UN CONTROLLER AUTO-EVOLUTIV PENTRU UN ROBOT FIZIC – CONTROLER BAZAT PE UN NOU ALGORITM DE EVITARE A OBSTACOLELOR

DAN-MARIUS DOBREA, ADRIANA SÎRBU, MONICA-CLAUDIA DOBREA

Rezumat: Una dintre problemele majore întâlnite în construcția de roboți (care urmează să opereze în medii reale) este și abilitatea acestora de a face față șirului continuu și, adesea neașteptat, de evenimente ce au loc în timp real. În cele ce urmează, vom prezenta un robot reactiv real care este capabil să evolueze singur astfel încât, în final, să ajungă să evite obstacolele într-un mod reflex. Comportamentul de evitare a obstacolelor este unul testat în medii reale și el se bazează, în principal, pe *baza de cunoștințe* extrasă într-o etapă anterioară, în cadrul unui proces de învățare simulat. Mai exact, pentru a obține *baza de date ale mișcărilor* pentru robotul real s-a implementat mai întâi, în mediul de simulare MobotSim 1.0.03, noul algoritm genetic (**GA**) de evitare a obstacolelor pe care-l vom descrie mai jos. Mediul de simulare folosit este un mediu de simulare configurabil, dedicat roboților cu roți și cu acționare diferențială. Baza de cunoștințe extrasă cuprinde *setul principal de reguli* care mapează informația de la senzorii robotului direct în comenzile corespunzătoare pentru motoare. În mediul real, robotul poate ajunge, însă, și în situații de mediu în care nici una dintre regulile extrase prin simulare nu se poate aplica[1]. În astfel de situații, pentru a evita o iminentă coliziune cu obstacolele din jur, robotul va recurge la extragerea de noi reguli prin aplicarea, online și în mediu real de această dată, a aceluiași algoritm GA folosit în simulare. În concluzie, ceea ce aduce nou lucrarea de față este, pe de o parte, algoritmul nou de învățare bazat pe GA iar, pe de altă parte, este abordarea mult facilitată de acest nou algoritm – abordare ce constă, mai întâi, în implementarea algoritmului GA într-un mediu simulat, finalizată cu extragerea unui set principal de reguli de mișcare, set de reguli care, mai apoi, este ulterior implementat alături de același algoritm GA și pe robotul real. Consecințele directe și avantajele nete ce rezultă din această abordare se regăsesc, în principal: **1**) la nivelul timpului (sensibil mai redus) petrecut de sistemul robotic în vederea extragerii setului principal de reguli de mișcare, **2**) în posibilitatea mai facilă de a urmări și îmbunătăți (în mediul de simulare) algoritmul de învățare, precum și **3**) în posibilitatea de a combina, *în mediul real*, exploatarea acestui set de reguli (extrase prin simulare) cu capacitatea robotului de a extrage, cu ajutorul aceluiași algoritm GA, noi reguli de mișcare corespunzătoare condițiilor noi (nemaiîntâlnite) din mediul real.

1. Introducere

În ultimii 20 de ani tehnicile soft de calcul au primit un suport tot mai puternic ca o consecință directă a progreselor înregistrate în industria chip-urilor – progrese care, în principal, au impulsionat piața procesoarelor (de exemplu, vorbim azi de frecvențe de lucru mai mari, de arhitecturi noi, de core-uri multiple ș.a.m.d.). Dintre tehnicile soft care se cunosc la ora actuală,

[1] Aceasta este echivalent cu a spune că noul context de mediu nu este similar (într-o măsură mai mică sau mai mare) cu nici una dintre situațiile de mediu învățate anterior de către robot.

cele care lucrează, în principal, cu date imprecise și incerte sunt foarte potrivite aplicațiilor ce includ medii reale (acestea din urmă având aceleași caracteristici).

Algoritmii de tip fuzzy-logic (FL), de tip rețele neuronale artificiale (RNA) și, respectiv, algoritmii genetici sunt tehnici soft larg utilizate în aplicații de mediu real așa cum este și cazul aplicațiilor de tip control robotic în timp real. Algoritmii genetici, spre exemplu, sunt extensiv utilizați pentru a dezvolta controllere pentru roboți, mai ales sub formele lor hibride – vorbim aici, în special, de controllere GA-fuzzy sau GA-RNA. În aceste abordări hibride algoritmul GA este cel care ajustează parametrii controller-ului robotic. Valori diferite ale acestor parametri (obținute în timpul unui proces evolutiv) generează comportamente diferite ale robotului. În cadrul procesului de evoluție genetică roboții cu cele mai bune performanțe sunt și cei care au șanse mai mari să-și transmit caracteristicile lor către moștenitori. La sfârșitul procesului, după câteva iterații, se obține un robot cu performanțe bune. Există la ora actuală numeroase aplicații de succes în care un algoritm GA găsește parametrii optimi ai unor controllere fuzzy-logic **[Tan, 2011]**, **[Martíneza, 2009]**. Mai mult chiar, în implementarea controllerelor fuzzy dedicate aplicațiilor de tip evitare a obstacolelor, se folosesc mai nou și alte tipuri de algoritmi cu rol în îmbunătățirea performanțelor tehnicilor GA; un astfel de exemplu este și algoritmul de tip "colonie de furnici" [Chiou, 2010].

În robotică, una dintre principalele aplicații ale algoritmilor GA constă în găsirea căii optime care ar trebui să fie urmată de către un sistem robotic în vederea atingerii unui anumit scop particular **[Hosseinzadeh, 2010]**, **[Repoussis, 2009]**. În general, putem spune că am obținut un astfel de algoritm de navigare eficient pentru un robot autonom doar în momentul în care acesta din urmă face dovada că este capabil simultan: (**a**) să își localizeze poziția sa curentă, (**b**) să execute (independent de un operator uman) o mișcare locală lipsită de orice coliziune cu obstacolele din jur și (**c**) să găsească o cale globală optimă care să-l ducă către scopul său final.

Dintre toate aceste trei obiective, doar unul singur – și anume, abilitatea (îmbunătățită a) robotului de a executa o traiectorie a mișcării lipsită de orice coliziune locală cu obstacolele din mediu – va face obiectul cercetării ce se va prezenta în continuare. Abordarea acestui obiectiv se va realiza în doi pași distincți. Într-o primă etapă se va realiza implementarea noului algoritm GA într-un mediu de simulare dedicat sistemelor robotice cu roți acționate diferențial. Scopul acestei etape este acela de a extrage, într-o primă fază, baza-nucleu de cunoștințe ale mișcării robotului (setul principal de reguli de mișcare). În cea de a doua etapă (finală), se va testa pe un robot real eficiența strategiilor dezvoltate de către controller în mediul de simulare (setul de reguli extras) și, simultan cu aceasta, robotul va avea deschisă și posibilitatea de a evolua în continuare (își va putea lărgi setul de reguli de mișcare) – de această dată, în timp real – datorită aceluiași algoritm GA implementat pe platforma robotică reală.

Conceptul de bază al acestui nou controller cu auto-evoluție este unul similar abordării de tip fuzzy (de exemplu, vorbim de existența unui set de reguli). Spre deosebire, însă, de abordarea fuzzy, în cazul nostru regulile se extrag folosind un GA și nu folosindu-ne de cunoștințele unor experți umani.

Principala aplicație a acestui nou controller GA este una în domeniul roboților comandați de la distanță. În astfel de aplicații, controller-ul GA este cel care ajustează, în mod autonom, comanda de la distanță ori de câte ori comanda operatorului uman pune robotul în pericol de coliziune iminentă.

2. Evitarea obstacolelor

Pentru roboții autonomi, incertitudinea mediilor reale este cea mai mare provocare. În acest sens, controller-ul unui sistem robotic autonom trebuie să facă față și să se descurce cu un număr mare de factori cum ar fi caracteristicile mecanice și electrice ale sistemului robotic, precum și complexitatea mediului (de exemplu, variația parametrilor spațiali și temporali, neliniaritățile și incertitudinea manifestate prin dinamici haotice și aleatorii ale obiectelor din mediu etc.). În consecință, fiecare robot autonom trebuie să fie capabil să exploreze, în mod independent, mediul său, să se redreseze în caz de eșec, să rezolve noi probleme de evitare a obstacolelor și, mai presus de toate, să facă toate aceste lucruri în timp real.

În cazul nostru, cu ajutorul unui algoritm genetic robotul învață, se adaptează și generalizează cunoștințele legate atât de el cât și de mediu, devenind, astfel, capabil să execute o mișcare fără coliziuni. Procesele de învățare și de adaptare sunt similare celor întâlnite în cazul subiecților umani care reacționează și învață experimentând.

3. Sistemul robotic

Sistemul robotic are trei grade de libertate, fiind capabil să execute două mișcări de bază: rotirea și translarea. În partea din spate a robotului sunt dispuse două roți comandate de două motoare cu acțiune diferențială, vezi **Figura 1**.

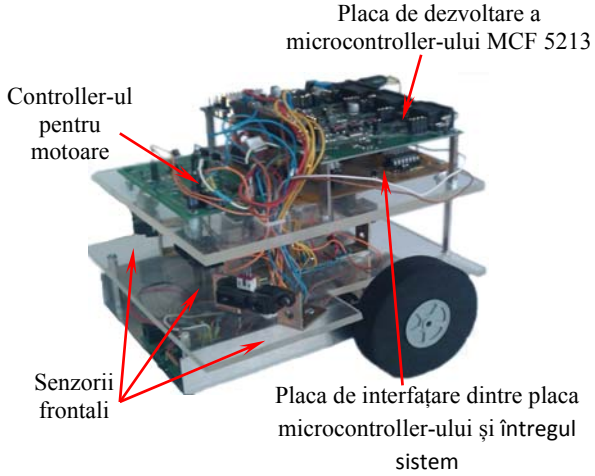

Figura 1: O imagine a implementării hardware a sistemului robotic

Cele două motoare de curent continuu sunt, la rândul lor, controlate prin intermediul unui microcontroller. O roată neacționată și plasată în partea frontală, cu dispunere centrală, asigură stabilitatea platformei robotice. Robotul are o viteză maximă medie de 0.3 m/s.

Sistemul robotic este dotat cu patru senzori infraroșii, IR, de proximitate – GP2D120XJ00F. Senzorii IR au o distanță activă ce acoperă intervalul [4, 30] cm. Trei dintre senzorii IR sunt dispuși frontal și ei supraveghează partea stângă, centrală și, respectiv, partea dreaptă a zonei frontale a mediului din vecinătatea robotului. Cel de-al patrulea sensor IR este plasat în partea din spate, în poziție centrală (vezi **Figura 1**). Senzorii returnează o valoare a tensiunii ce este proporțională cu distanța până la obstacolele detectate. Una dintre problemele principale ale senzorilor IR o constituie faptul că funcția caracteristică a acestora este una neliniară. Pentru a depăși acest neajuns, într-o primă etapă, de preprocesare, caracteristica senzorilor a fost liniarizată cu ajutorul unui algoritm de regresie **[Dobrea, 2010]**.

Sistemul robotic este controlat de un microcontroller puternic, pe 32 de biți, MCF5213 în care este implementat controller-ul GA cu auto-organizare. Microcontroller-ul aparține familiei ColdFire™ și el are la bază o arhitectură RISC ce cuprinde un număr mare de echipamente periferice cum ar fi 8 canale PWM, patru timer-e pe 32 de biți cu capacitate de cerere DMA, 8 canale ADC, 3 UARTs, 1 CAN etc.

Robotul este proiectat să învețe, într-o manieră adaptivă, folosindu-se pentru aceasta de informația primită de la senzori. Viteza roților este actualizată cu o frecvență de 3.3 Hz, prin intermediul celor două canale PWM ce comandă două punți H. Programul soft este dezvoltat în limbaj C, utilizând mediul de dezvoltare integrat CodeWarrier. Mai mult, pentru a reduce suplimentar încărcarea computațională s-a făcut uz de sub-sistemul de întreruperi al microcontroller-ului.

4. Algoritmul genetic

Algoritmii genetici, GA, sunt tehnici de căutare globală și de optimizare inspirați din mecanismul selecției naturale care există în natură.

Un GA se bazează pe o populație de soluții candidate numite cromozomi. Fiecare cromozom este evaluat și indexat corespunzător de către o funcție de evaluare numită *fitness*. În particular, funcția fitness oferă informația cu privire la cât de bun este fiecare cromozom în parte. Evoluția unui GA de la o generație la alta implică trei pași distincți: **1**) evaluarea funcției fitness, **2**) selecția cromozomului și **3**) construirea următoarei generații (bazată, în principal, pe operatori GA cum ar fi reproducerea și mutația). Scopul celui de-al treilea pas (crearea unei noi generații) este acela de îmbunătățire a valorii funcției fitness pentru populația evaluată.

5. Algoritmul de evitare a obstacolelor

Algoritmul de evitare a obstacolelor are ca principal nucleu un algoritm GA. Comenzile pentru cele două motoare, stâng și drept, le codificăm într-un singur cromozom, folosind pentru aceasta două caractere cu semn: $c_j = \{left\_eng_j, right\_eng_j\}$, unde j codifică al j-lea cromozom. În consecință, fiecare cromozom este reprezentat doar pe 16 biți. Fiecare dintre cele două secțiuni ale cromozomului – reprezentată pe 8 biți și corespunzând comenzii pentru unul dintre cele două motoare – poate lua valori doar în intervalul [-128,+127]. Semnificația atribuită acestor valori

este următoarea: o valoare de **+127** denotă putere maximă înainte a motorului, **-128** reprezintă putere maximă înapoi a motorului iar **0** corespunde unei comenzi stop a motorului.

După achiziționare, valorile, s_i, ale senzorilor IR sunt liniarizate **[Dobrea, 2010]** și, apoi, normalizate în intervalul [0,0.9], cu 0 denotând lipsa oricărui obstacol în vecinătatea imediată a robotului și 0.9 semnificând coliziune; mai mult, datorită zgomotului de la nivelul senzorilor, fiecare valoare a acestora mai mică de 0.02 este înlocuită cu valoarea 0.

Forma analitică a funcției de fitness, $f(\cdot)$, este cea prezentată în relația (1). Relația (1) este una ce respectă paradigma fundamentală care definește, în general, funcția de fitness în domeniul algoritmilor genetici. Această paradigmă fundamentală prevede că funcția de fitness ia cea mai mică valoare (zero, în cazul nostru) doar atunci când cromozomul rezolvă cu succes problema; în caz contrar, funcția fitness măsoară capacitatea fiecărui cromozom de a rezolva mai mult sau mai puțin problema, fiind, în cazul nostru, într-o relație direct proporțională cu distanța medie până la obstacolele din vecinătate.

$$f_j\left(c_j[n]\right) = \frac{1}{4}\sum_{i=1}^{4} s_i[n] \qquad (1)$$

După fiecare mișcare ce durează 300 ms, funcția de fitness este calculată cu ajutorul cromozomului c_j ce conține informația aferentă comenzilor date motoarelor. O mișcare fără coliziuni, specificată printr-un cromozom c_j, este caracterizată prin valori de zero pentru toți cei 4 senzori (reamintim că aceasta corespunde situației ideale în care, în vecinătatea robotului nu se semnalează nici un obstacol); într-o astfel de situație funcția de fitness, $f(c_j[n])$, ia de asemenea o valoare egală cu zero.

Într-o **primă etapă** (ce are loc în mediul de simulare), extragerea (în mod adaptiv) a unui set de reguli de mișcare se realizează printr-o interacțiune continuă a robotului cu mediul său simulat, fără intervenție umană și doar urmând o comandă globală de tip "mergi înainte". În aceste condiții, ori de câte ori robotul se apropie suficient de mult de un obstacol (cu alte cuvinte, unul sau mai multe obstacole intră în raza de acțiune a senzorilor IR), algoritmul GA intră în execuție și, astfel, este găsită cea mai bună soluție care să rezolve problema evitării obstacolului/obstacolelor respective. Detaliind, putem spune că, rolul algoritmului GA în astfel de situații constă în găsirea celui (celor) mai bun (buni) cromozom(i) care să încapsuleze acele comenzi pentru motoare care minimizează funcția de fitness.

Algoritmul GA lucrează cu o populație de cromozomi. În alte lucrări similare, fiecare cromozom în parte caracterizează un singur robot **[Messom, 2002]**. În cazul nostru, însă, dispunem doar de un singur robot care operează algoritmul GA – fapt care poate fi considerat ca reprezentând o limitare practică majoră. Pentru a depăși această limitare, într-o primă fază, robotul se va deplasa într-o direcție specificată de comenzile motoarelor încapsulate în primul cromozom din populație. După 300 ms robotul se oprește iar funcția de fitness asociată cu acest cromozom este evaluată.

Apoi, robotul revine în poziția inițială și algoritmul continuă, urmând aceeași procedură, cu ceilalți cromozomi din populație.

Tabel 1: Baza de cunoștințe ale mișcării – înregistrări reale

Senzori				Motoare	
Stânga	Centru	Dreapta	Spate	Stânga	Dreapta
⋮	⋮	⋮	⋮	⋮	⋮
0.54	0.19	0	0	105	-100
0	0	0.57	0	-109	117
0.49	0.37	0	0	103	-109
0	0	0.61	0	47	85
0.34	0	0	0	93	10
0	0	0.41	0	-104	56
0.43	0.62	0.31	0	-128	-72
0	0	0.40	0.32	75	91
⋮	⋮	⋮	⋮	⋮	⋮

După fiecare generație, valoarea funcției de fitness corespunzătoare celui mai bun cromozom din întreaga populație se îmbunătățește. La sfârșit, GA-ul obține cea mai bună soluție ce asigură evitarea respectivului obstacol particular întâlnit. Cu alte cuvinte, folosindu-se de cel mai bun cromozom obținut, robotul reușește să se deplaseze în mod corespunzător și să navigheze corect, fără nici un fel de coliziune. De fiecare dată după ce robotul evită cu succes obstacolul, el reîncepe să meargă conform comenzii globale "*mergi înainte*"; acest lucru se va întâmpla atât timp cât nici un obstacol nu va fi semnalat de către sistemul de senzori ai robotului. Când, însă, un nou obstacol va fi întâlnit, GA-ul va fi din nou activat în vederea rezolvării noii probleme de evitare apărute. Câteva astfel de secvențe alternante *deplasare conform comenzii "mergi înainte" – activare GA cu extragere de noi reguli de mișcare* vor forma așa-numia ***fază de învățare*** a robotului (vom detalia mai jos această fază).

De fiecare dată când GA-ul rezolvă un task de evitare de obstacole, soluția găsită este salvată într-un tabel – cunoscut și sub numele de ***bază de cunoștințe a mișcării***, BCA. Fiecare linie din acest tabel conține informația privind valorile senzorilor dinaintea execuției algoritmului GA (4 valori – câte o valoare pentru fiecare senzor), precum și informația stocată în cel mai bun cromozom obținut după execuția GA (două valori ce corespund comenzilor obținute pentru motoare), vezi **Tabel 1**.

De fiecare dată în faza de evitare propriu-zisă (când un obstacol intră în raza de acțiune a robotului) se calculează, mai întâi, o distanță euclidiană între valorile actuale ale senzorilor și valorile senzorilor stocate în baza de cunoștințe extrasă până în acel moment. Dacă printre distanțele calculate există și distanțe mai mici decât un prag predefinit atunci comenzile pentru motoare corespunzătoare setului de valori ale senzorilor din BCA pentru care s-a obținut cea mai mică distanță euclidiană vor constitui comenzile care se vor executa. În caz contrar, robotul intră

într-o nouă fază de învățare în care o nouă regulă a mișcării este extrasă și stocată în baza de cunoștințe. În acest mod, controller-ul cu auto-evoluție devine unul înzestrat cu capacitatea de auto-adaptare la orice nouă situație de mediu pe care nu a mai întâlnit-o anterior.

Concluzionând, din cele prezentate anterior deducem că în faza de evitare a unui obstacol robotul se poate găsi în una dintre cele două situații, respectiv, fie acesta folosește baza de cunoștințe extrasă anterior pentru a rezolva problema fie, în cazul în care BCA nu oferă o soluție, lansează un nou proces de învățare ce se va solda cu extragerea unei noi reguli de mișcare. Alegerea uneia sau alteia dintre aceste două alternative posibile se face funcție de întrunirea sau nu a criteriului privind pragul predefinit, amintit mai sus, pentru distanța euclidiană care se calculează.

Faza de învățare a robotului este compusă din toate acele momente în care algoritmul GA a fost lansat în execuție. În fapt, numărul acestor momente este unul egal cu numărul de reguli care au fost extrase și stocate în ***baza de cunoștințe***, BCA.

6. Rezultate și discuții

Prima implementare a controller-ului cu auto-evoluție

Într-o primă încercare, toți pașii prezentați anterior au fost implementați și testați direct pe un robot real (vezi **Figura 1**) – fără etapa premergătoare, de simulare. Metoda aleasă pentru selecția cromozomului a constat în *eșantionarea universal stohastică*; această metodă este una caracterizată atât prin bias zero cât și prin varianță minimă **[Baker, 1987]**. Metoda crossover folosită a fost una în două puncte. **Tabelul 2** prezintă lista principalilor parametri ai algoritmului GA, așa cum au fost ei implementați pe sistemul robotic.

Tabel 2: Principalii parametri ai algoritmului genetic

Parametru	Valoare
Mărimea populației	10
Generații	20
Generation gap	0.9
Probabilitatea de mutație	0.0437
Nr. puncte recombinare	2
Probabilitate recombinare	0.7

Ca rezultat al implementării de mai sus am obținut, în final, un prototip funcțional. Un mare dezavantaj, însă, al controller-ului cu GA implementat astfel l-a constituit timpul mare de calcul pe care l-a necesitat robotul în faza de învățare. La rândul său, acest timp de învățare foarte mare a generat alte probleme adiționale cum ar fi uzura părților mecanice ale robotului precum și necesitatea unui număr mare de cicluri de reîncărcare a bateriilor. Pentru a reduce acest timp am exploatat o serie de observații practice pe care le-am făcut pe parcursul cercetării. Si anume, nu de fiecare dată algoritmul GA necesita 20 de generații pentru a converge. În multe situații, după doar câteva generații (în câteva cazuri, chiar și numai după o singură generație) funcția de fitness

lua o valoare egală cu zero. Ținând cont de această observație am impus, ulterior, ca algoritmul GA să se oprească ori de câte ori funcția de fitness pentru un singur cromozom lua o valoare egală cu zero. În acest mod, timpul necesar învățării a fost, în mod considerabil, diminuat.

Însă, cu toate acestea, chiar și după toate îmbunătățirile software aduse algoritmului, timpul petrecut în faza de învățare a rămas unul, în continuare, ridicat. În plus față de această problemă, alte dificultăți întâlnite în cazul acestei implementări în mediu real și pe un robot fizic au fost cele legate de posibilitatea reală de a face debug pe modulele software dezvoltate. Aceste noi dificultăți, cumulate împreună cu timpul mare necesar pentru fiecare testare practică în parte, au făcut ca soluția noastră să fie una realmente nepractică. Din acest motiv, s-a impus în continuare necesitatea unei abordări alternative.

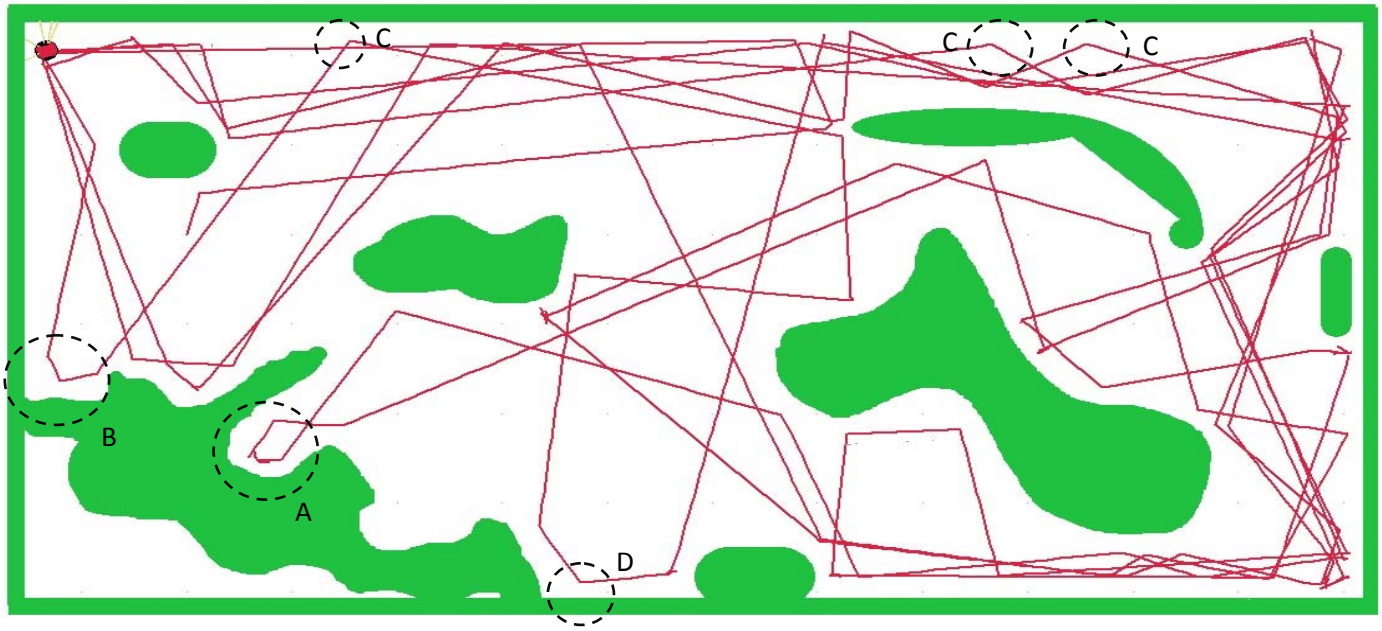

Figure 2: The **Learning Phase** of the Robot

Pentru a depăși toate dezavantajele menționate mai sus și, astfel, pentru a putea, totuși, confirma conceptele principale ale noului controller GA cu auto-evoluție, am ales următoarea abordare complementară care reduce considerabil procesul de dezvoltare a sistemului robotic:

(1) o parte consistentă a fazei de învățare are loc într-un mediu de simulare care, la final, pune la dispoziție **baza de cunoștințe-nucleu a mișcării** care a fost astfel extrasă urmată, apoi, de

(2) faza de testare a controller-ului; această a doua fază are loc într-un mediu real și ea implică exploatarea setului de reguli extras anterior, precum și posibilitatea de a extrage noi astfel de reguli printr-un process de învățare în timp real ce folosește același algoritm GA auto-evolutiv.

Simulatorul

O problemă nouă pe care o ridică această ultimă abordare rezidă în posibilitatea controller-ului cu auto-evoluție de a învăța un comportament de evitare ce este unul adaptat trăsăturilor particulare ale sistemului robotic simulat – avem în vedere aici acele trăsături al sistemului robotic simulat care nu au, în mod necesar, și un correspondent identic în lumea reală.

Acest nou tip de problemă a impus ca modelarea atât a robotului (caracteristicile sale mecanice, electrice și electronice) cât și a mediului său să se facă cât mai realistic posibil. Pentru atingerea acestui obiectiv am ales ca mediu de simulare simulatorul MobotSim, versiunea 1.0.03. MobotSim este un simulator 2D configurabil, dedicat roboților mobili cu roți și acționare diferențială. În cadrul acestui mediu de simulare limbajul folosit a fost Sax Basic – un limbaj compatibil cu limbajul Visual Basic for Applications™. Pentru robot următorii parametri au fost configurați: diametrul platformei robotice, distanța dintre roți, diametrul roților, numărul și dispunerea senzorilor (unghiul dintre senzori), raza cercului pe care se dispun senzorii, conul de radiație corespunzător activării senzorilor, domeniul de activare a senzorilor, precum și procentul de citiri eronate ale valorilor senzorilor. Toți acești parametri au fost setați în perfectă corespondență cu trăsăturile mecanice și electronice ale sistemului robotic real folosit, ulterior, în faza de testare.

A doua implementare a controller-ului cu auto-evoluție

În cea de a doua implementare a robotului am separat, așa cum am amintit anterior, ***faza de învățare*** de ***faza de testare***. În faza de învățare am folosit un mediu de simulare în timp ce faza de testare s-a făcut pe un robot real și într-un mediu real.

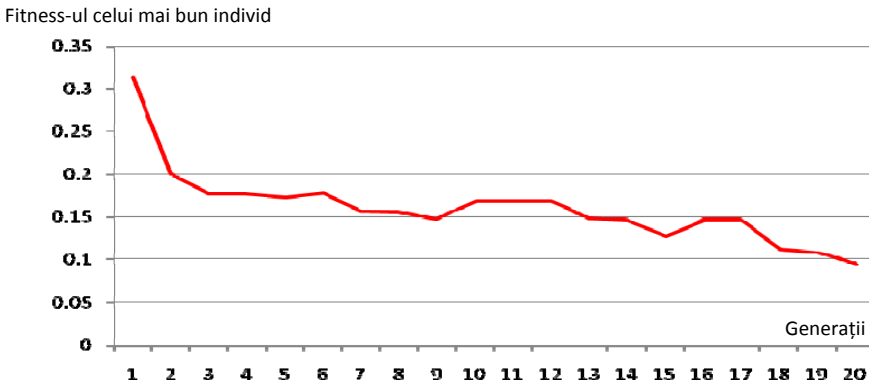

Figura 3: Evoluția funcției de fitness pentru cel mai bun cromozom din fiecare generație – reprezentare aferentă cazului D din Figura 2

Figura 2 prezintă rezultatele obținute în faza de învățare, pe mai mult de 200 de execuții ale algoritmului GA. Datele reprezentate în această figură au fost obținute pe un calculator Intel(R) Core(TM) i7, cu CPU la 2.8 GHz, cu 8GB de RAM iar timpul necesar obținerii lor a fost de 12 ore și 24 de minute. **Figura 3** prezintă și ea evoluția algoritmului GA (mai exact, vorbim aici de

evoluția fitness-ului pentru cel mai bun cromozom din fiecare generație) corespunzătoare cazului D redat în **Figura 2**. Din această reprezentare grafică se poate observa capacitatea algoritmului GA de a îmbunătăți performanțele în cadrul procesului de evitare a obstacolelor, precum și capacitatea acestuia de a rezolva fiecare dintre task-urile particulare de evitare a obstacolelor indicate în **Figura 2**.

Unul dintre parametrii principali ai algoritmului GA (cu un impact important atât asupra timpului de convergență cât și asupra calității soluției obținute) este *mărimea populației*. În cazul nostru, acest parametru este cu atât mai critic cu cât de fiecare dată când robotul întâlnește o situație nouă (nemaiîntâlnită până atunci), algoritmul GA este lansat în execuție, el trebuind să rezolve noua problemă de evitare a obstacolelor apărută. În acest ultim caz, și mai ales cu deosebire atunci când vorbim de aplicații reale – un timp de convergență scurt, precum și o soluție bună la problema de evitare apărută, oferite de algoritmul GA se dovedesc a fi cruciale în situații critice. După un număr mare de teste, o populație de 10 cromozomi s-a găsit a fi una suficientă pentru a asigura atingerea celor două deziderate amintite.

În **Figura 2** se poate observa că, controller-ul GA cu auto-evoluție este capabil să rezolve toate tipurile de coliziuni așa cum sunt, spre exemplu, situațiile în care robotul ajunge în colțuri sau unele situații chiar mai dificile decât acestea, similare celor marcate, de exemplu, cu A și B. Capacitatea robotului real de a rezolva situații reale este dată de diversitatea situațiilor încapsulate în baza de cunoștințe a mișcării. Din acest considerent, faza de învățare trebuie oprită nu după un timp specificat *a priori* sau după un număr dat de execuții ale algoritmului GA. Se va considera că faza de învățare s-a încheiat doar atunci când robotul simulat a rezolvat deja un număr foarte mare de situații de mediu cât mai variate. În caz contrar, există riscul ca baza de cunoștințe să conțină doar soluții pentru cele mai comune situații de mediu întâlnite – așa cum sunt, spre exemplu, cele marcate cu C în **Figura 2**.

Mediul real de testare a robotului a constat într-o zonă delimitată (de cca. 2m x 2m), cu un singur obstacol plasat aleator în această zonă. Delimitarea zonei s-a făcut cu ajutorul unor cutii, de dimensiuni și culori diferite, plasate astfel încât să se obțină un perimetru închis. Din întreaga bază de cunoștințe extrasă în faza de învățare (vezi **Figura 2)**, doar 50 de reguli de mișcare au fost reținute la testarea în mediul real. Primele 50 de reguli includ cazurile A și B ilustrate în **Figura 2**.

Cu ajutorul acestor reguli robotul a fost capabil să navigheze corect în mediul real, fără nici o coliziune. De asemenea, trebuie subliniat și faptul că, deși robotul a folosit în mediul real doar 50 din totalul regulilor extrase prin simulare, lansarea în execuție a GA nu s-a făcut niciodată. Mai mult, făcând comparație cu alte abordări existente în literatură, una dintre trăsăturile cele mai importante ale acestui controller robotic auto-evolutiv rămâne capacitatea lui de a naviga în medii reale fără nici un fel de coliziune – lucru care nu se regăsește în aceeași măsură și în faza de testare a altor soluții raportate precum sunt cele bazate pe RNA **[Dobrea, 2010]**, **[Tan, 2011]**.

7. CONCLUZII

Această lucrare prezintă un nou controller inteligent cu auto-evoluție dedicat pentru sistemele robotice mobile. Acest controller ajută robotul real să auto-evolueze la un comportament care să-i permit acestuia să simtă, să raționeze și să acționeze – și toate acestea în scopul principal de a evita obstacolele din mediul real înconjurător.

Așa cum am menționat anterior, o singură evaluare a întregii populații, chiar și pentru o populație mică de doar 10 cromozomi, necesită un timp de calcul semnificativ. Din acest considerent, principalul set de reguli de mișcare a fost extras, într-o primă etapă, în cadrul unui mediu de simulare. Acest set de reguli este cel care asigură, mai apoi, suportul principal în faza ulterioară, de testare în mediu real. În etapa de testare, atunci când o situație nemaiîntâlnită apare, se lansează în execuție automat același algoritm genetic implementat și în mediul de simulare. Acesta va extrage, în timp real, noua regulă a mișcării (capabilă să rezolve noua situație apărută) iar soluția găsită va fi stocată în BCA în vederea aplicării ei în viitor în toate acele situații ce vor apare și care vor fi similare celei care tocmai a fost soluționată.

În abordările clasice, transferul controller-elor (obținute prin evoluție într-un mediu de simulare) pe un robot fizic constituie o adevărată provocare **[Messom, 2002]**. În abordarea propusă în această lucrare, transferul bazei de cunoștințe obținută în cadrul mediului de simulare pe robotul real a constituit un succes, el putând fi realizat de o manieră foarte facilă. În consecință, un alt mare avantaj al acestei noi abordări propusă pentru problema evitării locale, autonome a obstacolelor, îl constituie și ușurința implementării soluției pe sisteme robotice reale.

Referințe

[Tan, 2011] Tan S., Yang S.X. și Zhu A.M.: *A Novel GA-Based Fuzzy Controller for Mobile Robots on Dynamic Environments with Moving Obstacles*, International Journal of Robotics & Automation, 26(2): 212-228, 2011

[Chiou, 2010] Chiou J.S., Wang C.J., Wang K.Y., Hu, Y.C., Cheng S.W. și Chen C.H.: *Hybrid Algorithm of FLC Design for Robot Soccer*, International Journal of Nonlinear Sciences and Numerical Simulation, 11: 119-122, 2010

[Martíneza, 2009] Martíneza R., Castilloa O. și Aguilarb L.T.: *Optimization of interval type-2 fuzzy logic controllers for a perturbed autonomous wheeled mobile robot using genetic algorithms*, Information Sciences, 179(13): 2158-2174, 2009

[Hosseinzadeh, 2010] Hosseinzadeh A. și Izadkhah H.: *Evolutionary Approach for Mobile Robot Path Planning in Complex Environment*, International Journal of Computer Science Issues, 7(4): 1-9, 2010

[Repoussis, 2009] Repoussis P.P., Tarantilis C.D. și Ioannou G.: *Arc-Guided Evolutionary Algorithm for the Vehicle Routing Problem With Time Windows*, IEEE Transactions on Evolutionary Computation, 13(3): 624 – 647, 2009

[Dobrea, 2010] Dobrea D.M. și Dobrea M.C.: *An auto-organization bio-inspired robotic system*, în Proceedings of the International Conference on Future Information Technology (Changsha, Ch, Dec. 14-15). IEEE, Picataway, N.J., 354-358, 2010

[Baker, 1987] Baker J.E.: *Reducing bias and inefficiency in the selection algorithm*, în Proceedings of the Second International Conference on Genetic Algorithms (Cambridge, USA). L. Erlbaum Associates Inc. Hillsdale, NJ, pp. 14-21, July 1987

[Tan, 2008] Tan A.H., Lu N. și Xiao D.: *Integrating Temporal Difference Methods and Self-Organizing Neural Networks for Reinforcement Learning With Delayed Evalutive Feedback*, în IEEE Transactions on Neural Network, 19: 230-244, 2008

[Messom, 2002] Messom C.: *Genetic Algorithms for Auto-tuning Mobile Robot Motion Control*, în Research Letters in the Information and Mathematical Sciences, 3: 129-134, 2002