x(k) + [r_r\omega_r(k)(\frac{\cos(\theta(k))}{2} - \frac{D\sin(\theta(k))}{L}) + r_l\omega_l(k)(\frac{\cos(\theta(k))}{2} + \frac{D\sin(\theta(k))}{L})]T_S, \\ y(k+1) = y(k) + [r_r\omega_r(k)(\frac{\sin(\theta(k))}{2} + \frac{D\cos(\theta(k))}{L}) + r_l\omega_l(k)(\frac{\sin(\theta(k))}{2} - \frac{D\cos(\theta(k))}{L})]T_S, \\ \theta(k+1) = \theta(k) + (\frac{r_r\omega_r(k) - r_l\omega_l(k)}{L})T_S \end{cases} ,(30)$$

Ecuaţiile modelului cinematic sunt utilizate în cadrul implementării de faţă în sistemul de odometrie, care se concretizează de fapt ca fiind calea de reacţie în bucla de control pentru poziţionare. În continuare sunt prezentate alte aspecte legate de cinematica robotului mobil şi studiul cinematicii roţilor. Astfel se fac următoarele presupuneri via-a-vis de mişcarea roţilor, mişcarea roţilor se va realiza numai în plan orizontal, contactul roţii cu solul va fi un punct, rotile nu sunt deformabile, mişcarea este de tip rostogolire pură, nu există alunecări şi nu există frecare în cazul rotirii în jurul punctului de contact, [18]. Roata pasivă a robotului este de tip castor şi este descrisă de ecuaţiile din figura următoare,

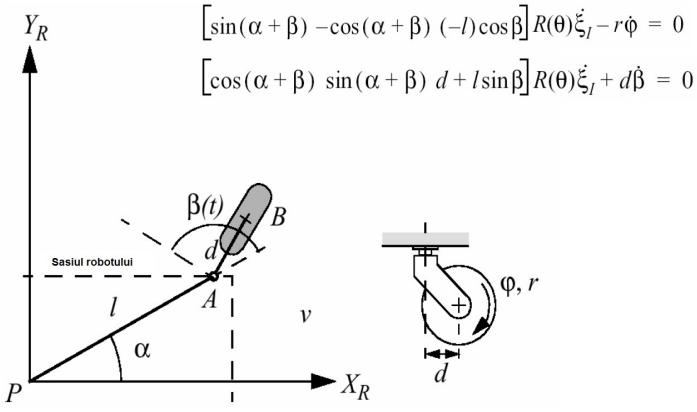


Figura 22 Ecuaţiile roţii pasive

Deşi un aspect important cinematica roţii pasive nu a fost considerată în implementarea de faţă doarece introducerea de noi termeni şi calcule suplimentare în modelul matematic ar fi determinat o creşterea a timpului de calcul şi astfel ar fi fost nevoie de o mărire a perioadei de eşantionare pentru a obţine rezultate valide pentru faza de control în timp real.

În cinematica roboţilor apar restricţii. Astfel fiecare roată fixă sau directoare standard poate impune restricţii în mişcarea robotului. Fiecare roată are o axa de rotaţie perpendiculară pe planul roţii. Roata trebuie să se mişte de-a lungul unui cerc cu raza R astfel încât centrul acestui cerc să fie axa roţii,. Centrul cercului este denumit centru instantaneu de rotaţie (ICR) şi astfel putem defini manevrabilitatea robotului mobil. Manevrabiliteta poate fi considerată o combinaţie între mobilitate (dată de restricţiile de alunecare) şi orientare (contribuţia direcţiei). În acest context limitările modelului cinematic al şasiului robotului redau

holonomia sistemului. Astfel un robot holonomic nu are restricţii în cinematică, poate merge în orice direcţie la orice moment de timp, iar un robot nonholonomic are restricţii cinematice impuse de roţile fixe şi de roţile directoare standard. Ultimul aspect legat de analiza modelului cinematic al robotului se referă la sinteza controllerului cinematic, care are rolul de a urmări traiectoria descrisă prin puncte (coordonate) şi / sau viteză în funcţie de timp, neţinând seama şi de mărimile dinamice ale sistemului, [18].

Această analiză preliminară a controlului robotului mobil a fost necesară pentru a stabili cadrul de interes în sinteza controllerului Sliding Mode pentru poziţionare utilizând modelul cinematic al robotului. Astfel, în cazul controlului mişcării în buclă deschisă putem considera de exemplu că traiectoria este împărţită în segmente diferite, linii, arce de cerc. Controllerul trebuie să calculeze o comandă astfel încât robotul să parcurgă o traiectorie lină. Apar însă probleme legate de fezabilitatea calculului traiectoriei, impunându-se limitări fizice în privinţa vitezelor şi acceleraţiilor robotului şi mai ales în cazul în care mediul se modifică dinamic.

3.2 Descrierea aplicaţiei de control în timp real

De interes major în dezvoltarea sistemului este aplicaţia de control tolerant la defecte în timp real. Astfel urmărind aspecte precum controlul distribuit, autonomia robotului, extensibilitatea structurii şi flexibilitatea implementării am dezvoltat o aplicaţie puternică care susţine la nivel inferior comunicarea facilă cu hardware-ul, implementarea taskurilor de control şi diagnoză în timp real la nivelul intermediar şi un nivel superior al comunicaţiei cu alte entităţi din mediul de operare, [19].

Orice aplicaţie în timp real este un sistem reactiv. Astfel ea se poate iniţia fără un set explicit de intrări şi apoi poate primi anumite intrări la care reacţionează prin anumite ieşiri. În continuare sunt prezentate câteva caracteristici importante ale unui sistem în timp real. Astfel după cum am menţionat sistemele în timp real au cerinţe dominate de constrângeri temporale, deci atributele cele mai importante fiiind determinismul la nivel logic (funcţional) dar şi la nivel temporal (dinamic) şi predictibilitatea. Un alt aspect se referă la complexitate, (sistemele în timp real presupunând un număr foarte mare de intrări şi ieşiri) şi la localizare, în genere sistemele în timp real având şi componente distribuite.

3.2.1 Nivelurile funcţionale ale aplicaţiei de conducere

În implementare caracteristicile de timp real au fost introduse la nivelul sistemului de operare ce rulează pe platforma de prelucrare a datelor de pe robot. Astfel utilizând o

platformă Intel ATOM D525 bazată pe un procesor x86 am instalat o distribuţie standard de Linux OS şi anume RedHat9 pe kernel 2.4.24 peste care am aplicat patchul RTAI (Real Time Application Interface), [22], pentru obţinerea de facilităţi hard real time. Motivaţia alegerii unui sistem de operare de tip UNIX rezidă în faptul că sistemul de operare este open-source şi astfel s-au minimizat costrurile implementării; în plus kernelul oferă fără nici o modificare caracteristici de timp real (soft real time) bune, însă pentru a permite creşterea performanţelor în contextul unor taskuri consumatoare de resurse şi minimizarea latenţelor s-a decis modificarea kernelului de bază pentru îmbunătăţirea caracteristicilor de timp real (hard real time) prin utilizarea RTAI. În continuare sunt prezentate acele trăsături de bază ale sistemului de operare în timp real care au fost urmărite la dezvoltare şi apoi se vor studia aspecte privind capacitatea şi mecanismele interioare ale kernelului Linux în ceea ce priveşte suportul pentru timp real, [19].

Sistemul de operare

Din punct de vedere funcţional un sistem de operare în timp real nu diferă de unul de uz general însă apar diferenţe la nivelul dimensiunilor (micro-kernel, nano-kernel), la modul de programare al taskurilor periodice cu sau fără mecanisme de IPC (Inter Process Communication), la planificarea strictă a taskurilor de timp real (introducerea unui sistem de priorităţi) şi nu în ultimul rând minimizarea latenţei prin introducerea preemptibilităţii. Sistemul de operare în timp real utilizat în cadrul proiectului este RTAI şi a fost dezvoltat de către DIAPM (Departamentul de Inginerie Aerospaţiale de la Politehnico Milano). Nucleul (kernelul) sistemului de operare are rolul de a media accesul la resursele hardware pentru aplicaţiile utilizator. Nucleul are două moduri (contexte) de execuţie, în mod utilizator sau în mod kernel, care sunt descrise în figura următoare,

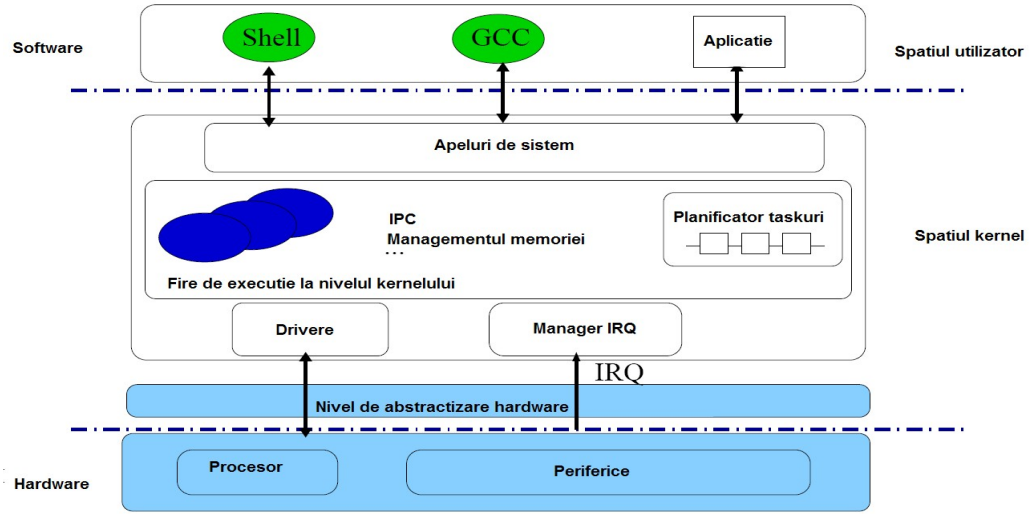


Figura 23 Arhitectura simplificată a sistemului de operare utilizat

Spaţiul utilizator este contextul de execuţie al aplicaţiei care poate accesa şi servicii ale nucleului prin apeluri de sistem şi apeluri de funcţii specifice oferite de interfaţa de programare pentru placa de achiziţie. Spaţiul kernel reprezintă contextul de execuţie al sistemului de operare, unde are loc managementul proceselor şi firelor de execuţie, planificarea şi managementul întreruperilor. Prin acest nivel se poate poate accesa nivelul hardware, însă important de menţionat este că există moduri privilegiate de acces pentru nivele, 22. Un aspect important pentru analiza facilităţilor de timp real oferite de sistemul de operare se referă la preemptibilitate şi latenţă. Preemptivitatea este capacitatea sistemului de operare de a întrerupe execuţia unui task în favoarea altuia cu o prioritate mai mare, fiind necesară pentru a satisface politica planificatorului şi de a putea asigura iniţierea unei schimbări de context.

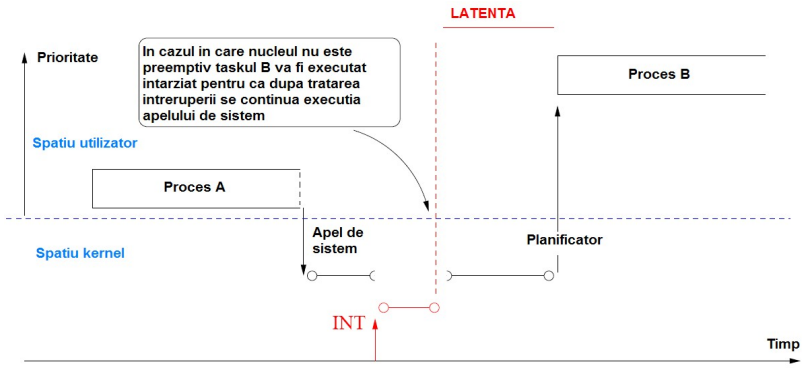


Figura 24 Scenarii de execuţie în nucleul sistemului de operare

Scenariile anterioare au avut rolul de a pune în evidenţă limitările sistemului de operare Linux de uz general în ceea ce priveşte performanţele de timp real, care se reflectă de fapt în optimizarea debitului de tratare a aplicaţiilor în detrimentul timpului de răspuns, utilizarea unui număr mare de puncte de preemptivitate care incetinesc sistemul de operare şi gestiunea ne-deterministică a memoriei şi perifericelor. Sistemul de operare în timp real încearcă să obţină o minimizare a latenţei globale a sistemului. Această latenţă globală este o sumă a căror termeni sunt descrişi în figura următoare,

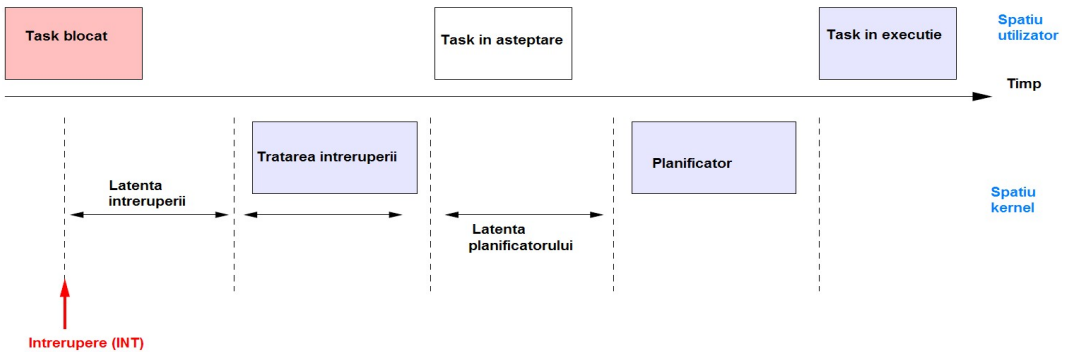


Figura 25 Latenţa globală a sistemului de operare

Pentru a face trecerea la un sistem de operare în timp real în contextul utilizării Linux există mai multe posibilităţi. Prima abordare se referă la modificarea nucleului, prin mărirea gradului de granularitate (gradul de preemptivitate a nucleului), prin reducerea numărul de secţiuni critice şi apeluri mai dese către planificator (dar mai judicios). A doua abordare care se referă şi la implementarea proprie are la bază conceptul de dual-kernel sau co-nucleu, care constă în adăugarea unui nucleu de timp real distribuţiei Linux existente fără a-i altera funcţionalitatea şi bazându-se pe mecanisme de tipul virtualizării întreruperilor şi unui mecanism specific pentru IPC între Linux şi micro-nucleul de timp real (domenii). În abordarea considerată micro-nucelul de timp real se inserează între Linux şi hardware, are un planificator separat şi nu depinde de secţiunile critice al Linux. La apariţia unei întreruperi micro-nucleul o capturează pentru a o utiliza în rutinele sale de timp real înainte de Linux, care va recepţiona doar o întrerupere virtuală, făcându-l astfel un domeniu de prioritate secundară. Micro-nucleul are timpi de comutare între contexte foarte mici, cu o latenţă sub 20 μs şi are acces la toată funcţionalitatea domeniului Linux, nemijlocit. În ceea ce priveşte mecanismul de la baza acestei abordări dual-kernel istoric vorbind au existat două implementări, RTHAL (RealTime Hardware Abstraction Layer) care este la baza primelor versiuni de RTAI şi ADEOS (Adaptive Domain Environment for Operating Systems), [20] care permite partajarea resurselor hardware între mai multe sisteme de operare concurente către care a migrat şi RTAI în 2003, [22]. La bază, ADEOS este un nivel de abstractizare al resurselor disponibil ca un patch peste nucleul de bază Linux, care permite existenţa mai multor sisteme de operare (domenii) pe aceeaşi maşină. Domeniile pot fi invizibile unul altuia însă toate sunt supervizate de ADEOS. Rolul minimal al unui domeniu este de a concura pentru a procesa evenimentele exterioare (întreruperi) sau cele interne (excepţii) în concordanţă cu prioritatea care i-a fost acordată. Pornind de la capacităţile sale de virtualizare ADEOS poate oferi o interfaţă de programare generică pentru domeniile supervizate care este independentă de arhitectura maşinii pe care operează. Structura de bază pe care se bazează ADEOS este lanţul de domenii client care concură pentru controlul evenimentelor, domeniile emiţând cereri de notificare pentru întreruperi externe sau virtuale, pentru apelurile de sistem emise de aplicaţiile Linux sau alte evenimente declanşate de codul executabil din nucleu. ADEOS asigură că evenimentele sunt distribuite domeniilor client în funcţie de prioritatea statică din sistem asigurând o livrare predictibilă şi bine sincronizată, [21]. Toate domeniile active sunt puse într-o coadă conform cu priorităţile lor formând un pipeline abstract. Astfel evenimentele care apar (incluzând întreruperile) sunt introduse în capul cozii (preluate de către cel mai prioritar domeniu) şi avansează către capătul cozii (domeniul cu prioritatea cea

mai scăzută). Pentru a realiza o distribuire a întreruperilor într-un mod prioritizat ADEOS implementează schema protecţiei optimiste a întreruperilor dezvoltată de Stodolsky, Chen şi Bershad. Astfel fiecare etaj al pipelineului ocupat de un anumit domeniu poate fi dezactivat, în sensul că următoarea întrerupere nu va mai fi livrată mangerului de întreruperi a domeniului şi nu se va mai realiza propagarea către domeniile de joasă prioritate. Întreruperile care apar între timp sunt acumulate în logul de întreruperi al domeniului şi vor fi tratate ulterior când etajul va fi activat printr-o operaţie de sincronizare. Prin această metodă domeniile îşi protejează secţiunile critice de preempţia cauzată de rutinele de tratare a întreruperilor. După ce un domeniu şi-a încheiat procesarea întreruperilor se emite un apel special către ADEOS care relizează o alocare a procesorului următorului domeniu ca prioritate din pipeline. Întreruperile nu sunt singurul tip de evenimente care pot parcurge pipelineul, întrucât însuşi kernelul Linux sau aplicaţiile care rulează pot genera astfel de evenimente, [21]. Ideea de bază este că evenimentele sunt notificări sincrone a unor capcane (traps), excepţii sau a altor acţiuni desfăsurate de către kernelul Linux. În continuare este prezentat specificul RTAI ca soluţie pentru dobândirea caracteristicilor de timp real pentru aplicaţia dezvoltată. RTAI este alcătuit din cinci părţi complementare. Prima componentă este nivelul de abstractizare al hardwareului (HAL) care oferă o interfăţă pentru accesul la hardware şi care redă suportul funcţional pentru Linux cu capacităţi hard real time. A doua componentă este nivelul de compatibilitate Linux care oferă o interfaţă către sistemul de operare Linux, şi astfel oferind RTAI posibilitatea de a fi integrat în mangementul taskurilor Linux, fără a influenţa operarea Linux, [23]. A treia componentă este nucleul de operare în timp real care introduce funcţionalitatea hard real time pentru planificarea taskurilor, tratarea întreruperilor şi securitate. A patra componentă este LX/RT care oferă suport pentru soft şi hard real time în spaţiul utilizator printr-o interfaţă de programare (API) pentru a oferi o funcţionalitate similară apelurilor de funcţii din spaţiul kernel şi din spaţiul utilizator şi un IPC simetric pentru cele două moduri. Ultima componentă a RTAI sunt pachetele de funcţionalitate extinsă , care cuprind drivere, interfeţe de programare pentru diverse dispozitive, watchdogs software, [27]. În această categorie intră şi o altă componentă importantă a aplicaţiei dezvoltate şi anume setul de drivere, Comedi, pentru I/O în timp real utilizând placa de achiziţie. În ceea ce priveşte managementul taskurilor şi planificarea (scheduling) RTAI oferă o întreagă varietate de taskuri de timp real şi planificatoare, oferind atât taskuri care ajung în cozile planificatorului sistemului de operare dar şi unităţi de execuţie care nu sunt planificabile (timere, tasklets, ASRuri).[23, 24] În ceea ce priveşte configuraţiile de planificare RTAI oferă alternative complementare, pentru planificare

uni/multi-procesor, pentru sisteme multiprocesor simetrice, cu planificare periodică şi one-shot, o planificare bazată pe priorităţi statice sau Round Robin pentru hard real time. În ceea ce priveşte metodele de comunicare inter-proces (IPC) RTAI oferă toată gama de primitive pentru sincronizare, semafoare, mutexuri, spinlockuri, variabile condiţionale şi flaguri. În ceea ce priveşte transferul, fluxul de date între procese RTAI implementează comunicarea prin mesaje, mailboxuri, cozi de mesaje POSIX, FIFOuri, memorie partajată şi remote procedure calls, iar în ceea ce priveşte managementul memoriei RTAI asigură o implementare simetrică şi un management dinamic al memoriei, [24]. Un aspect important este capacitatea de a permite integrarea de device drivere real time cum ar fi Comedi.

Utilizarea setului de drivere pentru operaţii I/O real time, Comedi

Comedi s-a dezvoltat ca un proiect open-source orientat pe dezvoltarea de drivere, instrumente şi librării care să ofere suportul pentru diferite plăci de achiziţie şi sisteme de achiziţie de date pentru efectuarea de operaţii I/O cu semnale analogice sau digitale, generare şi măsurare de frecvenţe, numărare impulsuri etc. Proiectul s-a dezvoltat sub forma unor module kernel şi a unei librării de funcţii pentru programarea în spaţiul utilizator, [25]. Astfel Comedi este colecţia de drivere pentru o mare varietate de plăci de achiziţie de date, driverele având un nucleu comun pentru funcţionalitate generică şi module individuale de nivel scăzut specific fiecărui dispozitiv. Comedilib este o librărie de funcţii destinată spaţiului utilizator, pentru programarea aplicaţiilor, configurare şi calibrarea dispozitivelor. Kcomedilib este un modul kernel care oferă aceeaşi interfaţă oferită de Comedilib în spaţiul utilizator, în spaţiul kernel, fiind indicată pentru taskuri real time. Un device driver este o componentă software care realizează interfaţarea cu anumit hardware, realizând conversia din funcţii de nivel superior emise de utilizator în comenzi dependente de dispozitiv. Plăcile de achiziţie de date suportate sub Comedi lucrează cu tipuri diferite de semnale: intrări analogice, ieşiri analogice, intrări digitale, ieşiri digitale, intrări pe counter/impuls, ieşiri pe timer/impulsuri. Lucrul cu semnale digitale nu necesită un efort prea mare sub Comedi, configurările necesare referindu-se la numărul canalului (uneori adresa pe bus) şi direcţia, intrare sau ieşire. Semnalele analogice sunt ceva mai greu de utilizat. Tipic un canal de achiziţie analogic poate fi programat pentru a genera sau citi o tensiune între două praguri superior şi inferior (în cazul de faţă -10V | +10V) şi utilizând suportul hardware al plăcii se poate seta eşantionarea unor anumite canale într-o anumită ordine şi de a memora datele pe placă, apoi se poate utiliza DMAul sau o rutină de tratare a întreruperilor pentru a realiza un dump al datelor într-

o anumită zona de memorie. În cazul semnalelor bazate pe impulsuri (semnale de la encodere, timere, countere) se adaugă, faţă de tratarea semnalelor tipic digitale, anumite specificaţii temporale semnalului. Comedi organizează hardwareul utilizând o ierarhie generică pornind de la canal (Channel), care este componenta hardware de la nivelul cel mai scăzut, reprezentând proprietăţile fizice ale unui singur canal de date (plaje de valori, tensiune de referinţă, polaritate, factor de conversie între mărimi fizice etc.); sub-dispozitivul (sub-device) caracterizează un set de canale cu funcţionalitate identică implementate fizic în acelaşi circuit (un set de 16 ieşiri analogice) menţinând informaţii despre numărul de canale şi tipul lor; dispozitivul (device) care este un set de sub-dispozitive implementate fizic pe aceeaşi placă. De exemplu placa utilizată în cadrul proiectului National Instruments PCI 6024E, are un sub-dispozitiv cu 16 canale de intrare analogice, un sub-dispozitiv distinct cu 2 ieşiri analogice şi un al treilea sub-dispozitiv cu 8 I/O digitale. Fiecare obiect dispozitiv încapsulează informaţii cu privire la codul de indentificare al producătorului, identificatorul dat de sistemul de operare, numărul de sub-dispozitive etc. În ceea ce priveşte funcţionalitatea pentru achiziţia de date oferită de Comedi aceasta se centrează pe lucrul cu canele sau seturi de canale. Un alt aspect care este surprins in interfaţa de programare a Comedi este managementul bufferelor şi caracterul event driven al achiziţiilor de date, totul sub posibilitatea implementării unor mecanisme de securitate pentru a asigura execuţia unor operaţii atomice asupra zonelor critice de cod sau de structuri de date. În continuare este prezentată arhitectura sistemului distribuit de control şi monitorizare al robotului.

3.2.2 Arhitectura distribuită client-server

Arhitectura sistemului software dezvoltat e formată din aplicaţia C++ care rulează pe unitatea de procesare embedded sub Linux-RTAI şi Comedi care oferă accesul direct la senzorii şi actuatorii robotului prin placa de achiziţie şi care prin intermediul unor threaduri server (de control şi de date) oferă o interfaţă de control aplicaţiei client Java sub Linux permiţându-i să activeze execuţia fazelor şi să primească informaţii direct de la senzorii robotului. Aplicaţia Java are rolul de a asigura activarea sau dezactivarea fazelor de execuţie ale robotului prin utilizarea unui instrument care-i permite să determine o execuţie secvenţială sau paralelă a taskurilor robotului, numit Grafcet, [26]. Structura se poate extinde cu un client Java cu drepturi restrânse cu rolul doar de a monitoriza execuţia şi de a primi informaţii despre starea sistemului, însă la momentul actual nu a fost implementat efectiv. Arhitectura concretă a aplicaţiei este descrisă în figura următoare,

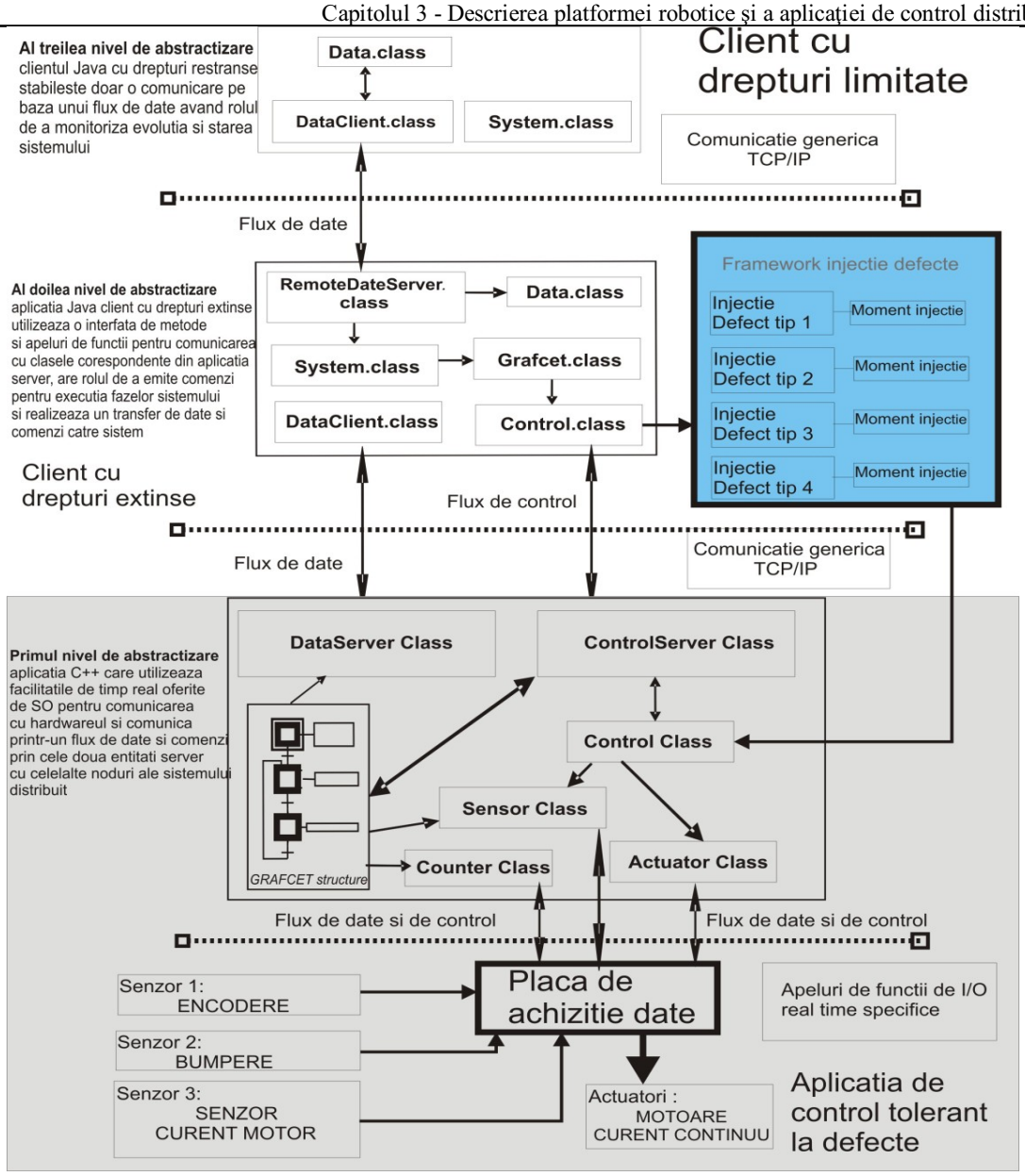


Figura 26 Arhitectura generală a aplicaţiei dezvoltate

Urmând o vedere de ansamblu aplicaţia dezvoltată se bazează pe un tipar de dezvoltare orientat obiect care conferă flexibilitate şi extensibilitate. În continuare sunt descrise nivelele individuale ale aplicaţiei.

Taskul de control al robotului în regim de trajectory tracking se realizează la o perioadă de eşantionare de 200ms. Perioada de eşantionare a fost aleasă pentru a permite obţinerea unor rezultate valide în ceea ce priveşte sistemul odometric (date valide şi consistente). Taskul de control se bazează pe sinteza unei structuri de conducere în cascadă cu o buclă internă pentru controlul turaţiei motoarelor de curent continuu, [29, 31], care se execută la o perioadă de eşantionare de 50 ms (reacţia fiind dată de sistemul de odometrie) şi

o buclă externă pentru controlul poziţionării robotului cu un controler sliding mode ce sintetizează o lege de comandă la fiecare 200ms (reacţia fiind dată prin sistemul odometric). Structura de conducere în cascadă a robotului în regim de trajectory tracking este prezentată în figura următoare împreuna cu modulul de monitorizare şi detecţie a defectelor.

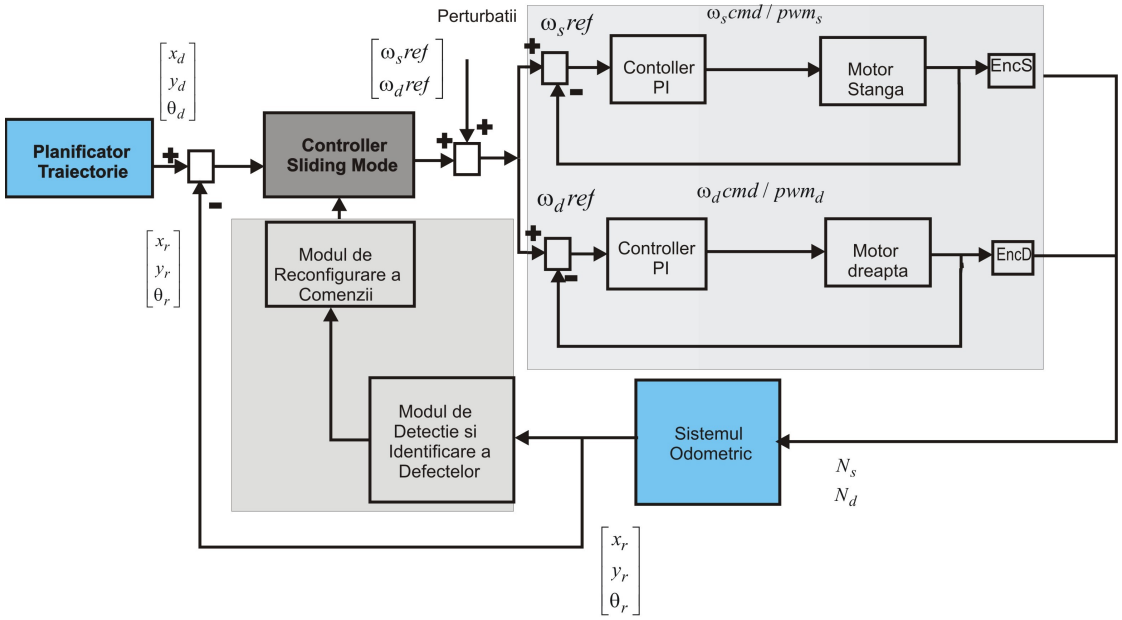


Figura 27 Structura de conducere în cascadă a robotului

Buclele de reglare interne realizează controlul turaţiei motoarelor de curent continuu ale robotului. Referinţa pentru buclele celor două motoare este dată de controllerul sliding mode sub forma unei viteze unghiulare . Controllerul sliding mode oferă la ieşire o comandă sub forma unei viteze unghiulare şi a unei viteze liniare pentru robot, care sunt însă utilizate la calculul vectorului de referinţe pentru motoare (prin transformare cinematică inversă), [28, 30]. Al doilea task este taskul de monitorizare şi detecţie a defectelor. În continuare este ilustrată structura modulului de detecţie, identificare a defectelor şi reconfigurarea controlului bazat pe un banc de 5 filtre EKF, [33].

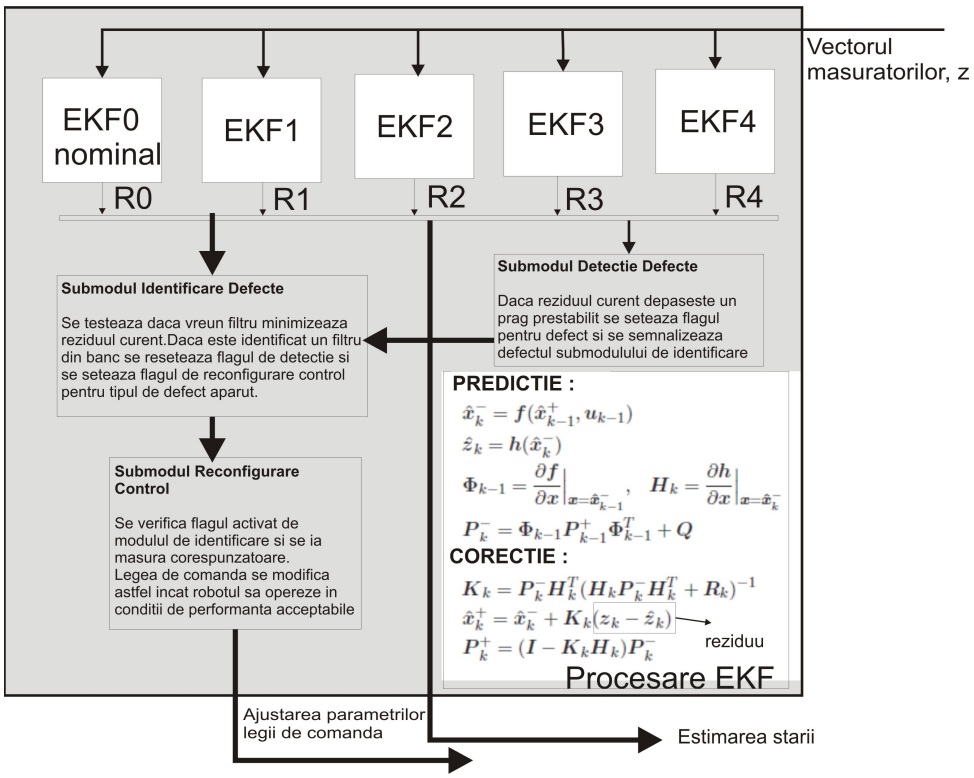


Figura 28 Modulul de detecţie, identificare şi reconfigurare

Fiecare EKF din banc înglobează în structura sa un model cinematic al robotului, dar cu parametrii diferiţi. Ideea de bază este că pentru acelaşi vector de intrare *z*, afectat de zgomot, fiecare filtru realizează o predicţie a stării robotului. Fiecare filtru este asociat unui anumit tip de defect, [32, 34].

În cele ce urmează este prezentată arhitectura internă a aplicaţiei server ce rulează pe robot.

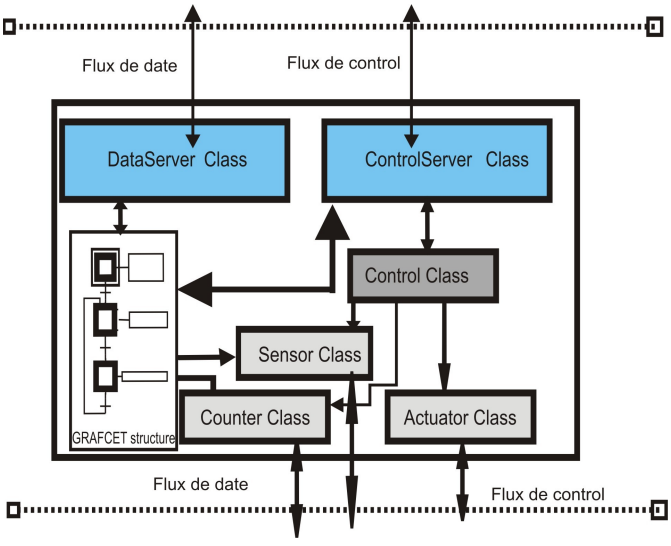


Figura 29 Arhitectura internă a aplicaţiei ce rulează pe robot

În continuare este redată arhitectura clientului Java accentul căzând pe clasele care asigură comunicaţia cu robotul.

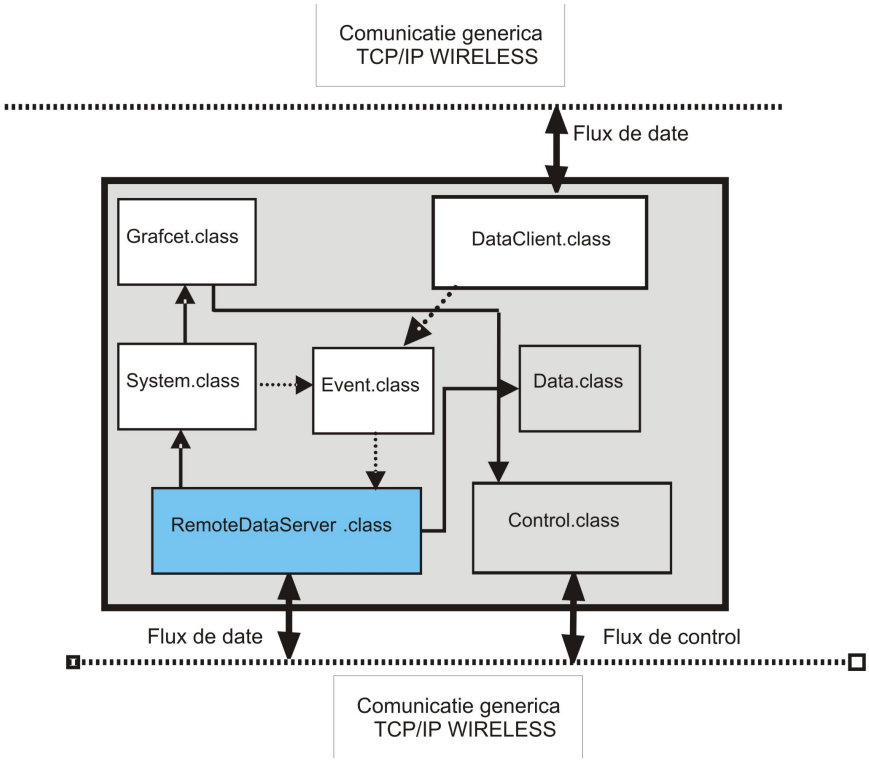


Figura 30 Arhitectura aplicaţiei client Java

Aplicaţia client Java are o structură asemenătoare cu a aplicaţiei de pe robot pentru a asigura compatibilitate la nivel funcţional. Astfel făcând referire la nivelul de comunicaţie, au fost implementate care asigură comunicarea printr-un flux de date cu aplicaţia ce rulează pe robot şi are rolul de a prelua informaţiile de stare şi parametrii curenţi ale mărimilor de interes pentru robot şi de a le transmite local pentru logging.

În capitolul de faţă a fost prezentată o descriere funcţională a aplicaţiei dezvoltate la nivelul claselor. S-au prezentat aspecte privind structura ierarhică şi dependenţele între clasele specifice fiecărui nivel. Pentru mai multe detalii şi un studiu direct al implementării se poate consulta listingul aplicaţiei din anexă.

Capitolul 4

Sinteza modulului de mapare a mediului

4. Sinteza modulului de mapare a mediului

În capitolul curent sunt redate aspecte generale de implementare pentru algoritmul SLAM utilizat în aplicaţia dezvoltată. Algoritmii SLAM (Simultaneous Localization And Mapping) sunt tehnici prin care robotul mobil are posibilitatea de a construi o reprezentare a mediului de operare şi în acelaşi timp să utilizeze această hartă pentru a-şi determina poziţia, [37]. Deşi în ultima perioadă dezvoltarea în domeniul SLAM s-a axat pe îmbunătăţirea eficienţei computaţionale a algoritmilor, au apărut noi probleme cum ar fi neliniarităţile, asocierea datelor şi caractrerizarea punctelor landmark care expun provocări în implementările practice SLAM.

4.1 Implementarea offline SLAM

În continuare sunt redate câteva aspecte privind implementarea SLAM utilizând formularea (regula) Bayes, prin care se doreşte obţinerea unei distribuţii de probabilitate / estimare a locaţiilor punctelor de marcaj (landmark) şi a poziţiei robotului, [35].

Algoritmul SLAM generic creşte în complexitate odată cu creşterea numărului de puncte de marcaj (landmarks) fapt care în implementările real-time devine o limitare majoră. Pentru reducerea acestei complexităţi au fost dezvoltate mai multe abordări. Din aceste abordări se impune să amintim, augmentarea spaţiului de stare, refacerea reprezentării informaţiei, updateuri partiţionate ale valorilor şi metode de sub-mapare. Algoritmii convenţionali SLAM au soluţii redundante fapt care se regăseşte în operarea în mediu dinamic şi în mediu static, [36].

O a doua problemă a SLAM este asocierea datelor, în ceea ce priveşte structura mediului şi problemele specifice care apar atunci când robotul revine la zone deja mapate după o lungă etapă de explorare (problema închiderii buclei – loop closing).

Marele avantaj al SLAM constă în faptul că nu are nevoie de informaţie apriori despre mediu pentru construirea hărţii, toată informaţia fiind acumulată în timp. În abordarea propusă pentru implementarea SLAM, se vor utiliza concepte prezentate anterior, cu privire la modelul senzorilor, matricile de ocupare, fuziunea Bayes şi localizarea continuă. Astfel, deşi informaţia de la encodere este afectată de erori în calculul odometriei, localizarea continuă va compensa inconvenientul şi va permite obţinerea hărţii, [35].

4.1.1 Proiectarea modulului offline SLAM

În continuare este redată arhitectura funcţională a aplicaţiei ce implementează algoritmului SLAM offline pentru maparea mediului pornind de la arhitectura prezentată pentru localizarea continuă. Abordarea este considerată offline întrucât informaţia este preluată online în timpul operării robotului, însă maparea are loc offline, după ce informaţia a fost achiziţiontă, [10]. După cum am menţionat anterior această metodă va îmbina tehnicile pentru modelul senzorilor, matricile de ocupare şi localizarea continuă fiecare fiind parametrizată şi configurată corespunzător pentru obţinerea unui grad de performanţă ridicat.

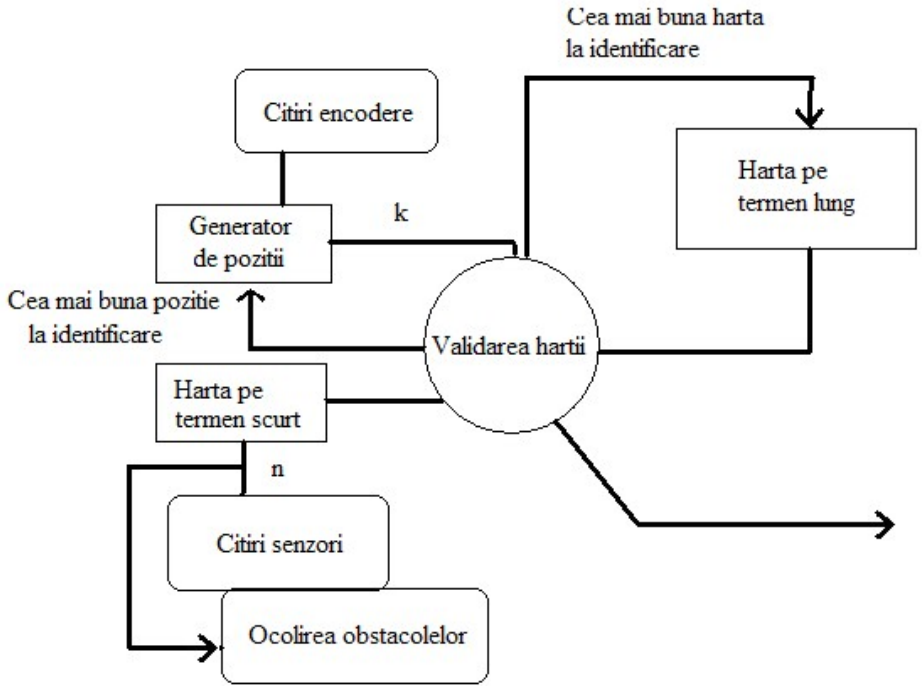


Figura 32 Metoda SLAM sintetizată

Întrucât nu este definită o metodă analitică de parametrizare configurarea s-a realizat empiric alegând parametrii potriviţi pentru aplicaţie şi hardwareul utilizat.

Aplicaţia SLAM dezvoltată are mai multe submodule funcţionale interne. Primul submodul are rolul de a implementa achiziţia de date şi logica de navigaţie. Astfel, robotul va naviga prin mediu şi în acelaşi timp va achiziţiona informaţie de la senzori şi va calcula informaţia de odometrie. Poziţia curentă a robotului şi citirile de la sonare sunt preluate la fiecare 50mS în interiorul aplicaţiei de control în timp real. Al doilea submodul are rolul de a implementa maparea şi localizarea. Astfel datele colectate de primul modul vor fi utilizate

pentru crearea hărţii mediului (harta pe termen lung) care este în acelaşi timp utilizată şi de localizarea continuă pentru a calcula şi corecta poziţia reală a robotului.

Pentru a obţine o bună viteză de execuţie şi posibilitatea determinării parametrilor buni pentru configurare s-a ales separarea în cele două submodule, de achiziţie a datelor şi localizare. În acest context robotul nu trebuie să repete secvenţa de explorare a mediului pentru fiecare set de parametri supuşi evaluării şi astfel apare un plus de viteză în execuţie. Un alt avantaj constă în faptul că evaluarea este mult mai precisă întrucât se va utiliza acelaşi set de citiri de la senzori şi poziţii (odometrie) pentru a valida diferite configuraţii de parametri. Separarea acestor două submodule poate fi eliminată atunci când s-au determinat parametrii potriviti aplicaţiei. Figura 32 redă interconexiunile între diferitele componente logice şi modul în care harta pe termen scurt este fuzionată cu harta pe termen lung după fiecare fază de validare. Cea mai bună poziţie la identificare, generată de modulul generator de poziţii, va contribui la updateul poziţiei robotului. După cum am menţionat anterior, în cazul în care localizarea continuă are acces la informaţie apriori despre mediu poate determina o diminuare a erorii la odometrie şi menţinerea acesteia la un nivel constant. Dar informaţia apriori nu este necesară la implementarea SLAM, întrucât algoritmul ar trebui să contruiască harta incremental pentru un mediu necunoscut. Din acest motiv modulul de validare al al hărţii prezintă nişte elemente specifice. Astfel, harta pe termen lung (care se generează) este iniţial vidă. Fiecare etapă (iteraţie) a localizării începe cu o hartă pe termen scurt vidă. Apoi, harta pe termen scurt este fuzionată cu citirile de la senzori până când se maturează. Procesul de validare a hărţii înregistrează harta pe termen scurt prin suprapunerea peste harta pe termen lung pentru fiecare din cele k poziţii oferite de generatorul de poziţii. Parametrul care defineşte spaţiul de căutare a validării hărţii determină ce poziţii returnează generatorul de poziţii. Diferenţa între poziţia presupusă a robotului şi poziţia cea mai bună la identificare este considerată a fi eroarea de odometrie. După updateul poziţiei robotului harta pe termen scurt este fuzionată cu harta pe termen lung şi o nouă etapă (iteraţie) de localizare începe cu o hartă pe termen scurt nouă şi vidă.

În acest sub capitol a fost descris modul de operare al algoritmului de mapare şi în continuare sunt redate aspecte specifice utilizate la proiectare şi implementare privind parametrii, configuraţia mediilor în teste şi simulări, alte probleme care apar şi soluţiile propuse pentru acestea.

4.1.2 Parametrizări şi configuraţia senzorilor

În dezvoltarea aplicaţiei s-a folost o gamă întregă de parametri pentru a determina obţinerea de performanţe ridicate. În continuare sunt descrişi parametrii definiţi, efectul lor asupra execuţiei, precum şi valorile alese pentru producerea unor rezultate bune.

Primul parametru de interes este dimensiunea spaţiului de căutare pentru validarea hărţii. Acest proces de validare (înregistrare) nu poate realiza identificarea între harta pe termen scurt şi harta pe termen lung pentru orice poziţie a robotului, întrucât ar presupune un efort computaţional major şi ar fi un procedeu supus erorilor. Funcţia care este implementată în generatorul de poziţii este limitată la căutarea a k poziţii probabile. Astfel apare problema determinării celor mai probabile k poziţii pe care le poate avea robotul în contextul în care robotul poate avea erori de odometrie în toate cele 3 grade de libertate (x, y, θ) obţinute de la ultima etapă de localizare (iteraţia anterioară). Generatorul de poziţii ar trebui să considere astfel aspecte cum ar fi poziţia curentă, viteza curentă şi mişcările efectuate de la ultima etapă de localizare, când generează poziţiile posibile. În cazul de faţă implementarea generatorului de poziţii este simplă dar eficientă. Astfel, funcţia de bază a generatorului presupune că eroarea de odometrie acumulată de la ultima etapă de localizare până în momentul curent este în domeniul a +/- 1 celule în matricea de ocupare în planul XoY. Eroarea pe axa orientării robotului (θ) este presupusă a fi în intervalul +/- 3º. Astfel putem considera că utilizând o rezoluţie de 1 celulă şi 1º va rezulta un total de 63 de poziţii de evaluat pentru procesul de validare (înregistrare) în timpul fiecărei etape de localizare.

Al doilea parametru de interes este maturitatea hărţii pe termen scurt, amintită şi anterior la descrierea modului în care se realizeaza validarea hărţii. Astfel, maturitatea hărţii pe termen scurt depinde de cantitatea de informaţie (citiri de la sonare) care va fi stocată în fiecare hartă pe termen scurt. Acest parametru poate fi dimensionat în vârsta (ex.: numărul de secunde de citire a senzorilor panâ la maturizare) sau în numărul de citiri. În aplicaţia de faţă am considerat alegerea unui număr de 45 de citiri de la senzori pentru fiecare hartă pe termen scurt.

Al treilea parametru este rezoluţia matricii de ocupare. Se consideră că atât harta pe termen lung cât şi harta pe termen scurt utilizează aceeaşi rezoluţie pentru reprezentare. Acest parametru determină cât de rafinată este informaţia stocată în hartă. Cu cât rezoluţia este mai mare cu atât creşte calitatea hărţii dar şi efortul computaţional. În experimentele efectuate s-a considerat o grilă de 35mmx35mm.

Ultimul parametru de interes este frecvenţa de comutare a sonarelor şi viteza robotului. Frecvenţa de comutare adresează de fapt temporizarea la momentul citirii pentru senzori şi cascadarea senzorilor pentru citiri pe toate direcţiile. Împreună cu viteza robotului aceste elementele determină densitatea de informaţie achiziţionată. În experimente robotul a rulat cu o viteză de 80mm/s şi a colectat informaţie la fiecare 50mS.

În continuare sunt descrise aspecte privind configuraţii de medii pentru experimente şi aspecte specifice pentru navigaţie şi localizare continuă.

4.1.3 Configuraţii de medii, navigaţie şi localizare continuă

În aplicaţia dezvoltată au fost testate mai multe configuraţii de mediu (camere). Pentru analiza metodei se impune prezentarea a două configuraţii care să pună în evidenţă avantajele şi problemele abordării propuse. Cele două configuraţii de mediu sunt caracterizate de centru deschis şi obiecte (volume) plasate pe conturul mediului pentru a asigura închiderea conturului. Cele două configuraţii propuse analizei în continuare sunt redate în figura următoare.

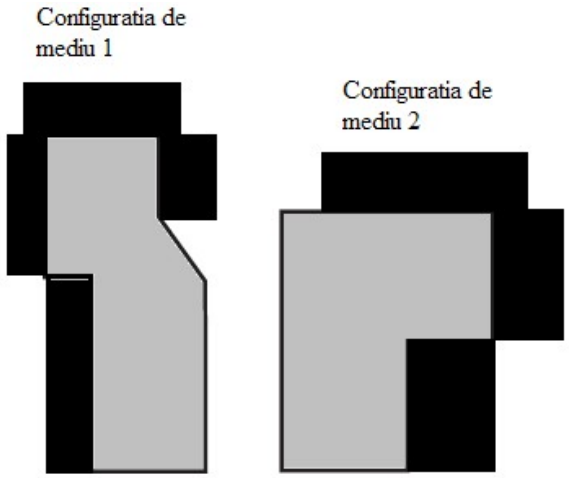


Figura 33 Configuraţii de mediu pentru teste

Extremităţile camerelor sunt suprafeţe de metal sau lemn şi astel se elimină riscul apariţiei fenomenului de reflexie speculară. Primul profil de cameră conţine pasaje relativ înguste, în timp ce a doua configuraţie conţine o zonă deschisă considerabilă care va intrduce noi probleme.

În ceea ce priveşte navigaţia robotul a fost programat sa opereze în regim trajectory tracking pe o traiectorie complexă, existând şi o posibilitate de comandă prin joystick. Datorită faptului că aplicaţia dezvoltată a fost separată în două componente funcţionale apare o limitare în navigaţie. Datorită faptului că navigaţia are loc în prima etapă şi harta e generată, offline, în

a doua etapă, comportamentul robotului trebuie să fie reactiv şi nu trebuie să necesite acces la harta pe termen lung. În operaţia de mapare a mediului este important ca robotul să îşi cunoască poziţia, altfel eroarea de odometrie va determina crearea unei hărţi distorsionate. În continuare sunt prezentate cele două configuraţii de camere propuse pentru analiză şi rezultatele obţinute în cazul în care encoderele sunt folosite ca singură sursă de informaţie de odometrie şi fără localizare continuă.

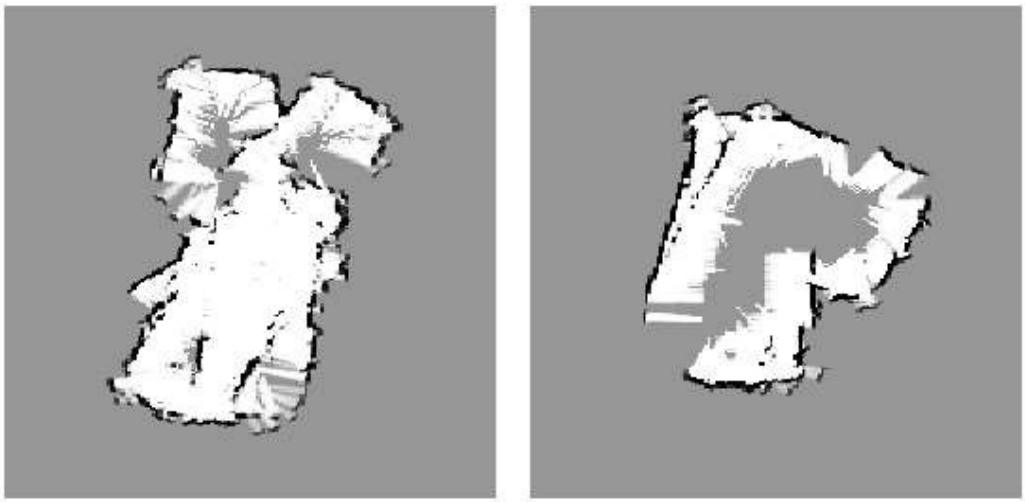


Figura 34 Maparea utilizând doar informaţia de la encodere

Înainte de a prezenta rezultatele utilizând localizarea continuă şi experimentele de parametrizare se aduce în discuţie un aspect foarte important şi anume, tratarea zonelor deschise, care au fost prezente şi în a doua configuraţie propusă analizei. Zonele deschise introduc probleme în localizarea continuă. În acest context, aceste zone se definesc ca porţiuni din mediul de operare în care senzorii nu pot detecta niciun obstacol şi astfel nu se poate primi informaţie de distanţă faţă de robot, afectând astfel eficienţa algoritmului. În timpul navigării prin mediul necunoscut robotul poate intra în zone libere şi astfel sonarele nu vor oferi informaţie utilă. Harta pe termen scurt este astfel vidă şi se va suprapune peste orice poziţie în harta pe termen lung, iar localizarea continuă nu va mai fi capabilă să compenseze erorile de odometrie introduse de encodere. În acest context dacă robotul operează în linie dreaptă cu viteză constantă eroare va fi mică şi se va acumula greu, însă navigaţia presupune un comportament variat în ceea ce priveşte traiectoria şi profilele de viteză pentru robot. O astfel de problemă este enunţată în continuare. Dacă presupunem că robotul navighează printr-o zonă deschisă dar un sonar detectează la un moment dat un perete. Astfel apare informaţie disponibilă pentru harta pe termen scurt. Dar acea citire va fi în ultima comutare pe etapa de localizare curentă astfel încât va fi singura informaţie din harta pe termen scurt şi se va realiza

astfel fuziunea cu harta pe termen lung. Apoi în următoarea perioadă de eşantionare harta pe termen scurt primeşte un număr semnificativ de citiri de la acelaşi perete. La validarea (înregistrarea) acestei hărţi pe termen scurt este suprapusă informaţia din aceasta cu informaţia existentă în harta pe termen lung existentă, care conţine acum doar o singură citire a sonarului. Neavând destulă informaţie pentru a realiza comparaţia între cele două hărţi în etapa de validare vom avea o precizie scăzută, astfel încât nu putem determina unghiul ci doar prezenţa obstacolului. Acest fenomen poate determina mărirea erorii de odometrie în sistem. Aplicaţia dezvoltată îşi propune să contracareze acest scenariu negativ asigurând că harta pe termen scurt are mereu destulă informaţie pentru o validare (înregistrare) corespunzătoare. Acest lucru este realizat prin parametrul de maturitate a hărţii pe termen scurt (de fapt prin stabilirea numărului de citiri de la sonare pentru etapa de fuziune şi nu de timpul de achiziţie a datelor) permiţând hărţii pe termen scurt sa fie activă un timp îndelungat la operarea în zone deschise.

În continuare sunt descrise aspecte de analiză şi interpretarea rezultatelor pentru metoda propusă precum şi îmbunătăţiri viitoare.

4.2 Analiză, limitări în proiectare şi îmbunătăţiri

Procesul de localizare continuă se bazează foarte mult pe precizia hărţii pe termen lung. În acelaşi timp calitatea hărţii petermen lung depinde de precizia procesului de localizare continuă. Parametrii descrişi în secţiunile anterioare (maturitatea, rezoluţia etc.) permit un control al preciziei hărţii rezultate, care pe lângă configuraţia mediului de operare şi robot, determină validitatea şi performanţele algoritmului. Se conturează astfel un set de dependenţe complexe care ne împiedică să determinăm parametrii optimi, o relaţie analitică, general valabilă. În figura următoare sunt redate rezultatele experimenteleor efectuate pe cele două configuraţii de test propuse anterior utilizând trei spaţii de căutare pentru componenta de validare (înregistrare) a hărţii.

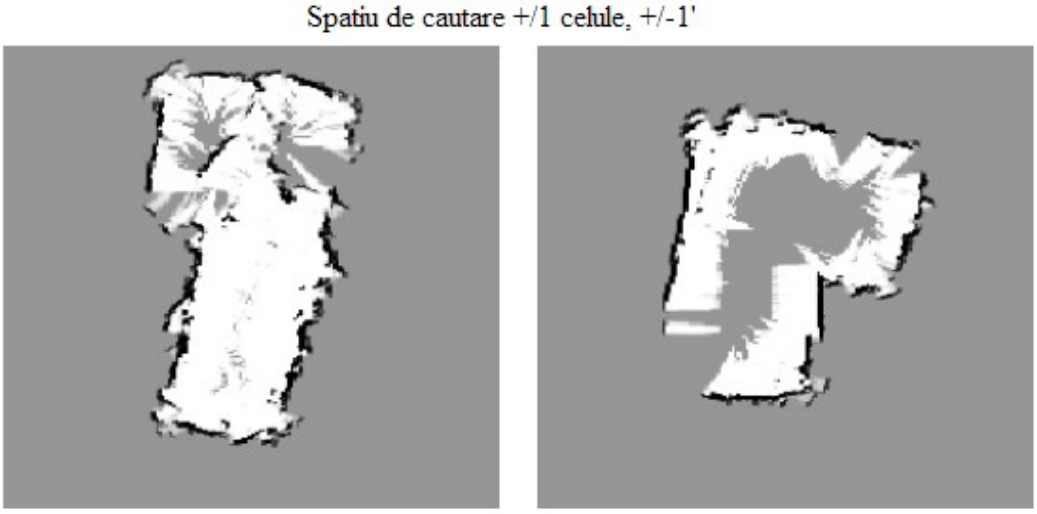


Figura 35 Rezultate pentru primul spaţiu de căutare

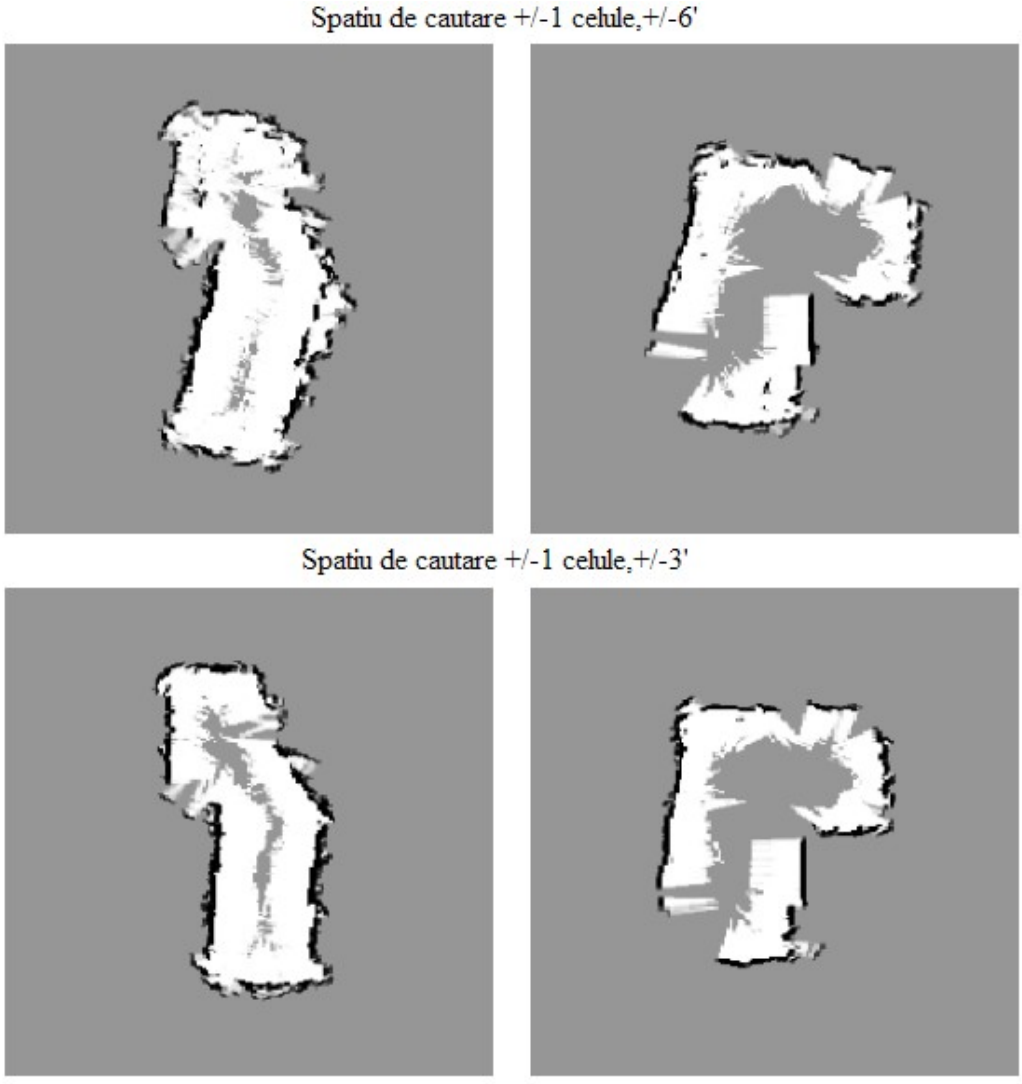


Figura 36 Rezultate pentru spaţii de căutare alternative

Se poate observa în figurile 35 şi 36 că dacă se utilizează un spaţiu de căutare mai mare se obţine un efort computaţional mai mare dar şi performanţe mai scăzute. Această observaţie întăreşte argumentul conform căruia alegerea parametrilor depinde mult de aplicaţie. Astfel, utilizând un spaţiu de căutare prea mare poate determina o validare (înregistrare) incorectă a hărţilor pe termen scurt, care determină la rândul lor sporirea erorii de odometrie cu implicaţii la crearea hărţii şi determinarea poziţiei robotului. În acelaşi mod, utilizând un spaţiu de căutare prea mic localizarea continuă poate fi dezactivată şi astfel erorile de odometrie nu pot fi eliminate. Pentru experimentele efectuate s-a ales ca spaţiu de căutare perechea (+/- 1 celule de matrice de ocupare, +/-3^o) pentru a obţine cele mai bune rezultate.

În ceea ce priveşte limitările, am menţionat anterior ca localizarea continuă încearcă să corecteze poziţia robotului prin compararea hărţii pe termen scurt cu harta pe termen lung. Precizia acestui proces este crucială pentru eficienţa întregului algoritm. Metoda propusă funcţionează corespunzător atât timp cât există citiri de la senzori legate de cel puţin un obiect din mediu care să existe şi în harta pe termen lung. Astfel se impune testarea în medii care nu conţin spaţii deschise mari, pentru a evita obţinerea unei hărţi vide pe termen scurt şi astfel procesul de înregistrare nu va mai estima corespunzător eroarea de odometrie. În aplicaţia propusă sonarele din dotarea robotului mobil au o rază de acţiune de la 0 (20cm) la 6.45m şi astfel cele mai bune rezultate se obţin în acest interval.

În final se poate concluziona că precizia globală a hărţii şi calitatea acesteia pot fi îmbunătăţite prin utilizarea unor modele optimizate pentru senzori. O posibilitate ar fi utilizarea unei rutine de calibrare a modelelor senzorilor utilizând o metoda de aproximare adaptivă (reţele neuronale, logică fuzzy) care să fie sensibilă la modificările asincrone din mediu şi din caracteristicile senzorului. Se propune pentru nivelul de navigaţie implementarea unui algoritm de ocolire a obstacolelor sau un algoritm follow-the-wall simplu. Pentru a extinde capabilităţile aplicaţiei dezvoltate se poate uitiliza abordarea bazată pe frontiere, dezvoltată de B. Yamauchi, [13], care defineşte frontiere între zonele ocupate şi cele libere. Abordarea bazată pe frontiere ar spori capacităţile şi precizia algoritmului dezvoltat prin faptul că utilizarea frontierelor ar determina adiţia de mai multă informaţie hărţii pe termen lung.

Capitolul 5

Sinteza nivelului de integrare a modulului de mapare cu aplicaţia de control în timp real pentru platforma robotică

5. Sinteza nivelului de integrare a modulului de mapare cu aplicaţia de control în timp real pentru platforma robotică

În capitolul curent sunt redate aspecte privind suportul hardware şi software pentru implementarea modulului de mapare a mediului precum şi nivelul de integrare cu aplicaţia de control în timp real. Astfel, în continuare sunt prezentate detalii privind platforma hardware de interfaţare cu sonarele, descrierea aplicaţiei de interfaţare cu modulele cu sonar, suportul de nivel superior pentru aplicaţia de mapare şi detalii privind proiectarea nivelului adaptor.

5.1 Suportul hardware pentru modulul de mapare a mediului

Modulul de mapare a mediului a fost proiectat ca o platforma hardware şi software care se aducă noi funcţionalităţi robotului mobil. Senzorii ultrasonici utilizaţi au fost prevăzuţi cu o placă auxiliară (module LV-MaxSonar-EZ0, [39]) care să realizeze condiţionarea (filtrare, atenuare, amplificare) semnalului, demultiplexarea şi conversia informaţiei de distanţă în mai multe forme, accesibile modulului cu MCU pentru prelucrare (semnal modulat PWM, semnal analogic, UART RS232). Modulul cu MCU este o placă de dezvoltare Olimex cu LPC2148 de la NXP, [38] şi interfaţează cu modulele cu sonar, determinând şi o tensiune de alimentare filtrată şi constantă pentru acestea cu ajutorul unei plăci cu 5 surse comandate cu LM317. În continuare sunt redate detalii privind proiectarea şi implementarea modulului hardware dar şi a aplicaţiei software de interfaţare cu sonarele.

5.1.1 Descrierea platformei cu sonare

Modulul cu MCU responsabil de interafaţa cu modulele cu sonare se bazează pe un MCU NXP LPC2148 de 32 de biţi care are în componenţă un nucleu ARM din familia ARM7TDMI-S. Acest MCU s-a dovedit a fi soluţia ideală pentru problema de proiectare propusă întrucât are un consum de putere redus şi oferă perfromanţe foarte bune precum şi un set puternic de periferice. LPC2148 prezintă on-chip, 40kB memorie SRAM, 32kB uz general + 8kB utilizaţi de modulul DMA pentru USB, 512kB memorie FLASH programabilă cu o interfaţă de accelerare de 128 de biţi pentru a asigura viteze mari de acces la 60 MHz. O interfaţă ISP/IAP este deasemenea prezentă prin intermediul unui bootloader on-chip pentru

scrierea şi citirea memoriei FLASH de program alături de un modul EmbeddedICE pentru debugging în timp real. Platforma prezintă deasemenea un controller USB2.0 cu 2kB de RAM pentru endpoint. Dintre modulele periferice se pot aminti, 2 convertoare ADC de 10 biţi cu 14 canale şi un timp de conversie de 2.44µs pe canal şi un singur DAC de 10 biţi pentru generarea de tensiuni variabile la ieşire, [40]. Arhitectura internă a LPC2148 este redată în figura următoare.

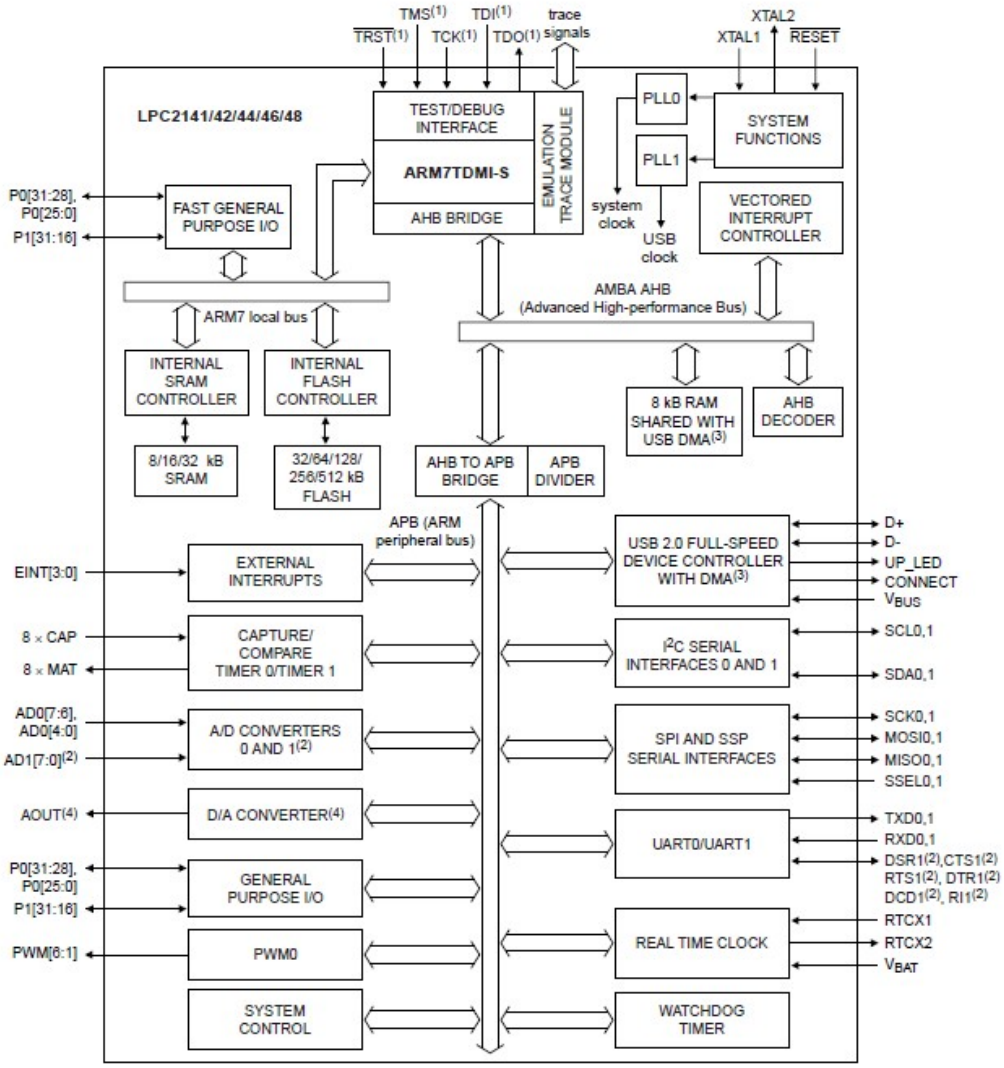


Figura 37 Arhitectura internă a MCU LPC2148 ARM7TMDI-S

Pentru temporizări LPC2148 prezintă în structură 2 timere de 32 de biţi pentru măsurarea intervalelor de timp sau numărarea evenimentelor externe. Modulele timer prezintă 4 unităţi input capture şi output compare, o unitate de generare PWM şi un watchdog. Componenta de comunicaţie serială este definită de prezenţa a două module UART, 16C550, două magistrale I2C de mare viteză (400kbit/s), un controller SPI şi SSP cu capacităţi de buffering şi transfer cu pachete de date cu lungime variabilă. Sistemul de întreruperi se bazează pe un controller cu

vectori de întrerupere cu priorităţi. Pentru a determina un consum de putere redus MCU este prevăzut cu o gama largă de facilităţi, cum ar fi scalarea frecvenţei funcţie de încărcare pentru optimizarea consumului, activarea / dezactivarea individuală a perifericelor.

Platforma embedded cu MCU LPC2148 utilizează un design eficient destinat aplicaţiilor de mică anvergură, cu consum redus de putere şi capabilităţi IO sporite. În continuare este redată schema modulului cu LPC2148.

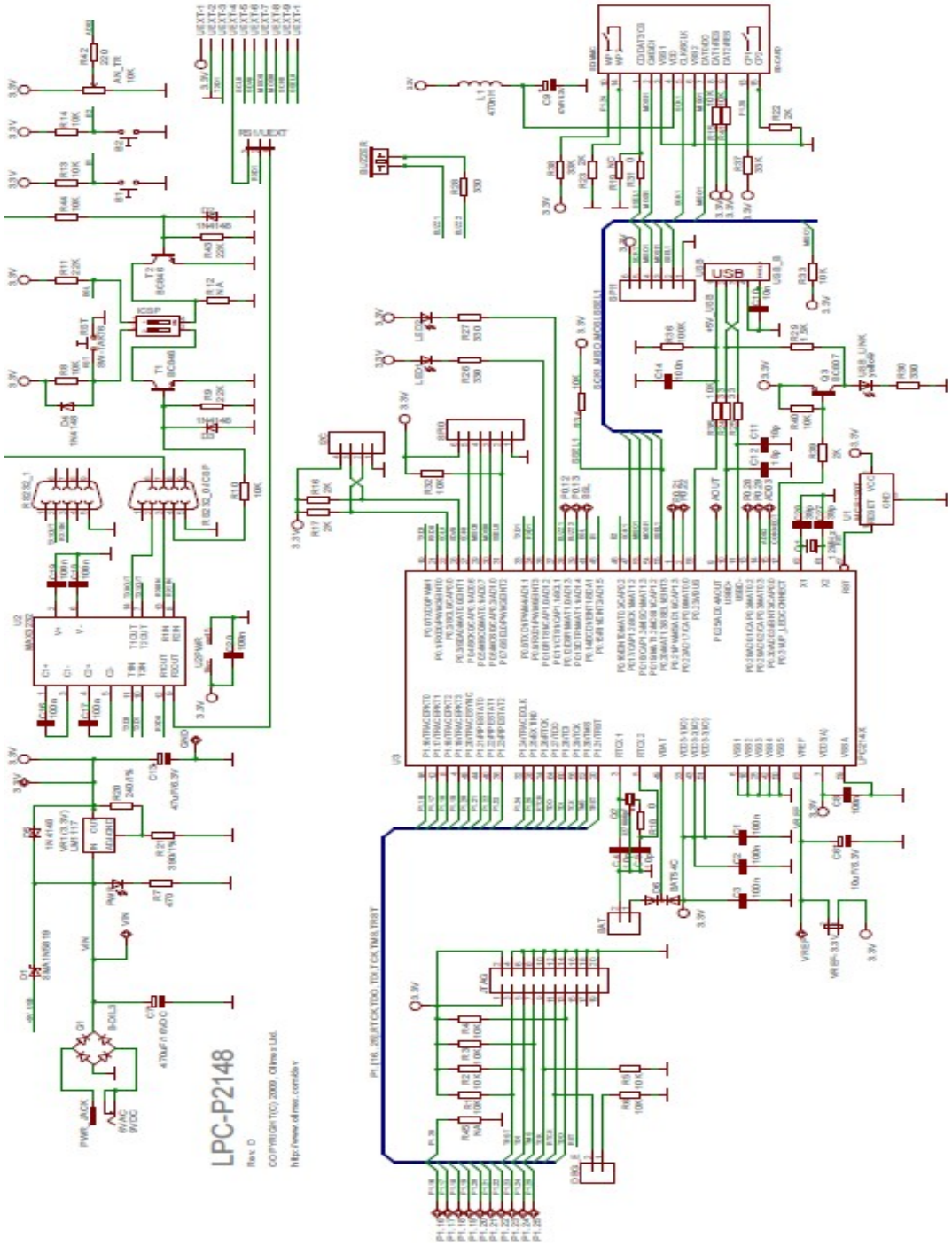


Figura 38 Schema modulului cu LPC2148 pentru interfaţarea cu sonarele

Platforma cu LPC2148 a fost prevăzută cu o conectică extinsă pentru aplicaţii embedded variate. Astfel, pe placa de dezvoltare se găseşte un modul de debugging JTAG, care este utilizat pentru programarea memoriei on-chip FLASH de program prin intermediul portului paralel utilizând utilitarele H-JTAG şi H-FLASHER. Pentru stocarea parametrilor sau pentru logging platforma are prevazut un connector şi modul de interfaţare pe SPI pentru carduri SD/MMC. Pentru a spori funcţionalitatea platforma este prevăzută cu un conector USB tip B pentru aplicaţii specifice de transfer de date sau VirtualCOM. Pe lângă aceste facilităţi platforma prezintă extensii şi elemente IO necesare simulării sau utilizării în aplicaţii simple (push-butons, trim-pot, buzzer). În aplicaţia curentă cele două porturi UART au fost programate individual pentru a servi în două taskuri concurente. Astfel, interfaţa full-modem prezentă la UART1 este utilizată pentru a interfaţa cu un modul GPS care emite pe suport RS-232 informaţia împachetată NMEA pentru validarea poziţiei robotului în operarea out-door. În acelaşi timp interfaţa UART0 este configurată pentru a transmite pachete cu informaţia de distanţă de la sonare către modulul embedded, cu CPU Intel Atom D525 de pe robot, responsabil cu implementarea taskurilor de control, monitorizare si comunicaţie.

Modulele cu sonare prezintă capabilităţi extinse de prelucrare a informaţiei brute de la senzorii ultrasonici oferind 3 posibilităţi de reprezentare a informaţiei de distanţă. Platforma cu senzor ultrasonic este echipată cu un MCU PIC16F676 care are rolul de a prelua informaţia brută de la senzorii ultrasonici şi de a realiza transformarea acestei informaţii în semnal PWM (147uS/in), anlogic (Vcc/512)/in şi Serial 0-Vcc, 9600-8n1. În continuare este redată schema unui modul cu sonar, în structura robotului existând 5 module de acest gen conectate în cascadă.

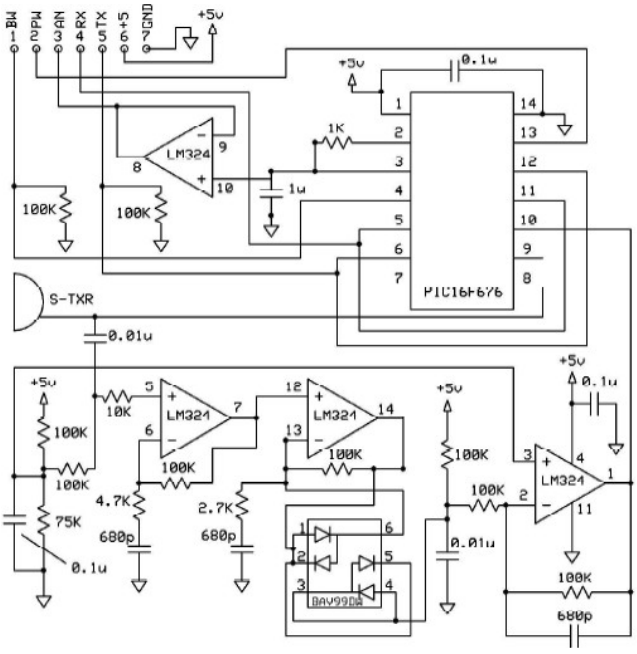


Figura 39 Modul de interfaţare cu senzorii ultrasonici

În continuare sunt redate câteva aspecte privind temporizările la citirea informaţiei de distanţă şi modul de operare al acestor module. Astfel, modulul cu sonar poate primi comandă de start achiziţie informaţie la activarea semnalului RX la 250mS după pornire. În cazul în care pinul RX este lăsat în gol sau conectat la 1 logic, senzorul va realiza un ciclu de calibrare de 49mS şi apoi va realiza citiri la fiecare 49mS, pinul RX fiind verificat mereu la finalul fiecărui ciclu. La fiecare 49mS după ce are loc citirea informaţiei modulul trimite 13 unde de 42KHz după care setează pinul de pulse witdth, PW, în 1 logic. La recepţia ecoului (la întalnirea unui obiect) pinul PW comută în 0 logic. În cazul în care nu se detectează niciun obiect pinul PW este ţinut în 1 logic 37.5mS şi diferenţa de 4.7mS (49mS-37.5mS) este dedicată ajustării semnalului analogic la un nivel corespunzător. Important de menţionat este faptul că atunci când se măsoară o distanţă mare imediat după măsurarea unei distanţe mici valoarea semnalului analogic poate să nu fie la nivelul corespunzător după primul ciclu.

După cum am menţionat anterior în momentul în care modulul cu sonar este alimentat acesta execută un ciclu de calibrare în primul ciclu de citire. Această informaţie de calibrare este utilă pentru a stabili sensibilitatea citirilor. Obiectul utilizat pentru calibrare trebuie să se afle la o distanţă potrivită ţinând cont şi de performanţele specificate în documentaţie (detecţie efectivă 0.2m-6.45m). Senzorul nu compensează variaţiile de temperatură, umiditate sau tensiune de alimentare şi de aceea este necesar un ciclu de calibrare de fiecare dată.

Pentru a asigura secvenţierea în citire a informaţiei de la sonare au fost propuse mai multe metode. Prima conexiune propusă este secvenţierea utilizând o buclă comandată, [39]. Exemplele de conectare sunt redate în figurile următoare pentru un număr de 3 sonare, putând exinde structura pentru cele 5 sonare utilizate în structura robotului mobil.

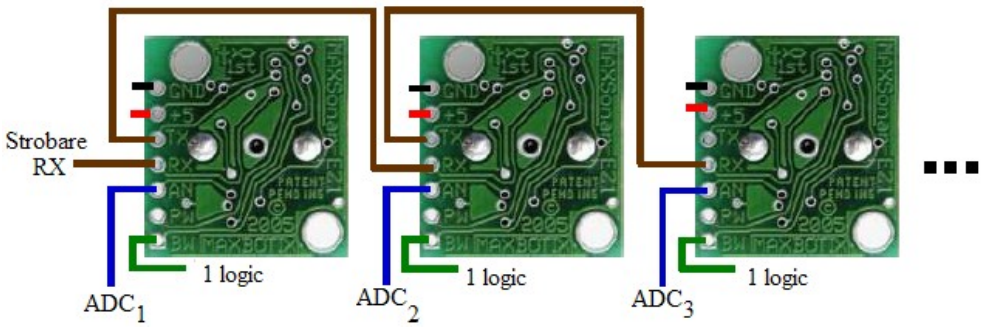


Figura 40 Conectarea în cascadă a modulelor cu sonar în buclă comandată

În configuraţia prezentată mai sus se conectează pinul TX cu pinul RX de la modulul următor şi se asigură că pinul de bandwidth, BW, este conectat la 1 logic. Pentru a asigura secvenţierea citirilor se realizează strobarea pinului RX de la primul modul, informaţia de distanţă putând fi citită pe intrările ADC ale MCU.

A doua configuraţie propusă pentru conectarea cascadată a modulelor cu sonare este redată în continuare, această configuraţie asigurând faţă de prima o funcţionare continuă în buclă constantă oferind la ieşire ultima valoare citită a distanţei, [39].

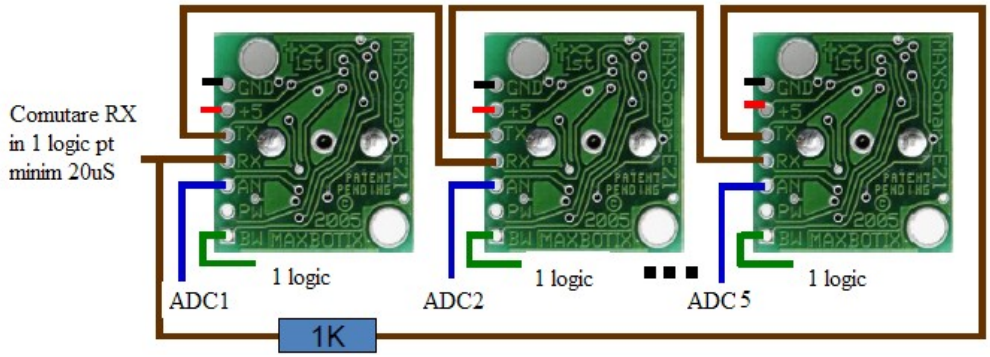


Figura 41 Conectarea în cascadă a modulelor cu sonar cu buclă constantă

Comparativ cu prima configuraţie se adaugă un rezistor pe reacţie de la pinul RX al modulului 5 până la pinul TX al modulului 1. Apoi, pentru a asigura pornirea modulelor se conectează pinul RX la 1 logic pentru 20uS după care MCU va pune acest pin in HZ pentru a asigura închiderea buclei de la ultimul modul la primul modul. În acest mod se asigură funcţionarea în cascadă şi inelul de senzori va cicla continuu menţinând validitatea valorilor citite. Pentru a asigura o tensiune de alimentare stabilizată pentru modulele cu sonare a fost proiectată şi o placă cu 5 surse comandate cu LM317 şi tranzitoare. Această placă are rolul de a seta tensiunea de alimentare a modulelor (între 2.5V şi 5.5V) şi de a asigura filtrarea tensiunii de alimentare, întrucât aceasta influenţează şi citirile. Placa adiţională se alimenteaza la 12V, direct de pe bateria robotului şi circuitul LM317 are rolul de a stabili la ieşire o tensiune constantă, stabilizată şi filtrată proporţională cu valoarea de reglare de pe terminalul adjust. Pentru a permite o comandă precisă s-a utilizat un tranzistor biploar pnp care are emitorul conectat la terminalul de output a regulatorului, LM317 şi colectorul la pinul de Vcc al modulului cu sonar. Comanda este dată de pe un pin de GPIO al MCU LPC2148 care este conectat în baza tranzitorului. Acest modul adiţional asigură şi un al doilea mod de secvenţiere a citirii de informaţie de la modulele cu sonar. În continuare este prezentată schema pentru modulul de alimentare a modulelor cu sonar care asigură şi interfaţa cu modulul cu LPC2148.

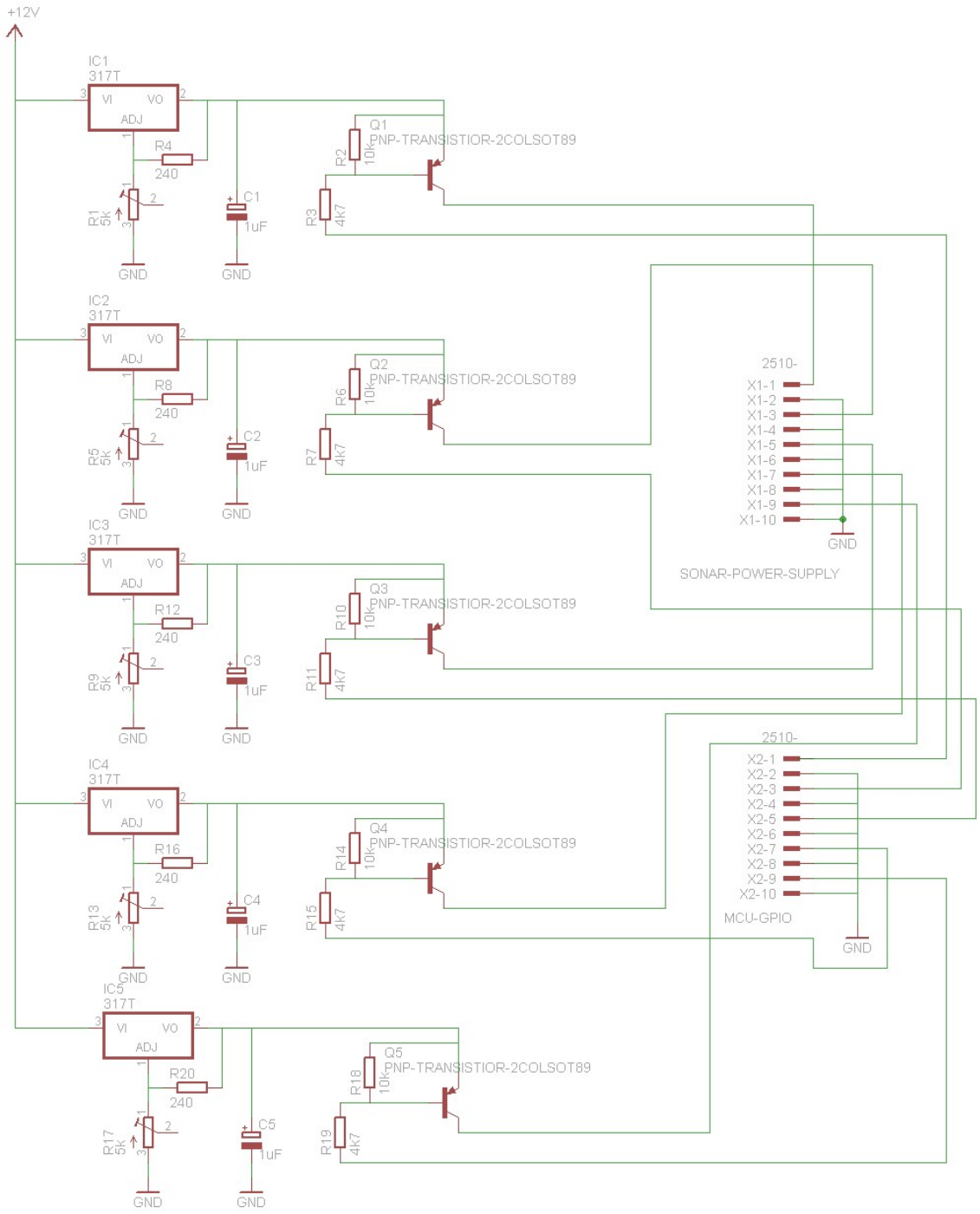


Figura 42 Modulul de alimentare şi de interfaţă cu sonarele pentru LPC2148

De-a lungul etapei de dezvoltare şi implementare au fost întâlnite câteva probleme privind secvenţierea sonarelor pentru citiri. Astfel, deşi au fost testate toate configuraţiile de cascadare, soluţia finală a fost una modificată, specifică problemei curente. Pornind de la configuraţia buclei comandate, s-au utilizat şi câteva modificări privind modul de comandă individual al modulelor cu sonar întrucât au apărut probleme la temporizări, probleme

manifestate datorită modului în care citirile sunt efectuate de către MCU şi momentul în care modulul cu sonar oferă informaţie de distanţă validă.

În continuare sunt redate câteva aspecte privind proiectarea şi implementarea aplicaţiei de interfaţare cu sonarele şi de comunicare cu platforma embedded de control şi monitorizare a robotului mobil.

5.1.2 Descrierea aplicaţiei de interfaţare cu sonarele şi interacţiunea cu nivelul superior

Modulul cu LPC2148, destinat interfaţării cu sonarele şi responsabil de a oferi informaţia de distanţă nivelului superior de control, rulează o aplicaţie C (5kB) în timp real. Aplicaţia are mai multe niveluri, fiind ierarhizată funcţie de apropierea de hardware sau de nivelul superior de comunicaţie cu aplicaţia de conducere în timp real. Astfel, la baza aplicaţiei ce rulează pe platforma cu LPC2148 , se află rutine software de iniţializare a sistemului şi a perifericelor. Ierarhia aplicaţiei este descrisă în figura următoare, fiind definite 3 nivele, nivelul de bază de acces la hardware , nivelul funcţional intermediar şi nivelul superior al aplicaţiei.

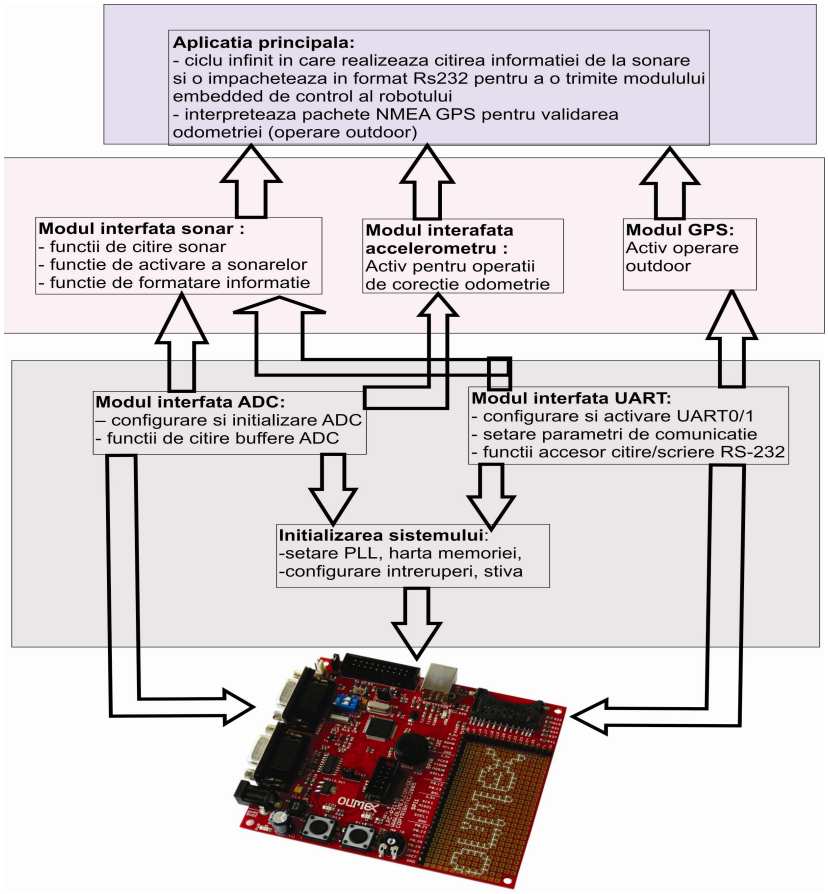


Figura 43 Ierarhia aplicaţiei de interfaţare cu sonarele, GPS şi accelerometru pt. LPC2148

Modulul principal utilizat în aplicaţia de faţă este modulul de interfaţare cu convertoarele ADC ale LPC2148. Acest modul implementează funcţii pentru iniţializarea convertoarelor ADC de 10 biţi prezente în structura de MCU LPC2148, setează funcţiile specifice pentru pinii ADC, power control, resetarea conversiilor şi biţii de control. Deasemenea acest modul exportă funcţionalitate pentru setarea frecvenţei de operare pentru ADC şi activarea modului de operare pentru achiziţie controlată software. Pe lângă funcţia de iniţializare modulul de interfaţare cu ADC, implementează şi două funcţii de citire a canalelor convertoarelor ADC0 şi ADC1. Astfel, se setează un anumit canal de citire, activând conversia, se setează apoi registrul de date corespunzător şi după finalul conversiei se realizează izolarea rezultatului conversiei. Următorul modul de bază din nivelul inferior este modulul de iniţializare a sistemului. Acest modul are rolul de a iniţializa diferitele subsisteme ale MCU, biţi pentru modurile de control şi întreruperi. O altă funcţie a acestui modul este de a seta configuraţia stivei şi memoriei heap, configuraţia frecvenţelor de lucru pentru periferice, PLL şi memorie, controllerul de memorie externă şi vectorii de întrerupere. Acest modul comunică cu modulul de interfaţă UART întrucât acesta are rolul de a seta frecvenţele de lucru pentru periferice, implicit şi pentru UART. În modulul de interfaţă UART se configurează bucla PLL şi frecvenţa de lucru funcţie de frecvenţa CPU, apoi se utilizează două funcţii de iniţializare a celor două porturi UART0/1. Pentru a accesa informaţia serială au fost dezvoltate două tipuri de funcţii accesor, de preluare şi de scriere a unui caracter sau şir de caractere pe linia serială RS-232. Pe lângă aceste funcţii de bază au fost dezvoltate şi funcţii auxiliare cu rolul de a converti octeţi binari în ASCII pentru transmiterea pe linia serială.

Cele două module din nivelul de bază exportă funcţionalitate nivelului funcţional intermediar, care asigură interfaţa cu sonarele, cu GPSul şi cu modulul cu accelerometru. Modulul de interfaţă cu sonarele prezintă intern funcţii dedicate pentru citirea informaţiei de la un anumit sonar folosind apeluri de funcţii specifice convertorului ADC corespunzător unde este conectat sonarul. Informaţia preluată este scalată pentru a obţine o reprezentare consistentă. Pe lângă funcţionalitatea de citire a sonarelor, acest modul de interfaţă oferă funcţii speciale pentru activarea sonarelor (setarea pinilor de BW, band-width) al modulelor cu sonar, funcţie pentru strobarea semnalului RX pentru operarea în cascadă. Deasemenea acest modul exportă o funcţie de întârziere în us pentru secvenţierea achiziţiei datelor şi o funcţie de formatare a datelor pentru transmisia către modulul superior de conducere şi monitorizare al robotului mobil.

Celelalte două module din nivelul funcţional intermediar, modulul de interfaţă cu GPS şi cel de interfaţă cu accelerometrul au rolul de a asigura un mecanism de validare a

odometriei. În primul caz modulul de interfaţă cu GPSul are rolul de a recepţiona, filtra şi parsa pachete GPS NMEA pentru extragerea informaţiei de poziţie. După parsarea şi extragerea informaţiei utile din pachete, se calculează coordonatele latitudine-longitudine la nivel de grade-minute-secunde. Apoi setându-se o origine, referinţă, se determină poziţia în plan a robotului. Această poziţie va fi comparată cu informaţia de la encodere şi va determina o corecţie a poziţiei robotului. De menţionat ca această funcţionalitate este disponibilă numai în funcţionarea out-door şi cu erori specifice GPS de până la 5m. Un al doilea mecanism dezvoltat pentru corecţiile informaţiei odometrice este modulul cu accelerometru. Acesta implementează o interfaţă minimală cu un accelerometru Freescale MMA7360L pe 3 axe(acces buffere ADC conectate cu ieşirile în tensiune X, Y ale accelerometrului). Informaţia de acceleraţie este supusă unor prelucrări recursive de integrare, filtrare şi estimare pentru a obţine informaţia de poziţie. De-a lungul testelor s-a observat că problemele care apar funcţie de pasul de integrare, metoda de integrare şi tehnica folosită pentru filtrare (filtru Kalman) pot degrada informaţia de poziţie. Pentru moment această funcţionalitate este destinată îmbunătăţirilor viitoare prin determinarea unui algoritm eficient de extragere a informaţiei dorite în contextul minimizării sau menţinerii constante a erorii care se acumulează la integrare.

Ultimul nivel al aplicaţiei de interfaţare cu sonarele este nivelul superior de execuţie şi comunicaţie cu platforma embedded de control şi monitorizare a robotului mobil. Această componentă are rolul de a iniţializa platforma şi de a asigura activarea sonarelor şi citirea sonarelor prin secvenţierea marcată de strobarea pinului RX al primului sonar din inel. La fiecare pas de eşantionare se citeşte informaţia, se formatează corespunzător şi se transmite pe linia serială. Pentru a evita valori extreme (instantanee) se realizează şi o mediere a valorilor citite înainte de a fi împachetate şi trimise pe linia serială.

Interacţiunea cu nivelul superior de conducere şi monitorizare a robotului mobil se realizează printr-o conexiune serială RS-232 între modulul cu LPC2148 şi modulul cu CPU Intel Atom D525. Informaţia transmisă de modulul cu LPC2148 este reprezentată sub forma unor pachete de date care conţin valoarea distanţei (în hexazecimal) de la sonar până la cel mai apropiat obiect pentru fiecare din cele 5 sonare. Pachetele au un caracter special „ > ” considerat antet, iar câmpul de date conţine caracterul „ , ” între valorile hexa pe 16biţi corespunzătoare fiecărui sonar. Valorile din câmpul de date sunt reprezentarea ASCII a valorilor citire de la ADC0/1 (valori de 10 biţi). Secvenţierea pachetelor se realizează utilizând CR şi NL (0x0D, 0x0A).

În continuare este prezentată implementarea suportului de nivel superior pentru mapare utilizând informaţia de la modulul de interfaţă cu sonarele. Se vor urmări aspecte privind bucla

de control în timp real şi modul în care este preluată şi prelucrată informaţia de la sonare precum şi modul în care informaţia recepţionată este exportată către clientul distribuit cu rolul de a implementa algoritmul SLAM offline.

5.2 Suportul de nivel superior pentru maparea mediului

Arhitectura sistemului software, pentru conducerea şi monitorizarea robotului mobil dezvoltat, are la bază o aplicaţie C++ care rulează pe unitatea de procesare embedded cu CPU Intel Atom D525, [41, 42], sub Linux-RTAI-Comedi şi oferă accesul direct la senzorii şi actuatorii robotului prin placa de achiziţie. Aceasta prin intermediul unor threaduri server (de control şi de date) oferă o interfaţă de control aplicaţiei client Java permiţându-i să activeze execuţia fazelor şi să primească informaţii direct de la senzorii robotului. Aplicaţia Java are rolul de a asigura activarea sau dezactivarea fazelor de execuţie ale robotului prin utilizarea unui instrument care-i permite să determine o execuţie secvenţială sau paralelă a taskurilor robotului, numit Grafcet. Structura se poate extinde cu un client Java cu drepturi restrânse cu rolul doar de a monitoriza execuţia şi de a primi informaţii despre starea sistemului, care în cazul de faţă va avea rolul de a implementa algoritmul de mapare a mediului utilizând offline SLAM.

Urmând o vedere de ansamblu aplicaţia dezvoltată se bazează pe un tipar de dezvoltare orientat obiect care conferă flexibilitate şi extensibilitate. În cele ce urmează sunt descrise detaliile de implementare a suportului concurent pentru realizarea taskurilor de conducere, monitorizare şi achiziţie de date pentru maparea offline a mediului.

5.2.1 Operarea multitasking concurent pentru conducere şi mapare offline

Aplicaţia de control tolereant la defecte în timp real ce rulează pe platforma embedded din structura robotului oferă suportul execuţiei a 2 macro-taskuri. Taskul de control implementează două bucle de conducere PI pentru reglarea turaţiei actuatorilor (MCC) şi o buclă externă de control a traiectoriei pentru operarea în regim trejectory tracking. Bucla de control este închisă de o cale de reacţie cu senzori care oferă informaţie utilă pentru odometrie şi pentru implementarea celui de-al doilea macro-task, taskul de monitorizare şi detecţie a defectelor. Acest task, preia informaţie de stare de la reacţia buclei şi aplicând

informaţia unui banc de filtre Kalman poate identifica apariţia şi discrimina defectele apărute în structura robotului în timpul operării.

Taskul de mapare offline la nivelul platformei embedded se concretizează doar într-un nivel de achiziţie a datelor şi de transmisie către clientul care implementează offline algoritmul SLAM. Efectiv în cadrul ciclului de control primar (terminologie definită în cadrul arhitecturii aplicaţiei de control tolereant la defecte) se realizează achiziţia de informaţie de la senzori şi se calculează comanda către actuatori la fiecare perioadă de eşantionare. Pe lângă aceste prelucrări, în această problemă propusă pentru proiectare, se adaugă variabile şi funcţii accesoriu specifice pentru interfaţarea cu modulul embedded cu LPC2148 responsabil cu achiziţia de date de la sonare.

Astfel, comunicaţia cu modulul de interfaţă cu sonarele se realizează prin intermediul unei conexiuni seriale RS-232, care determină transferul de pachete cu informaţia de distanţă de la sonare. În ciclul principal (alături de taskul de control şi odometrie) se iniţializează dimensiunea bufferului de recepţie pentru pachetele conţinând informaţia de distanţă de la sonare. Apoi se iniţializează elementele specifice pentru parsarea pachetelor (separatori câmp de date, headere) şi variabilele care vor reţine informaţia de distanţă accesibilă pentru operaţia de mapare. Următorul pas este iniţializarea descriptorului pentru controlul modemului pe linia RS-232. Pentru a realiza conexiunea cu modulul de interfaţă cu sonarele a fost creată o funcţie de accesare şi setare a parametrilor de comunicaţie pentru linia serială. Astfel, după accesul la dispozitivul serial, se salvează atributele curente de comunicaţie ale liniei şi se iniţializează o instanţă de configurare nouă. Se setează în continuare parametrii de comunicaţie (baudrate, paritate, dimensiunea caracterelor, caractere de control şi disciplina pe linie, activare procesare canonică) şi se reiniţializează modemul utilizând funcţii de nivel jos de control al io (ioctl).

Ciclul principal implementează funcţia control() care va fi executată la fiecare iteraţie (la fiecare perioadă de eşantionare) implementând taskul de control şi de achiziţie de date pentru mapare. Astfel, la fiecare perioadă de eşantionare se citesc 30 de octeţi (dimensiunea fixată a pachetului cu informaţie de distanţă de la modulul cu sonare) de pe linia serială şi se plasează în buffer. După citirea bufferului se realizează o operaţie de parsare a bufferului şi se extrage din câmpul de date informaţia de distanţă codificată ASCII hex pentru fiecare sonar. După extragerea informaţiei aceasta se converteşte în format zecimal pentru a putea fi uşor prelucrată în algoritmul de mapare SLAM. Alături de informaţia de la sonare se construieşte şi vectorul de valori ce conţine informaţia de odometrie, de la encodere.

În continuare este prezentat mecanismul de implementare a nivelului adaptor, care va realiza schimbul de informaţie între aplicaţia server ce rulează pe robot şi aplicaţia client care va executa algoritmul offline SLAM. În acest caz nivelul adaptor a fost implementat foarte simplu, pentru a elimina componente software adiţionale de transfer a informaţiei împachetate pentru TCP/IP (care ar putea introduce latenţe) s-a utilizat tehnologia NFS (Network File System). Astfel, în aplicaţia server ce rulează pe robot, informaţia odometrică şi cea achiziţionată de la sonare poate fi uşor scrisă în fişiere pe disk. Pentru a permite accesul aplicaţiei client la aceste fişiere se utilizează partajarea prin intermediul NFS. În acest mod, după achiziţia informaţiei, scrierea în fişier şi eliberarea descriptorilor de fişier, fişierele sunt accesibile remote şi clientului care poate prelua informaţia pentru algoritmul de mapare a mediului. Efectiv acest mecanism presupune utilizarea unei aplicaţii server daemon, nfsd, pe ambele maşini şi astfel fişierele cu parametrii pentru mapare să se afle de fapt local pe maşina client (unde va rula algoritmul SLAM) şi (aceste fişiere) să fie disponibile pentru scriere prin montare remote pe maşina server.

Astfel se realizează un mod eficient de acces partajat la fişierele cu informaţie pentru maparea mediului, setând drepturi de acces specifice clientului respectiv, funcţie de arhitectura sistemului distribuit.

În continuare sunt analizate perormanţele abordării propuse.

5.2.2 Analiza rezultatelor

În capitolul curent sunt prezentate aspecte de analiza a rezultateltor de validare obţinute de algoritmul de mapare a mediului pentru aplicaţia dezvoltată. Pornind de la exemplele de configuraţii de mediu prezentate în capitoul 4, la prezentarea metodei, în cele ce urmează sunt redate rezultatele obţinute în operarea robotului pentru 2 medii benchmark.

Prima configuraţie de mediu utilizată la validare este o cameră formată din două arii compuse care introduce şi problema mapării zonelor deschise şi se va analliza astfel şi modul în care algoritmul propus poate rezolva această problemă. Pentru primul test de validare se va paramteriza algoritmul de mapare astfel încât să se poată analiza performanţele şi compromisurile care se pot face pentru a îmbunătăţi rezultatele. În figurile minimale care redau mediul de test este redată şi poziţia de start a robotului, poziţia de odometrie 0.

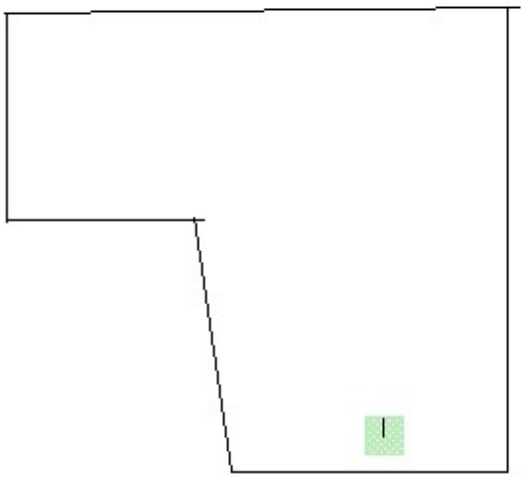


Figura 44 Harta minimală a camerei pentru primul test de validare

Etapa de explorare a mediului este prezentată în continuare, astfel putem observa care a fost comportamentul robotului şi astfel se poate justifica cum anumite porţiuni din hartă sunt mai bine reprezentate şi altele mai slab funcţie de numărul de citiri şi de modul în care robotul a parcurs mediul.

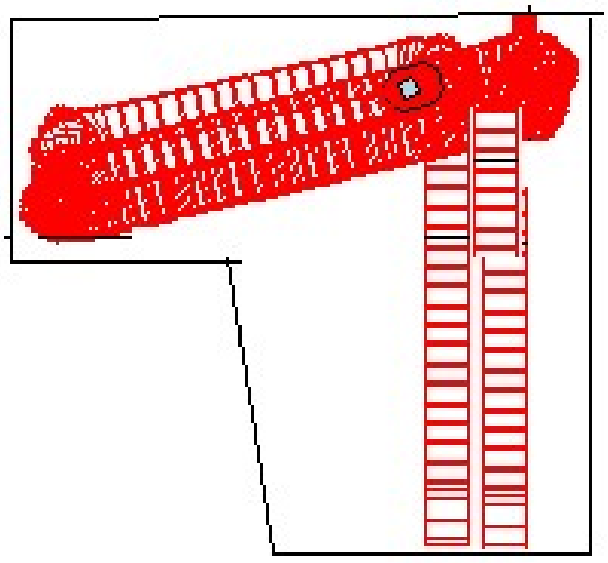


Figura 45 Etapa de explorare a mediului în vederea mapării

În figura următoare este prezentată harta obţinută în urma execuţiei algoritmului SLAM offline ce rulează pe maşina client distribuit în etapa de validare.

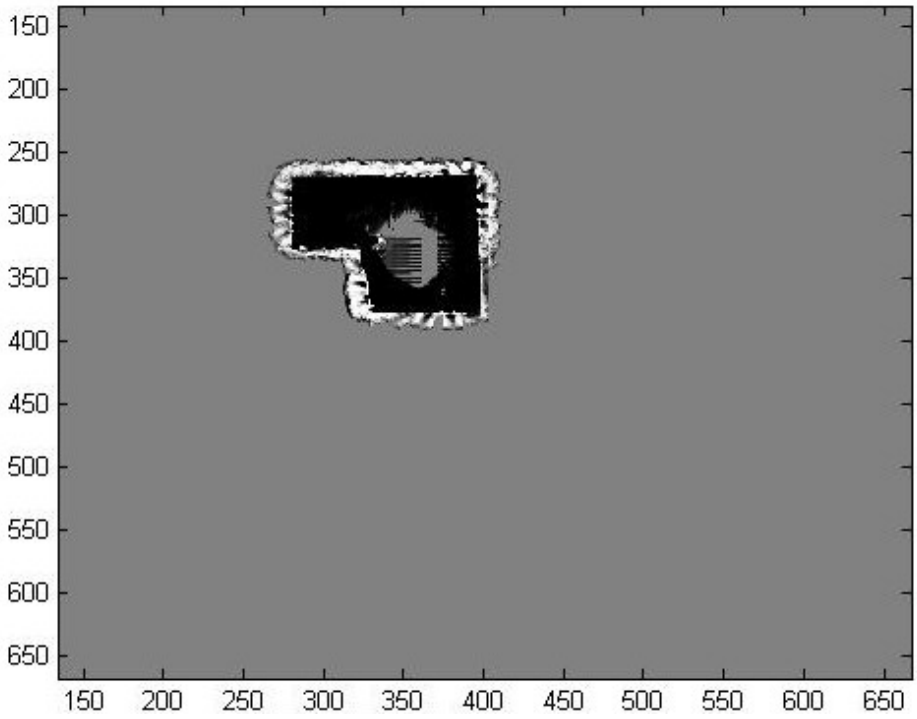


Figura 46 Harta obţinută cu algoritmul SLAM offline

Se poate observa din figură că rezultatul mapării este satisfăcător, conturul şi ariile sunt bine definite şi suprafaţa este acoperită complet. Testul de validare a fost completat cu încă un caz de parametrizare a algoritmului. Astfel, se prezintă contextul în care numărul de citiri de la sonare este dublat, se realizează astfel mai multe citiri şi precizia de reprezentare a hărţii este mai bună.

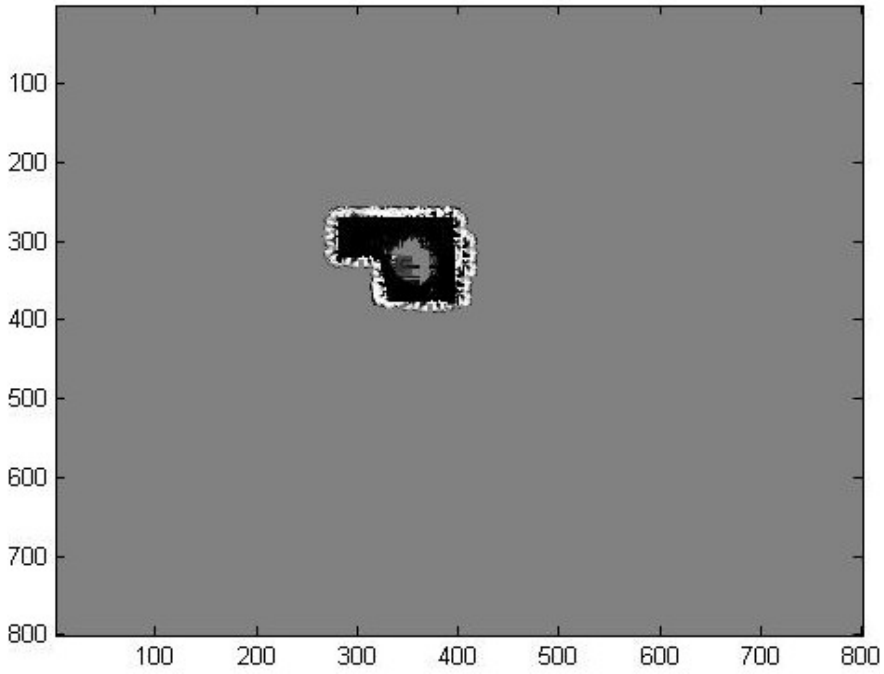


Figura 47 Harta obţinută cu algoritmul SLAM offline (nr. citiri sonare dublat)

Se poate observa în figura de mai sus ca anumite porţiuni ale conturului hărţii sunt mai bine definite acum şi calitatea este sporită. Pentru a realiza acest lucru însă a apărut un compromis la nivelul vitezei de rulare pentru a permite robotului să surprindă cât mai multă informaţie din mediu. Al doilea test de validare s-a realizat pe o configuraţie de coridor cu profil ∩.
Profilul celei de-a doua configuraţii de mediu pentru validare este redată în figura 48.

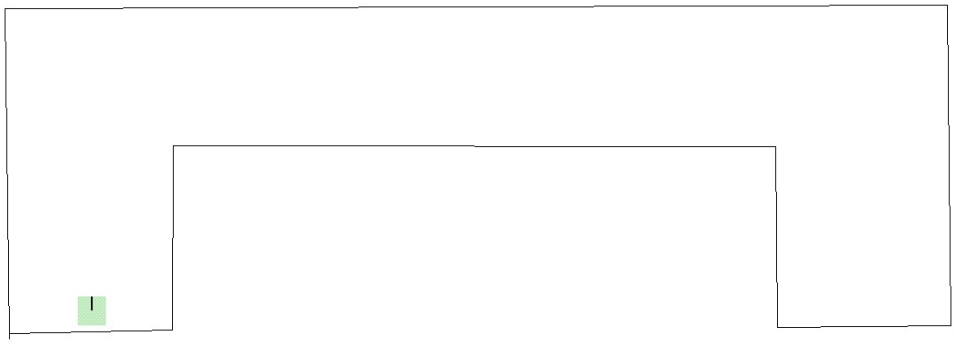


Figura 48 Harta minimală pentru al doilea test de validare – coridor

Pentru a da o imagine completă, dimensiunile configuraţiei pentru cel de-al doilea test de validare sunt de 14.8m pentru ramurile simetrice laterale, 26.6m distanţa între cele două ramuri şi 2.7m lărgimea coridorului. După etapa de achiziţie a datelor şi de execuţie a algoritmului SLAM s-a obţinut următoarea reprezentare a mediului.

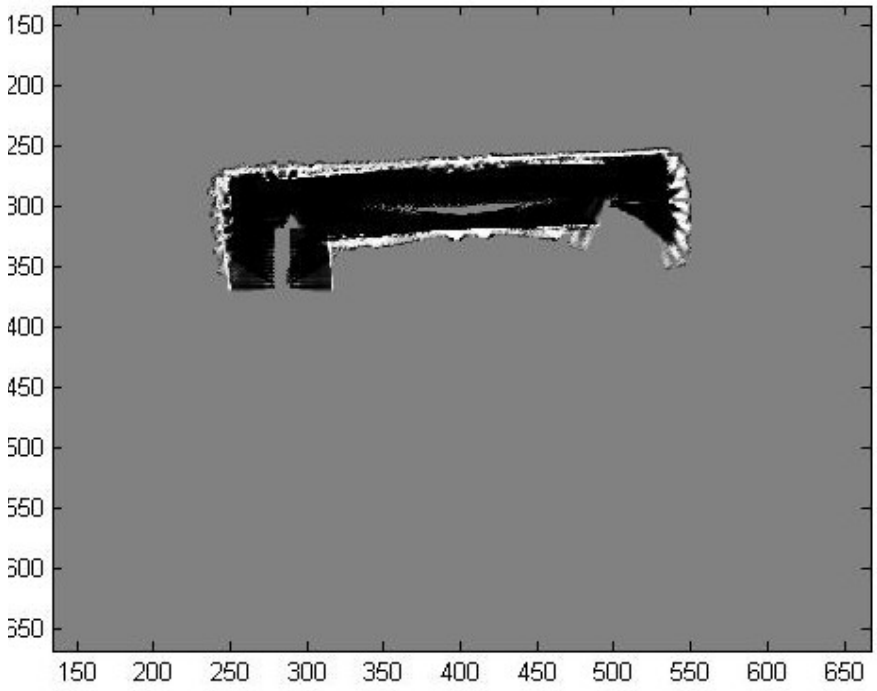


Figura 49 Harta obţinută cu algoritmul SLAM offline

Acestă figură surprinde foarte bine una din limitările algoritmului în ceea ce priveşte ponderea valorilor citite de sonare, în speţă a valorilor aflate la distanţă de robot. Un alt aspect se referă la zonele explorate de robot în scopul mapării. Astfel, în continuare pentru a valida harta obţinută anterior se prezintă modul în care robotul a parcurs harta propusă la validare.

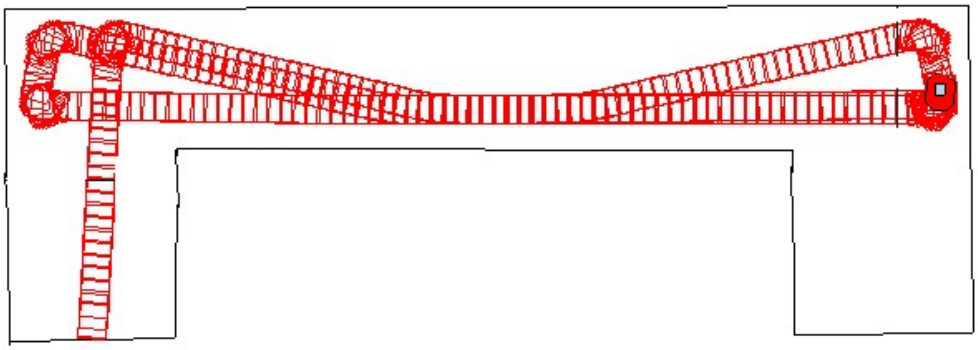


Figura 50 Etapa de explorare a mediului în vederea mapării

După cum se poate observa ramura din dreapta nu a fost parcursă efectiv de robot şi astfel măsurătorile de la sonare pentru acea zona au o pondere mică care va determina o reprezentare slabă a acelei zone.

În decursul acestui capitol au fost prezentate aspecte de sinteză hardware şi software pentru modulul de mapare a mediului propus în problema de proiectare. Pe lângă aspectele descriptive au fost analizate şi două cazuri în care algoritmul SLAM şi întreaga arhitectură au fost testate în scopul validării metodei dezvoltate. Analiza rezultatelor a avut rolul de a descrie avantajele şi limitările metodei propuse.

Capitolul 6

Concluzii

6. Concluzii

În lucrarea de faţă a fost introdusă o nouă abordare pentru maparea mediului. Pe lângă aspectele de proiectare hardware şi software de la fiecare nivel au fost prezentate şi aspecte de analiză a rezultatelor pentru a surprinde avantajele şi limitările metodei propuse. Astfel, pornind de la o aplicaţie de control tolerant la defecte în timp real s-au dezvoltat două module adiţionale care să confere structurii iniţiale flexibilitate şi autonomie, pe lângă funcţionalitatea nouă. Aplicaţia de timp real, prin taskul de control principal, a permis adăugarea unor apeluri de funcţii specifice de achiziţie de date de la modulul de interfaţă cu sonarele. Utilizând această informaţie aplicaţia server ce rulează pe robot poate comunica cu o maşină client care este responsabilă cu implementarea algoritmului SLAM offline. Parametrizarea algorimului este specifică problemei. Astfel se poate considera ca future work implementarea unei metode specializate de ajustare a parametrilor algoritmului SLAM. Un alt aspect deschis îmbunătăţirilor este modul de procesare intern al informaţiei (fuziunea Bayes a senzorilor) care se poare extinde prin utilizarea unor modele grafice probabilistice (grafuri de factori, reţele Markov) care să sporească eficienţa şi flexibilitatea la implementarea pe structuri hardware cu resurse limitate. Pentru a rafina performanţele algoritmului se propune şi utilizarea unui laser pentru ranging, precum şi un mecanism odometric vizual, utilizând algoritmi de prelucrare a imaginilor offboard, simplificând arhitectura embedded a robotului, limitând şi consumul de energie. Pentru operarea outdoor modulul GPS prezent va determina validarea odometriei şi va oferi suportul pentru localizare în cazul în care aplicaţia va fi extinsă la vehicule autonome ce vor rula în medii dinamice.

Capitolul 7

Bibliografie

7. Bibliografie

Descrierea aplicaţiei prezentate în această lucrare (partea de conducere în timp real) a fost publicată în diverse articole şi prezentată în cadrul unor prestigioase conferinţe de robotică şi automatică în anul 2010. În continuare sunt prezentate titlurile lucrărilor publicate :

1. Cristian Axenie, "Mobile Robot Fault Tolerant Control. Introducing ARTEMIC." In the 9th International Conference on Signal Processing, Robotics and Automation WSEAS Conference Proceedings Included in ISI/SCI Web of Science and Web of Knowledge, University of Cambridge, UK, February 2010, (ISBN:978-960-474-157-1, ISSN:1790-5117)

2 . Cristian Axenie, "A New Approach in Mobile Robot Fault Tolerant Control. Minimizing Costs and Extending Functionality", Included in ISI / SCI (Web of Science) WSEAS TRANSACTIONS ON SYSTEMS AND CONTROL ISSN: 1991-8763, pg 205-216

3. Cristian Axenie, Cernega Daniela, "Mobile Robot Fault Tolerant Control", IEEE/IACSIT ICIEE 2010 (International Conference on Information and Electronics Engineering), Shanghai, China, June 2010, Print ISBN: 978-1-4244-6367-1

4. Cristian Axenie, Cernega Daniela, " Adaptive Sliding Mode Controller Design for Mobile Robot Fault Tolerant Control", 19th IEEE International Workshop on Robotics in Alpe-Adria-Danube Region, Budapest, Hungary, June 2010, ISBN: 978-1-4244-6884-3 (Print ISBN 978-1-4244-6885-0)

5. Cristian Axenie, Razvan Solea, "Real Time Control Design for Mobile Robot Fault Tolerant Control", 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, July 15-17, 2010, Qingdao, ShanDong, China, ISBN: 978-1-4244-7102-7

În afară de lucrările publicate rezultatele obţinute au fost apreciate şi în cadrul unor competiţii studenţeşti de inginerie şi programare :

1. Lucrarea a fost premiată cu **locul I** (cea mai bună lucrare pe trackul robotică) şi cea mai bună lucrare a conferinţei internaţionale ZTS (Zilelel Tehnice Studenţeşti), organizată de Politehnica Timişoara în anul 2009

Lucrarea : „Mobile Robot Fault Tolerant Control" (Proceedings Vol. Pg. 352-358, ISSN 2066-3617)

2. Locul IV la Concursul Naţional de Programare Linux organizat de IBM cu tema „Linux based Mobile Robot Fault Tolerant Control application".

Capitolul 1

[1] P.K. Allen and Ioannis Stamos. Integration of range and image sensing for photorealistic 3D modeling. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 1435–1440, 2000.24
[2] D. Avots, E. Lim, R. Thibaux, and S. Thrun. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. Submitted for publication, 2002.
[3] R. Bajcsy, G. Kamberova, and Lucien Nocera. 3D reconstruction of environments for virtual reconstruction. In Proc. Of the 4th IEEE Workshop on Applications of Computer Vision, 2000.
[4] Y. Bar-Shalom and T. E. Fortmann. Tracking and Data Association. Academic Press, 1998.
[5] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun. Towards object mapping in dynamic environments with mobile robots. Submitted for publication, 2002.
[6] J. Borenstein, B. Everett, and L. Feng. Navigating Mobile Robots: Systems and Techniques. A. K. Peters, Ltd., Wellesley, MA, 1996.
[7] Graves, K., Adams, W., and SchultZ, A. Continuous localization in changing environments, 1997.
[8] Gutmann, J., Burgard, W., Fox, D., and Konolige, K. An experimental comparison of localization methods. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (1998).
[9] Moravec, H., and Elfes, A. E. High resolution maps from wide angle sonar. In Proceedings of the 1985 IEEE International Conference on Robotics and Automation (March 1985), pp. 116–121.

Capitolul 2

[10] Schultz, A. C., and Adams, W. Continuous localization using evidence grids. pp. 2833–2839.
[11] Van Dam, J. W. M., Kröse, B. J. A., and Groen, F. C. A. Adaptive sensor models. In 1996 IEEE/SICE/RSJ Intr. Conf. on Multisensor Fusion, and Integration for Intelligent Systems, Washington D.C (Dec. 8–11, 1996), pp. 705–712
[12] YAMAUCHI, B. A frontier-based approach for autonomous exploration. In Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (1997), pp. 146–151.
[13] Yamauchi, B., Schultz, A. C., and Adams, W. Mobile robot exploration and map-building with continuous localization. pp. 3715–3720.

Capitolul 3

[14] Bruno Siciliano, Oussama Khatib (Eds.) - Springer Handbook of Robotics, 2008
[15] Shuzhi Sam Ge, Frank L. Lewis - Autonomous Mobile Robots, Sensing, Control, Decision Making and Applications, CRC Press, 2008.
[16] J. Borenstein, H. R. Everett, and L. Feng - Where am I? Sensors and Methods for Mobile Robot Positioning .
[17] Owen Bishop - Robot Builder's Cookbook, Elsevier, Newnes, 2007.
[18] Mark W. Spong, Seth Hutchinson, M. Vidyasagar - Robot Modeling and Control , JOHN WILEY & SONS, INC., 2007.

[19] Axenie Cristian s.a. - A Client-Server Based Real-Time Control Tool for Complex Distributed Systems, Real-Time Linux Workshop 2007, Linz, Austria. (ftp://ftp.realtimelinuxfoundation.org/pub/events/rtlws-2007/Axenie.pdf)
[20] Karim Yaghmour – Adaptive Domain Environment for Operating Systems.
[21] Philippe Gerum - Life with Adeos, 2005.
[22] RTAI User Manual 3.4 – rev 0.3.
[23] Herman Bruyninckx - Real-Time and Embedded Guide, K.U.Leuven, Mechanical Engineering, 2002
[24] K Computing - Embedded and real-time Linux development.
[25] http://www.comedi.org/doc/index.html
[26] GRAFCET, http://www.lurpa.ens-cachan.fr/grafcet.html.
[27] Mark Mitchell, Jeffrey Oldham, and Alex Samuel - Advanced Linux Programming, NewRiders, 2001.
[28] Perruquetti Wilfrid, Jean Pierre Barbot - Sliding mode control in engineering, Marcel Dekker, 2002.
[29] D. Young s.a. - A Control Engineer's Guide to Sliding Mode Control, IEEE Trans. On Control Systems Technology v. 7 no. 3, pp. 328-342, 1999.
[30] J. Slotine and W. Li - Applied Nonlinear Control, Prentice Hall, New Jersey, 1991.
[31] Utkin, V.I. -Sliding Modes in Optimization and Control Problems, Springer Verlag, New York, 1992
[32] P. S. Mayback - The Kalman Filter: An Introduction to Concepts, in Autonomous Robot Vehicles, I.J. Cox, G. T.Wilfong (eds), Springer-Verlag, 1990.
[33] Mayback P.S. şi Hanlon R.D – Multiple Model Adaptive Estimation using a residual correlation kalman filter bank, IEEE Transactions on Aerospace and Electronic Systems, Vol36, No2, 2000.
[34]Yahya Chetouani - Using the Kalman Filtering for the Fault Detection and Isolation (FDI) in the Nonlinear Dynamic Processes, IJCRE 2008, Vol 6, A43.

Capitolul 4

[35] I.J. Cox and J.J. Leonard. Modeling a dynamic environment using a bayesian multiple hypothesis approach. Artificial Intelligence, 66(2):311–344, 1994.
[36] M. Csorba. Simultaneous Localisation and Map Building. PhD thesis, University of Oxford, Department of Engineering Science, 1997.
[37] A.J. Davison, Y.G. Cid, and N. Kita. Real-time SLAM. In IFAC/EURON Symposium on Intelligent Autonomous Vehicles, 2004.

Capitolul 5

[38] http://www.olimex.com/dev/lpc-p2148.html
[39] http://www.maxbotix.com/uploads/LV-MaxSonar-EZ0-Datasheet.pdf
[40] http://www.nxp.com/documents/user_manual/UM10139.pdf - LPC2148
[41] http://ark.intel.com/Product.aspx?id=49490 – Intel Atom D525
[42] www.asus.com - Asus AT5NM10-I INTEL NM10 (CPU ATOM D510 Dual Core on board)

Capitolul 8

Anexe

8. Anexe

8.1 Codul sursă C al modului de interfaţare cu senzorii ultrasonici bazat pe MCU LPC2148

CODUL DE INIŢIALIZARE MCU
Fişier : Startup.s

```
;/*****************************************************************************/
;/* STARTUP.S: Startup file for Philips LPC2000                               */
;/*****************************************************************************/
;/* <<< Use Configuration Wizard in Context Menu >>>                          */
;/*****************************************************************************/
;/* This file is part of the uVision/ARM development tools.                   */
;/* Copyright (c) 2005-2007 Keil Software. All rights reserved.               */
;/* This software may only be used under the terms of a valid, current,       */
;/* end user licence from KEIL for a compatible version of KEIL software      */
;/* development tools. Nothing else gives you the right to use this software. */
;/*****************************************************************************/
;/*
; *  The STARTUP.S code is executed after CPU Reset. This file may be
; *  translated with the following SET symbols. In uVision these SET
; *  symbols are entered under Options - ASM - Define.
; *
; *  REMAP: when set the startup code initializes the register MEMMAP
; *  which overwrites the settings of the CPU configuration pins. The
; *  startup and interrupt vectors are remapped from:
; *     0x00000000  default setting (not remapped)
; *     0x80000000  when EXTMEM_MODE is used
; *     0x40000000  when RAM_MODE is used
; *  EXTMEM_MODE: when set the device is configured for code execution
; *  from external memory starting at address 0x80000000.
; *
; *  RAM_MODE: when set the device is configured for code execution
; *  from on-chip RAM starting at address 0x40000000.
; *
; *  EXTERNAL_MODE: when set the PIN2SEL values are written that enable
; *  the external BUS at startup.
; */


; Standard definitions of Mode bits and Interrupt (I & F) flags in PSRs

Mode_USR        EQU     0x10
Mode_FIQ        EQU     0x11
Mode_IRQ        EQU     0x12
Mode_SVC        EQU     0x13
Mode_ABT        EQU     0x17
Mode_UND        EQU     0x1B
Mode_SYS        EQU     0x1F

I_Bit           EQU     0x80            ; when I bit is set, IRQ is disabled
F_Bit           EQU     0x40            ; when F bit is set, FIQ is disabled


;// <h> Stack Configuration (Stack Sizes in Bytes)
;//   <o0> Undefined Mode      <0x0-0xFFFFFFFF:8>
;//   <o1> Supervisor Mode     <0x0-0xFFFFFFFF:8>
;//   <o2> Abort Mode          <0x0-0xFFFFFFFF:8>
;//   <o3> Fast Interrupt Mode <0x0-0xFFFFFFFF:8>
;//   <o4> Interrupt Mode      <0x0-0xFFFFFFFF:8>
;//   <o5> User/System Mode    <0x0-0xFFFFFFFF:8>
;// </h>
```

```
UND_Stack_Size  EQU     0x00000000
SVC_Stack_Size  EQU     0x00000008
ABT_Stack_Size  EQU     0x00000000
FIQ_Stack_Size  EQU     0x00000000
IRQ_Stack_Size  EQU     0x00000080
USR_Stack_Size  EQU     0x00000400

ISR_Stack_Size  EQU     (UND_Stack_Size + SVC_Stack_Size + ABT_Stack_Size + \
                         FIQ_Stack_Size + IRQ_Stack_Size)

                AREA    STACK, NOINIT, READWRITE, ALIGN=3

Stack_Mem       SPACE   USR_Stack_Size
__initial_sp    SPACE   ISR_Stack_Size

Stack_Top


;// <h> Heap Configuration
;//   <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF>
;// </h>

Heap_Size       EQU     0x00000000

                AREA    HEAP, NOINIT, READWRITE, ALIGN=3
__heap_base
Heap_Mem        SPACE   Heap_Size
__heap_limit


; VPBDIV definitions
VPBDIV          EQU     0xE01FC100      ; VPBDIV Address

;// <e> VPBDIV Setup
;// <i> Peripheral Bus Clock Rate
;//   <o1.0..1>   VPBDIV: VPB Clock
;//               <0=> VPB Clock = CPU Clock / 4
;//               <1=> VPB Clock = CPU Clock
;//               <2=> VPB Clock = CPU Clock / 2
;//   <o1.4..5>   XCLKDIV: XCLK Pin
;//               <0=> XCLK Pin = CPU Clock / 4
;//               <1=> XCLK Pin = CPU Clock
;//               <2=> XCLK Pin = CPU Clock / 2
;// </e>
VPBDIV_SETUP    EQU     0
VPBDIV_Val      EQU     0x00000000


; Phase Locked Loop (PLL) definitions
PLL_BASE        EQU     0xE01FC080      ; PLL Base Address
PLLCON_OFS      EQU     0x00            ; PLL Control Offset
PLLCFG_OFS      EQU     0x04            ; PLL Configuration Offset
PLLSTAT_OFS     EQU     0x08            ; PLL Status Offset
PLLFEED_OFS     EQU     0x0C            ; PLL Feed Offset
PLLCON_PLLE     EQU     (1<<0)          ; PLL Enable
PLLCON_PLLC     EQU     (1<<1)          ; PLL Connect
PLLCFG_MSEL     EQU     (0x1F<<0)       ; PLL Multiplier
PLLCFG_PSEL     EQU     (0x03<<5)       ; PLL Divider
PLLSTAT_PLOCK   EQU     (1<<10)         ; PLL Lock Status

;// <e> PLL Setup
;//   <o1.0..4>   MSEL: PLL Multiplier Selection
;//               <1-32><#-1>
;//               <i> M Value
;//   <o1.5..6>   PSEL: PLL Divider Selection
;//               <0=> 1   <1=> 2   <2=> 4   <3=> 8
;//               <i> P Value
;// </e>
PLL_SETUP       EQU     1
```

```
PLLCFG_Val      EQU     0x00000024


; Memory Accelerator Module (MAM) definitions
MAM_BASE        EQU     0xE01FC000      ; MAM Base Address
MAMCR_OFS       EQU     0x00            ; MAM Control Offset
MAMTIM_OFS      EQU     0x04            ; MAM Timing Offset

;// <e> MAM Setup
;//   <o1.0..1>   MAM Control
;//               <0=> Disabled
;//               <1=> Partially Enabled
;//               <2=> Fully Enabled
;//               <i> Mode
;//   <o2.0..2>   MAM Timing
;//               <0=> Reserved  <1=> 1   <2=> 2   <3=> 3
;//               <4=> 4         <5=> 5   <6=> 6   <7=> 7
;//               <i> Fetch Cycles
;// </e>
MAM_SETUP       EQU     1
MAMCR_Val       EQU     0x00000002
MAMTIM_Val      EQU     0x00000004


; External Memory Controller (EMC) definitions
EMC_BASE        EQU     0xFFE00000      ; EMC Base Address
BCFG0_OFS       EQU     0x00            ; BCFG0 Offset
BCFG1_OFS       EQU     0x04            ; BCFG1 Offset
BCFG2_OFS       EQU     0x08            ; BCFG2 Offset
BCFG3_OFS       EQU     0x0C            ; BCFG3 Offset

;// <e> External Memory Controller (EMC)
EMC_SETUP       EQU     0

;//   <e> Bank Configuration 0 (BCFG0)
;//     <o1.0..3>   IDCY: Idle Cycles <0-15>
;//     <o1.5..9>   WST1: Wait States 1 <0-31>
;//     <o1.11..15> WST2: Wait States 2 <0-31>
;//     <o1.10>     RBLE: Read Byte Lane Enable
;//     <o1.26>     WP: Write Protect
;//     <o1.27>     BM: Burst ROM
;//     <o1.28..29> MW: Memory Width  <0=>  8-bit  <1=> 16-bit
;//                                   <2=> 32-bit  <3=> Reserved
;//   </e>
BCFG0_SETUP EQU         0
BCFG0_Val   EQU         0x0000FBEF

;//   <e> Bank Configuration 1 (BCFG1)
;//     <o1.0..3>   IDCY: Idle Cycles <0-15>
;//     <o1.5..9>   WST1: Wait States 1 <0-31>
;//     <o1.11..15> WST2: Wait States 2 <0-31>
;//     <o1.10>     RBLE: Read Byte Lane Enable
;//     <o1.26>     WP: Write Protect
;//     <o1.27>     BM: Burst ROM
;//     <o1.28..29> MW: Memory Width  <0=>  8-bit  <1=> 16-bit
;//                                   <2=> 32-bit  <3=> Reserved
;//   </e>
BCFG1_SETUP EQU         0
BCFG1_Val   EQU         0x0000FBEF

;//   <e> Bank Configuration 2 (BCFG2)
;//     <o1.0..3>   IDCY: Idle Cycles <0-15>
;//     <o1.5..9>   WST1: Wait States 1 <0-31>
;//     <o1.11..15> WST2: Wait States 2 <0-31>
;//     <o1.10>     RBLE: Read Byte Lane Enable
;//     <o1.26>     WP: Write Protect
;//     <o1.27>     BM: Burst ROM
;//     <o1.28..29> MW: Memory Width  <0=>  8-bit  <1=> 16-bit
;//                                   <2=> 32-bit  <3=> Reserved
```

```
;//   </e>
BCFG2_SETUP EQU         0
BCFG2_Val   EQU         0x0000FBEF

;//   <e> Bank Configuration 3 (BCFG3)
;//     <o1.0..3>   IDCY: Idle Cycles <0-15>
;//     <o1.5..9>   WST1: Wait States 1 <0-31>
;//     <o1.11..15> WST2: Wait States 2 <0-31>
;//     <o1.10>     RBLE: Read Byte Lane Enable
;//     <o1.26>     WP: Write Protect
;//     <o1.27>     BM: Burst ROM
;//     <o1.28..29> MW: Memory Width  <0=>  8-bit  <1=> 16-bit
;//                                   <2=> 32-bit  <3=> Reserved
;//   </e>
BCFG3_SETUP EQU         0
BCFG3_Val   EQU         0x0000FBEF

;// </e> End of EMC


; External Memory Pins definitions
PINSEL2         EQU     0xE002C014      ; PINSEL2 Address
PINSEL2_Val     EQU     0x0E6149E4      ; CS0..3, OE, WE, BLS0..3,
                                        ; D0..31, A2..23, JTAG Pins


                PRESERVE8


; Area Definition and Entry Point
;  Startup Code must be linked first at Address at which it expects to run.

                AREA    RESET, CODE, READONLY
                ARM


; Exception Vectors
;  Mapped to Address 0.
;  Absolute addressing mode must be used.
;  Dummy Handlers are implemented as infinite loops which can be modified.

Vectors         LDR     PC, Reset_Addr
                LDR     PC, Undef_Addr
                LDR     PC, SWI_Addr
                LDR     PC, PAbt_Addr
                LDR     PC, DAbt_Addr
                NOP                             ; Reserved Vector
;               LDR     PC, IRQ_Addr
                LDR     PC, [PC, #-0x0FF0]      ; Vector from VicVectAddr
                LDR     PC, FIQ_Addr

Reset_Addr      DCD     Reset_Handler
Undef_Addr      DCD     Undef_Handler
SWI_Addr        DCD     SWI_Handler
PAbt_Addr       DCD     PAbt_Handler
DAbt_Addr       DCD     DAbt_Handler
                DCD     0                       ; Reserved Address
IRQ_Addr        DCD     IRQ_Handler
FIQ_Addr        DCD     FIQ_Handler

Undef_Handler   B       Undef_Handler
SWI_Handler     B       SWI_Handler
PAbt_Handler    B       PAbt_Handler
DAbt_Handler    B       DAbt_Handler
IRQ_Handler     B       IRQ_Handler
FIQ_Handler     B       FIQ_Handler


; Reset Handler
```

```
                EXPORT  Reset_Handler
Reset_Handler


; Setup External Memory Pins
                IF      :DEF:EXTERNAL_MODE
                LDR     R0, =PINSEL2
                LDR     R1, =PINSEL2_Val
                STR     R1, [R0]
                ENDIF


; Setup External Memory Controller
                IF      EMC_SETUP <> 0
                LDR     R0, =EMC_BASE

                IF      BCFG0_SETUP <> 0
                LDR     R1, =BCFG0_Val
                STR     R1, [R0, #BCFG0_OFS]
                ENDIF

                IF      BCFG1_SETUP <> 0
                LDR     R1, =BCFG1_Val
                STR     R1, [R0, #BCFG1_OFS]
                ENDIF

                IF      BCFG2_SETUP <> 0
                LDR     R1, =BCFG2_Val
                STR     R1, [R0, #BCFG2_OFS]
                ENDIF

                IF      BCFG3_SETUP <> 0
                LDR     R1, =BCFG3_Val
                STR     R1, [R0, #BCFG3_OFS]
                ENDIF

                ENDIF   ; EMC_SETUP


; Setup VPBDIV
                IF      VPBDIV_SETUP <> 0
                LDR     R0, =VPBDIV
                LDR     R1, =VPBDIV_Val
                STR     R1, [R0]
                ENDIF


; Setup PLL
                IF      PLL_SETUP <> 0
                LDR     R0, =PLL_BASE
                MOV     R1, #0xAA
                MOV     R2, #0x55

;  Configure and Enable PLL
                MOV     R3, #PLLCFG_Val
                STR     R3, [R0, #PLLCFG_OFS]
                MOV     R3, #PLLCON_PLLE
                STR     R3, [R0, #PLLCON_OFS]
                STR     R1, [R0, #PLLFEED_OFS]
                STR     R2, [R0, #PLLFEED_OFS]

;  Wait until PLL Locked
PLL_Loop        LDR     R3, [R0, #PLLSTAT_OFS]
                ANDS    R3, R3, #PLLSTAT_PLOCK
                BEQ     PLL_Loop

;  Switch to PLL Clock
                MOV     R3, #(PLLCON_PLLE:OR:PLLCON_PLLC)
```

```
                STR     R3, [R0, #PLLCON_OFS]
                STR     R1, [R0, #PLLFEED_OFS]
                STR     R2, [R0, #PLLFEED_OFS]
                ENDIF   ; PLL_SETUP


; Setup MAM
                IF      MAM_SETUP <> 0
                LDR     R0, =MAM_BASE
                MOV     R1, #MAMTIM_Val
                STR     R1, [R0, #MAMTIM_OFS]
                MOV     R1, #MAMCR_Val
                STR     R1, [R0, #MAMCR_OFS]
                ENDIF   ; MAM_SETUP


; Memory Mapping (when Interrupt Vectors are in RAM)
MEMMAP          EQU     0xE01FC040      ; Memory Mapping Control
                IF      :DEF:REMAP
                LDR     R0, =MEMMAP
                IF      :DEF:EXTMEM_MODE
                MOV     R1, #3
                ELIF    :DEF:RAM_MODE
                MOV     R1, #2
                ELSE
                MOV     R1, #1
                ENDIF
                STR     R1, [R0]
                ENDIF


; Initialise Interrupt System
;  ...


; Setup Stack for each mode

                LDR     R0, =Stack_Top

;  Enter Undefined Instruction Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
                MOV     SP, R0
                SUB     R0, R0, #UND_Stack_Size

;  Enter Abort Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_ABT:OR:I_Bit:OR:F_Bit
                MOV     SP, R0
                SUB     R0, R0, #ABT_Stack_Size

;  Enter FIQ Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_FIQ:OR:I_Bit:OR:F_Bit
                MOV     SP, R0
                SUB     R0, R0, #FIQ_Stack_Size

;  Enter IRQ Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_IRQ:OR:I_Bit:OR:F_Bit
                MOV     SP, R0
                SUB     R0, R0, #IRQ_Stack_Size

;  Enter Supervisor Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit
                MOV     SP, R0
                SUB     R0, R0, #SVC_Stack_Size

;  Enter User Mode and set its Stack Pointer
                MSR     CPSR_c, #Mode_USR
                IF      :DEF:__MICROLIB

                EXPORT __initial_sp
```

```
                ELSE

                MOV     SP, R0
                SUB     SL, SP, #USR_Stack_Size

                ENDIF


; Enter the C code

                IMPORT  __main
                LDR     R0, =__main
                BX      R0


                IF      :DEF:__MICROLIB

                EXPORT  __heap_base
                EXPORT  __heap_limit

                ELSE
; User Initial Stack & Heap
                AREA    |.text|, CODE, READONLY

                IMPORT  __use_two_region_memory
                EXPORT  __user_initial_stackheap
__user_initial_stackheap

                LDR     R0, =  Heap_Mem
                LDR     R1, =(Stack_Mem + USR_Stack_Size)
                LDR     R2, = (Heap_Mem +      Heap_Size)
                LDR     R3, = Stack_Mem
                BX      LR
                ENDIF
                END
```

MODULUL ADC
Fişier : global_adc_init.h

```
/*
       Header pentru interfata generala de initializare a
       convertoarelor analog digitale adc0/1 ale LPC2148
       pentru interfatarea cu senzorii
*/
#include<lpc214x.h>

/* Functie de initializare a adc0 si adc1*/
void init_adc(void);
/* Functii de citire ADC functie de canal */
unsigned int read_adc0(unsigned char);
unsigned int read_adc1(unsigned char);
```

Fişier : global_adc_init.c

```
/*
       Interfata generala de initializare a convertoarelor analog
    digitale ADC0/1 ale LPC2148 pentru interfatarea cu senzorii
*/

#include "global_adc_init.h"

// Configuratia sonarelor
//  PINSEL0
//  -> bit 25:24 conf 1:1 AD1.3 pin P0.12      -> sonar1
//  -> bit 27:26 conf 1:1 AD1.4 pin P0.13      -> sonar2

//  PINSEL1
//  -> bit 11:10 conf 1:0 AD1.6 pin P0.21 -> sonar3
```

```
//  -> bit 13:12 conf 0:1 AD1.7 pin P0.22 -> sonar4
//  -> bit 25:24 conf 0:1 AD0.1 pin P0.28 -> sonar5

/* Functie de initializare a convertoarelor ADC0/1 */
void init_adc(void){
       // activarea functiei ADC pentru pini (ADC1)
       PINSEL0 |= ((1<<24) | (1<<25) | (1<<26) | (1<<27));
    // activarea functiei ADC pentru pini (ADC0)
       PINSEL1 |= ((1<<11) | (1<<12) | (1<<24));
       PINSEL1 &=~((1<<10) | (1<<13) | (1<<25));
       // power control pentru periferice
       PCONP |= ((1<<12)|(1<<20)); // activare power ADC1/0
       // resetare conversii si biti de control
       AD0CR = 0x0; // reg control ADC0
       AD1CR = 0x0; // reg control ADC1
       // setare clock conversie CLKDIV fct de VPB CLK
       // aici se considera 6 factor de divizare la 30Mhz
       // freccventa MCU 30/(6+1) = 4.2MHz <= 4.5MHz (impus)
       AD0CR |=((1<<9) | (1<<10));
       AD1CR |=((1<<9) | (1<<10));
       // dezactivare mod BURST, operare in mod controlat software
       AD0CR &=~(1<<16);
       AD1CR &=~(1<<16);
}

/* Functii globale de citire a unui canal ADC */
unsigned int read_adc0(unsigned char channel)
{
   // rezultatul conversiei
   unsigned int i;
   // initializare adc
   init_adc();
   // opreste conversiile anterioare si reseteaza canalele
   AD0CR &= ~((1<<24) | (1<<25) | (1<<26));
   // selecteaza canalul primit la intrare
   AD0CR |= (1<<channel);
   // seteaza ADC0 operational; bit PDN=1
   AD0CR |= (1<<21);
   // start conversie
   AD0CR |= (1<<24);
   do
   {
   switch(channel)   // determina canalul
   {
   case 0:
                i=AD0DR0; // preia valoarea din reg de date canal 0
                break;
   case 1:
                i=AD0DR1; // preia valoarea din reg de date canal 1
                break;
   case 2:
                i=AD0DR2; // preia valoarea din reg de date canal 2
                break;
   case 3:
                i=AD0DR3; // preia valoarea din reg de date canal 3
                break;
   case 4:
                i=AD0DR4; // preia valoarea din reg de date canal 4
                break;
   case 5:
                i=AD0DR5; // preia valoarea din reg de date canal 5
                break;
   case 6:
                i=AD0DR6; // preia valoarea din reg de date canal 6
                break;
   case 7:
                i=AD0DR7; // preia valoarea din reg de date canal 7
                break;
   default:
```

```
                break;
   } //end switch
   }
   //asteapta final conversie(bit DONE=1 in reg global de date)
   while ((AD0GDR & 0x80000000) == 0);
   // izoleaza rezultatul si revine
   return ((i >> 6) & 0x03FF);
}

unsigned int read_adc1(unsigned char channel)
{
// rezultatul conversiei
   unsigned int i;
   // initializare adc
   init_adc();
   // opreste conversiile anterioare si reseteaza canalele
   AD1CR &= ~((1<<24) | (1<<25) | (1<<26));
   // selecteaza canalul primit la intrare
   AD1CR |= (1<<channel);
   // seteaza ADC1 operational; bit PDN=1
   AD1CR |= (1<<21);
   // start conversie
   AD1CR |= (1<<24);
   do
   {
   switch(channel)   // determina canalul
   {
   case 0:
                i=AD1DR0; // preia valoarea din reg de date canal 0
                break;
   case 1:
                i=AD1DR1; // preia valoarea din reg de date canal 1
                break;
   case 2:
                i=AD1DR2; // preia valoarea din reg de date canal 2
                break;
   case 3:
                i=AD1DR3; // preia valoarea din reg de date canal 3
                break;
   case 4:
                i=AD1DR4; // preia valoarea din reg de date canal 4
                break;
   case 5:
                i=AD1DR5; // preia valoarea din reg de date canal 5
                break;
   case 6:
                i=AD1DR6; // preia valoarea din reg de date canal 6
                break;
   case 7:
                i=AD1DR7; // preia valoarea din reg de date canal 7
                break;
   default:
                break;
   } //end switch
   }
   //asteapta final conversie(bit DONE=1 in reg global de date)
   while ((AD1GDR & 0x80000000) == 0);
   // izoleaza rezultatul si revine
   return ((i >> 6) & 0x03FF);
}

MODULUL UART
Fişier : serial.h

#include<lpc214x.h>

/* Prototipurile functiilor modulului uart */

/* Variabile globale si macro */
```

```
#define OSCILLATOR_CLOCK_FREQUENCY  12000000 // 12 MHz
#define BAUD 9600
#define PCLK 30000000 // PCLK = ceas periferice 30 MHz

/* Functii specifice */
void init_uart0 (void);
void init_uart1 (void);
int putchar_uart0 (int);
int putchar_uart1 (int);
int getchar_uart0 (void);
int getchar_uart1 (void);

/* Functii auxiliare */
void init_system(void);

/* Functie de conversie un octet binar la doua caractere ascii hiniob, lonib */
void ascii_convert0(unsigned char);
void ascii_convert1(unsigned char);

/* Functii care transmit pe linia seriala datele formatate ASCII */
void uart0_send_ascii(unsigned char);
void uart1_send_ascii(unsigned char);
```

Fişier : serial.c

```
#include<lpc214x.h>
#include "serial.h"

/* Modul uart
  - initializare uart0
  - initializare uart1
  - functie citire uart0/uart1
  - functie scriere uart0/uart1
*/

 /* utile in conversia in reprezentare ASCII */
unsigned char lonib0, hinib0, lonib1, hinib1;

/* Functie de initializare a PLL */

void init_system(void){

  /* PLL Configuration Register
  - contine valorile de multiplicare si divizare pentru PLL
  - modificarile in acest registru au loc dupa o secventa de
    feed a buclei PLL
  ------------------------------------------------------------------------
  Valori biti:
   MSEL -->  00100 (bit 0:4) Multiplicare
   PSEL --> 01        (bit 6:5) Divizare
   */
  PLL0CFG = 0x24;

  /* Pentru a asigura modificarea valorilor in registrle de control si
  configurare a buclei PLL se impune o secventa de feed a buclei PLL*/
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;

   /* PLL Control Register
  - registru utilizat pentru a modifica biti de control ai PLL
  - se activeaza PLL si se conecteaza MCU la frecventa de iesire PLL
  - PLL0 este utilizata pentru a genera CLK pt procesor,
    PLL1 e dedicata USB
 ------------------------------------------------------------------------
  Valori biti :      (PLL Enable)
   PLLE --> 1 (bit 0)
  */
  PLL0CON = 0x01;
```

```
  /* secventa de feed a PLL */
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;

  /* asteapta pana cand PLL este locked pe faza */
  while(!(PLL0STAT & 0x400));

  /* PLL connect
   Valori biti :
   PLLC --> 1 (bit 1)
  */
  PLL0CON = 0x03;
  /* secventa de feed a PLL */
  PLL0FEED = 0xAA;
  PLL0FEED = 0x55;

  /* numar de cicli de ceas de acces la memoria Flash */
  /* 4 CLK */
  MAMCR = 0x2;
  MAMTIM = 0x4;
  /* divizor freceventa periferice */
  VPBDIV = 0x02; // PCLK = 30MHz
}

/* Functie de initializare uart0 */
void init_uart0(void){

   /*
   Pentru stabilirea baudrateului pentru uart0 se determina valoarea
   care va fi scrisa in registrul latch divisor LSB si MSB
   */
   int div =  PCLK/(16*BAUD);

   /* Pin Function Select Register 0  - PINSEL0
   are rolul de a controla functiile pinilor MCU
   - se activeaza liniile de TX si RX pentru uart0
   -----------------------------------------------
   Valori biti :
   P0.0 --> 01 (bit 1:0)
   P0.1 --> 01 (bit 3:2)
   */
   PINSEL0 |= 0x5;

   /* elibereaza FIFO urile */
   U0FCR=0x7;
   /* Line Control Register - U0LCR
   are rolul de a set aparametri de comunicatie pe uart
   - activeaza numarul de biti, paritate si biti de stop
   ----------------------------------------------------
   Valori biti :
   Word Length Select            --> 11 (bit 1:0) 8bit char
   Stop bit Select               --> 0  (bit 2)   1 stop bit
   Parity Select                 --> 0  (bit 3)   Paritate dezactivata
   Divisor Latch Acces Bit(DLAB) --> 1 (bit 7) pentru a accesa
   registrele latch divizor U0DLL, U0DLM */
   U0LCR = 0x83;

   /* Generarea baudrateului dorit se realizeaza prin utilizarea
   registrelor latch divizoare U0DLL, U0DLM (16 bit LSB,MSB) */

   U0DLL = div & 0xFF;
   U0DLM = (div>>8); // 9600 Baud Rate

   /* Pentru a accesa registrele U0RBR, U0THR (bufferele de date Rx/Tx)
   bitul DLAB trebuie resetat */
   U0LCR = 0x03;
}

/* Functie de initializare uart1 */
```

```
void init_uart1(void)
{
   int div =  PCLK/(16*BAUD);
   /* se activeaza liniile de TX si RX pentru uart0 */
   PINSEL0 |= 0x00050000;
   U1LCR = 0x83;
   U1DLL = div & 0xFF;
   U1DLM = (div>>8);
   U1LCR = 0x03;
}


/* Functie de scriere a unui carcater pe uart0 */
int putchar_uart0 (int ch){
        /* cat timp nu avem data in buffer asteapta */
        while(!(U0LSR & 0x20));
        /* altfel plaseaza in buffer caracterul de trimis */
             return (U0THR = ch);
}

/* Functie de scriere a unui sir de caractere pe uart0 */
void puts_uart0(char *ch)
{
   while(*ch){
             putchar_uart0(*ch++);
   }
}

/* Functie de scriere a unui caracter pe uart1 */
int putchar_uart1 (int ch){
        /* cat timp nu avem data in buffer asteapta */
        while(!(U1LSR & 0x20));
        /* altfel plaseaza in buffer caracterul de trimis */
             return (U1THR = ch);
}

/* Functie de scriere a unui sir de caractere pe uart1 */
void puts_uart1(char *ch)
{
   while(*ch){
             putchar_uart1(*ch++);
   }
}

/* Functie de citire a unui caracter primit pe uart0 */
int getchar_uart0 (void){
       /* cat timp nu avem date in registrul de receptie U0RBR asteapta */
       while(!(U0LSR & 0x01));
       /* cand avem date disponibile in buffer se preiau */
       return (U0RBR);
}

/* Functie de citire a unui caracter primit pe uart1 */
int getchar_uart1 (void){
       /* cat timp nu avem date in registrul de receptie U1RBR asteapta */
       while(!(U1LSR & 0x01));
       /* cand avem date disponibile in buffer se preiau */
       return (U1RBR);
}

/* Functii de conversie un octet binar la doua caractere ascii hiniob, lonib */
/* Valorile rezultate sunt in hexazecimal */
void ascii_convert0(unsigned char ch)
{
   hinib0=(ch&0xf0)>>4;
   lonib0=ch&0x0f;
   if(hinib0>9) hinib0=hinib0+'7';
   else hinib0=hinib0+'0';
   if(lonib0>9) lonib0=lonib0+'7';
```

```
   else lonib0=lonib0+'0';

}

void ascii_convert1(unsigned char ch)
{
   hinib1=(ch&0xf0)>>4;
   lonib1=ch&0x0f;
   if(hinib1>9) hinib1=hinib1+'7';
   else hinib1=hinib1+'0';
   if(lonib1>9) lonib1=lonib1+'7';
   else lonib1=lonib1+'0';

}


/* Functii care transmit pe linia seriala datele formatate ASCII */
void uart0_send_ascii(unsigned char ch)
{
    ascii_convert0(ch);
    putchar_uart0(hinib0);
    putchar_uart0(lonib0);

}
void uart1_send_ascii(unsigned char ch)
{
    ascii_convert1(ch);
    putchar_uart1(hinib1);
    putchar_uart1(lonib1);
}
```

MODULUL GPS

Fişier : gps_odometry.h

```
#include<lpc214x.h>

/* Prototipurile functiilor */

/* Functie care parseaza pachetele GPS (NMEA) receptionate */
int parse_gps_packet(void);
/* Functie de conversie a doi octeti in int */
int bytes2int(unsigned char, unsigned char);
/* Functie care preia numele (tipul) unui pachet */
void get_gps_packet_name(void);
/* Functie de comparare a doua stringuri util in filtrarea pachetelor receptionate
*/
int compare_strings(unsigned char buffer1[], unsigned char buffer2[]);
/* Functii de extragere a informatiei de latitudine din pachetele GPS */
void get_latitude_degrees(void);
void get_latitude_minutes(void);
void get_latitude_seconds(void);
/* Functii de extragere a informatiei de longitudine din pachetele GPS */
void get_longitude_degrees(void);
void get_longitude_minutes(void);
void get_longitude_seconds(void);
/* Functii de conversie a informatiei de pozitie in plan din informatia GPS */
long convert_long(void);
long convert_lat(void);
/* Functie care seteaza o pozitie de referinta pentru miscarea in plan */
void save_origin(void);
long get_y_position(void);
long get_x_position(void);
```

Fişier : gps_odometry.c

```
#include<lpc214x.h>
#include "gps_odometry.h"
#include <math.h>
```

```
char rxbuf[256];    // bufferul RX
char txbuf[256];    // bufferul TX
char rxdata;        // caracterul primit in bufferul RX
int idx;     // pointer catre elementul curent din buffer RX
int idxsav;         // aux pentru retine pointer catre buffer RX
int rbfull;     // var care retine starea bufferului de receptie

unsigned char packet_name[6];                                   // antet nume pachet
/*  Pachetele alese pentru extragerea informatiei sunt GPGLL (Geographic Position
Latitude/Longitude) */
unsigned char name1[6]={'G','P','G','L','L',0};      //  necesar  filtrarii  (doar
pachete cu acest header)

/* informatie GPS (NMEA) */
int latitude_degrees;
int latitude_minutes;
double latitude_seconds;

int longitude_degrees;
int longitude_minutes;
double longitude_seconds;

/* coordonatele in plan */
long x_origin, y_origin;
long current_x, current_y;

/*
   Functii pentru extragerea informatiei de pozitie din datele GPS
   necesare pentru calculul odometriei robotului si pentru a realiza
   corectia traiectoriei robotului.
*/

/* Functie care parseaza pachetele GPS (NMEA) receptionate */
int parse_gps_packet(void){
  int a;
  get_gps_packet_name();
  a=compare_strings(packet_name, name1);
    if(a==1)
    {
    get_latitude_degrees();
    get_latitude_minutes();
    get_latitude_seconds();
    get_longitude_degrees();
    get_longitude_minutes();
    get_longitude_seconds();
    return(1);
    }
  else return(0);
}


/* Functie de conversie a doi octeti in int */
int bytes2int(unsigned char ch1, unsigned char ch2){
   unsigned int val;
   ch1=ch1&0x0F;
   ch2=ch2&0x0F;
   val=ch1*10+ch2;;
   return(val);
}


/* Functie care preia numele (tipul) unui pachet */
void get_gps_packet_name(void){
  int i;
  for(i=1;i<6;i++) packet_name[i-1] = rxbuf[i];
  packet_name[i]=0;
}
```

```
/* Functie de comparare a doua stringuri util in filtrarea pachetelor receptionate
*/
int compare_strings(unsigned char buffer1[], unsigned char buffer2[]){
  int i;
  i=0;
  while(buffer1[i]!=0)
    {
    if(buffer1[i]!=buffer2[i]) return(0);
    else i++;
    }
  return(1);
}


/* Functii de extragere a informatiei de latitudine din pachetele GPS */
void get_latitude_degrees(void){
       latitude_degrees=bytes2int(rxbuf[7],rxbuf[8]);
}

void get_latitude_minutes(void){
       latitude_minutes=bytes2int(rxbuf[9],rxbuf[10]);
}

void get_latitude_seconds(void){
       int seconds;
       int decimals;

       seconds=bytes2int(rxbuf[12],rxbuf[13]);
       decimals=bytes2int(rxbuf[14],rxbuf[15]);
       latitude_seconds=(double)seconds+(double)decimals/100.0;
}


/* Functii de extragere a informatiei de longitudine din pachetele GPS */
void get_longitude_degrees(void){
       longitude_degrees=bytes2int(rxbuf[22],rxbuf[23]);
       if(rxbuf[21]=='1') longitude_degrees=longitude_degrees+100;
}

void get_longitude_minutes(void){
       longitude_minutes=bytes2int(rxbuf[24],rxbuf[25]);
}

void get_longitude_seconds(void){
       int seconds;
       int decimals;

       seconds=bytes2int(rxbuf[27],rxbuf[28]);
       decimals=bytes2int(rxbuf[29],rxbuf[30]);
       longitude_seconds=(double)seconds+(double)decimals/100.0;
}


/* Functii de conversie a informatiei de pozitie in plan din informatia GPS */
/* Conversie din informatia de latitudine */
long convert_lat(void){
   double temp;
   long i;

   temp=latitude_degrees*111132+(double)latitude_minutes*(111132.0/60.0);
   temp=temp+latitude_seconds*(111132.0/3600.0);
   i=floor(temp);
   return(i);
}


/* Conversie din informatia de longitudine */
```

```
long convert_long(void){
   double temp;
   long i;

   temp=longitude_degrees*78847+(double)longitude_minutes*(78847.0/60.0);
   temp=temp+longitude_seconds*(78847.0/3600.0);
   i=floor(temp);
   return(i);
}


/* Functie care seteaza o pozitie de referinta pentru miscarea in plan */
void save_origin(void){
   x_origin=convert_long();
   y_origin=convert_lat();
}

/* Functii care returneaza pozitia in plan */
long get_x_position(void){
              current_x=convert_long();
              current_x=current_x-x_origin;
       return current_x;
}

long get_y_position(void){
              current_y=convert_lat();
              current_y=current_y-y_origin;
       return current_y;
}
```

MODUL SONARE

Fişier : sinar_interface.h

```
/*
       interfata de conectare a senzorilor ultrasonici pentru
       detectia obstacolelor si maparea topologica a mediului
*/

#include "global_adc_init.h"

/* Functie de citire a unui sonar */
unsigned int read_sonar(int sonar);
/* Functie de activarea a operarii in daisy-chaining pentru sonare */
void enable_sonar_chain(void);
/* Functie care introduce o intarziere la achizitia datelor ADC */
void delay_us(int);
/* Functie de strobe pentru activarea citirilor de la sonare */
void sonar_strobe(void);
/* Functie care formateaza datele primite de la sonare si
 * trimite un pachet pe linia seriala.
 */
void format_sonar_data_and_send(unsigned int data);
```

Fişier : sonar_interface.c

```
/*
       Interfata de conectare a senzorilor ultrasonici pentru
       detectia obstacolelor si maparea topologica a mediului
*/

#include "sonar_interface.h"
#include "serial.h"

/*
 Determinarea iesirii pentru Uin = 4V

 Uin=5V -> out=~9.8mV/in. si Uin=3.3V -> out=~6.4mV/in.
 Uin=4.3V -> out = ?

 (x-6.4)/(9.8-6.4) = (4.3-3.3)/(5-3.3)=> x = 8.4mV/in =  3.36mV/cm */
```

```
#define SCALE 3.36f   // 3.36mV
#define V_SUPPLY 4300 // 4300mV
#define MAX_ADC 1024  // ADC 10 biti
/*
       Configuratia utilizata pt sonare :

  PINSEL0
  -> bit 25:24 conf 1:1 AD1.3 pin P0.12 -> sonar1
  -> bit 27:26 conf 1:1 AD1.4 pin P0.13 -> sonar2

  PINSEL1
//  -> bit 11:10 conf 1:0 AD1.6 pin P0.21 -> sonar3
//  -> bit 13:12 conf 0:1 AD1.7 pin P0.22 -> sonar4
//  -> bit 25:24 conf 0:1 AD0.1 pin P0.28 -> sonar5
*/

/*Functie de citire a ADC conectate la sonare */
unsigned int read_sonar(int sonar){
       /* valoarea tensiunii de la iesirea sonarului */
       unsigned int val;
       /* citeste valoarea corespunzatoare sonarului */
       switch(sonar){
                     case 1: val = read_adc1(3);
                                   break;
                     case 2: val = read_adc1(4);
                                   break;
                     case 3: val = read_adc1(6);
                                   break;
                     case 4: val = read_adc1(7);
                                   break;
                     case 5: val = read_adc0(1);
                                   break;
       }
       /* returneaza distanta in cm pana la primul obstacol */
       return (unsigned int)(((val*V_SUPPLY)/MAX_ADC)/SCALE);
}

/* Functie de activarea a modului de operare in chaining pentru
   sonare.
*/
void enable_sonar_chain(void){
         // seteaza pinii pentru BW (bandwidth)
         IO1DIR |= ((1<<16) | (1<<17) | (1<<18) | (1<<19) | (1<<20) | (1<<21));
         // BW este mentinut High (1)
         IO1SET |= ((1<<16) | (1<<17) | (1<<18) | (1<<19) | (1<<20));
         // conecteaza RX la 1 logic pt 50 us
         IO1SET |= (1<<21);
         delay_us(50);
      IO1CLR |= (1<<21);
         // comuta pinul RX in high-Z0
         IO1DIR &=~(1<<21);
}

/* Functie de strobe pentru activarea citirilor de la sonare */
void sonar_strobe(void){
          IO1SET |= (1<<21);
          delay_us(50);
       IO1CLR |= (1<<21);
          delay_us(50);
}

/* Functie care introduce intraziere la achizitia datelor ADC */
void delay_us(int x){
   int a,b;
   for(a=0;a<x;a++)
      for(b=0;b<4;b++);
}

/* Functie care formateaza datele primite de la sonare si
```

```
 * trimite un pachet pe linia seriala.
 */
void format_sonar_data_and_send(unsigned int data){
       // octeti ascii corespunzatori valorii citite
       unsigned char data_l, data_h;
       // se extrage octetul mai semnificativ al valorii
       data_h = (data&0x300)>>8;
       // se extrage octetul mai putin semnificativ
       data_l = data&0xff;
       // trimite octetii pe linia seriala dupa conversia ascii
       uart0_send_ascii(data_h);
       uart0_send_ascii(data_l);
       // separator
       putchar_uart0(',');
}
```

MODUL ACCELEROMETRU

Fişier : accelerometer_interface.h

```
 /*
       Interfata pentru accelerometru 3D. Informatia de acceleratie
       este integrata in informatie de positie utila la fuziunea
       senzorilor pentru pozitionarea globala a robotului mobil
*/

#include "global_adc_init.h"

/* Functie de citire a axei X */
unsigned int read_accel_x(void);
/* Functie de citire a axei Y */
unsigned int read_accel_y(void);
```

Fişier : accelerometer_interface.c

```
/*
       Interfata pentru accelerometru 3D. Informatia de acceleratie
       este integrata in informatie de positie utila la fuziunea
       senzorilor pentru pozitionarea globala a robotului mobil
*/

#include "accelerometer_interface.h"
// TODO Revizuire porul ADC pentru utilizare ACCelerometru

/*
       Configuratia utilizata pt accelerometru :
  -> bit 13:12 conf 0:1 AD1.7 pin P0.22 -> Accelerometru axa X
  -> bit 11:10 conf 1:0 AD1.6 pin P0.21 -> Accelerometru axa Y
*/

/* Functie de citire a axei X */
unsigned int read_accel_x(void){
     // valoarea tensiunii de la iesirea X a accelerometrului
        unsigned int value;
       // activare functie ADC pt P0.22 in PINSEL1
       PINSEL1 |= (1<<12);
       PINSEL1 &=~(1<<13);
     value = read_adc1(7);
        // returneaza valoarea citita
        return (value);
}

/* Functie de citire a axei Y */
unsigned int read_accel_y(void){
     // valoarea tensiunii de la iesirea Y a accelerometrului
        unsigned int value;
       // activare functie ADC pt P0.22 in PINSEL1
       PINSEL1 |= (1<<11);
```

```
       PINSEL1 &=~(1<<10);
      value = read_adc1(6);
        // returneaza valoarea citita
        return (value);
}
```

8.2 Codul sursă Matlab pentru aplicaţia SLAM de mapare offline (Listing, CD-ROM)

```
MODUL MODEL SONAR

Fişier : sonar_model.m

%%
% Aplicatie SLAM offline pt ARTEMIC
%
% Functie care implementeaza modelul sonarelor
%%
function model = sonar_model(sonars_readings, heading)
    % jumatate din unghiul de deschidere a conului
    beta = 7;
    % range maxim sonar
    range = 26;
    % scare de reprezentare a hartii
    scale = 35;
    % toleranta in reprezentarea hartii
    tolerance = 1;
    % valoare utilizata pentru definirea starii initiale
    percent_occupied = 0.98;
    % dimensiunea modelului sonarului pentru harta
    % pe termen scurt
    model_size = 150;
    % initializarea modelului
    model = zeros(model_size, model_size);
    model(:,:) = NaN;
    % corectie heading robot
    a = -heading/180*pi;
    T = [cos(a) -sin(a);sin(a) cos(a)];
   % pozitionarea sonarelor pe X Y si orientare
   % pe partea frontala a robotului
     sonars_setup=[  69      136     90
                    114      119     50
                    148      78      30
                    166      27      10
                    166     -27     -10
                    148     -78     -30
                    114    -119     -50
                     69    -136     -90 ];
     % se elimina senzorii out of range
     tmp = [];
     tmp2 = [];
     for si=1:length(sonars_setup)
         % distanta maxima de activitate a sonarului
         % in reprezentarea hartii
         if(sonars_readings(si)<2000)
            tmp = [tmp; sonars_setup(si,:)];
            tmp2 = [tmp2; sonars_readings(si)];
         end
     end
     % verificari suplimentare de dimensiune
     if(size(tmp,1)==0)
         return
     end
     % formarea modelului sonarului
     sonars_setup = tmp;
```

```
     sonars_readings = tmp2./scale;
     sonars_setup(:,1:2) = sonars_setup(:,1:2)*T./scale;
     sonars_setup(:,3) = sonars_setup(:,3)+heading;
     for yi=1:model_size
         for xi=1:model_size
             for si=1:size(sonars_setup, 1)
                 % calcul pozitie
                 x = xi-model_size/2;
                 y = yi-model_size/2;
                 r = ((x-sonars_setup(si,1))^2 + (y-sonars_setup(si,2))^2)^0.5;
                 b = abs(atan2(y-sonars_setup(si,2), x-sonars_setup(si,1))/pi*180-
sonars_setup(si,3));
                 while(b>180)
                     b = abs(b-360);
                 end
                 % verificarile dimensiunilor pentru afisare
                 if(b>beta || r>sonars_readings(si)+tolerance/2)
                     continue
                 end
                 % calculul probabilitatilor ca o celula a matricii de
                 % ocupare este ocupata tinand cont de citirea sonarului
                 if(abs(r-sonars_readings(si))<=tolerance)
                     probability = ((((range - r) / range) + ((beta -
b)/beta))/2)*percent_occupied;
                     probability = .5*probability+.5;
                 else
                     probability = (((range - r)/range)+((beta - b)/beta))/2;
                     probability = 1-(.5*probability+.5);
                 end
                 % formeaza modelul sonarului
                 model(xi,yi)=probability;
             end
         end
     end
end
```

MODUL FUZIUNE BAYES

Fişier : bayes_fusion.m

```
%%
% Aplicatie SLAM offline pt ARTEMIC
%
% Functia de fuziune a senzorilor utilizand regula lui Bayes
%%
function world = bayes_fusion(world, model, offset)
    prev = world(offset(1)+1:offset(1)+length(model),...
                 offset(2)+1:offset(2)+length(model));
    idx = find(~isnan(model));
    new = prev;
    new(idx)=(model(idx).*prev(idx))./(model(idx).* ...
    prev(idx)+(1-model(idx)).*(1-prev(idx)));
    world(offset(1)+1:offset(1)+length(model),...
          offset(2)+1:offset(2)+length(model)) = new;
end
```

MODUL DE CALCUL SCOR VALIDARE HARTA (HARTA TERMEN SCURT)

Fişier : compute_score.m

```
%%
% Aplicatie SLAM offline pt ARTEMIC
%
% Functia de estimare a potrivirii hartii pe termen scurt cu harta pe
% termen lung; scorul reda o dimensionare a rezultatului comparatiei
%%
function score = compute_score(world, model, offset)
    world = world(offset(1)+1:offset(1)+length(model),...
```

```
                offset(2)+1:offset(2)+length(model));
    if(isempty(find((world~=.5).*(model~=.5), 1)))
        score = 0;
        return;
    end
    score = sum(sum(abs(world - model)));
end
```

MODUL DE GENERARE POZIŢII

Fişier : pose_generator.m

```
%%
% Aplicatie SLAM offline pt ARTEMIC
%
% Functie care returneaza pozitiile posibile
%%
function poses = pose_generator(~)
poses = [0 0;-1 -1;1 1;1 -1;-1 1;0 -1;0 1;-1 0;1 0];
return
end
```

MODUL CLIENT SLAM
(implementarea algoritmului si programul principal)

Fişier : SLAM_client.m

```
% Aplicatie SLAM offline pt ARTEMIC
%
% Programul principal de simulare a SLAM
%%
% elibereaza mediul
clear all
clc
% valori scalare pentru afisare
scale = 50; % 20 pt camera | 50 pt hol
world_size =800; % 600 pt camera | 800 pt hol
world=zeros(world_size,world_size);
world(:,:)=0.5;
% grad suprapunere harti
overlap=0;
% ajustare heading (a se tine cont de pozitia de start)
degrees=[0 0 0 0 0 0 0];
% incarca fisierul cu datele de la senzori de pe disc
load robot_state saved*;
% alocarea variabilelor pentru valorile citite
readings=saved_readings;
pose=saved_pose;
% offset harti
off=1;
% prealocare vector pozitie
pos=[];

while(off<length(readings)-overlap)
    % initializari
    stm_size=0;
    num=0;
    % numarul de esantioane citite de le sonare (min45)
    while(num<45 && off+stm_size<length(readings))
        num=num+sum(readings(off+stm_size,:)<6000);
        stm_size = stm_size+1;
    end
    % prealocari
    rpos=[];
    rstm=[];
    rposes=[];
    rscore=[];
    rdegrees=[];
    for k=1:length(degrees)
```

```
        % porneste cu o harta initiala goala
        stm=zeros(world_size, world_size);
        % niciun punct nu e necunoscut
        stm(:,:)=0.5;
        for j=1:stm_size
            % calculul pozitiei
            pos=ceil((pose(off+j,:)./scale)+(world_size./2)-40);
            % genereaza modelul sonarului
            model=sonar_model(readings(off+j,:),pose(off+j,3)+degrees(k));
            % adauga in harta temporara (pe termen scurt)
            stm=bayes_fusion(stm, model, pos);
        end
        % limita impusa afisare
        margin= 5;%15;
        pos=ceil((pose(off+1,:)./scale)+(world_size./2)-40-margin);
        % formeaza harta pe termen scurt
        stm=stm(pos(1)+1:pos(1)+length(model)+2*margin,
pos(2)+1:pos(2)+length(model)+2*margin);
        % functia pose_generator() returneaza o matrice a posibilelor
        % pozitii pe care le-ar putea avea robotul, in functie depozitia
        % actuala
        % functia compute_score() pentru a vedea in care din aceste pozitii se
        % integreaza cel mai bine
        poses=pose_generator(pose(off+1,:));
        for(l=1:size(poses,1))
            pos2=pos(1:2)+poses(1,:);
            rscore=[rscore compute_score(world, stm, pos2)];
            rpos=[rpos;pos2];
            rstm(:,:,length(rscore)) = stm;
            rposes=[rposes;poses(1,:)];
            rdegrees=[rdegrees degrees(k)];
        end
    end
    % functia MIN returneaza locatia cea mai probabila
    [score, idx]=min(rscore);
    stm=rstm(:,:,idx);
    pos=rpos(idx,:);
    % inregistrarea (validarea) hartii porvizorii in (cu) harta globala
    world=bayes_fusion(world, stm, pos);
    left=off+stm_size-overlap:length(readings);
    a=-rdegrees(idx)/180*pi;
    T=[cos(a) -sin(a);sin(a) cos(a)];
    pos=[pose(left,1)-pose(off+1,1) pose(left,2)-pose(off+1,2)];
    pos=pos*T;
    pose(left,1:2)=[pos(:,1)+pose(off+1,1) pos(:,2)+pose(off+1,2)];
    pose(left,1) = pose(left,1)+(scale.*rposes(idx,1));
    pose(left,2) = pose(left,2)+(scale.*rposes(idx,2));
    pose(left,3) = pose(left,3)+rdegrees(idx);
    % afisarea hartii temporare (pe termen scurt, in timpul generarii)
    figure(1);
    colormap(gray);
    h = imrotate(world, 180);
    imagesc(h, [0 1]);
    drawnow;
    zoom on
    zoom(1.5)
    off=off+stm_size-overlap;
end
% harta globala ajustata
figure(2);
colormap(gray);
world(world<.7)=0;
world(world>=.7)=1;
h = imrotate(world, 180);
imagesc(h, [0 1]);
zoom on
zoom(1.5)
```