Lucrarea nr. 1
Identificarea mediului de simulare

1. Tema lucrării: Introducere în simularea computerizată: prezentarea noţiunilor de modelare a obiectelor şi a mediului. Sisteme de realitate virtuală
2. Scopul lucrării: prezentarea introductivă a sistemelor de realitate virtuală pentru modelarea mediului. Utilizarea limbajului de programare Visual Basic pentru construcţia mediului (scenei) de simulare.
3. Consideraţii teoretice

3.1 Introducere
3.2 Notiuni de modelare a obiectelor şi a mediului
3.3 Sisteme de realitate virtuală
3.4 Modelarea mediului sub formă de tip scenă statică

3. 1 Introducere

Dezvoltarea sistemelor tehnice moderne este caracterizată printr-o tendinţă continuă de creştere a gradului de complexitate, atât prin neomogenizarea componentelor utilizate, cât şi prin implementarea unui grad tot mai ridicat de inteligenţă. Această tendinţă impune creşterea încrederii utilizatorului în sistemele cu funcţii complexe.

În contextul actual al sistemelor mecatronice mobile se doreşte o realizare a cât mai multor sarcini de către acestea fără a li se specifica fiecare acţiune ce urmează să fie realizată.

Problema de planificare a mişcărilor unui sistem mecatronic mobil presupune realizarea unei deplasări de la o locaţie la altă locaţie, într-un mediu care conţine o mulţime finită de obstacole, astfel încât sistemul mecatronic mobil să nu intre în coliziune cu nici unul dintre obstacole. Abilitatea ocolirii obstacolelor este indispensabilă pentru orice sistem mecatronic real.

În vederea realizării unei planificări a mişcării cât mai eficientă pentru un sistem mecatronic mobil, de-a lungul anilor s-au realizat mai multe soluţii folosind diverşi algoritmi şi programe de simulare.

După cum se poate vedea din contextul amintit anterior, abilitatea unui sistem mecatronic mobil de a se descurca printre obstacole în mod independent de factorul uman sporeşte semnificativ potenţialul capacităţilor şi a şirului de aplicaţii ce sunt îndeplinite de acesta.

3.2 Notiuni de modelare a obiectelor şi a mediului

Dacă un robot este un sistem cu abilităţi în mişcare şi/sau de manipulare, atunci una dintre cele mai importante probleme de rezolvat este aceea de a îi planifica mişcările, ceea ce implică modelarea spaţiului de lucru corespunzător mediului în care se realizează integrarea, cu obstacolele pe care le conţine, şi a robotului, ca entitate de formă complexă şi variabilă.

Mediul de lucru reprezintă o parte a lumii reale – naturale şi/sau artificiale, iar pentru proiectarea şi realizarea acţiunilor asupra sa sau a răspunsurilor la stările sale, este necesară cunoaşterea părţii de lume reală corespunzătoare. Cunoaşterea lumii reale şi a părţilor sale se bazează pe construirea şi utilizarea de modele, care reprezintă ansamblul conceptelor, noţiunilor şi relaţiilor dintre acestea şi care descriu fenomene şi procese specifice, obţinându-se astfel o reprezentare conceptuală a lumii reale

În acest cadru, modelele se vor caracteriza, în principal, prin:

- nivelul de abstractizare, care exprimă gradul de formalizare conceptuală şi matematică a proceselor şi fenomenelor considerate;
- gradul de complexitate, care se referă atât la partea de fenomene şi procese modelate din totalitatea celor specifice părţii de lume reală considerate, cât şi la măsura în care sunt stabilite şi reprezentate relaţiile şi interacţiunile dintre fenomenele şi procesele respective;
- gradul de generalitate - măsură a posibilităţii de aplicare a modelului la nivelul altor părţi ale lumii reale sau la nivelul global al acesteia.

Dacă avem în vedere definiţiile mecatronicii ca ştiinţă, precum şi caracteristicile sale de eterogenitate, de deschidere şi de integrare sinergetică a domeniilor componente, rezultă că modelele şi modelarea vor căpăta caracteristici specifice suplimentare:

- modelarea devine eterogenă atât în cazul considerării sistemelor mecatronice ca rezultate ale integrării unor domenii diferite, cât şi în cazul analizei mediului de lucru, sub toate aspectele de interes la un moment dat sau în perspectivă;
- modelele capătă caracter sinergetic prin considerarea unei palete mai largi de fenomene şi procese şi prin stabilirea relaţiilor între acestea; astfel, modelele specifice mecatronicii vor fi caracterizate printr-un grad de complexitate mai mare.

Planificarea mişcărilor poate fi considerată ca fiind problema realizării algoritmilor pentru a calcula automat o traiectorie continuă pentru o mulţime de obiecte astfel încât să se deplaseze de la o poziţie la alta evitând coliziunile cu alte obiecte fixe sau având mişcare proprie.

Pentru un robot cu bază fixă problema se poate formula mai simplu prin alegerea unei traiectorii ferite de coliziuni pentru braţul robotului, între două poziţii, în cazul unui spaţiu închis. Soluţia acestei probleme este un pas înainte spre planificarea acţiunilor robotului la nivel de sarcină; aceasta înseamnă că se pot ignora secvenţele intermediare ce trebuie urmate pentru realizarea unei sarcini.

Cu cât sistemul care se mişcă este mai complex, iar spaţiul în care se mişcă mai populat cu alte obiecte, problema are complexitate mai mare. Însăşi trecerea de la cazul bidimensional la cel tridimensional complică problema chiar dacă rămâne în studiu acelaşi robot mobil în acelaşi spaţiu de lucru. Dificultatea sporeşte cu atât mai mult cu cât robotului îi creşte numărul gradelor de libertate, pentru că aceasta înseamnă o arhitectură mai complicată şi o modelare mai dificilă şi mai complexă atât a robotului, ca sistem de corpuri, cât şi a spaţiului propriu de lucru definit prin mişcarea efectorului final.

Este evident că problema reprezentării obiectelor are o importanţă crucială pentru robotică. Astfel procedeul prin care este creat un obiect grafic poartă numele de modelare. Modul cel mai convenabil de creare a scenelor virtuale este acela în care fiecare obiect este modelat într-un sistem de coordonate propriu, numit sistem de referinţă model (sau sistem de referinţă local), în care punctele (vârfurile) obiectului sunt precizate relativ la un anumit punct de referinţă local.

Instanţierea unui obiect în scena virtuală înseamnă amplasarea acestuia în sistemul de referinţă universal printr-o succesiune de scalări, rotaţii şi translaţii, care transformă obiectul din sistemul de referinţă local în sistemul de referinţă universal. Această succesiune de transformări este cunoscută sub numele de transformare de modelare.

Proprietăţile obiectelor tridimensionale care se modelează în aplicaţiile grafice se pot împărţii în două categorii: forma şi atributele de aspect.

1. Forma – determină modul în care obiectul apare în redarea grafică şi toate celelalte atribute se corelează cu forma obiectului. Din punct de vedere al formei, obiectele reprezentate în grafica pe calculator pot fi:

a) Solide – forma şi dimensiunile obiectului nu se modifică în funcţie de timp şi de poziţia în scenă;

b) Deformabile - forma şi dimensiunile unui obiect solid se modifică în funcţie de timp şi de poziţia în scenă.

2. Atributele de aspect – sunt atributele secundare ale aspectului unui obiect realizat în grafica tridimensională, cum sunt culoarea, textura, iluminarea, transparenţa, umbrirea, radiozitatea, etc.

Modelarea solidelor este o tehnică de proiectare, vizualizare şi analiză a modului în care obiectele reale se reprezintă în calculator. În ordinea importanţei şi a frecvenţei de utilizare, metodele de modelare şi reprezentare a obiectelor sunt următoarele:

- Modelarea poligonală – obiectele sunt aproximate printr-o reţea de feţe care sunt poligoane plane;
- Modelarea prin reţele de zone parametrice bicubice – obiectele sunt aproximate prin reţele de elemente spaţiale numite petice şi sunt reprezentate prin polinoame cu două variabile parametrice, cubice;
- Modelarea prin compunerea obiectelor – obiectele sunt reprezentate prin colecţii de obiecte elementare, cum sunt cilindrii, sfere, poliedre, etc.;
- Modelarea prin divizare spaţială – obiectele sunt încorporate în spaţiu prin atribuirea unei etichete fiecărui element spaţial, în funcţie de obiectul care ocupă elementul respectiv.

În domeniul reprezentării spaţiale, au fost folosite pe parcursul timpului mai multe tipuri de modele 3D şi 2D, mai ales în aplicaţiile grafice computerizate: geometria corpurilor, reprezentarea limitelor, descompunerea în celule sau sub-spaţii de altă natură decât geometrică, volume acoperitoare, arbori de decizie, etc.

Utilizarea doar a primitivelor standard poate limita aplicabilitatea sistemului. De aceea este necesar să se ofere utilizatorului posibilitatea de a defini, după necesităţi, entităţi geometrice primare. Se pot astfel crea curbe pornind de la puncte, suprafeţe pornind de la curbe şi volume pornind de la suprafeţe. În continuare vor fi prezentate următoarele două forme distincte de reprezentare:

a) Reprezentarea prin porţiuni parametrice, în care obiectele sunt reprezentate exact prin reţele poligoane curbilinii, numite în mod uzual petice. Acestea sunt exprimate prin polinoame de două variabile parametrice şi, de obicei, sunt cubice.

b) Reprezentarea prin tehnici de subdiviziune spaţială, ce constă din reprezentarea unui obiect prin asocierea de celule dintr-un spaţiu divizat în celule egale sau inegale, tehnica de asociere fiind bazată pe metode selectate din teoria grafurilor sau pe construcţia de arbori.

3.3 Sisteme de realitate virtuală

În cadrul teoriei generale a modelării se operează cu conceptul de sistem, modelarea propriu-zisă constituind ansamblul activităţilor prin care, pentru un anumit scop determinat, un

sistem sursă este înlocuit de un sistem model echivalent din anumite puncte de vedere, sistemul sursă putând fi real sau virtual.

Realitatea virtuală este o tehnologie care a fost abordată cu interes în ultimii ani, deoarece un sistem de realitate virtuală poate constitui atât un instrument de instruire, cât şi de experimentare ştiinţifică.

Realitatea virtuală înglobează o mare varietate de tehnologii şi echipamente care generează senzaţii adresate simţurilor umane şi are o arie de aplicabilitate în continuă extindere: simulatoare de zbor (aeronautică şi astronautică), simularea operaţiilor chirurgicale cu invaziune minimală (medicină), jocuri video şi programe educaţionale (artă şi divertisment), etc.

Există 4 categorii de sisteme de realitate virtuală utilizate des : sisteme de realitate virtuală imersive, sisteme proiective, sisteme de realitate îmbogăţită şi sisteme de realitate virtuală desktop:

a. Într-un sistem de realitate virtuală imersiv, contactul participantului cu lumea reală este complet întrerupt, acestuia permiţându-se să vadă numai imaginea mediului sintetic, să audă numai sunetele generate artificial şi să interacţioneze numai cu obiectele virtuale pe care la vede în scenă. Această incluziune totală a participantului în mediul virtual se obţine prin dispozitive de afişare (display-uri) montate pe cap (HMD – head-mounted display), căşti audio (headphones), mănuşă de date (data glove) şi îmbrăcăminte de date (data suits).

b. În sistemele de realitate virtuală proiectivă, imaginea mediului tridimensional este proiectată pe unul sau mai multe ecrane, care pot fi văzute de unul sau mai mulţi utilizatori. Imaginea afişată pe ecrane urmăreşte acţiunile unuia dintre utilizatori, care demonstrează anumite acţiuni sau concepte celorlalţi utilizatori din grup.

c. Sistemele de realitate îmbogăţită combină informaţiile generate de calculator cu cele ale unui mediu real. Spre deosebire de sistemele imersive, în realitatea îmbogăţită utilizatorul percepe lumea reală, cu obiecte virtuale suprapuse peste imaginea acesteia. Şi în aceste aplicaţii se foloseşte dispozitivul de afişare montat pe cap (HMD) care suprapune date generate de calculator peste imaginea mediului real.

d. În sistemele de realitate virtuală desktop, imaginea vizuală a mediului virtual tridimensional este afişată pe monitorul unui calculator (PC). Participantul interacţionează cu mediul virtual prin dispozitive de intrare standard (tastatură, mouse, joystick). Aceste sisteme permit observarea mediului virtual printr-o fereastră (ecranul monitorului) şi de aceea se mai numesc şi sisteme WOW (Window On the World).

Un sistem de realitate virtuală este compus din mai multe subsisteme care comunică între ele pentru redarea interacţiunii între utilizator şi mediul virtual:

- un sistem de calcul – care prelucrează datele de modelare a mediului virtual (conţinute de bazele de date) şi generează imaginile corespunzătoare acţiunilor efectuate de utilizator;
- dispozitivele de intrare – sunt traductoare care convertesc acţiunile utilizatorului într-o formă interpretabilă de către calculator: display-uri video, căşti audio, mănuşi senzitive;
- un sistem de timp real – impus de cerinţele de imersiune şi interactivitate pentru realitatea virtuală. Pentru ca utilizatorul să simtă o parte a mediului virtual, trebuie ca sistemul să răspundă acţiunilor sale într-un interval de timp suficient de mic, astfel încât diferenţa dintre momentul de execuţie a unei acţiuni şi răspunsul recepţionat să nu fie perceptibilă;
- imaginea vizuală – se generează prin calcul pornind de la modelele obiectelor din mediul virtual memorate sub formă de baze de date (scena virtuală), corespunzând poziţiei şi acţiunilor utilizatorului. Imaginea vizuală se completează (după caz) cu sunetul virtual şi senzaţia tactilă.

Se observă că un sistem de realitate virtuală este un sistem de control în buclă închisă, definit de interactivitate, imersivitate şi imaginaţie.

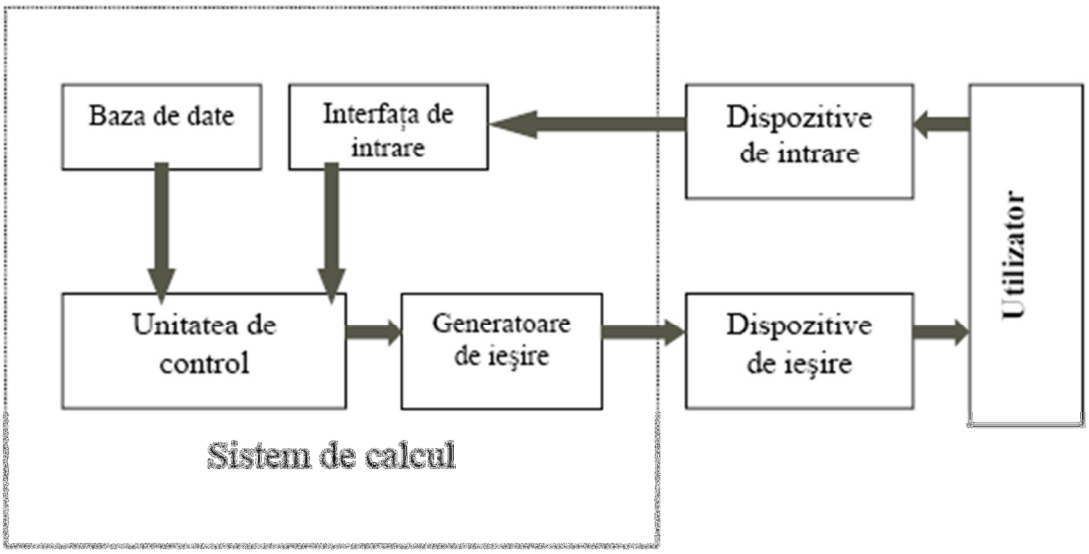


Figura 1. Schema bloc a unui sistem de realitate virtuală (după Ionescu F., opcit., p. 7)

Mediul sintetic în care evoluează utilizatorul unui sistem de realitate virtuală (numit şi cybernaut) este definit într-un sistem de coordonate cartezian tridimensional numit sistem de coordonate universal (world coordinate system). Fiecare obiect are coordonatele lui relativ la acest sistem de referinţă şi, de asemenea, utilizatorul are o poziţie şi o orientare definită în acest sistem de referinţă. Toate obiectele pe care utilizatorul le poate vedea la un moment dat se află în interiorul unei piramide numită piramidă de vizualizare, care este o subdiviziune a spaţiului, determinată de poziţia utilizatorului, orientarea capului acestuia şi unghiul de vizibilitate. Obiectele aflate în afara acestui volum se află încă în mediul virtual, dar nu sunt vizibile din poziţia şi orientarea dată a utilizatorului.

În sistemele grafice convenţionale sau în sistemele de realitate virtuală neimersive se poate urmării poziţia utilizatorului folosind dispozitive standard (mouse, trackball sau joystick) cu trei sau şase grade de libertate (dispozitivele cu trei grade de libertate permit măsurarea coordonatelor spaţiale x, y, z, iar cele cu şase grade de libertate măsoară în plus rotaţiile după cele trei axe).

În realitatea virtuală imersivă este necesară corelarea comenzilor senzoriale cu poziţia şi acţiunile utilizatorului, pentru a asigura senzaţia de imersiune. Dacă se urmăreşte poziţia capului utilizatorului, atunci când acesta întoarce capul într-o anumită direcţie, imaginea afişată pe display corespunde acelei direcţii. Cunoscându-se poziţia capului se poate calcula corect imaginea stereoscopică, compusă din două imagini diferite, corespunzătoare poziţiilor diferite ale celor doi ochi, şi sunetul virtual, corespunzător poziţiei celor două urechi.

În mod asemănător, este necesar să fie cunoscută poziţia mâinii utilizatorului, pentru calculul atingerii obiectelor virtuale. De aceea, trebuie să fie captată poziţia capului sau a mâinii, folosind dispozitive de captare montate pe casca de vizualizare, pe căştile audio sau pe mănuşa senzitivă. Dispozitivul de captare şi urmărire a poziţiei utilizatorului poartă numele de tracker şi

transmite datele de poziţie şi orientare către sistemul de generare a imaginilor, care le foloseşte pentru calculul şi afişarea obiectelor corespunzătoare şi a interacţiunii cu acestea.

Dispozitivele care pot fi folosite pentru captarea poziţiei utilizatorului în spaţiul tridimensional sunt de mai multe tipuri:

- magnetice – sunt compuse dintr-un ansamblu emiţător-receptor. Emiţătorul este montat într-o poziţie fixă şi generează un câmp magnetic de joasă frecvenţă care este recepţionat de antenele receptorului, ataşat utilizatorului aflat în mişcare . Din valoarea semnalului recepţionat se poate determina poziţia relativă a receptorului faţă de emiţător ;
- cu ultrasunete – folosesc emiţătoare şi receptoare de ultrasunete pentru determinarea poziţiei în spaţiu a utilizatorului. Emiţătorul este compus din trei difuzoare ultrasonore şi este montat într-o poziţie fixă, iar receptorul conţine trei microfoane şi este montat pe casca sau mănuşa utilizatorului. Semnalul captat de receptor permite determinarea poziţiei relative a receptorului faţă de emiţător;
- optice – folosesc grile de diode electroluminescente (LED) dispuse în poziţie fixă şi o cameră video montată pe casca utilizatorului. Diodele sunt activate în impulsuri, iar semnalul recepţionat de cameră este prelucrat pentru identificarea poziţiei relative a camerei faţă de grila de diode.

Generarea imaginii în realitatea virtuală implică două aspecte importante:

- crearea modelului scenei virtuale – este un proces off-line şi, de cele mai multe ori, de durată considerabilă, prin care se creează colecţia de modele ale obiectelor care constituie cea mai adecvată reprezentare a mediului virtual;
- Vizualizarea scenei virtuale – este un proces on-line, care se desfăşoară în timp real, cu participarea uneia sau mai multor persoane, în care scena virtuală este exploatată în mod interactiv şi, în fiecare moment, imaginea scenei redată pe display depinde de condiţiile de explorare (poziţie de observare, acţiuni interactive, etc.).

Pentru crearea unei imagini vizuale cât mai apropiate de realitatea fizică a mediului modelat, este necesar ca scena virtuală să conţină cât mai multe obiecte, redate cât mai realist. Generarea imaginii unui număr mare de obiecte ale scenei virtuale într-un interval de timp impus necesită o putere de calcul considerabilă şi de aceea generatoarele de imagine vizuală din sistemele de realitate virtuală sunt realizate pe baza unor arhitecturi paralele specializate care implementează accelerarea hardware a operaţiilor grafice.

Cel mai important indice de performanţă al unui generator de imagine vizuală este viteza de generare a imaginii, specificată prin numărul de poligoane redate pe secundă, iar calitatea imaginii generate este caracterizată de mai mulţi parametrii cum ar fi: rezoluţia, modelul de umbrire, capacitatea de texturare, filtrarea anti-aliasing, etc.

Pentru generarea imaginii în sistemele de realitate virtuală sunt necesare mai multe componente hardware:

- generatoare de imagine – sunt echipamente separate, care se conectează la un calculator gazdă (host), sau pot fi o componentă a sistemului de calcul, numită subsistem grafic sau accelerator grafic (expl: ProVision – firma Division; Evens&Sutherland; Thomson CSF; Silicon Graphics; etc.);
- dispozitive de afişare – în sistemele de realitate virtuală se utilizează două tipuri de astfel de dispozitive:

1. display-uri şi proiectoare – sunt folosite în sistemele de realitate virtuală neimersive sau semi-imersive, deoarece utilizatorul plasat în faţa unei imagini afişate pe display sau proiectate pe un ecran poate să mai perceapă şi elemente ale realităţii fizice. Imaginile afişate pe un singur display sau proiectate de un singur proiector au un câmp de vizualizare redus, de 35-45°. Astfel de sisteme de afişare se folosesc în sistemele de realitate virtuală desktop, unde display-ul de afişare este chiar consola calculatorului. În sistemele de realitate virtuală semi-imersive se folosesc mai multe display-uri juxtapuse sau proiectoare care proiectează imagini juxtapuse pe trei canale (canalul stâng, central şi drept);
2. dispozitivul de afişare montat pe cap (HMD) – casca de vizualizare permite imersiunea completă a utilizatorului în mediul virtual, prin percepţia vizuală numai a imaginii sintetizate a mediului virtual. Astfel de căşti sunt folosite în sistemele de realitate imersivă şi îmbogăţită.

Componentele software de generare a imaginii vizuale sunt aplicaţii ce se bazează pe sisteme de dezvoltare (toolkit-uri) sau pe biblioteci grafice care asigură interfaţa cu echipamentul hardware prin intermediul driverelor sistemului de operare.

- sistemele de dezvoltare toolkit – sunt de cele mai multe ori orientate către aplicaţie şi prevăd un set de funcţii de nivel înalt care permit crearea unui anumit tip de aplicaţie
- bibliotecile grafice – sunt pachete de funcţii care asigură interfaţa programului de aplicaţie (creat direct sau prin intermediul unui toolkit) cu echipamentele hardware ale sistemului grafic. Bibliotecile grafice reprezintă nivelul de programare în care se încearcă introducerea portabilităţii programelor grafice, prin asigurarea unei interfeţe independente de echipamentele hardware care să respecte anumite convenţii de reprezentare a entităţilor grafice descrise în standarde. Cele mai cunoscute biblioteci grafice sunt OpenGL, Direct3D şi QuickDraw.

3.4. Modelarea mediului sub formă de tip scenă statică

Pentru a putea realiza virtualizarea unui mediu trebuie construit un model al acestuia sub forma unei scene statice(spaţiu de lucru), în care se vor insera toate obiectele ce intră în componeţa acestuia. În continuare se vor prezenta câteva forme de modelare a unei scene statice (spatiu de lucru) folosite în simulare:

- Scenă statică de tip grilă – această formă de reprezentare a mediului constă în divizarea egală a mediului sub formă de pătrate, triunghiuri sau hexagoane (fig. 2, fig. 3, fig. 4) .

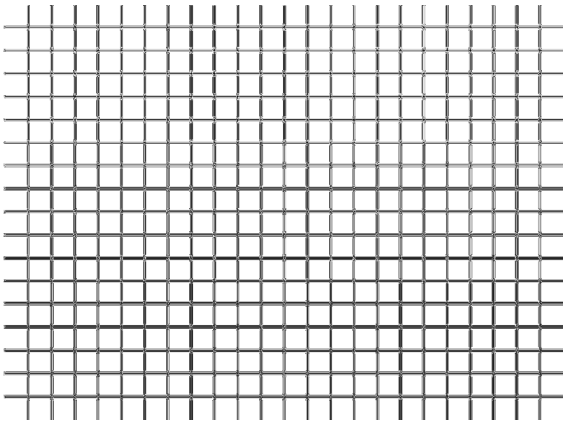


Figura 2. Scenă statică de tip grilă reprezentată prin pătrate

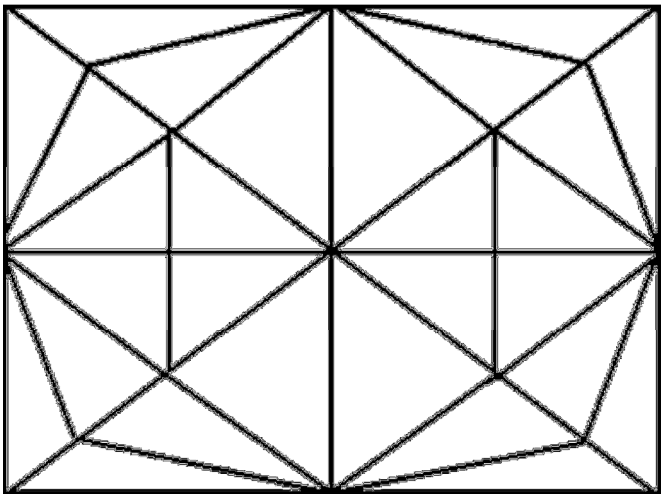


Figura 3. Scenă statică de tip grilă reprezentată prin triunghiuri

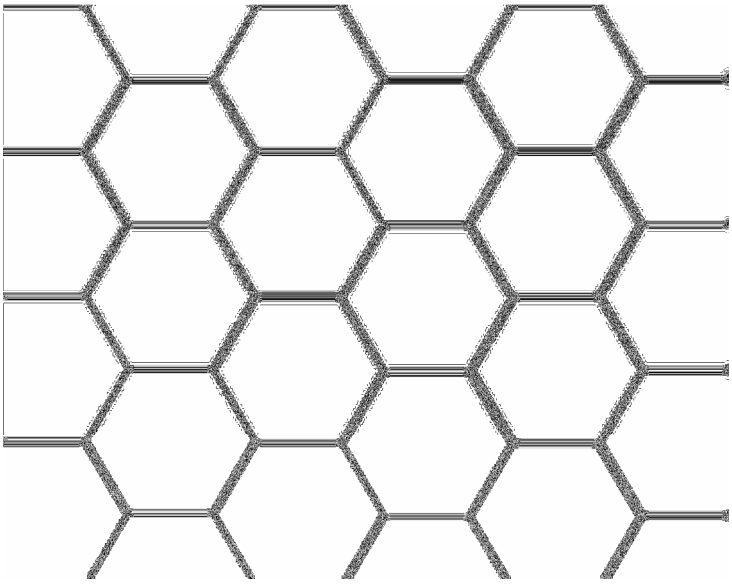


Figura 4. Scenă statică de tip grilă reprezentată prin hexagoane

- Scenă statică de tip poligonal – această modelare se foloseşte pentru mediile în care obstacolele sunt reprezentate sub formă de poligoane. Scena statică de tip poligonal se bazează în totalitate pe puncte de navigaţie (fig. 5).

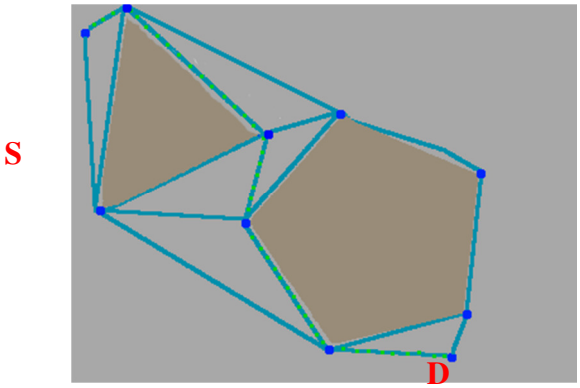


Figura 5. Scenă de tip poligonal [10]

După cum se poate observa si din figura 8, colţurile fiecărui obstacol au fost marcate cu puncte albastre iar punctul de start a fost marcat cu litera S iar cel de destinaţie cu litera D. Pentru implementarea acestui model unui algoritm trebuie în primul rând cunoscute nodurile şi ce puncte sunt unite. Pentru a putea determina acest lucru ne uitam la locurile punctelor ce nu sunt blocate de obstacole. Conform figuri marginile obstacolelor sunt liniile albastre iar punctele sunt nodurile ce unesc liniile albastre. Ultima etapă în implementarea unui algoritm pentru o astfel de scenă o constituie cunoaşterea distanţelor dintre puncte.

- scenă statică de tip ierarhic – acest tip de scenă se aplică algoritmilor care folosesc gasirea unui drum parcurgand mai multe nivele. Astfel dacă se dublează distanţă dintre punctul de start şi cel de destinaţie , durată de căutare a drumului va fi de două ori mai lungă.

Pentru a reduce problema căutări drumului pe distanţe mari este bine folosirea mai multor nivele de căutare.

De exemplu, dacă dorim să ajungem de la domiciliul nostru în alt oraş, vom realiza o cale de la scaunul pe care stăm la maşină, de la maşină la stradă, de la stradă la autostradă, de la autostradă la marginea oraşului, de la marginea oraşului la celălalt oraş, la strada din oraşul de destinaţie, la parcare şi în final la destinaţie [10]. Din exemplu de mai sus se pot observa câteva nivele de căutare a drumului:

1) nivelul străzii, unde suntem concentraţi să ajungem de la o locaţie la o altă locaţie apropiată;
2) nivelul oraşului, mergem de la o stradă la altă stradă până ajungem la autostradă;
3) nivelul statului, mergem dintr-un oraş în alt oraş.

Împărţind problema pe nivele, ne ajută să eliminăm mai multe alegeri ce ar fi mai costisitoare din punct de vedere al timpului şi distanţei parcurse. Astfel când dorim să ne deplasăm dintr-un oraş în altul nu luăm în considerare fiecare stradă din calea nostră ci ne ducem direct către autostradă. După cum se poate observa ajungând la autostradă problema devine mai mică iar rezolvarea ei mai rapidă.

- scenă statică de tip sferică – în acest tip de scenă obiectele sunt înfăşurate de la un capăt al scenei la celălalt capăt. Din acest motiv nu este clar cum ar trebui implementate datele într-un algoritm de căutare al drumului deoarece calea ar fi în orice direcţie şi din acestă cauză ar trebui explorate toate posibilităţiile.
- scenă statică de tip drum cu intersecţii – acestă scenă ne permite să luăm un mediu ce conţine o înşiruire de drumuri şi intersecţii (fig. 6).

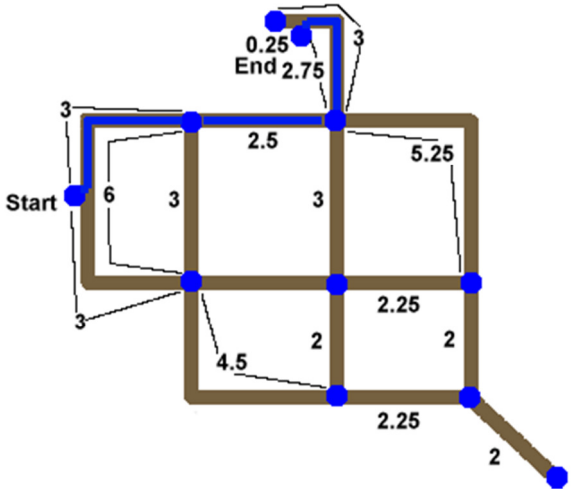


Figura 6 – Scenă statică de tip drum cu intersecţii [10]

După cum se observă din figura 6 fiecare intersecţie reprezintă un nod. Pentru a găsi drumul algoritmul caută calea prin fiecare nod de la poziţia de start la cea de destinaţie. Trebuie menţionat că în acestă metodă nodurile sunt deja cunoscute ceea ce va duce la găsirea mult mai rapid decât în cazul altor metode de reprezentare a mediului.

4. Aplicație

Să se implementeze inlimbajul visual basic mediul(scena) de lucru, pentru un robot mobil. Mediul de lucru va fi setat cu ajutorul unui "Form" principal în care se vor regăsi următorii parametri:

- Posibilitatea setarii nr. De obstacole și a dimensiunii acestora
- Posibilitatea setarii camerei în cm (lungime, lățime)
- Opțional se va putea seta Dimensiunea robotului și viteza de scanare pentru obstacolele aflate în cameră.

Rezolvare

În mediul de programare Visual Basic se vor crea două ferestre de tip "Form". O fereastra va fi destinată setărilor pentru mediului de lucru și o altă fereastră va fi destinată pentru afișarea mediului de lucru ce este simulat.

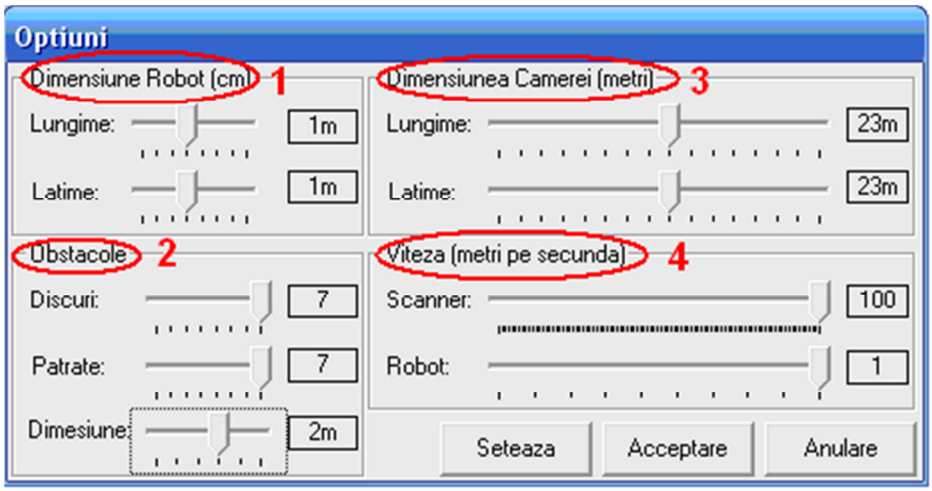


Figura 7. Opţiunile pentru setarea parametriloor de simulare

După cum se poate observa şi în figura 7 fereastra de opţiuni a fost împărţită în patru zone, fiecărei zone corespunzându-i căte un set de opţiuni:

- zona 1 – reprezintă zona unde se vor seta dimensiunile robotului. Dimensiunea robotului se setează atât pe lungime cât şi pe lăţime;
- zona 2 – reprezintă zona unde se vor seta numărul de obstacole cât şi dimensiunea acestora. Trebuie menţionat că programul nu realizează aceeaşi dimensionare a tuturor obstacolelor ci face o integrare a dimensiunilor obstacolelor între 0,5m şi valoarea setată;
- Zona 3 – reprezintă zona unde are loc setarea scenei statice (camerei);
- Zona 4 – repreznță zona unde are loc setarea vitezei pentru scanarea obstacolelor şi a deplasării robotului printre acestea.

Pentru început s-au declarat două variabile globale ce vor fi folosite pentru evenimentele butoanelor de Anulare şi Setare:

Dim Cancel, apply As Boolean

Pentru ca programul să poată păstra setările folosite cât şi salvarea lor s-a realizat o bază de date ce păstrează setări. Astfel la deschiderea ferestrei de opţiuni se face o verificare a bazei de date. Dacă baza de date există, setările vor fi preluate iar dacă baza de date nu există se va crea o bază de date nouă.

Private Sub Form_Load()

'se dau valorile de false pentru variabilele globale cancel si apply
Cancel = False
apply = False

'se incarca baza de date
Set Db = OpenDatabase(App.Path & "\settings.db", True, False, ";pwd=dbprotect")
'se preaiau valorile din fiecare camp al bazei de date

```
   Set RS = Db.OpenRecordset("setari")
   Set RSd = Db.OpenRecordset("snapshot")

'daca baza de date nu exista se va genera o noua baza de date
If RS.RecordCount > 0 Then
RS.MoveFirst
Slider2.Value = RS.Fields("camera_lungime")
Slider1.Value = RS.Fields("camera_latime")
Slider3.Value = RS.Fields("robot_lungime")
Slider4.Value = RS.Fields("robot_latime")
Slider5.Value = RS.Fields("obstacole_discuri")
Slider6.Value = RS.Fields("obstacole_patrate")
Slider7.Value = RS.Fields("obstacole_dimensiune")
Slider8.Value = RS.Fields("viteza_scanner")
Slider9.Value = RS.Fields("viteza_robot")
End If

End Sub
```

Pentru evenimentul de închidere al ferestrei de opţiuni programul execută o salvare a valorilor de simulare setate şi generează scena statică:

```
Private Sub Form_Unload(Cancel As Integer)

'daca butonul de anulare nu a fost apasat se trece la salvarea setarilor in baza de date
If Not Cancel Then

'se realizeaza o verificare a existentei bazei de date iar daca aceasta nu exista se creaza una
'noua iar in cazul in care exista se salveaza valorile setarilor
If RS.RecordCount > 0 Then
RS.Edit
Else
'adauga un nou camp
RS.AddNew
End If
RS.Fields("camera_lungime") = Slider2.Value
RS.Fields("camera_latime") = Slider1.Value
RS.Fields("robot_lungime") = Slider3.Value
RS.Fields("robot_latime") = Slider4.Value
RS.Fields("obstacole_discuri") = Slider5.Value
RS.Fields("obstacole_patrate") = Slider6.Value
RS.Fields("obstacole_dimensiune") = Slider7.Value
RS.Fields("viteza_scanner") = Slider8.Value
RS.Fields("viteza_robot") = Slider9.Value
RS.Update
```

```
If RSd.RecordCount > 0 Then
For n = 0 To RSd.RecordCount - 1
RSd.MoveFirst
RSd.Delete
Next
End If

For n = 0 To 6
RSd.AddNew
With Form1
RSd.Fields("disc_left") = .cerc(n).Left
RSd.Fields("disc_top") = .cerc(n).Top

RSd.Fields("disc_width") = .cerc(n).Width
RSd.Fields("disc_height") = .cerc(n).Height
RSd.Fields("patrat_width") = .patrat(n).Width
RSd.Fields("patrat_height") = .patrat(n).Height

RSd.Fields("patrat_left") = .patrat(n).Left
RSd.Fields("patrat_top") = .patrat(n).Top
End With
RSd.Update
Next

End If
'se activeaza scena statica
Form1.Enabled = True
End Sub
```

În momentul când utilizatorul va realiza setările dorite în cadrul ferestrei de opţiuni şi apăsă butonul “Setare”, va avea loc generarea tuturor elementelor simulării şi trimiterea lor în ferestra unde se vizualizează scena statică. Tot în cadrul acestui eveniment se vor mai declara trei variabile locale:

- o variabilă pentru păstrarea dimensiunilor maxime ale obstacolelor;
- două variabile pentru setarea coordonatelor obiectelor, astfel încât acestea să nu depăşească spaţiul setat din scena statică în care va avea loc simularea.

```
Dim dimMax As Double
Dim vx, vy As Integer
```

La pasul următor are loc calcularea conversiei pixelilor şi a metrilor pentru sliderele ce sunt folosite în dimensionarea obstacolelor, robotului, a camerei şi trimiterea acestor valori elementelor ce apar în scena statică (în cazul de faţă se face o trimitere către elemntele fereastrei Form1prin folosirea comenzi With Form1):

```
'Setez marimile conform sliderelor
With Form1
.room.ZOrder (1)
.room.Width = Slider1.Value * 20 + 1
.room.Height = Slider2.Value * 20 + 1
.robot.Width = Slider4.Value * 20 / 4 - 1
.robot.Height = Slider3.Value * 20 / 4 - 1
.rDest.Width = Slider4.Value * 20 / 4 - 1
.rDest.Height = Slider3.Value * 20 / 4 - 1
```

După trimiterea valorilor sliderelor elemntelor componente ale scenei statice se va realiza şi o trimitere a unei serii de comenzi ce au ca scop ascunderea obstacolelor de bază în afara scenei statice:

```
'Ascund obiectele si le mut in afara scenei
For n = 0 To 6
.cerc(n).Tag = "0"
.patrat(n).Tag = "0"
.cerc(n).Visible = False
.cOver(n).Visible = False
.pOver(n).Visible = False
.patrat(n).Visible = False
.cerc(n).Left = -1000
.cerc(n).Top = -1000
.cOver(n).Left = -1000
.cOver(n).Top = -1000
.pOver(n).Left = -1000
.pOver(n).Top = -1000
.patrat(n).Left = -1000
.patrat(n).Top = -1000
Next
```

În vederea afişării obstacoleor se va realiza clonarea acestora sub forma unei mulţimi de obstacole, fiecacare din acestea luând o poziţionare aleatorie în scena statică şi o dimensiune cuprinsă între valoarea minimă şi valoarea maximă setată:

- pentru obstacolele de tip pătrat rezultă următorul cod:

```
'Pun patratele aleator in camera fara sa depasesc dimensiunea maxima dimMax
For p = 1 To Slider6.Value
dimMax = Rot(Round((Rnd * (Slider7.Value / 2)) + 1, 1))
‘atribui dimensiunile patratelor prin convertirea acestora din metri in pixeli
.patrat(p - 1).Width = dimMax * 20 - 1
.patrat(p - 1).Height = dimMax * 20 - 1
.pOver(p - 1).Width = dimMax * 20 - 1
.pOver(p - 1).Height = dimMax * 20 - 1
```

'realizez pozitionarea obstacolelor de tip patrat avand grija ca acestea sa se afle in interiorul scenei statice
*vx = Int(Rnd * (.room.Width - .patrat(p - 1).Width - .robot.Width * 3) + .robot.Width * 1.5)*
*vy = Int(Rnd * (.room.Height - .patrat(p - 1).Height - .robot.Width * 3) + .robot.Width * 1.5)*

.pOver(p - 1).Left = vx
.pOver(p - 1).Top = vy
.pOver(p - 1).Visible = True

.patrat(p - 1).Left = vx
.patrat(p - 1).Top = vy
.patrat(p - 1).Visible = True
.patrat(p - 1).ZOrder (0)
Next p

- pentru obstacolele de tip cerc rezultă următorul cod:

'Pun cercurile aleator in camera fara sa depasesc dimensiunea maxima dimMax
For c = 1 To Slider5.Value
*dimMax = Rot(Round((Rnd * (Slider7.Value / 2)) + 1, 1))*
*.cerc(c - 1).Width = dimMax * 20 - 1*
*.cerc(c - 1).Height = dimMax * 20 - 1*
*.cOver(c - 1).Width = dimMax * 20 - 1*
*.cOver(c - 1).Height = dimMax * 20 - 1*

'realizez pozitionarea obstacolelor de tip cerc avand grija ca acestea sa se afle in interiorul scenei statice
*vx = Int(Rnd * (.room.Width - .cerc(c - 1).Width - .robot.Width * 4) + .robot.Width * 2)*
*vy = Int(Rnd * (.room.Height - .cerc(c - 1).Height - .robot.Width * 4) + .robot.Width * 2)*

.cOver(c - 1).Left = vx
.cOver(c - 1).Top = vy
.cOver(c - 1).Visible = True

.cerc(c - 1).Left = vx
.cerc(c - 1).Top = vy
.cerc(c - 1).Visible = True
.cerc(c - 1).ZOrder (0)
Next c

Datorită poziţionării aleatorii a obstacolelor a fost necesară implementarea unui sistem de scanare a acestora. În contextul acestui fapt, odată cu apăsarea de către utilizatorul al butonului "Setează" va avea loc şi iniţializarea scanerului. Deaoarece scaneru este compus din două linii ce se intersectează, în cadrul iniţializării trebuiesc specificate coordonatele în funcţie de lungimea şi lăţimea scenei statice pe care se desenează cele două linii.

```
'Initializez Scannerul
.xScan.X1 = 0
.xScan.Y1 = 10
.xScan.X2 = .room.Width
.xScan.Y2 = 10
.xScan.Visible = True

.yScan.X1 = 10
.yScan.Y1 = 0
.yScan.X2 = 10
.yScan.Y2 = .room.Height
.yScan.Visible = True

.Shape1.Left = .yScan.X1 - (.Shape1.Width / 2)
.Shape1.Top = .xScan.Y1 - (.Shape1.Height / 2)
.Shape1.Visible = True
.Timer1.Interval = 1000 / Slider8.Value

.yScan.ZOrder (0)
.xScan.ZOrder (0)
.Shape1.ZOrder (0)

End With
```

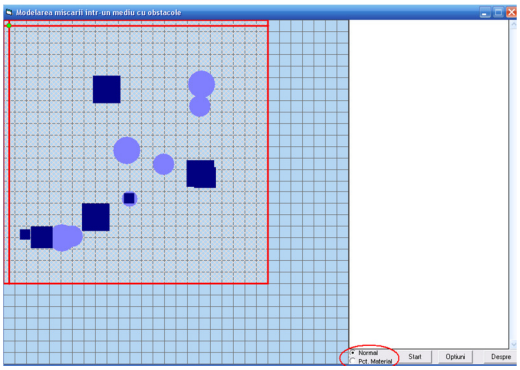


Figura 8. Mediul de lucru generat

Temă propusă:

1. În cadrul programului Visual Basic să se genere mediul desimulare pentru un robot cu baza fixă, a cărui spațiu este de mișcare este limitat de un arc de cerc de 180^0.

Lucrarea nr. 2

Grafuri modelare și simulare – Partea I

1. Tema lucrării: Introducere în grafuri. Utilizarea grafurilor pentru rezolvarea problemei drumului cel mai scurt
2. Scopul lucrării: prezentarea introductivă a noțiunilor de bază pentru grafuri și simulare problemei drumului cel mai scurt în mediul de programare Visual Basic.
3. Consideraţii teoretice
3.1 Definiții de bază
3.2 Problema drumului cel mai scurt

3.1 Definiții de bază

Un graf este definit prin două seturi de simboluri: noduri şi arcuri. Nodurile sau vârfurile sunt definite printr-un set de puncte V. Arcul constă dintr-o pereche ordonată de puncte și reprezint o direcție posibilă a acțiunii care se poate produce între acele puncte. De exemplu, dacă graful conține un arc (j, k), mișcarea posibilă este de la nodul j la nodul k. Pentru acest nod, j este nod inițial (extremitatea inițială), iar k nod terminal (extremitatea terminală).

De exemplu să presupunem că nodurile 1, 2, 3 şi 4 din figura 9 reprezintă 4 puncte de oprire ale unui robot şi fiecare arc reprezintă un drum (cu sens unic) care leagă două puncte.

Pentru acest graf:

- mulţimea vârfurilor (nodurilor) este V={1, 2, 3, 4}
- mulţimea arcelor este A={(1, 2), (2, 3), (3, 4), (4, 3), (4,1)}

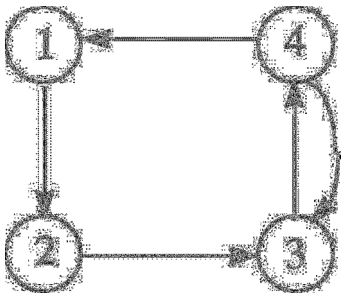


Fig. 9 – exemplu de graf

O succesiune de arce cu proprietatea că fiecare arc are doar un singur nod comun cu arcul anterior se numeşte lanţ.

Un drum este o succesiune de arce (un lanţ) astfel încât nodul reţea terminal al fiecărui arc coincide cu nodul iniţial al arcului următor.

Un drum finit care revine la punctul său de plecare se numeşte circuit. Dacă vârfurile prin care trece sunt toate distincte, circuitul este elementar, iar dacă arcele sunt toate distincte, circuitul este simplu.

În cazul exemplului menționat mai sus, (1, 2)-(2, 3)-(4, 3) este un lanţ dar nu este un drum. (1, 2)-(2, 3)-(3, 4) este un lanţ şi un drum. Drumul (1, 2)-(2, 3)-(3, 4) reprezintă o cale de a merge de la nodul 1 la nodul 5. (1, 2)-(2, 3)-(3, 4)-(4, 1) este un circuit.

3.2 Problema drumului cel mai scurt

În cazul problemelor la care se doreşte determinarea drumului cel mai scurt, se poate asocia fiecărui arc al grafului câte o lungime.

Acest tip de problemă presupune determinarea celei mai scurte distanţe între un punct sau nod iniţial şi un nod final sau terminal într-o reţea de drumuri. De exemplu reţeaua din figura 10, în care distanţele sunt date în minute necesare unui robot să parcurgă pe fiecare ramură un traseu printre obstacole.

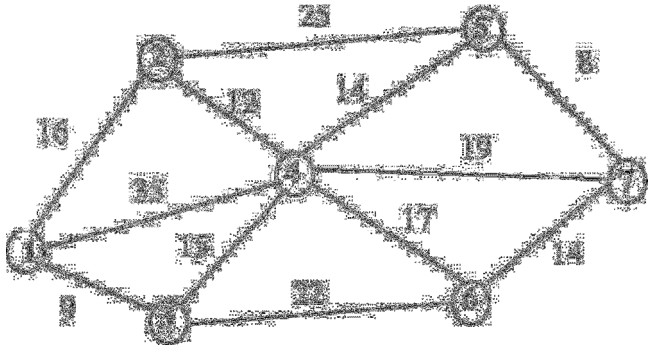


Figura 10 – Traseu robot

Se începe de la nodul iniţial, 1 şi se determină cea mai scurtă distanţă până la nodurile cu care acesta este conectat direct, noduri numite adiacente, respectiv nodurile 2, 3 şi 4. Cea mai scurtă distanţă este de la 1 la 3, de 9 minute. Astfel nodul 3 devine nod din mulţimea nodurilor permanente, mulţime care indică faptul că s-a găsit cea mai scurtă rută până la aceste noduri. Nodul 1 este automat parte a mulţimii nodurilor permanente deoarece nu există alt drum către acesta. Astfel rezultă situaţia din figura 11.

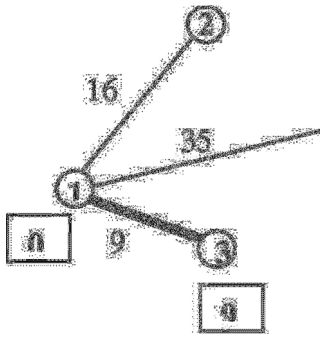


Fig. 11 – Determinarea ceeli mai scurte distanțe până la nodurile cu care este conectat direct

În continuare se determină toate nodurile direct conectate cu nodurile din mulţimea permanentă, respectiv cu nodurile 1 şi 3. Acestea sunt 2, 4 şi 6. Rezultă situaţia din figura 12.

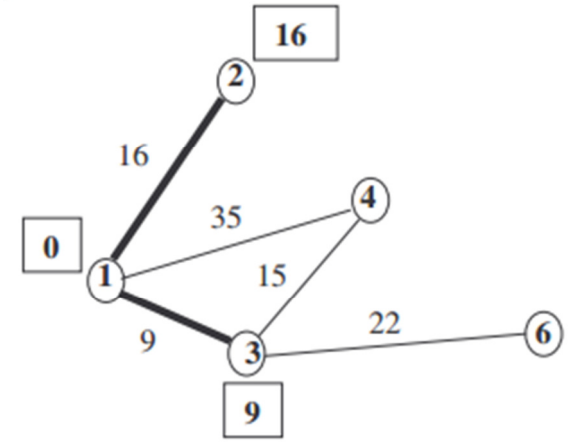


Fig 12 - Determinarea tutror nodurilor direct conectate cu nodurile din mulțime permanentă

Noua mulţime a nodurilor permanente este acum {1,2,3}, deoarece ramura 1 - 3 este acum cea mai scurtă, ceea ce înseamnă că am găsit cele mai scurte drumuri între aceste noduri. În continuare se găsesc noile noduri care sunt direct legate de nodurile din mulţimea permanentă. Acestea sunt 4, 5 şi 6. Astfel rezultă situaţia din figura 13:

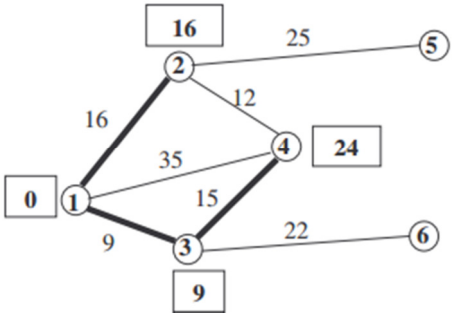


Figura 13 a

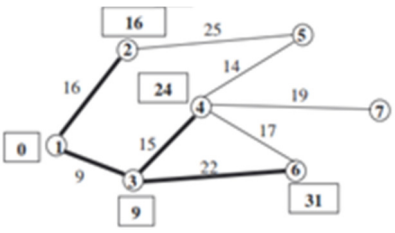


Figura 13 b

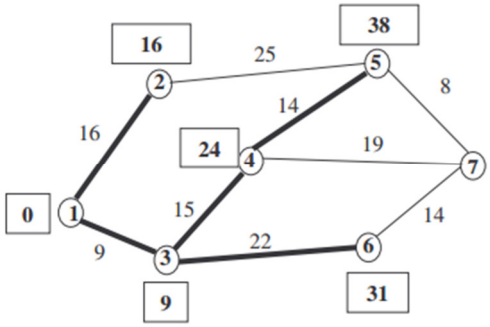


Fig 14 a

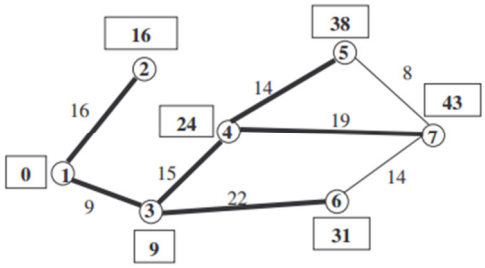


Fig 14 b

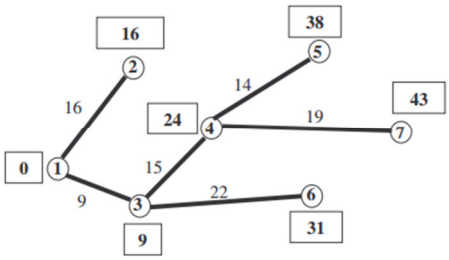


Fig 15

Algoritmul este următorul:

Pasul 1. Se selectează nodul cu cea mai scurtă rută directă către nodul iniţial;
Pasul 2. Se stabilişte mulţimea permanentă că fiind construită din nodul iniţial şi nodul selectat la pasul 1;
Pasul 3. Se stabilesc toate nodurile direct conectate mulţimii nodurilor permanente;
Pasul 4. Se selectează nodul cu cea mai scurtă ramură de la grupul de noduri direct conectate cu mulţimea nodurilor permanente;
Pasul 5. Se repetă paşii 4 şi 5 până când toate nodurile s-au adăugat mulţimii nodurilor permanente.

4. Aplicație

Să se implementeze în mediul de programare algoritmul pentru problema drumului cel mai scurt în cadrul programului din primul laborator.
Rezolvare

Se va implementa în cadrul programului realizat în laboratorul 1 codul următor:

```
Namespace cases.algorithms.traversal

    Public  Class Graph
```

```
                Public graphNodes AS System.Collections.ArrayList

                Public Overridable Sub traverse()

                System.Console.WriteLine("Preparing to visit the graph ...")

                Dim nodesSize AS Integer = graphNodes.Count

                If   True

                Dim i AS Integer = 0

                While i<nodesSize

                Dim node AS cases.algorithms.traversal.Node = CType
(graphNodes.Item(i),cases.algorithms.traversal.Node)

                traverseNode(node)

                i = i + 1

                End While

                End If

                System.Console.WriteLine("Graph Visited Successfully")

                End Sub

                Public Overridable Sub traverseNode(n As cases.algorithms.traversal.Node)

                If n.visitStatus<> 0 Then

                End If

                System.Console.WriteLine("Visiting node : " + n.name)

                n.visitStatus = 1

                Dim adjacentNum AS Integer = n.adjacentNodes.Count
                        System.Console.WriteLine("Count = " +
n.adjacentNodes.Count.toString())

                If   True
```

```
                Dim i AS Integer = 0

                While i<adjacentNum
                Dim node AS cases.algorithms.traversal.Node = CType
(n.adjacentNodes.Item(i),cases.algorithms.traversal.Node)

               traverseNode(node)

               i = i + 1

                End While

                End If

               System.Console.WriteLine("Visited Successfully" + n.name)

               End Sub

                        Sub  New ()
                  graphNodes = new System.Collections.ArrayList()
                End Sub

               End Class
    End Namespace
```

După rularea programului se va opține un drum asemănător celui din figura 16.

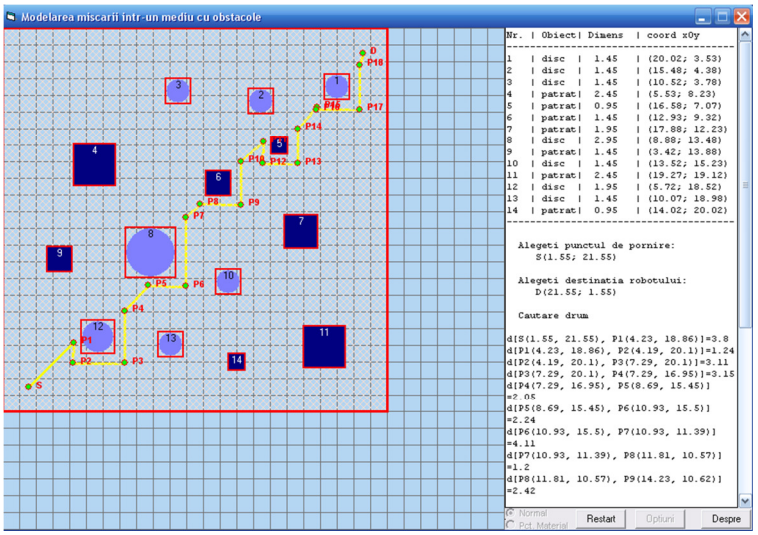
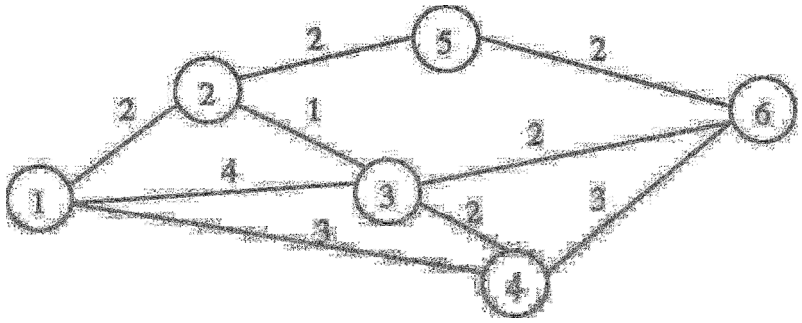


Figura 14 – Identificarea drumului cu ajutorul grafurilor

Temă propusă

Să se realizeze o aplicație în cadrul programului Visual Basic pentru simularea drumului cel mai scurt având ca model graful din imagine:



Lucrarea nr. 3
Grafuri modelare și simulare – Partea a II-a

1. Tema lucrării: Introducere în grafuri. Utilizarea grafurilor pentru rezolvarea problemei arborelui de deschidere minimă
2. Scopul lucrării: Prezentarea problemei arborelui de deschidere minimă și simularea acestuia în mediul de programare Visual Basic.
3. Consideraţii teoretice

3.1 Problema arborelui de deschidere minimă

Problema are ca obiectiv conectarea tuturor nodurilor într-o reţea astfel încât lungimea totală a ramurilor să fie minimă. Astfel toate nodurile sunt conectate într-o reţea cu distanţă minimă.

Aplicație:

Se doreşte să unească nodurile din reţeaua de mai jos, noduri care reprezintă zone de conectare internet, astfel încât să folosească cât mai puţin cablu. Figura redă totalitatea căilor posibile şi lungimea necesară a cablului (în km) pe fiecare cale.

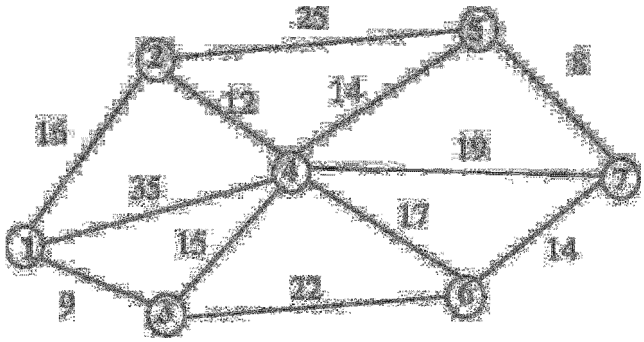


Figura 15 –Graful zonelor

Rezolvare:

Pentru a rezolva problema se alege un nod de pornire. De obicei se alege primul nod, dar acest lucru nu este obligatoriu. Începând de la nodul 1 se alege cel mai apropiat nod care să se alăture nodurilor de pe arborele cu lungime minimă. Cea mai scurtă ramură este de la 1 la 3, cu lungime de 9 km. Această ramură este indicată în figura 16 prin linia îngroşată.

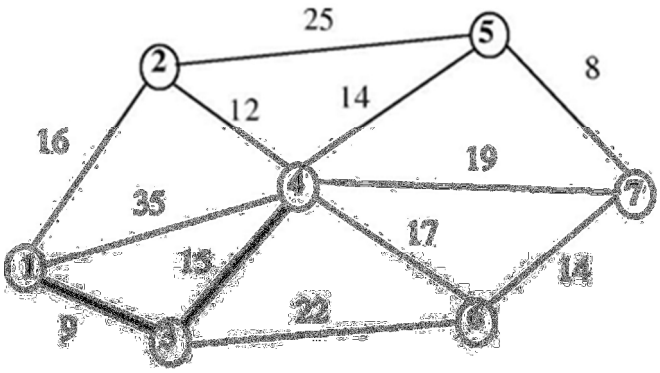


Figura 16

Procesul se repetă căutând cel mai apropiat nod de nodurile 1, 3 şi 4. Următorul nod care se adaugă arborelui este nodul 2. Repetând procedura, urmează apoi nodul 5 să se alăture arborelui, apoi nodul 7 şi în final nodul 6. Rezultatatele intermediare şi apoi rezultatul final sunt prezentate în figura 17.

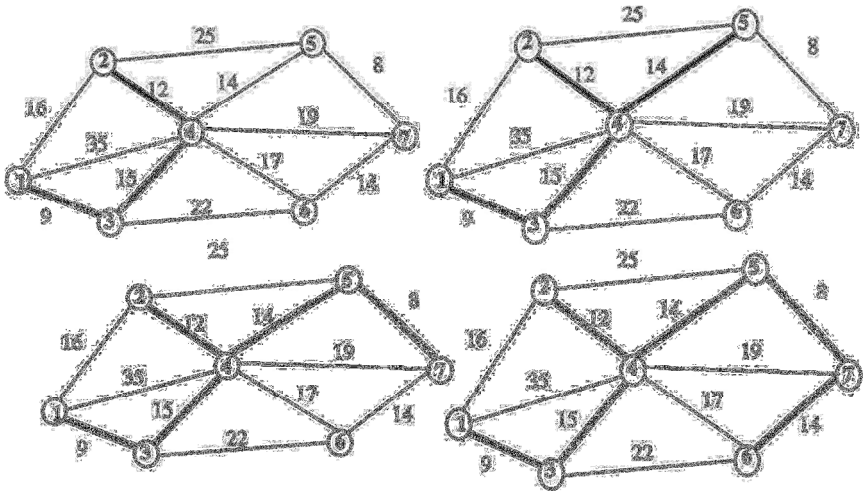


Figura 17 – Rezultatele intermediare și rezultatul final

Codul algoritmului este urmatorul:

```
Namespace Algo

    Public Class Graph

        '//fields
        Private needsCalculate As Boolean = True

        '//properties
        Private _verticies As VertexCollection
        Public ReadOnly Property Verticies() As VertexCollection
            Get
                If Me._verticies Is Nothing Then
                    Me._verticies = New VertexCollection(Me)
```

```
                End If
                Return Me._verticies
            End Get
        End Property

        Public ReadOnly Property Finished() As Boolean
            Get
                Return Me.Verticies.Finished
            End Get
        End Property

        Private _edges As EdgeCollection
        Public ReadOnly Property Edges() As EdgeCollection
            Get
                If Me._edges Is Nothing Then
                    Me._edges = New EdgeCollection(Me)
                End If
                Return Me._edges
            End Get
        End Property

        '//methods
        Private Sub Algo(ByVal initialVertex As Vertex)
            If initialVertex Is Nothing Then
                Throw New ArgumentNullException("initialVertex")
            End If
            If Not Me.needsCalculate Then
                Return
            End If
            '//initialize starting values
            For Each vertex As Vertex In Me.Verticies
                vertex.SetDistance(Double.PositiveInfinity)
                vertex.SetVisited(False)
            Next
            initialVertex.SetDistance(0.0R)
            Try
                '//calculate shortest paths
                Me.Algo(Me, initialVertex)
            Catch ex As Exception
                Throw New AlgorithmException("The graph, vertex, or edges are invalid.
Either they are not all connected, or there are edges missing.", ex)
            End Try
        End Sub

        Private Sub Algo(ByVal graph As Graph, ByVal current As Vertex)
            '//loop each neighboring vertex
            For Each neighbor In current.Neighbors
                If Not neighbor.Visited Then
                    '//vertex has not been visited yet
                    Dim edge = graph.Edges(current, neighbor)
                    '//get the distance between the verticies
                    Dim distance = (current.Distance + edge.Distance)
                    '//check if the distance is smaller than it's previous distance
                    If distance < neighbor.Distance Then
                        neighbor.SetDistance(distance)
```

```
                        '//sets the vertex that you would follow to get to this
neighboring one
                        neighbor.PreviousVertex = current
                    End If
                End If
            Next
            '//mark Vertex visited
            current.SetVisited(True)
            If Not graph.Finished Then
                '//graph has unvisited verticies
                Me.Algo(graph, graph.FindShortestVertex())
            End If
            Me.needsCalculate = False
        End Sub

        Private Function FindShortestVertex() As Vertex
            Dim result As Vertex = Nothing
            Dim min = Double.PositiveInfinity
                        For Each vertex As Vertex In Me.Verticies
                If Not vertex.Visited Then
                    If vertex.Distance < min Then
                        '//set the current smallest vertex
                        min = vertex.Distance
                        result = vertex
                    End If
                End If
            Next
            '//after all distances are evaluated return result
            Return result
        End Function

        Public Function AddVertex(ByVal key As String) As Vertex
            Dim vertex = New Vertex(Me, key)
            Me.Verticies.Add(vertex)
            Return vertex
        End Function

        Public Sub RemoveVertex(ByVal vertex As Vertex)
            Me.Verticies.Remove(vertex)
        End Sub


        Public Function AddEdge(ByVal first As Vertex, ByVal second As Vertex, ByVal
distance As Double) As Edge
            Dim edge = New Edge(Me, first, second, distance)
            Me.Edges.Add(edge)
            Return edge
        End Function
        Public Function AddEdge(ByVal firstKey As String, ByVal secondKey As String,
ByVal distance As Double) As Edge
            Dim edge = New Edge(Me, Me.Verticies(firstKey), Me.Verticies(secondKey),
distance)
            Me.Edges.Add(edge)
            Return edge
        End Function
        Public Sub RemoveEdge(ByVal edge As Edge)
```

```
            Me.Edges.Remove(edge)
        End Sub

        Public Sub Calculate(ByVal initialVertex As Vertex)
            Me.Algo(initialVertex)
        End Sub

        Public Sub Reset()
            '//clear all contents
            Me.Verticies.Clear()
            Me.Edges.Clear()
            '//notify graph that it will need to recalculate
            Me.NotifyRecalculate()
        End Sub
        Public Sub Save(ByVal fileName As String)

            '//create the document
            Dim doc = <?xml version="1.0" encoding="utf-8"?>
                      <root vertexCount=<%= Me.Verticies.Count %> edgeCount=<%=
Me.Edges.Count %>></root>

            '//loop all verticies
            For Each vertex As Vertex In Me.Verticies
                doc.Root.Add(<vertex>
                                 <key><%= vertex.Key %></key>
                             </vertex>)
            Next

            '//loop all edges
            For Each edge As Edge In Me.Edges
                doc.Root.Add(<edge>
                                 <firstKey><%= edge.First.Key %></firstKey>
                                 <secondKey><%= edge.Second.Key %></secondKey>
                                 <distance><%= edge.Distance %></distance>
                             </edge>)
            Next

                 doc.Save(fileName)
        End Sub

        Public Shared Function Load(ByVal fileName As String) As Graph
            '//try to read the Xml file
            Try
                '//load the Xml and create the new graph
                Dim doc = XDocument.Load(fileName)
                Dim graph = New Graph()

                '//get all verticies
                For Each node In doc.<root>.<vertex>
                    graph.AddVertex(node.<key>.Value())
                Next

                '//get all edges
                For Each node In doc.<root>.<edge>
                    graph.AddEdge(node.<firstKey>.Value(), node.<secondKey>.Value(), _
                                  Convert.ToDouble(node.<distance>.Value()))
```

```
                Next

                Return graph
            Catch ex As Exception
                Return Nothing
            End Try
        End Function

        Public Function GetDistance(ByVal initialVertex As Vertex, ByVal endingVertex As
Vertex) As Double
            Me.Algo(initialVertex)
            Return endingVertex.Distance
        End Function
        Public Function GetPath(ByVal initialVertex As Vertex, ByVal endingVertex As
Vertex) As String
            Return Me.GetPath(initialVertex, endingVertex, False)
        End Function

        Public Function GetPath(ByVal initialVertex As Vertex, ByVal endingVertex As
Vertex, ByVal reverse As Boolean) As String
            Me.Algo(initialVertex)
            Dim verticies = Me.GetVerticies(initialVertex, endingVertex, reverse)
            Dim builder = New Text.StringBuilder()
            For vertex = 0 To verticies.Count - 2
                builder.AppendFormat("{0} -> ", verticies(vertex).Key)
            Next
            builder.Append(verticies(verticies.Count - 1).Key)
            Return builder.ToString()
        End Function

        Public Function GetElapsed(ByVal initialVertex As Vertex) As TimeSpan
            '//notify graph that it will need to recalculate
            Me.NotifyRecalculate()
            Dim watch = Stopwatch.StartNew()
            '//run the algorithm
            Me.Algo(initialVertex)
            '//return results
            watch.Stop()
            Return watch.Elapsed
        End Function


        Public Function GetVerticies(ByVal initialVertex As Vertex, ByVal endingVertex As
Vertex) As Vertex()
            Return Me.GetVerticies(initialVertex, endingVertex, False)
        End Function


        Public Function GetVerticies(ByVal initialVertex As Vertex, ByVal endingVertex As
Vertex, ByVal reverse As Boolean) As Vertex()
            Me.Algo(initialVertex)
            Dim current = endingVertex
            Dim verticies = New List(Of Vertex)()
            Do Until current Is Nothing
                verticies.Add(current)
                current = current.PreviousVertex
```

```
            Loop
            If Not reverse Then
                verticies.Reverse()
            End If
            Return verticies.ToArray()
        End Function

        Friend Sub NotifyRecalculate()
            Me.needsCalculate = True
        End Sub

    End Class

    End Namespace
```

Temă propusă

Se doreşte să unească nodurile din reţeaua de mai jos, noduri care reprezintă zone de conectare internet, astfel încât să folosească cât mai puţin cablu. Figura redă totalitatea căilor posibile şi lungimea necesară a cablului (în km) pe fiecare cale.

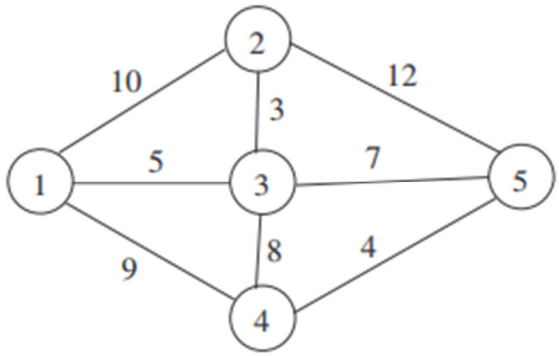


Figura 18

Lucrarea nr. 4
Modelarea mişcării – Partea I

1. Tema lucrării: Introducere în planificarea şi modelarea mişcării sistemelor mecatronice
2. Scopul lucrării: Reprezentarea mişcării sistemelor mecatronice de tip robot în mediul virtual cu ajutorul limbajului Visual Basic.
3. Consideraţii teoretice

3.1 Planificarea dinamică a mişcării sistemelor mecatronice
3.2 Metode de planificare a mişcării roboţilor mobili într-un spaţiu de lucru populat cu obstacole fixe şi/sau mobile
3.3 Planificarea mişcării roboţilor articulaţi, mobili, printre obstacole
3.4 Utilizarea algoritmului A* pentru simularea mişcării

3.1 Planificarea dinamică a mişcării sistemelor mecatronice

Un sistem mecatronic trebuie să realizeze diferite sarcini fără a-i fi specificată fiecare acţiune care urmează să fie realizată. Pentru a realiza un sistem mecatronic autonom, este necesar să fie sintetizate multe tehnici, inclusiv unele elemente de inteligenţă artificială. În mod obişnuit, robotul trebuie să obţină informaţii din lumea înconjurătoare folosind senzori vizuali şi auditorii, să elaboreze un plan pentru executarea sarcinii date, să rezolve fenomenele neaşteptate care vin fie din mediul exterior, fie de la robot şi să înveţe din experienţă pentru a-şi îmbunătăţi performanţele.

Problema planificării mişcării unui sistem mecatronic mobil este aceea a găsirii unei mişcări pentru un sistem care trebuie să se deplaseze de la o configuraţie dată, la o destinaţie stabilită, într-un mediu care conţine o mulţime de obstacole prestabilite, astfel încât robotul să nu intre în coliziune de nici unul din acestea. Într-o problemă concretă, obstacolele nu sunt întotdeauna statice, iar sistemul nu poate fi modelat ca un singur obiect rigid, precum în cazul problemei de bază a planificării. Este evident că un sistem mecatronic care se mişcă printre obstacolele mobile este capabil de performanţe mult mai mari şi de o serie de sarcini mult mai complexe. Această teorie are în vedere planificarea mişcării în medii de timp variabil unde atât obstacolele, cât şi destinaţia, sunt în mişcare.

Abilitatea ocolirii obstacolelor este indispensabilă pentru orice sistem mecatronic real. Se consideră, spre exemplu, un robot tip maşină ce se deplasează de-a lungul unui drum stabilit. Sistemul senzorial al robotului poate dintr-o dată să depisteze un obiect care îi taie drumul. În cazul acesta ar trebui să fie capabil să producă şi să execute o mişcare pentru a evita cu siguranţă obiectul. De asemenea, într-un mediu echipat cu calculatoare, unui robot i se poate cere să ajungă la un obiect aflat în mişcare, în timp ce evită obstacolele. După cum se poate vedea din aceste exemple, abilitatea de a se descurca cu obstacolele în mişcare sporeşte semnificativ potenţialul capacităţilor şi şirul aplicaţiilor unui robot inteligent.

Din punct de vedere tehnic, prezenţa obstacolelor în mişcare conduce la apariţia unor noi aspecte privind problema planificării mişcării. De exemplu, când este implicată mişcarea obstacolelor, cel mai scurt drum nu se face întotdeauna în cel mai scurt timp.

Optimizarea unei probleme de planificare, în contextul amintit, presupune un consum foarte mic de energie, dar, în acelaşi timp, trebuie să se acorde atenţie şi studiului vitezelor şi acceleraţiilor robotului mobil. Astfel problema planificării mişcării printre obiecte (obstacole)

mobile este în mai multe feluri diferită şi mai complexă decât problema planificării mişcării cu obstacole staţionare.

3.2 Metode de planificare a mișcării roboților mobili într-un spațiu de lucru populat cu obstacole fixe și/sau mobile

Problema planificării traiectoriilor în prezenţa obstacolelor poate fi enunţată astfel: fiind dat un spaţiu de lucru populat cu obstacole cunoscute prin frontierele lor, trebuie să se determine o traiectorie fără coliziuni cu obstacolele, aducând obiectele mobile din configuraţia iniţială în cea finală.

Problema poate fi abordată în două moduri (global sau local), de unde rezultă două tipuri de metode de planificare: globale şi locale. Aplicarea unei metode globale necesită cunoaşterea completă "a priori" a spaţiului de lucru, modelarea corespunzătoare a spaţiului liber, cercetarea tuturor traiectoriilor posibile şi selectarea unei anumite traiectorii corespunzătoare unui criteriu de cost minim. O astfel de metodă garantează existenţa sau inexistenţa unei soluţii. De asemenea, metodele globale de planificare se pot adapta uşor la programarea off-line.

Aplicarea unei metode locale necesită cunoaşterea parţială a spaţiului de lucru. O astfel de metodă nu garantează atingerea configuraţiei finale, dar prezintă avantajul unei bune adaptări în timp real.

În ambele cazuri, globale sau locale, rezolvarea problemei de planificare presupune rezolvarea unor probleme de natură geometrică (geometrie pură) sau de geometrie combinată cu cinematică şi/sau dinamică. În astfel de situaţii se utilizează cu precădere rezultatele geometriei algoritmice.

Pentru un robot la un post de lucru fix, mobilele pentru care trebuie să se determine traiectoriile fără coliziuni sunt constrânse între ele de anumite legături. Astfel, o problemă de planificare este deci "a priori" complexă. Totuşi, prin transformări adecvate, operate asupra obstacolelor, o problemă de planificare poate fi redusă la o problemă de "navigare" a unui robot punctiform, considerat ca un mobil liber care evoluează printre obstacolele transformate. Astfel, planificarea traiectoriilor unui robot comportă, în general, două etape:

- modelarea spaţiului de lucru, transformarea obstacolelor (determinarea C-obstacole-lor) astfel încât robotul poate fi considerat ca un punct material;
- cercetarea unei traiectorii, fără coliziune, pentru acest punct.

În cadrul unui sistem CAD/CAM dotat cu un algoritm de cercetare a spaţiului liber se poate aplica planificarea traiectoriilor fără coliziuni, în mod interactiv cu operatorul, prin vizualizarea acestui spaţiu.

În general, aplicarea unei metode de planificare trebuie să satisfacă anumite restricţii, ca de exemplu: drumul cel mai sigur, drumul cel mai scurt, etc.

3.3 Planificarea mişcării roboţilor articulaţi, mobili, printre obstacole

Se consideră că spaţiul de lucru al robotului este populat cu obstacole staţionare. Se presupune că toate informaţiile privind obstacolele sunt complet cunoscute. Se doreşte să se planifice o mişcare pentru robot de la o configuraţie de start dată, la o configuraţie finală, în prezenţa acestor obstacole. Această problemă este "problema planificării dinamice a mişcării".

Când spaţiul de lucru al robotului este populat doar cu obstacole staţionare, problema este "problema planificării statice a mişcării".

În problema planificării statice a mişcării, se determină un drum pentru robot astfel încât, în tot timpul mişcării drumul nu intersectează nici unul din obstacolele staţionare. Termenul "traiectorie" este folosit uneori deoarece sunt necesare şi informaţii despre viteza şi acceleraţia robotului de-a lungul drumului. Pentru început este determinat un drum posibil, apoi, ţinând seama de consideraţiile dinamice, cum ar fi viteza şi acceleraţia, se caută optimizarea acestuia. Într-un mediu dinamic, un drum liber la un moment dat poate să nu fie liber pentru o altă perioadă de timp. De aceea, în acest caz, drumul trebuie să fie specificat ca funcţie de timp de la început. În majoritatea cazurilor, criteriul de optimizare constă în minimizarea timpului de parcurgere a drumului de către robot, deoarece factorul de timp joacă un rol crucial în mediile dinamice. Multe dintre problemele ce au de-a face cu obstacolele staţionare pot fi rezolvate într-un timp polinomial, în timp ce câteva probleme de bază, care sunt cu obstacole în mişcare, nu sunt rezolvabile într-un timp polinomial. Totuşi, există un număr de algoritmi care au fost propuşi să rezolve problemele planificării dinamice a mişcării. Aceşti algoritmi lucrează în domenii limitate şi produc o mişcare într-un timp polinomial.

Kant şi Zucker au propus primul algoritm practic pentru planificarea mişcării printre obstacolele în mişcare. Problema de planificare este divizată în două subprobleme:

- problema planificării drumului printre obstacole staţionare
- problema planificării vitezei de-a lungul unui drum fixat.

În problema planificării drumului printre obstacole staţionare se planifică un drum printre obstacole staţionare în timp ce toate obstacolele în mişcare sunt ignorate. În problema planificării vitezei de-a lungul unui drum fixat este folosit un graf ordonat pentru a defini regiuni printre care un robot nu poarte să treacă când urmează drumul estimat. Poziţia acestor regiuni influenţează alegerea vitezei.

După cum Kant şi Zucker au arătat, acestea sunt cazurile în care această apropiere eşuează în producerea unei mişcări chiar şi atunci când acolo există mişcare. Acest lucru se întâmplă când o parte a unui obstacol coincide cu drumul fixat în prima parte. Aceasta pentru că drumul este fixat în a doua etapă; de aceea robotului nu i se permite să înconjoare obstacolul în mişcare. De asemenea, deoarece drumul este fixat în prima etapă, metoda propusă nu poate încorpora cu uşurinţă un punct de destinaţie aflat în mişcare.

Problema planificării dinamice a mişcării interacţionează în unele cazuri cu o altă categorie de probleme, numite "problemele mişcării coordonate". O astfel de problemă presupune planificarea mişcărilor coordonate pentru mai mulţi roboţi. Această problemă este dinamică în sensul că cel ce planifică trebuie să ia în considerare mişcările mai multor roboţi în acelaşi timp (Problema planificării mişcării roboţilor multipli), unde mai mulţi roboti se mişcă independent în acelaşi spaţiu de lucru, printre obstacole staţionare. O abordare simplă conceptual a acestei probleme este de a considera diferiţii roboţi ca şi componente ale unui robot compus şi de a planifica un drum liber în spaţiul configuraţiilor acestui robot (planificare centralizată). Dificultatea acestei abordări constă în dimensiunea mare a spaţiului configuraţiilor, complexitatea temporală a planificării variind exponenţial cu această dimensiune. Altă abordare constă în planificarea unui drum pentru fiecare robot, mai mult sau mai puţin independent faţă de ceilalţi roboţi, şi considerând interacţiunile dintre drumuri (planificare decuplată). Această abordare reduce din calcul, dar se pierde din completitudinea algoritmului (fig. 19).

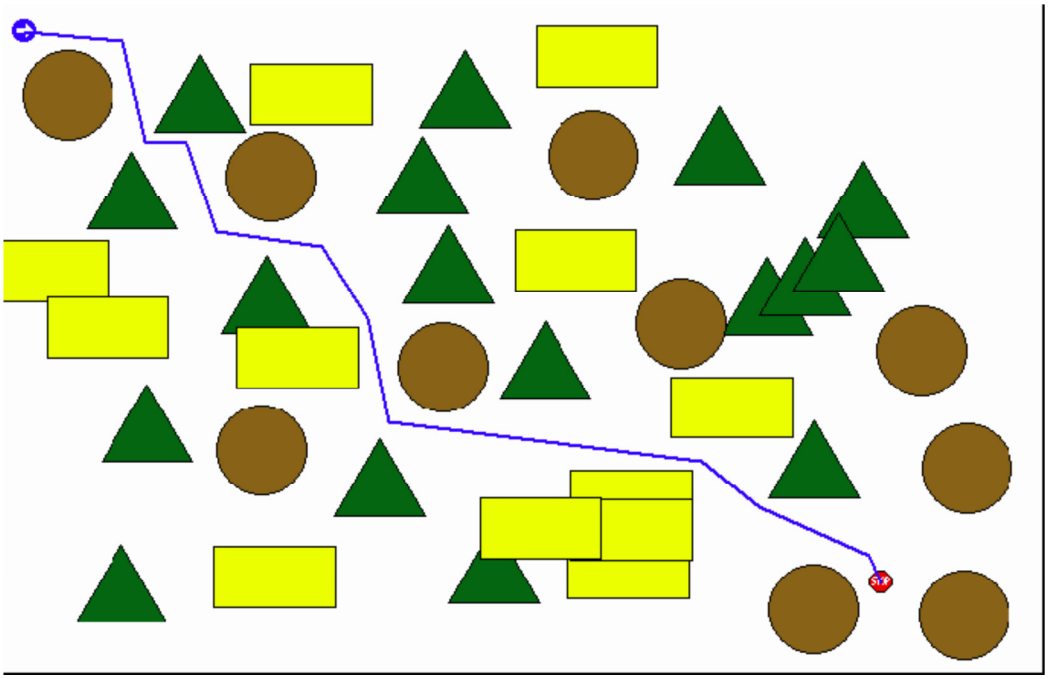


Figura 19 . Determinarea unui drum printre obstcaole fixe pentru un robot mobil [8]

3.4 **Utilizarea algoritmului A* pentru simularea mișcării**

Algoritmul A* este unul dintre cei mai folositi algoritmi în realizarea de căuatre a drumurilor într-un mediu virtual, deoarece este flexibil şi se integrează cu uşurinţă celor mai multor probleme.

Acest algoritm are capacitatea de a căuta un drum pe suprafețe mari.

Să presupunem că cineva doreşte să ajungă din punctul A în punctul B iar aceste două puncte sunt separate de un obstacol.

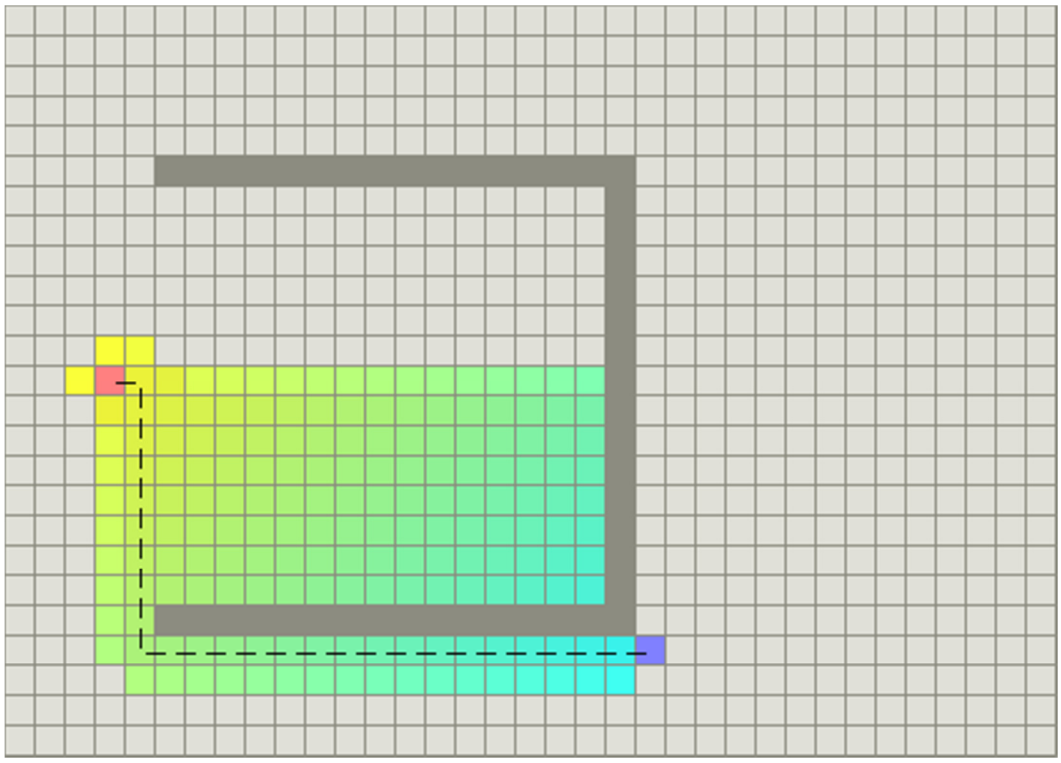


Figura 20. Căutarea drum,ului folosind algoritmul A* [10]

După cum se poate observa din figura 11, zona de căutare a fost împărţită într-o grilă de pătrate. Prin simplificarea zonei de căutare am realizat primul pas în căutarea drumului. Această metodă particulară reduce aria de căutare la un sistem cu două dimensiuni. Fiecare obiect din sistem reprzintă un pătrat din grilă, iar poziţia acestora este înregistrată ca find : zonă liberă sau zonă ocupată. Drumul este găsit prin deducerea căror pătrate din grilă trebuiesc folosite pentru a ajunge din punctul A în punctul B. Când drumul a fost găsit , acesta este parcurs trecând din centrul unui pătrat al grilei în centrul celuilalt pătrat până se ajunge la destinaţia dorită.Centrele acestor pătrate sun denumite noduri.

După simplificarea zonei de căutare în noduri, următorul pas este de a căuta un drum optim de la punctul A la punctul B.

Parcurgerea metodei de căutare este următoarea:

1) Începând din punctul de pornire A se vor trece toate pătratele apropiate de acest punct ce trebuiesc luate în considerare într-o listă.
2) Pasul doi este acela de a selecta toate pătratele din grilă care nu sunt ocupate de obstacole si adăugarea acestora în lista de la punctul 1
3) Pasul trei constă în adăugarea punctului A celor mai apropiate pătrate din grilă

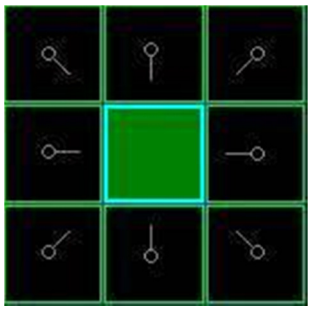


Figura 21 . Adăugarea pătratelor libere ce sunt apropiate de punctul A [9]

Pentru a putea să ne dăm seama care este cel mai bun pătrat din grilă când căutăm drumul, folosim următoparea ecuaţie:

$$F=G+H$$

G= costul mişcării de la punctul de pornire A către un pătrat apropiat din grilă
H=costul estimate al mişcării de la un pătrat dat al grilei la punctul final B
Astfel considerăm valoarea 10 pentru orizontala şi verticală pătratelor iar 14 pentru diagonala acestora. Dacă ne uităm la pătratul din (cel în care apare notatia G şi H) observăm că G ia valoarea 10 iar H valoarea 30.

Dacă dorim să facem verificarea putem calcula astfel primul pătrat apropiat de punctul de pornire va avea G = 10 (orizontala pătratului către punctul A va lua valoarea 14 conform explicaţiilor de mai sus) iar H=30 (adică se va lua valoare orizontalei tuturor pătratelor ce sunt până în punctul B).

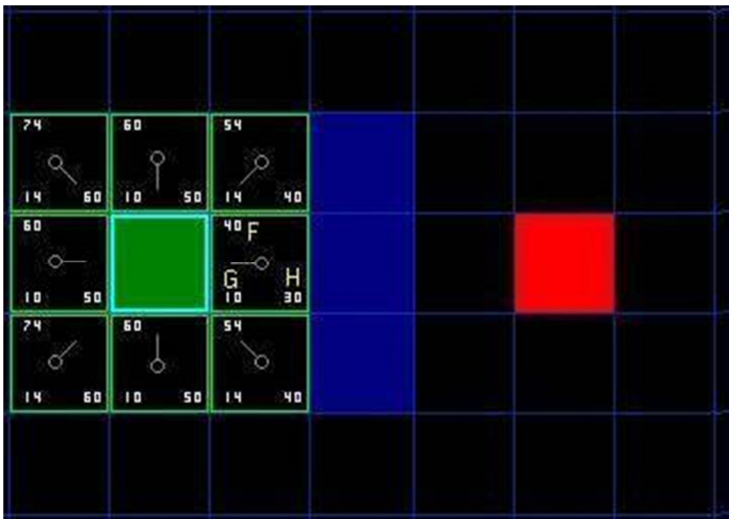


Figura 22. Adaugarea elementelor din ecuatia folosită [9]

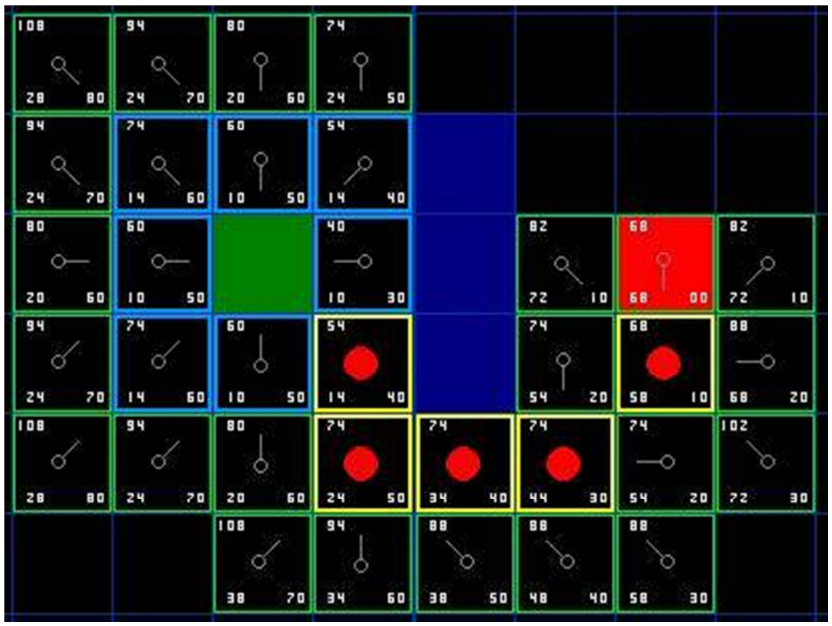


Figura 23. Parcurgerea drumului din punctul A în punctul B [9]

După cum se poate observa din figura 23 drumul optim de la punctul A la punctul B a fost găsit și se poate genera mișcarea.

Aplicație

Să se implementeze pentru programul realizat în laboratorul algoritmul A*.

Rezolvare

Se va implementa următorul cod:

```
Module AStarAlgorithms
    Public Structure Node
        Dim XCoord As Integer
        Dim YCoord As Integer
        Dim Cost As Integer
        Dim XParent As Integer
        Dim YParent As Integer
        Dim FValue As Integer

        ' Dim State As String
    End Structure

    Const StraightCost As Integer = 10
    Const DiagCost As Integer = 14
    Public ClosedList As New ArrayList
```

```
    Public OpenList As New ArrayList

    Public GoalReached As Boolean = False
    Public GoalNode As Node
    Public PathToFollow As New ArrayList


    Public Sub PathSearch(ByRef XStartCoord As Integer, ByRef YStartCoord As Integer, ByRef
XGoalCoord As Integer, ByRef YGoalCoord As Integer)
        Dim CurrentNode As New Node
        Dim WasClosed As Boolean = False
        Dim XHerDistance As Integer
        Dim YHerDistance As Integer
        Dim Heuristic As Integer
        Dim IsClosed As Boolean = False

        GoalNode.XCoord = XGoalCoord
        GoalNode.YCoord = YGoalCoord
        ClosedList.Clear()
        OpenList.Clear()

        CurrentNode.XCoord = XStartCoord
        CurrentNode.YCoord = YStartCoord
        CurrentNode.Cost = 0

        XHerDistance = Math.Abs(CurrentNode.XCoord - XGoalCoord)
        YHerDistance = Math.Abs(CurrentNode.YCoord - YGoalCoord)
        If XHerDistance > YHerDistance Then
            Heuristic = DiagCost * YHerDistance + StraightCost * (XHerDistance - YHerDistance)
        Else
            Heuristic = DiagCost * XHerDistance + StraightCost * (YHerDistance - XHerDistance)
        End If
        CurrentNode.FValue = Heuristic

        OpenList.Add(CurrentNode)


        Do While OpenList.Count > 0
            'set current node to something random and high for comparison
            CurrentNode.XCoord = 99
            CurrentNode.YCoord = 99
            CurrentNode.Cost = 99999
            CurrentNode.FValue = 999999999
            'Select the node on the openlist with the lowest F Value
            For Each TempNode As Node In OpenList
```

```
            If TempNode.FValue < CurrentNode.FValue Then CurrentNode = TempNode
        Next
        'Move Current Node from open list to closed list
        ClosedList.Add(CurrentNode)
        OpenList.Remove(CurrentNode)
        If GoalReached = True Then Exit Do
        'Test East
        Testing(CurrentNode, 1, 0, XGoalCoord, YGoalCoord)
        'Test North East
        Testing(CurrentNode, 1, 1, XGoalCoord, YGoalCoord)
        'Test North
        Testing(CurrentNode, 0, 1, XGoalCoord, YGoalCoord)
        'Test North West
        Testing(CurrentNode, -1, 1, XGoalCoord, YGoalCoord)
        'Test West
        Testing(CurrentNode, -1, 0, XGoalCoord, YGoalCoord)
        'Test South west
        Testing(CurrentNode, -1, -1, XGoalCoord, YGoalCoord)
        'Test South
        Testing(CurrentNode, 0, -1, XGoalCoord, YGoalCoord)
        'Test South East
        Testing(CurrentNode, 1, -1, XGoalCoord, YGoalCoord)
    Loop
    Dim Direction1 As String = Nothing
    Dim Direction2 As String = Nothing
    Dim UD As String
    If GoalReached = True Then
        CurrentNode = GoalNode
        Do While CurrentNode.XParent <> Nothing And CurrentNode.YParent <> Nothing
            If CurrentNode.XCoord - CurrentNode.XParent = 1 Then
                Direction2 = "E"
            ElseIf CurrentNode.XCoord - CurrentNode.XParent = -1 Then
                Direction2 = "W"
            End If
            If CurrentNode.YCoord - CurrentNode.YParent = 1 Then
                Direction1 = "N"
            ElseIf CurrentNode.YCoord - CurrentNode.YParent = -1 Then
                Direction1 = "S"
            End If
            UD = Direction1 + Direction2
            PathToFollow.Add(UD)
            For Each PathNode As Node In ClosedList
                If PathNode.XCoord = CurrentNode.XParent And PathNode.YCoord =
CurrentNode.YParent Then
                    CurrentNode = PathNode
```

```
                End If
            Next
        Loop
'This is just for testing purposes
        For Each Path In PathToFollow
            MessageBox.Show(Path)
        Next
'End testing purposes block
    Else
        MessageBox.Show("No path found")
    End If

End Sub

Public Sub Testing(ByRef CurrentNode As Node, ByRef XMod As Integer, ByRef YMod As
Integer, ByRef XGoal As Integer, ByRef YGoal As Integer)
    Dim TestNode As New Node
    Dim MovementCost As Integer
    Dim XHerDistance As Integer
    Dim YHerDistance As Integer
    Dim Heuristic As Integer
    If XMod <> 0 And YMod <> 0 Then
        'Diagonal Movement detected
        MovementCost = DiagCost
    Else
        'Horizontal or vertical Movement only
        MovementCost = StraightCost
    End If

    XHerDistance = Math.Abs((CurrentNode.XCoord + XMod) - XGoal)
    YHerDistance = Math.Abs((CurrentNode.YCoord + YMod) - YGoal)
    If XHerDistance > YHerDistance Then
        Heuristic = DiagCost * YHerDistance + StraightCost * (XHerDistance - YHerDistance)
    Else
        Heuristic = DiagCost * XHerDistance + StraightCost * (YHerDistance - XHerDistance)
    End If

    'Test if we can walk on the Node, If we can't exit sub otherwise continue on
    If MapData(CurrentNode.XCoord + XMod, CurrentNode.YCoord + YMod) = 1 Then
        Exit Sub
    End If
    'Is this node the goal? If is Flag that the goal has been found so that path can be constructed.
    If CurrentNode.XCoord + XMod = XGoal And CurrentNode.YCoord + YMod = YGoal
Then
        GoalNode.XParent = CurrentNode.XCoord
```

```
            GoalNode.YParent = CurrentNode.YCoord
            GoalReached = True
            Exit Sub
        End If
        'Is this node on the closed list?
        For Each ClosedNode As Node In ClosedList
            If ClosedNode.XCoord = CurrentNode.XCoord + XMod And ClosedNode.YCoord =
CurrentNode.YCoord + YMod Then
                'Check to see if the cost is better, if so lower cost, reopen node with new F Value
                If ClosedNode.Cost < CurrentNode.Cost + MovementCost Then
                    ClosedList.Remove(ClosedNode)
                    ClosedNode.Cost = CurrentNode.Cost + MovementCost
                    ClosedNode.XParent = CurrentNode.XCoord
                    ClosedNode.YParent = CurrentNode.YCoord
                    ClosedNode.FValue = MovementCost + Heuristic
                    OpenList.Add(ClosedNode)
                    Exit Sub
                End If
            End If
        Next
        'None of the above? Congrats on a new node!!!
        TestNode.XCoord = CurrentNode.XCoord + XMod
        TestNode.YCoord = CurrentNode.YCoord + YMod
        TestNode.Cost = CurrentNode.Cost + MovementCost
        TestNode.XParent = CurrentNode.XCoord
        TestNode.YParent = CurrentNode.YCoord
        TestNode.FValue = MovementCost + Heuristic
        OpenList.Add(TestNode)
    End Sub
End Module
```

Lucrarea nr. 5
Modelarea mişcării – Partea a II-a

1. Tema lucrării: Construcţia unui mediu de simulare bazat al mişcării bazat pe împărţirea scenei virtuale în patru zone.
2. Scopul lucrării: Reprezentarea mişcării sistemelor mecatronice de tip robot în mediul virtual.
3. Consideraţii teoretice

3.1. Algoritmul propus

Parcurgerea unui drum de către un sistem mecatronic printr-un set de obstacole dispuse aleator, presupune utilizarea unui algoritm specializat. Deşi există mai mulţi algoritmi ce pot fi utilizaţi pentru rezolvarea problemei parcurgeri unui drum, în lucrarea de faţă se va prezenta un algoritm bazat pe faptul că drumul cel mai scurt dintre două puncte este o linie dreaptă(fig. 24).

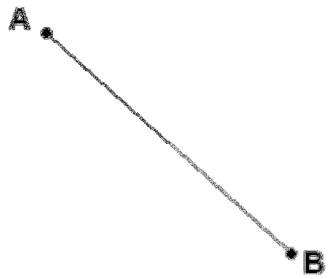


Figura 24. Drumul cel mai scurt dintre două puncte

Datorită faptului că sistemele mecatronice mobile (roboţii mobili) trebuie să realizeze sarcini în incinte unde se pot găsi şi obiecte dispuse aleator, algoritmul prezentat va include şi rezolvarea problemei evitării obstacolelor dispuse aleator printr-o metodă de coliziune şi un set de reguli.

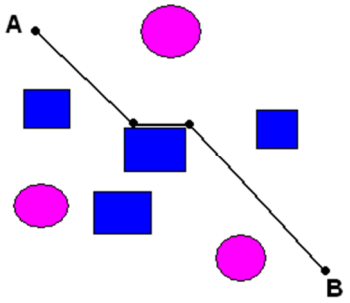


Figura 25. Parcurgerea unui drum printre obstacole dispuse aleator

Să presupunem că un sistem mecatronic mobil (robot) trebuie să parcurgă un drum din punctul A în punctul B, într-un spaţiu de dimensiuni date, presărat cu obicte ce sunt dispuse în mod aleator. Spaţiul în care acţionează sistemul mecatronic va fi de fapt o bucată de dimensiuni mai mici dintr-un spaţiu mai mare. De exemplu dacă luăm ca model o hală de producţie de dimensiuni 100mX100m şi punem un robot să îndeplinească anumite sarcini într-un spaţiu de dimensiuni 10mX10m, acesta va trebui să cunoscă numai această zonă cu toate obiectele din ea. Cu alte cuvinte dorim găsirea unui drum din punctul A în punctul B printr-un set de obiecte (obstacole) dispuse aleator într-un spaţiu de dimensiuni date.

Pentru a putea aplica algoritmul creat robotului trebuie în primul rând identificate variabilele de intrare şi virtualizarea spaţiului de lucru. Astfel în prima etapă se vor specifica variabilele de intrare:

S- punctul de coordonate x,y din care pleacă robotul

D- punctul de coordonate x,y în care ajunge robotul

O_n(x,y) - mulţimea de obiecte ce sunt dispuse la coordonatele x,y

Totodată se va considera scena statică ca find împărţită în patru zone, fiecare zonă având propiul ei sistem de coordonate (fig. 26).

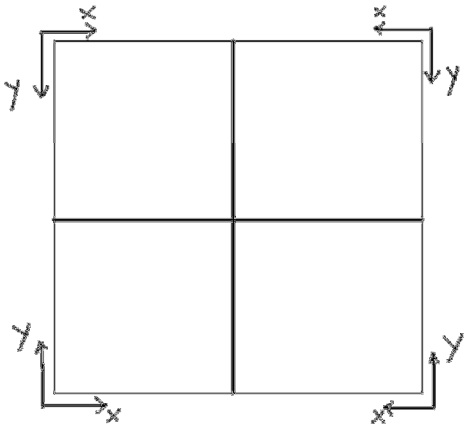


Figura 26. Modelarea scenei statice pentru algoritmul propus

Odată cu modelarea spaţiului de lucru, vor trebui modelate şi obiectele ce intră în componenţa acestuia. Astfel obiectele vor fi modelate ca ansambluri compuse din patru părţi (fig. 27):

- parte superioară : TOP
- parte inferioară: DOWN
- parte laterală stânga:LEFT
- parte laterală dreapta: RIGHT

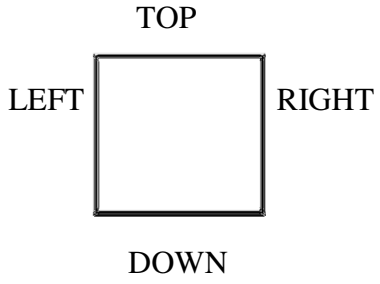


Figura 27. Cele patru părţi considerate ale obiectelor

Datorită faptului că scena statică este împărţită în patru zone fiecare având propiul ei sistem de coordonate, dacă punctul de plecare se va afla în zona A iar cel de destinaţi în zona D, obiectul ce se deplasează va trebui în momentul intrării în zona D să respecte noile coordonate şi regulile (stânga va deveni dreapta şi dreapta va deveni stânga) zonei în care intră (fig. 28).

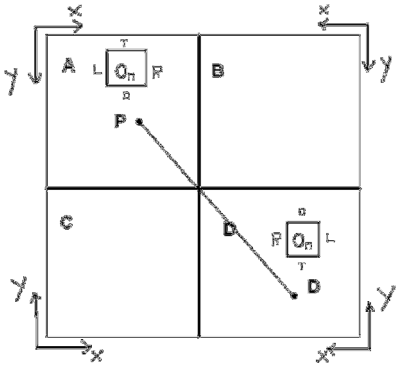


Figura 28. Regulile obiectelor bazate pe coordonatele zonelor

Odată stabilite cerinţele problemei, a virtualizării spaţiului de lucru cât şi a identificării variabilelor de intrare se va trece la rezolvarea problemei folosind algoritmul creat. Astfel se vor identifica două subprobleme ce apar în implementarea algoritmului:

I. Parcurgerea drumului de la punctual de plecare P la punctul de destinaţie D
II. Considerarea evitării obstacolelor

I. Parcurgerea drumului de la punctul de plecare P la punctul de destinaţie D

Se iau coordonatele punctului de plecare P, al punctului de destinaţie D cât şi coordonatele obstacolelor ce se găsesc în spaţiul de lucru. În continuare se va deplasa sistemul mecatronic mobil din punctul de plecare P de coordonate x,y în punctul de destinaţi D de coordonate x,y. Dacă sistemul mecatronic mobil ajunge la punctul de destinaţie algoritmul se încheie.

D1. [iniţializare] Se iau coordonatele P(x,y), D(x,y) şi $O_n(x,y)$

D2. [Deplasare din P spre D] Se deplasează P(x,y) către D(x,y)

D3. [comparare] Dacă P=D, algoritmul se încheie

II. Considerarea evitării obstacolelor

După cum am menţionat anterior în acest subcapitol algoritmul face o evitare a obiectelor pe baza unei metode de coliziune şi a unui set de reguli.

Din cauza dispunerii aleatoare a obstacolelor dintre punctele P şi D, a metodei de coliziune şi a celor patru zone ale scenei statice rezultă mai multe condiţii de evitare a obiectelor. După cum se ştie o coliziune între obiecte poate fi de mai multe tipuri. În continuare se vor prezenta numai 3 tipuri de coleziuni ce intervin în acest algoritm:

- coliziune simplă – caracterizată prin aceea că un sistem mecatronic mobil loveşte un singur obiect (fig. 29);

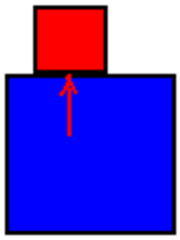


Figura 29. Coliziune simplă

- Coliziune dublă – caracterizată prin aceea că un sistem mecatronic mobil loveşte simultan două obiecte (fig. 30)

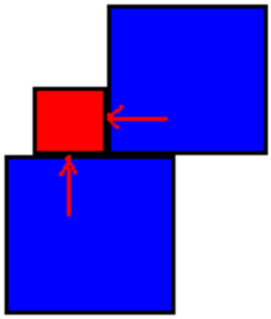


Figura 30 –Coliziune dublă

- Coliziune triplă – caracterizată prin aceea că un sistem mecatronic mobil loveşte simultan trei obiecte (fig. 31).

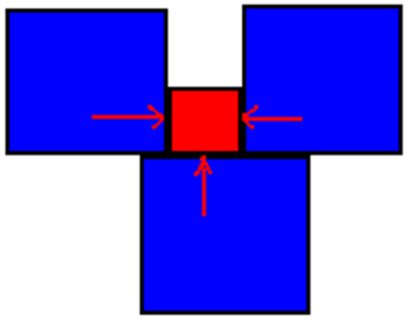


Figura 32. Coliziune triplă

Condiţiile pentru evitarea obiectelor sunt:

Dacă sistemul mecatronic mobil pleacă din zona A a scenei statice şi loveşte un obiect în partea superioară a acestuia, atunci sistemul mecatronic se deplasează spre dreapta pe axa x până ce acesta se va elibera după care se continuă drumul către punctul de destinaţie, iar dacă sistemul mecatronic mobil plecă din zona A a scenei şi loveşte un obiect în partea laterală stână , atunci sistemul mecatronic se deplasează în sus pe axa y până când acesta se va elibera după care se continuă drumul către punctul de destinaţie.

Pentru zona B a scenei setul de reguli este semănător dar privit în oglindă. Dacă în cazul anterior la coliziunea sistemului mecatronic mobil cu un obiect în partea superioară a acestuia, acesta o lua spre dreapta acum sistemul mecatronic mobil o va lua spre stânga. În cazul loviri obiectului în partea laterală stânga regula rămâne aceeaşi ca şi în cazul zonei A. În cazul zonelor C şi D ale scenei, datorită scimbării axelor de coordonate obiectele devin văzute în oglindă şi răsturnat, rezultând că regulile zonei A vor corespunde şi zonei C iar regulile zonei B vor corespunde şi zonei D.

În aceste condiţii se vor determina restul de reguli pentru zonele A şi B.

Dacă sistemul mecatronic mobil pleacă din zona A a scenei statice şi loveşte un obiect în partea superioară a acestuia şi un alt obiect în partea laterală stângă , atunci sistemul mecatronic se va deplasa în sus pe axa de coordonate y până se va elibera după care se continuă drumul către punctul de destinaţie. Pentru zona B se consideră că un obiect este lovit în partea superioară iar

un alt obicet este lovit în partea laterală dreaptă, regula de evitare a obiectelor rămând aceaşi ca şi în cazul zonei A.

Dacă sistemul mecatronic mobil pleacă din zona A şi loveşte trei obiecte simultan: unul în partea superioară, altul în partea laterală dreapta şi cel de al treilea în partea laterală stânga, atunci sistemul mecatronic se va deplasa în sus pe axa y până ce se va elibera după care se contiună drumul.

Dacă sistemul mecatronic mobil va lovi în loc de partea superioară a unui obiect partea inferioară a acestuia în oricare din combinaţile descrise mai sus regulile vor fi identice cu cele prezentate anterior.

D4. [Evitare obstacole] Dacă P(x,y)= $O_n(TOP)$ atunci se deplasează P spre dreapta până când $P(x,y) \neq O_n(TOP)$. Se marchează acest punct liber după care se reia deplasarea din noul punct către D

- Dacă P(x,y)={ $O_n(TOP)$, $O_n(LEFT)$ } se deplasează P în sus pe axa de coordonate y până când $P(x,y) \neq O_n(LEFT)$. Se marchează acest punct liber, după care se reia deplasarea din noul punct marcat către D
- Dacă P(x,y)={ $O_n(TOP)$, $O_n(RIGHT)$ } se deplasează P în sus pe axa de coordinate y până când $P(x,y) \neq O_n(RIGHT)$. Se marcheză acest punct liber, după care se reia deplasarea către D.
- Dacă P(x,y)={ $O_n(TOP)$, $O_n(RIGHT)$, $O_n(LEFT)$ } se deplaseză P spre stânga mergând în sus pe marginea obstacolului $O_n(LEFT)$ până când $P(x,y) \neq O_n(LEFT)$. Se marcheză acest punct liber , după care se reia depălasarea din noul punct marcat către D.
- Dacă P(x,y)= $O_n(DOWN)$ atunci se deplasează P spre stânga până când $P(x,y) \neq O_n(DOWN)$. Se marchează acest punct liber după care se reia deplasarea din noul punct către D

- Dacă P(x,y)={ $O_n(DOWN), O_n(LEFT)$ } se deplasează P în sus pe axa de coordonate y până când $P(x,y) \neq O_n(DOWN)$. Se marchează acest punct liber, după care se reia deplasarea din noul punct marcat către D
- Dacă P(x,y)={ $O_n(DOWN), O_n(RIGHT)$ } se deplasează P în sus pe axa de coordonate y până când $P(x,y) \neq O_n(DOWN)$. Se marchează acest punct liber, după care se reia deplasarea către D.
- Dacă P(x,y)={ $O_n(DOWN), O_n(RIGHT), O_n(LEFT)$ } se deplasează P spre stânga mergând în jos pe marginea obstacolului $O_n(LEFT)$ până când $P(x,y) \neq O_n(LEFT)$. Se marchează acest punct liber , după care se reia depălasarea din noul punct marcat către D.

D5.[Încheiere după ocolire] Dacă poziția curentă P(x,y)=D(x,y) atunci stop.

4. Aplicație

Să se implementeze algoritmul propus pentru generare mișcării în mediul limbajul Visual Basic.

Rezolvare

În mediul de dezvoltare Visual Basic se va implementa următorul cod:

```
Private Sub Picture1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
Static trap As Byte 'definim variabila trap pentru ca a realiza o verificare a pozitionari robotului

'daca s-a terminat procesul de scanare si variabila trap are valoarea  0 atunci robotul poate fi pozitionat pe scena statica
If scanat = True And trap = 0 Then

'se realizeaza o noua verificare pentru a nu pozitiona robotul in afara scenei statice
  If X <= room.Width - (robot.Width / 2) And Y <= room.Height - (robot.Height / 2) Then
    robot.Left = X - (robot.Width / 2)
    robot.Top = Y - (robot.Height / 2)

'daca este activat procedeul de simulare la nivel de punct material atunci robotul se 'dimensioneaza la nivelul unui punct
```

```
    If Option2 = True Then
    robot.Shape = 0
    robot.Left = robot.Left + (robot.Width \ 2) - 2
    robot.Top = robot.Top + (robot.Height \ 2) - 2
    robot.Width = 4
    robot.Height = 4
    End If

'pentru evitarea pozitionari robotului de catre utilizator pe un obstacol se realizeaza o verificare a pozitionari
    For n = 0 To 13
    While coliziune(robot, contur(n))
    Exit Sub
    Wend
    Next
    robot.Visible = True
    Text1.Text = Text1.Text & vbCrLf & "     S(" & Round(X / 20, 2) & "; " & Round(Y / 20, 2) & ")" & vbCrLf
    Text1.Text = Text1.Text & vbCrLf & "  Alegeti destinatia robotului: "
    trap = 1
    Exit Sub
  End If

End If

'alegem punctul de destinaţie prin verificarea valori variabilei trap
If scanat = True And trap = 1 Then 'IF 1

'se realizează o nouă verificare pentru a nu se pune destinaţia în afara scenei statice
  If X <= room.Width - (rDest.Width / 2) And Y <= room.Height - (rDest.Height / 2) Then 'IF 2
    rDest.Left = X - (rDest.Width / 2)
    rDest.Top = Y - (rDest.Height / 2)
'se realizează o dimensionare a punctului de destinaţie egală cu cea a robotului
    If Option2 = True Then
    rDest.Shape = 0
    rDest.Left = rDest.Left + (rDest.Width \ 2) - 2
    rDest.Top = rDest.Top + (rDest.Height \ 2) - 2
    rDest.Width = 4
    rDest.Height = 4
    End If
'se verifică dacă punctul de destinaţie nu a fost ales pe un obstacol
    For n = 0 To 13
    While coliziune(rDest, contur(n))
    Exit Sub
    Wend
```

```
    Next

    rDest.Visible = True
    Text1.Text = Text1.Text & vbCrLf & "     D(" & Round(X / 20, 2) & "; " & Round(Y / 20, 2)
& ")" & vbCrLf

    Text1.Text = Text1.Text & vbCrLf & "  Cautare drum" & vbCrLf & vbCrLf
    trap = 2
```

Conform necesităţilor cerute de către algoritm pentru modelarea scenei statice s-a realizat o împărţire a acesteia în patru zone în funcţie de direcţia de deplasare a robotului. Cele patru zone au fost denumite generic U-L,U-R,D-L,D-R. Pornind de la definirea variabilelor în cadrul programului:

```
Dim DestX, DestY As Double 'Coordonatele desinatiei
Dim CurX, CurY As Double 'Coordonatele robotului
Dim KoefX, KoefY As Double 'Coeficientul de deplasare al robotului
Dim SizeX, SizeY As Double 'Lungimea segmentelor de dreaptă
Dim MoveX, MoveY As Double 'Coeficientul de deplasare final
Dim SizeC As Double 'Lungimea drumului
Dim speed As Double 'Viteza de deplasare
Dim check As Boolean 'Adevarat daca s-a produs o coliziune
Dim Directie As String 'Directia in care trebuie deplasat robotul in cele 4 cadrane
Dim ColizMade(1) As Integer 'Memoreaza primele 2 coliziuni care se produc pentru a nu se
face a 2-a oara (in acest caz nu se poate sa ajunga la destinatie)
```

se trece la scrierea condiţilor folosite la definirea zonelor pentru direcţia robotului:

```
'Directia de deplasare a robotului
   If robot.Left <= rDest.Left And robot.Top <= rDest.Top Then Directie = "D-R"
   If robot.Left <= rDest.Left And robot.Top >= rDest.Top Then Directie = "U-R"
   If robot.Left >= rDest.Left And robot.Top >= rDest.Top Then Directie = "U-L"
   If robot.Left >= rDest.Left And robot.Top <= rDest.Top Then Directie = "D-L"
```

În timpul deplasării robotului, acesta trebuie să ţină cont de parametri setaţi în opţiunile simulării. Astfel după definirea zonelor de direcţie ale robotului se implementează realizarea unui calcul bazat pe viteza de deplasare şi a pixelilor astfel încât robotul să păstreze o concordanţă între viteza setată şi cea la care se realizează sistemul de referinţă în cadrul simulării precum şi realizarea marcări punctelor în care robotul realizează ocolirea propiuzisă a obstacolelor împreună cu desenarea şi calularea distanţelor dintre aceste puncte pe baza unor segmente de dreaptă :

```
'marcarea punctelor  în care are loc ocolirea obstacolelor şi trasarea segmentelor de dreaptă
drum(iDr).X1 = robot.Left + (robot.Width / 2)
drum(iDr).Y1 = robot.Top + (robot.Height / 2)
cordDr(iDr).Visible = True
```

```
cordDr(iDr).Left = drum(iDr).X1 - (cordDr(iDr).Width / 2)
cordDr(iDr).Top = drum(iDr).Y1 - (cordDr(iDr).Height / 2)
textDr(iDr).Left = drum(iDr).X1 + 10
textDr(iDr).Top = drum(iDr).Y1 - 10
textDr(iDr).Caption = "S"
Text1.Text = Text1.Text & "d[S(" & Round(drum(iDr).X1 / 20, 2) & ", " & Round(drum(iDr).Y1
/ 20, 2) & "), "
```

‘menţinerea vitezei de deplasare a robotului

```
While SizeC >= 1
DoEvents
SizeX = Abs(DestX - CurX)
SizeY = Abs(DestY - CurY)
SizeC = Sqr(SizeX * SizeX + SizeY * SizeY) '//c2=a2+b2

KoefX = SizeX / SizeC
KoefY = SizeY / SizeC

If DestX > CurX Then
   MoveX = speed * KoefX
ElseIf DestX < CurX Then
   MoveX = (speed * KoefX) * -1
End If
If DestY > CurY Then
   MoveY = speed * KoefY
ElseIf DestY < CurY Then
   MoveY = (speed * KoefY) * -1
End If

If Not check Then
CurX = CurX + MoveX
CurY = CurY + MoveY
robot.Left = CurX
robot.Top = CurY
drum(iDr).X2 = robot.Left + (robot.Width / 2)
drum(iDr).Y2 = robot.Top + (robot.Height / 2)
drum(iDr).Visible = True
End If
```

‘secvenţă de cod folosită pentru poziţionarea robotului pe punctul de destinaţie

```
If coliziune(robot, rDest) Then
DestX = rDest.Left
DestY = rDest.Top
CurX = robot.Left
CurY = robot.Top
```

```
While SizeC >= 1
SizeX = Abs(DestX - CurX)
SizeY = Abs(DestY - CurY)
SizeC = Sqr(SizeX * SizeX + SizeY * SizeY) '//c2=a2+b2
KoefX = SizeX / SizeC
KoefY = SizeY / SizeC
If DestX > CurX Then
   MoveX = speed * KoefX
ElseIf DestX < CurX Then
   MoveX = (speed * KoefX) * -1
End If
If DestY > CurY Then
   MoveY = speed * KoefY
ElseIf DestY < CurY Then
   MoveY = (speed * KoefY) * -1
End If
CurX = CurX + MoveX
CurY = CurY + MoveY
robot.Left = CurX
robot.Top = CurY
drum(iDr).X2 = robot.Left + (robot.Width / 2)
drum(iDr).Y2 = robot.Top + (robot.Height / 2)
drum(iDr).Visible = True
Wend
robot.Left = rDest.Left
robot.Top = rDest.Top
Timer2.Enabled = True
Exit Sub
End If
```

Pentru simplificarea procedeului de scriere a setului de reguli bazat pe coliziuni în programul software a fost necesară realizarea a trei funcţi. O funcția în care se specifică procedeul de modelare a mişcarii robotului pentru deplasarea sa în sus, în jos, spre stânga şi spre dreapta:

```
'se foloseşte comanda case pentru alegerea tipului de mişcare
Select Case direct
'când are loc o mişcare spre stânga
Case "left"
'MsgBox "left"
      rez = ob1.Left
      While ob1.Left >= ob2.Left - ob1.Width - 1
      DoEvents
      rez = rez - Koef * speed
      ob1.Left = rez
      drum(iDr).X2 = robot.Left + (robot.Width / 2)
      drum(iDr).Y2 = robot.Top + (robot.Height / 2)
```

```
        drum(iDr).Visible = True
        For n = 0 To 13
        realCol = advColiz(robot, contur(n))
        If realCol > 0 And n <> col Then
        secCol = True

        Exit Sub
        End If
        Next
        Wend

        SizeDrX = Abs(drum(iDr).X2 - drum(iDr).X1)
        SizeDrY = Abs(drum(iDr).Y2 - drum(iDr).Y1)
        SizeDrC = Sqr(SizeDrX * SizeDrX + SizeDrY * SizeDrY) '//c2=a2+b2
        totalDr = totalDr + SizeDrC
        Text1.Text = Text1.Text & "P" & iDr + 1 & "(" & Round(drum(iDr).X2 / 20, 2) & ", " &
Round(drum(iDr).Y2 / 20, 2) & ")]=" & Round(SizeDrC / 20, 2) & vbCrLf
        iDr = iDr + 1
        Load drum(iDr)
        drum(iDr).X1 = robot.Left + (robot.Width / 2)
        drum(iDr).Y1 = robot.Top + (robot.Height / 2)

        Load cordDr(iDr)
        cordDr(iDr).ZOrder (0)
        cordDr(iDr).Left = drum(iDr).X1 - (cordDr(iDr).Width / 2)
        cordDr(iDr).Top = drum(iDr).Y1 - (cordDr(iDr).Height / 2)
        cordDr(iDr).Visible = True

        Load textDr(iDr)
        textDr(iDr).ZOrder (0)
        textDr(iDr).Left = drum(iDr).X1 + 10
        textDr(iDr).Top = drum(iDr).Y1 - 10
        textDr(iDr).Caption = "P" & iDr
        textDr(iDr).Visible = True
        Text1.Text = Text1.Text & "d[P" & iDr & "(" & Round(drum(iDr).X1 / 20, 2) & ", " &
Round(drum(iDr).Y1 / 20, 2) & "), "

''când are loc o mişcare spre dreapta
Case "right"
'MsgBox "right"
        rez = ob1.Left
        While ob1.Left <= ob2.Left + ob2.Width + 1
        DoEvents
        rez = rez + Koef * speed
        ob1.Left = rez
```

```
drum(iDr).X2 = robot.Left + (robot.Width / 2)
drum(iDr).Y2 = robot.Top + (robot.Height / 2)
drum(iDr).Visible = True

For n = 0 To 13
realCol = advColiz(robot, contur(n))
If realCol > 0 And n <> col Then
secCol = True

Exit Sub
End If
Next
Wend

SizeDrX = Abs(drum(iDr).X2 - drum(iDr).X1)
SizeDrY = Abs(drum(iDr).Y2 - drum(iDr).Y1)
SizeDrC = Sqr(SizeDrX * SizeDrX + SizeDrY * SizeDrY) '//c2=a2+b2
totalDr = totalDr + SizeDrC
Text1.Text = Text1.Text & "P" & iDr + 1 & "(" & Round(drum(iDr).X2 / 20, 2) & ", " &
Round(drum(iDr).Y2 / 20, 2) & ")]=" & Round(SizeDrC / 20, 2) & vbCrLf

iDr = iDr + 1
Load drum(iDr)
drum(iDr).X1 = robot.Left + (robot.Width / 2)
drum(iDr).Y1 = robot.Top + (robot.Height / 2)

Load cordDr(iDr)
cordDr(iDr).ZOrder (0)
cordDr(iDr).Left = drum(iDr).X1 - (cordDr(iDr).Width / 2)
cordDr(iDr).Top = drum(iDr).Y1 - (cordDr(iDr).Height / 2)
cordDr(iDr).Visible = True

Load textDr(iDr)
textDr(iDr).ZOrder (0)
textDr(iDr).Left = drum(iDr).X1 + 10
textDr(iDr).Top = drum(iDr).Y1 - 10
textDr(iDr).Caption = "P" & iDr
textDr(iDr).Visible = True
Text1.Text = Text1.Text & "d[P" & iDr & "(" & Round(drum(iDr).X1 / 20, 2) & ", " &
Round(drum(iDr).Y1 / 20, 2) & "), "
```

'când are loc o mişcare spre partea superioară

```
Case "up"
'MsgBox "up"
rez = ob1.Top
```

```
    While ob1.Top >= ob2.Top - ob1.Height - 1
    DoEvents
    rez = rez - Koef * speed
    ob1.Top = rez
    drum(iDr).X2 = robot.Left + (robot.Width / 2)
    drum(iDr).Y2 = robot.Top + (robot.Height / 2)
    drum(iDr).Visible = True

    For n = 0 To 13
    realCol = advColiz(robot, contur(n))
    If realCol > 0 And n <> col Then
    secCol = True

    Exit Sub
    End If
    Next
    Wend

    SizeDrX = Abs(drum(iDr).X2 - drum(iDr).X1)
    SizeDrY = Abs(drum(iDr).Y2 - drum(iDr).Y1)
    SizeDrC = Sqr(SizeDrX * SizeDrX + SizeDrY * SizeDrY) '//c2=a2+b2
    totalDr = totalDr + SizeDrC
    Text1.Text = Text1.Text & "P" & iDr + 1 & "(" & Round(drum(iDr).X2 / 20, 2) & ", " &
Round(drum(iDr).Y2 / 20, 2) & ")]=" & Round(SizeDrC / 20, 2) & vbCrLf

    iDr = iDr + 1
    Load drum(iDr)
    drum(iDr).X1 = robot.Left + (robot.Width / 2)
    drum(iDr).Y1 = robot.Top + (robot.Height / 2)

    Load cordDr(iDr)
    cordDr(iDr).ZOrder (0)
    cordDr(iDr).Left = drum(iDr).X1 - (cordDr(iDr).Width / 2)
    cordDr(iDr).Top = drum(iDr).Y1 - (cordDr(iDr).Height / 2)
    cordDr(iDr).Visible = True

    Load textDr(iDr)
    textDr(iDr).ZOrder (0)
    textDr(iDr).Left = drum(iDr).X1 + 10
    textDr(iDr).Top = drum(iDr).Y1 - 10
    textDr(iDr).Caption = "P" & iDr
    textDr(iDr).Visible = True
    Text1.Text = Text1.Text & "d[P" & iDr & "(" & Round(drum(iDr).X1 / 20, 2) & ", " &
Round(drum(iDr).Y1 / 20, 2) & "), "
```

```
'când are loc o mişcare spre partea inferiaoară
Case "down"
'MsgBox "down"
    rez = ob1.Top
    While ob1.Top <= ob2.Top + ob2.Height + 1
    DoEvents
    rez = rez + Koef * speed
    ob1.Top = rez
    drum(iDr).X2 = robot.Left + (robot.Width / 2)
    drum(iDr).Y2 = robot.Top + (robot.Height / 2)
    drum(iDr).Visible = True

    For n = 0 To 13
    realCol = advColiz(robot, contur(n))
    If realCol > 0 And n <> col Then
    secCol = True

    Exit Sub
    End If
    Next
    Wend

    SizeDrX = Abs(drum(iDr).X2 - drum(iDr).X1)
    SizeDrY = Abs(drum(iDr).Y2 - drum(iDr).Y1)
    SizeDrC = Sqr(SizeDrX * SizeDrX + SizeDrY * SizeDrY) '//c2=a2+b2
    totalDr = totalDr + SizeDrC
    Text1.Text = Text1.Text & "P" & iDr + 1 & "(" & Round(drum(iDr).X2 / 20, 2) & ", " &
Round(drum(iDr).Y2 / 20, 2) & ")]=" & Round(SizeDrC / 20, 2) & vbCrLf

    iDr = iDr + 1
    Load drum(iDr)
    drum(iDr).X1 = robot.Left + (robot.Width / 2)
    drum(iDr).Y1 = robot.Top + (robot.Height / 2)

    Load cordDr(iDr)
    cordDr(iDr).ZOrder (0)
    cordDr(iDr).Left = drum(iDr).X1 - (cordDr(iDr).Width / 2)
    cordDr(iDr).Top = drum(iDr).Y1 - (cordDr(iDr).Height / 2)
    cordDr(iDr).Visible = True

    Load textDr(iDr)
    textDr(iDr).ZOrder (0)
    textDr(iDr).Left = drum(iDr).X1 + 10
    textDr(iDr).Top = drum(iDr).Y1 - 10
    textDr(iDr).Caption = "P" & iDr
```

```
    textDr(iDr).Visible = True
    Text1.Text = Text1.Text & "d[P" & iDr & "(" & Round(drum(iDr).X1 / 20, 2) & ", " & Round(drum(iDr).Y1 / 20, 2) & "), "
End Select
```

Următoarele două seturi de funcţii descriu coliziunile simple şi coliziunile de tip dublu şi triplu. Astfel au rezultat următoarele secvenţe de cod:

```
'definirea funcţiei pentru coliziunea simplă
Function coliziune(ob1 As Object, ob2 As Object) As Boolean
If ((ob1.Left >= ob2.Left And ob1.Left <= ob2.Left + ob2.Width) Or (ob2.Left >= ob1.Left And ob2.Left <= ob1.Left + ob1.Width)) And _
  ((ob1.Top >= ob2.Top And ob1.Top <= ob2.Top + ob2.Height) Or (ob2.Top >= ob1.Top And ob2.Top <= ob1.Top + ob1.Height)) Then

coliziune = True
Else
coliziune = False
End If
End Function

'definirea funcţiei pentru coliziunea dublă şi triplă
Function advColiz(ob1 As Object, ob2 As Object) As Byte
'0 –nu există coliziune
'1 - coliziune right
'2 - coliziune left
'3 - coliziune bottom
'4 - coliziune top

If ((ob1.Left >= ob2.Left And ob1.Left <= ob2.Left + ob2.Width) Or (ob2.Left >= ob1.Left And ob2.Left <= ob1.Left + ob1.Width)) And _
  ((ob1.Top >= ob2.Top And ob1.Top <= ob2.Top + ob2.Height) Or (ob2.Top >= ob1.Top And ob2.Top <= ob1.Top + ob1.Height)) Then
'advColiz = 5
If (ob1.Left >= ob2.Left + ob2.Width - 1) And (ob1.Left <= ob2.Left + ob2.Width + 1) Then
advColiz = 1
ElseIf (ob1.Top >= ob2.Top + ob2.Height - 1) And (ob1.Top <= ob2.Top + ob2.Height + 1) Then
advColiz = 3
ElseIf (ob1.Left + ob1.Width >= ob2.Left - 1) And (ob1.Left + ob1.Width <= ob2.Left + 1) Then
advColiz = 2
ElseIf (ob1.Top + ob1.Height >= ob2.Top - 1) And (ob1.Top + ob1.Height <= ob2.Top + 1) Then
advColiz = 4
End If
```

```
Else
advColiz = 0
End If
End Function
```

Deoarece algoritmul creat se foloseşte de sistemul de coliziuni ce generează un set de reguli, în cadrul programului acestea au trebuit definite:

```
'Verificare de coliziune
'0 – nu există coliziune
'1 - coliziune right
'2 - coliziune left
'3 - coliziune bottom
'4 - coliziune top

If ColizMade(1) = n Then 'Daca s-a mai produs a 2-a coliziune odata atunci nu se poate ajunde la destinatie
MsgBox "Robotul nu poate ajunge la destinatie"
Command2.Enabled = True
Command2.Caption = "Restart"
Exit Sub
End If

If ColizMade(0) > 0 And ColizMade(1) < 1 Then ColizMade(1) = n
If ColizMade(0) < 1 Then ColizMade(0) = n

check = True

   'Coliziune de sus
   If col = 4 Then
   robot.Top = robot.Top - 1

   If Directie = "D-R" Then
   Misca "right", robot, contur(n), KoefX, speed, n
   If secCol Then Exit For
   Misca "down", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
   End If

   If Directie = "D-L" Then
   Misca "left", robot, contur(n), KoefX, speed, n
   If secCol Then Exit For
   Misca "down", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
```

```
End If

If Directie = "U-R" Then
Misca "left", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
Misca "down", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
Misca "right", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
End If

If Directie = "U-L" Then
Misca "right", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
Misca "down", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
Misca "left", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
End If

CurX = robot.Left
CurY = robot.Top
Exit For
End If

'Coliziune din stanga
If col = 2 Then
robot.Left = robot.Left - 1

If Directie = "D-R" Then
Misca "up", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
Misca "right", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
Misca "down", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
End If

If Directie = "D-L" Then
Misca "down", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
Misca "right", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
If coliz2 = 3 Then
   Misca "up", robot, contur(n), KoefY, speed, n
```

```
    If secCol Then Exit For
    Misca "left", robot, contur(n), KoefX, speed, n
    If secCol Then Exit For
  End If
  End If

  If Directie = "U-R" Then
  Misca "down", robot, contur(n), KoefY, speed, n
  If secCol Then Exit For
  Misca "right", robot, contur(n), KoefX, speed, n
  If secCol Then Exit For
  Misca "up", robot, contur(n), KoefY, speed, n
  If secCol Then Exit For
  End If

  If Directie = "U-L" Then
  Misca "up", robot, contur(n), KoefY, speed, n
  If secCol Then Exit For
  Misca "right", robot, contur(n), KoefX, speed, n
  If secCol Then Exit For
  If coliz2 = 4 Then
    Misca "down", robot, contur(n), KoefY, speed, n
    If secCol Then Exit For
    Misca "left", robot, contur(n), KoefX, speed, n
    If secCol Then Exit For
  End If
  End If

  CurX = robot.Left
  CurY = robot.Top
  Exit For
  End If

  'Coliziune de jos
  If col = 3 Then
  robot.Top = robot.Top + 1

  If Directie = "D-R" Then
  Misca "left", robot, contur(n), KoefX, speed, n
  If secCol Then Exit For
  Misca "up", robot, contur(n), KoefY, speed, n
  If secCol Then Exit For
  Misca "right", robot, contur(n), KoefX, speed, n
  If secCol Then Exit For
  End If
```

```
If Directie = "D-L" Then
Misca "right", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
Misca "up", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
Misca "left", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
End If

If Directie = "U-R" Then
Misca "right", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
Misca "up", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
End If

If Directie = "U-L" Then
Misca "left", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
Misca "up", robot, contur(n), KoefY, speed, n
If secCol Then Exit For

End If

CurX = robot.Left
CurY = robot.Top
Exit For
End If

'Coliziune din dreapta
If col = 1 Then
robot.Left = robot.Left + 1

If Directie = "D-R" Then
Misca "down", robot, contur(n), KoefY, speed, n
If secCol Then Exit For
Misca "left", robot, contur(n), KoefX, speed, n
If secCol Then Exit For
If coliz2 = 3 Then
   Misca "up", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
   Misca "right", robot, contur(n), KoefX, speed, n
   If secCol Then Exit For
End If
```

```
   End If

   If Directie = "D-L" Then
   Misca "up", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
   Misca "left", robot, contur(n), KoefX, speed, n
   If secCol Then Exit For
   Misca "down", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For

   End If

   If Directie = "U-R" Then
   Misca "up", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
   Misca "left", robot, contur(n), KoefX, speed, n
   If secCol Then Exit For
   If coliz2 = 4 Then
      Misca "down", robot, contur(n), KoefY, speed, n
      If secCol Then Exit For
      Misca "right", robot, contur(n), KoefX, speed, n
      If secCol Then Exit For
   End If
   End If

   If Directie = "U-L" Then
   Misca "down", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
   Misca "left", robot, contur(n), KoefX, speed, n
   If secCol Then Exit For
   Misca "up", robot, contur(n), KoefY, speed, n
   If secCol Then Exit For
   End If

   CurX = robot.Left
   CurY = robot.Top
   Exit For
   End If

Else
   check = False
End If 'End Coliziune
Next 'For 1
coliz2 = col
```

Pentru realizarea animaţiei ce prezintă deplasarea robotului s-a folosit un nou timer. În cadrul acestui timer s-au scris evenimentele necesare realizarii unei deplasări sub o formă animată, totodată rezultând şi trasarea drumului parcurs cu afişarea distanţelor în funcţie de fiecare ocolire a obstacolelor:

‘calcularea segmentelor de dreaptă dintre puncte
SizeDrX = Abs(drum(iDr).X2 - drum(iDr).X1)
SizeDrY = Abs(drum(iDr).Y2 - drum(iDr).Y1)
*SizeDrC = Sqr(SizeDrX * SizeDrX + SizeDrY * SizeDrY) '//c2=a2+b2*
totalDr = totalDr + SizeDrC
Text1.Text = Text1.Text & "D(" & Round(drum(iDr).X2 / 20, 2) & ", " & Round(drum(iDr).Y2 / 20, 2) & ")]=" & Round(SizeDrC / 20, 2) & vbCrLf

‘reluarea drumului din noul punct rezultat în urma ocolirii obstacolelor spre destinaţie
iDr = iDr + 1
Load cordDr(iDr)
cordDr(iDr).ZOrder (0)
cordDr(iDr).Left = drum(iDr - 1).X2 - (cordDr(iDr).Width / 2)
cordDr(iDr).Top = drum(iDr - 1).Y2 - (cordDr(iDr).Height / 2)
‘marcarea punctului de schimbare a direcţiei robotului
cordDr(iDr).Visible = True

‘afişarea numelor punctelor rezultate în urma evitării obstacolelor de către robot
Load textDr(iDr)
textDr(iDr).ZOrder (0)
textDr(iDr).Left = drum(iDr - 1).X2 + 10
textDr(iDr).Top = drum(iDr - 1).Y2 - 10
textDr(iDr).Caption = "D"
textDr(iDr).Visible = True
robot.Visible = False
rDest.Visible = False
Timer2.Enabled = False
‘afişarea distanţei totale de la punctul de pornire S la punctul de destinaţie D
Text1.Text = Text1.Text & vbCrLf & vbCrLf & "d(S, D)= " & Round(totalDr / 20, 2)
MsgBox "Done"
Command2.Enabled = True
Command2.Caption = "Restart"

Lucrarea nr. 6
Simularea unor procese fizice – Partea I

1. Tema lucrării: Simularea proceselor fizice pentru sistemele mecatronice. Simularea în mediul MATLAB.
2. Scopul lucrării: prezentarea introductivă a noţiunilor despre procesele fizice şi simularea modelelor în mediul MATLAB.
3. Consideraţii teoretice

3.1 Definiţii de bază pentru procesele fizice
3.2 Dinamica sistemelor mecatronice articulare

3.1 Prin sistem întelegem o structura fizica (un reactor chimic, masina electrica, un amplificator electronic, atmosfera Pamantulu societatea omeneasca, economia unei tari, etc.) in cadrul careia se desfasoara un anumit proces în conformitate cu legitatile care guverneaza.

De regula legile care guverneaza desfasurarea proceselor sun legile generale ale naturii, însotite în cazuri speciale (cum ar functionarea economiei unei tari) de protocoale, conventii, reglementari. Procesul se defineste ca o succesiune de operatii care au loc într-anumita ordine într-o structura fizica în conformitate cu legile general ale naturii.

Deorece un proces nu poate sa se desfasoare decat în cadrul unei structuri fizice, putem spune în consecinta ca, modelul este în egala masura atat al sistemului cat si al procesului cu precizarea ca atunci cand vorbim de modelul unui sistem întelegem o structura fizica în care se desfasoara un proces.

3.2 Dinamica sistemelor mecatronice articulare

Se va considera în continuare un sistem mecanic articulat compus din N corpuri. Procedura generală de obţinere a modelului sub forma ecuaţiilor de stare este următoarea:

1. Fixarea unui sistem de referinţă inerţial în spaţiul celor *N* referenţiale mobile fixate centrelor de greutate ale celor *N* corpuri ale sistemului;
2. Scrierea ecuaţiilor ce descriu restricţiile de mişcare şi de legături la care este supus sistemul.Va rezulta numărul de grade de libertate ale sistemului;
3. Scrierea ecuaţiilor de mişcare (translaţie şi rotaţie) pentru fiecare dintre coordonate,în *y* incluzându-se şi forţele de legătură corespunzătoare restricţiilor (metoda coeficienţilor lui Lagrange);
4. Eliminarea coeficienţilor lui Lagrange şi a coordonatelor redundante.

În continuare se va detalia procedura enunţată, explicitându-se conceptele noi (grade de libertate, coeficienţii lui Lagrange, coordonate redundante). Un exemplu tipic al metodei îl constituie modelul dinamic al unui braţ manipulator cu două grade de libertate.

1.Definirea coordonatelor

În spaţiul Ω al sistemului format din N corpuri se fixează un sistem de referinţă inerţial. Fiecărui corp (centrului său de greutate) îi este ataşat câte un sistem de referinţă mobil. În orice moment de timp, sistemul de N corpuri este caracterizat de vectorul de coordonate:

$\xi = \left[x_1\, y_1 \theta_1 x_2\, y_2 \theta_2 ... x_n\, y_n \theta_n\right]^T$, de dimensiune 3N.

2.Exprimarea restricţiilor de mişcare

Mişcarea unui sistem mecanic articulat poate fi supusă la două tipuri de restricţii (denumite restricţii geometrice): pe de o parte restricţii de parcurs, iar pe de altă parte restricţii determinate de legăturile dintre corpuri. Aceste restricţii pot fi exprimate sub forma unui ansamblu de relaţii algebrice între coordonate:

$\Psi(\xi) = 0$

unde Ψ este o aplicaţie $\Omega \to \Re^p$,p fiind numărul de restricţii.Conform teoremei funcţiilor implicite, în vecinătatea oricărui punct ξ al spaţiului Ω, există o partiţie $\xi = (x, \bar{x})$ a vectorului de coordonate astfel încât:

- Dimensiunea lui $\bar{x}$, notată cu σ este egală cu rangul matricii Jacobiene a aplicaţiei Ψ:

$$\sigma = \dim \bar{q} = rang \frac{\partial \Psi}{\partial \xi}$$

- Se pot exprima coordonatele $\bar{x}$ în funcţie de coordonatele x:

$$\left[\bar{x}\right] = \left[\Phi(x)\right]$$

rezultă că se poate utiliza relaţia pentru eliminarea coordonatelor redundante $\bar{x}$ din descrierea sistemului.Dimensiunea vectorului x de coordonate ce vor fi conservate reprezintă numărul de grade de libertate ale sistemului,notate cu δ:

$\delta = 3N - \sigma$

3.Ecuaţiile de mişcare

Se scriu ecuaţiile de mişcare pentru fiecare corp, incluzându-se în y forţele de legătură corespunzătoare restricţiilor. Partiţia de coordonate $(x, \bar{x})$va determina o partiţie similară în ansamblul ecuaţiilor de mişcare:

(13)

$$\left[J\right]\left[\ddot{x}\right] + \left[b(x, \bar{x})\right] = \left[B(x, \bar{x})\right]\left[u\right] + \left[v\right]$$
$$\left[\bar{J}\right]\left[\ddot{\bar{x}}\right] + \left[\bar{b}(x, \bar{x})\right] = \left[\bar{B}(x, \bar{x})\right]\left[u\right] + \left[\bar{v}\right]$$

În ecuaţiile de mai sus [v] şi $\left[\bar{v}\right]$ reprezintă forţele de legătură ce garantează respectarea în orice moment a restricţiilor.

În mecanica teoretică se demonstrează că forţele de legătură se exprimă:

$$\left[v\right] = -\left[A(x)\right]\left[\lambda\right]$$
$$\left[\bar{v}\right] = \left[\lambda\right]$$

în care [λ] este vectorul coeficienţilor lui Lagrange (de dimensiune σ), iar A(x) este matricea, de dimensiune $\delta x \sigma$ definită ca :

$$[A(x)] = \left(\frac{\partial \Phi}{\partial x} \right)^T$$

4.Eliminarea coordonatelor redundante

Din (14), ţinând cont şi de (16), se exprimă [λ]:

$[\lambda] = [\bar{J}][\ddot{\bar{x}}] + [b(x, \dot{\bar{x}})] - [\bar{B}(x, \dot{\bar{x}})][u]$

Înlocuind expresia obţinută în (13), ţinând cont şi de (15), se obţine:

$[J][\ddot{x}] + [A(x)][\bar{J}][\ddot{\bar{x}}] + [b(x, \Phi(x))] + [A(x)][\bar{b}(x, \Phi(x))] = \{[B(x, \Phi(x))] + [A(x)][\bar{B}(x, \Phi(x))]\}[u]$

În formula de mai trebuie eliminat $[\ddot{\bar{x}}]$.Pentru aceasta, se derivează de două ori în raport cu timpul expresia , rezultând:

$$[\dot{\bar{x}}] = [A(x)]^T [\dot{x}]$$

$$[\ddot{\bar{x}}] = [A(x)]^T [\ddot{x}] + [\dot{A}(x)]^T [\dot{x}]$$

Înlocuind şi făcând notaţiile:

$[M(x)] = [J] + [A(x)][\bar{J}][A(x)]^T$

$[f(x, \dot{x})] = [A(x)][\bar{J}][\dot{A}(x)]^T [\dot{x}]$

$[g(x)] = [b(x, \Phi(x))] + [A(x)][\bar{b}(x, \Phi(x))]$

$[G(x)] = [B(x, \Phi(x))] + [A(x)][\bar{B}(x, \Phi(x))]$

rezultă în final modelul dinamic general al unui sistem mecanic articulat:

$$[M(x)][\ddot{x}] + [f(x, \dot{x})] + [g(x)] = [G(x)][u]$$

În această ecuaţie, semnificaţia mărimilor este următoarea:

- [x] – vectorul (de dimensiune δ) coordonatelor necesare descrierii sistemului;
- [M(x)] –matricea de inerţie (de dimensiune $\delta x \delta$) simetrică şi pozitiv definită;
- [f$(x, \dot{x})$] – vectorul (de dimensiune δ) ce reprezinră forţele şi cuplurile rezultate din legăturile corespunzătoare restricţiilor.Acest vector se mai poate scrie:

  $[f(x, \dot{x})] = [H(x, \dot{x})][\dot{x}]$

  unde matricea $[H(x, \dot{x})]$,de dimensiune $\delta x \delta$ este

  $[H(x, \dot{x})] = [A(x)][\bar{J}][\dot{A}(x)]^T$
- [g(x)] – vectorul (de dimensiune δ) ce reprezintă forţele şi cuplurile rezultate datorită gravitaţiei;
- [u] – vectorul (de dimensiune m) stimulilor externi, respectiv al forţelor şi cuplurilor aplicate sistemului;
- [G(x)] – matrice cinematică, de dimensiune $\delta x m$

Pe baza modelului dinamic general descris de (21), se obţine în final modelul dinamic sub forma ecuaţiilor de stare (modelul de stare):

$$[\dot{x}] = [v]$$

$$[\ddot{x}] = [M(x)]^{-1}\{-[f(x,v)] - [g(x)] + [G(x)][u]\}$$

în care [x] este vectorul poziţiilor iar [v]=$[\dot{x}]$ este vectorul vitezelor.

5. Aplcație

Să se simuleze modelul de stare este prezentat schematic în figură pentru un braț manipulator, cu ajutorul mediului de simulare MATLAB.

Rezolvare

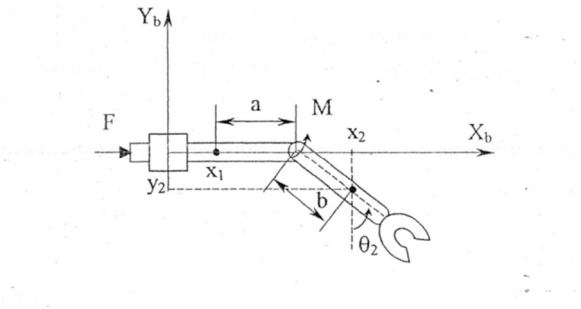


Figura 33

Rezolvare:

Robotul manipulator este format din două segmente articulate între ele. Primul poate realiza doar o mişcare de translaţie pe direcţia axei OX_b a sistemului inerţial general (X_bOY_b), sub acţiunea forţei F. Cel de al doilea poate executa doar o mişcare de rotaţie după axa OZ_b (perpendiculară pe planul X_bOY_b, sub acţiunea cuplului motor M). În figură au fost notate:

a – poziţia centrului de greutate al primului segment;

x_1 – coordonata centrului de greutate al primului segment (y_1, $0_1 = 0$);

b – poziţia centrului de greutate al celui de-al doilea segment;

x_2,y_2 - coordonatele centrului de greutate al celui de-al doilea segment;

θ_2 - unghiul dintre cel de al doilea segment şi verticală.

Sistemul este supus următoarelor restricţii:

- Restricţii de mişcare:

  $y_1 = 0$

  $\theta_1 = 0$

- Restricţii de legături (de articulaţii);

  $x_2 - b\sin\theta_2 - x_1 - a = 0$

  $y_2 + b\cos\theta_2 = 0$

Restricţiile de mişcare exprimă faptul că primul braţ nu se poate deplasa decât orizontal, iar cele de legături exprimă relaţia existentă între coordonatele cartesiene ale centrelor de greutate ale celor două braţe, ţinându-se cont de articulaţia dintre ele. Matricea Jacobiană a restricţiilor se scrie:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & -b\sin\theta_2 \\ 0 & 0 & 0 & 0 & 1 & +b\cos\theta_2 \end{bmatrix}$$

Se observă că matricea are rangul σ=4 şi deci sistemul are:

δ=3*2-4=2

grade de libertate (ceea ce era de aşteptat).În continuare, se poate defini partiţia de coordonate:

$$[x]=\begin{bmatrix} x_1 \\ \theta_1 \end{bmatrix}$$

$$[\bar{x}]=\begin{bmatrix} y_1 \\ \theta_1 \\ x_2 \\ y_2 \end{bmatrix}$$

De asemenea, se poate verifica uşor că întreg spaţiul (X_bOY_b), coordonatele $[\bar{x}]$ pot fi exprimate ca funcţii explicite $[\bar{x}]=[\Phi(x)]$, în funcţie de coordonatele $[x]$:

$$y_1=0$$
$$\theta_1=0$$
$$x_2=x_1+b\sin\theta_2+a$$
$$y_2=-b\cos\theta_2$$

Se pot elimina deci din descrierea sistemului coordonatele $[\bar{x}]=[y_1,\theta_1,x_2,y_2]^T$, păstrându-se doar coordonatele $[x]=[x_1,\theta_2]^T$ .Matricea [A(x)] va fi deci:

$$[A(x)]=\left[\frac{\partial\Phi}{\partial x}\right]^T=\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & b\cos\theta_2 & b\sin\theta_2 \end{bmatrix}$$

Cele şase ecuaţii de mişcare, corepunzătoare aplicării legii lui Newton pe fiecare dintre cele trei coordonate ale fiecărui corp sunt:

$$m_1\ddot{x}_1 = F-\lambda_3$$
$$I_2\ddot{\theta}_2 = -\lambda_3 b\cos\theta_2 - \lambda_4 b\sin\theta_2 + M$$
$$m_1\ddot{y}_1 = -m_1 g+\lambda_1$$
$$I_1\ddot{\theta}_1 = \lambda_2$$

$$m_2\ddot{x}_2 = \lambda_3$$
$$m_2\ddot{y}_2 = -m_2 g + \lambda_4$$

Combinând expresiile restricţiilor cu ecuaţiile de mişcare , se obţin următoarele valori pentru λ_1 şi λ_2:

$\lambda_1=m_1g$

$\lambda_2=0$

Valorile obţinute exprimă forţele de legătură ce sunt aplicate celor două corpuri pentru a satisface restricţiile de mişcare pe parcursul evoluţiei sistemului.

Pe de altă parte, eliminând λ_3 şi λ_4 între ecuaţiile de mişcare se obţine:

$$\begin{bmatrix} m_1\ddot{x}_1 \\ I_2\theta_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ b\cos\theta_2 & b\sin\theta_2 \end{bmatrix} \begin{bmatrix} m_2\ddot{x}_2 \\ m_2\ddot{y}_2 \end{bmatrix} = \begin{bmatrix} F \\ M - bm_2 g\sin\theta 2_2 \end{bmatrix}$$

Derivând apoi de două ori restricţiile (24) şi (25) se obţine:

$$\begin{bmatrix} m_2\ddot{x}_2 \\ m_2\ddot{y}_2 \end{bmatrix} = \begin{bmatrix} 1 & b\cos\theta_2 \\ 0 & b\sin\theta_2 \end{bmatrix} + m_2 b\dot{\theta}_2^2 \begin{bmatrix} -\sin\theta_2 \\ \cos\theta_2 \end{bmatrix}$$

Înlocuind acum (33) în (32) se obţine modelul sistemului sub forma dorită:

$$[M(x)][\ddot{x}] + [H(x,\dot{x})][\dot{x}] + [g(x)] = [G(x)][u]$$

în care:

$[M(x)] = \begin{bmatrix} m_1 + m_2 & m_2 b\cos\theta_2 \\ m_2 b\cos\theta_2 & I_2 + m_2 b^2\sin^2\theta_2 \end{bmatrix}$ este matricea de inerţie,

$[H(x,\dot{x})] = \begin{bmatrix} 0 & -m_2 b\dot{\theta}_2\sin\theta_2 \\ 0 & 0 \end{bmatrix}$ este matricea de impuls,

$[g(x)] = \begin{bmatrix} 0 \\ gm_2 b\sin\theta_2 \end{bmatrix}$ este matricea forţelor datorate gravitaţiei, iar

$[G(x)][u] = \begin{bmatrix} F \\ M \end{bmatrix}$ este matricea mărimilor de intrare(a stimulilor externi).

Modelul de stare, se obţine acum punând modelul matematic descris de sub forma ecuaţiilor de stare:

$$[\ddot{x}] = [C(x,u)] - [A(x,\dot{x})][\dot{x}] - [B(x,g)]$$

în care:

$$[A(x,\dot{x})] = [M(x)]^{-1}[H(x,\dot{x})]$$
$$[B(x,g)] = [M(x)]^{-1}[g(x)]$$
$$[C(x,u)] = [M(x)]^{-1}[G(x)][u]$$

Matricea $[M(x)]^{-1}$ este:

$[M(x)]^{-1} = \frac{1}{\Delta_M}\begin{bmatrix} I_2 + m_2 b^2\sin^2\theta_2 & -m_2 b\cos\theta_2 \\ -m_2 b\cos\theta_2 & m_1 + m_2 \end{bmatrix}$, cu

$$\frac{1}{\Delta_M} = \frac{1}{(m_1 + m_2)(I_2 + m_2 b^2\sin^2\theta_2) - (m_2 b\cos\theta_2)^2}$$

Făcând calculele matriceale se obţin:

$$[A(x,\dot{x})]*[\dot{x}]=\frac{1}{\Delta_M}\begin{bmatrix}-\left(I_2+m_2b^2\sin^2\theta_2\right)m_2b\sin\theta_2*\dot{\theta}_2^2\\(m_2b)^2\sin\theta_2\cos\theta_2*\dot{\theta}_2^2\end{bmatrix}$$

$$[B(x,g)]=\frac{1}{\Delta_M}\begin{bmatrix}-g(m_2b)^2\sin\theta_2\cos\theta_2\\g(m_1+m_2)m_2b\sin\theta_2\end{bmatrix}$$

$$[C(x,u)]=\frac{1}{\Delta_M}\begin{bmatrix}\left(I_2+m_2b^2\sin^2\theta_2\right)F-(m_2b\cos\theta_2)M\\-(m_2b\cos\theta_2)F+(m_1+m_2)M\end{bmatrix}$$

Rezultă în final expresiile, sub forma ecuaţiilor de stare ale celor două variabile:

$$\ddot{x}_1=\frac{1}{\Delta_M}\left\{\left(I_2+m_2b^2\sin^2\theta_2\right)F-(m_2b\cos\theta_2)M-\left[-\left(I_2+m_2b^2\sin^2\theta_2\right)m_2b\sin\theta_2*\dot{\theta}_2^2\right]-\left[-g(m_2b)^2\sin\theta_2\cos\theta_2\right]\right\}$$

respectiv

$$\ddot{\theta}_2=\frac{1}{\Delta_M}\left[-(m_2b\cos\theta_2)F+(m_1+m_2)M-(m_2b)^2\sin\theta_2\cos\theta_2*\dot{\theta}_2^2-g(m_1+m_2)m_2b\sin\theta_2\right]$$

Lucrarea nr. 7
Simularea unor procese fizice – Partea a II-a

1. Tema lucrării: Simularea proceselor fizice pentru sistemele mecatronice. Simularea în mediul MATLAB.
2. Scopul lucrării: Simularea proceselor fizice în mediul MATLAB.
3. Aplicaţie

În laboratorul anterior a fost considerat doar sistemul mecatronic articulat, format din corpuri rigide, fără flexibilitate sau supleţe în legături şi articulaţii. În multe situaţii (aplicaţii) o astfel de ipoteză nu este realistă.

O manieră simplă de introducere a elasticităţii (supleţei) în articulaţiile unui sistem mecatronic articulat constă în plasarea unui mic resort (fictiv) de masă nulă în legăturile dintre corpuri, ca în fiura 34. Acest resort exercită o forţă de tracţiune asupra fiecăruia din cele două corpuri de care este fixat. Această forţă se aplică în punctul de fixare a resortului, fiind monoton crescătoare în funcţie de elongaţia resortului. Această forţă elastică *se adaugă* celorlalte forţe aplicate sistemului.

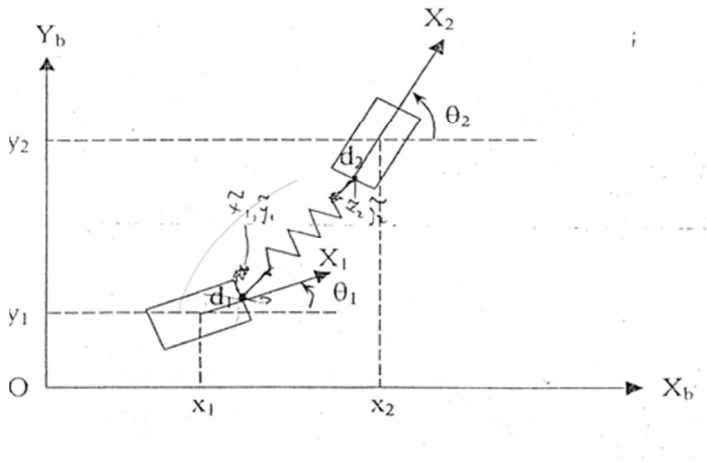


Figura 34 – Modelarea unei articulaţii elastice

Deoarece elasticitatea este introdusă în articulaţia dintre două corpuri ale sistemului, este evident că, pentru articulaţia respectivă, restricţiile de legătură (de articulaţie) dispar, iar în consecinţă, numărul gradelor de libertate creşte. Se va exemplifica metoda pentru un sistem simplu, format din două corpuri.

Se va considera sistemul cu două corpuri din figgura 34.Ecuaţiile de mişcare ale celor două corpuri sunt:

$$m_1\ddot{x}_1 = F_x$$
$$m_1\ddot{y}_1 = F_y$$
$$I_1\ddot{\theta}_1 = F_y d_1 \cos\theta_1 - F_x d_1 \sin\theta_1$$
$$m_2\ddot{x}_2 = -F_x$$

$$m_2 \ddot{y}_2 = -F_y$$
$$I_2 \ddot{\theta}_2 = F_x d_2 \sin\theta_2 - F_y d_2 \cos\theta_2$$

în care F_x şi F_y sunt componentele după axele sistemului inerţial (X_bOY_b) ale forţelor de tracţiune datorate resortului.

Coordonatele cartesiene ale punctelor de fixare ale resortului pe cele două corpuri se exprimă:

$\bar{x}_1 = x_1 + d_1 \cos\theta_1$

$\bar{y}_1 = y_1 + d_1 \sin\theta_1$

$\bar{x}_2 = x_2 - d_2 \cos\theta_2$

$\bar{y}_2 = y_2 - d_2 \sin\theta_2$

Elongaţia resortului, definită vectorial după cele două axe are componentele:

$\varepsilon_x = \bar{x}_2 - \bar{x}_1$

$\varepsilon_y = \bar{y}_2 - \bar{y}_1$

Componentele forţelor de tracţiune pot fi modelate ca şi funcţii monoton crescătoare în funcţie de elongaţie(fig. 7):

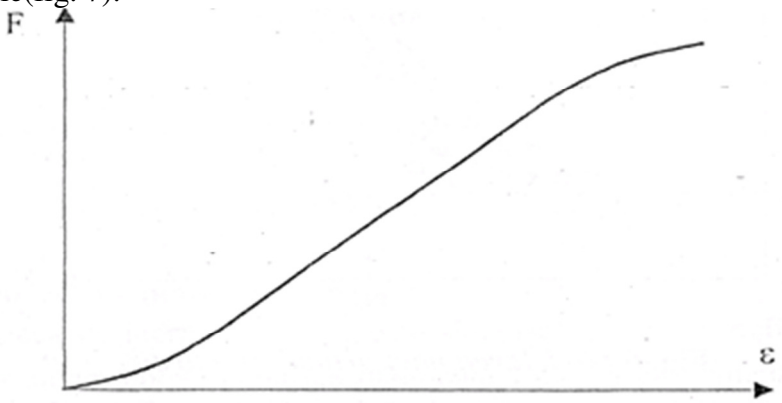


Figura 35 – Forța de tracțiune a unui resort în funcție de elongație

$F_x = r(\varepsilon_x)$

$F_y = r(\varepsilon_y)$

Frecvent, pentru simplificarea modelului, se consideră doar porţiunea liniară a funcţiei r(ε), respectiv:

$F_x = k_0(\bar{x}_2 - \bar{x}_1) = k_0((x_2 - x_1) - (d_1 \cos\theta_1 + d_2 \cos\theta_2))$

$F_y = k_0(\bar{y}_2 - \bar{y}_1) = k_0((y_2 - y_1) - (d_1 \sin\theta_1 + d_2 \sin\theta_2))$

în care k_0 este constanta de elasticitate a resortului.

Înlocuind expresiile componentelor forţei elastice în ecuaţiile de mişcare, acestea devin:

$$m_1\ddot{x}_1 = k_0\left((x_2 - x_1) - (d_1\cos\theta_1 + d_2\cos\theta_2)\right)$$
$$m_1\ddot{y}_1 = k_0\left((y_2 - y_1) - (d_1\sin\theta_1 + d_2\sin\theta_2)\right)$$
$$I_1\ddot{\theta}_1 = k_0 d_1\left((x_1 - x_2 + d_1\cos\theta_1 + d_2\cos\theta_2)\sin\theta_1 + (y_2 - y_1 - d_1\sin\theta_1 - d_2\sin\theta_2)\cos\theta_1\right)$$
$$m_2\ddot{x}_2 = -k_0\left((x_2 - x_1) - (d_1\cos\theta_1 + d_2\cos\theta_2)\right)$$
$$m_2\ddot{y}_2 = -k_0\left((y_2 - y_1) - (d_1\sin\theta_1 + d_2\sin\theta_2)\right)$$
$$I_2\ddot{\theta}_2 = k_0 d_2\left((x_2 - x_1 - d_1\cos\theta_1 - d_2\cos\theta_2)\sin\theta_2 + (y_1 - y_2 + d_1\sin\theta_1 + d_2\sin\theta_2)\cos\theta_2\right)$$

Rezultă deci că în cazul sistemelor mecanice articulate cu legături elastice,modelul dinamic general exprimat devine

$$[M(x)][\ddot{x}] + [f(x,\dot{x})] + [g(x)] + [k(x)] = [G(x)][u]$$

în care apare suplimentar termenul [k(x)], ce reflectă efectele forţelor elastice datorate articulaţiilor elastice din sistem.

Bibliografie

1. Eugenie Diatcu, Ioana Armaş – Bazele roboticii şi mecatronicii, Editura Victor, Bucureşti, 2001
2. Eugenie Diarcu, Ioana Armaş – Fiabilitatea sistemelor mecatronice, Editura HYPERION XXI, Bucureşti, 1998
3. Ioana Armaş - Calitatea şi fiabilitatea sistemelor mecatronice, Editura Victor, Bucureşti, 2004
4. Cezar Botezatu - Manual „Algoritmi şi structuri de date: fundamente ale programării structurate” , Editura Universitarã, Bucureşti
5. Ionescu Felicia, Grafica în realitatea virtuală, Bucureşti, 2000
6. Vince John, Realitatea virtuală, Bucureşti, 2000
7. Donald E. Knuth – Arta programării calculatoarelor (Volumul 3 – Sortare şi căutare) , Editura Teora, Bucureşti
8. Internet: http://cmpicsu.utt.ro
9. Internet: http://www.gamedev.net
10. Internet: http://theory.stanford.edu
11. Microsoft Visual Basic 6.0 – Ghidul programatorului, Editura Teora, Bucureşti
12. Christopher J. Bockmann, Lars Klander, Lingyan Tang – Visual Basic , Biblioteca Programatorului, Editura Teora, Bucureşti
13. Popescu Marius – Modelarea sistemelor și proceselor
14. Internet: http://www.drl.ro/webtt/discipline/co/lectii/cursuri/CO207%20-%20Modelarea%20prin%20grafuri%201.pdf