UNIVERSITATEA POLITEHNICA TIMIŞOARA

FACULTATEA DE MECANICĂ

TEZĂ DE DOCTORAT

SINTEZA INTELIGENȚEI ARTIFICIALE PE BAZA INTERACȚIUNII PRIN VEDERE A ROBOTULUI CU MEDIUL

Autor: ing. DAN MIHAI IOSIF

Cond. ştiinţific: prof. dr. ing. DOINA DRĂGULESCU

Cuprins

➢ Introducere

La fel ca în cazul organelor umane, capacitățile de vedere îi oferă unui robot un mecanism foarte sofisticat care îi permite să răspundă acțiunilor din mediul de lucru într-un mod inteligent şi flexibil. Utilizarea vederii cât şi a altor simțuri, este motivată de continua necesitate de creştere a flexibilității şi utilizărilor sistemelor robotizate. În timp ce senzorii de proximitate, atingere şi control al forței au un rol important în îmbunătățirea performanțelor robotului, vederea este recunoscută drept cel mai puternic senzor al unui robot. Aşa cum este de aşteptat, senzorii, conceptele şi hardware-ul necesar procesării asociate cu vederea artificială sunt în mod considerabil mai complexe decât cele asociate cu celelalte tipuri de senzori.

Vederea artificială poate fi definită ca un proces de extragere, caracterizare şi interpretare a informațiilor dintr-o imagine a lumii tridimensionale. Acest proces, poate fi împărțit în şase domenii principale: achiziție, preprocesare, segmentare, descriere, recunoaştere şi interpretare.

- ***Achiziția*** este procesul care generează o imagine pe computer.
- ***Preprocesarea*** se ocupă cu reducerea zgomotului şi îmbunătățirea detaliilor imaginii respective.
- ***Segmentarea*** este procesul care împarte o imagine în obiecte de interes.
- ***Descrierea*** tratează calculul diferitelor caracteristici (dimensiunea, forma, etc) ce se pretează pentru diferențierea unui obiect de altul.
- ***Recunoaşterea*** este procesul care identifică obiectele descrise anterior.
- ***Interpretarea*** dă un sens întregului ansamblu de obiecte existente într-o imagine.

Se obişnuieşte să se grupeze aceste domenii în funcție de gradul de complexitate necesar în implementarea lor. În teză se vor considera trei nivele de procesare: joasă, medie şi înaltă. Astfel: în categoria ***joasă*** intră acele procese care sunt considerate reacții automate şi nu necesită inteligență din partea sistemului de vedere artificială, cum ar fi: achiziția şi preprocesarea adică formarea imaginii, compensarea la zgomote şi în final extragerea de primitive de imagine aşa cum ar fi discontinuitățile de contur. Vederea de nivel ***mediu*** se referă la procesele de extragere, caracterizare şi marcare a componentelor dintr-o imagine rezultată din nivelul anterior. Din această categorie fac parte

segmentarea, descrierea şi recunoaşterea obiectelor individuale. Vederea de nivel înalt conține interpretarea, adică înțelegerea ansamblului de obiecte afişate pe ecran.

În timp ce algoritmii de nivel jos şi mediu sunt în general bine definiți, noțiunea de vedere de nivel ***înalt*** este una doar speculativă şi tratată deocamdată pe cazuri particulare utilizând restricții şi idealizări pentru a restrânge complexitatea acestei proceduri.

Principalul scop urmărit în teză este conducerea unui robot mobil printr-un spațiu populat cu obstacole. Pe baza informațiilor primite de la un sistem de vedere toate noțiunile prezentate au fost selectate astfel încât să servească scopului final ales. La majoritatea capitolelor pe lângă metodele implementate sunt prezentate din punct de vedere teoretic şi alte metode pentru scoaterea în evidență a avantajelor şi dezavantajelor metodelor utilizate practic.

Teza este structurată pe trei părți, fiecare dintre acestea reprezentând cele trei nivele prezentate anterior.

- ***Partea I-a : Achiziția şi preprocesarea imaginii***

Prima parte tratează vederea de complexitate joasă ocupându-se de achiziția şi preprocesarea imaginii. Partea de achiziție tratează dispozitivele hardware, punându-se accentul pe cele utilizate în realizarea experimentelor, precum şi software-ul folosit pentru achiziția imaginii. În partea de soft se va prezenta librăria Video for Windows a firmei Microsoft, librărie ce va fi utilizată pentru achiziția imaginii în cadrul rutinelor realizate. Odată ce există o imagine capturată, următoarea cerință în vederea realizării scopului propus este preprocesarea acestei imaginii. Necesitatea preprocesării rezultă din faptul că în urma achiziției imaginea prezintă zgomote, diferențe de iluminare, etc, plus, imaginea achiziționată este o imagine color iar pentru a simplifica vederea de nivel mediu este necesară o imagine binară (în alb şi negru). Având trasate astfel principale direcții de implementat, se vor prezenta diferite tehnici de iluminare, urmate de metodele de filtrare pentru eliminarea zgomotelor şi de metodele de îmbunătățire a imaginii pentru eliminarea efectelor de iluminare. În continuare se vor prezenta tehnicile de binarizare a imaginii şi cele de detecție de contur, tehnici care se vor folosi ulterior pentru descrierea obiectelor.

- ***Partea a II-a : Procesarea imaginii***

În cadrul acestei părți se realizează implementarea vederii de nivel mediu, adică se extrag corpurile din cadrul imaginii achiziționate şi preprocesate cu ajutorul metodelor din partea I-a, se descriu într-o manieră care să permită uşor recunoaşterea lor, şi în final se recunosc dintr-o bază de date.

- ***Partea a III-a : Planificarea mişcărilor roboților autonomi pe baza informațiilor obținute prin vedere artificială***

Aici se prezintă o variantă de interpretare a rezultatelor ce are drept rezultat conducerea unui robot printr-un mediu format din figuri geometrice regulate. În această parte sunt descrise de asemenea principiile de realizare a roboților cu care s-au efectuat experimentele şi metodele de planificare a mişcării acestora. În finalul capitolului este prezentată şi o descriere de ansamblu a programului realizat.

În finalul tezei sunt prezentate sintetic principalele probleme soluționate şi contribuțiile originale ale autorului.

Prezenta lucrare nu a putut fi elaborată fără sprijinul întregului colectiv din care autorul face parte şi care este condus cu profesionalism şi competență de domnul prof. dr. ing. Ioan Mureşan. Multe din ideile asupra cărora conținutul tezei s-a axat se datorează sugestiilor dumnealui atât în perioada de pregătire a examenelor şi referatelor cât şi în timpul elaborării tezei.

Gratitudinea mea se îndreaptă şi spre domnul prof. dr. ing. Nicolae Robu care mi-a pus la dispoziție primul robot mobil pe care s-au efectuat încercări.

Al doilea set de încercări experimentale s-au derulat pe robocarul conceput şi realizat de colegul s.l. ing. Dan Andreiciuc, căruia îi mulțumesc şi pe această cale.

Toate experimentările s-au derulat în laboratorul de Robotică al Catedrei de Organe de Maşini şi Mecanisme din cadrul Facultății de Mecanică, motiv pentru care îmi exprim recunoştința spre domnul prof. dr. ing. Dan Perju, şeful acestei catedre şi colegului Aurel Diaconu care a participat la experimentări.

Elaborarea acestei lucrări şi întreaga activitate de pregătire s-a desfăşurat în cadrul Departamentului de Automatică şi Informatică Industrială al Facultății de automatică şi Calculatoare, climatul de muncă din acest colectiv influențându-mă în mod pozitiv.

Pentru aceasta consider că trebuie să mulțumesc domnilor profesori Ştefan Holban şi Toma Leonida Dragomir.

În final, dar nu în ultimul rând, doresc să mulțumesc şi pe această cale conducătorului ştiințific, doamnei prof. dr. ing. Doina Drăgulescu, care m-a susținut şi m-a încurajat pe tot parcursul realizării acestei teze, oferindu-mi atât sprijinul material format din documentații, bibliografii, informații cât şi sprijinul moral atât de necesar încheierii cu succes a acestei perioade relativ lungi de studii şi experimente.

Partea I

Achiziția și preprocesarea imaginilor

1.1 Introducere

Partea de achiziție prezintă principalele dispozitive hardware utilizate în cadrul experimentelor, incluzând camerele video și plăcile de captură folosite.

Din punct de vedere software este descrisă utilizarea driverelor plăcilor de captură folosind librăria Video for Windows a firmei Microsoft. Vor fi prezentate descrierile funcțiilor utilizate precum și exemple de cod în limbaj C.

În partea de preprocesare sunt prezentați atât din punct de vedere teoretic cât și practic, folosindu-se exemplificarea prin imagini, o largă varietate de algoritmi utilizați în scopul pregătirii imaginii pentru faza de procesare propriu-zisă.

Sunt prezentați diverși algoritmi de filtrare ce au rolul eliminării zgomotelor din cadrul imaginii achiziționate, urmați de algoritmi de îmbunătățire a imaginii ce tratează problema eliminării efectelor datorate iluminării spațiului de lucru.

În scopul obținerii unor imagini de o calitate cât mai ridicată, se vor descrie și câteva tehnici de iluminare, ce au ca scop evitarea apariției umbrelor, a efectelor de tip „spot-light", etc.

Pe lângă metodele de tip spațial vor fi prezentate din punct de vedere teoretic și metode din domeniul de frecvență, dar care nu au fost implementate practic din cauza necesarului mare de calcule și timp pe care le implică, condiție neacceptabilă într-un sistem de vedere artificială industrială, care trebuie să funcționeze practic în timp real.

În finalul capitolului se prezintă câțiva algoritmi de detecție de contur, care realizează trecerea la următoarea fază în prelucrarea imaginilor, cea de procesare.

1.2 Achiziția imaginii

Informația vizuală este convertită în semnale electrice de către senzorii de vedere. Când aceste semnale sunt eşantionate spațial şi cuantificate ca nivele de amplitudine, rezultatul obținut va fi o imagine digitală. Se tratează două aspecte principale:

- cele mai importante dispozitive hardware utilizate în achiziția imaginii
- prezentarea software-ului necesar fazei de achiziție

În categoria dispozitivelor hardware, va fi descrisă funcționarea camerelor video CCD şi a celor cu tub vidat, cât şi plăcile de achiziție ce sunt utilizate în cadrul experimentelor de pe parcursul realizării tezei.

În categoria software va fi prezentată librăria Video for Windows de la firma Microsoft [133], cu toate facilitățile pe care aceasta le oferă pentru realizarea unei achiziții optime în cadrul unei aplicații. Vor fi exemplificate funcții şi porțiuni de cod necesare unei înțelegeri exacte a posibilităților oferite de această librărie.

1.2.1 Dispozitive hardware de achiziție

Dispozitivele principale utilizate în vederea artificială sunt camerele de luat vederi, constând fie dintr-un tub vidat, fie dintr-un senzor de imagine solid împreună cu electronica lor aferentă [96], [97]. Se prezintă, pe scurt, principiile de operare ale unui tub electronic cu vid şi ale dispozitivelor CCD (charge-coupled device). Dispozitivele CCD oferă un mare număr de avantaje asupra camerelor cu tub concretizate în :

- greutate mai mică
- dimensiuni mai mici
- viață mai lungă
- consum mai mic de energie

Cu toate acestea, rezoluția acestora este mai mică decât a celor cu tub electronic.

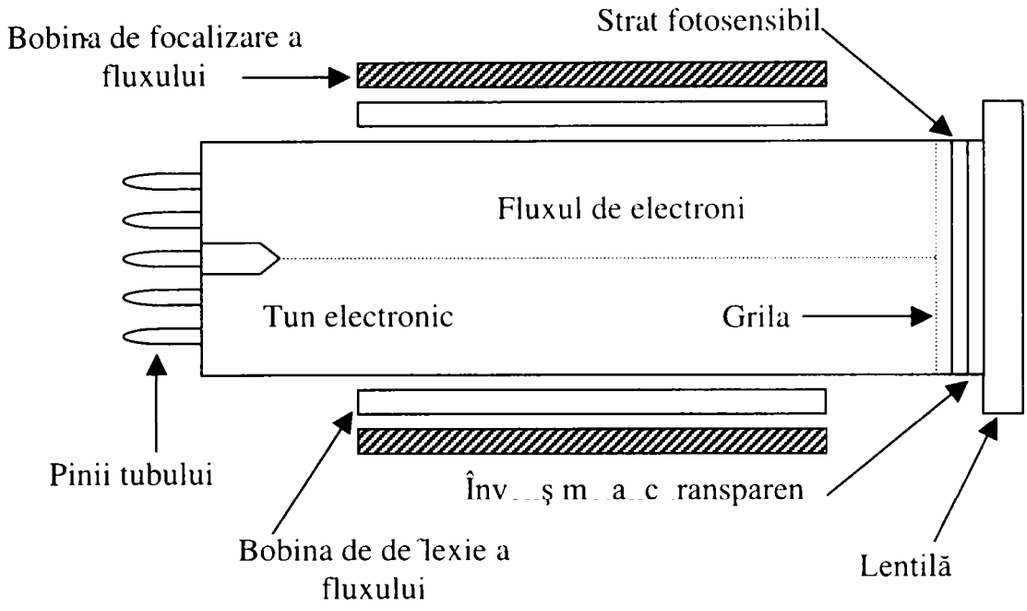


Fig. 1.2.1. Schema unui tub electronic

Aşa cum se prezintă în schema din figura 1.2.1, camera cu tub vidat este un cilindru de sticlă ce conține un tun electronic la un capăt şi o lentilă cu o țintă la celălalt. Fluxul este focalizat şi deviat prin tensiuni aplicate celor două bobine. Circuitul de deviere forțează fluxul să scaneze suprafața interioară a țintei în scopul citirii imaginii. Partea interioară a suprafeței lentilei de sticlă este învelită cu un film metalic transparent care formează un electrod din care este derivat un semnal video electric. Pe acest film este depus un strat subțire fotosensibil. Acest strat constă din globule rezistive foarte mici a căror rezistență este invers proporțională cu intensitatea luminoasă. În spatele acestei ținte fotosensibile se află o grilă metalică foarte fină încărcată pozitiv care decelerează electronii emişi de tun în aşa fel încât ei ajung să atingă suprafața de țintă cu o viteză egală, în principiu, cu zero.

În modul de operare normal, o tensiune pozitivă este aplicată învelişului metalic al lentilei. În absența luminii, materialul fotosensibil se comportă ca un dielectric, fluxul de electroni plasând un strat de electroni pe suprafața interioară a zonei de țintă pentru a echilibra încărcarea pozitivă a învelişului metalic. Pe măsură ce fluxul de electroni scanează suprafața stratului de țintă, acest strat fotosensibil devine un condensator cu sarcina negativă pe partea interioară şi pozitivă pe cea exterioară. Când lumina loveşte stratul de țintă, rezistența acestuia este redusă iar electronii pot să treacă prin el şi să neutralizeze sarcina pozitivă. Deoarece cantitatea de sarcină negativă care trece dintr-o parte în alta este proporțională cu cantitatea de lumină din

orice arie locală a țintei, acest efect produce o imagine pe stratul țintă care este identică cu imaginea luminoasă de pe lentila tubului. Astfel, concentrația de electroni rămasă este mare în zonele întunecate ale tubului şi mică în zonele luminoase. Pe măsură ce fluxul scanează ținta el înlocuieşte sarcina negativă pierdută, astfel generând un curent care curge din stratul metalic către unul din pinii tubului. Acest curent este proporțional cu numărul de electroni înlocuiți, adică cu intensitatea luminoasă dintr-o locație particulară a fluxului de scanare. Această variație de curent ce se produce în timpul mişcării fluxului de scanare, după ce este trecută prin circuitele electronice ale camerei este convertită în semnal video proporțional cu intensitatea imaginii .

Principiul standard de scanare utilizat în toate camerele cu tub vidat este prezentat în figura 1.2.2.

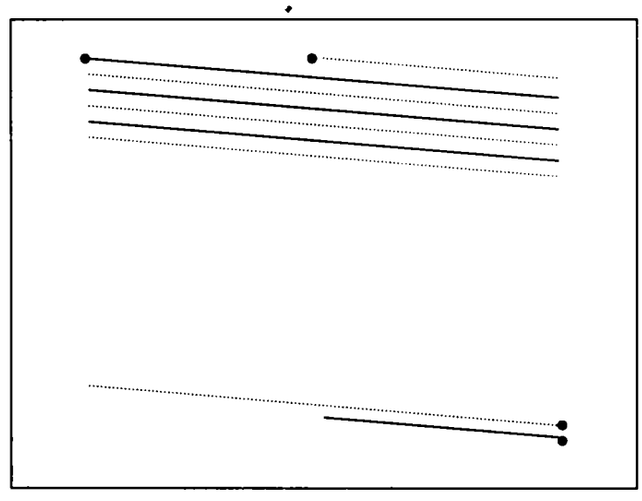


Fig. 1.2.2 Modul de scanare a fluxului de electroni

Fluxul de electroni scanează întreaga suprafață a țintei de 30 de ori pe secundă, fiecare scanare completă numită cadru constă din 525 de linii (SUA şi Japonia, iar Europa 625 de linii) din care 480 conțin informații de imagine (SUA).

Dacă liniile ar fi scanate secvențial iar rezultatul ar fi afişat pe un monitor, imaginea ar tremura (fenomenul de flicker). Acesta poate fi înlăturat prin utilizarea unui mecanism de scanare în care fiecare cadru este împărțit în două semicadre întrețesute, fiecare constând din 262,5 linii scanate de 60 de ori pe secundă, adică de două ori rata de scanare a unui singur cadru. Primul semicadru al fiecărui cadru scanează liniile impare (desenate cu linie punctată în figură), iar cel de-al doilea pe cele pare. Această metodă de scanare numită convenția de scanare RETMA (Radio

Electronics Television Manufactures Association) este standardul de televiziune utilizat în Statele Unite.

Când se discută despre dispozitive CCD, se obişnuieşte să se împartă aceşti senzori în două categorii:

- senzori de scanare linie
- senzori suprafață

Componenta de bază a unui senzor CCD linie este un rând de elemente de siliciu sensibile la lumină numite fotocelule. Fotonii imaginii trec printr-un structură de porți de siliciu policristalin transparent şi sunt absorbiți în cristalul de siliciu, creându-se astfel perechi electron-gol. Fotoelectronii rezultați sunt colectați în fotocelule, având cantitatea de sarcină colectată pe fiecare fotocelule proporțională cu intensitate luminoasă în acea locație. Un senzor linie tipic, (figura 1.2.3), este compus dintr-un rând de elemente de imagine, două porți de transfer utilizate pentru a sincroniza încărcarea conținutului elementelor de imagine în aşa numitele registre de transport şi o poartă de ieşire utilizată pentru sincronizarea transferului conținutului regiştrilor de transport în amplificator a cărei ieşire este un semnal în tensiune proporțional cu conținutul rândului de fotocelule.

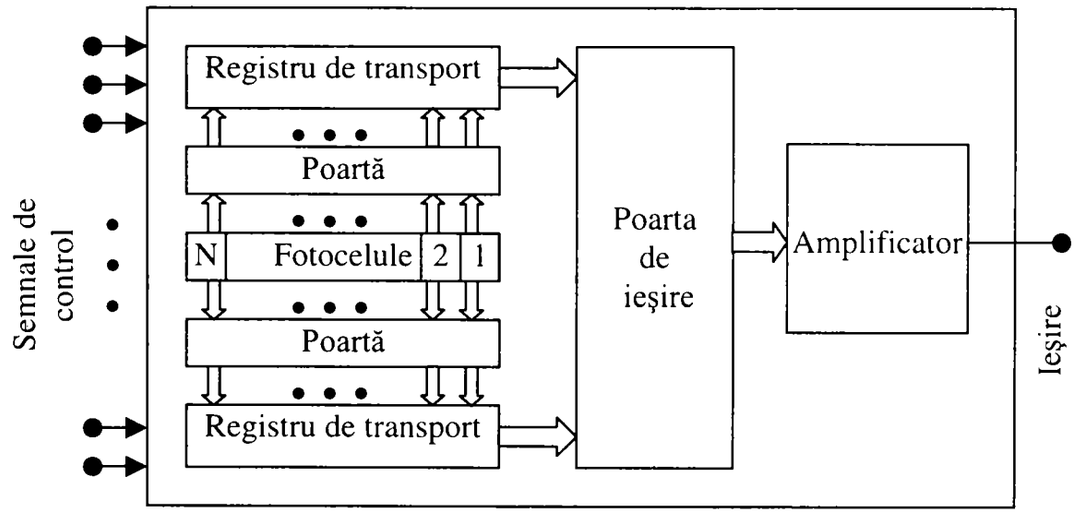


Fig. 1.2.3 Senzor CCD linie

Matricile CCD de suprafață sunt similare cu senzorii linie, cu excepția faptului că fotocelulele sunt aranjate într-un format matricial şi că există un registru de transport care combină între ele coloanele de fotocelule (figura 1.2.4).

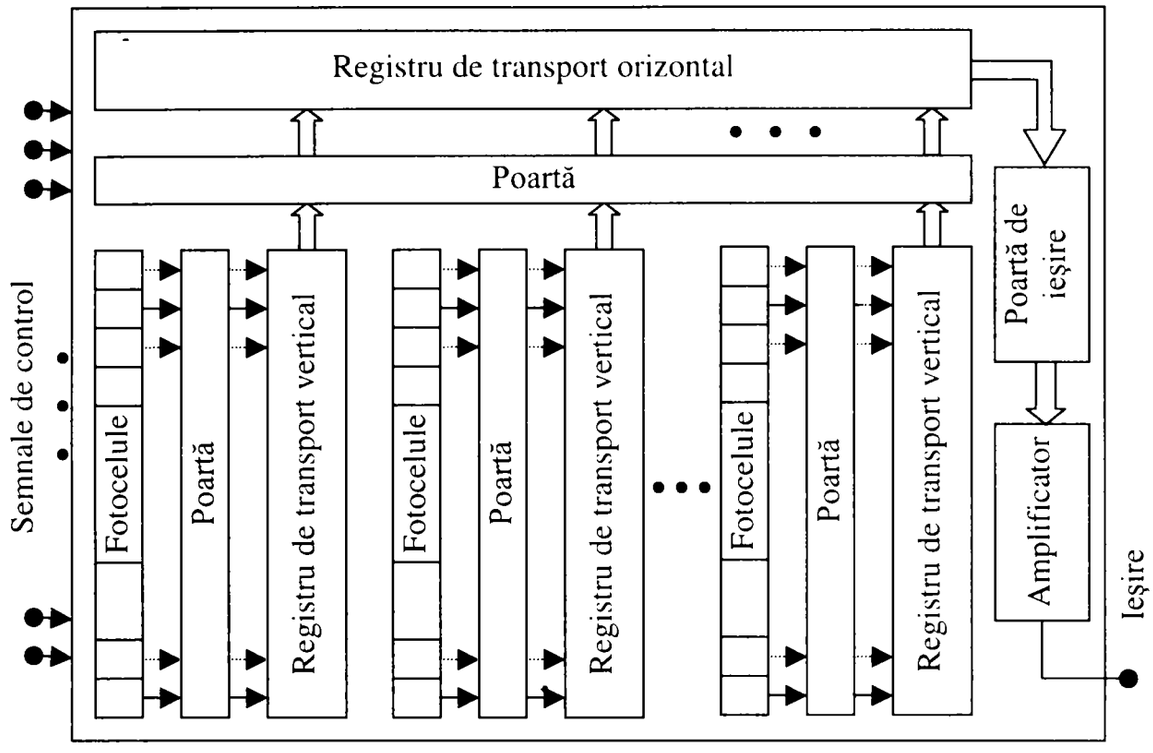


Fig. 1.2.4. Senzor CCD de suprafață

Conținutul fotocelulelor impare este încărcat secvențial în regiştrii de transport vertical iar după aceea într-un registru de transport orizontal. Conținutul acestui registru este trecut printr-un amplificator a cărui ieşire este semnalul video. Prin repetarea acestei proceduri pentru liniile pare, se completează cel de-al doilea semicadru dintr-un cadru TV. Acest mecanism de scanare se repetă de 30 de ori pe secundă.

Camerele de luat vederi care folosesc un singur senzor linie vor obține în mod evident o singură linie din imagine. Aceste dispozitive sunt ideale pentru aplicații în care obiectele se deplasează prin fața senzorului (pe benzi rulante). Mişcarea unui obiect pe o direcție perpendiculară pe senzor produce o imagine bidimensională. Rezoluțiile comune pentru acest tip de senzori sunt cuprinse între 256 şi 2048 de elemente. Rezoluțiile senzorilor de suprafață sunt cuprinse între 32 x 32 pentru cele joase şi 256 x 256 pentru cele de nivel mediu. Senzorii de nivel înalt au o rezoluție cuprinsă între 480 x 380 până la 1024 x 1024 de elemente (sau chiar mai mult).

Pe lângă camerele video, o altă componentă foarte importantă în cadrul procesului de achiziție o reprezintă placa de captură video. Aceste dispozitive hardware convertesc semnalul analogic obținut de la camera video într-un semnal

digital care este afişat sub forma unei imagini pe ecranul monitorului de către driverul respectivei plăci de achiziţie. Cele mai importante caracteristici ale acestor plăci de achiziţie sunt numărul de cadre achiziţionate pe secundă şi rezoluţia la care se poate realiza această achiziţie. În partea experimentală a tezei de doctorat s-au utilizat două plăci de achiziţie. În prima perioadă de experimentări, datorită lipsei unei oferte pe piaţa românească în anul 1994, s-a folosit o placă de captură Intel Smart VCR în sistem NTSC ce permite o rată de 15 cadre pe secundă la o rezoluţie maximă de 320 x 240 de pixeli. Ulterior a fost achiziţionată o placă de captură ATI All in Wonder Pro în sistem PAL, ce permite achiziţia a 30 de cadre pe secundă şi o rezoluţie maximă de 1024 x 68 de pixeli pentru imagini statice şi 640 x 480 de pixeli pentru o secvenţă de film. Nivelul de zgomot la cea din urmă este mult mai redus decât la prima, iar imaginile achiziţionate au nevoie de mult mai puţine filtrări pentru eliminarea zgomotelor, şi în concluzie o viteză mai mare de prelucrare.

1.2.2 Software-ul de achiziţie a imaginii în timp real. Librăria Video for Windows de la Microsoft

Microsoft a dezvoltat librăria Video for Windows pentru a se putea încorpora foarte uşor facilităţile de captură video în orice aplicaţie prin utilizarea clasei Windows AVICap [133]. Această clasă pune la dispoziţia programatorilor o interfaţă simplă bazată pe mesaje pentru accesarea achiziţiei hardware, atât video cât şi audio, şi în acelaşi timp oferă posibilităţi de control asupra întregului proces de salvare a capturii video pe disc.

- **Descrierea generală a facilităţilor oferite de clasa AVICap**

Clasa AVICap suportă atât capturarea unui film video cât şi capturarea unui singur cadru în timp real. În plus, AVICap asigură controlul asupra surselor video de tip Media Control Interface (MCI), astfel că utilizatorul poate accesa (printr-o aplicaţie) poziţiile de start şi stop ale sursei video, şi poate determina operaţia de captură să lucreze în mod pas cu pas.

Ferestrele care se pot crea folosind clasa Windows AVICap pot realiza următoarele operaţii:

- Capturarea filmelor audio video într-un fişier de tip AVI (audio video interleaved).

- Conectarea şi deconectarea dispozitivelor de intrare video şi audio în mod dinamic.
- Vizualizarea în direct (live) a unui semnal video folosind modurile vizionare (preview) şi suprapus (overlay).
- Specificarea unui fişier atunci când se capturează un film sau o imagine.
- Setarea ratei de captură.
- Afişarea de cutii de dialog ce permit controlul sursei şi formatului video.
- Crearea, salvarea şi încărcarea de palete.
- Copierea de imagini şi de palete în clipboard.
- Capturarea şi salvarea unei singure imagini într-un format bitmap independent de dispozitiv (DIB).

➢ **Prezentarea software-ului minim necesar pentru introducerea în aplicaţie a capturii video**

Captura video digitizează un şir de date video şi audio, şi îl salvează pe hard disc sau pe alt dispozitiv de stocare permanentă. Acest capitol descrie modul de adăugare a unei capturi video într-o aplicaţie folosind doar trei instrucţiuni. De asemenea este descris modul de terminare a sesiunii de captură prin trimiterea de mesaje către fereastra de captură.

O fereastră de captură AVICap se ocupă de toate detaliile legate de salvarea filmelor video şi audio în fişiere AVI. Acest lucru scuteşte aplicaţia de implicare în formatul fişierelor AVI, de managementul buferelor audio şi video şi de accesul la nivel de jos al driverelor audio şi video. Clasa AVICap asigură o interfaţă flexibilă pentru aplicaţii. Programatorul poate adăuga captura video în aplicaţii folosind următoarele linii de cod:

```
hWndC = capCreateCaptureWindow("Fereastra de captura",
                                WS_CHILD | WS_VISIBLE, 0, 0, 160, 120, hwndParent, nID);
SendMessage(hWndC,WM_CAP_DRIVER_CONNECT, 0 /* wIndex */, 0L);
SendMessage (hWndC, WM_CAP_SEQUENCE, 0, 0L);
```

De asemenea , este valabilă o interfaţă pa bază de macrouri care permite o alternativă la utilizare funcţiei SendMessage şi care asigură o îmbunătăţire a uşurinţei cu care codul unei aplicaţii poate fi citit. Exemplul următor demonstrează utilizarea acestei interfeţe în adăugarea de captură video la o aplicaţie:

```
hWndC = capCreateCaptureWindow("Fereastra de captura",
                                WS_CHILD | WS_VISIBLE , 0, 0, 160, 120, hwndParent, nID);
capDriverConnect(hWndC, 0);
capCaptureSequence(hWndC);
```

După ce aplicația creează o fereastră de captură de tipul clasei AVICap şi o conectează cu un driver video, fereastra de captură este gata pentru a recepționa datele. În acest punct, aplicația trebuie doar să trimită mesajul WM_CAP_SEQUENCE (sau cu macroul capCaptreSequence) pentru a începe captura. Utilizând setările implicite, WM_CAP_SEQUENCE porneşte captura semnalului de intrare video şi audio către un fişier numit CAPTUTRE.AVI. Captura continuă până când unul din următoarele evenimente are loc:

- Utilizatorul apasă tasta *Esc* sau un buton de mouse.
- Aplicație se termină sau se renunță la operația de captură.
- Discul de salvare se umple.

Într-o aplicație, se poate opri captura către un fişier prin trimiterea mesajului WM_CAP_STOP (sau cu macroul capCaptureStop) către fereastra de captură. De asemenea se poate termina operația de captură prin trimiterea mesajului WM_CAP_ABORT (sau prin utilizarea macroului capCaptureAbort) către fereastra de captură.

➢ Opțiuni de bază ale operației de captură

Prin modificarea unuia sau mai multor parametrii definiți în structura CAPTUREPARAMS, programatorul poate realiza următoarele operații:

- Schimbarea ratei de captură a cadrelor.
- Specificarea controlului cu mouse-ul sau cu tastatura pentru încheierea sesiunii de captură.
- Specificarea unei durate de timp pentru sesiunea de captură.

❖ Rata de captură

Rata de captură reprezintă numărul de cadre care sunt capturate în fiecare secundă, pe timpul sesiunii de captură. Se poate obține rata curentă de captură prin utilizarea masajului WM_CAP_GET_SEQUENCE_SETUP (sau cu ajutorul macroului capCaptureSetup). Rata curentă este memorată în membrul dwRequestMicroSecPerFrame al structurii CAPTUREPARAMS. Rata de captură

poate fi setată prin specificarea numărului de microsecunde între cadre succesive ca valoare a acestui membru, după care se trimite structura CAPTUREPARAMS modificată către fereastra de captură folosind mesajul WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup). Valoarea implicită a lui dwRequestMicroSecPerFrame este 666667, ceea ce corespunde la 15 cadre pe secundă.

❖ **Tastele de terminare a capturii**

I se poate permite utilizatorului aplicației terminarea sesiunii de captură prin apăsarea unei taste sau a unei combinații de taste de la tastatură, sau prin apăsarea butonului din dreapta sau stânga al mouse-ului. Dacă utilizatorul întrerupe o sesiune de captură în timp real, conținutul fișierului de captură va fi șters. Dacă utilizatorul întrerupe o sesiune de captură pas cu pas, conținutul fișierului de captură până la punctul de întrerupere este salvat.

Se pot obține setările curente despre întreruperea sesiunii de captură prin folosirea mesajului WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Setările de taste curente sunt stocate în membrul vKeyAbort al structurii CAPTUREPARAMS; setările de mouse curente sunt stocate în membri fAbortLeftMouse și fAbortRightMouse. Se poate seta o nouă tastă sau o nouă combinație de taste (cum ar fi utilizarea combinațiilor cu *Ctrl* sau *Shift*) ca valori pentru vKeyAbort, sau se pot seta butoanele din stânga sau dreapta ale mouse-ului ca tastă de întrerupere prin specificarea membrilor fAbortLeftMouse sau fAbortRightMouse. După setarea acestor membri, se trimite structura modificată CAPTUREPARAMS către fereastra de captură folosind mesajul WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup). Valoarea implicită a membrului vKeyAbort este VK_ESCAPE. Trebuie apelată funcția RegisterHotKey înainte de specificarea tastei care să întrerupă o sesiune de captură. Valorile implicite pentru fAbortLeftMouse și fAbortRightMouse sunt TRUE.

❖ **Limita de timp**

Se poate limita durata sesiunii de captură prin utilizarea membrilor fLimitEnabled și wTimeLimit ai structurii CAPTUREPARAMS. Membrul fLimitEnabled indică faptul că operația de captură este limitată în timp, iar wTimeLimit specifică durata maximă a sesiunii de captură.

Programatorul poate să obțină valorile curente pentru acești membrii prin utilizarea mesajului WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Se poate seta limitarea sesiunii de captură prin specificarea valorii TRUE membrului fLimitEnabled și prin setarea duratei operației de captură prin specificarea în secunde a membrului wTimeLimit. După setarea acestor membri, se trimite structura CAPTUREPARAMS modificată către fereastra de captură folosind mesajul WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup). Valoarea implicită pentru fLimitEnabled este FALSE.

➢ **Fereastra de captură**

Ferestrele de captură sunt din punct de vedere conceptual similare cu controalele standard (așa cum ar fi butoanele, cutiile de tip listă sau barele de scrolare). În mod normal, ferestrele de captură folosesc două stiluri Windows: WS_CHILD și WS_VISIBLE.

❖ **Crearea unei ferestre de captură AVICap**

Se poate crea o fereastră de captură de tipul clasei Windows AVICap prin utilizarea funcției capCreateCaptureWindow. Această funcție returnează un handle de fereastră care identifică fereastra de captură și care este utilizat de aplicație pentru trimiterea de mesaje către respectiva fereastră. Se pot crea una sau mai multe ferestre de captură într-o aplicație și se pot conecta fiecare dintre ele cu un dispozitiv de captură diferit.

❖ **Conectarea unei ferestre de captură cu un driver de captură**

Conectarea sau deconectarea unei ferestre de captură cu un driver de captură se poate face în mod dinamic. Se poate conecta sau asocia o fereastră de captură cu un driver de captură folosind mesajul WM_CAPDRIVER_CONNECT (sau cu macroul capDriverConnect). După ce o fereastră de captură și un driver de captură au fost conectate, se pot trimite mesaje specifice dispozitivului respectiv către driverul de captură asociat ferestrei de captură respective. Dacă există instalate mai multe dispozitive de captură în sistem, se poate conecta o fereastră de captură cu un dispozitiv de captură particular prin specificarea unui valori întregi pentru parametru wParam al mesajului WM_CAP_DRIVER_CONNECT. Valoarea întreagă este

indexul care identifică un driver de captură video listat în registry sau în secţiunea [drivers] a fişierului SYSTEM.INI. Se utilizează zero pentru prima intrare în index.

Se poate obţine numele şi versiunea driverului de captură instalat prin folosirea funcţiei capGetDriverDescription. Aplicaţia poate folosi funcţia pentru a enumera driverele şi dispozitivele instalate, în aşa fel încât utilizatorul să poată selecta ce dispozitiv de captură va fi conectat cu fereastra de captură. Se poate obţine numele driverului dispozitivului de captură conectat cu o fereastră de captură prin folosirea mesajului WM_CAP_DRIVER_GET_NAME (sau cu macroul capDriverGetName). Pentru a se obţine versiunea driverului de captură instalat se utilizează mesajul WM_CAP_DRIVER_GET_VERSION (sau cu macroul capDriverGetVersion).

Se poate deconecta o fereastră de captură de la un driver de captură folosind mesajul WM_CAP_DRIVER_DISCONNECT (sau cu macroul capDriverDisconnect). Când o fereastră de captură este închisă, orice driver de captură conectat cu fereastra respectivă este automat deconectat.

❖ **Interacţiunea ferestrele părinte-copil**

Câteva din mesajele de nivel sistem, cum ar fi WM_PALETTECHNGED şi WM_QUERYNEWPALETTE, sunt trimise doar ferestrelor de nivel top şi suprapuse. Dacă o fereastră de captură este o fereastra copil, părintele ei trebuie să trimită mai departe aceste mesaje.

În mod similar, dacă fereastra părinte îşi schimbă dimensiunile, trebuie să trimită mesaje de notificare către fereastra de captură. Invers, dacă dimensiunile ferestrei de captură se modifică, aceasta trebuie să trimită mesajele de notificare către fereastra părinte.

Cel mai simplu mod pentru a gestiona aceste evenimente este acela de a păstra întotdeauna dimensiunile ferestrei egale cu cele ale imaginii video recepţionate, sesizând părintele de fiecare dată când aceste dimensiuni se modifică.

❖ **Starea ferestrei de captură**

Se poate obţine starea curentă a ferestrei de captură folosind mesajul WM_CAP_GET_STATUS (sau cu macroul capGetStatus). Acest mesaj întoarce o copie a structurii CAPSTATUS având complectaţi membrii acesteia cu valorile lor curente. Structura CAPSTATUS conţine informaţii referitoare la dimensiunile imaginii, poziţia scrolului, şi care din modurile suprapus sau vizionare este pornit.

Deoarece informația reprezentată în CAPSTATUS este dinamică, aplicația trebuie să reînoiască conținutul acestei structuri ori de câte ori dimensiunea formatului semnalului video capturat se poate modifica (ca de exemplu după afișarea formatului video al driverului de captură). Modificarea dimensiunii ferestrei de captură nu are efect asupra dimensiunilor semnalului video capturat. Cutia de dialog despre formatul video afișată de către driverul dispozitivului de captură controlează dimensiunile semnalului de captură video.

➢ **Driverul de captură video şi cel de captură audio**

Un driver de captură video şi hardware-ul lui corespunzător pot dicta diverse aspecte ale capturii video, incluzând sursele video acceptate, opțiunile de afişare, formatele şi opțiunile de compresie. Un driver audio specifică formatul audio şi o compresie opțională ce se utilizează asupra datele audio capturate.

❖ **Capabilitățile driverului de captură video**

Se pot obține capabilitățile hardware ale driverului de captură curent prin folosirea mesajului WM_CAP_DRIVER_GET_CAPS (sau cu macroul capDriverGetCaps). Acest mesaj întoarce capabilitățile driverului de captură şi ale hardware-ului asociat într-o structură CAPDRIVERCAPS.

❖ **Cutiile de dialog video**

Fiecare driver de captură poate oferi până la patru cutii de dialog pentru controlul diferitelor aspecte referitoare la achiziția video şi asupra procesului de captură, cât şi pentru definirea atributelor de compresie utilizate în reducerea dimensiunii datelor video. Conținutul acestor cutii de dialog este definit de către driverul de captură video.

Cutia de dialog Sursa Video controlează selectarea canalului video de intrare şi parametrii care afectează imaginea achiziționată în buferul de cadre. Această cutie de dialog enumeră tipurile de semnale care conectează sursa video cu placa de achiziție (în mod normal SVHS şi intrări compozite), şi asigură controlul asupra luminozității, contrastului şi saturației. Dacă această cutie de dialog este suportată de driverul de captură video, ea se poate afişa şi modifica folosind mesajul WM_CAP_DLG_VIDEOSOURCE (sau cu macroul capDlgVideoSource).

Cutia de dialog Format Video controlează selecția dimensiunii cadrelor imaginii achiziționate, a adâncimii de culoare şi a opțiunilor de compresie ale semnalului video capturat.

În cazul în care cutia de dialog este suportată de către driverul de captură, aceasta se poate afişa folosind mesajul WM_CAP_DLG_VIDEOFORMAT (sau cu macroul capDlgVideoFormat).

Cutia de Afişare Video controlează aspectul semnalului video pe monitor în timpul capturii. Controalele din această cutie nu au nici un efect asupra datelor video achiziționate, ele afectând doar prezentarea semnalului digital. De exemplu, dispozitivele de captură care suportă suprapunere pot permite alterarea luminozității şi saturației, culorii cheie sau alinierii suprapunerii. Dacă cutia de dialog este suportată de către driverul de captură, aceasta poate fi afişată şi modificată folosind mesajul WM_CAP_DLG_VIDEODISPLAY (sau cu macroul capDlgVideoDisplay).

Cutia de dialog Compresie Video controlează atributele de compresie post captură. Dacă cutia este suportată de către driverul de captură, aceasta se poate afişa folosind mesajul WM_CAP_DLG_VIDEOCOMPRESSION (sau cu macroul capDlgVideoCompression).

❖ **Modurile vizionare şi suprapunere**

Un driver de captură poate implementa două moduri de urmărire a unui semnal video: modul vizionare şi modul suprapus. Dacă un driver de captură implementează ambele metode, utilizatorul poate alege pe care din cele două metode vrea să o folosească. Modul vizionare transferă cadrele achiziționate de către hardware-ul de captură în memoria sistem şi afişează aceste cadre într-o fereastră de captură utilizând funcțiile de interfață ale dispozitivului grafic (GDI). Aplicațiile pot să micşoreze rata de vizionare când fereastra părinte pierde focusul şi să mărească rata de vizionare când fereastra părinte reprimeşte focusul. Această acțiune îmbunătățeşte performanțele generale ale sistemului deoarece operațiunea de vizionare consumă foarte mult din timpul de procesor.

Există trei mesaje care controlează operația de vizionare:

- Prin trimiterea mesajului WM_CAP_SET_PREVIEW (sau cu macroul capPreview) către fereastra de captură, se porneşte sau se opreşte modul vizionare

- Prin trimiterea mesajului WM_CAP_SET_PREVIEWRATE (sau cu macroul capPreviewRate) către fereastra de captură, se setează rata de afişare a cadrelor în modul vizionare.
- Prin trimiterea mesajului WM_CAP_SET_SCALE (sau cu macroul capPreviewScale), se setează sau se resetează scalarea semnalului video vizionat.

Când vizionarea şi scalarea sunt ambele setate, cadrul video capturat este scalat la dimensiunile ferestrei de captură. Pornind modul vizionare automat modul suprapus este oprit.

Modul suprapus este o funcţie hardware care afişează conţinutul buferului de captură direct pe monitor fără utilizarea procesorului. Se poate porni sau opri modul suprapus prin trimiterea mesajului WM_CAP_SET_OVERLAY (sau cu macroul capOverlay) către fereastra de captură. Prin pornirea modului suprapus automat se opreşte modul vizionare.

Se poate seta de asemenea şi poziţia de scrolare în cadrul zonei client a ferestrei de captură pentru modurile vizionare şi suprapus prin trimiterea mesajului WM_CAP_SET_SCROLL (sau cu macroul capSetScrollPos) către fereastra de captură.

❖ **Formatul video**

Se poate obţine structura care specifică formatul video sau dimensiunea acestei structuri prin trimiterea mesajului WM_CAP_GET_VIDEOFORMAT (sau cu macrourile capGetVideoFormat şi capGetVideoFormatSize) către fereastra de captură. Se poate seta formatul datelor video capturate prin trimiterea mesajului WM_CAP_SET_VIDEOFORMAT (sau cu macroul capSetVideoFormat) către fereastra de captură.

❖ **Setările capturii video**

Structura CAPTUREPARAMS conţine parametrii de control pentru semnalul video capturat. Această structură controlează câteva aspecte ale procesului de captură, care permit utilizatorului să realizeze următoarele funcţii:

- Specificarea ratei cadrelor.
- Specificarea numărului de bufere video alocate.
- Pornirea sau oprirea capturii audio.

- Specificarea intervalului de timp pentru captură.
- Specificarea faptului că un dispozitiv MCI (VCR sau disc video) este utilizat pe timpul capturii.
- Specificarea controlului de mouse sau de tastatură utilizat pentru terminarea capturii.
- Specificarea tipului de aproximare video utilizat pe timpul capturii.

Se pot obține setările curente din cadrul structurii CAPTUREPARAMS prin trimiterea mesajului WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureSetup) către fereastra de captură. Se pot seta unul sau mai mulți parametrii din această structură prin modificarea membrilor corespunzători din structura CAPTUREPARAMS şi prin trimiterea mesajului WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup) către fereastra de captură.

❖ **Formatul audio**

Se poate obține formatul curent de captură a datelor audio sau dimensiunea structurii ce conține formatul audio prin trimiterea mesajului WM_CAPGET_AUDIOFORMAT (sau cu macrourile capGetAudioFormat şi capGetAudioFormatSize) către fereastra de captură. Formatul audio implicit este mono, pe 8 biți, la 11 kHz PCM (Pulse Code Modulation). Când se obține formatul folosind mesajul WM_CAP_GET_AUDIOFORMAT, întotdeauna se va utiliza structura WAVEFORMATEX.

Se poate seta formatul de captură audio prin trimiterea mesajului WM_CAP_SET_AUDIOFORMAT (sau cu macroul capSetAudioFormat) către fereastra de captură. Când se setează formatul audio, utilizatorul trebuie să trimită un pointer către o structură WAVEFORMAR,WAVEFORMATEX sau PCMWAVEFORMAT, în funcție de formatul audio specificat.

➢ **Fişierul şi buferele de captură**

❖ **Numele fişierului de captură**

Implicit, AVICap, direcționează datele video şi audio de la fereastra de captură către un fişier numit CAPTURE.AVI aflat în directorul rădăcină al discului curent. Se poate da un alt nume acestui fişier prin trimiterea mesajului

WM_CAP_FILE_SET_CAPTURE_FILE (sau cu macroul capFileSetCaptureFile). Se poate obține numele fișierului curent de captură prin trimiterea mesajului WM_CAP_FILE_GET_CAPTURE_FILE (sau cu macroul capFileGetCaptureFile) către fereastra de captură.

❖ **Salvarea datelor capturate într-un fișier nou**

Dacă se dorește salvarea datelor capturate, aplicația trebuie să salveze conținutul fișierului de captură într-un alt fișier folosind mesajul WM_CAP_FILE_SAVEAS (sau cu macroul capFileSaveAs). Acest mesaj nu schimbă numele sau conținutul fișierului de captură. Aplicația trebuie să specifice un nume pentru noul fișier deoarece fișierul de captură își păstrează numele original. În mod normal un fișier de captură are prealocată o dimensiune egală cu cel mai mare segment de captură anticipat, și doar o porțiune a lui va fi utilizată pentru datele capturate. Acest mesaj copiază doar porțiunea din fișier ce conține date capturate.

❖ **Prealocarea spațiului pe disc pentru fișierul de captură**

Prealocarea spațiului pe disc pentru procesul captură creează un fișier de o anumită dimensiune specificată pe disc înainte de începerea operațiunii de captură. Prealocarea reduce procesul necesar salvării pe disc în timpul capturii și are ca efect micșorarea numărului de cadre pierdute. Se poate prealoca un fișier de captură prin utilizarea mesajului WM_CAP_FILE_ALLOCATE (sau cu macroul capFileAlloc).

În mod normal, aplicația ar trebui să prealoce suficient spațiu pe disc pentru a putea conține cel mai mare fișier de captură anticipat. Prealocarea spațiului pe disc nu restricționează dimensiunea fișierului de captură. Dimensiunea fișierului de captură este mărită în mod automat dacă datele capturate depășesc spațiul alocat. Operațiile ulterioare de scriere în fișierul de captură vor reutiliza spațiul alocat pentru fișier, prezervând dimensiunea și fragmentarea acestuia.

Se poate îmbunătăți performanța de captură prin defragmentarea acestui fișier. Pentru a defragmenta fișierul se poate folosi orice utilitar de defragmentare, ca de exemplu Norton Speed Disk sau Disk Defragmenter. Dacă se utilizează un fișier de captură defragmentat, iar ulterior acesta este mărit, este recomandabil să fie defragmentat din nou, deoarece prin mărirea unui fișier de captură defragmentat porțiunea expandată va reduce performanța procesului de captură fiind în majoritatea cazurilor fragmentată. De asemenea se poate îmbunătăți performanța de captură prin

utilizarea unui disc necoprimat. Comprimarea datelor pe timpul capturii limitează rata de transfer între fereastra de captură şi fişier.

Aplicaţiile ar trebui să păstreze un fişier permanent de captură pentru a elimina timpul prealocării şi defragmentării acestuia de fiecare dată când se porneşte aplicaţia. Deoarece un fişier de captură necesită foarte mult spaţiu pe disc, iar prealocarea şterge toate datele dintr-un fişier de captură existent, aplicaţiile trebuie să permită utilizatorului alegerea tipului de fişier de captură folosit, permanent sau temporar.

❖ **Dimensiunea indexului**

Fiecare fişier AVI foloseşte un index de dimensiune specificată pentru a localiza segmentele de date audio şi video din cadrul fişierului. O intrare în acest index localizează un cadru video sau un bufer audio. Astfel, valoarea dimensiunii indexului, indirect, determină limita superioară a numărului de cadre care pot fi capturate în cadrul fişierului. Se poate obţine dimensiunea curentă a indexului prin utilizarea mesajului WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Dimensiunea curentă a indexului este stocată în membrul dwIndexSize al structurii CAPTUREPARAMS. Se poate specifica o nouă dimensiune a indexului prin modificarea valorii membrului dwIndexSize şi trimiterea structurii CAPTUREPARAMS modificate către fereastra de captură prin utilizarea mesajului WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup). Dimensiunea implicită a indexului este de 34.952 de intrări (asigurând 32K de cadre şi un număr proporţional de bufere audio).

❖ **Granularitatea segmentelor video şi audio**

Granularitatea segmentelor este dimensiunea logică a unui bloc pentru un fişier AVI şi este folosită pentru scrierea şi citirea segmentelor de date audio şi video. Când se scriu segmente de date audio şi video pe disc, AVICap adaugă atâtea segmente de umplere câte sunt necesare pentru a umple fiecare bloc logic de date. Se poate obţine granularitatea curentă a segmentelor cu ajutorul mesajului WM_CAP_DET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Granularitatea curentă este stocată în membrul wChunkGranularity al structurii CAPTUREPARAMS. Se poate modifica această valoare şi se poate trimite noua structură CAPTUREPARAMS modificată către fereastra de captură prin utilizarea mesajului WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul

capCaptureSetSetup). De asemenea se poate da valoarea zero pentru granularitate, astfel că ea este setată la dimensiunea sectoarelor de pe disc.

❖ **Buferele video**

Buferele folosite de către captura video se află în memoria heap. Numărul buferelor folosite într-o operație de captură variază în funcție de valoarea membrului wNumVideoRequested a structurii CAPTUREPARAMS şi de memoria disponibilă. Se poate obține valoarea curentă a buferelor video prin utilizarea mesajului WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Se poate modifica locul şi numărul acestor bufere prin modificarea valorii acestui membru şi prin trimiterea structurii CAPTUREPARAMS modificate, către fereastra de captură cu mesajul WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup).

❖ **Buferele audio**

Există trei modalități de control ale porțiunii audio din cadrul operației de captură:

- Includerea sau excluderea părții audio din operația de captură.
- Cererea unui anumit număr de bufere audio.
- Cererea unei anumite dimensiuni pentru buferele audio.

Se pot obține setările curente pentru buferele audio cu ajutorul mesajului WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Membrul fCaptureAudio al structurii CAPTUREPARAMS specifică dacă partea audio este inclusă sau exclusă din operația de captură. Numărul curent de bufere audio este stocat în membrul wNumAudioRequested, iar dimensiunea curentă a buferelor audio este stocată în membrul dwAudioBufferSize. Se pot modifica aceşti parametrii după care se trimite structura CAPTUREPARAMS către fereastra de captură cu ajutorul mesajului WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup). Implicit, partea audio este inclusă în operația de captură şi îi sunt alocate patru bufere audio. Dimensiunea unui bufer audio implicită este de 10K de date audio sau de 0,5 secunde în funcție de mărimea lor.

➢ Tipuri de capturi disponibile

Pe lângă captura continuă bazată pe un interval constant de timp, AVICap suportă şi următoarele tipuri de capturi:

- Captura de cadre manuală.
- Captura imaginilor statice.
- Captura fără utilizarea discului magnetic.
- Captura continuă de la un dispozitiv MCI (în timp real şi pas cu pas).

❖ Captura manuală

Dacă se doreşte specificarea individuală a cadrelor dintr-un semnal video continuu, se poate controla secvenţierea folosind mesajele WM_CAP_SINGLE_FRAME_OPEN, WM_CAP_SINGLE_FRAME şi WM_CAP_SINGLE_FRAME_CLOSE (sau macrourile capCaptureSingleFrameOpen, capCAptureSingleFrame şi capCaptureSingleFrameClose). În mod normal aceste mesaje sunt utilizate pentru crearea de animaţie prin concatenarea cadrelor individuale într-un fişier de captură.

WM_CAP_SINGLE_FRAME_OPEN deschide un fişier pentru o operaţie de captură condusă manual.

WM_CAP_SINGLE_FRAME capturează cadre individuale şi le adaugă fişierului de captură.

WM_CAP_SINGLE_FRAME_CLOSE închide fişierul utilizat pentru captura cadrelor manual.

Acest tip de captură nu se poate folosi pentru captura simultană audio şi video.

❖ Capturarea imaginilor statice

Dacă se doreşte capturarea unui singur cadru ca o imagine statică, se poate folosi mesajul WM_CAP_GRAB_FRAME_NONSTOP sau WM_CAP GRAB_FRAME (sau cu macrourile capGrabFrameNonStop şi capGrabFrame) pentru capturarea imaginii achiziţionate într-un bufer cadru intern. Se poate îngheţa afişarea imaginii capturate folosind mesajul WM_CAP_GRAB_FRAME. Dacă nu se doreşte îngheţarea se foloseşte mesajul WM_CAP_GRAB_FRAME_NONSTOP.

Odată capturată, se poate copia imaginea pentru a o utiliza în alte aplicaţii. Se poate copia imaginea din buferul cadru în clipboard, utilizând mesajul

WM_CAP_EIT_COPY (sau cu macroul capEditCopy). De asemenea se poate copia imaginea din buferul cadru într-un bitmap independent de dispozitiv (DIB) prin mesajul WM_CAP_FILE_SAVEDIB (sau cu macroul capFileSaveDIB). Aplicația poate folosi şi ea cele două mesaje de captură a cadrelor statice pentru a edita o secvență cadru cu cadru, sau pentru a crea o secvență de fotografii în care să se observe trecerea timpului.

❖ **Captura fără utilizarea discului magnetic**

Se pot folosi serviciile de captură fără scrierea datelor într-un fişier pe disc folosind WM_CAP_SEQUENCE_NOFILE (sau cu macroul capCaptureSequenceNoFile). Acest mesaj este util doar în conjuncție cu funcțiile callback care permit aplicației să utilizeze datele video şi audio în mod direct. De exemplu, aplicațiile de tip videoconferință pot folosi acest mesaj pentru a obține cadre din semnalul video continuu. Funcțiile callback vor transfera imaginile capturate către calculatorul destinație.

❖ **Captura continuă de la un dispozitiv MCI**

Dispozitivele MCI folosesc operația de captură în două moduri: captură în timp real şi captură pas cu pas. Se poate selecta un dispozitiv MCI, aşa cum ar fi un videodisc sau un video recorder (VCR), pentru a fi sursa video în cadrul operației de captură prin mesajul WM_CAP_SET_MCI_DEVICE.(sau cu macroul capSetMCIDeviceName) având specificat numele dispozitivului. Se poate de asemenea obține numele dispozitivului curent prin mesajul WM_CAP_GET_MCI_DEVICE (sau cu macroul capGetMCIDeviceName).

În captura în timp real, fereastra de captură sincronizează operația de captură şi compensează întârzierile datorate poziționării sursei video MCI şi inițializează resursele (aşa cum ar fi buferele de captură) necesare pentru datele capturate. Fereastra de captură aşteaptă ca un dispozitiv MCI valid să fie instalat în sistem pentru a captura date în acest mod.

Specificațiile pentru controlul dispozitivului MCI sunt stocate în membrii structurii CAPTUREPARAMS. Sursele video compatibile MCI conțin video recordere şi discuri laser. Dacă membrul fMCIControl este setat la TRUE, atunci fereastra de captură coordonează operația MCI. Fereastra de captură foloseşte parametrii specificați în membrii dwMCIStartTime şi dwMCIStopTime pentru a

obține în milisecunde, pozițiile de strat și stop ale secvenței. Dacă valoarea lui fMCIControl este FALSE, sursa video nu este tratată ca un dispozitiv MCI și conținutul membrilor dwMCIStartTime și dwMCIStopTime este ignorat.

Se poate folosi utilitarul Windows Media Player pentru o verificare rapidă a conectării corecte a dispozitivului MCI la sistem. Pornind un dispozitiv cu Media Player se verifică corectitudinea configurației pentru dispozitivul MCI. Dacă apare o imagine pe ecranul video, sursa video este conectată corect la hardware-ul de captură.

- **Opțiuni avansate de captură**

❖ **Măsurarea calității video**

Una din metodele de măsurare a calității video este limitarea numărului de cadre pierdute pe timpul procesului de captură. Când captura continuă s-a terminat, valoarea de calitate este comparată cu procentul cadrelor pierdute din numărul total de cadre. Dacă procentul cadrelor pierdute este mai mare decât valoarea membrului wPercentDropForError al structurii CAPTUREPARAMS, AVICap trimite un mesaj de eroare către funcția callback de eroare (dacă aceasta este instalată).

Se poate obține limita curentă a cadrelor pierdute (exprimată în procente) cu mesajul WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Se poate seta o nouă limită prin specificarea procentului în membrul wPercentDropForError al structurii CAPTUREPARAMS, și trimiterea acestei structuri către fereastra de captură cu mesajul WM_CAP_SET_SEQUENCE_SETUP (sau cu macroul capCaptureSetSetup). Valoarea implicită a acestui membru este 10%.

❖ **Captură inițiată de către utilizator**

Se poate obține valoarea curentă a fanionului care indică captura inițiată de către utilizator cu mesajul WM_CAP_GET_SEQUENCE_SETUP (sau cu macroul capCaptureGetSetup). Valoarea acestui fanion este stocată în membrul fMakeUserHitOKToCapture din cadrul structurii CAPTUREPARAMS. Acest fanion, dacă este setat la TRUE, permite utilizatorului un control exact al momentului de început al sesiunii de captură. Dacă acest fanion este setat pe TRUE, AVICap afișează o cutie de dialog după alocarea tuturor buferelor video și audio pentru sesiunea de captură curentă. Aceasta, permite utilizatorului să elimine întârzierile de captură datorate inițializărilor software. Dacă aplicația utilizează un număr mic de bufere

video, această cutie de dialog nu mai este necesară. Valoarea implicită a acestui fanion este FALSE.

❖ **Lucrul cu palete**

Inițial, dacă formatul de captură video necesită o paletă, fereastra de captură foloseşte paleta furnizată de către driverul de captură. Această paletă poate consta din valori de nivele de gri pentru o reproducere alb-negru, sau o largă selecție de valori colorate. Se poate obține o paletă existentă, pentru a înlocui paleta implicită prin mesajele WM_CAP_PAL_PASTE sau WM_CAP_PAL_OPEN (sau cu macrourile capPalettePaste sau capPaletteOpen). În mod alternativ se poate crea o paletă dedicată pentru a înlocui paleta implicită cu mesajele WM_CAP_PAL_AUTOCREATE sau WM_CAP_PAL_MANUALCREATE (sau cu macrourile capPaletteAuto sau capPaletteManual). După înlocuirea paletei implicite, fereastra şi driverul de captură vor folosi noua paletă până la crearea sau deschiderea uneia noi.

Mesajele WM_CAP_PAL_AUTOCREATE şi WM_CAP_PAL_MANUALCREATE, creează o paletă optimizată bazată pe semnalul video curent de intrare. Această paletă dedicată oferă unei secvențe video cea mai bună fidelitate de culoare deoarece este bazată pe culorile existente în semnalul video. Fereastra de captură creează o histogramă tridimensională a culorilor din cadrul semnalului de probă. După aceasta, este redus numărul de culori prin examinarea erorii absolute dintre culorile adiacente şi prin consolidarea acestora cu valoarea cea mai mică de eroare.

Când se trimite mesajul WM_CAP_PAL_AUTOCREATE, trebuie specificat numărul de cadre al semnalului de probă şi dimensiunea paletei. Când se specifică numărul de cadre, trebuie incluse suficiente cadre pentru a asigura prezența tuturor culorilor din semnalul de probă.

Dacă se utilizează mesajul WM_CAP_PAL_MANUALCREATE, atunci vor trebui selectate manual o serie de cadre semnificative pentru a crea o paletă care să conțină culorile dorite de utilizator.

O paletă poate conține până la 256 de culori. Dacă se concatenează palete sau dacă secvența video trebuie afişată simultan cu altă secvență video sau imagine, trebuie utilizată o selecție de culori mai mică astfel încât culorile diferitelor imagini sau secvențe video, să coexiste.

Se poate salva o nouă paletă cu mesajul WM_CAP_PAL_SAVE (sau cu macroul capPaletteSave), iar ulterior se poate deschide cu mesajul WM_CAP_PAL_OPEN. Se poate salva o paletă pentru procesarea ei ulterioară sau pentru utilizarea ei în alte aplicații.

Se poate copia o paletă de pe clipboard în fereastra de captură folosind mesajul WM_CAP_PAL_PASTE. Fereastra de captură trimite paletă către driverul de captură. Alte aplicații pot copia paleta pe clipboard. Se poate de asemenea copia paleta pe clipboard prin folosirea mesajului WM_CAP_EDIT_COPY (sau cu macroul capEditCopy). Acest mesaj copiază pe clipboard buferul cadrului video curent, inclusiv paleta.

❖ **Introducerea de segmente de informație în fișierul AVI**

Se pot insera segmente de informație, cum ar fi text sau date dedicate, în fișierul AVI cu mesajul WM_CAP_FILE_SET_INFOCHUNK (sau cu macroul capFileSetInfoChunk). Se poate folosi acest mesaj și la ștergerea de segmente de informații din fișierul AVI.

❖ **Mesaje de date utilizator**

Se pot asocia date cu fereastra de captură prin utilizarea mesajelor WM_CAP_GET_USER_DATA sau WM_CAP_SET_USER_DATA (sau cu macrourile capGetUserData sau capSetUserData). Se poate obține o valoare de date de tip LONG cu mesajul WM_CAP_GET_USER_DATA și se poate seta o valoare LONG de date, cu mesajul WM_CAP_SET_USER_DATA.

➢ **Funcțiile callback ale clasei AVICap**

Aplicațiile pot înregistra funcții callback cu fereastra de captură astfel încât aceasta să le sesizeze modificarea de stare, apariția unei erori, faptul că buferele video și audio sunt disponibile, sau tratarea diferitelor funcții necesare ale aplicației pe timpul capturii continue.

❖ **Controlul precis al capturii**

Mesajul WM_CAP_SET_CALLBACK_CAPCONTORL specifică funcția callback din cadrul aplicației care este apelată pentru a asigura un control asupra

punctelor de start şi stop ale capturii. Se poate folosi şi macroul capSetCallbackOnCapControl.

Primul mesaj trimis de către driverul de captură către funcţia callback setează parametrul nState pe CONTROLCALLBACK_PREROLL, după alocarea tuturor buferelor şi după ce toate pregătirile pentru captură au fost încheiate. Acest mesaj permite aplicaţiei să poată derula sursele video. (Funcţia callback specifică nState ca al doilea parametru al ei). Funcţia callback revine la codul apelant exact în momentul începerii înregistrării. O valoare de întoarcere TRUE din partea funcţiei callback permite continuarea capturii. O valoare FALSE abandonează captura. Odată ce captura a început, funcţia callback este chemată frecvent, cu parametrul nState pe CONTROLCALLBACK_CAPTURING pentru a permite aplicaţiei încheierea capturii prin întoarcerea valorii FALSE de către funcţia callback.

❖ **Tratarea erorilor**

Mesajul WM_CAP_SET_CALLBACK_ERROR specifică funcţia callback din cadrul aplicaţiei care este apelată când apare o eroare AVICap de tipul:

- lipsa spaţiului pe disc
- încercarea de scriere într-un fişier read-only
- imposibilitatea conectării cu placa de achiziţie
- pierderea prea multor cadre

Se poate folosi şi macroul capSetCallbackOnError. Conţinutul notificării de eroare conţine un identificator de mesaj şi un text formatat gata pentru afişare. Aplicaţia poate folosi identificatorul de mesaj pentru a filtra notificările în scopul de a limita mesajele afişate utilizatorului. Un identificator de mesaj de valoare zero indică faptul că începe o nouă operaţie, funcţia callback trebuind să şteargă orice mesaj de eroare afişat.

❖ **Notificarea de cadre**

Mesajul WM_CAP_SET_CALLBACK_FRAME specifică funcţia callback din cadrul aplicaţiei care este chemată când este disponibil un nou cadru capturat pentru vizionare. Se poate folosi şi macroul capSetCallbackOnFrame.

Fereastra de captură permite aceste notificări doar dacă rata de vizionare este diferită de zero şi dacă captura continuă nu este în curs de desfăşurare.

❖ **Funcția callback de stare**

Mesajul WM_CAP_SET_CALLBACK_STATUS specifică funcția callback din cadrul aplicației care este apelată pe timpul capturări unui semnal video pe disc sau pe parcursul altor operații de durată, pentru a notifica aplicația despre derularea operației. Se poate folosi şi macroul capSetCallbackOnStatus.

Informația de stare include un identificator de mesaj şi un text formatat gata pentru a fi afişat. Aplicația poate folosi identificatorul de mesaj pentru a filtra notificările în scopul de a limita mesajele afişate utilizatorului. Pe parcursul operației de captură, primul mesaj trimis funcției callback este întotdeauna IDS_CAP_BEGIN iar ultimul este întotdeauna IDS_CAP_END. Un identificator de mesaj de valoare zero indică începutul unei noi operații iar funcția callback trebuie să-şi reseteze starea curentă.

❖ **Buferele video**

Mesajul WM_CAP_SET_CALLBACK_VIDEOSTREAM specifică funcția callback din cadrul aplicației care este apelată în timpul capturii continue când un nou bufer video devine disponibil, adică pentru procesarea uni cadru video capturat. Se poate folosi şi macroul capSetCallbackOnVideoStream.

Fereastra de captură cheamă această funcție callback chiar înainte de a scrie fiecare cadru capturat pe disc.

❖ **Buferele audio**

Mesajul WM_CAP_SET_CALLBACK_WAVESTREAM specifică funcția callback din cadrul aplicației care este apelată în timpul capturii continue când un nou bufer video devine disponibil pentru a fi procesat. Se poate folosi şi macroul capSetCallbackOnWaveStream.

Fereastra de captură apelează funcția callback chiar înainte de a scrie fiecare bufer audio pe disc.

❖ **Tratarea diferitelor funcții necesare ale aplicației pe timpul procesului de captură**

Mesajul WM_CAP_SET_CALLBACK_YIELD specifică funcția callback din cadrul aplicației care este chemată pe timpul capturii continue atunci când se redă,

pentru puțin timp controlul aplicației și preia controlul pe timpul procesului de captură. Se poate folosi și macroul capSetCallbackOnYield.

De obicei o astfel de funcție constă dintr-o buclă în care sunt chemate funcțiile PeekMessage, TranslateMessage și DispatchMessage. Fereastra de captură cheamă această funcție callback cel puțin o dată pentru fiecare cadru video capturat, dar rata exactă depinde de rata de achiziție a cadrelor și de timpul necesar pentru scrierea pe disc.

➢ Utilizarea AVICap

❖ Crearea unei ferestre de captură

Se prezintă un exemplu prin care se creează o fereastră de captură folosind funcția capCreateCaptureWindow:

```
hWndC = capCreateCaptureWindow((LPSTR) "Fereastra de captură", // numele ferestrei
                               WS_CHILD | WS_VISIBLE,      // stilul ferestrei
                               0, 0, 160, 120,             // poziția și dimensiunile ferestrei
                               (HWND) hwndParent,
                               (int) nID /* ID-ul copilului */);
```

❖ Conectarea cu un driver de captură

Următorul exemplu conectează fereastra de captură având handle-ul hWndC cu driverul MSVIDEO și o deconectează după aceea cu macroul capDriverDisconnect:

```
fOK = SendMessage (hWndC, WM_CAP_DRIVER_CONNECT, 0, 0L);
//
// Sau, se folosește macroul pentru conectarea cu driverul MSVIDEO:
// fOK = capDriverConnect(hWndC, 0);
//
// Aici se pune codul pentru setări și pentru începerea capturii video.
//
capDriverDisconnect (hWndC);
```

❖ Enumerarea driverelor de captură instalate

Următorul exemplu folosește funcția capGetDriverDescription pentru a obține numele și versiunea driverelor de captură instalate:

```
char szDeviceName[80];
char szDeviceVersion[80];

for (wIndex = 0; wIndex < 10; wIndex++)
  {
  if (capGetDriverDescription (wIndex, szDeviceName,  sizeof (szDeviceName), szDeviceVersion,
                               sizeof (szDeviceVersion))
```

```
    {
    // Se adaugă numele listei de driveruri şi se permite
    // utilizatorului selectarea unuia dintre ele.
    }
  }
```

❖ Obținerea capabilităților driverului de captură

Mesajul WM_CAP_DRIVER_GET_CAPS întoarce capabilitățile driverului de captură şi a hardware-ului existent în structura CAPDRIVERCAPS. De fiecare dată când o aplicație conectează un nou driver la fereastra de captură, trebuie să actualizeze structura CAPDRIVERCAPS. Spre exemplu se foloseşte macroul capDriverCaps pentru obținerea capabilităților driverului:

```
CAPDRIVERCAPS CapDrvCaps;

SendMessage (hWndC, WM_CAP_DRIVER_GET_CAPS,
                sizeof (CAPDRIVERCAPS), (LONG) (LPVOID) &CapDrvCaps);

// Sau, se foloseşte macroul pentru obținerea capabilităților  driverului.
// capDriverGetCaps(hWndC, &CapDrvCaps, sizeof (CAPDRIVERCAPS));
```

❖ Obținerea stării ferestrei de captură

Următorul exemplu foloseşte funcția SetWindowPos pentru setarea dimensiunii ferestrei de captură la dimensiunile semnalului video recepționat obținute cu macroul capGetStatus în structura CAPSTATUS:

```
CAPSTATUS CapStatus;

capGetStatus(hWndC, &CapStatus, sizeof (CAPSTATUS));

SetWindowPos(hWndC, NULL, 0, 0, CapStatus.uiImageWidth,
                CapStatus.uiImageHeight, SWP_NOZORDER | SWP_NOMOVE);
```

❖ Afişarea cutiilor de dialog pentru setarea caracteristicilor video

Fiecare driver de captură poate oferi până la trei cutii de dialog diferite pentru controlul aspectelor legate de achiziția video şi procesul de captură. Se prezintă un exemplu ce demonstrează cum se afişează aceste cutii de dialog. Înainte de afişarea fiecărei cutii de dialog, se cheamă macroul capDriverGetCaps şi se verifică în structura CAPDRIVERCAPS dacă driverul de captură poate afişa respectiva cutie.

```
CAPDRIVERCAPS CapDrvCaps:

capDriverGetCaps(hWndC, &CapDrvCaps, sizeof (CAPDRIVERCAPS));
 // Cutia de dialog a sursei video.
```

```
if (CapDriverCaps.fHasDlgVideoSource)
  capDlgVideoSource(hWndC);
 // Cutia de dialog a formatului video.
if (CapDriverCaps.fHasDlgVideoFormat)
  {
  capDlgVideoFormat(hWndC);
  // Există dimensiuni noi de imagine?
  capGetStatus(hWndC, &CapStatus, sizeof (CAPSTATUS));
  // Dacă da, semnalizează părintelui schimbarea dimensiunilor.
  }
 // Cutia de dialog a afişării video.
if (CapDriverCaps.fHasDlgVideoDisplay)
  capDlgVideoDisplay(hWndC);
```

❖ Obținerea şi setarea formatului video

Structura BITMAPINFO este de lungime variabilă, pentru a putea cuprinde diverse formate de date standard şi comprimate. Din acest motiv, aplicațiile trebuie întotdeauna să afle în prealabil dimensiunea acestei structuri şi să aloce memoria înainte de obținerea formatului video curent. Se prezintă un exemplu ce foloseşte macroul capGetVideoFormatSize pentru obținerea dimensiunii buferului, iar după aceea macroul capGetVideoFormat pentru obținerea formatului video curent:

```
LPBITMAPINFO lpbi;
DWORD dwSize;

dwSize = capGetVideoFormatSize(hWndC);
lpbi = GlobalAllocPtr (GHND, dwSize);
capGetVideoFormat(hWndC, lpbi, dwSize);
// Citirea formatului video şi eliberarea memoriei alocate.
```

Aplicațiile pot folosi macroul capSetVideoFormat (sau mesajul WM_CAP_SET_VIDEOFORMAT) pentru trimiterea unui antet BITMAPINFO către fereastra de captură. Deoarece formatele video sunt specifice fiecărui dispozitiv, aplicațiile trebuie să verifice valoarea întoarsă pentru a constata dacă respectivul format a fost acceptat.

❖ Vizionarea semnalului video

Pentru setarea ratei de afişare a cadrelor se foloseşte macroul capPreviewRate pentru modul vizionare la 66 milisecunde pe cadru, iar după aceea, macroul capPreview pentru punerea ferestrei de captură în modul vizionare:

```
capPreviewRate(hWndC, 66);    // rata, în milisecunde
capPreview(hWndC, TRUE);      // pornirea vizionării
// Vizionare
```

```
capPreview(hWnd, FALSE);        // oprirea vizionării
```

❖ Activarea modului suprapus

Pentru a determina dacă driverul de captură are capabilități de suprapunere se foloseşte macroul capDriverGetCaps iar dacă da, se activează modul suprapus:

```
CAPDRIVERCAPS CapDrvCaps;

capDriverGetCaps(hWndC, &CapDrvCaps, sizeof (CAPDRIVERCAPS));
if (CapDrvCaps.fHasOverlay)
  capOverlay(hWndC, TRUE);
```

❖ Denumirea fişierului de captură

Pentru a specifica un alt nume (TEST.AVI) pentru fişierul de captură se foloseşte macroul capFileSetCaptureFile şi se utilizează macroul capFileAlloc pentru a prealoca un fişier de 5 MB:

```
char szCaptureFile[] = "TEST.AVI";

capFileSetCaptureFile( hWndC, szCaptureFile);
capFileAlloc( hWndC, (1024L * 1024L * 5));
```

❖ Formatarea capturii audio

Pentru setarea formatului audio la 11-kHz PCM 8 biți, stereo se foloseşte macroul capSetAudioFormat:

```
WAVEFORMATEX wfex;

wfex.wFormatTag = WAVE_FORMAT_PCM;
wfex.nChannels = 2;             // Use stereo
wfex.nSamplesPerSec = 11025;
wfex.nAvgBytesPerSec = 22050;
wfex.nBlockAlign = 2;
wfex.wBitsPerSample = 8;
wfex.cbSize = 0;

capSetAudioFormat(hWndC, &wfex, sizeof(WAVEFORMATEX));
```

❖ Schimbarea setării capturii video

Pentru a schimba rata de captură de la valoarea implicită (15 cadre pe secundă) la 10 cadre pe secundă se folosesc macrourile capCaptureGetSetup şi capCaptureSetSetup:

```
CAPTUREPARMS CaptureParms;
float FramesPerSec = 10.0;

capCaptureGetSetup(hWndC, &CaptureParms, sizeof(CAPTUREPARMS));

CaptureParms.dwRequestMicroSecPerFrame = (DWORD) (1.0e6 / FramesPerSec);
capCaptureSetSetup(hWndC, &CaptureParms, sizeof (CAPTUREPARMS));
```

❖ Captura datelor

Pentru a porni captura video se foloseşte macroul capCaptureSequence şi macroul capFileSaveAs pentru a copia datele capturate din fişierul de captură în fişierul TEST.AVI:

```
char szNewName[] = "TEST.AVI";

// Pornirea operaţiei de captură.
capCaptureSequence(hWndC);
// Captura.
capFileSaveAs(hWndC, szNewName);
```

❖ Includerea unui segment de informaţie

Dacă se doreşte includerea altor informaţii în aplicaţie, în plus faţă de cele audio şi video, se pot crea segmente de informaţie şi se pot insera în fişierul de captură. Segmentele de informaţie pot conţine diferite tipuri de informaţie, inclusiv detaliile legate de copyright, identificarea sursei video sau informaţii de temporizare exterioare.

Se prezintă stocarea informaţiilor de temporizare (un cod de timp de tip SMPTE-Society of Motion Picture and Television Engineers) într-un segment de informaţie şi adăugarea acestui segment la fişierul de captură folosind macroul capFileSetInfoChunk:

```
// Acest exemplu presupune că aplicaţia controlează sursa video
// pentru derulare înainte şi înapoi.
CAPINFOCHUNK cic;

cic.fccInfoID = infotypeSMPTE_TIME;
cic.lpData = "00:20:30:12";
cic.cbData = strlen (cic.lpData) + 1;
capFileSetInfoChunk (hwndC, &cic);
```

❖ Adăugarea unei funcții callback la aplicație

O aplicație poate să înregistreze funcții callback cu fereastra de captură în așa fel încât aceasta să semnalizeze aplicației următoarele situații:

- Schimbarea stării.
- Apariția unei erori.
- Devenirea unui bufer video sau audio disponibil.
- Cedarea controlului aplicației pe timpul capturii continue.

Se prezintă într-un exemplu crearea unei ferestră de captură și înregistrarea funcțiilor callback pentru stare, eroare, captură continuă și cadre, în bucla de procesare a mesajelor a aplicației. Exemplul include de asemenea prezentarea dezactivării unei funcții callback.

```
case WM_CREATE:
  {
  char    achDeviceName[80] ;
  char    achDeviceVersion[100] ;
  char    achBuffer[100] ;
  WORD    wDriverCount = 0 ;
  WORD    wIndex ;
  WORD    wError ;
  HMENU   hMenu ;

  // Crearea unei ferestre de captură folosind macroul
  // capCreateCaptureWindow.
  ghWndCap = capCreateCaptureWindow((LPSTR)"Fereastra de captura",
     WS_CHILD | WS_VISIBLE, 0, 0, 160, 120, (HWND) hWnd, (int) 0);

  // Înregistrarea funcției callback pentru erori cu macroul
  // capSetCallbackOnError.
  capSetCallbackOnError(ghWndCap, fpErrorCallback);

  // Înregistrarea funcției callback pentru stare cu macroul
  // capSetCallbackOnStatus
  capSetCallbackOnStatus(ghWndCap, fpStatusCallback);

  // Înregistrarea funcției callback pentru captura continuă cu
  // macroul capSetCallbackOnVideoStream
  capSetCallbackOnVideoStream(ghWndCap, fpVideoCallback);

  // Înregistrarea funcției callback pentru cadre cu macroul
  // capSetCallbackOnFrame macro.
  capSetCallbackOnFrame(ghWndCap, fpFrameCallback<);

  // Conectarea cu driverul de captură

  break;
  }
case WM_CLOSE:
  {
  // Utilizarea macroului capSetCallbackOnFrame pentru dezactivarea
  // funcției callback pentru cadre. Apeluri similare există și pentru
```

```
// celelalte tipuri de funcţii callback.

capSetCallbackOnFrame(hWndC, NULL);

break;
}
```

❖ Crearea funcţiei callback de stare

Următorul exemplu este o funcţie callback de stare. Aceasta se înregistrează cu macroul capSetCallbackOnStatus:

```
// StatusCallbackProc: funcţia callback de stare
// hWnd:            handle-ul ferestrei de captură
// nID:             codul de stare pentru starea curentă
// lpStatusText:    textul de stare pentru starea curentă
//
LRESULT PASCAL StatusCallbackProc(HWND hWnd, int nID,  LPSTR lpStatusText)
{
  if (!ghWndMain)
    return FALSE;

  if (nID == 0)
    {       // Ştergerea mesajului de stare vechi.
    SetWindowText(ghWndMain, (LPSTR) gachAppName);
    return (LRESULT) TRUE;
    }
  // Afişarea codului şi textului de stare...
  wsprintf(gachBuffer, "Status# %d: %s", nID, lpStatusText);

 SetWindowText(ghWndMain, (LPSTR)gachBuffer);
 return (LRESULT) TRUE;
}
```

❖ Crearea unei funcţii callback de eroare

Pentru prezentarea unei funcţii callback de eroare se utilizează macroul capSetCallbackOnError:

```
// ErrorCallbackProc: funcţia callback de eroare
// hWnd:            handle-ul ferestrei de captură
// nErrID:          codul de eroare pentru eroarea întâlnită
// lpErrorText:     textul de eroare pentru eroarea întâlnită
//
LRESULT PASCAL ErrorCallbackProc(HWND hWnd, int nErrID, LPSTR lpErrorText)
{
  if (!ghWndMain)
    return FALSE;
  if (nErrID == 0)          // Pornirea unei noi funcţii.
    return TRUE;          // Ştergerea vechilor erori.

  // Afişarea identificatorului de eroare şi a textului.
  wsprintf(gachBuffer, "Error# %d", nErrID);

  MessageBox(hWnd, lpErrorText, gachBuffer,  MB_OK | MB_ICONEXCLAMATION);
```

```
  return (LRESULT) TRUE;
 }
```

❖ Crearea unei funcții callback de cadre

Înregistrarea unei funcții callback de cadre se face cu macroul capSetCallbackOnFrame:

```
// FrameCallbackProc: funcția callback de cadre
// hWnd:          handle-ul ferestrei de captură
// lpVHdr:        pointer la structura ce conține informația
//                cadrului capturat
//
LRESULT PASCAL FrameCallbackProc(HWND hWnd, LPVIDEOHDR lpVHdr)
{
  if (!ghWndMain)
    return FALSE;

  wsprintf(gachBuffer, "Rata de vizionare# %ld ", gdwFrameNum++);
  SetWindowText(ghWndMain, (LPSTR)gachBuffer);
  return (LRESULT) TRUE ;
}
```

În continuare se prezintă funcțiile implementate pentru captura imaginii în cadrul programului principal:

- Funcția de enumerare a driverelor de captură existente în sistem:

```
BOOL vidcapEnumerateDrivers(void)
{
 #define MAXVIDDRIVERS       10
 char   achDeviceVersion[80] ;
 char   achDeviceAndVersion[160] ;
 UINT    uIndex ;
 short   gDriverCount = 0 ;

 for (uIndex = 0 ; uIndex < MAXVIDDRIVERS ; uIndex++)
   {
   if (capGetDriverDescription(uIndex,
                               (LPSTR)achDeviceAndVersion, sizeof(achDeviceAndVersion),
                               (LPSTR)achDeviceVersion, sizeof(achDeviceVersion)))
     {
     // Concatenate the device name and version strings
     lstrcat (achDeviceAndVersion, ",  ");
     lstrcat (achDeviceAndVersion, achDeviceVersion);
    gDriverCount++;
     }
   else
     break;
   }
 return (gDriverCount);
}
```

- Funcția de conectare a ferestrei de captură cu driverul de captură

```
BOOL vidcapInitHardware(HWND hwndCap, UINT uIndex)
{
```

```
 char    szName[MAX_PATH];
 char    szVersion[MAX_PATH];
 BOOL    gbHaveHardware;

// Try connecting to the capture driver
if (!capDriverConnect(hwndCap, uIndex))
  {
  MessageBox(NULL,"Error initializing video capture device","Roboview",MB_OK);
  gbHaveHardware = FALSE;
  }
else
    gbHaveHardware = TRUE;
// Get the capabilities of the capture driver
capDriverGetCaps(hwndCap, &gCapDriverCaps, sizeof(CAPDRIVERCAPS)) ;
// Get the settings for the capture window
capGetStatus(hwndCap, &gCapStatus , sizeof(gCapStatus));
// Unlike all other capture drivers, Scrncap.drv needs to use
// a Yield callback, and we don't want to abort on mouse clicks,
// so determine if the current driver is Scrncap.drv
capGetDriverDescription (uIndex,   szName, sizeof (szName),  szVersion, sizeof (szVersion));
// set the preview rate (units are millisecs)
capPreviewRate(hwndCap,1);
capPreviewScale(hwndCap,TRUE);
// set live/overlay to default
capPreview(hwndCap,TRUE);
return gbHaveHardware;
}
```

- Funcția de creare a ferestrei de captură :

```
HWND vidframeCreate(int x,int y,int cx,int cy,HWND FAR * phwndCap)
{
 HWND hwnd, hwndCap;
 WNDCLASS wc;
  static FirstTime = 1;

 if(FirstTime)
  {
  wc.lpszClassName = "VideoCapture";
  wc.hInstance     = GetMainInstance();
  wc.lpfnWndProc   = DefWindowProc;
  wc.hCursor       = LoadCursor(NULL, IDC_ARROW) ;
  wc.hIcon         = NULL;
  wc.lpszMenuName  = NULL;
  wc.hbrBackground = NULL;
  wc.style         = CS_HREDRAW | CS_VREDRAW ;
  wc.cbClsExtra    = 0 ;
  wc.cbWndExtra    = 0 ;
  if(!RegisterClass(&wc))
    MessageBox(NULL,"Error registring the class","Roboview",MB_OK);
  FirstTime = 0;
  }
 hwnd = CreateWindow(
                    "VideoCapture",
                    "VideoCapture",
                    WS_CHILD | WS_VISIBLE | WS_CLIPCHILDREN | WS_THICKFRAME,
                    x, y, cx, cy,
                    GetMainWindowHwnd(),
                    (HMENU) 0,
                    GetMainInstance(),
```

```
                    NULL);
 // create an AVICAP window within this window. Leave vidframeLayout to do the layout
 hwndCap = capCreateCaptureWindow(
                    NULL,
                    WS_CHILD | WS_VISIBLE,
                    0, 0, cx, cy,
                    hwnd,          // parent window
                    1              // child window id
                    );
 if (hwndCap == NULL)
  return(NULL);
 *phwndCap = hwndCap;
 return(hwnd);
}
```

- Funcția principală de captură video :

```
void VideoCapture(void)
{
 ghWndFrame = vidframeCreate(652,32,146,110,&ghWndCap);
 if ((ghWndFrame == NULL) || (ghWndCap == NULL))
  MessageBox(NULL,"Window initialization failure","Roboview",MB_OK);
 // Get the default setup for video capture from the AVICap window
 BOOL test = capCaptureGetSetup(ghWndCap, &gCapParms, sizeof(CAPTUREPARMS)) ;
 // Create a list of all capture drivers and append them to the Options menu
 if (!vidcapEnumerateDrivers())
  MessageBox(NULL,"No video capture device found","Roboview",MB_OK);
 // Try to connect to a capture driver
 else
  if (vidcapInitHardware(ghWndCap,0))
   {
   VideoCaptureInit = TRUE;
   // Hooray, we now have a capture driver connected!
   //    vidcapSetCaptureFile(gachCaptureFile);
   }
 return;
}
```

1.3 Fundamentarea teoretică

Prezenta teză de doctorat prezintă mai multe metode de preprocesare folosite în sistemele de vedere artificială. Deşi numărul tehnicilor disponibile pentru preprocesarea imaginilor este semnificativ, doar o submulțime din acestea satisfac cerințele de viteză şi cost de implementare redus, elemente esențiale într-un sistem industrial de vedere. Gama de metode discutate în această lucrare satisface aceste două cerințe.

Se vor considera două metode de bază în preprocesare:

- Metodă bazată pe tehnici ce utilizează domenii spațiale
- Metodă bazată pe conceptul de domenii de frecvență cu ajutorul transformatei Fourier

Împreună, aceste două metode acoperă majoritatea algoritmilor folosiți în sistemele de vedere artificială.

Se va utiliza notația f(x,y) [111],[115] pentru a descrie o imagine bidimensională, unde x şi y reprezintă coordonatele în planul imaginii, iar valoarea f în fiecare punct (x,y) este proporțională cu intensitatea luminoasă a imaginii în acel punct. Figura 1.3.1 ilustrează acest concept, împreună cu sistemul de coordonate care se utilizează ulterior.

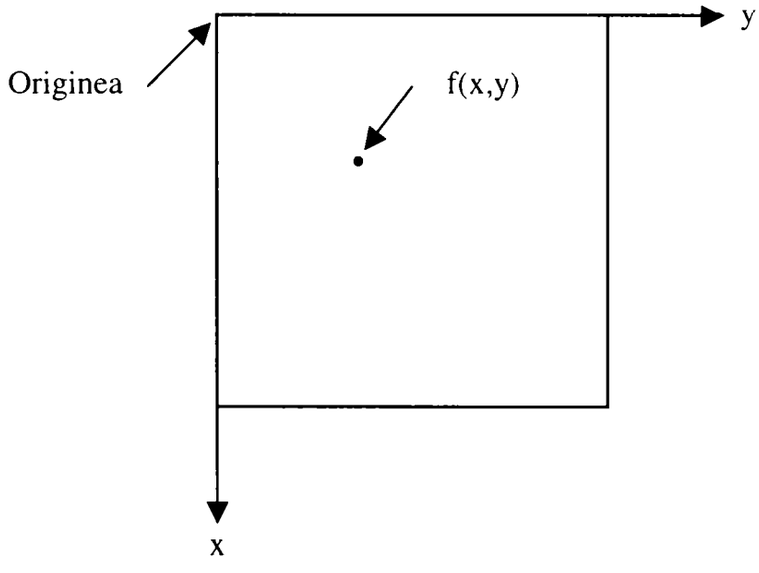


Fig. 1.3.1
Sistemul de coordonate pentru reprezentarea unei imagini

Se va utiliza în plus şi variabila z pentru a nota variaţia de intensitate într-o imagine când locaţia spaţială a acestei variaţii nu este importantă.

În scopul de a fi reprezentată într-o formă adecvată procesării pe calculator, funcţia de imagine f(x,y) trebuie să fie digitizată atât spaţial cât şi în amplitudine (intensitate). Digitizarea spaţiului de coordonate (x,y) poartă numele ***de eşantionarea imaginii***, iar digitizarea amplitudinii poartă numele de ***cuantificarea nivelelor de gri***. Ultimul termen este aplicabil imaginilor monocrome şi reflectă faptul că aceste imagini variază de la negru la alb în nivele de gri. Termenii de intensitate şi nivel de gri reprezintă în acest context acelaşi lucru.

Având dată o imagine continuă, după eşantionarea uniformă într-o matrice de N linii şi M coloane, în care fiecare celulă este de asemenea cuantificată în intensitate se va obţine în final o imagine digitală, care are următoarea reprezentare:

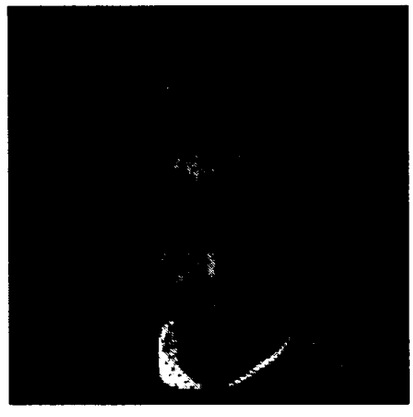
$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & f(0,M-1) \\ f(1,0) & f(1,1) & f(1,M-1) \\ \cdots & \cdots & \cdots \\ f(N-1,0) & f(N-1,1) & f(N-1,M-1) \end{bmatrix} \qquad (1.3.1)$$

unde x şi y sunt acum variabile discrete: x = 0, 1, 2, ..., N-1; y = 0, 1, 2, ..., M-1. Fiecare element din matrice este numit ***element de imagine*** sau ***pixel***. Făcând o referinţă la imaginea din figura 1.3.1, trebuie remarcat faptul că f(0,0) reprezintă pixelul de origine al imaginii, f(0,1) pixelul din dreapta lui şi aşa mai departe. În practică se obişnuieşte ca valorile lui N şi M împreună cu numărul de nivele de gri să fie puteri ale lui 2.

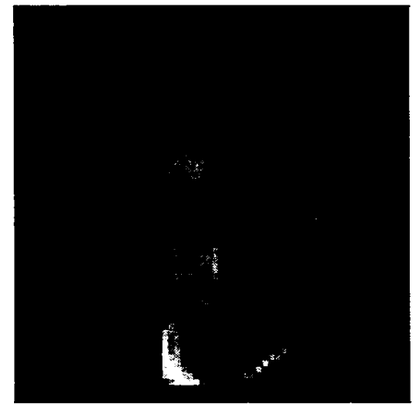
În figura 1.3.2 este prezentat un exemplu de eşantionare. În figura 1.3.2,a este prezentată o imagine eşantionată într-o matrice de N x N pixeli cu N = 512. Intensitatea fiecărui pixel este cuantificată în unul din cele 256 de nivele de gri. Figurile 1.3.2,b până la 1.3.2,e prezintă aceeaşi imagine dar cu N = 256, 128, 64 şi 32. În toate cazurile numărul de intensităţi permise este acelaşi, 256. Deoarece zona de afişare folosită pentru fiecare imagine este aceeaşi (512 x 512 puncte de afişare), pixelii din imaginile cu rezoluţii mici sunt duplicaţi în scopul umplerii întregii suprafeţe de afişare. Acest lucru produce un efect de tablă de şah care este vizibil în special în imaginile de rezoluţie foarte joasă. Imaginile cu N egal cu 512, 256 şi 128 au o calitate acceptabilă faţă de cele cu N egal cu 64 sau 32.

(a)

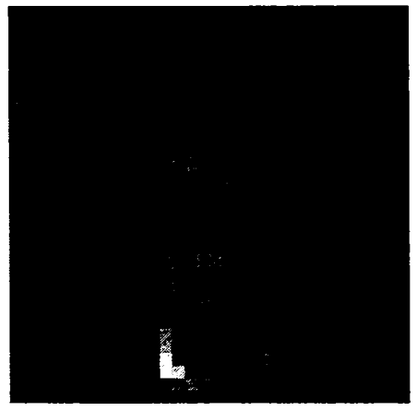
(b)



(c)



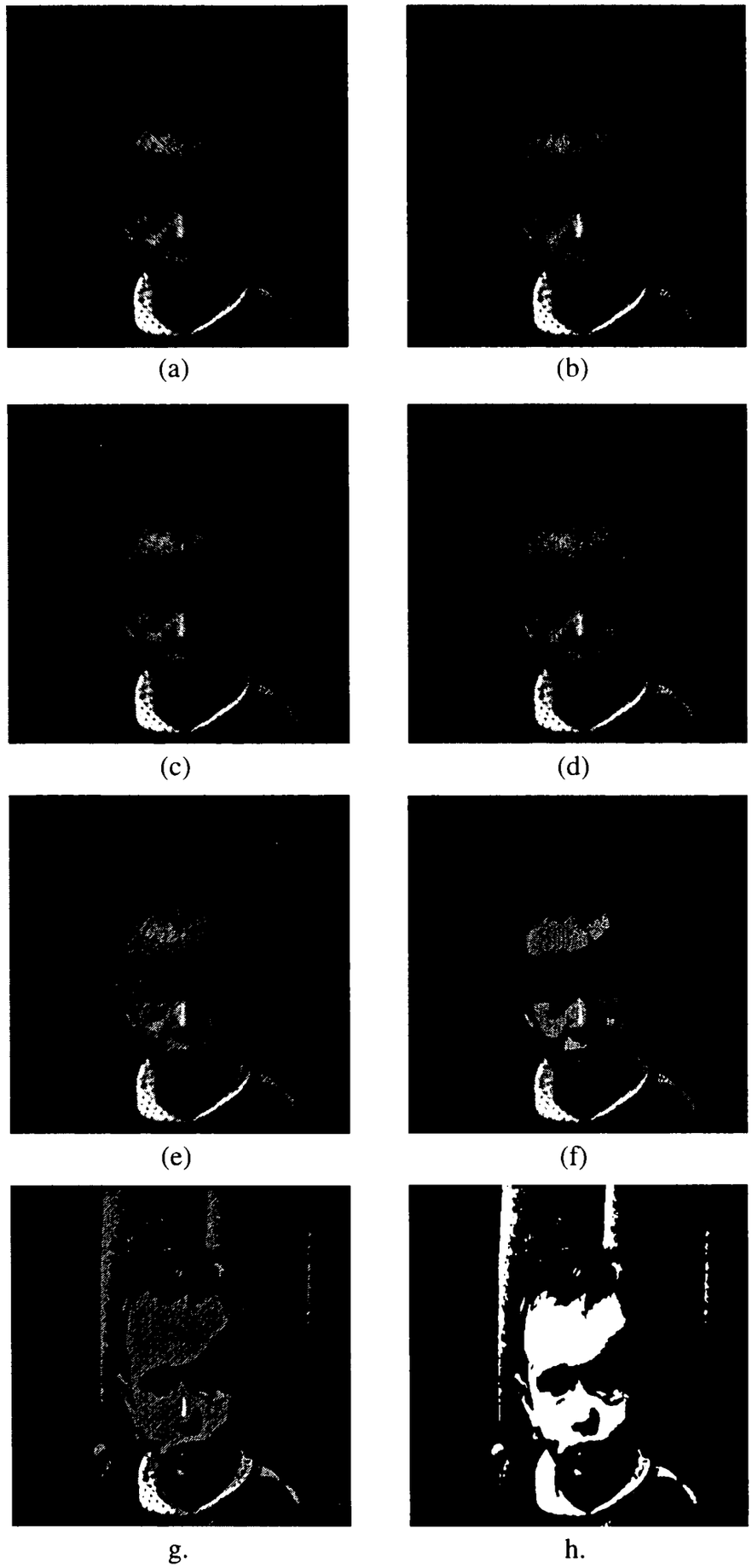
(d)



(e)

Fig. 1.3.2
Efectele reduceri dimensiunii rastrului de eşantionare.
(a) 512 x 512
(b) 256 x 256
(c) 128 x 128
(d) 64 x 64
(e) 32 x 32

Figura 1.3.3 ilustrează efectul produs de reducerea numărului de nivele de gri în timp ce rezoluția spațială se păstrează constantă la 512 x 512. Imaginile cu 256, 128 sau 64 de nivele de gri sunt bune din punct de vedere calitativ. Imaginea cu 32 de nivele de gri începe să prezinte o uşoară degradare (în special în zonele cu intensitate aproape constantă) ca un rezultat al utilizării unui număr mai mic de nivele de intensitate pentru reprezentarea fiecărui pixel. Acest efect este mult mai vizibil în imaginea cu doar 16 nivele de gri apărând efectul de contur fals şi creşte pe măsură ce numărul de nivele de gri scade.



(a) (b)

(c) (d)

(e) (f)

g. h.

Fig. 1.3.3. O imagine 512 x 512 afişată în 256, 128, 64, 32, 16, 8, 4, 2 nivele de gri

Numărul de pixeli şi de nivele de gri necesare pentru a produce o reproducere utilă (din punctul de vedere al robotului) a unei imagini originale depinde de imaginea în sine cât şi de aplicaţia în care se foloseşte. Spre exemplu pentru obţinerea unei calităţi comparabile cu cea a unui televizor alb negru este necesară asigurarea a 512 x 512 pixeli cu 128 de nivele de gri.

1.3.1 Tehnici de iluminare

Iluminarea unei scene este un factor important care afectează deseori complexitatea algoritmilor de vedere. Iluminarea arbitrară a unui câmp de lucru nu este acceptabilă deoarece poate genera imagini cu contrast scăzut, reflexii nedorite, umbre şi accentuarea detaliilor neesenţiale. Un sistem de iluminare bine proiectat, asigură o iluminare a scenei în aşa fel încât complexitatea imaginii rezultate să fie minimă, iar informaţia necesară pentru detecţia şi extragerea obiectelor să fie îmbunătăţită.

Figura 1.3.4 prezintă patru dintre principalele scheme de iluminare folosite într-un spaţiu de lucru al unui robot [120], [123].

Iluminarea cu lumină difuză prezentată în figura 1.3.4,a poate fi folosită pentru obiecte ce sunt caracterizate de suprafeţe netede şi regulate. De obicei această schemă de iluminare este utilizată în aplicaţiile în care caracteristicile de suprafaţă sunt importante.

Iluminarea din spate prezentată în figura 1.3.4,b produce o imagine binară în alb şi negru. Această tehnică este ideală pentru aplicaţiile în care silueta obiectelor este suficientă pentru recunoaştere şi măsură.

Iluminarea structurată prezentată în figura 1.3.4,c constă din proiectarea de puncte, dungi sau grile peste suprafaţa de lucru. Această tehnică de iluminare are două avantaje importante. În primul rând, ea stabileşte un model de lumină cunoscut peste spaţiul de lucru, iar perturbările acestui model indică prezenţa unui obiect, astfel simplificându-se problema detecţiei obiectelor. În al doilea rând, prin analizarea modului în care este distorsionat modelul de lumină, este posibilă determinarea caracteristicilor tridimensionale ale obiectului.

Iluminarea cu lumină direcțională prezentată în figura 1.3.4,d este utilizată în mod special pentru inspectarea suprafețelor obiectelor. Defectele de suprafață, de tipul cavități sau zgârieturi, pot fi detectate prin folosirea unei raze de lumină foarte bine direcționate (de exemplu o rază laser) şi măsurarea dispersiei. Pentru suprafețele netede, foarte puțină lumină este difuzată spre cameră. Pe de altă parte, prezența unei cavități va mări cantitatea de lumină difuzată către cameră, facilitând astfel detecția defectului.

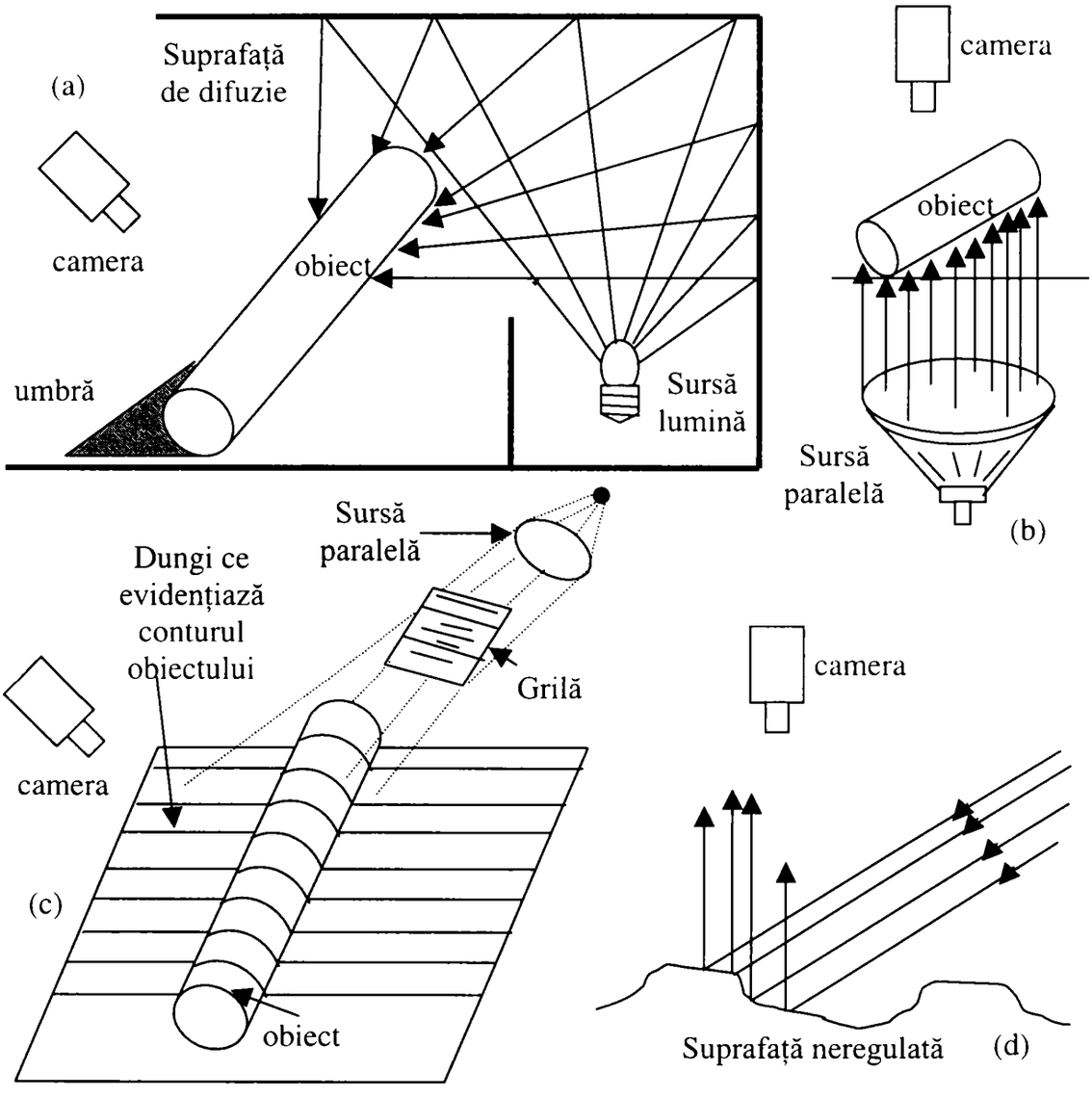


Fig. 1.3.4 Scheme diferite de iluminare

1.3.2 Metode bazate pe domeniile spațiale

Domeniile spațiale se referă la mulțimea de pixeli ce compun o imagine, iar metodele bazate pe domeniile spațiale sunt proceduri care operează direct asupra

acestor pixeli. O funcție de preprocesare într-un domeniu spațial poate fi exprimată ca:

$$g(x,y) = h[f(x,y)] \qquad (1.3.2)$$

unde:

- f(x, y) este imaginea sursă
- g(x, y) este imaginea preprocesată
- h este un operator asupra lui f, definit peste o vecinătate a lui (x,y).

Este de asemenea posibil ca h să opereze asupra unui set de imagini sursă, cum ar fi calculul unei sume de K imagini pentru reducerea zgomotului.

Principala metodă folosită în definirea unei vecinătăți în jurul pixelului (x,y) [137] este folosirea unei subimagini dreptunghiulare sau pătratice centrată în (x,y), așa cum se arată în figura 1.3.5. Centrul subimaginii este deplasat pixel cu pixel pornind, de exemplu, din colțul din stânga sus, și aplicând operatorul la fiecare locație (x,y) pentru a obține pe g(x,y). Se pot folosi și alte forme pentru vecinătăți, cum ar fi cercul, dar matricile pătratice sunt cel mai des folosite din cauza ușurinței implementării lor.

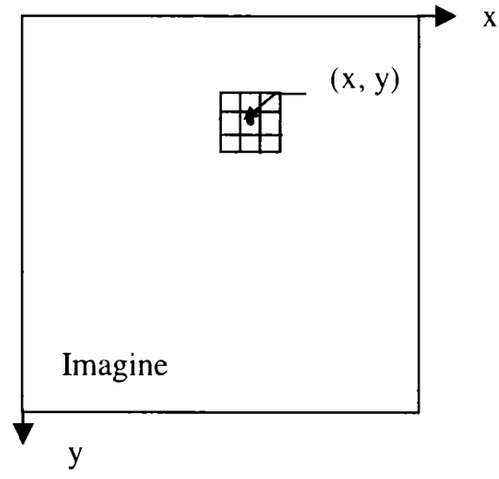


Fig. 1.3.5
O vecinătate 3x3 in jurul
punctului (x,y) dintr-o imagine

Cea mai simpla formă a lui h este aceea care consideră o vecinătate de 1x1 și astfel, g depinde doar de valoarea lui f în punctul (x,y). În acest caz h devine o hartă a intensității sau o transformare T de forma :

$$s = T(r) \qquad (1.3.3)$$

unde, pentru simplificare, s-au folosit s şi r ca variabile reprezentând intensitatea lui f(x,y) respectiv g(x,y) în orice punct (x,y).

Una din tehnicile bazate pe domeniile spaţiale folosite frecvent este utilizarea aşa numitelor măşti de convoluţie (cunoscute şi ca şabloane, ferestre sau filtre). De fapt o mască este o matrice de mici dimensiuni (de exemplu 3x3) ca şi cea din figura 1.3.5, a cărei coeficienţi sunt aleşi în aşa fel încât să poată detecta o anumită proprietate dintr-o imagine. Se presupune, spre exemplu, o imagine de intensitate constantă care conţine pixeli izolaţi a căror intensitate este diferită de cea de fond. Aceste puncte pot fi detectate folosind o mască ca în figura 1.3.6.

-1	-1	-1
-1	8	-1
-1	-1	-1

Fig. 1.3.6
Masca pentru detectarea punctelor izolate diferite de un fond constant

Procedura este următoarea: centrul măştii (numerotat cu 8) este deplasat pe întreaga imagine după indicaţiile de mai jos. Pentru fiecare pixel din imagine, se va multiplica fiecare pixel care este conţinut în aria măştii cu coeficientul din mască corespunzător. Astfel pixelul din centrul măştii este multiplicat cu 8, în timp ce, cei 8 vecini ai lui sunt multiplicaţi cu -1. Rezultatele acestor nouă multiplicări sunt adunate. Dacă toţi pixelii din aria măştii au aceeaşi valoare (fond constant), suma va fi egală cu zero. Dacă pe de altă parte, centrul măştii este deasupra unui punct izolat, suma va fi diferită de zero. Dacă punctul izolat nu va fi în centrul măştii, suma va fi de asemenea diferită de zero, dar magnitudinea răspunsului va fi mai mică. Acest răspuns mai slab poate fi eliminat comparând suma cu un prag.

Conform figurii 1.3.7, dacă se vor reprezenta coeficienții măştii prin $w_1, w_2, ..., w_9$ şi se vor considera cei 8 vecini ai lui (x,y), se va putea generaliza exemplul precedent prin următoarea operație pe o vecinătate de 3x3 a lui (x,y):

$$h[f(x,y)] = w_1 f(x-1, y-1) + w_2 f(x-1, y) + w_3 f(x-1, y+1) + w_4 f(x, y-1) +$$

$$w_5 f(x,y) + w_6 f(x, y+1) + w_7 f(x+1, y-1) + w_8 f(x+1, y) + w_0 f(x+1, y+1) \quad (1.3.4)$$

Este de precizat faptul că noțiunea de procesare bazată pe vecinătăți nu este limitată la o arie de 3x3 şi nici doar la cazul tratat De exemplu, se mai folosesc operații în vecinătăți şi pentru reducerea zgomotului, pentru obținerea unor praguri variabile de imagine, pentru calculul măsurilor de structură, pentru obținerea scheletului unui obiect, etc.

w_1 (x-1,y-1)	w_2 (x-1,y)	w_3 (x-1,y+1)
w_4 (x, y-1)	w_5 (x, y)	w_6 (x, y+1)
w_7 (x+1,y-1)	w_8 (x+1,y)	w_9 (x+1,y-+1)

Fig. 1.3.7
Masca generală 3 x 3 pentru vizualizarea coeficienților şi a locațiilor de pixeli corespunzătoare din imagine

1.3.3 Metode bazate pe domenii de frecvență.

Domeniile de frecvență se referă la o mulțime complexă de pixeli rezultată din aplicarea ***transformatei Fourier*** unei imagini [24]. Conceptul de ***frecvență*** este des folosit în interpretarea transformatei Fourier şi a luat naştere din faptul că această transformare particulară este compusă din reprezentări complexe. Datorită necesarului mare de calcul, metodele bazate pe domeniile de frecvență nu sunt chiar aşa de răspândite în vederea artificială ca cele bazate pe domeniile spațiale. Cu toate acestea transformata Fourier joacă un rol important în domenii ca analiza mişcării unui obiect,

descrierea unui obiect, etc. Multe din tehnicile spațiale pentru îmbunătățire și restaurare sunt bazate pe concepte a căror origine se află în formularea transformatei Fourier.

Se consideră funcții discrete de o variabilă f(x), x = 0,1,2,...,N-1. Transformata Fourier a funcției f(x) este definită astfel :

$$F(u)=\frac{1}{N}\sum_{x=0}^{N-1}F(x)e^{-jux2\pi/N} \tag{1.3.5}$$

pentru u = 0,1,2,...,N-1. În această ecuație $j=\sqrt{-1}$ și u este așa numita ***variabilă de frecvență***. Inversa transformatei Fourier a funcției F(u) este f(x) și este definită ca :

$$f(x)=\sum_{u=0}^{N-1}F(u)e^{jux2\pi/N} \tag{1.3.6}$$

pentru x = 0,1,2,...,N-1. Validitatea acestor expresii, numite ***perechea de transformate Fourier***, este ușor de verificat prin înlocuirea ecuației (1.3.5) pentru F(u) în ecuația (1.3.6), sau invers. În ambele cazuri se va obține o identitate.

O implementare directă a ecuației (1.3.5) pentru u = 0, 1, 2, ..., N-1 va necesita un număr de adunări și multiplicări de ordinul N^2. Folosirea unui algoritm rapid pentru calculul transformatei Fourier (FFT) reduce semnificativ acest număr la $N\log_2 N$, unde N este o putere întreagă a lui 2. Aceleași comentarii sunt valabile și pentru ecuația (1.3.6) pentru x = 0, 1, 2, ..., N-1 . Există deja un număr mare de algoritmi pentru FFT într-o largă varietate de limbaje de programare.

Perechea de transformări Fourier bidimensionale pentru o imagine NxN este definită ca :

$$F(u,v)=\frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi(ux+vy)/N} \tag{1.3.7}$$

pentru u, v = 0, 1, 2, ..., N-1, și :

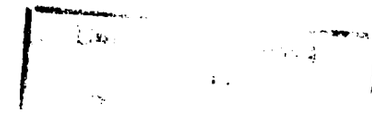
$$f(x,y)=\frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi(ux+vy)/N} \tag{1.3.8}$$

pentru x, y = 0, 1, 2, ..., N-1.

Se poate demonstra că, folosind mici artificii matematice, fiecare din aceste două ecuații poate fi exprimată separat ca o sumă unidimensională ca cea din ecuația (1.3.5) sau (1.3.6). Aceasta conduce la un algoritm secvențial pentru calculul transformatei Fourier bidimensionale folosind doar un algoritm FFT unidimensional: în primul rând se calculează și se salvează transformata fiecărui rând din f(x,y), în

acest fel realizând o matrice bidimensională de rezultate intermediare. Aceste rezultate sunt multiplicate cu N şi astfel este calculată transformata pentru fiecare coloană. Rezultatul final este F(u,v). În acelaşi mod se procedează pentru calculul f(x,y) când se dă F(u,v). Ordinea calcului rând-coloană poate fi inversată în coloană-rând fără afectarea rezultatului final.

Transformata Fourier poate fi utilizată în numeroase moduri într-un sistem de vedere artificială. De exemplu, tratând conturul unui obiect ca şi o matrice unidimensionala de puncte şi calculând transformata lor Fourier, valorile selectate ale lui F(u) pot fi utilizate ca descriptori a formei conturului. Transformata Fourier unidimensională poate fi folosită de asemenea ca un puternic mijloc pentru detecţia mişcării obiectelor. Transformata Fourier discretă bidimensională se aplică în reconstrucţia de imagini, îmbunătăţirea imaginii şi restaurări. Totuşi folosirea acestui mod de prelucrare a imaginii este puţin folosit în vederea artificială utilizată în robotică datorită calculelor numeroase necesare pentru implementare.



1.4 Filtrarea imaginilor

Operațiile de filtrare sunt folosite pentru reducerea zgomotului şi a altor efecte perturbante care pot fi prezente într-o imagine ca rezultat al eşantionării, transmisiei sau perturbațiilor din mediu din timpul achiziției imaginilor. Se prezintă câteva metode rapide de filtrare care sunt posibile de implementat în cadrul unui sistem de vedere artificială.

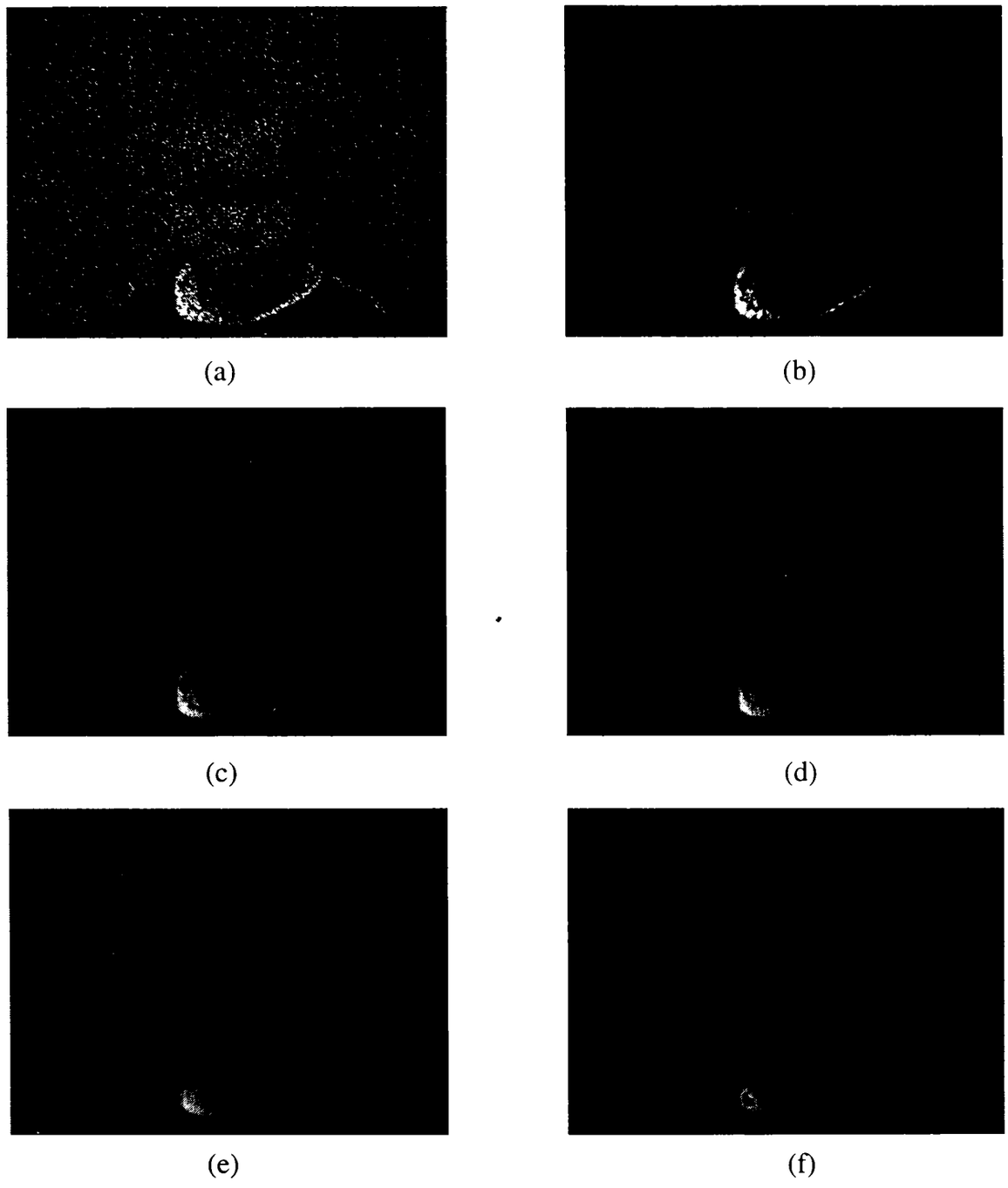
1.4.1 Metoda mediei vecinătății

Este o tehnică bazată pe domeniile spațiale de filtrare a imaginii. Fiind dată o imagine f(x,y), procedura va genera o imagine filtrată g(x,y) a cărei intensitate în fiecare punct (x,y) se obține prin medierea valorilor de intensitate a pixelilor din f conținuți într-o vecinătate predefinită a lui (x,y) [122]. Cu alte cuvinte imaginea filtrată se obține folosind relația

$$g(x,y)=\frac{1}{P}\sum_{(n,m)\in S} f(n,m) \tag{1.4.1}$$

pentru toți x şi y din f(x,y). *S* reprezintă mulțimea de coordonate a punctelor din vecinătatea lui (x,y), inclusiv (x,y), iar *P* este numărul total de puncte din vecinătate. Dacă se foloseşte o vecinătate 3x3 se va observa prin comparație că ecuația (1.4.1) este un caz special al ecuației (1.3.4), cu $w_i = 1/9$. Se observă că ecuația (1.4.1) nu impune limitarea la vecinătăți pătratice, dar, aşa cum s-a menționat, acestea sunt cele predominante în sistemele de vedere artificială. În urma testării cu vecinătăți de diferite dimensiuni (3x3, 5x5, 7x7, 11x11) se remarcă faptul că gradul de filtrare depinde foarte mult de dimensiunea vecinătății utilizate. Ca la orice procesare ce foloseşte o mască, valoarea filtrată a fiecărui pixel este determinată înainte ca orice pixel să fie modificat.

Figura 1.4.1 prezintă efectul acestui tip de filtrare asupra unei imagini cu zgomot, în care nivelul de zgomot este de 50%. Figura 1.4.1,a reprezintă imaginea originală, iar figurile 1.4.1,b până la 1.4.1,f reprezintă rezultatul aplicării filtrului folosind vecinătăți de dimensiuni 3x3, 5x5, 7x7, 9x9 şi 11x11.



(a) (b)

(c) (d)

(e) (f)

Fig. 1.4.1

(a) Imaginea cu zgomot; (b)-(f) rezultatul aplicării unui filtru bazat pe tehnica mediei vecinătății, pe o vecinătate: 3x3, 5x5, 7x7, 9x9, 11x11.

Implementarea acestui filtru este prezentată prin funcția:

```
void CMainFrame::OnToolsSmoothingNeighbourhoodAveraging()
{
 CROBOVIEWDoc  *SourceDoc;
 CROBOVIEWDoc  *DestinDoc;
 CNeighbourhoodDimensionsDlg nd_dlg;
 short  m_x = 3;
 short  m_y = 3;
 short  x,y,x_local,y_local;
```

```
short DestinValue;
short count;

nd_dlg.m_x.Format("%d",m_x);
nd_dlg.m_y.Format("%d",m_x);
if(nd_dlg.DoModal() == IDOK)
 {
 sscanf(nd_dlg.m_x,"%d",&m_x);
 sscanf(nd_dlg.m_y,"%d",&m_y);
 }
else
 return;
SourceDoc = GetActiveSourceDocument();
DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
for(x=0;x<Xmax;x++)
 for(y=0;y<REALYMAX;y++)
  {
  DestinValue = 0;
  count = 0;
  for(x_local=x-(short)(m_x/2);x_local<=x+(short)(m_x/2);x_local++)
   for(y_local=y-(short)(m_y/2);y_local<=y+(short)(m_y/2);y_local++)
    if(x_local>= 0 && x_local<Xmax && y_local>=0 && y_local<REALYMAX)
     {
     DestinValue += SourceDoc->BitmapMatrix[x_local][y_local][0];
     count++;
     }
  DestinValue = DestinValue/(count);
  DestinDoc->BitmapMatrix[x][y][0]=DestinDoc->BitmapMatrix[x][y][1]=
                                   DestinDoc->BitmapMatrix[x][y][2] = (BYTE)DestinValue;
  }
}
```

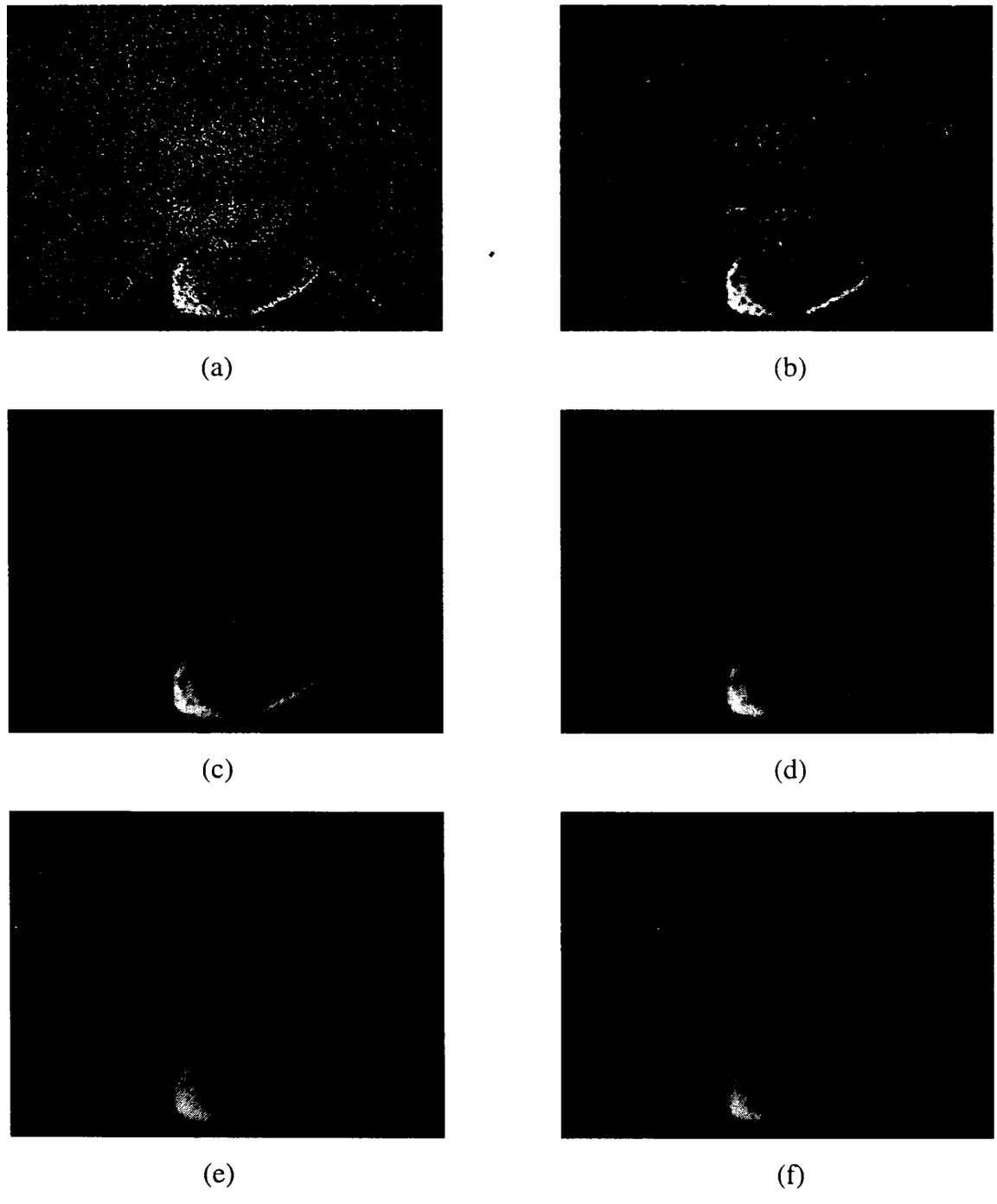
1.4.2 Metoda filtrării mediane

Una din principalele dificultăți apărute în metoda mediei vecinătății este aceea că generează o neclaritate a contururilor şi a altor detalii evidențiate. Această neclaritate se poate reduce în mare parte folosind aşa numitul filtru median, în care se va înlocui intensitatea fiecărui pixel cu valoarea de mijloc a intensităților dintr-o vecinătate predefinită a unui pixel, în loc de media valorilor [122].

Se menționează că valoarea de mijloc M a unei mulțimi de valori este aceea pentru care jumătate din valori sunt mai mici decât M iar cealaltă jumătate sunt mai mari decât M. Pentru a realiza filtrarea mediană într-o vecinătate a unui pixel, se vor sorta valorile pixelului şi ale vecinilor lui, după care se va determina valoarea de mijloc, şi se va atribui pixelului acea valoare. De exemplu, într-o vecinătate 3x3, valoarea de mijloc este cea de-a 5-a valoare în ordine crescătoare, într-o vecinătate 5x5, cea de-a 13-a, etc. Când mai multe valori dintr-o vecinătate sunt la fel, se vor grupa toate valorile egale după cum urmează: se presupune că într-o vecinătate 3x3

există valorile (10, 20, 20, 20, 15, 20, 20, 25, 100). Prin sortare rezultă: (10, 15, 20, 20, 20, 20, 20, 25, 100), al căror mijloc este 20.

În concluzie, este de remarcat că principala funcție a filtrării mediene este forțarea punctelor cu intensități foarte distincte să se apropie mai mult de vecinii lor, în acest fel eliminând vârfurile de intensitate care apar izolate în regiunea măştii de filtrare. În cazul concentrărilor de zgomot se pot aplica două sau mai multe treceri prin filtrul median pentru eliminarea lor.



(a) (b)

(c) (d)

(e) (f)

Fig. 1.4.2

(a) Imaginea cu zgomot; (b)-(f) rezultatul aplicării unui filtru median pe o vecinătate 3x3, 5x5, 7x7, 9x9, 11x11.

Figura 1.4.2 prezintă efectul acestui tip de filtrare asupra unei imagini cu zgomot, în care nivelul de zgomot este de 50%. Figura 1.4.2,a reprezintă imaginea originală, iar figurile 1.4.2,b până la 1.4.2,f reprezintă rezultatul aplicării filtrului folosind vecinătăți de dimensiuni 3x3, 5x5, 7x7, 9x9 şi 11x11. Se observă îmbunătățirea evidentă a imaginii față de media vecinătății, mai ales pentru a vecinătate 5x5;

Implementarea acestui filtru este prezentată prin următoarea funcție:

```
void CMainFrame::OnToolsMedianFiltering()
{
 CROBOVIEWDoc        *SourceDoc;
 CROBOVIEWDoc        *DestinDoc;
 CNeighbourhoodDimensionsDlg nd_dlg;
  short              m_x = 3;
 short               m_y = 3;
 short               x,y,x_local,y_local;
 short               DestinValue;
 short               count;
 BYTE                matrix[1000];

 nd_dlg.m_x.Format("%d",m_x);
 nd_dlg.m_y.Format("%d",m_x);
 if(nd_dlg.DoModal() == IDOK)
  {
  sscanf(nd_dlg.m_x,"%d",&m_x);
  sscanf(nd_dlg.m_y,"%d",&m_y);
  }
 else
  return;
 SourceDoc = GetActiveSourceDocument();
 DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
    {
    DestinValue = 0;
    count = 0;
    for(x_local=x-(short)(m_x/2);x_local<=x+(short)(m_x/2);x_local++)
      for(y_local=y-(short)(m_y/2);y_local<=y+(short)(m_y/2);y_local++)
        if(x_local>= 0 && x_local<Xmax && y_local>=0 && y_local<REALYMAX)
         {
         matrix[count] = SourceDoc->BitmapMatrix[x_local][y_local][0];
         count++;
         }
    qsort( (void *)matrix, count, sizeof(BYTE), compare);
    DestinValue = matrix[(count/2)+1];
    DestinDoc->BitmapMatrix[x][y][0]=DestinDoc->BitmapMatrix[x][y][1]=
                                        DestinDoc->BitmapMatrix[x][y][2] = (BYTE)DestinValue;
    }
}
```

1.4.3 Metoda mediei imaginilor

Fie o imagine cu zgomot g(x, y) care este formată prin adunarea zgomotului n(x, y) la o imagine clară f(x, y) [95]:

$$g(x,y) = f(x,y) + n(x,y) \tag{1.4.2}$$

unde se presupune că zgomotul este necorelat şi are valoarea medie 0. Obiectivul următoarei proceduri este obţinerea unei imagini filtrate prin adunarea unui set dat de imagini cu zgomot, $g_i(x, y)$, i = 1, 2, ..., K.

Dacă zgomotul satisface condiţiile impuse, este simplu de arătat [122] că dacă o imagine $\bar{g}(x,y)$ este formată prin medierea a K imagini cu zgomot diferite:

$$\bar{g}(x,y) = \frac{1}{K}\sum_{i=1}^{K} g_i(x,y) \tag{1.4.3}$$

atunci rezulta că:

$$E\{\bar{g}(x,y)\} = f(x,y) \tag{1.4.4}$$

şi:

$$\sigma_{\bar{g}}^2(x,y) = \frac{1}{K}\sigma_n^2(x,y) \tag{1.4.5}$$

unde $E\{\bar{g}(x,y)\}$ este valoarea aşteptată a lui $\bar{g}$, iar $\sigma_{\bar{g}}^2(x,y)$ şi $\sigma_n^2(x,y)$ sunt variaţiile lui $\bar{g}$ şi n, raportate la coordonatele (x, y). Deviaţia standard în orice punct din imaginea mediată este dată de

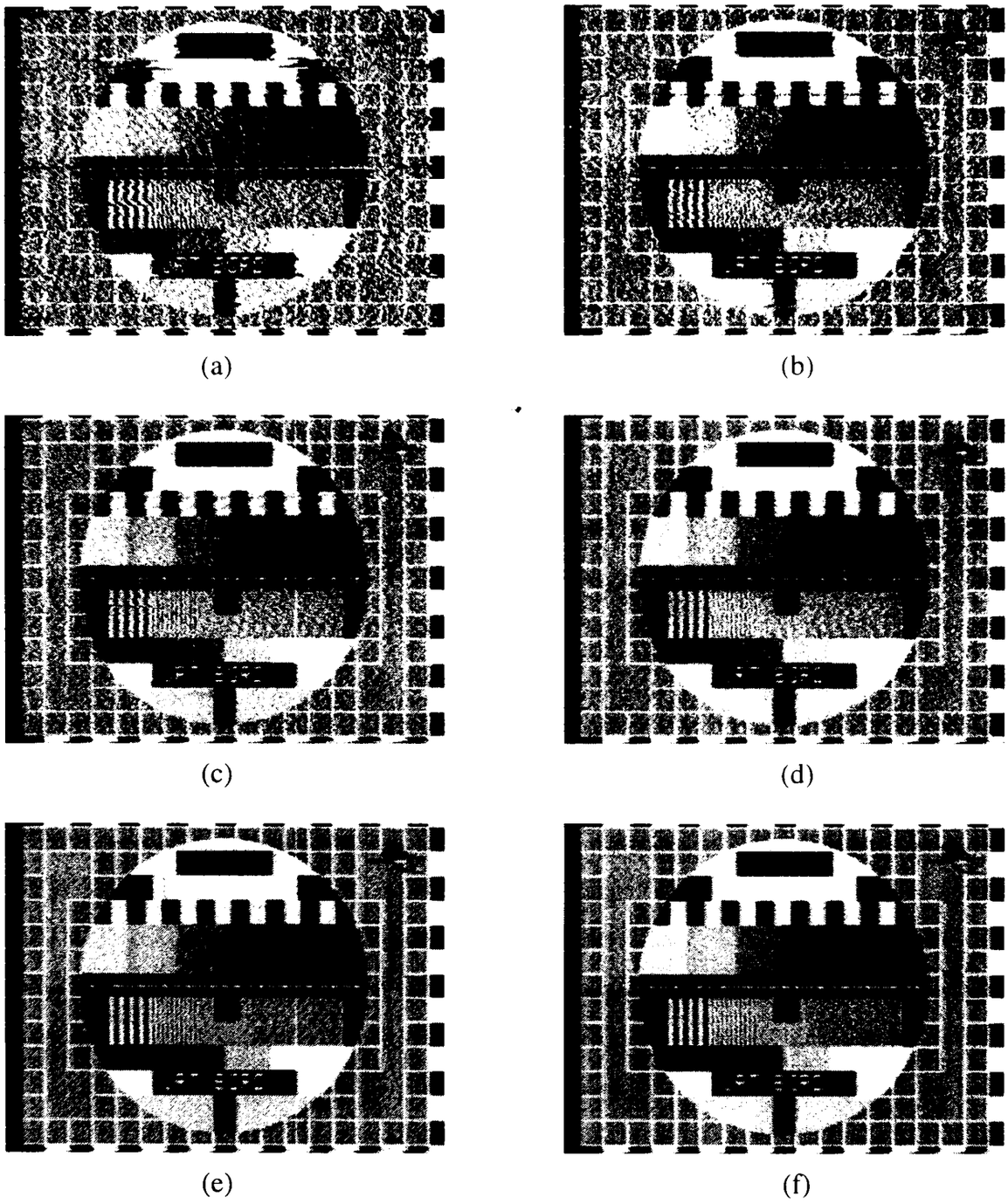
$$\sigma_{\bar{g}}(x,y) = \frac{1}{\sqrt{K}}\sigma_n(x,y) \tag{1.4.6}$$

Ecuaţiile (1.4.5) şi (1.4.6) indică faptul că, pe măsura ce K creşte, posibilitatea de variaţie a valorii pixelului descreşte. Deoarece $E\{\bar{g}(x,y) = f(x,y)\}$, aceasta înseamnă că $\bar{g}(x,y)$ va aproxima din ce în ce mai bine imaginea clară f(x, y) pe măsură ce numărul de imagini cu zgomot folosite în procesul de mediere, creşte.

Este foarte important de remarcat că această tehnică presupune implicit faptul că toate imaginile cu zgomot sunt înregistrate spaţial, doar cu variaţia permisă a intensităţii pixelilor. În termeni din vederea artificială, aceasta înseamnă că orice obiect din spaţiul de lucru trebuie să fie în repaus în raport cu camera video pe timpul procesului de mediere.

Multe sisteme de vedere au capacitatea de achiziţie a unei imagini complete într-un singur cadru (1/30 dintr-o secundă). Astfel că adunarea a 16 imagini va dura aproximativ 1/2 dintr-o secundă, timp în care nu are voie sa aibă loc nici o mişcare.

Figura 1.4.3 prezintă efectul acestui tip de filtrare asupra unei imagini cu zgomot. Figura 1.4.3,a reprezintă imaginea originală, iar figurile 1.4.3,b până la 1.4.3,f reprezintă rezultatul aplicării filtrului folosind media a 4, 8, 16, 32 şi 64 de imagini. Se remarcă faptul că rezultatele sunt acceptabile pentru K=32 de imagini.



(a) (b)

(c) (d)

(e) (f)

Fig. 1.4.3

(a) Imaginea cu zgomot; (b)-(f) rezultatul aplicării medierii unui număr de 4, 8, 16, 32 şi 64 de imagini.

Implementarea acestui filtru este prezentată prin următoarea funcție:

```
void CMainFrame::OnToolsSmoothingImageaveraging()
{
 CimageAveragingDlg  ia_dlg;
 CROBOVIEWDoc    *SourceDoc;
 short                x,y;
 UINT                 i,NumberOfImages;
 UINT                 BitmapMatrix[320][240];

 memset(&BitmapMatrix,0,sizeof(UINT)*320*240);
 ia_dlg.m_Number = 32;

 if(!ghWndCap)
  VideoCapture();
   if(ia_dlg.DoModal() == IDOK)
    NumberOfImages = ia_dlg.m_Number;
   else
    return;
 InitGlobalVariablesFromCapture(&SourceDoc);
//Bucla de captura si prelucrare
 for(i=0;i<NumberOfImages;i++)
  {
  if(CaptureSingleFrame(&SourceDoc))
   continue;
  EmptyMessageQueue();
   for(y=Ymax-1;y>=0;y--)
    for(x=0;x<Xmax;x++)
     BitmapMatrix[x][y] += SourceDoc->BitmapMatrix[x][y][0];
  EmptyMessageQueue();
  }
 for(y=Ymax-1;y>=0;y--)
  for(x=0;x<Xmax;x++)
   SourceDoc->BitmapMatrix[x][y][0]=SourceDoc->BitmapMatrix[x][y][1]=
   SourceDoc->BitmapMatrix[x][y][2]=
   (BYTE)(BitmapMatrix[x][y]/NumberOfImages);
 SourceDoc->UpdateAllViews(NULL);
 return;
}
```

1.4.4 Metoda convoluției şi corelației

Convoluția [137], deşi implică un proces complex de sondare a imaginii, este foarte simplă. Principial convoluția calculează o nouă valoare a intensității pixelilor din imagine, bazându-se pe vecinii pixelului. Fiecare pixel vecin contribuie cu o pondere proprie la calculul noului pixel. Termenul folosit pentru acest proces este ***compensare*** şi se realizează în convoluție prin folosirea ***nucleelor de convoluție***. Fiecare element al unui nucleu de convoluție este numit ***coeficient de convoluție***. În figura 1.4.4 este prezentat un exemplu al unui nucleu de filtrare şi aplicarea sa la convoluția unei imaginii.

Se poate explica convoluția pe un exemplu simplu. În figura 1.4.4, se remarcă faptul că dimensiunea nucleului de filtrare este 3x3, şi că filtrul 3x3 se centrează în

jurul pixelului de calculat. Se mai observă faptul că toți cei opt vecini ai pixelului pot contribui la noua lui valoare. Exemplul prezentat utilizează domeniul filtrelor trece-jos. În filtrarea trece-jos vecinii unui pixel au o mare contribuție la noua lui valoare. Astfel un nucleu de filtrare trece-jos, în acest caz, un nucleu de mediere, se poate prezenta astfel:

$$\text{Filtru} = \begin{matrix} f_{-1,-1} & f_{0,-1} & f_{1,-1} \\ f_{-1,0} & f_{0,0} & f_{1,0} \\ f_{-1,1} & f_{0,0} & f_{1,1} \end{matrix} = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

normalizare 9

Se observă că noua valoare a pixelului va fi bazată în mod egal pe cei opt pixeli înconjurători şi pe vechea valoare a pixelului.

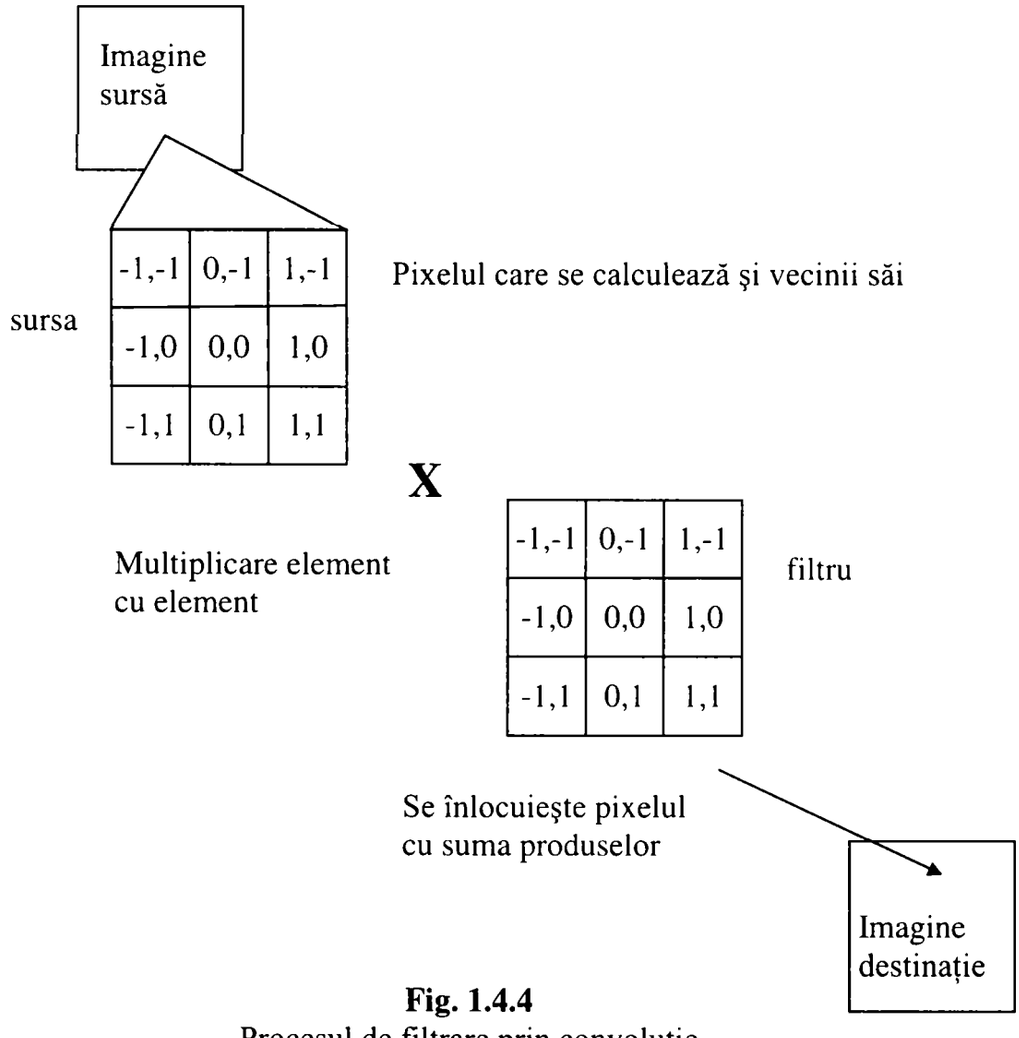


Fig. 1.4.4
Procesul de filtrare prin convoluție

Calculul noului pixel va fi:

$$\text{noua\_valoare\_a\_pixelului} = (f_{-1,-1} * s_{-1,-1}) + (f_{0,-1} * s_{0,-1}) + (f_{1,-1} * s_{1,-1}) + (f_{-1,0} * s_{-1,0}) + (f_{0,0} * s_{0,0}) + (f_{1,0} * s_{1,0}) + (f_{-1,1} * s_{-1,1}) + (f_{0,1} * s_{0,1}) + (f_{1,1} * s_{1,1})$$

unde *s* este o parte din imaginea sursei, şi *f* este elementul unui nucleu de filtrare.

O consecinţă negativă a acestei metode este mărirea luminozităţii imaginii . Din această cauza trebuie normalizată valoarea pixelului. Pentru a realiza acest lucru trebuie să se împartă noua valoare a pixelului cu o valoare normalizată egală cu numărul coeficienţilor nucleului. În acest caz, numărul coeficienţilor nucleului de filtrare este 9, adică:

noua valoare a pixelului = vechea valoare a pixelului / 9

Trebuie remarcat faptul că nucleul de filtrare are coeficienţi întregi, şi este prezentat cu un factor de normalizare. Rezultă următoarea relaţie:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Mai trebuie remarcat, în legătură cu convoluţia, faptul că nucleul de filtrare trebuie să fie impar în ambele dimensiuni (3x3, 5x5) şi că nu trebuie să fie pătratic (3x5, 5x9). Un nucleu mai mare va mări şi flexibilitatea filtrării, permiţând accentuarea anumitor detalii din cadrul imaginii. Se poate privi acum nucleul de filtrare 3x3 ca fiind versiunea unor filtre mult mai mari şi mai selective. Intuitiv se poate deja afirma faptul că dacă se foloseşte un nucleu mai mare într-un filtru trece-jos (de exemplu 5x5 sau 7x7), se va tulbura imaginea mult mai mult decât cu un nucleu de filtrare 3x3.

La filtrele de convoluţie, importante sunt condiţiile de margine, deoarece nu se poate aplica filtrul peste marginile imaginii (doar dacă se creează o margine falsă). De exemplu, dacă pixelul testat se află la locaţia (0,44), filtrul utilizat are nevoie de nouă pixeli între coordonatele (-1,43) şi (1,45). Deoarece valoarea (-1) pe axa x nu există, şi sintetizarea nu se obişnuieşte pentru imagini al căror interes este central, se va putea folosi filtrul peste o regiune mai mică cu un pixel înăuntrul imaginii. Astfel, pentru o imagine de 640 x 480, se va calcula un interior de 638 x 478, şi se vor copia marginile din imaginea originală în cea nouă.

O altă problemă ce apare când se folosesc nucleele de filtrare, este obținerea de valori negative. Deoarece nucleele de filtrare au în mod normal şi coeficienți negativi, este foarte posibil să se obțină intensități negative de pixel în urma calculului. Pentru scopurile urmărite se vor converti toate valorile negative la 0 deoarece în caz contrar ele nu au sens. Alte metode utilizate ar fi folosirea valorii absolute a noii intensități sau adunarea de ofseturi pentru a se asigura obținerea de rezultate pozitive sau nule.

Mai este important de subliniat procedeul de scalare prin însumare, în care majoritatea coeficienților nucleelor de filtrare, conduc la 0 sau 1. În figura 1.4.5 este prezentat un filtru unitate care întoarce imaginea originală înapoi (un filtru trece-tot la care doar pixelii testați contribuie la rezultat). De remarcat că suma lui, ca şi a majorității filtrelor trece-jos sau trece-sus este 1. La filtrul utilizat trece-jos prezentat anterior, dacă se adună coeficienții nucleului se obține 9. Dacă astfel de numere se aplică peste întreaga imagine, aceasta se va albi foarte repede.

Aşa cum s-a menționat soluția este scalarea valorilor înapoi. La filtrul trece-jos din exemplu s-au adunat coeficienții şi s-a împărțit noua valoare cu numărul coeficienților. Acesta este un procedeu tipic, în special pentru filtrele trece-jos, unde se adună contribuțiile mai multor pixeli înconjurători.

$$\frac{1}{9}\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Fig. 1.4.5

Nucleul de filtrare spațial unitate (suma egală cu 1)

Implementarea acestui filtru este prezentată prin următoarea funcție:

```
void CMainFrame::OnToolsSmoothingConvolution()
{
 CConvolutionDlg     c_dlg;
 CROBOVIEWDoc   *SourceDoc;
 CROBOVIEWDoc   *DestinDoc;
 short                x,y;
 short                DestinValue;
 short                NormalizationFactor;

 c_dlg.m_C1=c_dlg.m_C2=c_dlg.m_C3=1;
 c_dlg.m_C4=c_dlg.m_C5=c_dlg.m_C6=1;
 c_dlg.m_C7=c_dlg.m_C8=c_dlg.m_C9=1;
 if(c_dlg.DoModal() != IDOK)
```

```
  return;
 SourceDoc = GetActiveSourceDocument();
 DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
 NormalizationFactor = c_dlg.m_C1+c_dlg.m_C2+c_dlg.m_C3+
                       c_dlg.m_C4+c_dlg.m_C5+c_dlg.m_C6+
                       c_dlg.m_C7+c_dlg.m_C8+c_dlg.m_C9;
 for(x=1;x<Xmax-1;x++)
  for(y=1;y<REALYMAX-1;y++)
    {
    DestinValue  = SourceDoc->BitmapMatrix[x-1][y-1][0]*c_dlg.m_C1;
    DestinValue += SourceDoc->BitmapMatrix[x  ][y-1][0]*c_dlg.m_C2;
    DestinValue += SourceDoc->BitmapMatrix[x+1][y-1][0]*c_dlg.m_C3;
    DestinValue += SourceDoc->BitmapMatrix[x-1][y][0]*c_dlg.m_C4;
    DestinValue += SourceDoc->BitmapMatrix[x  ][y][0]*c_dlg.m_C5;
    DestinValue += SourceDoc->BitmapMatrix[x+1][y][0]*c_dlg.m_C6;
    DestinValue += SourceDoc->BitmapMatrix[x-1][y+1][0]*c_dlg.m_C7;
    DestinValue += SourceDoc->BitmapMatrix[x  ][y+1][0]*c_dlg.m_C8;
    DestinValue += SourceDoc->BitmapMatrix[x+1][y+1][0]*c_dlg.m_C9;
    if(DestinValue < 0)
      DestinValue = 0;
    else
      DestinValue = (BYTE)((double)DestinValue/NormalizationFactor);
   DestinDoc->BitmapMatrix[x][y][0]=DestinDoc->BitmapMatrix[x][y][1]=
                                    DestinDoc->BitmapMatrix[x][y][2] = (BYTE)DestinValue;
    }
}
```

1.4.5 Metoda filtrelor spațiale trece-jos

Un filtru trece-jos lasă să treacă doar componentele de frecvență joasă dintr-o imagine atenuându-le pe cele înalte [24]. Aceste filtre sunt folosite de obicei pentru eliminarea zgomotului dintr-o imagine, în cazul în care unde zgomotul are lățimea echivalentă a 1 sau maxim 2 pixeli. O diagramă a unui filtru trece-jos este prezentată în figura 1.4.6.

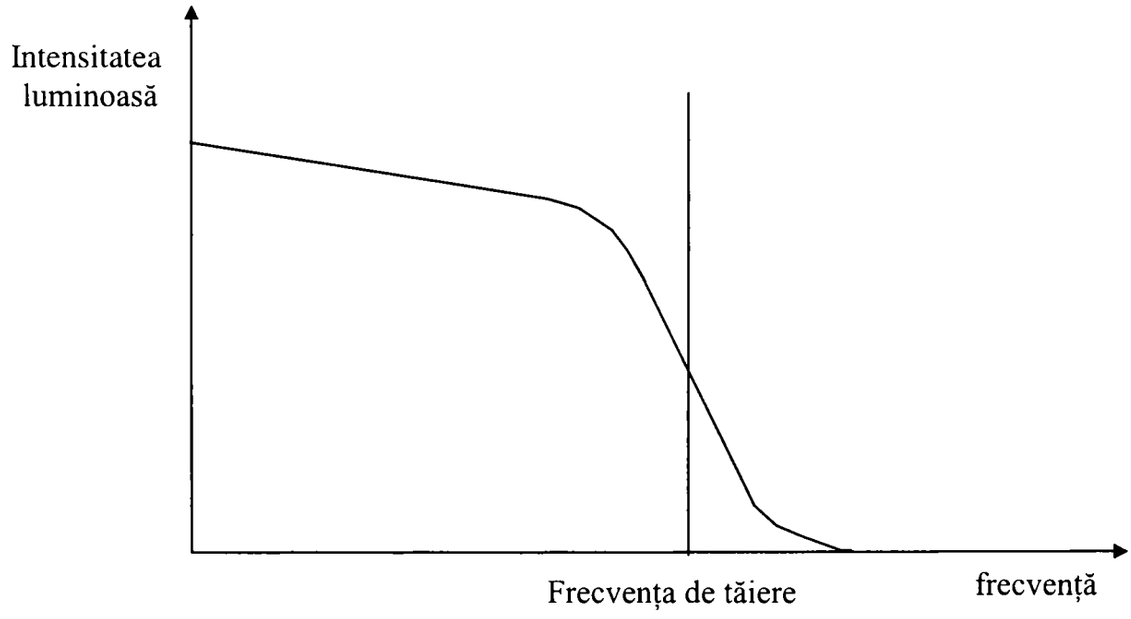


Fig. 1.4.6
Diagrama unui filtru trece-jos

Frecvența de tăiere (frecvență la care filtrul oprește semnalul sau informația de imagine) din figura 1.4.6 este determinată de dimensiunea nucleului de filtrare și de coeficienții nucleului. Coeficienții nucleului pentru un filtru trece-jos dau întotdeauna o sumă mai mare decât 1. Această sumă este ulterior normalizată, astfel încât coeficienții reprezintă un procent din contribuția acelui pixel la noul pixel. În figura 1.4.7 se prezintă un tabel cu cele mai comune nuclee de filtrare trece-jos.

1 1 1	1 1 1	1 1 1	1 1 1	1 2 1
1 1 1	1 2 1	1 4 1	1 12 1	2 4 2
1 1 1	1 1 1	1 1 1	1 1 1	1 2 1
9	10	12	20	16
Media	TJ1	TJ2	TJ3	Gauss

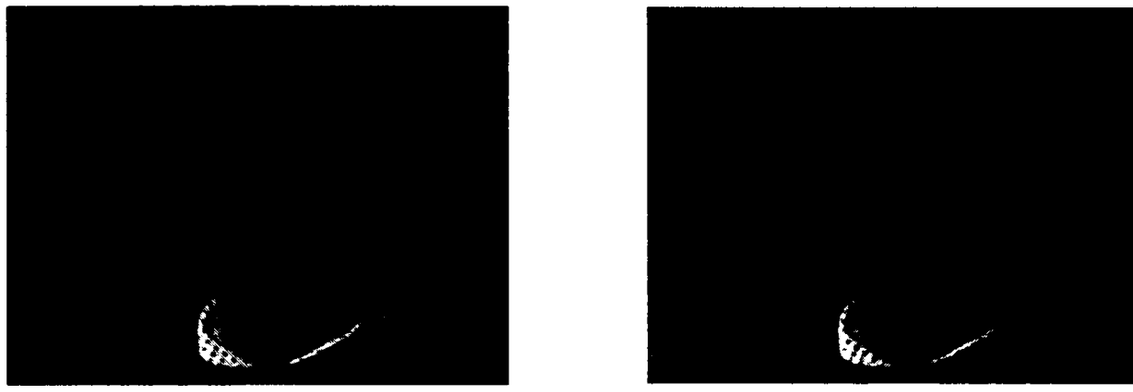
Fig. 1.4.7

Filtre spațiale trece-jos (suma normalizată = 1)

Funcțional, aceste filtre se folosesc în zonele imaginii în care există variații mici ale culorii. În timp ce filtrul parcurge o astfel de zonă, compensarea făcută de pixelii vecini conduce spre o nouă valoare a pixelului, foarte apropiată de cea veche. Aceasta se întâmplă deoarece pixelul și vecinii lui sunt foarte apropiați ca valoare.

Pentru acest motiv atenția se va fixa asupra contribuției procentuale a fiecărui pixel la noua valoare de pixel, însemnând că fiecare pixel contribuie cu ponderea egală cu 1/(numărul coeficienților). Filtrul utilizat va menține componentele de frecvență joasă din imagine. Apoi, se mută filtrul peste o parte a imaginii cu mai multe schimbări. În acest caz se vede cum funcționează în realitate filtrul. Zonele cu o mare schimbare de intensitate de la un pixel la altul se mediază cu pixelii vecini, astfel că se obține un conținut redus de înaltă frecvență al imaginii. Rezultatul este o tulburare și o finisare a imaginii, eliminându-se zgomotele.

Doi pași importanți trebuie urmați pentru a face ca utilizarea filtrului trece-jos (care tulbură imaginea) să îi mărească, de fapt, contrastul. Se va vedea în cele ce urmează că făcând diferența dintre imaginea originală și cea filtrată, se mărește contrastul imaginii.



(a) (b)

Fig. 1.4.8
(a) Imagine originală; (b) Imaginea filtrată cu filtrul TJ1

1.4.6 Metoda filtrelor spațiale trece-sus

Filtrele trece-sus se folosesc pentru a amplifica detaliile de înaltă frecvență dintr-o imagine, în timp ce integritatea frecvențelor joase rămâne nemodificată [24]. Cu filtrele trece-sus părțile de înaltă frecvență din imagine vor deveni mai luminoase în timp ce cele de joasă frecvență vor deveni mai întunecate. Contrastul imaginii va creşte de obicei, cu un efect negativ asupra amplificării zgomotului. În figura 1.4.9 este prezentată o diagramă a unui filtru trece-sus, iar în figura 1.4.10 sunt prezentate mai multe nuclee de filtre trece-sus.

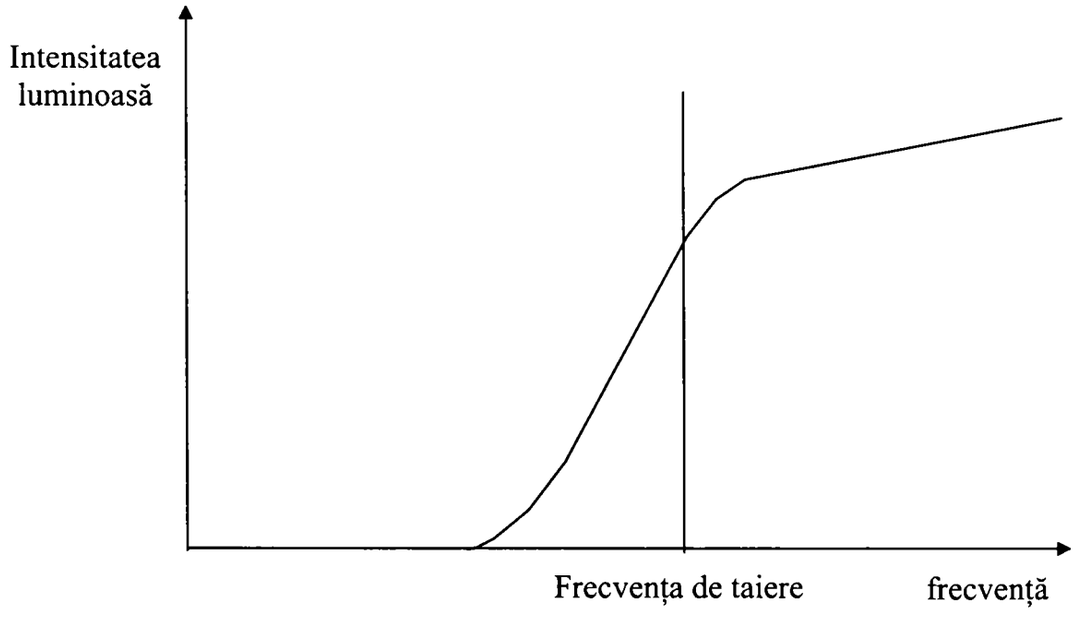


Fig. 1.4.9
Diagrama unui filtru trece-sus

-1 -1 -1	0 -1 0	1 -2 1	0 -1 0
-1 9 -1	-1 5 -1	-2 5 -2	-1 20 -1
-1 -1 -1	0 -1 0	1 -2 1	0 -1 0
Eliminarea valorii	HP1	HP2	HP3

Fig. 1.4.10

Filtre spațiale trece-sus (suma normalizată = 1)

Ca şi în cazul filtrelor trece-jos, înțelegerea conceptului de filtrare cu nucleu stă la baza modificărilor necesare pentru acest scop specific.

Filtrele trece-sus au în general coeficienții centrali din nuclee foarte mari. În timp ce aceste centre se deplasează peste o imagine ce prezintă variații de la un pixel la altul, noua valoare a pixelului este multiplicată de multe ori. Coeficienții mici negativi sunt folosiți pentru a nega efectul factorilor centrali de compensare mari. Aceste zone cu variații mari ale intensității pixelilor, vor fi accentuate în timp ce celelalte vor rămâne nemodificate.

(a)

(b)

Fig. 1.4.11

(a) Imagine originală; (b) Imaginea filtrată cu filtrul TS1

1.4.7 Metoda convoluției încrucişate folosind un filtru Gauss

Până acum, s-a tratat convoluția ca un proces constând dintr-o singură operație. Adică, se realiza convoluția cu două mulțimi de numere: un grup de pixeli din imagine şi un nucleu de filtrare. Se poate însă folosi convoluția imaginii rând cu rând cu un filtru unidimensional, după care se va aplica filtrul, coloană cu coloană, pe noua imagine. Acest procedeu de aplicare a două convoluții unidimensionale succesive este numită ***convoluție încrucişată***.

Se va prezenta un exemplu simplu de convoluție încrucișată folosind curba lui Gauss. Astfel, se presupune că o suprafață este creată din această formă de clopot cu ecuațiile:

$$z = e^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{1.4.7}$$

unde σ este abaterea medie pătratică.

Se remarcă faptul că este nevoie de un nucleu de filtrare foarte mare pentru a reprezenta aceasta ca un filtru. Deoarece timpul de calcul necesar pentru multiplicarea unei reprezentări rezonabile a nucleului de filtrare cu imaginea, este proporțional cu pătratul dimensiunii matricii, se va aplica metoda convoluției încrucișate.

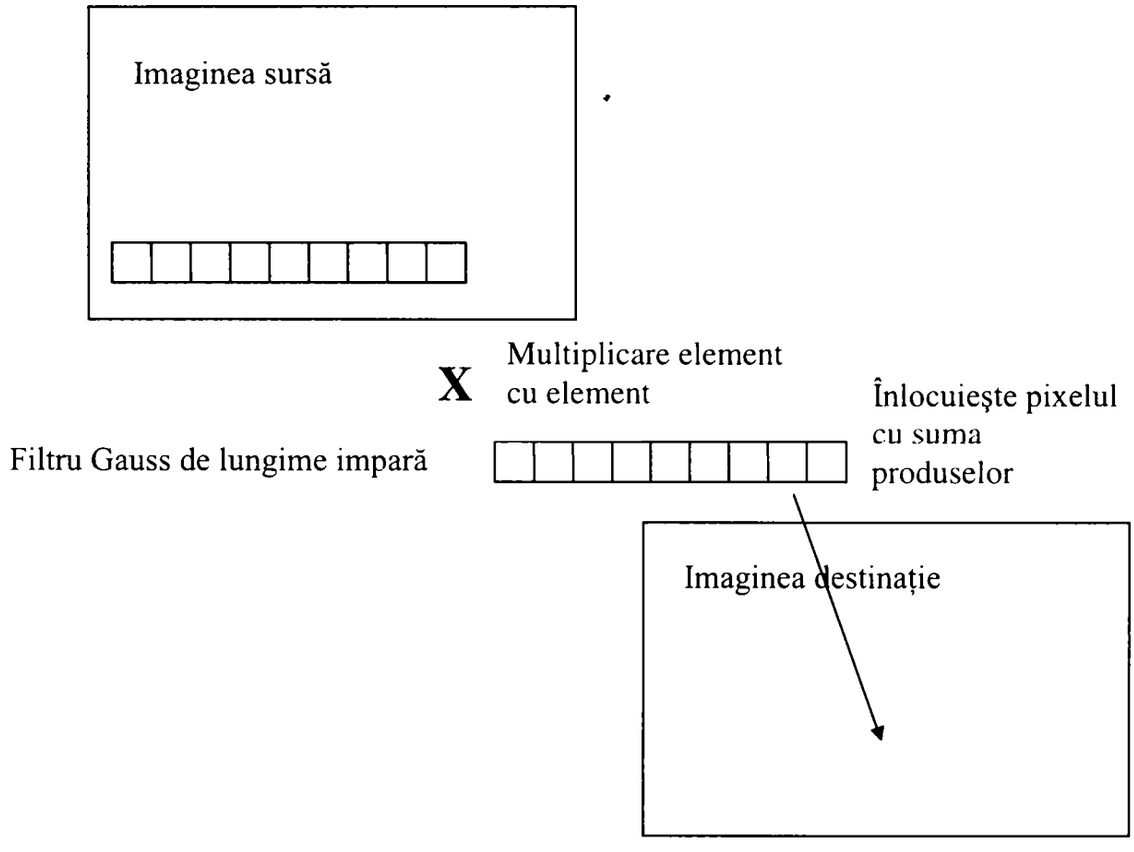


Fig. 1.4.12
Convoluția încrucișată

Din cauză că forma profilului filtrului este aceeaşi şi pe axa x şi pe axa y, se poate aplica acelaşi filtru de-a lungul rândurilor şi apoi de-a lungul coloanelor, şi se vor obține rezultatele dorite.

Un grafic al folosirii convoluției încrucișate este prezentat în figura 1.4.12.

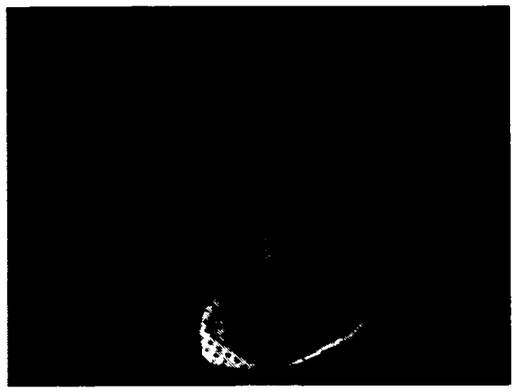
Aşa cum este de aşteptat, acest filtru este folosit pentru finisarea şi tulburarea imaginii. Cu cât σ este mai mare în ecuaţia precedentă, cu atât imaginea va fi mai tulbure. Aceasta apare deoarece un pixel aflat la distanţă de pixelul testat va deveni mai important (din cauză că valoarea nucleelor de filtrare vor fi mai mari, aceşti pixeli îndepărtaţi vor avea o pondere mai mare). Se va folosi un astfel de filtru când se va dori mărirea contrastului unei imagini scăzând imaginea rezultat din imaginea originală.

1.4.8 Metoda filtrelor minim şi maxim

În mod similar cu filtrul median, se vor sorta valorile pixelilor de la cel mai mare la cel mai mic [137]. În schimb pentru calculul noii valori, se vor folosi valorile extreme. Pentru filtrul minim se va folosi valoarea cea mai mică ca şi noua valoare filtrată. Pentru filtrul maxim se va folosi valoarea cea mai mare pentru noua valoare filtrată. Filtrul minim se mai numeşte şi ***filtru de compresie*** sau ***filtru de eroziune***, deoarece multiple aplicări ale acestui filtru vor cauza o scădere a intensităţii conturului obiectelor din cadrul imaginii. Astfel, acestea par că se erodează. Pe de altă parte, filtrul maxim se mai numeşte ***filtru de decompresie*** sau de ***expansiune***, deoarece aplicări succesive ale acestui filtru vor cauza o creştere a intensităţii contururilor obiectelor din cadrul imaginii. Astfel, dimensiunile obiectelor vor părea mărite.

O tehnică simplă care foloseşte atât filtrul minimal cât şi pe cel maximal se numeşte ***închidere***. Într-o închidere se aplică în primul rând filtrul minimal pentru eliminarea zgomotului. Acest filtru afectează intensitatea informaţiei utile din imagine. Pentru a contracara acest fenomen se aplică după aceea filtrul maximal pentru a readuce intensitatea la locul ei. Rezultatul este eliminarea zgomotului.

O altă tehnică simplă ce foloseşte ambele filtre se numeşte ***deschidere***. Într-o deschidere se aplică mai întâi filtrul maximal pentru a mări dimensiunile şi pentru a vedea mai multe detalii. Acest filtru va afecta intensitatea unor părţi din imaginea utilă. Pentru a contracara acest efect se va aplica filtrul minimal pentru a micşora intensitatea în zonele utile ale imaginii. Rezultatul este de obicei o expansiune a imaginii.



(a)

(b)

(c)

Fig. 1.4.13
(a) Imaginea originală
(b) Aplicarea filtrului minim
(c) Aplicarea filtrului maxim

Implementarea acestui filtru este identică cu cea a filtrului median, cu excepția selecției finale care se va realiza în acest caz în punctul de intensitate minimă sau maximă.

1.4.9 Metoda filtrării imaginilor binare

Imaginile binare rezultă în urma folosirii luminii din spatele obiectului, a unei lumini structurate, în urma unui proces de detecție de contur sau de folosire a unui prag, etc. Se foloseşte convenția că punctele întunecate sunt reprezentate prin 1 iar cele luminoase prin 0. Astfel, deoarece imaginile binare sunt bazate doar pe două valori, zgomotul va produce diferite efecte ca : contururi neregulate, mici găuri, colțuri lipsă şi puncte izolate.

Ideea care stă la baza acestei metode este specificarea unei funcții booleene evaluată pe o vecinătate centrată pe un pixel *p*, şi atribuirea de 1 sau 0 lui *p*, în funcție de aranjamentul spațial şi de valorile binare ale vecinilor lui. Datorită limitării timpului de procesare în aplicațiile de vedere industrială, analiza este de obicei

limitată la cei 8 vecini ai lui *p*, aceasta conducând la o mască 3x3 ca şi cea din figura 1.4.14.

a	b	c
d	p	e
f	g	h

Fig. 1.4.14
Vecinii lui *p* folosiți pentru filtrarea imaginilor binare

Procedeul de filtrare cuprinde următorii paşi [122], [129], [144], [145]:

1. Umple găurile de dimensiunea unui pixel într-o zonă normal întunecată
2. Umple micile crestături dintr-un segment de contur drept
3. Elimină punctele întunecate singulare
4. Elimină micile umflături de-a lungul segmentelor de contur drepte
5. Înlocuieşte punctele din colțurile lipsă

Relativ la figura 1.4.14, primii doi paşi în procesul de filtrare se realizează folosind următoarea expresie booleană [122]:

$$B_1 = p + b \bullet g \bullet (d + e) + d \bullet e \bullet (b + g) \qquad (1.4.8)$$

unde "$\bullet$" şi "+" se referă la operațiile logice ŞI respectiv SAU.

Folosind convențiile stabilite anterior, un pixel întunecat conținut în suprafața de mascare are valoarea 1 iar un pixel luminos are valoarea logică 0. Astfel dacă B_1=1, se va atribui valoarea 1 lui *p*, în caz contrar acest pixel va lua valoarea 0. Relația (1.4.8) este aplicată tuturor pixelilor simultan, în sensul că noua valoare a fiecărei locații de pixel este determinată înainte ca oricare dintre pixeli să fie modificați.

Paşii (3) şi (4) din procesul de filtrare sunt realizaţi evaluând următoarea expresie booleană

$$B_2 = p \bullet \left[(a + b + d) \bullet (e + g + h) + (b + c + e) \bullet (d + f + g) \right] \qquad (1.4.9)$$

simultan pentru toţi pixelii. Ca mai sus, *p* va lua valoarea 1 dacă B_2=1 şi 0 în caz contrar.

Punctele lipsă din colţurile dreapta sus sunt umplute utilizând expresia:

$$B_3 = \bar{p} \bullet (d \bullet f \bullet g) \bullet \overline{(a + b + c + e + h)} + p \qquad (1.4.10)$$

unde bara de deasupra are semnificaţia complementului logic al mulţimii respective.

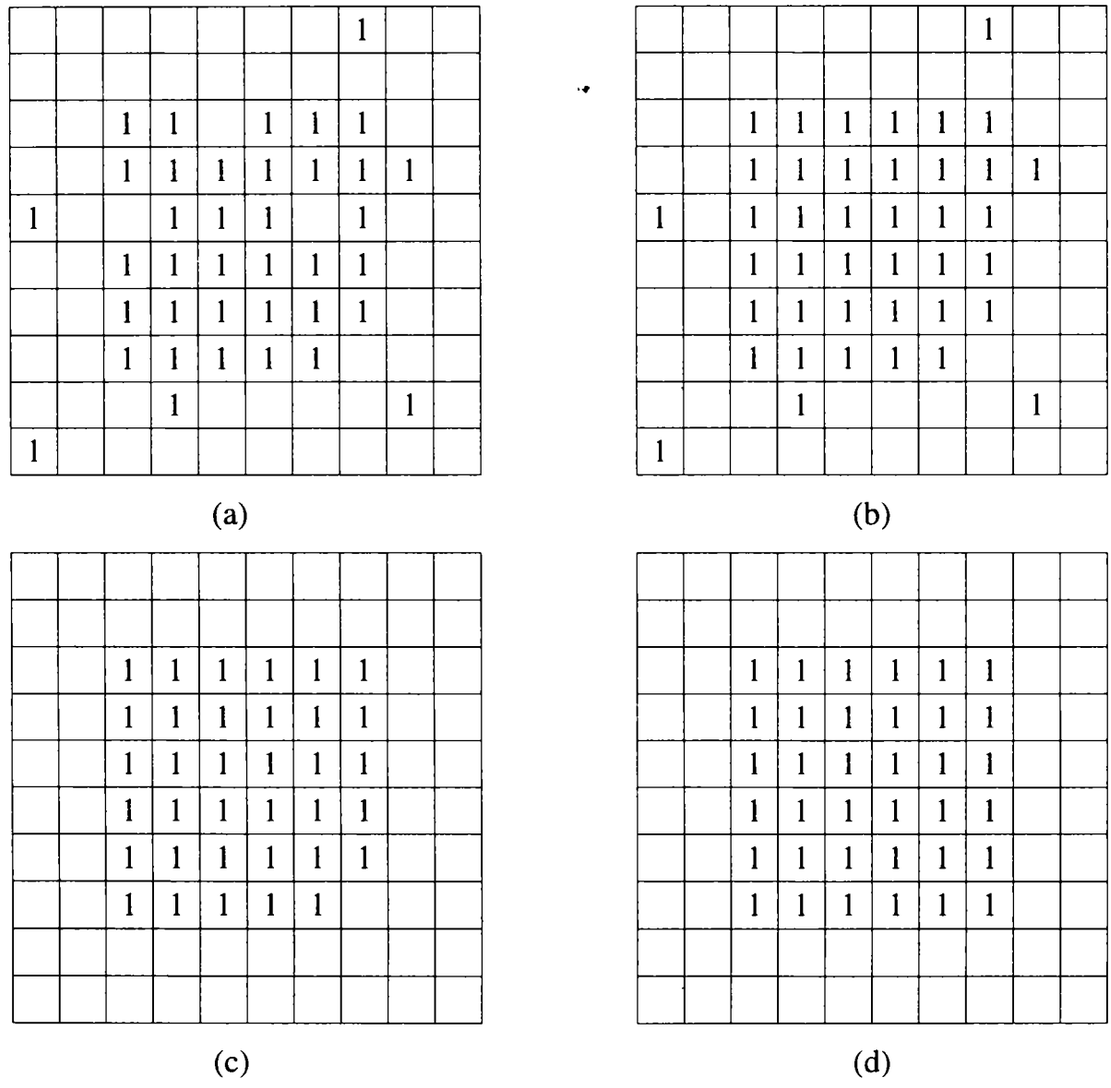


Fig. 1.4.15
(a) Imaginea originală ; (b) Rezultatul aplicării lui B_1; (c) Rezultatul aplicării lui B_2; (d) Rezultatul final după aplicarea lui B_3 până la B_6.

Similar, colţurile lipsă din dreapta jos, stânga sus şi stânga jos sunt umplute utilizând expresiile :

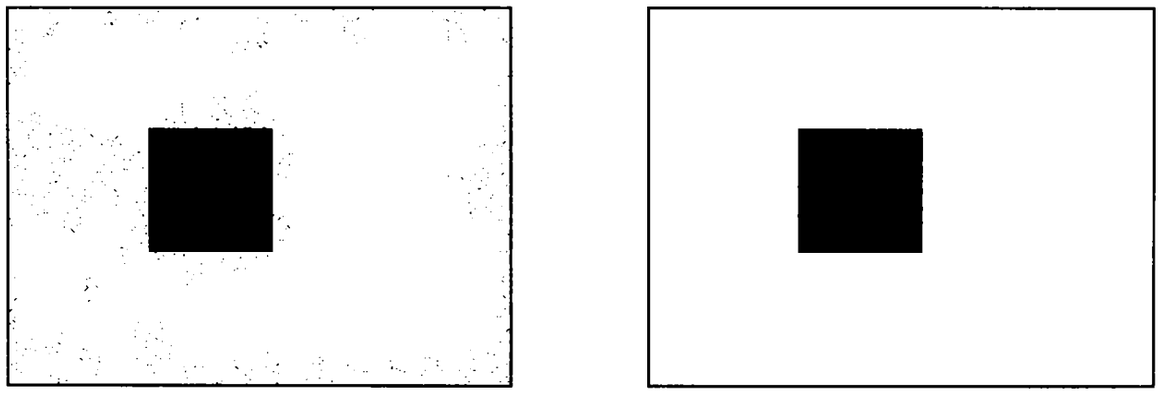
$$B_4 = \bar{p} \bullet (a \bullet b \bullet d) \bullet \overline{(c + e + f + g + h)} + p \qquad (1.4.11)$$

$$B_5 = \bar{p} \bullet (e \bullet g \bullet h) \bullet \overline{(a + b + c + d + f)} + p \qquad (1.4.12)$$

$$B_6 = \bar{p} \bullet (b \bullet c \bullet e) \bullet \overline{(a + d + f + g + h)} + p \qquad (1.4.13)$$

Relaţiile 1.4.10-1.4.13 implementează pasul (5) din procedura de filtrare.

Conceptele prezentate sunt ilustrate în figura 1.4.15. Figura 1.4.15,a reprezintă o imagine binară zgomotoasă, iar figura 1.4.15,b reprezintă rezultatul aplicării expresiei B_1. Se remarcă faptul că lipsurile din contur şi găurile din regiunea întunecată sunt umplute. Figura 1.4.15,c reprezintă rezultatul aplicării lui B_2 imaginii 1.4.15,b. Aşa cum era de aşteptat umflăturile de-a lungul conturului şi punctele izolate sunt îndepărtate. În final figura 1.4.15,d reprezintă rezultatul aplicării expresiilor B_3-B_6 asupra imaginii din 1.4.15,c. Doar B_4 are efect în acest caz particular.



(a) (b)

Fig. 1.4.16
(a) Imagine binară cu zgomot
(b) Imagine filtrată binar

Implementarea acestui filtru este prezentată prin următoarea funcţie:

```
void CMainFrame::SmoothingBinaryImages(BOOL NewDocument)
{
 CROBOVIEWDoc        *SourceDoc;
 CROBOVIEWDoc        *DestinDoc;
 BinaryErrorDlg       be_dlg;
 BOOL                 BitmapMatrix1[320][240];
 BOOL                 BitmapMatrix2[320][240];
 short                x,y;
 SourceDoc = GetActiveSourceDocument();
 if(SourceDoc->BitmapColorType  != BINARY)
```

```
    {
    be_dlg.DoModal();
    return;
    }
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
      if(SourceDoc->BitmapMatrix[x][y][0] == 0)
        BitmapMatrix1[x][y] = BitmapMatrix1[x][y] = BitmapMatrix1[x][y] = 1;
      else
        BitmapMatrix1[x][y] = BitmapMatrix1[x][y] = BitmapMatrix1[x][y] = 0;
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
/*B1*/  BitmapMatrix2[x][y]=BitmapMatrix1[x][y] || (BitmapMatrix1[x][y-1] &&
          BitmapMatrix1[x][y+1] && (BitmapMatrix1[x-1][y] || BitmapMatrix1[x+1][y])) ||
          (BitmapMatrix1[x-1][y] && BitmapMatrix1[x+1][y] && (BitmapMatrix1[x][y-1] ||
          BitmapMatrix1[x][y+1]));
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
/*B2*/  BitmapMatrix1[x][y] = BitmapMatrix2[x][y] && (((BitmapMatrix2[x-1][y-1] ||
          BitmapMatrix2[x][y-1] || BitmapMatrix2[x-1][y]) && (BitmapMatrix2[x+1][y] ||
          BitmapMatrix2[x][y+1] || BitmapMatrix2[x+1][y+1])) || ((BitmapMatrix2[x][y-1] ||
          BitmapMatrix2[x+1][y-1] || BitmapMatrix2[x+1][y]) && (BitmapMatrix2[x-1][y] ||
          BitmapMatrix2[x-1][y+1] || BitmapMatrix2[x][y+1])));
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
/*B3*/  BitmapMatrix2[x][y] = !BitmapMatrix1[x][y] && BitmapMatrix1[x-1][y] &&
          BitmapMatrix1[x-1][y+1] && BitmapMatrix1[x][y+1] && (!(BitmapMatrix1[x-1][y-1] ||
          BitmapMatrix1[x][y-1] || BitmapMatrix1[x+1][y-1] || BitmapMatrix1[x+1][y] ||
          BitmapMatrix1[x+1][y+1])) || BitmapMatrix1[x][y];
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
/*B4*/  BitmapMatrix1[x][y] = !BitmapMatrix2[x][y] && BitmapMatrix2[x-1][y-1] &&
          BitmapMatrix2[x][y-1] && BitmapMatrix2[x-1][y] && (!(BitmapMatrix2[x+1][y-1] ||
          BitmapMatrix2[x+1][y] || BitmapMatrix2[x-1][y+1] || BitmapMatrix2[x][y+1] ||
          BitmapMatrix2[x+1][y+1])) || BitmapMatrix2[x][y];
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
/*B5*/   BitmapMatrix2[x][y] = !BitmapMatrix1[x][y] && BitmapMatrix1[x+1][y] &&
           BitmapMatrix1[x][y+1] && BitmapMatrix1[x+1][y+1] && (!(BitmapMatrix1[x-1][y-1] ||
           BitmapMatrix1[x][y-1] || BitmapMatrix1[x+1][y-1] || BitmapMatrix1[x-1][y] ||
           BitmapMatrix1[x-1][y+1])) || BitmapMatrix1[x][y];
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
/*B6*/  BitmapMatrix1[x][y] = !BitmapMatrix2[x][y] && BitmapMatrix2[x][y-1] &&
          BitmapMatrix2[x+1][y-1] && BitmapMatrix2[x+1][y] && (!(BitmapMatrix2[x-1][y-1] ||
          BitmapMatrix2[x-1][y] || BitmapMatrix2[x-1][y+1] || BitmapMatrix2[x][y+1] ||
          BitmapMatrix2[x+1][y+1])) || BitmapMatrix2[x][y];
  if(NewDocument)
    DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
  else
    DestinDoc = SourceDoc;
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
      if(BitmapMatrix1[x][y] == 0)
        DestinDoc->BitmapMatrix[x][y][0] = DestinDoc->BitmapMatrix[x][y][1] =
                                           DestinDoc->BitmapMatrix[x][y][2] = 255;
      else
        DestinDoc->BitmapMatrix[x][y][0] = DestinDoc->BitmapMatrix[x][y][1] =
                                           DestinDoc->BitmapMatrix[x][y][2] = 0;
}
```

1.5 Îmbunătățirea imaginii

Una din principalele dificultăți apărute în multe din aplicațiile vederii artificiale este aceea de a se putea adapta automat la schimbările de iluminare. Trebuie subliniat faptul că îmbunătățirea este o parte principală din cadrul procesării digitale a imaginii, şi că discuția asupra acesteia se va limita doar la tehnicile aplicabile în realitate în vederea artificială, adică la viteză de procesare mare şi necesități hardware minime. Există două abordări posibile de îmbunătățire a imaginii [98],[108]:

- Globală, care lucrează cu întreagă imagine
- Locală, care lucrează pe zone mici de imagine, ce au caracteristici diferite de cele ale restului imaginii.

În ambele variante se utilizează două metode principale de îmbunătățire:

- Egalizarea histogramei, care încearcă să realizeze răspândirea uniformă a valorilor de intensitate pe întreg spectrul disponibil
- Specificarea histogramei, care încearcă să realizeze scoaterea în evidență a anumitor nivele de intensitate ce se cunosc „a priori".

1.5.1 Egalizarea histogramei

Se consideră că *r* reprezintă intensitatea pixelilor dintr-o imagine ce trebuie îmbunătățită. Se presupune că inițial *r* este normalizată şi continuă în intervalul $0 \le r \le 1$. Cazul discret se va lua în considerare ulterior.

Pentru orice *r* din intervalul [0,1], atenția se va focaliza asupra unor transformări de tipul :

$$s = T(r) \tag{1.5.1}$$

care determină o valoare a intensității *s* pentru fiecare valoare de pixel *r* din imaginea originală. Se presupune că funcția de transformare T satisface următoarele condiții:

1. T(r) are o valoare unică şi creşte monoton pe intervalul $0 \le T(r) \le 1$.
2. $0 \le T(r) \le 1$ pentru $0 \le r \le 1$.

Condiția 1 menține ordinea de la negru la alb pe scara de intensitate, iar cea de-a doua condiție garantează o mapare care este în concordanță cu domeniul [0,1] permis

pentru valorile de pixel. O funcție de transformare ce satisface aceste condiții este cea din figura 1.5.1.

Funcția de transformare inversă de la *s* la *r* este data de relația:

$$r = T^{-1}(s) \tag{1.5.2}$$

unde se presupune ca $T^{-1}(s)$ satisface de asemenea cele două condiții menționate.

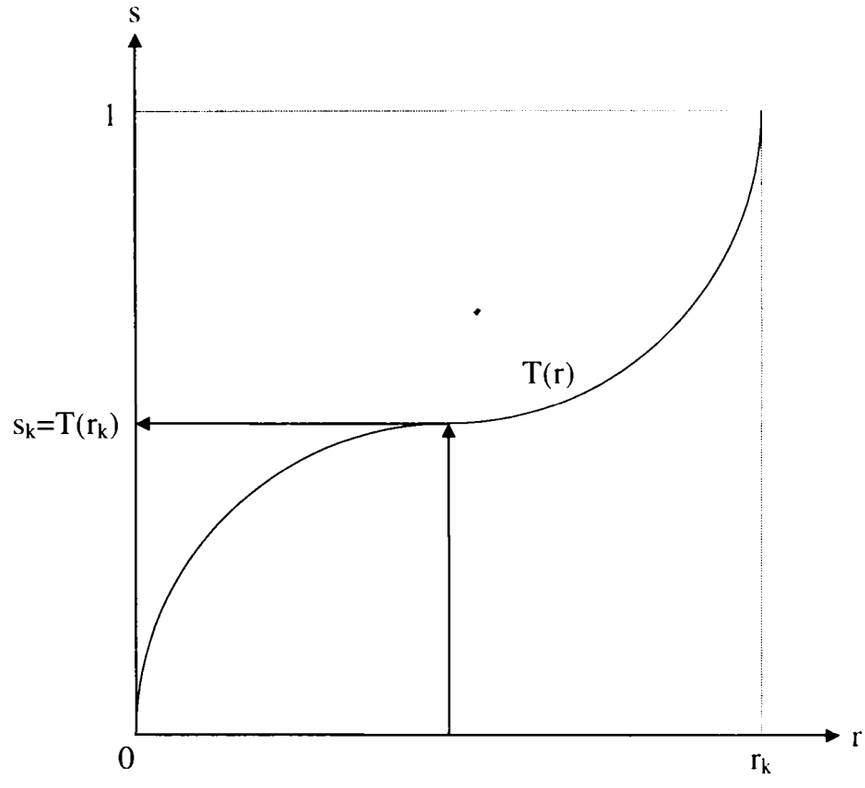


Fig. 1.5.1
Funcția de transformare a intensității

Variabilele de intensitate *r* şi *s* sunt cantități aleatoare în intervalul [0,1] şi, din această cauză, sunt caracterizate de funcțiile de densitatea de probabilitate (FDP): $p_r(r)$ şi $p_s(s)$.

Prin prisma acestor funcții rezultă multe informații despre intensitatea unei imagini. De exemplu, o imagine cu pixelii având o funcție de densitate a probabilității ca cea din figura 1.5.2,a va avea o caracteristică întunecată, deoarece majoritatea valorilor de pixeli sunt concentrate spre partea întunecată a scării intensității. Pe de altă parte, o imagine cu un FDP ca cel din figura 1.5.2,b va avea predominant o luminozitate mare.

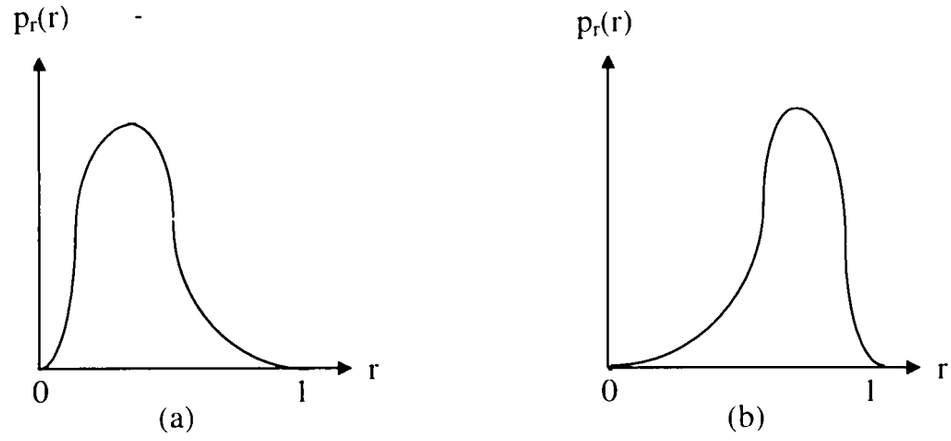


Fig. 1.5.2
(a) FDP pentru o imagine întunecată. (b) FDP pentru o imagine luminoasă

Din teoria probabilității [135], [138], rezultă că dacă $p_r(r)$ şi T(r) sunt cunoscuți, şi $T^{-1}(r)$ satisface condiția 1, atunci FDP a transformatei intensității este dată de [3] :

$$p_s(s) = \left[p_r(r) \frac{dr}{ds} \right]_{r=T^{-1}(s)} \tag{1.5.3}$$

Se alege o funcție de transformare specifică data de relația:

$$s = T(r) = \int_0^r p_r(w)dw \qquad 0 \le r \le 1 \tag{1.5.4}$$

unde *w* este o variabilă oarecare de integrare. Partea din dreapta a ecuației se numeşte ***funcția de distribuție cumulativă*** a lui $p_r(r)$, care se ştie că satisface cele două condiții menționate. Derivata lui *s* în raport cu *r* pentru aceasta funcție de transformare specială este:

$$\frac{ds}{dr} = p_r(r) \tag{1.5.5}$$

Înlocuirea derivatei (1.5.5) în ecuația (1.5.3) conduce la:

$$p_s(s) = \left[p_r(r) \frac{1}{p_r(r)} \right]_{r=T^{-1}(s)} = [1]_{r=T^{-1}(s)} = 1 \quad 0 \le s \le 1 \tag{1.5.6}$$

ceea ce indică o densitate uniformă pe intervalul de definiție al variabilei transformate *s*. De remarcat este faptul că rezultatul este independent de inversa funcției de

transformare. Această proprietate este importantă deoarece de obicei este foarte dificilă determinarea analitică a lui $T^{-1}(s)$. De asemenea este de remarcat faptul că folosind funcția de transformare dată de ecuația (1.5.4) vor rezulta intensități transformate care au întotdeauna un FDP liniar, independent de forma lui $p_r(r)$, această proprietate fiind ideală pentru îmbunătățirea imaginii în mod automat. Efectul net al acestei transformări este echilibrarea distribuției de intensități. Acest proces poate avea un efect dramatic asupra aspectului unei imagini.

În scopul utilizării pentru procesarea digitală, conceptele dezvoltate mai sus trebuie să fie formulate într-o formă discretă. Pentru intensități ce iau valori discrete se va lucra cu probabilități date de relația:

$$p_r(r_k) = \frac{n_k}{n} \qquad 0 \le r_k \le 1 \qquad (1.5.7)$$

$$k=0,1,2,...,L-1$$

unde :

- L este numărul de nivele discrete ale intensității
- $p_r(r_k)$ este o estimare a probabilității intensității r_k
- n_k este numărul de apariții a acestei intensități în imagine
- n este numărul total de pixeli din imagine

Un grafic a lui $p_r(r_k)$ în funcție de r_k reprezintă o ***histogramă***, iar tehnica folosită pentru obținerea unei histograme uniforme este cunoscută ca ***egalizarea histogramei*** sau ***liniarizarea histogramei***.

Forma discretă a ecuației (1.5.3) este dată de

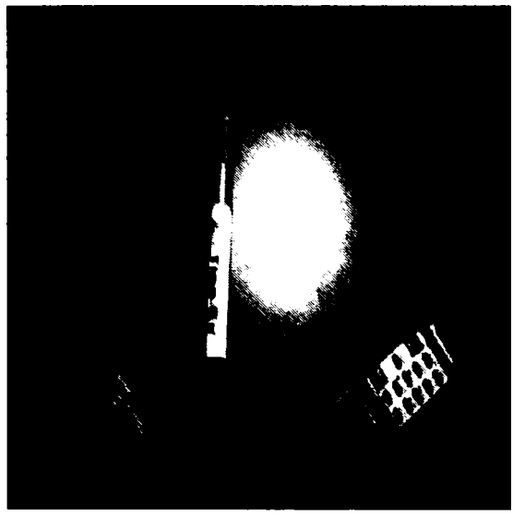
$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} \qquad = \sum_{j=0}^{k} p_r(r_j) \qquad (1.5.8)$$

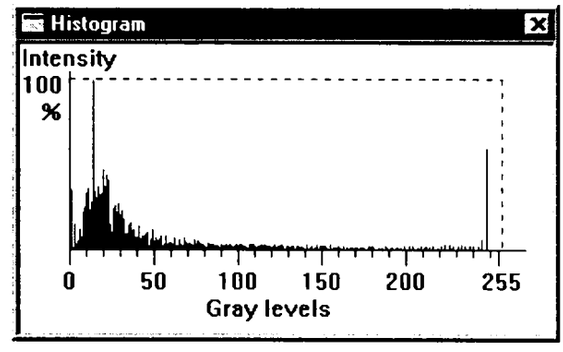
pentru $0 \le r_k \le 1$ şi k=0,1,2,...,L-1. Din această ecuație rezultă că pentru a obține valoarea s_k corespunzătoare lui r_k, vor trebui doar adunate componentele histogramei de la 0 la r_k.

Transformata inversă discretă este dată de relația

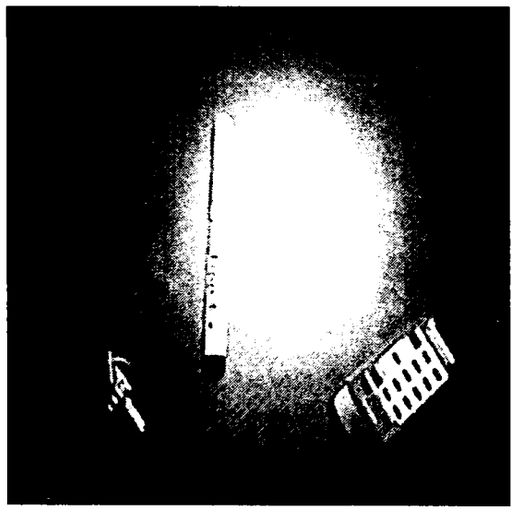
$$r_k = T^{-1}(s_k) \qquad 0 \le s_k \le 1 \qquad (1.5.9)$$

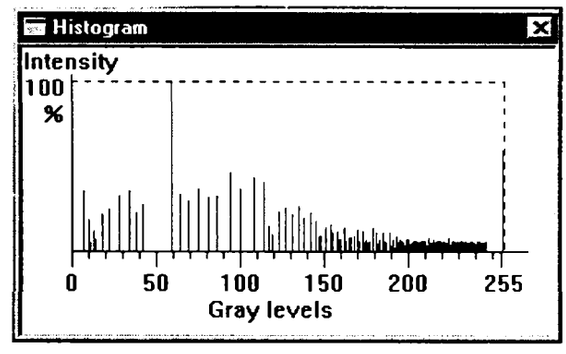
unde atât $T(r_k)$ cât şi $T^{-1}(s_k)$ se presupune că satisfac condiţiile 1 şi 2 valabile pentru funcţia definită prin relaţia (1.5.1). Deşi $T^{-1}(s_k)$ nu se foloseşte în egalizarea histogramei, ea joacă un rol important în specificarea histogramei.





(a) (b)





(c) (d)

Fig. 1.5.3

(a) Imagine dominată de un puternic spot luminos
(b) Histograma imaginii (a)
(c) Imagine cu histogramă egalizată
(d) Histograma imaginii (c)

Implementarea acestui algoritm s-a realizat prin funcția:

```
void CMainFrame::OnToolsEnhancementHistogramequalization()
{
 CROBOVIEWDoc  *SourceDoc;
 CROBOVIEWDoc  *DestinDoc;
 long                nk[256];
 double              Pk[256],sk[256];
 short               sk1[256];
 short               x,y;
 long                n;

 SourceDoc = GetActiveSourceDocument();
 DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
 memset(nk,0,sizeof(nk));
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
   nk[SourceDoc->BitmapMatrix[x][y][0]]++;
 n = Xmax * REALYMAX;
 for(x=0;x<256;x++)
  Pk[x] = (double)((double)nk[x]/(double)n);
 sk[0] = Pk[0];
 for(x=1;x<256;x++)
  sk[x] = sk[x-1] + Pk[x];
 for(x=0;x<256;x++)
  for(y=0;y<256;y++)
   if(sk[x] >= (double)((double)y/255) && sk[x] < (double)((double)(y+1)/255))
    {
    if((sk[x] - (double)((double)y/255)) <= ((double)((double)(y+1)/255) - sk[x]))
     sk1[x] = y;
    else
     sk1[x] = y+1;
    break;
    }
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
   DestinDoc->BitmapMatrix[x][y][0]=DestinDoc->BitmapMatrix[x][y][1]=
   DestinDoc->BitmapMatrix[x][y][2] = (unsigned char)sk1[SourceDoc->BitmapMatrix[x][y][0]];
}
```

1.5.2 Specificarea histogramei

Egalizarea histogramei este ideală pentru îmbunătățirea automată deoarece se bazează pe o funcție de transformare care este determinată unic de histograma imaginii originale. Cu toate acestea metoda este limitată, în sensul că unica ei funcție este liniarizarea histogramei, proces care nu se poate aplica când este disponibilă o informație inițială referitoare la o anumită formă a histogramei de ieşire.

Se va generaliza conceptul procesării histogramei prin dezvoltarea unei metode capabile să genereze o imagine cu o histogramă a intensității specificată. Egalizarea histogramei este un caz particular al acestei tehnici.

Fie $p_r(r)$ şi $p_z(z)$ intensităţile originale, respectiv dorite ale FDP. Se presupune că pentru o imagine dată, în primul rând se egalizează histograma folosind ecuaţia (1.5.3) adică:

$$s = T(r) = \int_0^r p_r(w)dw \tag{1.5.10}$$

Dacă există o imagine dorită, nivelele ei pot fi de asemenea egalizate folosind funcţia de transformare

$$v = G(z) = \int_0^z p_z(w)dw \tag{1.5.11}$$

Procesul invers, $z=G^{-1}(v)$ va regenera nivelele dorite.

Aceasta, este o formulare ipotetică deoarece nivelele de z sunt exact ceea ce se încearcă să se obţină. Este de remarcat că $p_s(s)$ şi $p_v(v)$ sunt densităţi uniforme identice atâta timp cât folosirea ecuaţiilor (1.5.10) şi (1.5.11) garantează o densitate uniformă, relativ la forma lui FDP de sub semnul integralei. Astfel că, în loc să se folosească v în procesul invers, se vor folosi nivelele inverse s obţinute din imaginea originală, nivelele rezultate $z=G^{-1}(s)$ vor avea o FDP dorită, $p_z(z)$.

Presupunând că $G^{-1}(s)$ este o valoare unică, procedura poate fi realizată în următoarele etape succesive :

1. Egalizarea nivelelor imaginii originale folosind ecuaţia (1.5.10)
2. Specificarea FDP-ului de intensitate dorit şi obţinerea funcţiei de transformare G(z) folosind ecuaţia(1.5.11).
3. Se aplică transformata inversă $z = G^{-1}(s)$ nivelelor de intensitate ale imaginii cu histograma egalizată obţinută la pasul 1.

Urmând această procedură se va genera o imagine de ieşire cu intensitatea specificată de FDP.

Cele două transformări cerute de specificarea histogramei, T(r) şi $G^{-1}(s)$, pot fi combinate într-o singura transformare:

$$z = G^{-1}(s) = G^{-1}[T(r)] \tag{1.5.12}$$

care leagă pe *r* de *z*. Este de remarcat faptul că, dacă $G^{-1}[T(r)] = T(r)$, această metodă se reduce la egalizarea histogramei.

Ecuația (1.5.12) arată că pentru a se putea specifica histograma, imaginea de intrare trebuie să aibă histograma egalizată explicit. Tot ceea ce este necesar se rezumă la faptul că T(r) trebuie să fie determinat şi combinat cu $G^{-1}(s)$ într-o singură transformare care este aplicată direct imaginii de intrare. Problema care apare în folosirea celor două transformări sau la folosirea reprezentării lor combinate la variabilele continue este obținerea funcției inverse în mod analitic.

În cazul discret, această problemă este ocolită de faptul că numărul nivelelor de intensitate distincte este de obicei relativ mic (normal 256) şi este posibil să se calculeze şi să se memoreze o mapare pentru fiecare valoare întreagă de pixel.

Formularea discretă a procedurii este următoarea:

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j) \tag{1.5.13}$$

$$G(z_i) = \sum_{j=0}^{i} p_z(z_j) \tag{1.5.14}$$

$$z_i = G^{-1}(şi) \tag{1.5.15}$$

unde $p_r(r_j)$ este calculat din imaginea de intrare iar $p_z(z_j)$ este specificat.

1.5.3 Îmbunătățirea locală a imaginii

Metodele prezentate pentru egalizarea şi specificarea histogramei, sunt globale, în sensul că pixelii sunt modificați de o funcție de transformare care este bazată pe distribuția intensității peste întreaga imagine. În timp ce această tratare globală este aplicabilă pentru o îmbunătățire a întregii imagini, este deseori necesar să se îmbunătățească detalii din zone mai mici. Din cauză că numărul de pixeli din aceste zone pot avea o influență neglijabilă asupra calculului transformatei globale, folosirea tehnicilor globale va conduce rareori la o îmbunătățire locală acceptabilă. Soluția acestei probleme este dezvoltarea de funcții de transformare ce se bazează pe distribuția intensității, sau a altor proprietăți, într-o vecinătate a fiecărui pixel dintr-o imagine dată [121], [130], [135], [154].

Tehnicile de procesare a histogramei discutate mai sus sunt uşor de aplicat şi în cazul îmbunătățirii locale. Procedura trebuie să-şi definească o vecinătate *n* x *m* şi să mute centrul acesteia pixel cu pixel. La fiecare locație, se va calcula histograma pe

n x *m* puncte din vecinătate şi se va obține fie o funcție de egalizare a histogramei, fie una de specificare. Această funcție este folosită în final pentru maparea intensității pixelului aflat în centrul vecinătății. Centrul regiunii *n* x *m* este apoi mutat într-o locație de pixel adiacentă şi procedura se repetă. Din cauză că se modifică doar un rând sau o coloană a vecinătății în timpul translației pixel cu pixel a regiunii, este posibil să se actualizeze histograma obținută în locația precedentă cu noile date introduse la fiecare pas al mişcări. Această abordare are avantaje evidente asupra calculului repetat al histogramei asupra a *n* x *m* pixeli de fiecare dată când regiunea este mutată cu o locație de pixel.

O altă abordare folosită des pentru reducerea calculului este impunerea restricției ca regiunile să nu se suprapună, dar aceasta produce de obicei un efect nedorit de tablă de şah.

În locul folosirii histogramelor, se poate realiza îmbunătățirea locală pe baza altor proprietăți ale intensității pixelilor într-o vecinătate. Valoarea şi variația (derivata) intensității sunt două astfel de proprietăți care sunt utilizate frecvent datorită importanței lor în aspectul unei imagini. Astfel valoarea intensității este o măsură a mediei luminozității iar variația sa o măsură a contrastului. O transformare locală tipică bazată pe aceste concepte mapează intensitatea unei imaginii originale f(x,y) într-o nouă imagine g(x,y) cu ajutorul următoarei transformări în fiecare poziție de pixel (x,y):

$$g(x, y) = A(x, y)[f(x, y) - m(x, y)] + m(x, y) \qquad (1.5.16)$$

unde :

$$A(x, y) = k\frac{M}{\sigma(x, y)} \qquad 0 < k < 1 \qquad (1.5.17)$$

În această formulare :

- m(x,y) reprezintă valoarea intensității calculată într-o vecinătate centrată pe (x,y)
- $\sigma(x, y)$ reprezintă derivata standard calculată într-o vecinătate centrată pe (x,y)
- M este valoarea globală a lui f(x,y)

- k este o constantă în domeniul indicat mai sus.

Este important de remarcat că A, m, şi σ sunt variabile ce depind de o vecinătate predefinită a lui (x,y). Aplicarea ***factorului de câştig local*** A(x,y) diferenței dintre f(x,y) şi valorii locale a intensității, amplifică variația locală. Deoarece A(x,y) este invers proporțional cu derivata standard a intensității, zonele cu contrast redus vor obține un câştig mai mare. Valoarea este readunată la relația (1.5.16) pentru a restaura nivelul mediu de intensitate al imaginii în regiunea locală.

În practică este bine de multe ori să se readune o fracțiune din valoarea locală şi să se restricționeze variația lui A(x,y) între cele doua limite $[A_{min}, A_{max}]$ în scopul echilibrării variațiilor prea largi de intensitate în regiunile izolate.

1.6 Detecția de contur

Detecția de contur joacă un rol central în vederea artificială, servind ca pas inițial în preprocesare pentru o mare varietate de algoritmi de detecție a obiectelor. Se vor evidenția fundamentele tehnicilor de detecție a punctelor de contur.

1.6.1 Formulări de bază

În principiu ideea ce stă la baza celor mai multe tehnici de detecție este calculul operatorului local de derivare. Acest concept poate fi uşor ilustrat cu ajutorul figurii 1.6.1. Figura 1.6.1,a prezintă o imagine a unui obiect simplu luminos pe un fond întunecat, profilul intensității de-a lungul unei linii orizontale de scanare a imaginii, prima şi a doua derivată a profilului [119], [142]. Este de remarcat din profil că un contur (tranziția de la întunecat la luminos) este modelat cu o rampă, mai degrabă decât printr-un un salt brusc de intensitate. Acest model este reprezentativ prin faptul că în imaginile digitale contururile sunt în general puțin tulburi ca rezultat al achiziției.

Prima derivată a unui contur modelat în această manieră este zero în toate regiunile de intensitate constantă şi are o valoare constantă în timpul unei tranziții de intensitate. Pe baza acestor remarci şi pe conceptele ilustrate în figura 1.6.1, este evident că mărimea primei derivate poate fi folosită pentru detectarea prezenței unui contur, în timp ce semnul celei de-a doua derivate poate fi folosit pentru determinarea apartenenței unui pixel la partea întunecată (fond) sau la cea luminoasă (obiect) de-a lungul unui contur.

Semnul derivatei a doua din figura 1.6.1,a de exemplu, este pozitiv pentru pixelii ce se află pe partea întunecată atât pe rampa de urcare cât şi pe cea de coborâre a unui obiect, şi este negativ pentru pixelii aflați pe partea luminoasă a acestor contururi. Comentarii similare se pot aplica şi în cazul unui obiect negru pe un fond luminos ca în figura 1.6.1,b. Se remarcă aceeaşi interpretare a semnului derivatei a doua.

Deşi prezentarea principală a fost limitată la un profil unidimensional orizontal, argumente similare se aplică la un contur cu orice orientare în imagine. Va fi necesară doar definirea unui profil perpendicular pe direcția conturului în fiecare punct dat şi interpretarea rezultatelor ca în cazul anterior.

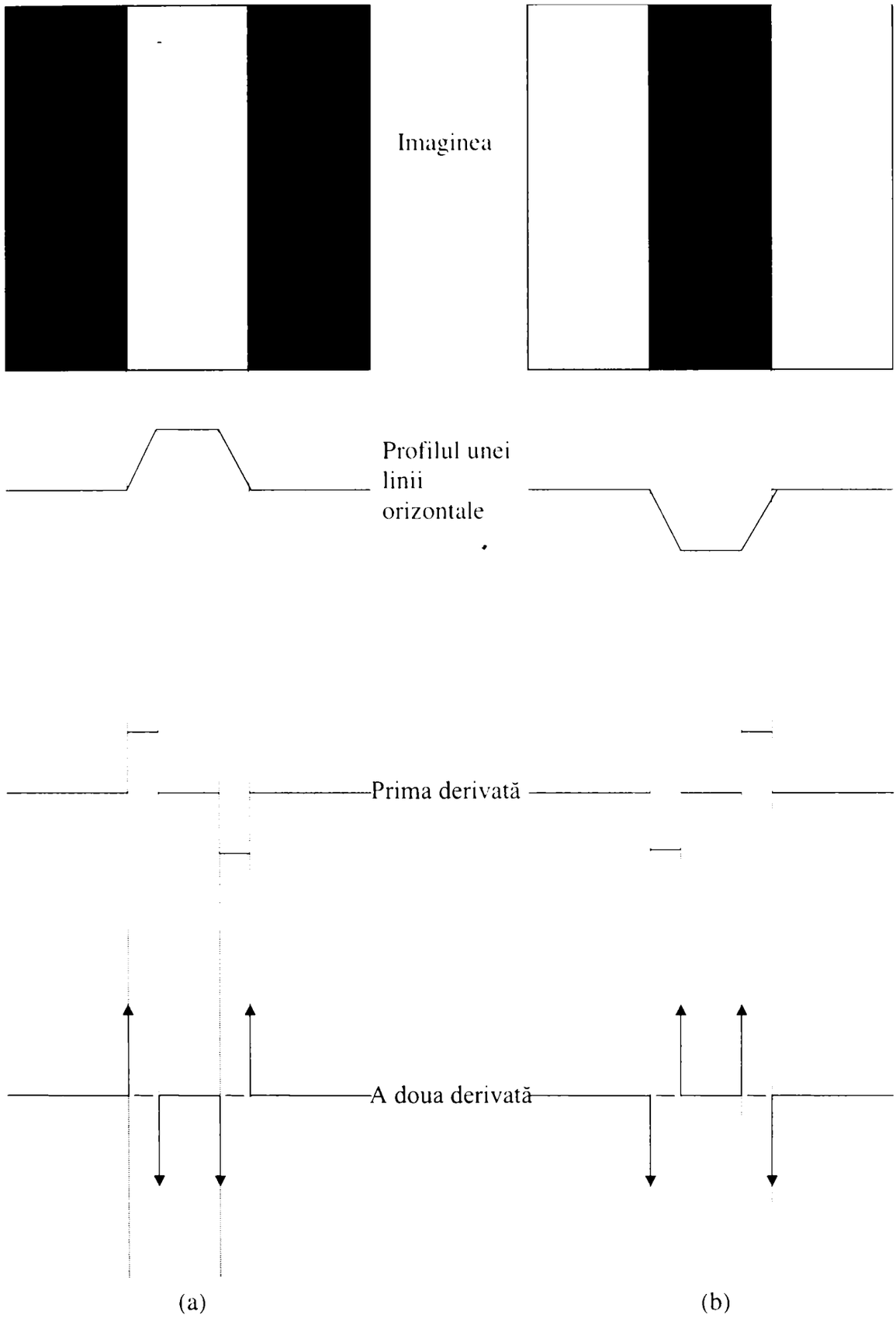


Fig. 1.6.1

Elemente ale detecției de contur folosind operatori de derivare
(a) Obiect luminos pe fond întunecat
(b) Obiect întunecat pe fond luminos

Prima derivată în fiecare punct din imagine poate fi obținută folosind mărimea gradientului în acel punct, iar derivata a doua este dată de Laplacian.

1.6.2 Operatori de tip gradient

Gradientul unei imagini f(x,y) la locația (x,y) este definit ca un vector bidimensional:

$$G[f(x,y)]=\begin{bmatrix} G_x \\ G_y \end{bmatrix}=\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{1.6.1}$$

Pentru detecția de contur interesează, numai mărimea acestui vector, |G[f(x,y)]|:

$$|G[f(x,y)]|=\left[G_x^2+G_y^2\right]^{1/2}=\left[\left(\frac{\partial f}{\partial x}\right)^2+\left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \tag{1.6.2}$$

Se obișnuiește de multe ori să se aproximeze gradientul (1.6.2) cu suma valorilor absolute ale componentelor:

$$G[f(x,y)]\approx|G_x|+|G_y| \tag{1.6.3}$$

Această aproximare este considerabil mai uşor de implementat, în special dacă se utilizează un hardware dedicat.

Se remarcă din ecuația (1.6.2) posibilitatea calculului gradientului bazat în mod simplu pe utilizarea derivatelor de ordinul 1 ($\partial f/\partial x$ şi $\partial f/\partial y$). Există câteva moduri pentru introducerea acestui calcul într-o imagine digitală. Unul din ele este folosirea diferențelor de gradul 1 între pixelii adiacenți, adică:

$$G_x=\frac{\partial f}{\partial x}=f(x,y)-f(x-1,y) \tag{1.6.4}$$

$$G_y=\frac{\partial f}{\partial y}=f(x,y)-f(x,y-1) \tag{1.6.5}$$

O definiție mai complicată folosind pixelii dintr-o vecinătate 3x3 centrată în (x, y) este dată de:

$$G_x = \frac{\partial f}{\partial x} = [f(x + 1, y - 1) + 2f(x + 1, y) + f(x + 1, y + 1)] - [f(x - 1, y - 1) +$$

$$2f(x - 1, y) + f(x - 1, y + 1)] = (g + 2h + i) - (a + 2b + c) \quad (1.6.6)$$

$$G_y = \frac{\partial f}{\partial y} = [f(x - 1, y + 1) + 2f(x, y + 1) + f(x + 1, y + 1)] - [f(x - 1, y - 1) +$$

$$2f(x, y - 1) + f(x + 1, y - 1)] = (c + 2e + i) - (a + 2d + g) \quad (1.6.7)$$

unde s-au folosit literele de la *a* până la *i* pentru reprezentarea vecinilor unui punct (x, y). Vecinătatea 3x3 a lui (x, y) folosind această notație simplificată este prezentată în figura 1.6.2,a.

a	b	c
d	(x, y)	e
g	h	i

(a)

-1	-2	-1
0	0	0
1	2	1

(b)

-1	0	1
-2	0	2
-1	0	1

(c)

Fig. 1.6.2
(a) O vecinătate 3x3 a punctului (x, y)
(b) Masca folosită pentru calculul lui G_x
(c) Masca folosită pentru calculul lui G_y

Se remarcă faptul că pixelii cei mai apropiați lui (x,y) au ponderea 2 în acest caz particular de definiție a derivatei digitale.

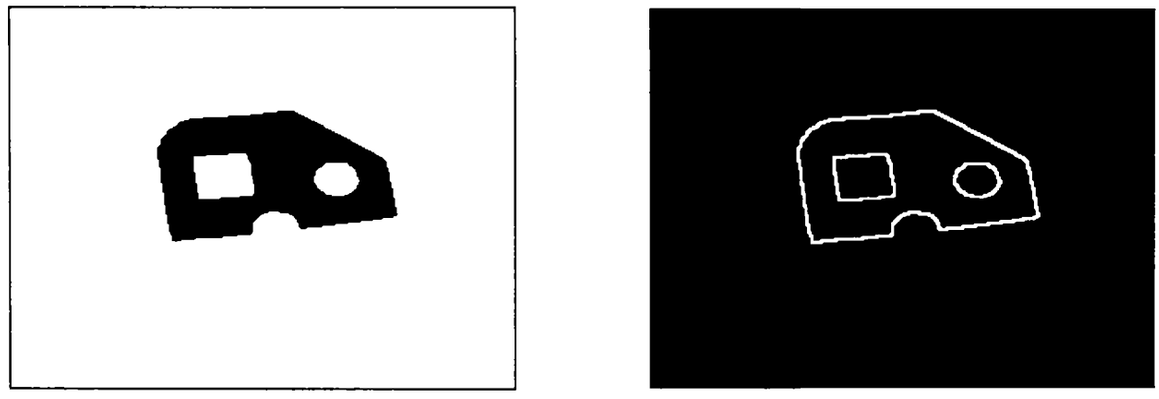
Metoda are avantajul creşterii aproximării, făcând astfel gradientul mai puțin sensibil la zgomot. Este posibilă definirea gradientului peste o vecinătate mai mare, dar operatorii 3x3 sunt cei mai utilizați în aplicațiile de vedere artificială în robotica industrială deoarece au o viteză mare de calcul şi cerințe hardware modeste.

Aşa cum indică relația (1.6.6), G_x poate fi calculat folosind masca din figura 1.6.2.b. Similar, G_y poate fi obținut folosind masca din figura 1.6.2c. Aceste două măşti sunt cunoscute sub numele de ***operatori Sobel*** [137]. Răspunsurile acestor două măşti în orice punct (x,y) sunt combinate folosind ecuațiile (1.6.2) sau (1.6.3) pentru a obține o aproximare a gradientului în acel punct. Deplasând aceste măşti de-a lungul imaginii f(x,y) se va obține gradientul tuturor punctelor din imagine.

Există numeroase moduri în care se poate genera o imagine de ieşire, g(x,y), bazată pe calculul gradienților. Cel mai simplu mod de abordare este atribuirea valorii lui g la coordonatele (x,y) a gradientul imaginii de intrare f în acel punct, adică:

$$g(x, y) = |G[f(x, y)]| \qquad (1.6.8)$$

Un exemplu folosind această metodă de generare a unei imagini gradient este prezentat în figura 1.6.3.



(a) (b)

Fig. 1.6.3
(a) Imaginea de intrare
(b) Imaginea de ieşire

O altă metodă este aceea de a crea o imagine binară folosind relația :

$$g(x,y)=\begin{cases}1 & \text{daca } |G[f(x,y)]>T\\ 0 & \text{daca } |G[f(x,y)]\le T\end{cases} \qquad (1.6.9)$$

unde T este un prag pozitiv. În acest caz doar pixelii a căror gradienți sunt mai mari decât T sunt considerați importanți.

Astfel, folosirea relației (1.6.9) poate fi considerată ca o procedură care extrage doar acei pixeli ce sunt caracterizați de o tranziție de intensitate semnificativă (determinată de T). O analiză ulterioară a pixelilor rezultați este necesară pentru a şterge punctele izolate şi pentru a uni pixeli de-a lungul contururilor adecvate care în final vor determina obiectul din cadrul imaginii.

Funcția care implementează câteva măşti de detecție se prezintă astfel :

```
void CMainFrame::EdgeDetection(long type,BOOL NewDocument)
{
 EdgeDetectionMethodDlg  edm_dlg;
 CROBOVIEWDoc            *SourceDoc;
 CROBOVIEWDoc            *DestinDoc;
 BYTE                    BitmapMatrix[320][240];
 short                   x,y,i,Maxim;
 short                   DestinValue[8];
 short                   FreemanMask[8][9]= {{ 1, 1, 1, 1,-2, 1,-1,-1,-1},
                                             { 1, 1, 1,-1,-2, 1,-1,-1, 1},
                                             {-1, 1, 1,-1,-2, 1,-1, 1, 1},
                                             {-1,-1, 1,-1,-2, 1, 1, 1, 1},
                                             {-1,-1,-1, 1,-2, 1, 1, 1, 1},
                                             { 1,-1,-1, 1,-2,-1, 1, 1, 1},
                                             { 1, 1,-1, 1,-2,-1, 1, 1,-1},
                                             { 1, 1, 1, 1,-2,-1, 1,-1,-1}};
 short                   SobelMask[2][9]=    {{-1,-2,-1, 0, 0, 0, 1, 2, 1},
                                              {-1, 0, 1,-2, 0, 2,-1, 0, 1}};
 short                   PrewittMask[2][9]=  {{-1,-1,-1, 0, 0, 0, 1, 1, 1},
                                              {-1, 0, 1,-1, 0, 1,-1, 0, 1}};
 short                   KirschMask[1][9]=   {{ 5, 5, 5,-3, 0,-3,-3,-3,-3}};
 short                   VerticalMask[1][9]= {{-1, 2,-1,-1, 2,-1,-1, 2,-1}};
 short                   HorizontalMask[1][9]={{-1,-1,-1, 2, 2, 2,-1,-1,-1}};
 short                  _45Mask[1][9]=         {{-1,-1, 2,-1, 2,-1, 2,-1,-1}};

 if(type == 0)
  {
  if(edm_dlg.DoModal() != IDOK)
   return;
  else
   type = edm_dlg.m_CheckedRadioButton;
  }
 SourceDoc = GetActiveSourceDocument();
 if(NewDocument)
  DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
 else
  memset(&BitmapMatrix,0xFF,320*240);
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
   {
```

```
switch(type)
 {
 case IDC_RADIO_FREEMAN : DestinValue[0]=ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[0]));
                          DestinValue[1]=ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[1]));
                          DestinValue[2] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[2]));
                          DestinValue[3] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[3]));
                          DestinValue[4] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[4]));
                          DestinValue[5] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[5]));
                          DestinValue[6] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[6]));
                          DestinValue[7] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&FreemanMask[7]));
                          Maxim = -32000;
                          for(i=0;i<8;i++)
                          if(DestinValue[i] > Maxim)
                            Maxim = DestinValue[i];
                          break;
 case IDC_RADIO_SOEBEL : DestinValue[0] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&SobelMask[0]));
                          DestinValue[1] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&SobelMask[1]));
                          Maxim = (short)sqrt((double)(pow((double)DestinValue[0],2) +
                                                  pow((double)DestinValue[1],2)));
                          break;
 case IDC_RADIO_PREWITT:  DestinValue[0] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&PrewittMask[0]));
                          DestinValue[1] = ApplyMask(((unsigned char *)(&(SourceDoc
                              ->BitmapMatrix))),x,y,(short *)(&PrewittMask[1]));
                          Maxim = abs(DestinValue[0]) + abs(DestinValue[1]);
                          break;
 case IDC_RADIO_KIRSCH :  memset(&DestinValue,0,sizeof(DestinValue));
                          if(x>0 && y>0)
                           DestinValue[0] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x-1,y-1,(short *)(&KirschMask[0])));
                          if(y>0)
                           DestinValue[1] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x,y-1,  (short *)(&KirschMask[0])));
                          if(x<Xmax && y>0)
                           DestinValue[2] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x+1,y-1,(short *)(&KirschMask[0])));
                          if(x>0)
                           DestinValue[3] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x-1,y,  (short *)(&KirschMask[0])));
                          if(x<Xmax)
                          DestinValue[4] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x+1,y,  (short *)(&KirschMask[0])));
                          if(x>0 && y<REALYMAX)
                          DestinValue[5] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x-1,y+1,(short *)(&KirschMask[0])));
                          if(y<REALYMAX)
                           DestinValue[6] = abs(ApplyMask(((unsigned char *)(&(Source
                              Doc->BitmapMatrix))),x,y+1,  (short *)(&KirschMask[0])));
                           if(x<Xmax && y<REALYMAX)
                           DestinValue[7] = abs(ApplyMask(((unsigned char *)(&(Source
```

```
                                    Doc->BitmapMatrix))),x+1,y+1,(short *)(&KirschMask[0])));
                                  Maxim = 0;
                                  for(i=0;i<8;i++)
                                   if(DestinValue[i] > Maxim)
                                    Maxim = DestinValue[i];
                                  break;
    case IDC_RADIO_VERTICAL:  Maxim = ApplyMask(((unsigned char *)(&(SourceDoc
                                        ->BitmapMatrix))),x,y,(short *)(&VerticalMask[0]));
                                  break;
    case IDC_RADIO_HORIZONTAL:    Maxim = ApplyMask(((unsigned char *)(&(SourceDoc
                                      ->BitmapMatrix))),x,y,(short *)(&HorizontalMask[0]));
                                  break;
    case IDC_RADIO_45         : Maxim = ApplyMask(((unsigned char *)(&(SourceDoc
                                        ->BitmapMatrix))),x,y,(short *)(&_45Mask[0]));
                                  break;
    default                   : return;
    }
  if(Maxim > 0)
    {
    if(NewDocument)
      DestinDoc->BitmapMatrix[x][y][0] = DestinDoc->BitmapMatrix[x][y][1] =
                                        DestinDoc->BitmapMatrix[x][y][2] = 0xFF;
    else
      BitmapMatrix[x][y] = 0xFF;
    }
  else
    {
    if(NewDocument)
      DestinDoc->BitmapMatrix[x][y][0] = DestinDoc->BitmapMatrix[x][y][1] =
                                        DestinDoc->BitmapMatrix[x][y][2] = 0;
    else
      BitmapMatrix[x][y] = 0;
    }
 }
 if(!NewDocument)
  {
  for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
      SourceDoc->BitmapMatrix[x][y][0] = SourceDoc->BitmapMatrix[x][y][1] =
                                        SourceDoc->BitmapMatrix[x][y][2] = BitmapMatrix[x][y];
  GetActiveFrame()->RedrawWindow();
  }
}
```

1.6.3 Operatorul Laplacian

Pentru o funcție continuă Laplacianul este un operator derivativ de ordinul 2 :

$$L\left[f(x,y)\right]=\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2} \qquad (1.7.10)$$

Pentru imaginile digitale, Laplacianul este definit de forma [95], [99], [134]:

$$L[f(x, y)] = [f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1)] - 4f(x, y) \qquad (1.7.11)$$

Această formulare digitală a Laplacianului are valoare zero în regiunile constante şi variaţie rampă în jurul conturului, aşa cum se presupune pentru derivata a doua. Implementarea relaţiei (1.7.11) poate fi bazată pe masca din figura 1.6.4.

0	1	0
1	-4	1
0	1	0

Fig. 1.6.4
Masca folosită în calculul Laplacianului

Deşi, Laplacianul răspunde la tranziţii de intensitate, el este rar folosit singur în detecţia de contur. Motivul este acela că, fiind un operator de derivare de ordinul 2, Laplacianul este de obicei inacceptabil de sensibil la zgomot. Astfel, acest operator are de obicei un rol secundar, servind ca detector pentru stabilirea apartenenţei unui pixel la partea întunecată sau luminoasă a conturului.

1.6.4. Detecţia de contur prin deplasare şi diferenţă

Aşa cum îi indică şi numele, algoritmul constă din scăderea unei copii spaţiale deplasată a unei imagini, din imaginea originală. O implementare simplă a acestei operaţii se face prin procesul de convoluţie.

Examinând figura 1.6.5, nucleele de deplasare şi diferenţă, ar trebui să creeze ideea că imaginea se va deplasa faţă de ea însăşi cu un pixel pe orizontală şi unul pe verticală, separat sau împreună, şi se va scădea copia deplasată din original. Trebuie remarcat faptul că filtrul unitate are coeficientul central al nucleului egal cu 1, aceasta reprezentând imaginea originală după convoluţie.

Ca exemplu, se va folosi filtrul orizontal de deplasare şi diferenţă. Filtrul va deplasa imaginea vertical şi o va scădea din imaginea iniţială . Aceasta va avea efectul iluminării contururilor orizontale.

Algoritmul folosit este foarte simplu. Deplasând şi scăzând această imagine deplasată din imaginea originală, se realizează o măsură a vitezei variaţiei de intensitate. În zonele în care variaţia de intensitate a pixelilor este minimă, panta rezultată va fi apropiată de 0 şi calculul pixelului va conduce spre o valoare de pixel apropiată de 0. În zonele în care variaţia de intensitate a pixelilor este mare, scăderea va genera o pantă mare, şi noua valoare de pixel va fi una foarte luminoasă. Cu cât este mai mare panta, cu atât pixelul va fi mai luminos (până când nu se va mai putea maximaliza valoarea intensităţii şi va trebui limitată).

Trebuie remarcat faptul că tranziţiile de la alb la negru din imagine vor genera pante negative. Se va putea considera fie valoarea absolută a noii valori calculate pentru a putea sesiza atât tranziţiile de la alb la negru cât şi cele de la negru la alb din cadrul imaginii, sau se va limita studiul doar la tranziţiile de la negru la alb şi se vor seta valorile negative obţinute la zero (adică negru).

0 -1 0	0 0 0	-1 0 0
0 1 0	-1 1 0	0 1 0
0 0 0	0 0 0	0 0 0
1	1	1
Orizontal	Vertical	Oriz/Vert

Fig. 1.6.5
Filtre de diferenţă şi de deplasare

1.7 Stabilirea unui prag pentru prelucrare imaginii

Stabilirea unui prag pe imagine este una din principalele tehnici utilizate în sistemele industriale de vedere pentru detectarea obiectelor, în special în aplicațiile ce conțin un mare număr de rezultate.

Se presupune că histograma de intensitate din figura 1.7.1 corespunde unei imagini, f(x,y) [148], compusă din obiecte luminoase pe un fond negru, în aşa fel încât pixelii obiectului şi fondului au intensități grupate în două grupuri dominante. Unul din modurile evidente pentru extragerea obiectelor din fond este selectarea unui prag T care separă cele două grupuri de intensitate.

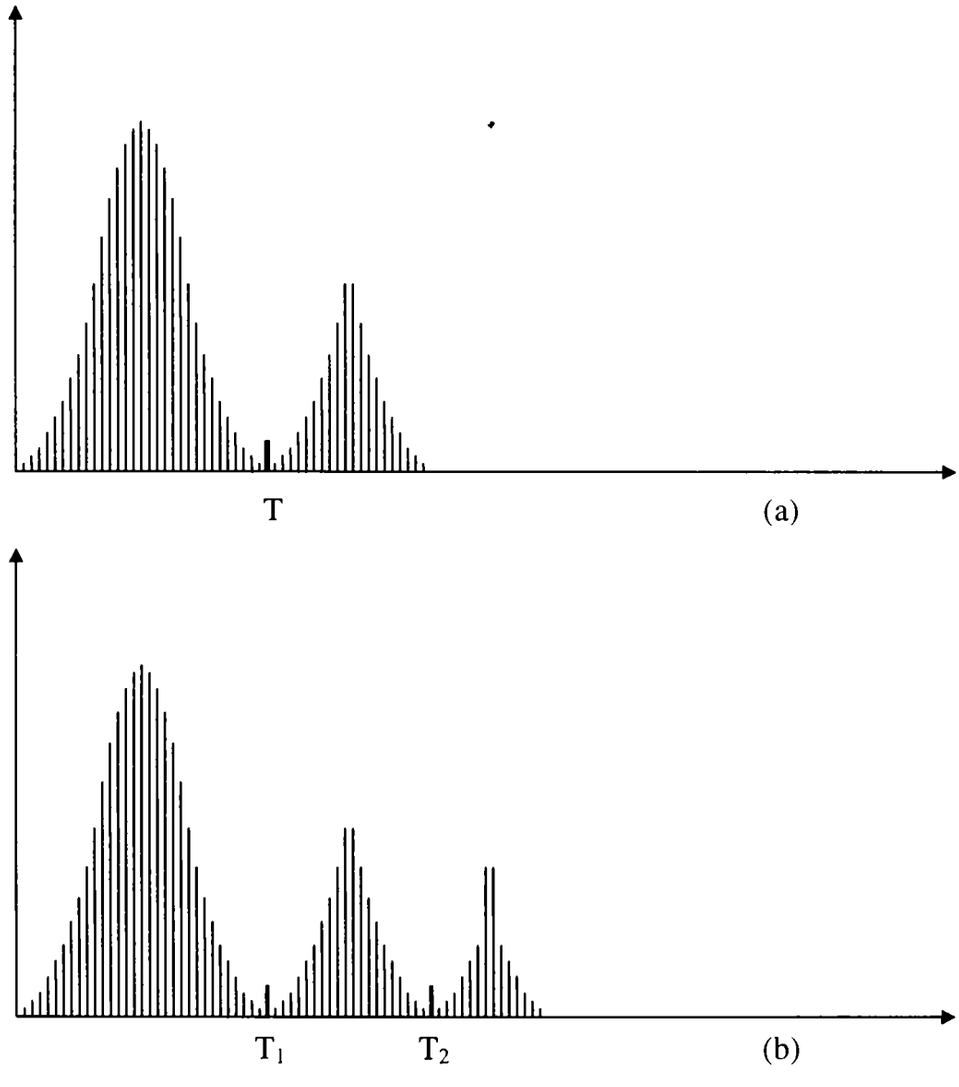


Fig. 1.7.1
Histograme de intensitate ce pot fi partiționate de un singur prag (a) şi de praguri multiple (b)

Astfel, orice punct (x,y) pentru care f(x,y) > T este numit ***punct al obiectului***, toate celelalte sunt ***puncte de fond***.

O abordare mult mai generală a acestui caz este prezentată în figura 1.7.1,b . În acest caz histograma imaginii este caracterizată de trei grupări dominante (de exemplu două tipuri de obiecte luminoase pe un fond întunecat). Şi aici, se poate folosi aceeaşi abordare de bază şi se va clasifica un punct (x,y) ca aparţinând unei clase de obiecte dacă $T_1 < f(x,y) \le T_2$, celeilalte clase dacă f(x,y) > T_2, sau fondului dacă $f(x,y) \le T_1$.

Acest tip de ***praguri multinivel*** sunt în general mai puţin sigure decât cel cu un singur prag datorită dificultăţii în determinarea pragurilor multiple care izolează efectiv o regiune de interes, în special în cazul în care numărul de grupări din histogramă este mare. Dacă probleme de această natură, sunt tratate prin praguri cel mai bine sunt adresate de o singură variabilă de prag.

Pe baza conceptelor anterioare, se poate imagina folosirea pragurilor ca o operaţie ce presupune testări cu o funcţie T de forma :

$$T = T[x, y, p(x, y), f(x, y)] \tag{1.7.1}$$

unde :

- f(x,y) este intensitatea punctului (x,y)
- p(x,y) reprezintă unele proprietăţi locale ale acestui punct, de exemplu, media intensităţii într-o vecinătate centrată pe (x,y)

Se va crea o imagine cu prag g(x,y) prin următoarea definiţie:

$$g(x,y) = \begin{cases} 1 & \text{daca} \quad f(x,y) > T \\ 0 & \text{daca} \quad f(x,y) \le T \end{cases} \tag{1.7.2}$$

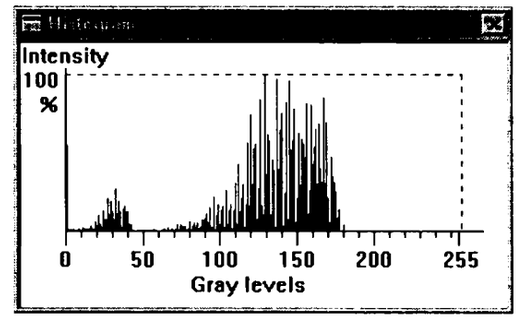
Prin examinarea lui g(x,y), se va aprecia faptul că pixelii notaţi cu 1 (sau orice alt nivel convenţional de intensitate) corespund obiectelor, în timp ce pixelii notaţi cu 0 corespund cu fondul.

Când T depinde doar de f(x, y), operaţia de folosire a pragului se numeşte ***globală*** (figura 1.7.1,a). Dacă T depinde atât de f(x,y) cât şi de p(x,y) atunci se va numi ***locală***. Dacă în plus T depinde de coordonatele spaţiale x şi y, se va numi ***prag dinamic***.

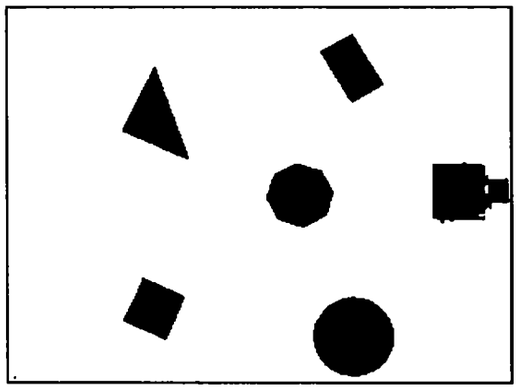
Figura 1.7.2 prezintă o exemplificare a metodei globale. Histograma 1.7.2,b a imaginii originale 1.7.2,a, scoate în evidenţă existenţa unor obiecte a căror intensitate

este mai mică de 50 şi pixelii de fond cu o intensitate mai mare 70. Prin selectarea unui prag T=60 se va obține o imagine binară 1.7.2,c în care obiectele sunt extrase din cadrul imaginii originale.

(a)



(b)



(c)

Fig. 1.7.2
(a) Imagine originală
(b) Histograma imaginii (a)
(c) Imaginea binarizată cu un prag de 60

Funcția care implementează acest algoritm este prezentată în continuare :

```
void CMainFrame::OnToolsTreshold()
{
 CROBOVIEWDoc        *SourceDoc;
 CROBOVIEWDoc        *DestinDoc;
 TresholdDlg          t_dlg;
 short                x,y,TresholdValue;

 SourceDoc = GetActiveSourceDocument();
 OnToolsEnhancementShowhistogram();

 t_dlg.m_TresholdValue = 127;
 if(t_dlg.DoModal() == IDOK)
  TresholdValue = t_dlg.m_TresholdValue;
 else
  {
```

```
  HistogramWindow->MDIDestroy();
  return;
  }
 HistogramWindow->MDIDestroy();
 DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
   if(SourceDoc->BitmapMatrix[x][y][0] <= TresholdValue)
    DestinDoc->BitmapMatrix[x][y][0] = DestinDoc->BitmapMatrix[x][y][1] =
                                        DestinDoc->BitmapMatrix[x][y][2] = 0;
   else
    DestinDoc->BitmapMatrix[x][y][0] = DestinDoc->BitmapMatrix[x][y][1] =
                                        DestinDoc->BitmapMatrix[x][y][2] = 255;
 DestinDoc->BitmapColorType = BINARY;
}
```

Funcția ce urmează calculează şi afişează histograma unei imagini:

```
void CMainFrame::OnToolsEnhancementShowhistogram()
{

 CROBOVIEWDoc      *SourceDoc;
 long              IntensityNumber[256];
 long              MaxDensity;
 short             x,y;
 CDC                       *HistogramDC;
 RECT              HistogramRect={161,131,487,331};
 CPen              SolidRedPen(PS_SOLID,1,RGB(255,0,0)),
                   DotBlackPen(PS_DOT,1,RGB(0,0,0)),
                   SolidBlackPen(PS_SOLID,1,RGB(0,0,0));


 SourceDoc = GetActiveSourceDocument();
 memset(IntensityNumber,0,sizeof(IntensityNumber));
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
   IntensityNumber[SourceDoc->BitmapMatrix[x][y][0]]++;
 MaxDensity = 0;
 for(x=0;x<256;x++)
  if(IntensityNumber[x] > MaxDensity)
   MaxDensity = IntensityNumber[x];
 HistogramWindow = new CMDIChildWnd;
 HistogramWindow->Create(NULL,"Histogram",WS_CHILD | WS_VISIBLE | WS_OVERLAPPED |
                         WS_CAPTION | WS_SYSMENU, HistogramRect, NULL,NULL);
 HistogramDC = HistogramWindow->GetDC();
 CPen *OldPen = HistogramDC->SelectObject(&SolidBlackPen);
 HistogramDC->MoveTo(29,15);
 HistogramDC->LineTo(29,121);
 HistogramDC->LineTo(301,121);
 HistogramDC->SelectObject(&DotBlackPen);
 HistogramDC->MoveTo(29,20);
 HistogramDC->LineTo(286,20);
 HistogramDC->LineTo(286,121);
 HistogramDC->SelectObject(&SolidBlackPen);
 for(x=0;x<255;x+=10)
  {
  HistogramDC->MoveTo(29+x,121);
  HistogramDC->LineTo(29+x,126);
  }
 for(x=0;x<250;x+=50)
```

```
  {
  HistogramDC->MoveTo(29+x,121);
  HistogramDC->LineTo(29+x,131);
  }
 HistogramDC->MoveTo(29+255,121);
 HistogramDC->LineTo(29+255,131);
 HistogramDC->TextOut(25,132,"0",1);
 HistogramDC->TextOut(71,132,"50",2);
 HistogramDC->TextOut(117,132,"100",3);
 HistogramDC->TextOut(167,132,"150",3);
 HistogramDC->TextOut(217,132,"200",3);
 HistogramDC->TextOut(274,132,"255",3);
 HistogramDC->TextOut(110,148,"Gray levels",11);
 HistogramDC->TextOut(1,0,"Intensity",9);
 HistogramDC->TextOut(1,16,"100",3);
 HistogramDC->TextOut(13,32,"%",1);
 HistogramDC->SelectObject(&SolidRedPen);
 for(x=0;x<256;x++)
  {
  HistogramDC->MoveTo(x+30,120);
  HistogramDC->LineTo(x+30,120-((100*IntensityNumber[x])/MaxDensity));
  }
 HistogramDC->SelectObject(OldPen);
 HistogramWindow->ShowWindow(SW_SHOWNORMAL);
 HistogramWindow->UpdateWindow();
 ReleaseDC(HistogramDC);
}
```

1.8 Domeniile de frecvență

Există situații în care nu este suficientă operarea în domeniul spațial, adică nu se va lucra adresând punctele prin coordonatele lor (x, y), ci prin domeniul de frecvență, în care coordonatele sunt (u, v) unde: u=1/x iar v=1/y.

Unul din principalele motive ale acestei abordări, este acela că în anumite situații ea este mult mai eficientă, iar în altele cazuri de filtrări este unica metodă aplicabilă. De exemplu, convoluția în domeniul spațial este o operație dificilă, implicând manipularea multor pixeli, dar în domeniul de frecvență, ea se reduce la o simplă înmulțire. Mai trebuie precizat faptul că utilizarea domeniilor de frecvență este strâns legată de tehnicile de restaurare şi deconvoluție.

1.8.1 Transformata Fourier a unei imagini

Orice semnal poate fi descompus într-un număr infinit de componente sinusoidale definite prin frecvență, amplitudine şi fază. Transformata Fourier este un operator matematic care redă densitatea spectrală a unei imagini, adică distribuția diferitelor componente de frecvență ale imaginii.

Astfel, spectrul unui semnal sinusoidal de perioadă T este o linie la frecvența $v = 1/T$, iar un semnal care este o mixare a două semnale sinusoidale de perioade diferite are un spectru care conține două linii puternic conturate şi un număr de linii mai puțin conturate ce corespund armonicilor. Un spectru continuu reprezintă un semnal neperiodic.

Un astfel de spectru include atât informația de amplitudine cât şi informația de fază a unei sinusoide care aproximează semnalul. Informația de amplitudine generează un spectru de amplitudine, iar informația de fază generează un spectru de fază. Se poate inversa această situație, iar prin combinarea componentelor spectrului dat, va rezulta semnalul original. Această transformare poartă numele de ***transformata Fourier inversă***.

Transformata Fourier discretă a unei serii de puncte reale f(x) în domeniul de frecvență generează o componentă reală $R(v)$ şi o componentă imaginară $I(v)$ [24]:

$$R(v) = \Delta t \sum_{x=0}^{N-1} f(x) \cos(2\pi v x \Delta t) \qquad (1.8.1)$$

$$I(v) = -\Delta t \sum_{x=0}^{N-1} f(x) \sin(2\pi v x \Delta t) \tag{1.8.2}$$

Amplitudinea spectrului este:

$$A(v) = \sqrt{R(v)^2 + I(v)^2} \tag{1.8.3}$$

iar puterea spectrului este :

$$A^2(v) = R(v)^2 + I(v)^2 \tag{1.8.4}$$

Există o limită de eşantionare a semnalului, limită peste care nu se va mai obține nici o informație suplimentară. Această limită de frecvență este $v = \frac{1}{2\Delta t}$. Pe de altă parte, dacă eşantionarea este rară, se va obține o distorsionare. Prin substituirea acestei frecvențe în ecuațiile (1.8.1) şi (1.8.2) se obține:

$$R(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) \cos(2\pi \frac{xu}{N}) \tag{1.8.5}$$

$$I(u) = -\frac{1}{N} \sum_{x=0}^{N-1} f(x) \sin(2\pi \frac{xu}{N}) \tag{1.8.6}$$

Transformata Fourier inversă va fi dată de relația:

$$f(x) = \sum_{u=0}^{N-1} R(u) \cos(2\pi \frac{xu}{N}) - I(u) \sin(2\pi \frac{xu}{N}) \tag{1.8.7}$$

Dacă se realizează un program care să calculeze transformata Fourier printr-o metodă directă, timpul de calcul va fi proporțional cu N^2. Acest lucru creează probleme mari deoarece N atinge valori de ordinul câtorva mii în mod frecvent. Pentru rezolvarea acestei situații s-a conceput un algoritm mult mai rapid numit ***transformata Fourier rapidă*** sau TFR. Pentru acest algoritm timpul de calcul este proporțional cu $N \log_2 N$ care este mult mai avantajos.

Ceea ce s-a prezentat reprezintă transformata Fourier în cazul unidimensional. Pentru prelucrarea imaginilor este necesară transformata în 2D. Pentru aceasta, se vor exprima funcțiile trigonometrice în termeni exponențiali cu argumente imaginare. Astfel, ecuațiile pentru transformata Fourier şi inversa sa în unidimensional vor avea următoarea formă:

$$F(u)=\frac{1}{N}\sum_{x=0}^{N-1}f(x)\exp(-2\pi i\frac{xu}{N}) \tag{1.8.8}$$

$$f(x)=\sum_{u=0}^{N-1}F(u)\exp(2\pi i\frac{xu}{N}) \tag{1.8.9}$$

În cazul bidimensional, vor trebui calculate pentru două variabile:

$$F(u,\mathrm{v})=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)\exp\left\{-2\pi i\left[\left(\frac{xu}{M}\right)+\left(\frac{y\mathrm{v}}{N}\right)\right]\right\} \tag{1.8.10}$$

$$f(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,\mathrm{v})\exp\left\{2\pi i\left[\left(\frac{xu}{M}\right)+\left(\frac{y\mathrm{v}}{N}\right)\right]\right\} \tag{1.8.11}$$

Dacă matricea utilizată este un pătrat (adică în cazul algoritmului TFR) relațiile vor deveni următoarele:

$$F(u,\mathrm{v})=\frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}f(x,y)\exp\left(-2\pi i\frac{xu+y\mathrm{v}}{N}\right) \tag{1.8.12}$$

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}F(u,\mathrm{v})\exp\left(2\pi i\frac{xu+y\mathrm{v}}{N}\right) \tag{1.8.13}$$

Deoarece $F(u,\mathrm{v})$ este o funcție separabilă, aceste ecuații se mai pot scrie astfel:

$$F(u,\mathrm{v})=\frac{1}{N}\sum_{x=0}^{N-1}G(x,y)\exp\left(-2\pi i\frac{xu}{N}\right) \tag{1.8.14}$$

$$G(x,y)=\frac{1}{N}\sum_{y=0}^{N-1}f(x,y)\exp\left(-2\pi i\frac{y\mathrm{v}}{N}\right) \tag{1.8.15}$$

După cum se poate observa, tot ceea ce trebuie făcut în cazul bidimensional este să se calculeze transformata Fourier mai întâi pe rânduri şi după aceea pe coloane, astfel că, de fapt, s-a redus problema bidimensională la două probleme unidimensionale.

1.8.2 Convoluția

Una din facilitățile transformatei Fourier este şi operarea cu convoluții. Convoluția a două semnale din domeniul spațial este concepută ca o simplă multiplicare a transformatelor celor două semnale în domeniul de frecvență şi invers.

Acest lucru este foarte util pentru filtre. Este mult mai uşoară şi mai rapidă multiplicarea de numere într-un program pe calculator decât aplicarea unei convoluții asupra unor funcții. Deci, este mai uşoară aplicarea unei convoluții asupra a două imagini. După calcul transformatei Fourier a fiecărei funcții, se calculează produsul modulelor şi suma argumentelor. Urmează aplicarea transformatei inverse acestui rezultat. Produsul celor două imagini în frecvență va produce o deplasare a unei componente în raport cu cealaltă în domeniul spațial. Pentru a preveni aceasta, va trebui adus centrul celei de-a doua imagini, peste originea (aici colțul din stânga jos) primei imagini.

1.8.3 Corelația

Se consideră că există două imagini şi că se doreşte obținerea elementelor lor comune.

Se iau cele două imagini, se va deplasa una în raport cu cealaltă, măsurându-se gradul de corelație cu ajutorul funcției de corelație încrucişată:

$$a(k) = \sum_{l=-\infty}^{l=\infty} x(l)y(l-k) \qquad (1.8.16)$$

Cu cât este mai mare valoarea lui a(k) cu atât mai apropiate sunt cele două imagini.

În domeniul de frecvență, această operație este o simplă înmulțire a două transformate:

$$A(u) = X(u)Y^{*}(u) \qquad (1.8.17)$$

Una din aplicațiile corelației încrucişate este scoaterea în evidență a elementelor vagi din mijlocul imaginii.

1.8.4 Deconvoluția

Pentru reconstrucția unei imagini şi pentru compensarea deteriorării semnalului sunt disponibile mai multe tehnici. Tehnicile de deconvoluție formează o clasă care are ca scop restaurarea unei imagini prin crearea originalului său.

O imagine poate fi deteriorată din multe motive: optica camerei de luat vederi nu este perfectă, imaginea a fost luată de pe un scaner sau printr-o sticlă murdară, etc. Pentru oricare din aceste motive, dacă există o imagine ce are un anumit grad de deteriorare, şi se doreşte restaurarea ei, va trebui aplicată următoarea procedură: se presupune că imaginea existentă este o convoluție dintre imaginea originală şi o funcție de distribuție a punctelor, h(x, y), în care fiecare punct apare în aceleaşi condiții.

Pentru a realiza aceasta trebuie determinat dacă există o parte a imaginii care să pară că are un punct mânjit (dacă nu este posibil, se pot încerca câteva variante intuitive şi se va vedea efectul acestei tehnici, după care se va păstra cea mai buna imagine):

$$f(x, y) = \text{imaginea observată} = g(x, y)*h(x, y) \qquad (1.8.18)$$

Dacă se face transformata Fourier a acestei funcții, rezultă:

$$F(u, v) = G(u, v) \bullet H(u, v) \qquad (1.8.19)$$

Dacă se cunoaşte funcția de distribuție a punctelor, se va putea restaura imaginea aplicând relația:

$$G(u,v) = F(u,v)/H(u,v) \qquad (1.8.20)$$

Astfel, imaginea din domeniul de frecvență F(u,v) este multiplicată cu un filtru a cărui funcție de transfer este 1/H(u,v). Se poate însă întâmpla ca H(u,v) să aibă valoarea zero. În acest caz trebuie adunat un zgomot suplimentar N(u,v) deoarece el este oricum prezent în imagine. Astfel că, ecuația în spațiul frecvențelor fi:

$$F(u, v) = G(u, v) \bullet H(u, v) + N(u, v) \qquad (1.8.21)$$

şi rezultă:

$$G(u,v) = F(u,v)/H(u,v) - N(u,v)/H(u,v) \qquad (1.8.22)$$

Mai trebuie luată în considerare şi o altă problemă posibilă. Când H(u,v) este prea mic, atunci amândoi termenii (inclusiv termenul de zgomot) vor deveni prea mari, şi zgomotul poate fi amplificat. Aceasta înseamnă că imaginea iniţială va fi incomplet restaurată.

O tehnică dezvoltată pentru a elimina aceste probleme este filtrarea Wiener. Aceasta constă din scrierea imaginii în forma ei restaurată ca :

$$G(u,v)=[F(u,v)*H^{*}(u,v)]/[H(u,v)*H^{*}(u,v) + nu] \qquad (1.8.23)$$

unde :

- $H^{*}(u,v)$ este conjugata complexă a lui H(u,v)
- nu este zgomotul pe rata semnalului

Aceasta este o cantitate euristică, şi empirică. Lipsa zgomotului înseamnă că nu=0, şi filtrul Wiener devine filtru invers.

Pentru a reduce zgomotul din filtrarea Wiener se va realiza restaurarea imaginii într-o manieră iterativă. Se presupune că există următoarea ecuaţie iterativă:

$$q(x, y)_{k+1} = q(x, y)_k + [f(x, y) - q(x, y)_k \bullet h(x, y)] \qquad (1.8.24)$$

unde :

- k este indexul de iteraţie
- $q(x,y)_{k+1}$ este imaginea estimată la a k iteraţie,
- f(x,y) este imaginea originală
- h(x,y) este funcţia de distribuţie a punctelor

Convoluţia este reprezentată prin •. Se observă ca pe măsura ce indexul de iteraţie creşte, $[f(x,y) - q(x,y)_k \bullet h(x,y)]$ tinde la zero, şi tot ce rămâne este imaginea ideală q(x,y);

Avantajul acestei metode este lipsa transformatelor Fourier, astfel că toate calculele necesare pot fi realizate pe un PC. Convoluţia în domeniul spaţial se poate realiza doar cu câteva iteraţii (şi nici o transformată Fourier).

Partea II-a
Procesarea imaginii

În scopul caracterizării numeroaselor tehnici şi abordări folosite în vederea artificială, s-au introdus trei nivele de procesare :

- Joasă
- Medie
- Înaltă

Primul nivel a fost subiectul părții întâi a acestei teze de doctorat. Cel de-al doilea nivel va fi descris în continuare pe parcursul acestei părți.

Conceptul de ***inteligență*** este relativ vag definit, mai ales în cazul în care se referă la un robot. Cu toate acestea nu este greu de definit modul de comportare care s-ar putea caracteriza ca inteligent. Câteva caracteristici evidente se pot menționa :

- abilitatea de a extrage informații pertinente dintr-un fond de detalii irelevante
- capacitatea de a învăța din exemple şi de a generaliza aceste cunoştințe în aşa fel încât să le poată aplica în circumstanțe noi şi diferite
- abilitatea de a deduce fapte din informații incomplete
- capacitatea de a-şi genera scopuri dorite de sine însuşi, şi de a formula planuri pentru îndeplinirea acestor scopuri.

Este posibil de proiectat şi implementat un sistem de vedere cu aceste caracteristici într-un mediu limitat, dar în nici un caz nu se poate realiza un astfel de sistem cu performanțele adaptive ale sistemului de vedere uman. Deşi cercetările din domeniul sistemelor biologice descoperă continuu concepte noi şi promițătoare, starea de fapt din domeniul vederii artificiale este în cea mai mare parte bazată pe formulări analitice proiectate pentru a realiza anumite scopuri specifice. Perioada de timp necesară pentru a realiza roboți apropiați vederii umane este de domeniul speculațiilor. Trebuie totuşi subliniat faptul că, imitarea naturii nu este întotdeauna unica soluție de rezolvare a unei probleme.

Procesarea imaginilor tratează probleme legate de segmentare, descriere, şi recunoaştere a obiectelor individuale. Aceste noțiuni traversează o largă varietate de abordări care sunt foarte bine fundamentate pe concepte analitice.

Materialul prezentat în această teză prezintă o largă varietate de noțiuni din domeniul vederii artificiale, cu orientare spre tehnici aplicabile în vederea artificială.

Cele trei tehnici care se vor prezenta în cadrul acestei părți realizează:

- în faza de segmentare extragerea obiectelor din cadrul unei imagini
- în faza de descriere transpunerea caracteristicilor obiectelor într-o formă prelucrabilă pe calculator
- în faza de recunoaştere compararea descriptorilor

2.1 Segmentare

Segmentarea este procesul care divide imaginea recepționată în părțile ei constituente sau în obiecte. Segmentarea reprezintă unul dintre cele mai importante elemente ale unui sistem automat de vedere pentru că în această etapă de procesare obiectele sunt extrase din imagine pentru fazele următoare de recunoaştere şi analiză.

Algoritmii de segmentare sunt în general bazați pe unul din cele două principii de bază : discontinuitatea şi similaritatea. Principala metodă de abordare din cadrul primei categorii este detecția de contur în timp ce, în cea de-a două categorie, baza o constituie stabilirea de praguri şi creşterea regiunilor.

Aceste concepte sunt aplicabile atât imaginilor statice cât şi celor dinamice. În acest ultim caz, mişcarea poate fi folosită ca un element esențial pentru a îmbunătăți performanțele algoritmilor de segmentare.

2.1.1 Conectarea marginilor şi detecția de contur

Tehnicile de detecție de contur sesizează discontinuitățile de intensitate. În cazul ideal, aceste tehnici vor scoate în evidență doar pixelii ce se află pe marginea dintre obiecte şi fond. În realitate, această mulțime de pixeli rareori vor caracteriza un contur complet din cauza zgomotului, a găurilor din contur cauzate de iluminarea neuniformă, şi a altor efecte care introduc discontinuități false de intensitate. Astfel, algoritmii de detecție de contur sunt urmați în mod normal de conectare şi de alte proceduri de detecție de contur proiectate să asambleze pixelii de pe un contur într-un set inteligibil de contururi ale obiectelor. Se vor prezenta în cele ce urmează mai multe tehnici adecvate acestui scop.

- **Analiza locală**

Una dintre cele mai simple metode folosite pentru conectarea punctelor de contur este analizarea caracteristicilor pixelilor aflați într-o mică vecinătate (de exemplu 3x3 sau 5x5) din jurul fiecărui punct (x,y) dintr-o imagine care a fost supusă

unui proces de detecție de contur. Toate punctele care sunt similare sunt conectate, formând în felul acesta un contur de pixeli care au anumite proprietăți comune.

Există două proprietăți principale care se folosesc pentru a determina similitudini între pixelii de contur în cadrul acestui tip de analiză :

1. mărimea răspunsului operatorului gradient folosit pentru a produce un pixel de contur
2. direcția gradientului.

Prima proprietate este dată de valoarea lui G[f(x,y)]. Astfel, se spune că un pixel de contur având coordonatele (x',y') şi făcând parte dintr-o vecinătate a lui (x,y) este similar în modul cu pixelul de la coordonatele (x,y) dacă

$$\left|G\left[f(x,y)\right] - G\left[f(x',y')\right]\right| \leq T \tag{2.1.1}$$

unde T este pragul.

Direcția gradientului poate fi stabilită prin unghiul vectorului gradient :

$$\theta = arctg\left(\frac{G_y}{G_x}\right) \tag{2.1.2}$$

Astfel, se poate spune că un pixel de contur la coordonatele (x',y') într-o vecinătate predefinită a lui (x,y) are un unghi similar cu al pixelului de coordonate (x,y) dacă

$$|\theta - \theta'| < A \tag{2.1.3}$$

unde A este un unghi de prag.

Este de remarcat faptul că direcția conturului la coordonatele (x,y) este perpendiculară pe direcția vectorului gradient în acel punct. Cu toate acestea, în scopul comparării a direcțiilor, se utilizează relația (2.1.3) cu rezultate similare.

Pe baza conceptelor prezentate, se conectează un punct dintr-o vecinătate predefinită a lui (x,y) cu pixelul de coordonate (x,y) dacă atât criteriile de mărime cât şi de direcție sunt satisfăcute. Acest proces se repetă pentru fiecare locație din cadrul imaginii, creând o listă de puncte conectate pe măsură ce vecinătatea este deplasată din pixel în pixel. O procedură simplă va asigura un nivel de gri diferit pentru fiecare set de pixeli de contur conectați.

- **Analiza globală cu ajutorul transformatei Hough**

Se pune problema conectării punctelor de contur pentru a determina dacă ele se află sau nu pe o curbă de o formă dată. Se presupune inițial că, fiind date *n* puncte

în planul xOy al unei imagini, se urmăreşte determinarea mulţimii de puncte care aparţin unor linii drepte. Una din posibilele soluţii este ca, în prima fază, să se determine toate liniile pe care le generează fiecare pereche de puncte şi după aceea să se găsească toate mulţimile de puncte care sunt apropiate liniilor particulare. Problema care apare în această procedură este aceea că presupune găsirea a $n(n-1)/2 \approx n^2$ linii şi efectuarea a $n[n(n-1)]/2 \approx n^3$ comparaţii a fiecărui punct cu toate liniile.

Aceeaşi problemă poate fi privită şi dintr-un alt punct de vedere folosind o abordare cunoscută ca şi ***transformarea Hough*** [86], [101].

Se consideră un punct (x_i,y_i) şi ecuaţia generală a unei linii drepte $y_i = ax_i + b$. Există o infinitate de linii ce trec prin (x_i,y_i), care toate satisfac ecuaţia $y_i = ax_i + b$ pentru valori diferite ale lui a şi b. Dacă însă se scrie ecuaţia în forma $b = -x_i a + y_i$, şi se consideră planul ab (numit şi ***spaţiul parametrilor***), atunci se obţine ecuaţia unei singure linii pentru o pereche fixă (x_i,y_i).

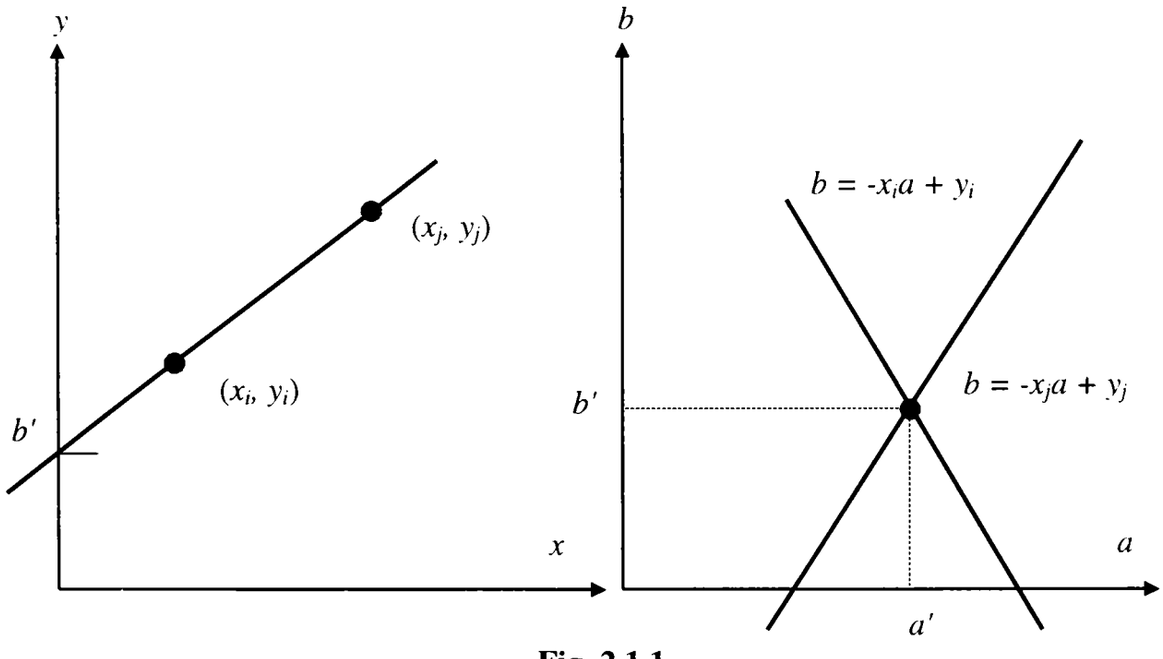


Fig. 2.1.1

(a) (a) planul xy (b)

(b) spaţiul parametrilor

Mai mult, un al doilea punct (x_j,y_j) va avea de asemenea o linie în spaţiul parametrilor asociată lui. Această linie va intersecta linia asociată punctului (x_i,y_i) la coordonatele (a',b') unde a' şi b' sunt coeficienţii dreptei care conţine atât pe (x_i,y_i) cât şi pe (x_j,y_j) în planul xy. De fapt, tuturor punctelor conţinute în linia respectivă le corespund linii în spaţiul parametrilor cu intersecţia la (a',b') (figura 2.1.1).

Din punct de vedere al calculului atractivitatea transformatei Hough constă în fragmentarea spațiului parametrilor în așa numitele ***celule acumulator***, așa cum este prezentat în figura 2.1.2. În aceasta figură (a_{max},a_{min}) și (b_{max},b_{min}) reprezintă ***domeniul coeficienților***. Celulele acumulator $A(i, j)$ corespund pătratului asociat cu coordonatele spațiului parametrilor (a_i,b_i).

Inițial, aceste celule sunt setate la zero. După aceea, pentru fiecare punct (x_k,y_k) din planul imaginii, se lasă parametrul a să ia toate valorile permise de subdiviziunile axei a și se determină toate b-urile corespunzătoare din ecuația $b = -x_k a + y_k$.. Valorile b rezultate sunt ulterior rotunjite prin trunchiere până la cea mai apropiată valoare pe axa b. Dacă alegerea lui a_p generează soluția b_q, i se va atribui lui $A(p,q) = A(p,q) + 1$.

La sfârșitul acestei proceduri, o valoare M în celula $A(i, j)$ corespunde la M puncte în planul xOy care se află pe linia $y = a_i x + b_j$. Acuratețea coliniarității acestor puncte este determinată de numărul de subdiviziuni din planul ab.

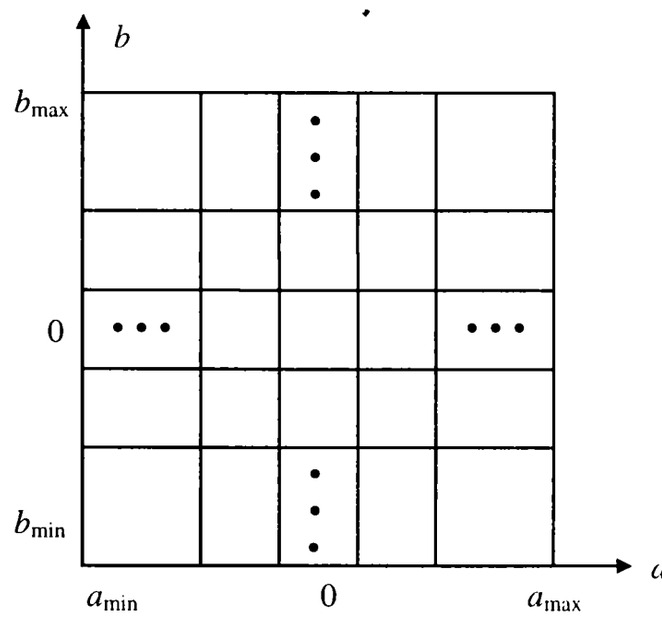


Fig. 2.1.2
Cuantizarea planului parametrilor în celule
pentru folosirea în transformata Hough

Se poate observa că dacă se împarte axa a în K părți, atunci pentru fiecare punct (x_k, y_k) se obțin K valori ale lui b corespunzând celor K valori posibile ale lui a. Deoarece sunt n puncte de imagine, aceasta presupune nK calcule. Astfel, procedura discutată este liniară în funcție de n.

Una din problemele care apar la folosirea formei $y = ax + b$ pentru ecuația dreptei, este aceea că dacă aceasta tinde spre verticală atât a cât și b vor tinde la infinit. Una din modalitățile de a evita această situație este folosirea reprezentării dreptei sub forma:

$$x\cos\theta + y\sin\theta = \rho \tag{2.1.4}$$

Parametrii folosiți în ecuația (2.1.4) sunt prezentați în figura 2.1.3,a. Utilizarea acestei reprezentări în construcția tabelei de acumulatori este identică cu metoda deja prezentată; unica diferență este aceea că, în loc de linii drepte, există curbe sinusoidale în planul (θ, ρ). Ca și anterior, M puncte coliniare ce se află pe linia $x\cos\theta_i + y\sin\theta_i = \rho_i$ vor genera M curbe sinusoidale care intersectează la (θ_i, ρ_j) spațiul parametrilor. Dacă se folosește metoda incrementării lui θ și se determină ρ, procedura va genera M intrări în acumulatorul $A(i, j)$ asociat cu celula determinată de (θ_i, ρ_j). Împărțirea spațiului parametrilor este prezentată în figura 2.1.3,b.

Deși s-au urmărit numai liniile drepte, transformata Hough se poate aplica oricăror funcții de forma g(x, c) = 0, unde x este o matrice de coordonate iar c este o matrice de coeficienți.

De exemplu, locul geometric al punctelor ce se află pe un cerc este dat de relația :

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \tag{2.1.5}$$

și poate fi foarte ușor detectat folosind abordarea prezentată. Diferența esențială este aceea că, acum există trei parametri, c_1, c_2, și c_3, ceea ce va conduce într-un spațiu tridimensional al parametrilor la celule în formă de cuburi și acumulatori de forma $A(i, j, k)$. Procedura se bazează pe incrementarea lui c_1 și c_2, determinându-l pe c_3 care satisface ecuația (2.1.5), și reactualizarea acumulatorului corespunzător celulei asociate cu tripletul (c_1, c_2, c_3).

Se poate observa faptul că complexitatea transformatei Hough este dependentă de numărul de coordonate și coeficienți din reprezentarea funcției.

Se mai precizează că există posibilitatea unei generalizări mai avansate a transformatei Hough pentru a detecta curbe cu o reprezentare analitică complicată.

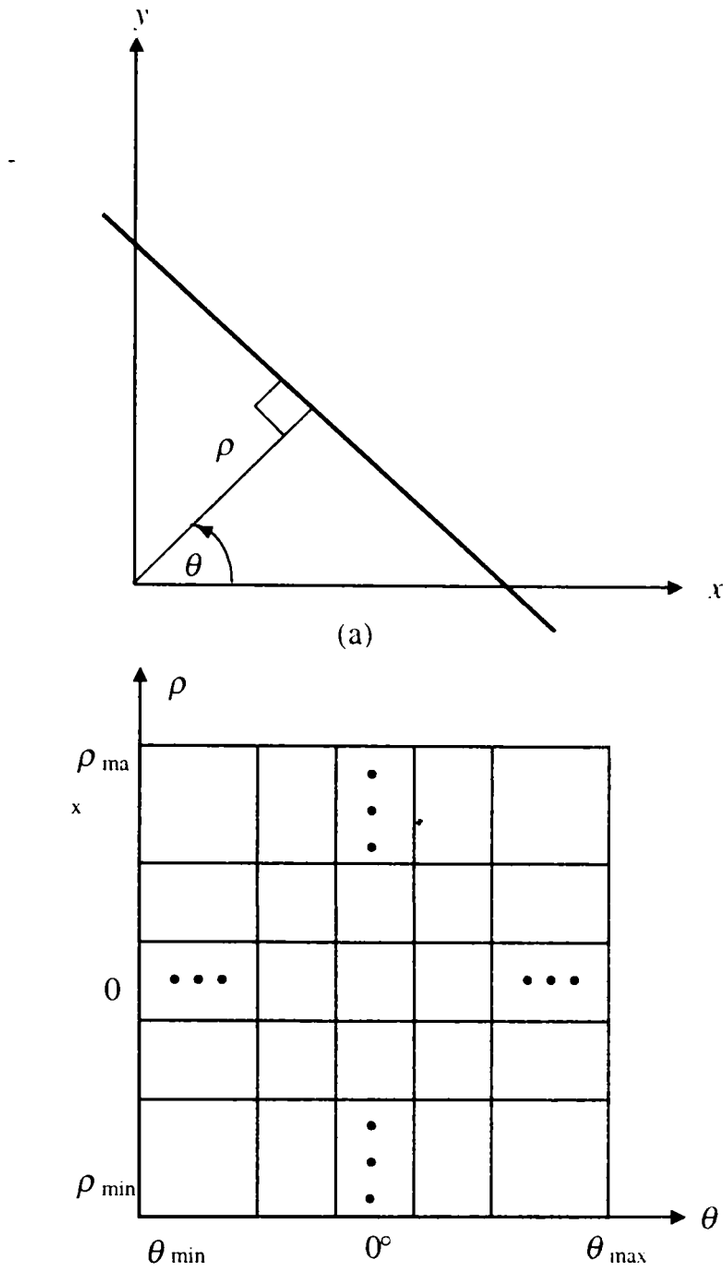


Fig. 2.1.3
(a) Reprezentarea normală a unei linii
(b) Cuantizarea planului $\theta\rho$ în celule

- **Analiza globală cu ajutorul tehnicilor teoriei grafurilor**

Metoda discutată anterior se bazează pe deținerea unui set de puncte de contur obținute în mod normal în urma unei operații de tip gradient. Deoarece gradientul este o derivată, el va evidenția variațiile puternice de intensitate, astfel că, este rar folosit în faza de preprocesare în situații caracterizate prin zgomot mare. Se poate realiza însă şi o abordare globală bazată pe reprezentarea segmentelor de contur în forma unei

structuri de graf şi pe căutarea drumului minim care corespunde segmentelor semnificative. Această reprezentare este grosolană, dar eficientă în prezența zgomotului.

Pentru prezentarea algoritmului sunt necesare câteva definiții de bază [10], [101], [132], [145], [148]. Un ***graf*** $G = (N,A)$ este o mulțime finită, nevidă de noduri N, împreună cu o mulțime A de perechi neordonate de elemente distincte ale lui N. Fiecare pereche (n_i,n_j) ale lui A este numită ***arc***. Un graf ale cărui arce sunt orientate este numit ***graf orientat***. Dacă un arc este orientat de la nodul n_i la nodul n_j, atunci spunem că n_j este ***succesor*** **al** ***părintelui*** său, n_i. Procesul prin care se identifică succesorii unui nod este denumit ***expansiunea*** nodului. În fiecare graf se definesc ***nivele***, cum ar fi nivelul 0 care conține un singur nod, numit ***nod de start***, iar nodurile din ultimul nivel sunt numite ***noduri țintă***. ***Costul*** $c(n_i,n_j)$ poate fi asociat fiecărui arc (n_i,n_j). O secvență de noduri $n_1, n_2,...,n_k$ în care fiecare nod n_i este succesor al nodului n_{i-1} este numită ***drumul*** de la n_1 la n_k, şi costul drumului este dat de relația :

$$c = \sum_{i=2}^{k} c(n_{i-1}, n_i) \tag{2.1.6}$$

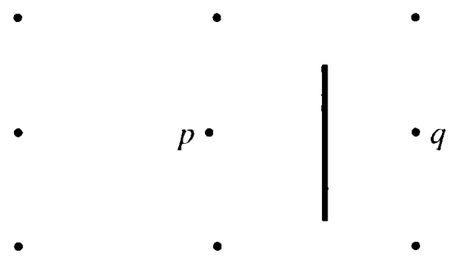


Fig. 2.1.4
Element de contur dintre pixelii p şi q

În final, se definește un ***element de contur*** ca graniță dintre doi pixeli p şi q, în așa fel încât p şi q sunt vecini de tipul 4, (figura 2.1.4). În acest context un ***contur***, este o secvență de elemente de contur.

În scopul de a ilustra cum funcționează aceste concepte pentru detecția de contur, se ia în considerare imaginea 3x3 prezentată în figura 2.1.5, unde numerele de pe margine sunt coordonate de pixel iar numerele din paranteze reprezintă intensitatea. Fiecărui element de contur definit de pixelii p şi q i se va asocia un cost exprimat de relația :

$$c(p,q) = H - [f(p) - f(q)] \tag{2.1.7}$$

	0	1	2
0	• (7)	• (2)	• (2)
1	• (5)	• (7)	• (2)
2	• (5)	• (1)	• (0)

Fig. 2.1.5
O imagine 3x3

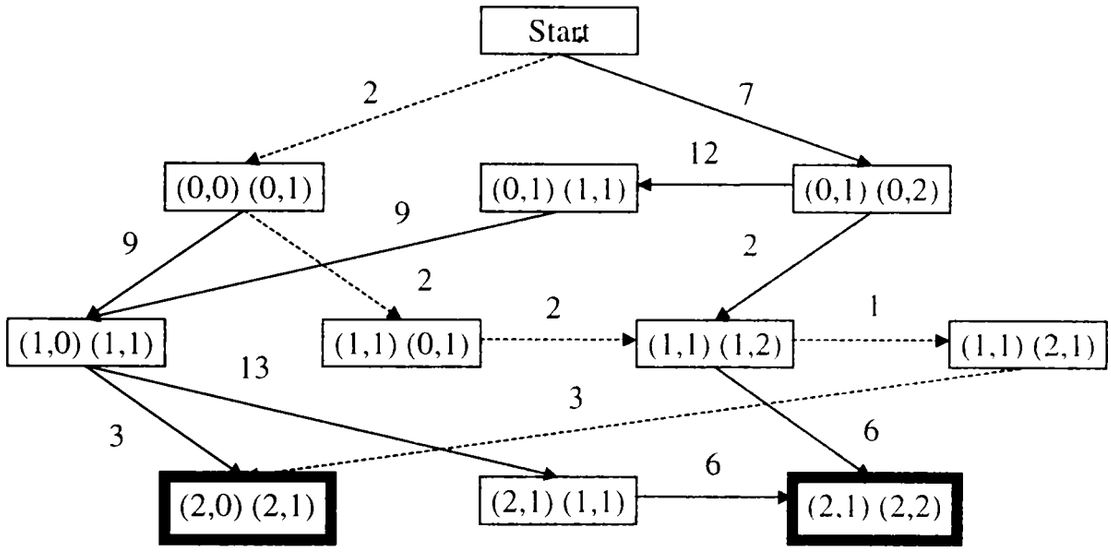


Fig. 2.1.6
Graful folosit pentru determinarea unui contur în imaginea din figura 2.1.5
Perechile (a, b) (c, d) din fiecare căsuță se referă la punctele p respectiv q.

unde :

- H este cea mai mare valoare de intensitate din imagine (7 în acest exemplu),
- $f(p)$ este valoarea intensității lui p
- $f(q)$ este valoarea intensității în q.

Aşa cum s-a precizat mai sus p şi q sunt vecini de tipul 4.

Graful pentru această aplicație este prezentat în figura 2.1.6.

Fiecare nod din graf corespunde unui element de contur. Un arc există între două noduri dacă două elemente de contur succesive sunt parte dintr-un contur. Costul fiecărui element de contur, calculat cu relația (2.1.7), este reprezentat de un arc ce ajunge la el, iar nodurile țintă sunt desenate în dreptunghiuri ingroşate. Fiecare drum de la nodul de start până la nodul țintă este un posibil contur.

Pentru simplificare, se presupune că se începe conturul în rândul de sus şi se termină în ultimul rând, astfel că primul element de contur poate fi doar [(0,0), (0,1)] sau [(0,1), (0,2)] iar ultimul element [(2,0), (2,1)] sau [(2,1), (2,2)]. Drumul cu cost minim, calculat folosind relația (2.1.6), este reprezentat punctat în figura 2.1.6, iar conturul corespunzător este desenat în figura 2.1.7.

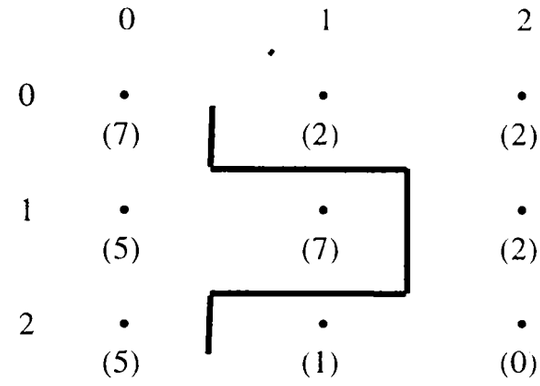


Fig. 2.1.7
Conturul corespunzător drumului cu cost minim

În general, problema găsirii drumului cu cost minim nu este una neînsemnată din punct de vedere al calculului. În mod normal, algoritmul folosit sacrifică varianta optimă pentru a câştiga în viteză, iar cel prezentat în continuare este reprezentativ pentru o clasă de proceduri care folosesc metode euristice în scopul de a reduce efortul de căutare.

Fie $r(n)$ o estimare a costului drumului minim de la nodul de start s până la un nod țintă, unde drumul este constrâns să treacă prin nodul n. Acest cost poate fi exprimat ca estimare a costului drumului minim de la s la n, plus o estimare a costului drumului de la n la un nod țintă :

$$r(n) = g(n) + h(n) \tag{2.1.8}$$

unde :

- $g(n)$ poate fi ales ca cel mai mic cost al unui drum de la s la n găsit până în această fază
- $h(n)$ este obținut folosind orice informație euristică disponibilă (de exemplu : expandând numai anumite noduri pe baza costurilor găsite anterior în încercarea de a ajunge la acel nod).

Un algoritm care foloseşte $r(n)$ ca pe o bază pentru realizarea căutării într-un graf are următorii paşi :

1. Marchează nodul de start DESCHIS şi atribuie lui $g(s) = 0$.
2. Dacă nodul este DESCHIS, ieşi cu eroare; în caz contrar continuă.
3. Marchează ca ÎNCHIS nodul DESCHIS a cărui estimare $r(n)$ calculată cu ecuația (2.1.8) este mai mică. (Egalitățile în cazul a două valori minime ale lui r sunt rezolvate arbitrar dar întotdeauna în favoarea unui nod țintă).
4. Dacă n este un nod țintă, ieşi cu soluția drumului obținută parcurgând lista înapoi prin pointeri; în caz contrar continuă.
5. Expandează nodul n, generând toți succesorii lui. (Dacă nu are succesori, du-te la pasul 2).
6. Dacă un succesor n_i nu este marcat, atribuie :

$$r(n_i) = g(n) + c(n, n_i)$$

marchează-l DESCHIS, şi direcționează pointerii de la el înapoi la n.

7. Dacă un succesor n_i este marcat ÎNCHIS sau DESCHIS, reactualizează-i valoarea astfel

$$g'(n_i) = \min\left[g(n_i)\right], g(n) + c(n, n_i)$$

Marchează DESCHIS acei succesori ÎNCHIŞI ale căror valori g' sunt cele mai mici şi redirecționează către n toți pointerii pentru toate nodurile ale căror valori g' sunt mici. Du-te la pasul 2.

În general, acest algoritm nu garantează găsirea drumului minim; avantajul lui este viteza datorită folosirii tehnicii euristice.

Se poate demonstra, totuşi, că dacă $h(n)$ este un cost cu o aproximație bună a costului drumului minim, atunci procedura va găsi un drum optim către țintă.

2.1.2 Folosirea pragului pentru segmentarea imaginii

Conceptul de prag a fost introdus ca o operație care presupune un test cu o funcție de forma :

$$T = T\left[x, y, p(x, y), f(x, y)\right] \tag{2.1.9}$$

unde :

- $f(x,y)$ este intensitatea punctului (x,y)
- $p(x,y)$ denotă unele proprietăți locale măsurate în vecinătatea acestui punct.

O imagine generată cu un prag, $g(x,y)$ este creată cu ajutorul următoarei definiții

$$g(x, y) = \begin{cases} 1 & dacă \quad f(x, y) > T \\ 0 & dacă \quad f(x, y) \le T \end{cases} \tag{2.1.10}$$

în așa fel încât pixelii din $g(x,y)$ notați cu 1 corespund obiectelor, în timp ce pixelii notați cu 0 corespund fondului. Ecuația (2.1.10) presupune că intensitatea obiectelor este mai mare decât intensitatea fondului. Condiția opusă este tratată de aceeaşi relație prin inversarea sensului inegalităților.

➢ **Praguri globale şi locale**

Când funcția T din ecuația (2.1.9) depinde doar de $f(x,y)$, pragul folosit se numeşte ***prag global***. Dacă T depinde atât de $f(x,y)$ cât şi de $p(x,y)$, atunci pragul este numit ***prag local***. Dacă în plus, T depinde şi de coordonatele spațiale x şi y, pragul se numeşte ***prag dinamic***.

Pragurile globale se folosesc când o aplicație este în situația de a prelucra o imagine care are o definiție clară între obiecte şi fond, şi unde iluminarea este relativ uniformă. Tehnicile de iluminare din spate sau iluminare structurată generează de obicei imagini care pot fi segmentate cu praguri globale.

Pentru cele mai multe cazuri, însă, iluminarea arbitrară a mediului de lucru generează imagini, care dacă sunt tratate prin praguri, vor necesita anumite tipuri de analize locale pentru a compensa efecte cum ar fi neuniformități de iluminare, umbre, sau reflecții.

➢ **Selectarea pragului optim**

Deseori este posibil să se trateze o histogramă ca fiind formată din suma funcțiilor de densitate de probabilitate. În cazul unei histograme bimodale funcția care aproximează histograma este dată de

$$p(z) = P_1 p_1(z) + P_2 p_2(z) \tag{2.1.11}$$

unde:

- z este o variabilă aleatoare ce reprezintă intensitatea
- $p_1(z)$ şi $p_2(z)$ sunt funcţiile de densitate probabilistică
- P_1 şi P_2 sunt numite probabilităţi presupuse.

Aceste ultime valori sunt simple probabilităţi de apariţie a două tipuri de nivele de intensitate într-o imagine.

De exemplu, se consideră o imagine a cărei histogramă este prezentată în figura 2.1.8,a. Histograma acoperitoare poate fi aproximată de suma a două funcţii de densitate de probabilitate (figura 2.1.8,b). Dacă se cunoaşte faptul că pixelii luminoşi reprezintă obiecte şi că 20% din suprafaţa imaginii este ocupată de pixelii obiectelor, atunci $P_1 = 0.2$. Este necesar ca

$$P_1 + P_2 = 1 \tag{2.1.12}$$

ceea ce indică faptul că, ceilalţi 80% sunt pixeli de fond.

Se definesc două funcţii de z :

$$d_1(z) = P_1 p_1(z) \tag{2.1.13}$$

şi

$$d_2(z) = P_2 p_2(z) \tag{2.1.14}$$

Eroarea medie asumată în procesul de calificare a unui pixel obiect ca şi pixel de fond, sau invers, este minimizată prin folosirea următoarei reguli: fiind dat un pixel cu valoarea intensităţii z, se înlocuieşte această valoare în ecuaţiile (2.1.13) şi (2.1.14). Apoi, se clasifică pixelul ca un pixel obiect dacă $d_1(z) > d_2(z)$ sau ca un pixel de fond dacă relaţia se inversează, adică $d_2(z) > d_1(z)$.

Pragul optim este dat de valoarea lui z pentru care există relaţia $d_1(z) = d_2(z)$. Astfel, dacă se atribuie $z = T$ în ecuaţiile (2.1.13) şi (2.1.14), se determină pragul optim care satisface ecuaţia :

$$P_1 p_1(T) = P_2 p_2(T) \tag{2.1.15}$$

Dacă sunt cunoscute formele funcţiilor $p_1(z)$ şi $p_2(z)$ se poate folosi această relaţie pentru a determina pragul optim care separă obiectele de fond. Odată ce acest prag a fost determinat, ecuaţia (2.1.10) poate fi folosită pentru segmentarea unei imagini date.

Se poate utiliza ecuaţia (2.1.15), presupunând că $p_1(z)$ şi $p_2(z)$ sunt funcţiile lui Gauss de densitate a probabilităţii, adică:

$$p_1(z)=\frac{1}{\sqrt{2\pi\sigma_1}}e^{\left[\frac{(z-m_1)^2}{2\sigma_1^2}\right]} \qquad (2.1.16)$$

şi

$$p_2(z)=\frac{1}{\sqrt{2\pi\sigma_2}}e^{\left[\frac{(z-m_2)^2}{2\sigma_2^2}\right]} \qquad (2.1.17)$$

unde:

- m_1 şi m_2 sunt mediile
- σ_1 şi σ_2 sunt abaterile medii pătratice

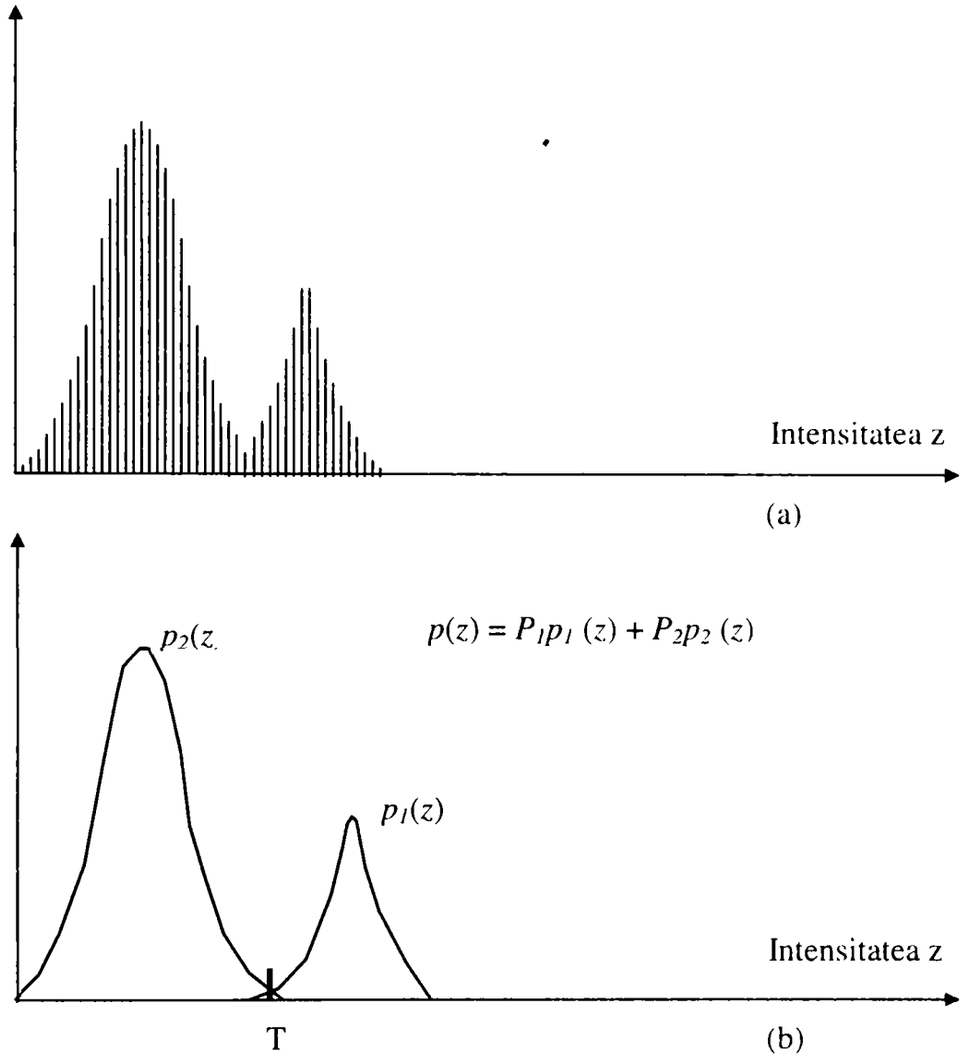


Fig. 2.1.8

(a) Histogramă de intensitate

(b) Aproximarea ca sumă a două funcții de densitate probabilistică

Se face atribuirea $z = T$ în aceste expresii, care înlocuite în ecuația (2.1.15), după simplificare va conduce la :

$$AT^2 + BT + C = 0 \tag{2.1.18}$$

unde :

$$A = \sigma_1^2 - \sigma_2^2$$

$$B = 2\left(m_1\sigma_2^2 - m_2\sigma_1^2\right) \tag{2.1.19}$$

$$C = \sigma_1^2 m_2^2 - \sigma_2^2 m_1^2 + 2\sigma_1^2\sigma_2^2 \ln\frac{\sigma_2 P_1}{\sigma_1 P_2}$$

Posibilitatea existenței a două soluții indică faptul că pot fi necesare două valori de prag pentru obținerea unei soluții optime.

Dacă deviațiile standard sunt egale, $\sigma_1 = \sigma_2 = \sigma$, o singură valoare de prag este suficientă

$$T = \frac{m_1 + m_2}{2} + \frac{\sigma^2}{m_1 - m_2}\ln\frac{P_2}{P_1} \tag{2.1.20}$$

Dacă $\sigma = 0$ sau $P_1 = P_2$, pragul optim este media aritmetică a mediilor m_1 şi m_2. Prima condiție înseamnă doar că atât intensitățile obiectelor cât şi ale fondului sunt constante peste întreaga imagine. A doua condiție înseamnă că pixelii obiectelor şi ai fondului au aceeaşi probabilitate de apariție, condiție întâlnită de fiecare dată când numărul pixelilor obiectelor din imagine este egal cu cel al pixelilor fondului.

Abordarea prezentată este aplicabilă şi în selecția pragurilor multiple. Se presupune că se poate modela o histogramă multimodală ca sumă a n funcții de densitate de probabilitate după cum urmează :

$$p(z) = P_1 p_1(z) + \ldots + P_n p_n(z) \tag{2.1.21}$$

După aceasta, problema pragului optim poate fi abordată ca o clasificare a unui pixel dat ca aparținând uneia din cele n categorii posibile. Decizia cu eroare minimă se bazează în acest caz pe n funcții de forma :

$$d_i(z) = P_i p_i(z) \qquad i = 1, 2, \ldots, n \tag{2.1.22}$$

Un pixel dat cu intensitatea z este atribuit categoriei k dacă $d_k(z) > d_j(z)$, $j = 1, 2, \ldots, n$; $j \neq k$. Pragul minim între categoria k şi categoria j, notat prin T_{kj}, este obținut rezolvând ecuația :

$$P_k p_k\left(T_{kj}\right) = P_j p_j\left(T_{kj}\right) \tag{2.1.23}$$

Cu toate acestea problema reală în folosirea histogramelor cu praguri multiple este determinarea sensului modurilor histogramei.

➢ **Selectarea pragului pe baza caracteristicilor de margine**

Una din cele mai importante aspecte în selectarea pragului este capabilitatea de a identifica vârfurile modurilor dintr-o histogramă dată. Acest lucru este important în selectarea automată a pragului în situații în care caracteristicile imaginii aparțin unei game largi de distribuții ale intensității. Şansa de a selecta un prag bun creşte considerabil dacă vârfurile histogramei sunt înalte, ascuțite, simetrice, şi separate prin văi adânci.

O modalitate de a îmbunătăți forma unei histograme este aceea de a considera doar acei pixeli care se află pe sau în jurul graniței dintre obiecte şi fond.

O îmbunătățire imediată şi evidentă este aceea că face histograma mai puțin dependentă de dimensiunea relativă între obiecte şi fond. De exemplu, histograma intensității unei imagini compuse dintr-un fond mare şi cu intensitate aproape constantă şi un obiect mic va fi dominată de un vârf mare datorat concentrației pixelilor de fond.

Pe de altă parte, dacă sunt folosiți doar pixelii aflați pe sau în jurul graniței dintre obiecte şi fond, histograma rezultată va avea vârfuri ale căror înălțimi sunt mai echilibrate. În plus, probabilitatea ca un pixel dat să fie aproape de marginea unui obiect este practic egală cu probabilitatea ca el să se afle pe marginea fondului.

Funcțiile gradient, $G[f(x,y)]$ şi Laplacian $L[f(x,y)]$, în fiecare punct din imagine se pot folosi pentru a forma o imagine pe trei nivele, astfel :

$$s(x,y)=\begin{cases}0 & dacă \quad G[f(x,y)]<T \\ + & dacă \quad G[f(x,y)]\geq T \quad şi \quad L[f(x,y)]\geq 0 \\ - & dacă \quad G[f(x,y)]\geq T \quad şi \quad L[f(x,y)]<0\end{cases} \qquad (2.1.24)$$

unde : simbolurile 0, +, - reprezintă oricare trei nivele distincte de gri, iar T este pragul.

Presupunând un obiect întunecat pe un fond luminos, şi făcând referire la figura 1.6.1,b, folosirea ecuației (2.1.24) produce o imagine $s(x,y)$ în care toți pixelii care nu sunt pe o margine (aşa cum îi determină $G[f(x,y)]$ ca fiind mai mici decât pragul T) sunt marcați "0", toți pixelii de pe partea întunecată a conturului sunt marcați cu "+", iar toți pixelii de pe partea luminoasă a conturului sunt marcați cu "-".

Simbolurile + şi - din ecuaţia (2.1.24) sunt inversate pentru un obiect luminos aflat pe un fond întunecat.

Informaţia obţinută folosind această procedură poate fi folosită pentru a genera o imagine binară segmentată, în care 1 corespunde obiectelor de interes iar 0 corespunde fondului. În primul rând se remarcă faptul că tranziţia (de-a lungul unei linii orizontale sau verticale) de la un fond luminos la un obiect întunecat trebuie să fie caracterizată de apariţia unui - urmat de un + în $s(x,y)$. Interiorul obiectului este compus din pixeli care sunt marcaţi ori cu 0 ori cu +. În final, tranziţia înapoi de la obiect la fond este caracterizată de apariţia unui + urmat de un -.

Astfel se obţine o linie orizontală sau verticală conţinând o secţiune a unui obiect de următoarea formă:

(. . .) (-, +) (0 sau +) (+, -) (. . .)

unde (. . .) reprezintă orice combinaţie de +, -, sau 0. Parantezele interioare conţin pixeli ai obiectului şi vor fi marcaţi cu 1. Toţi ceilalţi pixeli aflaţi pe aceeaşi linie de scanare sunt marcaţi cu 0, cu excepţia oricărei alte secvenţe (0 sau +) mărginită de (-, +) şi (+, -).

- **Praguri bazate pe mai multe variabile**

În unele aplicaţii este posibilă folosirea mai multor variabile pentru a caracteriza fiecare pixel dintr-o imagine, în acest fel îmbunătăţind capacitatea de a diferenţia nu doar între obiecte şi fond, dar chiar de a face distincţie între obiecte [2], [12], [21], [90], [123], [157].

Un exemplu important este vederea în culori, unde componentele roşu, verde si albastru (RGB) sunt folosite pentru a forma o imagine compusă în culori. În acest caz, fiecare pixel este caracterizat de trei valori şi este posibil să se construiască o histogramă tridimensională.

Procedura de baza este aceeaşi ca şi acea folosită în cazul unei singure variabile. De exemplu, fiind date trei imagini pe 16 nivele corespunzând componentelor RGB a unui senzor de culoare, se formează o reţea 16x16x16 (un cub) şi se inserează în fiecare celulă a cubului numărul de pixeli a căror componente RGB au intensităţile corespunzătoare cu coordonatele care definesc locaţia celulei particulare. Fiecare intrare poate fi împărţită la numărul total de pixeli din imagine pentru a forma o histogramă normalizată.

Conceptul selecției pragului se transformă în descoperirea grupurilor de puncte din spațiul tridimensional, în care fiecare grup legat este similar cu un mod dominant într-o histogramă cu o singură variabilă.

Se presupune de exemplu, că s-au găsit două grupuri semnificative de puncte într-o histogramă dată, unde un grup corespunde obiectelor şi altul fondului. Ținând seama de faptul că fiecare pixel are trei componente şi, astfel, poate fi văzut ca un punct într-un spațiu tridimensional, se poate segmenta o imagine folosind următoarea procedură : pentru fiecare pixel din imagine se calculează distanța dintre acel pixel şi centrul fiecărui grup. Astfel, dacă pixelul este mai apropiat de centrul grupului de obiecte, va fi marcat cu 1; altfel, va fi marcat cu 0.

Acest concept este foarte uşor de extins la mai multe componente de pixel şi, normal la mai multe grupuri. Dificultatea principală este depistarea grupurilor semnificative deoarece pe măsură ce creşte numărul de variabile creşte şi această procedură.

2.1.3 Segmentarea orientată pe regiuni

Obiectivul segmentării este împărțirea imaginii în regiuni. Se poate realiza acest obiectiv prin găsirea de margini între regiuni pe baza discontinuităților de intensitate, iar ulterior segmentarea se realizează prin praguri pe baza distribuției proprietăților pixelilor (intensitatea sau culoarea).

Se poate realiza segmentarea şi prin descoperirea directă a regiunilor.

Fie R întreaga regiune a imaginii. Se imaginează procesul de segmentare ca o procedură ce împarte regiunea R în n subregiuni, R_1, R_2,...,R_n, în aşa fel încât :

1. $\bigcup_{i=1}^{n} R_i = R$
2. R_i este o regiune conexă, i = 1, 2, ..., n
3. $R_i \cap R_j = \varnothing$ pentru toți i şi j, $i \neq j$
4. $P(R_i)$ = TRUE pentru i = 1, 2, ..., n
5. $P(R_i \cup R_j)$ = FALSE pentru $i \neq j$

unde :

- $P(R_i)$ este predicatul logic definit peste punctele din mulțimea R_i
- $\varnothing$ este mulțimea vidă

Condiția 1 indică faptul că segmentarea trebuie să fie completă; adică oricare pixel trebuie să fie într-o regiune. A doua condiție cere ca punctele dintr-o regiune să fie conexe. Condiția 3 indică faptul că regiunile trebuie să fie disjuncte. Condiția 4 impune proprietățile care trebuie satisfăcute de pixelii dintr-o regiune segmentată. Ultima condiție indică faptul că regiunile R_i și R_j sunt diferite în sensul predicatului P. Utilizarea acestor condiții în algoritmii de segmentare este discutată în subcapitolele următoare.

- **Creşterea regiunilor prin adăugarea de pixeli**

Creşterea regiunilor este o procedură care grupează pixelii sau subregiunile în regiuni mai mari [15], [19], [22], [104], [111], [133],[140]. Cea mai simplă dintre aceste abordări este ***adăugarea de pixeli***, unde se porneşte cu o mulțime de (nuclee) de puncte şi din acestea se cresc regiunile prin adăugarea la fiecare punct ***nucleu*** a acelor pixeli vecini care au proprietăți similare (intensitate, culoare, textură, etc.).

Ca o ilustrare simplă a acestei proceduri se consideră figura 2.1.9,a, unde numerele din interiorul celulelor reprezintă valori de intensități.

Se presupune că punctele de coordonate (3,2) şi (3,4) sunt folosite ca şi nuclee. Folosind două puncte de start se realizează o segmentare ce va conține cel mult doua regiuni : R_1 asociată nucleului (3,2) şi R_2 asociată nucleului (3,4).

Proprietatea P care se va utiliza pentru a include un pixel într-una dintre regiuni este aceea că diferența absolută dintre intensitatea pixelului şi intensitatea nucleului să fie mai mică decât pragul T (orice pixel ce satisface simultan această proprietate pentru amândouă nucleele este asociat arbitrar în regiunea R_1). Rezultatul obținut folosind $T = 3$ este prezentat in figura 2.1.9,b. În acest caz segmentarea constă din două regiuni, unde punctele din R_1 sunt marcate cu a iar punctele din R_2 sunt marcate cu b.

Este de remarcat faptul că oricare punct de start din aceste două regiuni ar fi generat acelaşi rezultat. Dacă, pe de altă parte, s-ar fi ales $T = 8$, ar fi rezultat o singură regiune, aşa cum se arată în figura 2.1.9,c.

0	0	5	6	7
1	1	5	8	7
0	1	5	7	7
2	0	7	6	6
0	1	5	6	5

(a)

a	a	b	B	b
a	a	b	B	b
a	a	b	B	b
a	a	b	B	b
a	a	b	B	b

(b)

a	a	a	A	a
a	a	a	A	a
a	a	a	A	a
a	a	a	A	a
a	a	a	A	a

(c)

Fig. 2.1.9

Exemplu de creştere a regiunilor

(a) Matricea originală a imaginii

(b) Segmentarea rezultată folosind o diferență mai mică decât 3 între nivelele de intensitate

(c) Rezultatul folosirii unei diferențe mai mici decât 8

Exemplul precedent, deşi este simplu, scoate în evidență unele dificultăți importante care apar în creşterea regiunilor. Acestea sunt :

- selectarea nucleelor inițiale care să reprezinte regiunile de interes
- selectarea unor proprietăți adecvate pentru includerea punctelor în diferitele regiuni pe parcursul procesului de creştere
- formularea unei reguli de oprire

Când o informație a priori nu este disponibilă, se poate proceda la calcularea în fiecare pixel a aceloraşi proprietăți care în final se vor folosi la atribuirea unui pixel la regiuni, în timpul procesului de creştere. Dacă rezultatul acestor calcule arată existența grupurilor de valori, atunci pixelii ale căror proprietăți îi plasează cel mai aproape de centrul grupurilor pot fi folosiți ca nuclee. În exemplul de mai sus, o

histogramă a intensităților va arăta faptul că punctele cu intensitatea 1 şi 7 sunt cele dominante.

Selecția de criterii similare este dependentă nu doar de problema tratată, dar şi de tipul de date de imagine disponibil. Din păcate, posibilitatea deținerii de date multispectrale sau de alte tipuri este mai degrabă o excepție decât o regulă în vederea artificială. În mod normal, analiza regiunilor trebuie să se desfăşoare folosind un set de descriptori bazați pe intensitate şi alte proprietăți spațiale (momente, textură, etc.) a unei singure surse de imagini [12], [14], [23], [89], [98], [137], [153].

Este important faptul că folosirea numai a descriptorilor poate duce la rezultate greşite dacă nu se foloseşte şi informația de conectivitate sau adiacență în procesul de creştere. Această observație se poate uşor vizualiza considerând un aranjament de pixeli cu doar trei valori de intensitate distinctă. Gruparea pixelilor cu aceeaşi intensitate pentru a forma o regiune, fără a ține seama de conectivitate va genera o segmentare care este total eronată.

În mod normal, se opreşte creşterea când nici un pixel nu mai satisface criteriile de includere în acea regiune.

S-au menționat doar criterii ca : intensitatea, textura, şi culoarea, care sunt locale şi nu țin seama de istoria creşterii regiunii. Criteriile adiționale care măresc puterea unui algoritm de creştere încorporează conceptul de dimensiune, asemănarea dintre un pixel candidat şi un pixel adăugat şi forma unei regiuni în creştere. Utilizarea acestor tipuri de descriptori este bazată pe presupunerea că se dispune parțial de un model a ceea ce ar trebui să se obțină.

- **Divizarea şi reuniunea regiunilor**

Se studiază creşterea regiunilor pornind de la o mulțime de puncte nucleu [1], [7],]17], [90], [112], [148]. O altă alternativă ar fi ca inițial să se împartă o imagine într-o mulțime de regiuni arbitrare şi disjuncte, iar ulterior să fie unite sau divizate într-o încercare de a satisface condițiile impuse inițial. Algoritmul de divizare şi reuniune care lucrează iterativ în scopul satisfacerii restricțiilor impuse poate fi explicat astfel :

Fie R întreaga regiune a imaginii, şi P un predicat selectat. Se presupune că există o imagine pătrată şi în acest caz, un procedeu de segmentare a lui R este divizarea lui în regiuni pătrate din ce în ce mai mici, în aşa fel încât pentru orice regiune R_i, $P(R_i)$ = TRUE. Procedura începe cu regiunea întreagă R. Dacă $P(R)$ = FALSE, se divizează imaginea în pătrate, şi aşa mai departe.

Această tehnică particulară de împărțire are o reprezentare adecvată în forma așa numitului ***arbore pătratic*** (un arbore în care fiecare nod are exact patru descendenți). O ilustrare simplă este prezentată în figura 2.1.10.

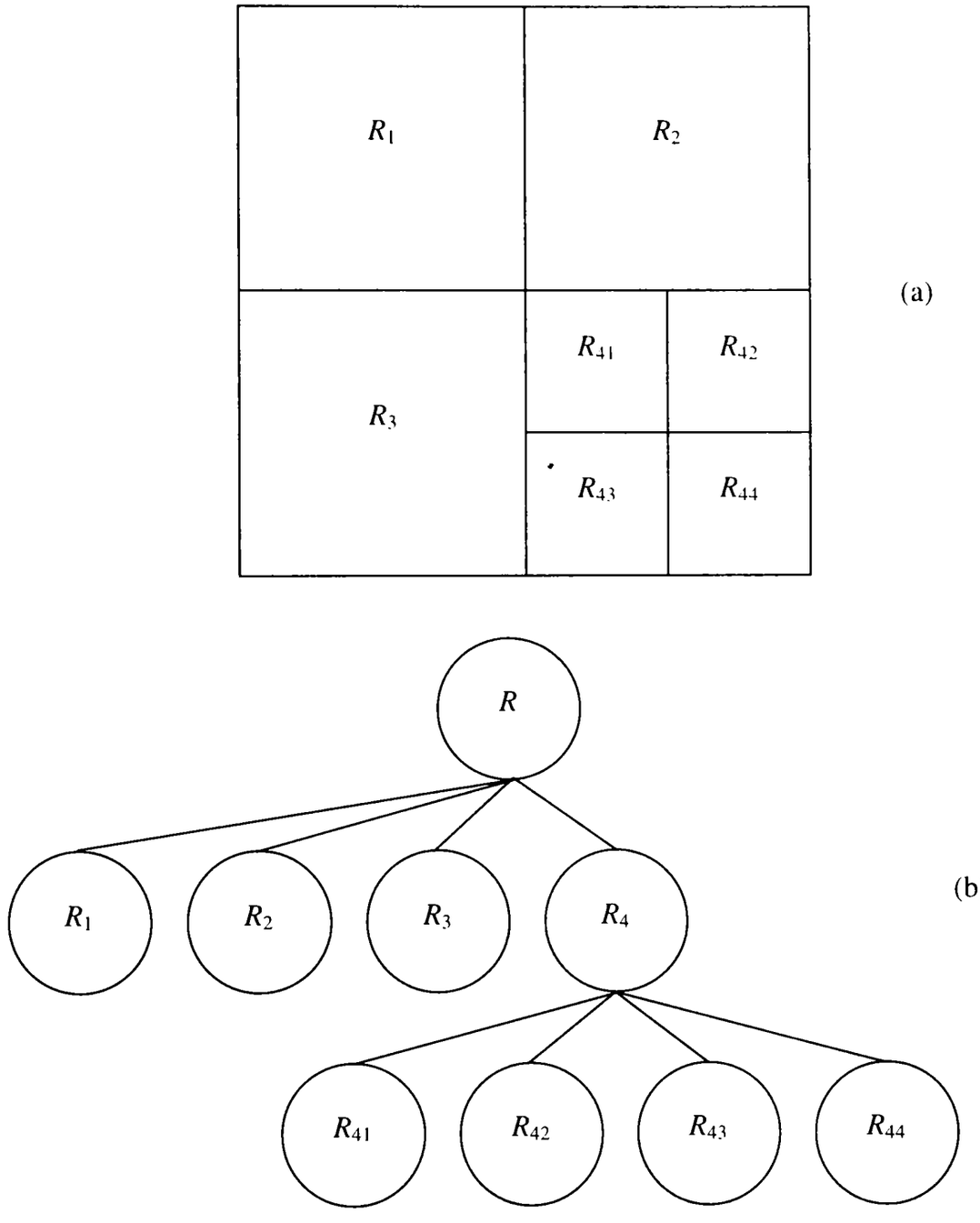


Fig. 2.1.10
(a) Imaginea partiționată
(b) Arborele pătratic corespunzător

Este de remarcat faptul că rădăcina arborelui corespunde cu întreaga imagine și fiecare nod corespunde unei divizări. În acest caz doar R_4 este divizat mai departe.

Dacă se folosesc doar divizări, este foarte probabil ca partiția finală să conțină regiuni adiacente cu proprietăți identice. Aceasta se poate remedia realizând

reuniunea, la fel ca şi în cazul divizării. În scopul satisfacerii condiţiilor de segmentare impuse anterior, se reunesc doar acele regiuni adiacente ale căror pixeli combinaţi satisfac predicatul P; adică, se reunesc două regiuni adiacente R_i şi R_k doar dacă $P(R_i \cup R_k) = \text{TRUE}$.

Rezultă următoarea procedură în care, la fiecare pas, se vor efectua :

1. Se divizează în patru pătrate disjuncte fiecare regiune R_i pentru care $P(R_i) = \text{FALSE}$
2. Se reunesc oricare două regiuni adiacente R_j şi R_k pentru care $P(R_j \cup R_{kl}) = \text{TRUE}$
3. Când nu se mai pot realiza nici reuniuni şi nici divizări, se opreşte procedura.

Se pot realiza şi variante ale procedeului. Astfel, una din posibilităţi este că iniţial se divizează imaginea într-un set de blocuri pătratice. Divizările ulterioare se desfăşoară în aceeaşi idee, dar reuniunea este iniţial limitată la grupuri de patru blocuri care sunt descendente din arborele pătratic şi care satisfac predicatul P. Când nu mai sunt posibile reuniuni de acest tip, procedura se termină cu o reuniune finală a regiunilor care satisfac pasul 2 de mai sus.

Este evident că, regiunile care sunt astfel reunite pot diferi ca dimensiune. Avantajul principal al acestei abordări este utilizarea aceluiaşi arbore pătratic pentru divizare şi reuniune până în momentul reuniunii finale.

Prezentarea utilizării unui algoritm de divizare şi reuniune este ilustrată în figura 2.1.11. Imaginea luată în considerare constă dintr-un singur obiect şi fond. Pentru simplificare, se presupune că atât obiectul cât şi fondul au intensităţi constante şi că $P(R_i)$ = TRUE dacă toţi pixelii din R_i au aceeaşi intensitate. Astfel, pentru întreaga regiune a imaginii R, va rezulta $P(R)$ = FALSE, aşa că imaginea se va diviza ca şi în figura 2.1.11,a. La pasul următor doar colţul din stânga sus satisface predicatul şi prin urmare va rămâne neschimbat, în timp ce celelalte trei pătrate se divizează în pătrate mai mici, aşa cum se arată în figura 2.1.11,b. În acest punct al procedurii mai multe regiuni pot fi reunite, cu excepţia celor două pătrate ce se află în partea de jos a imaginii. Rezultatul operaţiei de divizare şi reuniune este prezentat în figura 2.1.11,c.

Toate regiunile satisfac predicatul P, şi reuniunea regiunilor similare rezultate din ultima operaţie de divizare va genera rezultatul final, adică segmentarea dorită ca în figura 2.1.11,d.

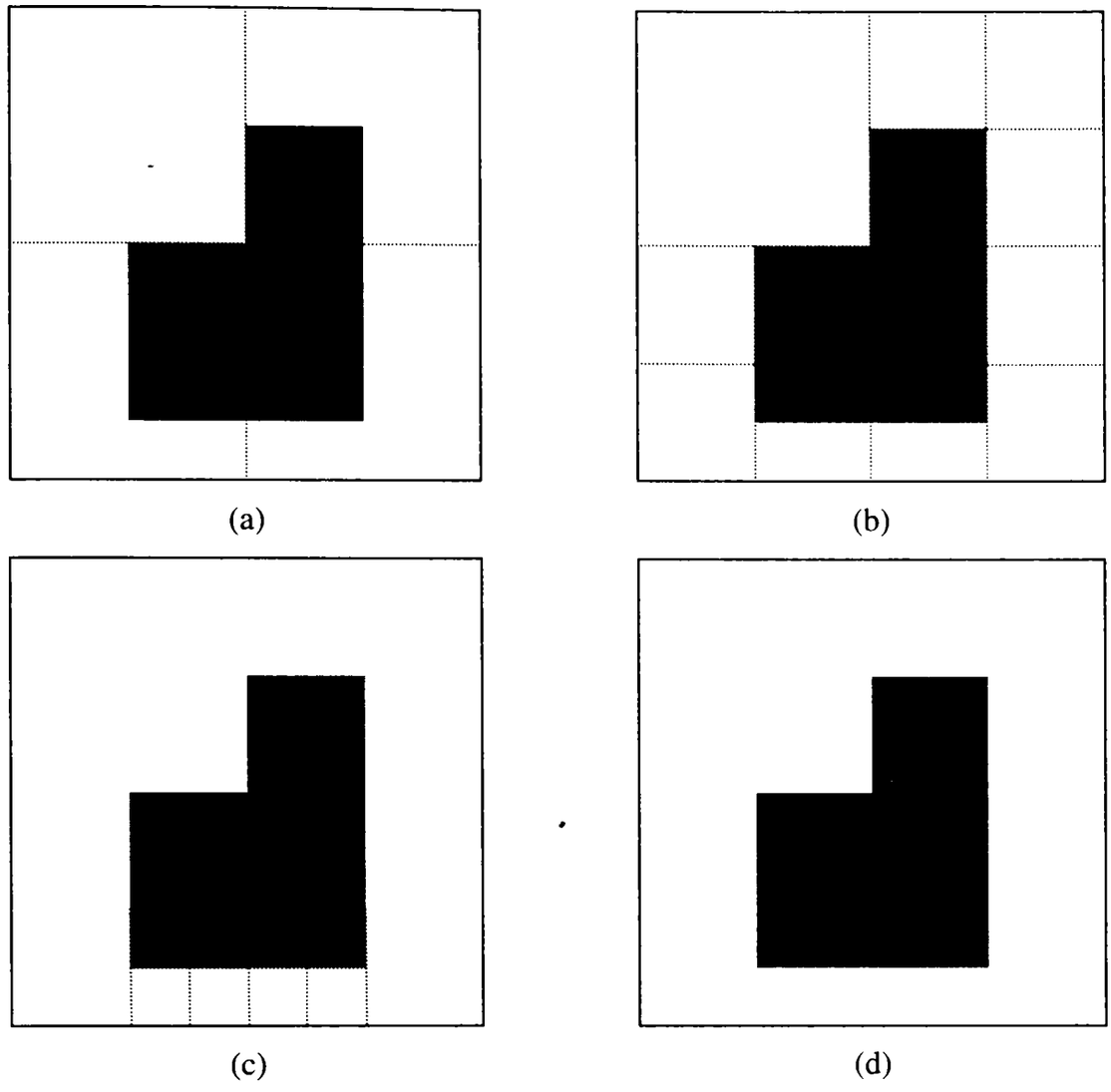


Fig. 2.1.11
Exemplu de algoritm de divizare şi reuniune

2.1.4 Utilizarea mişcării

Mişcarea este un procedeu utilizat pentru a extrage obiectele de interes dintr-un fond. În vederea artificială, mişcarea apare în mai multe variante: mai multe obiecte se mişcă pe o bandă rulantă, un senzor montat pe o mână mişcătoare înregistrează alte obiecte statice ori în mişcare sau, mişcarea întregului sistem robotizat şi a corpurilor din jur. Problema care se pune este sesizarea mişcării prin analiza imaginilor obiectelor.

- **Abordarea de bază**

Una din cele mai simple abordări pentru detectarea schimbărilor dintre două imagini $f(x,y,t_i)$ şi $f(x,y,t_j)$ luate la timpii t_i respectiv t_j,, este compararea celor două imagini pixel cu pixel. Una din procedurile de a realiza acest lucru este *diferenţa imaginii.*

Se presupune că există o imagine de referință conținând doar componente staționare. Dacă se compară această imagine cu o imagine următoare care are același fond dar conține și un obiect mișcător, diferența dintre cele două imagini va anula componentele staționare, lăsând doar intrările nenule care corespund componentelor nestaționare ale imaginii.

O imagine diferență dintre două imagini luate la timpii t_i și t_j poate fi definită astfel :

$$d_{ij}(x,y)=\begin{cases}1 & dacă \quad \left|f(x,y,t_i)-f(x,y,t_j)\right|>\theta \\ 0 & altfel\end{cases} \tag{2.1.25}$$

unde : θ este pragul.

Este de remarcat faptul că $d_{ij}(x,y)$ are un 1 la coordonatele spațiale (x,y) doar dacă diferența de intensitate dintre cele două imagini este apreciabilă la coordonatele respective, așa cum este impus de pragul θ.

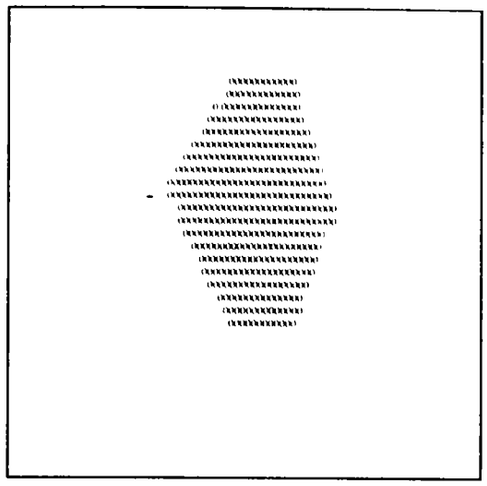
În analiza dinamică a imaginii, toți pixelii din $d_{ij}(x,y)$ cu valoarea 1 sunt considerați rezultatul mișcării obiectului.

Această abordare este aplicabilă doar dacă cele două imagini sunt înregistrate și iluminarea este relativ constantă între limitele stabilite de pragul θ. În realitate, intrările marcate cu 1 în $d_{ij}(x,y)$ apar deseori și din pricina zgomotului. În mod normal, acestea vor fi puncte izolate în imaginea diferență și o metodă simplă de eliminare a lor este formarea de regiuni 4 sau 8 conexe de 1 în $d_{ij}(x,y)$. Apoi se ignoră acele regiuni care au un număr de intrări mai mic decât cel predeterminat.

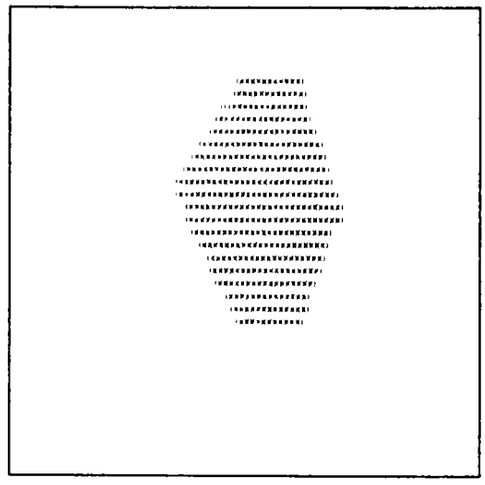
Acest procedeu poate duce la ignorarea unor obiecte în mișcare, foarte mici sau foarte lente, dar mărește șansele ca intrările rămase în imaginea diferență să fie datorate doar mișcării.

Conceptele prezentate mai sus sunt exemplificate în figura 2.1.12. Figura 2.1.12,a reprezintă imaginea referință luată la timpul t_i și conține un singur obiect de intensitate constantă care se deplasează cu viteză constantă peste o suprafață de fond, de asemenea de intensitate constantă. Figura 2.1.12,b arată aceeași imagine luată la timpul t_j, iar figura 2.1.12,c reprezintă imaginea diferență calculată folosind ecuația (2.1.25) cu un prag mai mare decât intensitatea constantă a fondului.

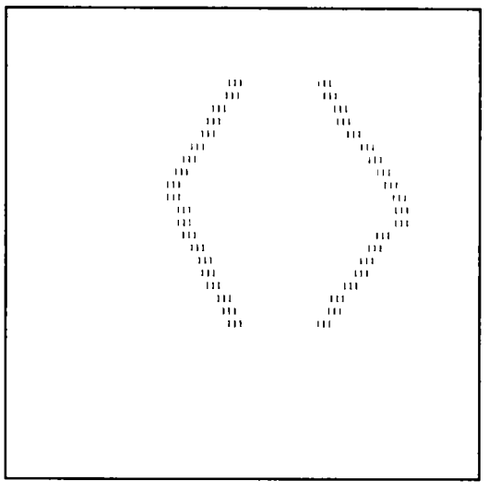
Se remarcă faptul că s-au obținut două regiuni distincte din procesul de diferență: o regiune este rezultatul feței obiectului iar cealaltă, a spatelui obiectului mișcător.



(a)



(b)



(c)

Fig. 2.1.12
(a) Imaginea luată la timpul t_i
(b) Imaginea luată la timpul t_j
(c) Imaginea diferență

➢ **Diferențe acumulative**

Este evident că o imagine diferență va conține deseori intrări izolate datorate zgomotului. Deşi s-a văzut că numărul acestor intrări poate fi redus sau eliminat cu ajutorul unei analize de conectivitate cu prag, acest proces de filtrare poate elimina din păcate obiecte mici sau cu viteză de deplasare redusă.

Se tratează această problemă luând în considerare modificările la nivel de pixel pe mai multe cadre de imagine, introducând astfel o "memorie" în proces. Ideea de bază este aceea de a ignora acele modificări care apar sporadic pe parcursul unei secvențe de imagini, fiind atribuite zgomotului aleator [8], [11], [18], [89], [93], 103], [115], [124],]133], [155].

Se consideră o secvență de imagini $f(x,y,t_1)$, $f(x,y,t_2)$, ..., $f(x,y,t_n)$, și fie $f(x,y,t_1)$ ***imaginea de referință***. O ***imagine diferență acumulativă*** este formată prin compararea acestei imagini de referință cu fiecare imagine din cadrul secvenței. Un contor pentru fiecare locație de pixel în cadrul imaginii acumulative este incrementat de fiecare dată când apare o diferență la acea locație de pixel între imaginea de referință și imaginea din secvență. Astfel, când imaginea k este comparată cu imaginea de referință, intrarea dintr-un pixel dat din imaginea acumulativă arată de câte ori intensitatea la acea locație a fost diferită de valoarea pixelului corespunzător din imaginea de referință. Diferențele se stabilesc cu ajutorul relației (2.1.25).

Această abordare teoretică este exemplificată în figura 2.1.13. Figurile de la (a) la (e) reprezintă un obiect rectangular (reprezentat de zerouri) care se deplasează spre dreapta cu o viteză constantă de 1 pixel/cadru. Imaginile reprezintă secvențe de timp luate la o deplasare de pixel. Figura 2.1.13,a este imaginea de referință. Figurile 2.1.13,b până la 2.1.13,d sunt cadrele 2 până la 4 din secvență, iar figura 2.1.13,e este cadrul 11. Figurile 2.1.13,f până la 2.1.13,i sunt imaginile acumulative. În figura 2.1.13,f, coloana de 1 din stânga rezultă ca diferență dintre obiectul din figura 2.1.13,a și fondul din figura 2.1.13,b. Coloana de 1 din dreapta este cauzată de diferența dintre fondul din imaginea de referință și marginea din față a obiectului mișcător. În momentul celui de-al patrulea cadru, figura 2.1.13,d, prima coloană diferită de zero a imaginii de diferențe acumulative arată contorul 3, indicând trei diferențe totale dintre coloana din imaginea de referință și coloana corespunzătoare din secvența de cadre. În final, figura 2.1.13,i, arată un total de 10 ("A" în hexazecimal) modificări la acea locație. Celelalte intrări din figură sunt explicate într-un mod similar.

Se obișnuiește să se ia în considerare trei tipuri de imagini diferență acumulative: absolute (AADI), pozitive (PADI), și negative (NADI). Ultimele două cantități sunt obținute folosind ecuația (2.1.25) fără valoarea absolută, folosind imaginea de referință în loc de $f(x,y,t_i)$. Presupunând că intensitățile unui obiect sunt numeric mai mari decât cele ale fondului, atunci dacă diferența este pozitivă, ea va fi comparată cu un prag pozitiv; dacă este negativă, ea va fi comparată cu un prag negativ. Această definiție se inversează dacă intensitățile unui obiect sunt mai mici decât fondul.

Figura 2.1.14,a până la c arată cele trei tipuri de imagini diferență acumulative AADI, PADI, și NADI pentru un obiect de mărimea 20x20 pixeli a cărui intensitate

este mai mare decât cea a fondului, şi care se deplasează cu viteză constantă în direcţia sud-est.

Este important de remarcat faptul că, atunci când obiectul este deplasat din poziţia lui originală-creşterea spaţială a unei imagini PADI se opreşte. Cu alte cuvinte, când un obiect a cărui intensitate este mai mare decât cea a fondului este complet deplasat din poziţia lui iniţială din imaginea de referinţă, nu vor mai apărea intrări noi în imaginea diferenţă acumulativă pozitivă. Astfel, când creşterea se opreşte, imaginea PADI reprezintă poziţia iniţială a obiectului în cadrul imaginii de referinţă.

Această proprietate poate fi utilizată în scopul realizării unei referinţe pentru o secvenţă dinamică de imagini.

Se mai observă din figura 2.1.14 faptul că o imagine AADI conţine atât regiunile unei imagini PADI cât şi cele ale unei imagini NADI, şi că, intrările din aceste imagini dau o indicaţie asupra vitezei şi direcţiei mişcării obiectului.

- **Stabilirea unei imagini de referinţă**

Problema cea mai importantă în aplicarea tehnicilor anterioare este obţinerea unei imagini de referinţă faţă de care să se realizeze comparaţiile. Aşa cum s-a menţionat, diferenţa dintre două imagini într-o problemă de imagini dinamice are tendinţa de a anula toate componentele staţionare, şi a lăsa doar elementele de imagine care corespund zgomotului şi obiectelor mişcătoare.

Problema zgomotului poate fi tratată prin filtrare aşa cum s-a prezentat anterior sau prin formarea de imagini acumulative.

În practică nu este posibilă întotdeauna obţinerea unei imagini de referinţă care să conţină doar elemente staţionare şi devine necesară construirea unei imagini referinţă dintr-un set de imagini conţinând unul sau mai multe obiecte mişcătoare. Aceasta se poate întâmpla în situaţii care descriu scene aglomerate sau în cazuri în care este necesară o frecvenţă de citire ridicată [16], [19], [29], [113], [150].

Una din modalităţile de generare a unei imagini de referinţă este următoarea: Se presupune că prima imagine din secvenţă este cea de referinţă. Când un obiect nestaţionar s-a mutat complet de pe poziţia lui iniţială din imaginea referinţă, fondul corespunzător din imaginea curentă poate fi duplicat în locaţia originală ocupată de obiect în imaginea referinţă. Când toate obiectele mişcătoare s-au mutat complet din poziţia lor originală, se va crea o imagine de referinţă conţinând doar componentele

staționare. Deplasarea completă a obiectelor poate fi stabilită monitorizând creşterea imaginii PADI.

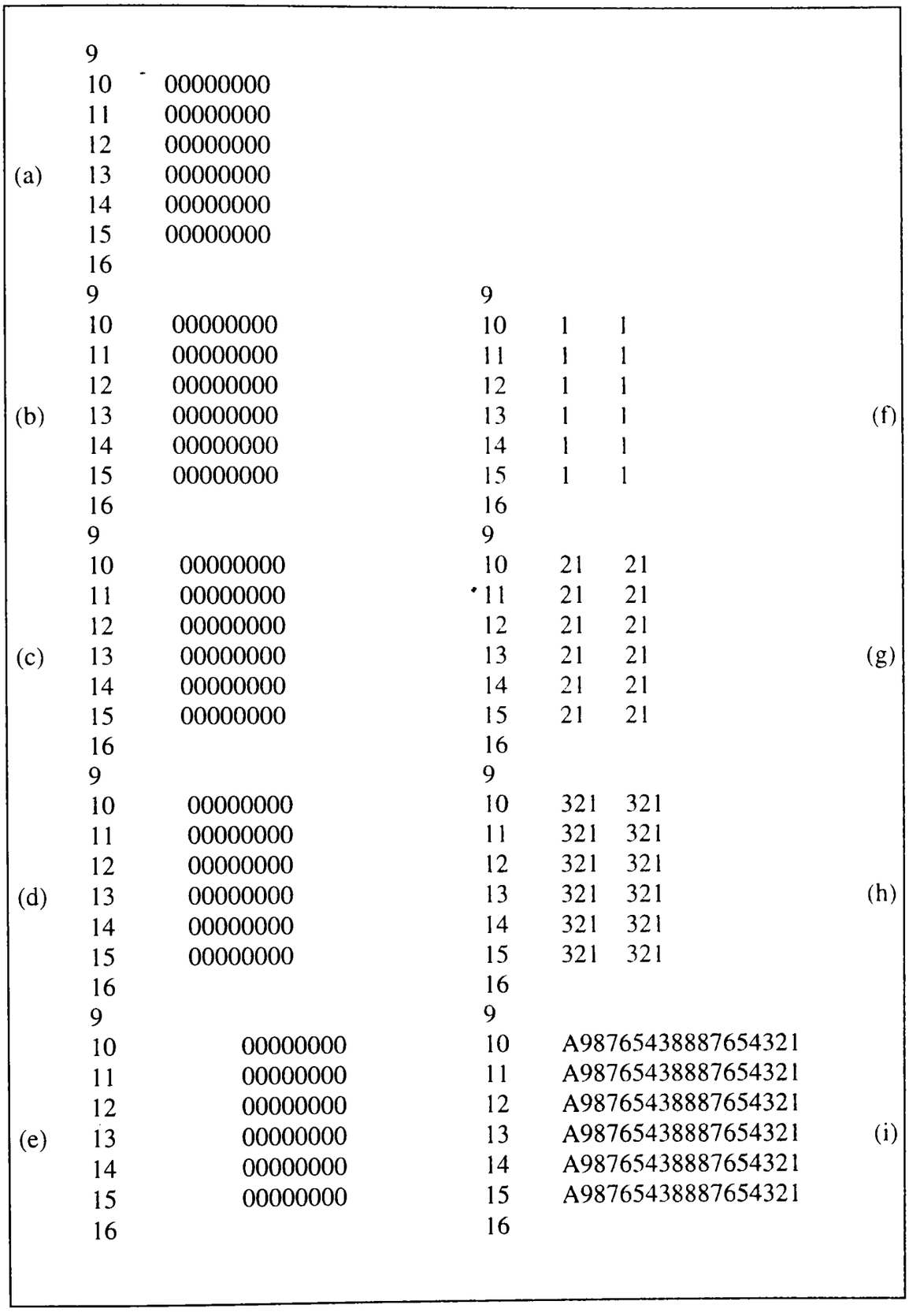


Fig. 2.1.13
Deplasarea unui obiect rectangular

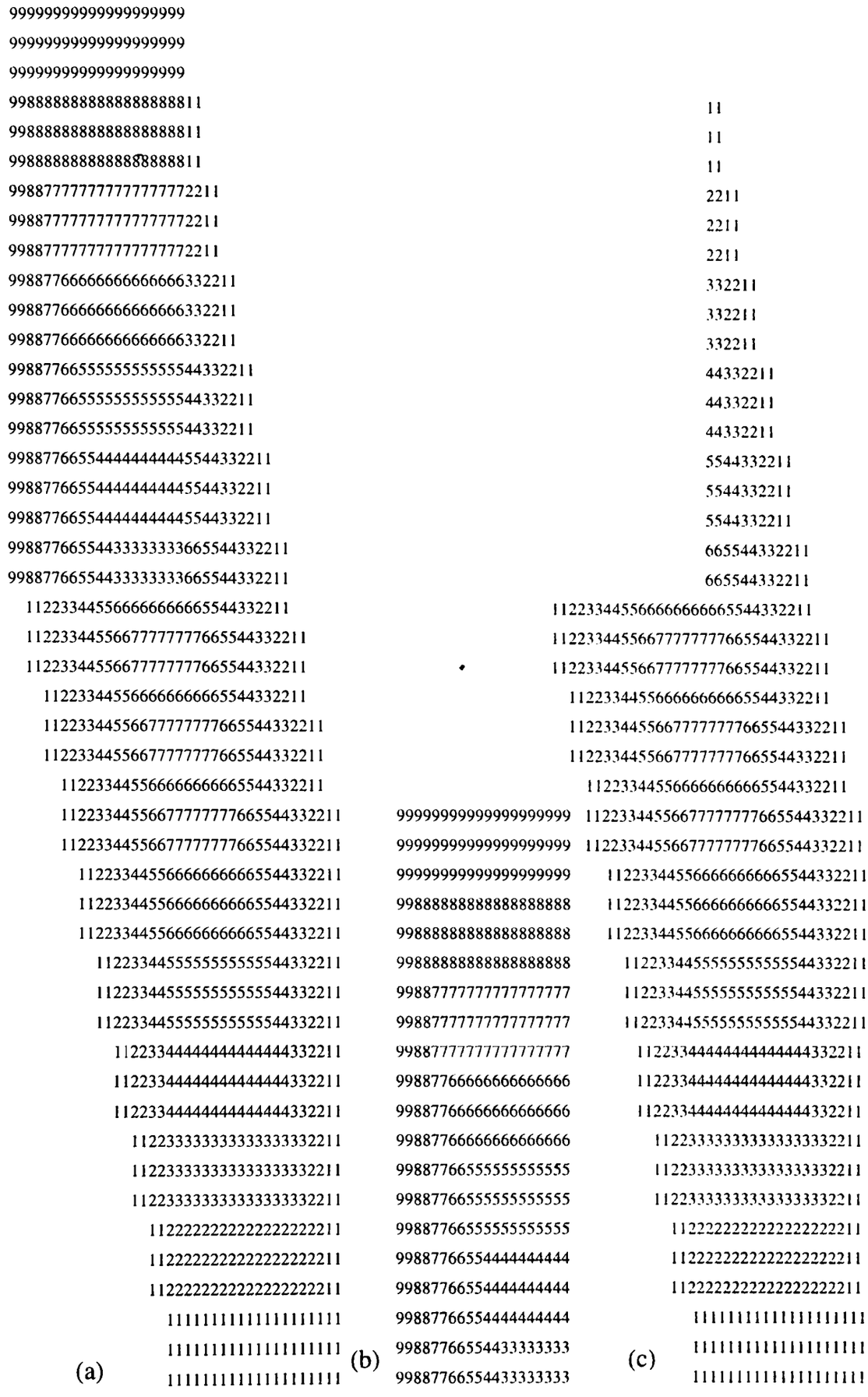


Fig. 2.1.14

Imagini diferență acumulative a unui obiect de 20x20 pixeli cu intensitate mai mare decât cea a fondului şi care se mişcă în direcția sud-est. (a) Absolută, (b) pozitivă, (c) negativă.

2.1.5. Implementarea unui algoritm de segmentare folosind tehnica pragului

Se prezintă implementarea unui algoritm ce are ca punct de plecare folosirea pragului ca prim pas în procesul de segmentare, urmat de aplicarea unei detecții de contur asupra obiectelor rezultate în urma binarizării imaginii cu ajutorul pragului [26],]27]. În final se vor obține punctele de contur care descriu forma obiectului, puncte cu ajutorul cărora se va realiza ulterior descrierea acestuia.

Imaginea originală, din figura 2.1.15,a, conține un număr de forme geometrice plane, iar scopul acestui algoritm, în faza de segmentare este separarea fiecărei figuri geometrice din cadrul imaginii şi determinarea punctelor care o alcătuiesc, puncte ce vor fi folosite ulterior în faza de descriere.

În vederea utilizării unui prag pentru separarea pixelilor care aparțin obiectelor şi celor care aparțin fundalului, este necesară extragerea histogramei imaginii. Din histograma prezentată în figura 2.1.15,b se determină un prag optim de separare egal cu nivelul de gri 75.

În urma aplicării pragului de binarizare, rezultă imaginea din figura 2.1.15,c, în care figurile geometrice sunt clar conturate şi individualizate.

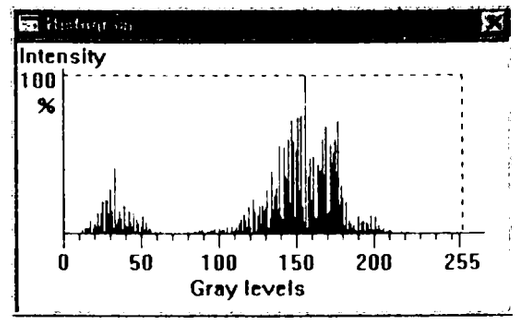
Următorul pas în vederea segmentării şi obținerii punctelor care descriu figurile geometrice, constă în alegerea unui rastru.

În cazul acestui exemplu rastrul este de 10 x 10 pixeli ca în figura 2.1.15,d. Cu cât rastrul este mai mic cu atât descrierea obiectelor va fi mai exactă, dar el este la 4 x 4 pixeli datorită faptului că grosimea conturului ce va rezulta în urma detecției de contur este de 2 pixeli, câte unul de o parte şi de cealaltă a marginii obiectului, astfel ca suma a două linii de contur adiacente să fie de 4 pixeli.

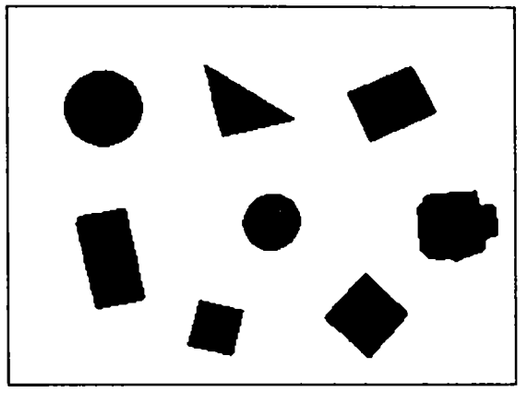
Rastrul este necesar deoarece o descriere la nivel de pixel ar face imposibilă recunoaşterea datorită zgomotului şi diferențelor de iluminare. În urma aplicării rastrului rezultă imaginea din figura 2.1.15,e în care fiecare ochi de rastru a fost umplut sau golit de pixelii obiectului dacă numărul de pixeli ce aparțin obiectului, depăşeşte 50% din numărul total de pixeli din ochiul de rastru.

În imaginea 2.1.15,f se scoate în evidență modelarea obiectelor după rastru. Ultima fază a procesului de segmentare înainte de obținerea punctelor ce descriu figurile geometrice este detecția de contur care generează imaginea din figura 2.1.15,g.

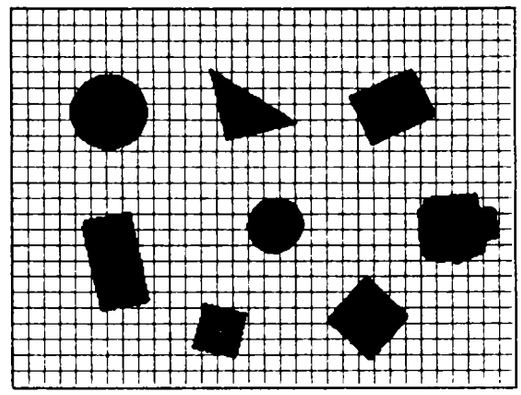
(a)



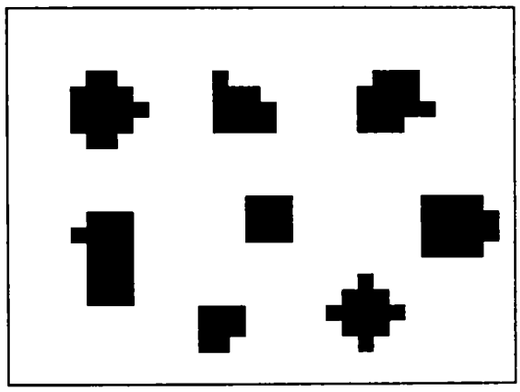
(b)



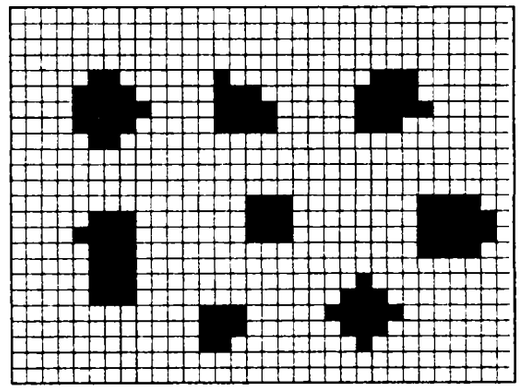
(c)



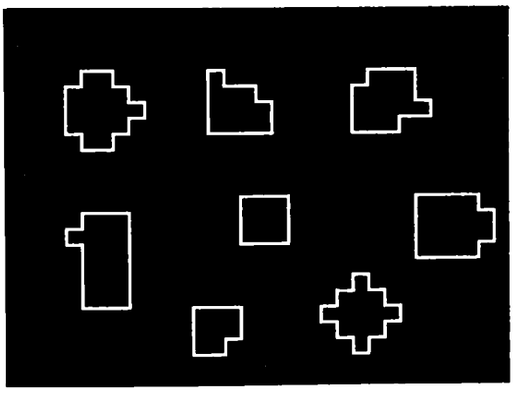
(d)



(e)



(f)



(g)

Fig. 2.1.15
(a) Imaginea originală
(b) Histograma imaginii originale
(c) Imaginea binarizată cu un prag de 75
(d) Rastrul 10 x 10 aplicat
(e) Imaginea rezultată în urma aplicării rastrului
(f) Identificarea rastrului cu noua imagine
(g) Detecția de contur finală

Procesul de segmentare nu se opreşte aici, ci doar reprezentarea grafică a rezultatelor, deoarece urmează faza cea mai importantă şi anume obţinerea punctelor ce individualizează fiecare obiect în cadrul imaginii. Acest algoritm de extragere a punctelor obiectului se aplică imaginii din figura 2.1.15,g şi începe în ochiul de rastru din colţul stânga sus al imaginii.

Prin incrementarea cu un pas de rastru se caută primul pixel de contur, alb, ce aparţine unui obiect. Acest pixel este cel mai de sus pixel al obiectului şi cel mai din stânga din cei care îndeplinesc condiţia de cel mai de sus. Odată găsit un astfel de pixel de contur se caută prezenţa următorului pixel de contur la o distanţă de un pas de rastru în funcţie de orientarea segmentului anterior găsit. În cazul primului punct, unde nu există segment anterior următorul punct de rastru este întotdeauna la dreapta.

Regula este foarte simplă, ea aplicând-se astfel : dacă segmentul anterior avea orientarea 0 (de la stânga la dreapta) se va căuta următorul pixel de rastru la dreapta acestuia, în caz negativ deasupra acestuia şi în final dedesubtul acestuia. Regula se păstrează şi pentru celelalte trei orientări anterioare. Unica limitare a acestui algoritm este ca distanţa dintre două obiecte să fie de cel puţin doi paşi de rastru în orice punct al lor.

Procedura se reia pentru fiecare obiect, verificându-se la căutarea punctului de start al obiectului dacă acesta nu aparţine unui alt obiect găsit anterior.

În urma aplicării acestui algoritm obiectele din imagine sunt individualizate şi în acelaşi timp creează baza pentru descrierea obiectelor cu ajutorul codurilor de înlănţuire.

2.2. Descrierea

Problema descrierii în vederea artificială este aceea a extragerii de caracteristici dintr-un obiect în scopul recunoaşterii. Descriptorii ar trebui să fie independenţi de caracteristicile obiectului, poziţia şi orientarea sa. Ei trebuie să conţină suficiente informaţii discriminatorii pentru a identifica un obiect faţă de altul.

Descrierea este un element central în proiectarea unui sistem de vedere artificială în sensul că descriptorii afectează nu doar complexitatea algoritmilor de recunoaştere dar şi performanţele acestora.

Descriptorii se împart în trei categorii principale:

- descriptori de contur
- descriptori regionali
- descriptori specializaţi pe reprezentarea structurilor tridimensionale

2.2.1. Descriptori de contur

➢ **Coduri de înlănţuire**

Codurile de înlănţuire sunt utilizate pentru a reprezenta un contur ca pe o mulţime de segmente de dreaptă de o anumită lungime şi direcţie. În mod normal, această reprezentare se determină pe un rastru rectangular folosind conectivitatea 4 sau 8, aşa cum se arată în figura 2.2.1.

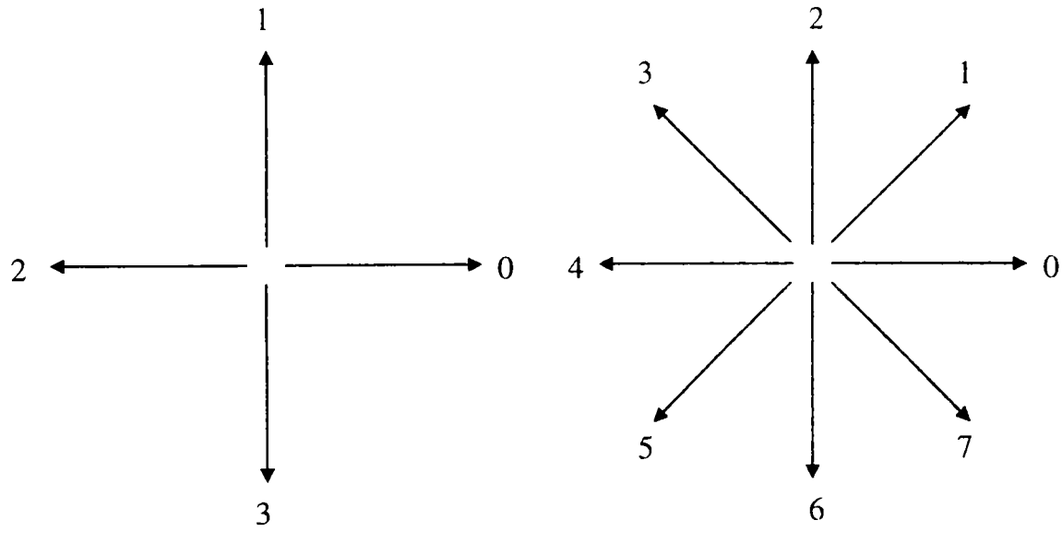
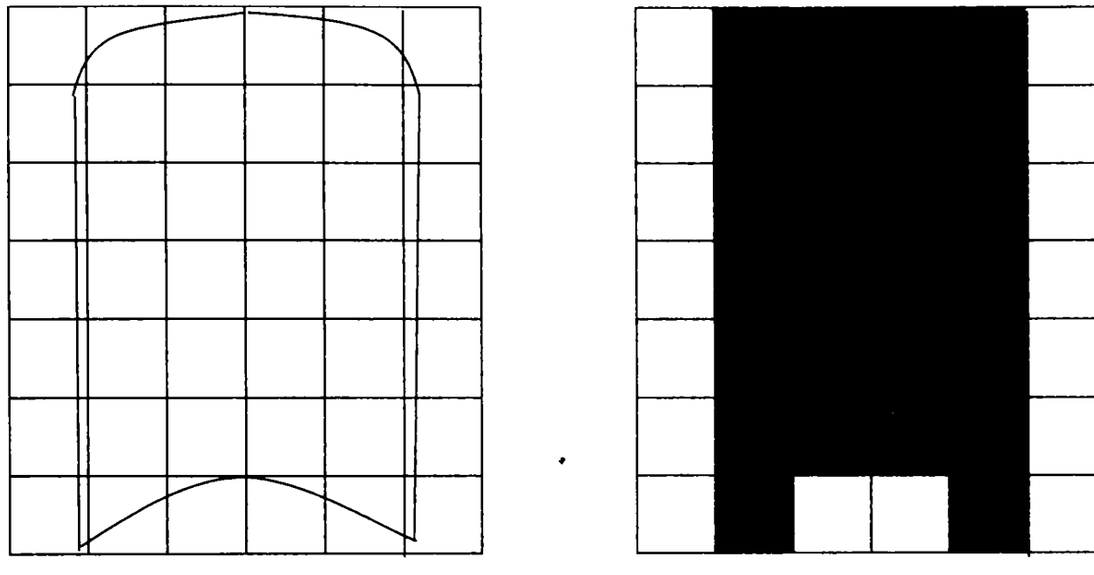


Fig. 2.2.1
(a) cod de înlănţuire 4 direcţional
(b) cod de înlănţuire 8 direcţional

Lungimea fiecărui segment este stabilită de rezoluția rastrului, iar direcția este dată de codul ales. Este de remarcat faptul că sunt suficienți 2 biți pentru a reprezenta toate direcțiile în codul 4, şi de trei biți pentru codul 8. Desigur, este posibilă stabilirea de coduri de înlănțuire cu mai multe direcții, dar codurile prezentate în figura 2.2.1 sunt frecvent folosite în practică.

Pentru a genera codul de înlănțuire al unui anumit contur se selectează în primul rând spațierea rastrului (figura 2.2.2,a).



(a) (b)

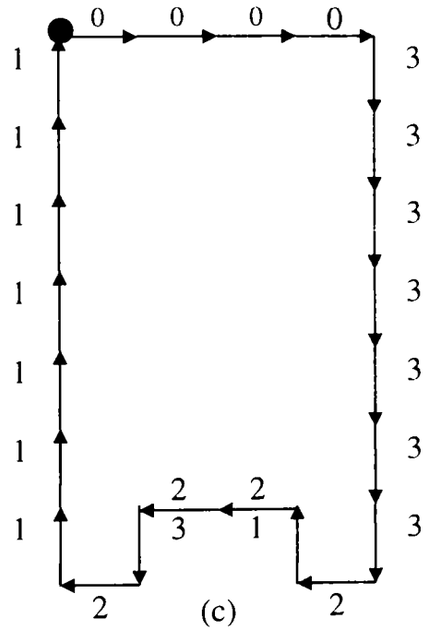


Fig. 2.2.2
Paşii necesari pentru obținerea codului de înlănțuire.
Codul de înlănțuire:
0 0 0 0 3 3 3 3 3 3 3 2 1 2 2 3 2 1 1 1 1 1 1 1

După aceea, dacă o celulă este mai mult decât o anumită cantitate specificată (de obicei 50%) în interiorul conturului, i se atribuie acelei celule valoarea 1; în caz contrar i se atribuie valoarea 0.

Figura 2.2.2,b ilustrează acest proces, unde celulele cu valoarea 1 sunt marcate întunecat. În final, se codifică conturul dintre cele două regiuni folosind codurile direcționale prezentate în figura 2.2.1,a. Rezultatul este prezentat în figura 2.2.2,c, unde codificarea a început în colțul din stânga sus și a continuat în direcția dată de acele de ceasornic.

O procedură alternativă este divizarea conturului în segmente de lungime egală (fiecare segment având același număr de pixeli), conectarea capetelor segmentelor prin linii drepte, și atribuirea fiecărei linii a direcției cu cea mai apropiată direcție permisă din codul de înlănțuire. Un exemplu de folosire a acestei metode folosind patru direcții este prezentată în figura 2.2.3.

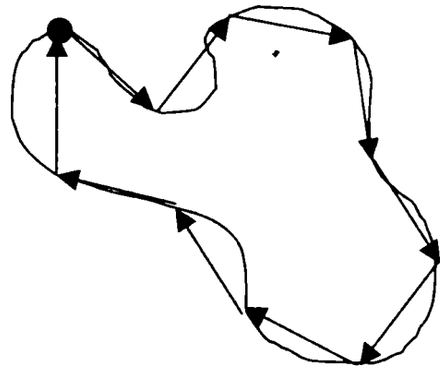


Fig. 2.2.3
Generarea codului de înlănțuire prin divizarea conturului:
0 1 0 3 3 3 2 1 2 1

Codul de înlănțuire al unui anumit contur depinde de punctul de start. Este posibil să se normalizeze acest cod prin următoarea procedură:

Fiind dat un cod de înlănțuire generat prin pornirea dintr-un punct arbitrar, el se va trata ca o secvență circulară de numere de direcție și se va redefini punctul de start în așa fel încât secvența rezultată să formeze un număr de mărime minimă.

Se poate realiza normalizarea pentru rotație și folosirea primei diferențe a codului de înlănțuire, în loc de codul însuși. Diferența este calculată simplu numărând (în sens trigonometric) numărul de direcții care separă două elemente adiacente din cod. De exemplu, prima diferență a codului înlănțuit 10103322 este 3133030.

Dacă se tratează codul ca o secvență circulară, atunci primul element al diferenței este calculat folosind tranziția între ultimul și primul element din înlănțuire. În acest exemplu rezultatul este 33133030.

Normalizarea dimensiunii poate fi realizată prin împărțirea tuturor contururilor obiectelor în același număr egal de segmente și prin ajustarea lungimii segmentelor de cod pentru a intra în această divizare.

Normalizarea precedentă este exactă doar dacă contururile însăși sunt invariante la rotații și schimbări de scală. În practică acest caz este foarte rar. De exemplu, același obiect digitizat în două orientări diferite va avea în general contururi diferite, având gradul de nesimilaritate proporțional cu rezoluția imaginii. Acest efect poate fi redus prin selectarea elementelor de înlănțuire care sunt mai mari față de distanța dintre pixelii din imaginea digitizată sau prin orientarea rastrului din figura 2.2.2 după axele principale ale obiectului care trebuie codificat. Problema se poate rezolva prin intermediul numerelor de formă.

- **Semnături**

O semnătură este reprezentarea unidimensională a funcției de contur. Există mai multe modalități de generare a unei semnături. Una din cele mai simple metode este reprezentarea distanței din centrul obiectului până la contur ca pe o funcție de unghi, așa cum se prezintă în figura 2.2.4.

Semnăturile generate în acest mod sunt în mod evident dependente de dimensiune și de poziția punctului de start. Normalizarea dimensiunii poate fi realizată simplu normalizând curba $r(\theta)$, la valoarea maximă unitate. Problema punctului de start poate fi rezolvată prin obținerea în primul rând a codului de înlănțuire al conturului după care se va folosi metoda prezentată anterior.

Exprimarea distanței în funcție de unghi, nu este unicul mod de a genera o semnătură. Se poate, de exemplu, traversa conturul și reprezenta unghiul dintre o linie tangentă cu conturul și linia de referință ca pe o funcție de poziție de-a lungul conturului. Semnătura rezultată, deși foarte diferită de forma curbei $r(\theta)$, va deține informații de bază despre caracteristicile conturului. De exemplu, segmentele orizontale din curbă vor corespunde liniilor drepte de-a lungul conturului deoarece unghiul tangentei va fi constant aici.

O variantă a acestei metode este utilizarea ca semnătură a ***funcției de densitate de pantă***. Această funcție este o simplă histogramă a valorilor unghiurilor tangentei. Deoarece o histogramă este o măsură a concentrației de valori, funcția de densitate de pantă va răspunde foarte pronunțat acelor părți din contur ce au unghiurile tangentei constante (segmente drepte sau aproape drepte) și va avea văi adânci în părțile în care au loc variații rapide de unghi (colțuri sau alte discontinuități de formă).

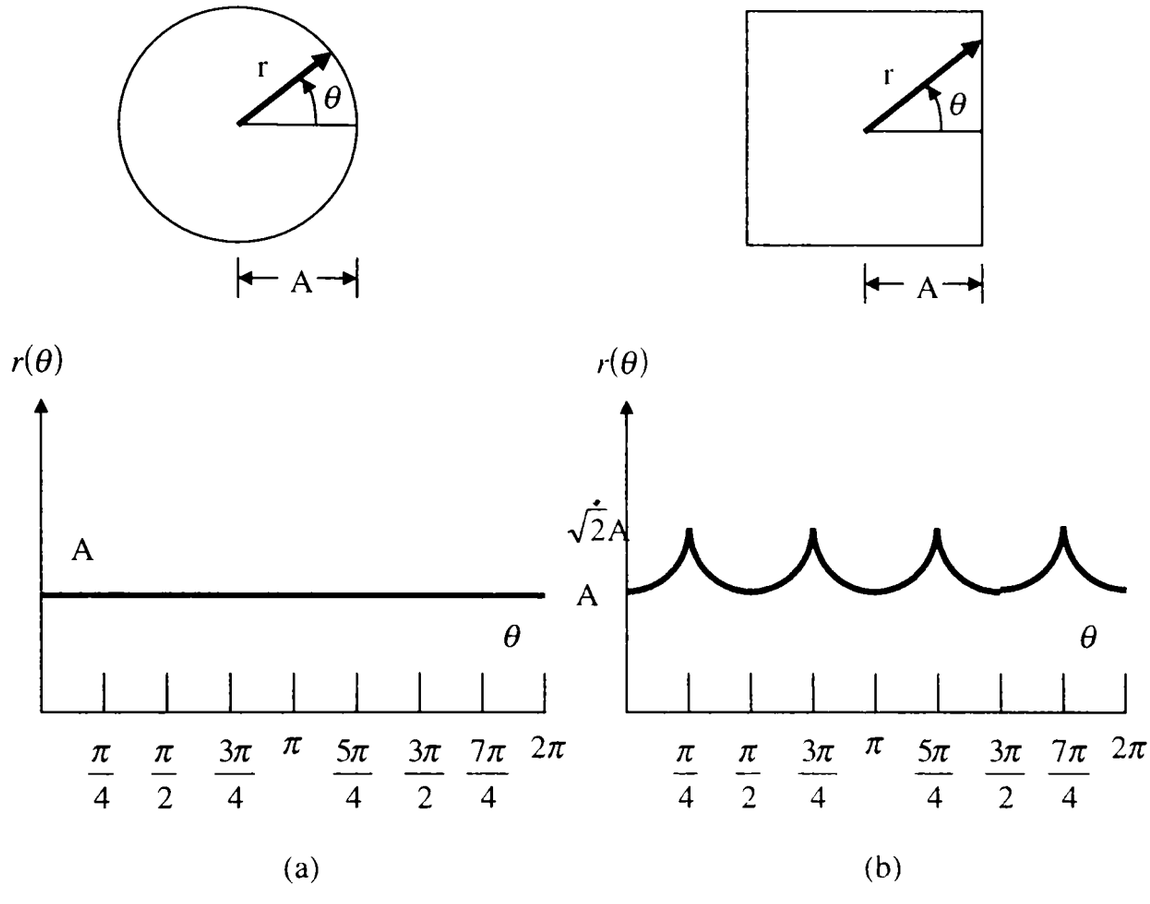


Fig. 2.2.4
Două contururi simple și semnăturile lor unghi
(a) $r(\theta)$ este constant.
(b) $r(\theta) = A \sec\theta$

O dată ce s-a obținut o semnătură, mai rămâne problema de a o descrie într-un mod care să permită diferențierea față de alte semnături corespunzătoare cu alte forme de contur. Această problemă, este mai simplă, deoarece acestea sunt doar funcții unidimensionale.

O abordare folosită în mod frecvent pentru a caracteriza o semnătură este calcularea ***momentelor*** ei. Se presupune că a este o variabilă aleatoare discretă care reprezintă variații de amplitudine într-o semnătură, și fie $p(a_i)$, $i = 1, 2, ..., K$,

histograma corespunzătoare, unde K este numărul de incremente discrete de amplitudine ale lui a. Al n-lea moment al lui a este definit ca:

$$\mu_n = \sum_{i=1}^{K} (a_i - m)^n p(a_i) \tag{2.2.1}$$

unde

$$m = \sum_{i=1}^{K} a_i p(a_i) \tag{2.2.2}$$

- m reprezintă media valorilor lui a
- μ_2 reprezintă variația lui

Doar primele câteva momente sunt în general necesare pentru a diferenția între semnăturile unor forme evident distincte.

➢ Aproximări poligonale

Un contur digital poate fi aproximat cu o precizie arbitrară de un poligon. Pentru o curbă închisă, aproximația este exactă când numărul de laturi ale poligonului este egal cu numărul de puncte din contur în așa fel încât fiecare pereche de puncte adiacente să definească o latură de poligon.

În practică, scopul aproximării poligonale este capturarea esenței unei forme de contur cu cât mai puține segmente poligonale posibile. Cu toate că această problemă nu este simplă şi poate deveni foarte uşor o procedură mare consumatoare de timp, există un număr de tehnici de aproximare poligonală a căror complexitate modestă şi necesar de procesare le recomandă pentru aplicații de vederea artificială.

O primă metodă ar urmări determinarea poligoanelor de perimetru minim. Spre exemplu, făcând referire la figura 2.2.5, se presupune că se include un contur dat într-un set de celule concatenate, ca în figura 2.2.2,a. Se poate vizualiza această includere ca două ziduri corespunzătoare contururilor exterioare şi interioare ale celulelor de demarcație şi se imaginează conturul obiectului ca un material elastic ce se întinde între aceşti pereți. Dacă se permite materialului elastic să se micşoreze, el va lua forma din figura 2.2.2,b, realizând astfel un poligon de perimetru minim care se potriveşte în geometria stabilită de celulele de demarcație. Dacă celulele sunt alese în aşa fel încât fiecare celulă încercuieşte doar câte un punct din contur, atunci eroarea din fiecare celulă dintre conturul original şi aproximarea prin materialul elastic va fi

de cel mult $\sqrt{2}d$, unde d este distanța dintre pixeli. Această eroare poate fi redusă la jumătate forțând fiecare celulă să fie centrată pe pixelul corespunzător.

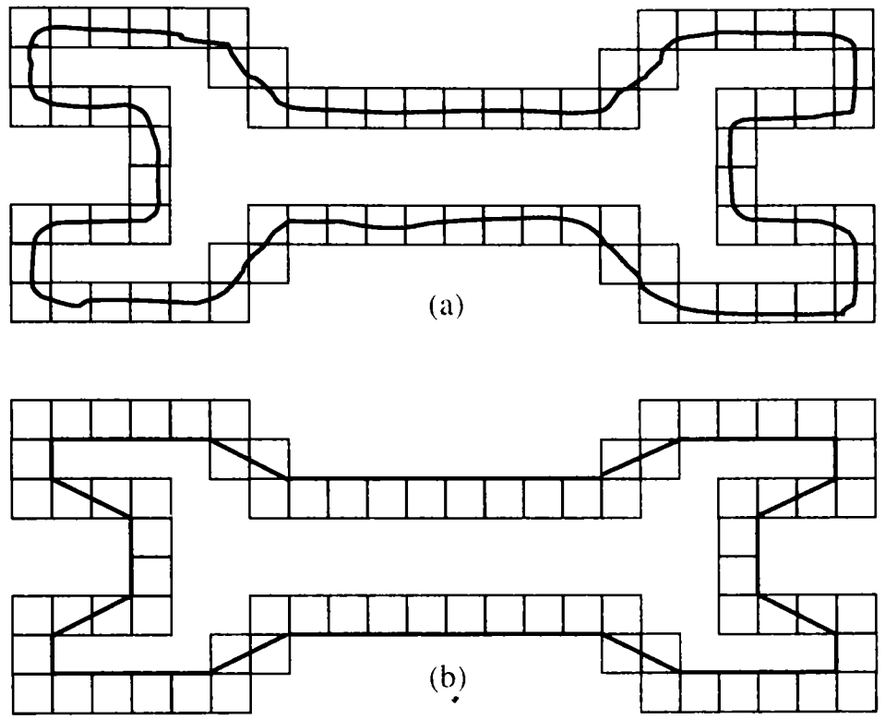


Fig. 2.2.5
(a) conturul obiectului încercuit de celule.
(b) Poligonul de perimetru minim.

- **Numere de formă**

Un contur codificat înlănțuit are mai multe prime diferențe, în funcție de punctul de start. ***Numărul de formă*** al unui astfel de contur, bazat pe codul 4 direcțional din figura 2.2.1,a este definit ca prima diferență de cea mai mică mărime. ***Ordinul*** *n* al unui număr de formă este definit ca numărul de digiți din reprezentarea sa.

Figura 2.2.6 prezintă toate formele de ordinul 4, 6 şi 8, împreună cu reprezentarea codului de înlănțuire, prima diferență, şi numerele de formă corespunzătoare. Trebuie subliniat faptul că prima diferență este calculată tratând codul înlănțuit ca pe o secvență circulară în manieră discutată anterior [1], [9], [15], [17], [22], [25], [87], [90], [95], [99], [100],]108], [109], [117], [133] .

Cu toate că prima diferență a unui cod înlănțuit este independentă de rotație, conturul codificat în general va depinde de orientarea rastrului de codificare.

Una din metodele de normalizare a rastrului este următoarea: ***Axa mare*** a conturului este o linie dreaptă care unește punctele cele mai depărtate una de alta. ***Axa mică*** este perpendiculară pe axa mare și are lungimea în așa fel aleasă încât să formeze un dreptunghi care cuprinde exact conturul selectat. Raportul între axa mare și axa mică este numit ***excentricitatea*** conturului, iar dreptunghiul descris anterior este numit ***dreptunghiul de bază***.

În cele mai multe cazuri un număr de formă unic se va obține prin alinierea rastrului codului de înlănțuire cu marginile dreptunghiului de bază.

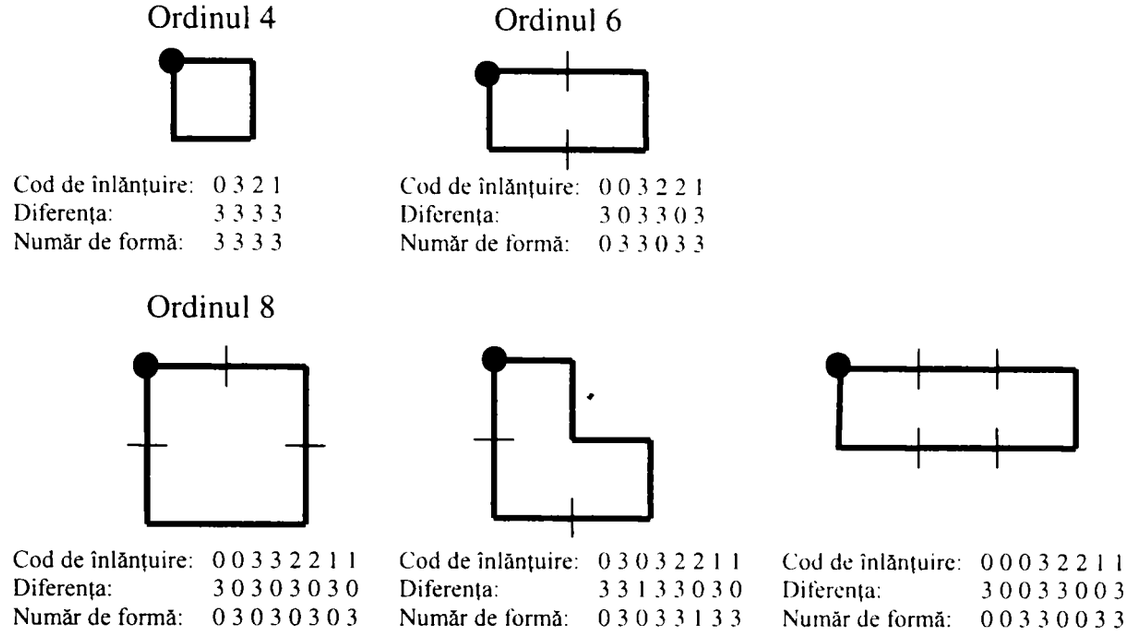


Fig. 2.2.6
Toate formele de ordinul 4, 6 și 8.
Direcțiile sunt date de figura 2.2.1, iar punctul indică poziția de start.

În practică, fiind dat un ordin de formă dorit, se va găsi dreptunghiul de ordinul *n* a cărui excentricitate aproximează cel mai bine dreptunghiul de bază și se va folosi acest nou dreptunghi pentru a determina dimensiunea rastrului. De exemplu, dacă $n = 12$, toate dreptunghiurile de ordin 12 (acelea al căror perimetru este 12) sunt 2x4, 3x3, și 1x5. Dacă excentricitatea dreptunghiului 2x4 se potrivește cel mai bine cu cea a dreptunghiului de bază al unui contur dat, se va stabili un rastru 2x4 centrat pe dreptunghiul de bază și se va folosi procedura prezentată deja pentru obținerea codului de înlănțuire.

Numerele de formă rezultă din prima diferență a acestui cod, așa cum s-a indicat mai sus. Deși ordinul numărului de formă rezultat va fi de obicei egal cu *n* din cauza felului în care a fost selectată spațierea rastrului, contururile cu depresiuni

comparabile cu spațierea vor genera uneori numere de formă de ordin mai mare decât n. În acest caz, se specifică un dreptunghi de ordin mai mic decât n și se repetă procedura până când numărul de formă rezultat va avea ordinul n.

Fie în spațiul de lucru bidimensional în care evoluează un robot, un obiect de forma din figura 2.2.7 care va trebui descris prin metoda numerelor de formă.

Se presupune că este specificat $n = 22$ pentru conturul din figura 2.2.7,a. În vederea obținerii unui număr de formă de acest ordin se urmăresc pașii menționați.

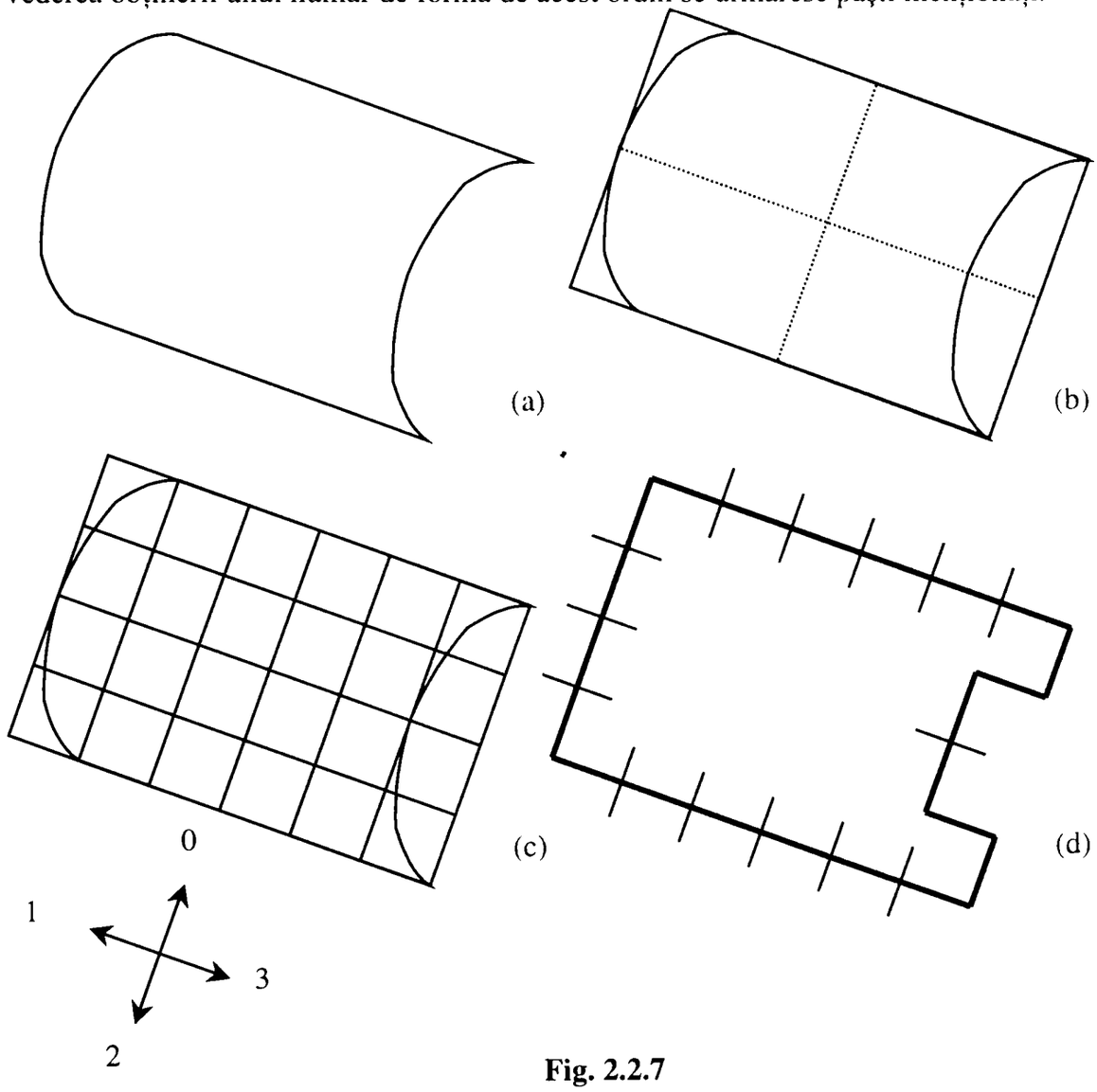


Fig. 2.2.7

Pași în generarea numărului de formă

Cod de înlănțuire:	1111110000333333212232
Prima diferență:	3000003000300000331013
Număr de formă:	0000030003000003310133

În primul rând se determină dreptunghiul de bază, (figura 2.2.7,b), cel mai apropiat dreptunghi de ordinul 22 fiind un dreptunghi 4 x 6. Se împarte dreptunghiul de bază așa ca în figura 2.2.7,c, unde se remarcă faptul că direcțiile codului de

înlănțuire sunt aliniate cu rastrul rezultat. În final, se obține codul de înlănțuire și se folosește prima diferență pentru a calcula numărul de formă, așa cum se prezintă în figura 2.2.7,d.

➢ Utilizarea numerelor de formă în descrierea figurilor geometrice plane

Problema principală, este descrierea figurilor geometrice plane uzuale: cerc, pătrat, dreptunghi și triunghi echilateral, ce pot intra ca elemente constitutive în orice figură complexă. Ca element de descriere s-au folosit numerele de formă. Numărul de formă de ordin 20 a fost ales pentru că putea permite descrierea unică cea mai simplă și cea mai reprezentativă. Numerele de formă mai mici prezintă probleme de descriere iar cele mari sunt prea sensibile la zgomote. Astfel utilizând acest număr de ordin și utilizând o înlănțuire 4-direcțională vor exista următoarele forme de contur care să reprezinte cele patru figuri geometrice selectate (figura 2.2.8).

În scopul recunoașterii acestor figuri geometrice indiferent de dimensiunea sau orientarea lor, precum și de punctul start din care se începe generarea codului de înlănțuire, trebuie aplicate trei metode de normalizare a acestui cod:

- la orientare
- la mărime
- la poziția punctului de start

❖ Normalizarea la orientare

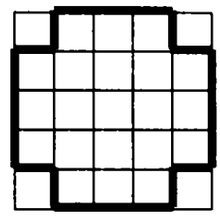
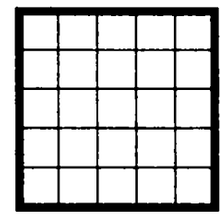
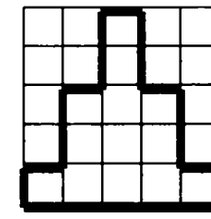
Cel mai simplu mod de normalizare la orientare este alinierea figurii geometrice după axele sale (mare și mică). Dar, această metodă nu este ușor de utilizat în cazul dreptunghiurilor și pătratelor, pentru că în cazul acestor figuri geometrice axa mare este una dintre diagonale, iar o orientare după diagonală ar face descrierea dreptunghiului foarte dificilă (figura 2.2.9). Cea mai bună orientare pentru descrierea dreptunghiului este de-a lungul laturii lui mari, însă determinarea acesteia pentru un dreptunghi cu orientare oarecare necesită utilizarea unui algoritm special care trebuie să fie valabil atât în cazul pătratului, cercului sau triunghiului, cât și în cazuri mai generale.

Ideea principală a întregului algoritm este rotirea punctelor periferice ale corpului din grad în grad de la 0° la 180° și determinarea minimului distanței celei

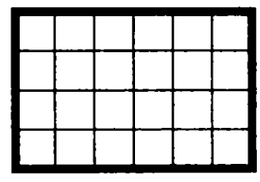
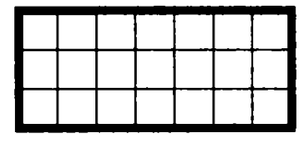
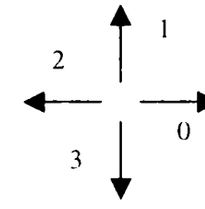
mai mari dintre două puncte ale corpului pe axa Ox și reținerea unghiului care a generat acea distanță.

Există două moduri de abordare a problemei: unul care pune accent pe viteza de calcul și altul care pune accent pe precizie. În principiu ele sunt asemănătoare diferența constând în cantitatea de puncte care se utilizează în cadrul algoritmului pentru determinarea laturii mari.

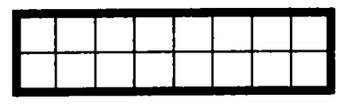
Cel mai precis mod de determinare a laturii mari este rotirea tuturor punctelor rezultate în urma detecției de contur. Datorită numărului mare de puncte care descriu astfel un contur, metoda este lentă dar foarte precisă excluzând posibilitatea încadrării într-o altă clasă a corpului.

Figură	Cerc	Pătrat	Triunghi echilateral
Cod înlănțuire	01000303332322212111	0000033333222221111	01101103303303222221
Prima diferență	00313003130030300313	00003000030000300003	10310330130133000033
Număr de formă	00303003130031300313	00003000030000300003	00003310310330130133

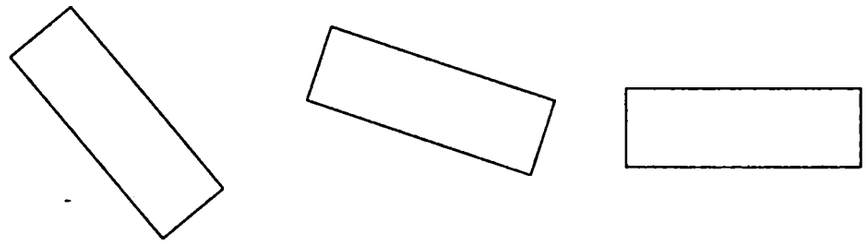




Figură	Dreptunghi	Dreptunghi	Cod înlănțuire 4-direcțional
Cod înlănțuire	00000033332222221111	00000003332222222111	
Prima diferență	00030000030003000003	00300000030030000003	
Număr de formă	00000300030000030003	00000030030000003003	




Figură	Dreptunghi	Dreptunghi
Cod înlănțuire	0000000332222222211	0000000032222222221
Prima diferență	0300000030300000003	3000000033000000003
Număr de formă	0000000303000000303	0000000033000000033

Fig. 2.2.8
Codurile de înlănțuire, primele diferențe și numerele de formă ale principalelor figuri geometrice plane pentru ordinul 20



Dreptunghi într-o poziție oarecare | Dreptunghi orientat după axa mare | Dreptunghi orientat după latura mare

Fig. 2.2.9
Moduri de normalizare la orientare a unui dreptunghi

Cea de-a doua metodă presupune descrierea corpului prin punctele rezultate în urma aplicării unui rastru. Cel mai mic rastru aplicabil este de 4 pixeli, deoarece în urma detecției de contur grosimea conturului este de 2 pixeli. Numărul de puncte astfel rezultat este mult mai mic dar există posibilitatea unei erori de încadrare care să nu ducă la rezultat corect, în special în cazul corpurilor de dimensiuni mici.

Acest algoritm de orientare după latura mare se poate aplica cu succes şi la pătrate care nu sunt decât dreptunghiuri cu laturile egale, la cercuri la care nu contează rotația cât şi la triunghiuri. Acestea din urmă vor fi aliniate cu latura cea mai mare pe axa Ox (ceea ce nu este cazul la cele echilaterale, dar metoda se poate extinde şi la alte tipuri de triunghiuri). Ambele metode pot fi mult îmbunătățite din punct de vedere al vitezei pentru a fi compatibile cu calculul în timp real prin utilizarea tehnicilor de rotire din grafica 3D (exemplu: folosirea de tabele de sinusuri în loc de calculul efectiv al acestora). Implementarea acestui algoritm este prezentată în cadrul funcției GetShapeNumbers de la sfârşitul acestui capitol.

❖ **Normalizarea la mărime**

Faza următoare determinării unghiului de orientare şi rotirii adecvate, este generarea unui cod de înlănțuire de acelaşi ordin indiferent de dimensiunea corpului. Pentru a realiza această operație trebuie determinat un rastru de dimensiune variabilă, funcție de dimensiunea corpului, astfel încât ordinul numărului de formă rezultat să fie întotdeauna 20.

În acest scop se determină dreptunghiul de bază al corpului, adică dreptunghiul de dimensiuni minime care să cuprindă toate punctele corpului [26],[27], [28]. Deoarece corpul este deja orientat după latura mare, determinarea dreptunghiului este foarte simplă, el fiind generat de punctele extreme ale corpului.

Având determinat dreptunghiul de bază urmează determinarea excentricității acestuia ca fiind raportul dintre latura mare şi latura mică a acestuia. Cu excentricitatea calculată, se va determina dreptunghiul de ordinul 20 a cărui excentricitate aproximează cel mai bine dreptunghiul de bază. Există 5 dreptunghiuri de ordinul 20, şi anume: 1X9, 2X8, 3X7, 4X6 şi 5X5.

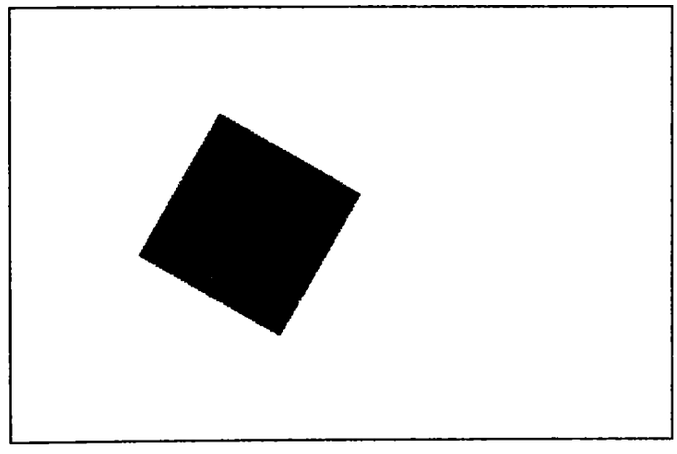
După determinarea excentricității standard se va calcula dimensiunea rastrului astfel încât dreptunghiul de bază să fie centrat pe acest rastru, şi în acest scop se va translata corpul astfel încât colțul stânga sus al acestuia să fie colțul dreapta jos al primului element din rastru.

❖ Normalizarea la punctul de start

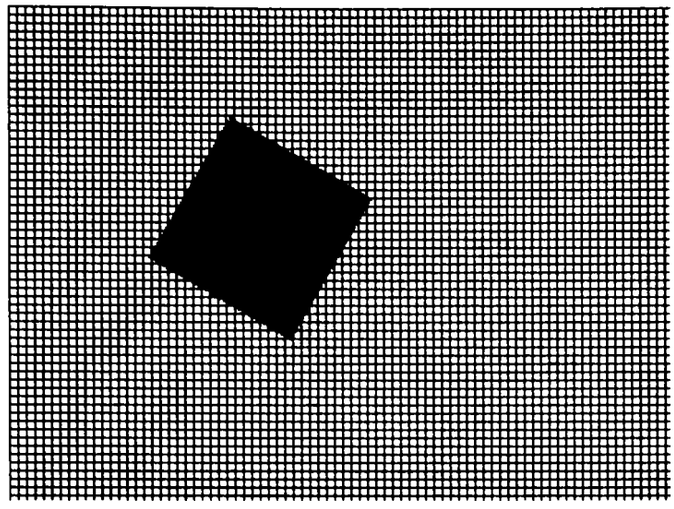
Este de remarcat faptul că, stabilirea codului de înlănțuire al unui anumit contur depinde de punctul de start. Este totuşi posibil să se normalizeze acest cod prin următoarea procedură: fiind dat un cod de înlănțuire generat prin pornirea dintr-un punct arbitrar, el se va trata ca o secvență circulară de numere de direcție şi se va redefini punctul de start în aşa fel încât secvența rezultată să formeze un număr de magnitudine minimă [26],[27],[28].

În locul acestui procedeu se poate utiliza normalizarea la orientare folosind prima diferență a codului de înlănțuire. Diferența este calculată simplu numărând (în sens antitrigonometricc) numărul de direcții care separă două elemente adiacente din cod. De exemplu, prima diferență a codului înlănțuit 10103322 este 3133030 [26].

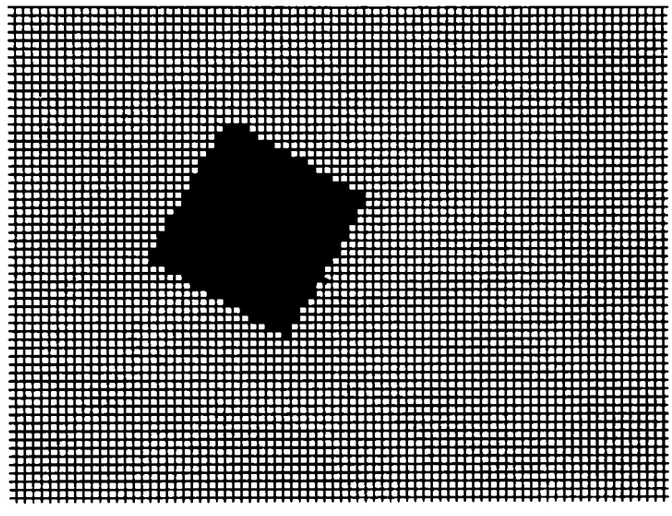
❖ Etapele procesului de descriere



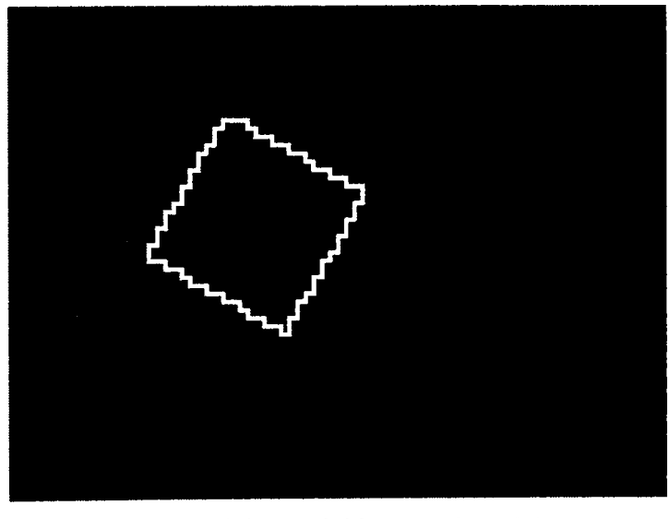
Sesizarea imaginii unui pătrat într-o poziție oarecare



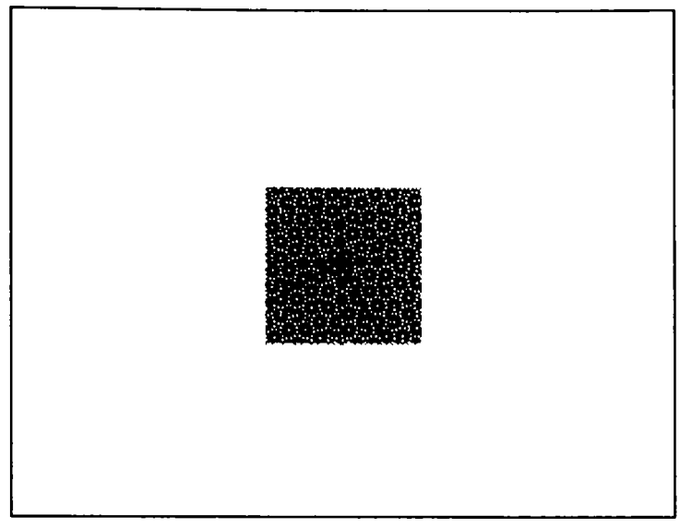
Selectarea unui rastru de dimensiune 4X4



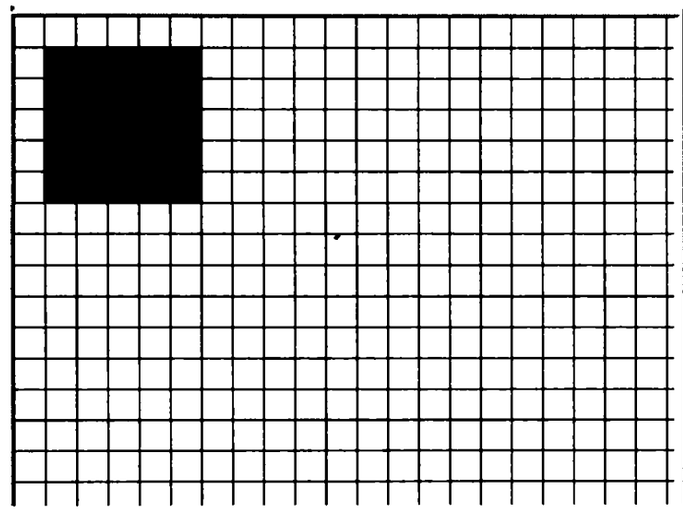
Rezultatul aplicării rastrului



Obținerea conturului corpului în urma detecției de contur



Rezultatul rotirii corpului după latura mare.
Din cauza utilizării numerelor naturale pentru pixeli, după rotire trebuie aplicat un filtru binar.



Rezultatul translatării corpului în colțul din dreapta jos al primului ochi de rastru, a cărui dimensiune s-a calculat astfel încât dreptunghiul de bază să aibă perimetru 5X5

Fig. 2.2.10
Prezentarea etapelor procesului de descriere

Funcția care implementează algoritmul de segmentare şi de generare a codului de înlănțuire este :

```
CROBOVIEWDoc  *CMainFrame::ChainCode(int height, int width, int treshold, BOOL
                                        NewDocument, CROBOVIEWDoc **ReturnSourceDoc)
{
 CROBOVIEWDoc       *SourceDoc;
 CROBOVIEWDoc       *DestinDoc;
 CROBOVIEWDoc       *OriginalDoc;
 CDC                *ActiveFrameDC;
 ChainCodeDlg        ChainCodeDialogBox;
 short               x,y,i,j,Xstart,Ystart,NumberOfBlackPixels;
 CPen                SolidRedPen(PS_SOLID,1,RGB(255,0,0));
 CPen               *OldPen;
 BOOL                EdgePixelFound = TRUE;
```

```
CString                 ChainCode;
unsigned char           LastDirection;
CCorp *                 CurrentCorp;
POINT                   point;
char                    TempString[4]={0,0,0,0};
BYTE                    BitmapMatrix[320][240];

if(height == -1 && width == -1 && treshold == -1)
 {
 width = height = 10;
 treshold = 50;
 ChainCodeDialogBox.m_treshold = treshold;
 itoa(height,TempString,10);
 ChainCodeDialogBox.m_height = TempString;
 itoa(width,TempString,10);
 ChainCodeDialogBox.m_width = TempString;
 ChainCodeDrawGrid(height,width);
 if(ChainCodeDialogBox.DoModal() == IDOK)
  {
  treshold = ChainCodeDialogBox.m_treshold;
  height = atoi(ChainCodeDialogBox.m_height);
  width = atoi(ChainCodeDialogBox.m_width);
  RedrawWindow(NULL,NULL,RDW_INVALIDATE | RDW_UPDATENOW );
  ChainCodeDrawGrid(height,width);
  }
 else
  return NULL;
 }
else
 ChainCodeDrawGrid(height,width);
OriginalDoc = SourceDoc = GetActiveSourceDocument();
if(NewDocument)
 DestinDoc = OpenNewDestinationDocument(SourceDoc->BitmapColorType);
else
 memset(&BitmapMatrix,0xFF,320*240);
for(x=0;x<Xmax;x+=width)
 for(y=0;y<REALYMAX;y+=height)
  {
  NumberOfBlackPixels = 0;
  for(i=0;i<width;i++)
   for(j=0;j<height;j++)
    {
    if(x+i<Xmax && y+j<REALYMAX)
     if(SourceDoc->BitmapMatrix[x+i][y+j][0] == 0)
      NumberOfBlackPixels++;
    }
  if(NumberOfBlackPixels >= (width*height*treshold)/100)
   for(i=0;i<width;i++)
    for(j=0;j<height;j++)
     {
     if(x+i<Xmax && y+j<REALYMAX)
      if(NewDocument)
       DestinDoc->BitmapMatrix[x+i][y+j][0] = DestinDoc->BitmapMatrix[x+i][y+j][1] =
                                              DestinDoc->BitmapMatrix[x+i][y+j][2] = 0;
      else
       BitmapMatrix[x+i][y+j] = 0;
     }
  else
   for(i=0;i<width;i++)
    for(j=0;j<height;j++)
```

```
      {
      if(x+i<Xmax && y+j<REALYMAX)
       if(NewDocument)
         DestinDoc->BitmapMatrix[x+i][y+j][0] = DestinDoc->BitmapMatrix[x+i][y+j][1] =
                                                DestinDoc->BitmapMatrix[x+i][y+j][2] = 0xFF;
       else
         BitmapMatrix[x+i][y+j] = 0xFF;
      }
   }
if(NewDocument)
 DestinDoc->BitmapColorType = BINARY;
else
 {
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
   SourceDoc->BitmapMatrix[x][y][0] = SourceDoc->BitmapMatrix[x][y][1] =
                                      SourceDoc->BitmapMatrix[x][y][2] = BitmapMatrix[x][y];
 }
GetActiveFrame()->RedrawWindow();
ActiveFrameDC = GetActiveFrame()->GetDC();
OldPen = ActiveFrameDC->SelectObject(&SolidRedPen);
for(x=1;x<Xmax;x+=width)
 {
 ActiveFrameDC->MoveTo(x,1);
 ActiveFrameDC->LineTo(x,REALYMAX);
 }
for(y=1;y<REALYMAX;y+=height)
 {
 ActiveFrameDC->MoveTo(1,y);
 ActiveFrameDC->LineTo(Xmax,y);
 }
ActiveFrameDC->SelectObject(OldPen);
ReleaseDC(ActiveFrameDC);
EdgeDetection(IDC_RADIO_FREEMAN,NewDocument);
SourceDoc = GetActiveSourceDocument();
while(EdgePixelFound)
 {
 EdgePixelFound = FALSE;
 for(x=width;x<Xmax;x+=width)
  {
  for(y=height;y<REALYMAX;y+=height)
   if(SourceDoc->BitmapMatrix[x][y][0] == 0xFF && !(SourceDoc->PixelBelongsToACorp(x,y)))
    {
    EdgePixelFound = TRUE;
     break;
     }
  if(EdgePixelFound)
   break;
  }
 if(!EdgePixelFound)
  break;
 CurrentCorp = SourceDoc->NewCorp();
 point.x = Xstart = x;
 point.y = Ystart = y;
 CurrentCorp->m_pointArray.Add(point);
 point.x += width;
 if(SourceDoc->BitmapMatrix[point.x][point.y][0] == 0xFF)
  CurrentCorp->m_pointArray.Add(point);
 ChainCode     = "0";
 while(point.x != Xstart || point.y != Ystart)
```

```
    {
    LastDirection = ChainCode[ChainCode.GetLength()-1];
    if(LastDirection != '2')
     if(SourceDoc->BitmapMatrix[point.x+width/2][point.y][0] == 0xFF)
      {
      point.x += width;
      CurrentCorp->m_pointArray.Add(point);
      ChainCode  += "0";
      continue;
      }
    if(LastDirection != '0')
     if(SourceDoc->BitmapMatrix[point.x-width/2][point.y][0] == 0xFF)
      {
      point.x -= width;
      CurrentCorp->m_pointArray.Add(point);
      ChainCode  += "2";
      continue;
      }
    if(LastDirection != '3')
     if(SourceDoc->BitmapMatrix[point.x][point.y-height/2][0] == 0xFF)
      {
       point.y -= height;
       CurrentCorp->m_pointArray.Add(point);
       ChainCode += "1";
       continue;
       }
    if(LastDirection != '1')
     if(SourceDoc->BitmapMatrix[point.x][point.y+height/2][0] == 0xFF)
      {
      point.y += height;
      CurrentCorp->m_pointArray.Add(point);
      ChainCode  += "3";
      continue;
      }
    }
   CurrentCorp->m_ChainCode = ChainCode;
   }
 *ReturnSourceDoc = SourceDoc;
 return OriginalDoc;
}
```

Funcția care implementează extragerea numerelor de formă este prezentată în continuare:

```
void CROBOVIEWDoc::GetShapeNumbers(CROBOVIEWDoc  *OriginalDoc)
{
 POSITION  CurrentPosition,DestinCurrentPosition;
 CCorp       *CurrentCorp,*DestinCurrentCorp;
 int           i,j,x,y,NumberOfPoints;
 double       Angle;
 POINT       RotationCenter;
 struct        YDistance{
                        POINT Point1;
                        POINT Point2;
                        double Angle;
                        }MaximY,MinimY;
 struct        BASICRECTANGLE{
                                RECT   Rect;
```

```
                                    int   MajorAxis;
                                    int   MinorAxis;
                                    double Eccentricity;
                                    }BasicRectangle,BestMatchRectangle;
POINT       TestPoint_i,TestPoint_j,TestPoint_k,Translate,GridSize;
CROBOVIEWDoc        *DestinDoc;
CROBOVIEWDoc        *TempDoc;
int           ShapeNumberOrder = 20;
double        MinimDifference;
BYTE          BitmapMatrix[320][240];

DestinDoc = ((CMainFrame*)AfxGetMainWnd())->OpenNewDestinationDocument(BINARY);
CurrentPosition = m_CorpList.GetHeadPosition();
while(CurrentPosition)
 {
 MaximY.Point1.x = 0;
 MinimY.Point1.x = 10000;
 MaximY.Point1.y = 0;
 MinimY.Point1.y = 10000;
 MaximY.Point2.x = MinimY.Point2.x = 0;
 MaximY.Point2.y = MinimY.Point2.y = 0;
 MaximY.Angle   = MinimY.Angle    = 0;
 BasicRectangle.Rect.left  = 10000;
 BasicRectangle.Rect.right  = 0;
 BasicRectangle.Rect.top   = 10000;
 BasicRectangle.Rect.bottom = 0;
 CurrentCorp = m_CorpList.GetNext(CurrentPosition);
 NumberOfPoints = CurrentCorp->m_pointArray.GetSize();
 //Determine bounding rectangle of the given object
 for(i=0;i<NumberOfPoints;i++)
  {
  if(CurrentCorp->m_pointArray[i].x < BasicRectangle.Rect.left)
   BasicRectangle.Rect.left = CurrentCorp->m_pointArray[i].x;
  if(CurrentCorp->m_pointArray[i].x > BasicRectangle.Rect.right)
   BasicRectangle.Rect.right = CurrentCorp->m_pointArray[i].x;
  if(CurrentCorp->m_pointArray[i].y < BasicRectangle.Rect.top)
   BasicRectangle.Rect.top = CurrentCorp->m_pointArray[i].y;
  if(CurrentCorp->m_pointArray[i].y > BasicRectangle.Rect.bottom)
   BasicRectangle.Rect.bottom = CurrentCorp->m_pointArray[i].y;
  }
 //Determine the rotaion center of the bounding rectangle of the given object
 RotationCenter.x = (int)((BasicRectangle.Rect.left+BasicRectangle.Rect.right)/2);
 RotationCenter.y = (int)((BasicRectangle.Rect.top +BasicRectangle.Rect.bottom)/2);
 //Determine the angle of rotation so thet the major axis will be along x and the minor axis along y
 for(Angle = 0;Angle < PI;Angle += 0.02)
  {
  memset(&MaximY,0,sizeof(YDistance));
  for(i=0;i<NumberOfPoints-1;i++)
   for(j=i+1;j<NumberOfPoints;j++)
    {
    TestPoint_i = Rotate(CurrentCorp->m_pointArray[i],Angle,RotationCenter);
    TestPoint_j = Rotate(CurrentCorp->m_pointArray[j],Angle,RotationCenter);
    if(abs(MaximY.Point2.y-MaximY.Point1.y) < abs(TestPoint_i.y-TestPoint_j.y))
     {
     MaximY.Point1.y = TestPoint_i.y;
     MaximY.Point2.y = TestPoint_j.y;
     }
    }
  if(abs(MaximY.Point2.y-MaximY.Point1.y) <= abs(MinimY.Point2.y-MinimY.Point1.y))
   {
```

```
    MinimY.Point1.y = MaximY.Point1.y;
    MinimY.Point2.y = MaximY.Point2.y;
    MinimY.Angle    = Angle;
    }
   }
//Rotate all the pixels of the given corp with the selected angle and translate them so that the rotation
center will be in the center of the bitmap
TestPoint_k.x = Xmax/2 - RotationCenter.x;
TestPoint_k.y = REALYMAX/2 - RotationCenter.y;
memset(&DestinDoc->BitmapMatrix,0xFF,320*240*3);
for(TestPoint_i.x=BasicRectangle.Rect.left-4; TestPoint_i.x<=BasicRectangle.Rect.right+4;
                                                                          TestPoint_i.x++)
  for(TestPoint_i.y=BasicRectangle.Rect.top-4; TestPoint_i.y<=BasicRectangle.Rect.bottom+4;
                                                                          TestPoint_i.y++)
    {
    if(OriginalDoc->BitmapMatrix[TestPoint_i.x][TestPoint_i.y][0] == 0)
      {
      TestPoint_j = Rotate(TestPoint_i,MinimY.Angle,RotationCenter);
      TestPoint_j.x += TestPoint_k.x;
      TestPoint_j.y += TestPoint_k.y;
      DestinDoc->BitmapMatrix[TestPoint_j.x][TestPoint_j.y][0] =
        DestinDoc->BitmapMatrix[TestPoint_j.x][TestPoint_j.y][1] =
        DestinDoc->BitmapMatrix[TestPoint_j.x][TestPoint_j.y][2] = 0;
      }
    }
((CMainFrame*)AfxGetMainWnd())->SmoothingBinaryImages(FALSE);
 //Determine the basic rectangle of the given object.
 BasicRectangle.Rect.left   = 10000;
 BasicRectangle.Rect.right  = 0;
 BasicRectangle.Rect.top    = 10000;
 BasicRectangle.Rect.bottom = 0;
 for(x=0;x<Xmax;x++)
   for(y=0;y<REALYMAX;y++)
     {
     if(DestinDoc->BitmapMatrix[x][y][0] == 0)
       {
       if(x < BasicRectangle.Rect.left)
         BasicRectangle.Rect.left = x;
       if(x > BasicRectangle.Rect.right)
         BasicRectangle.Rect.right = x;
       if(y < BasicRectangle.Rect.top)
         BasicRectangle.Rect.top = y;
       if(y > BasicRectangle.Rect.bottom)
         BasicRectangle.Rect.bottom = y;
       }
     }

 BasicRectangle.MajorAxis = BasicRectangle.Rect.bottom-BasicRectangle.Rect.top;
 BasicRectangle.MinorAxis = BasicRectangle.Rect.right-BasicRectangle.Rect.left;
 BasicRectangle.Eccentricity= (double)BasicRectangle.MajorAxis/
                              (double)BasicRectangle.MinorAxis;
 //Determine the best match rectangle of order 20 of the given object
 MinimDifference = 10000;
 for(i=1;i<=ShapeNumberOrder/2-i;i++)
   {
   if((double)i/(double)((double)ShapeNumberOrder/2-(double)i)-BasicRectangle.Eccentricity <
                                                                          MinimDifference)
     {
     BestMatchRectangle.MinorAxis = i;
     BestMatchRectangle.MajorAxis = ShapeNumberOrder/2-i;
```

```
    BestMatchRectangle.Eccentricity = (double)i/(double)((double)ShapeNumberOrder/2-(double)i);
    MinimDifference  = fabs(BestMatchRectangle.Eccentricity-BasicRectangle.Eccentricity);
    }
   }
  GridSize.x = round((double)((double)(BasicRectangle.Rect.right-
                                BasicRectangle.Rect.left)/(double)BestMatchRectangle.MajorAxis));
  GridSize.y = round((double)((double)(BasicRectangle.Rect.bottom-
                                BasicRectangle.Rect.top)/(double)BestMatchRectangle.MinorAxis));
  memset(&BitmapMatrix,0xFF,320*240);
  //Translate all the pixels of the object to the first grid
  Translate.x = BasicRectangle.Rect.left-GridSize.x;
  Translate.y = BasicRectangle.Rect.top -GridSize.y;
  for(x=0;x<Xmax;x++)
   for(y=0;y<REALYMAX;y++)
    if(DestinDoc->BitmapMatrix[x][y][0] == 0)
     BitmapMatrix[x-Translate.x][y-Translate.y] = 0;
   for(x=0;x<Xmax;x++)
    for(y=0;y<REALYMAX;y++)
     DestinDoc->BitmapMatrix[x][y][0]=DestinDoc->BitmapMatrix[x][y][1] =
                                        DestinDoc->BitmapMatrix[x][y][2]=BitmapMatrix[x][y];
  ((CMainFrame*)AfxGetMainWnd())->ChainCode(GridSize.y,GridSize.x,50,FALSE,&TempDoc);
  DestinCurrentPosition = DestinDoc->m_CorpList.GetHeadPosition();
  DestinCurrentCorp = DestinDoc->m_CorpList.GetNext(DestinCurrentPosition);
  DestinCurrentCorp->m_FirstDifference = NormalizeChainCodeForRotation(
                                                          DestinCurrentCorp->m_ChainCode);
  DestinCurrentCorp->m_ShapeNumber = NormalizeChainCodeForStartingPoint(
                                                  DestinCurrentCorp->m_FirstDifference);
  DestinCurrentCorp->m_Geometric = Recognition(DestinCurrentCorp->m_ShapeNumber);
  CurrentCorp->m_ChainCode = DestinCurrentCorp->m_ChainCode;
  CurrentCorp->m_FirstDifference = DestinCurrentCorp->m_FirstDifference;
  CurrentCorp->m_ShapeNumber = DestinCurrentCorp->m_ShapeNumber;
  CurrentCorp->m_Geometric = DestinCurrentCorp->m_Geometric;
  DestinDoc->DeleteCorps();
  }
}
```

➢ Descriptori Fourier

Transformata Fourier discretă, unidimensională poate fi deseori folosită pentru a descrie un contur bidimensional [6], [10], [12], [19], [30], [96], [102], [103], [114], [118], [134], [140], [142], [156].

Se presupune că sunt disponibile M puncte ale conturului. Dacă, aşa cum se arată în figura 2.2.11, se vizualizează acest contur ca fiind plasat în planul complex, fiecare punct de contur bidimensional (x, y) fiind redus la un număr complex $x + iy$. Secvența de puncte de-a lungul conturului formează o funcție a cărei transformată Fourier este $F(u)$, $u = 0, 1, 2, ..., M-1$. Dacă M este o putere întreagă a lui 2, $F(u)$ poate fi calculată folosind un algoritm TFR.

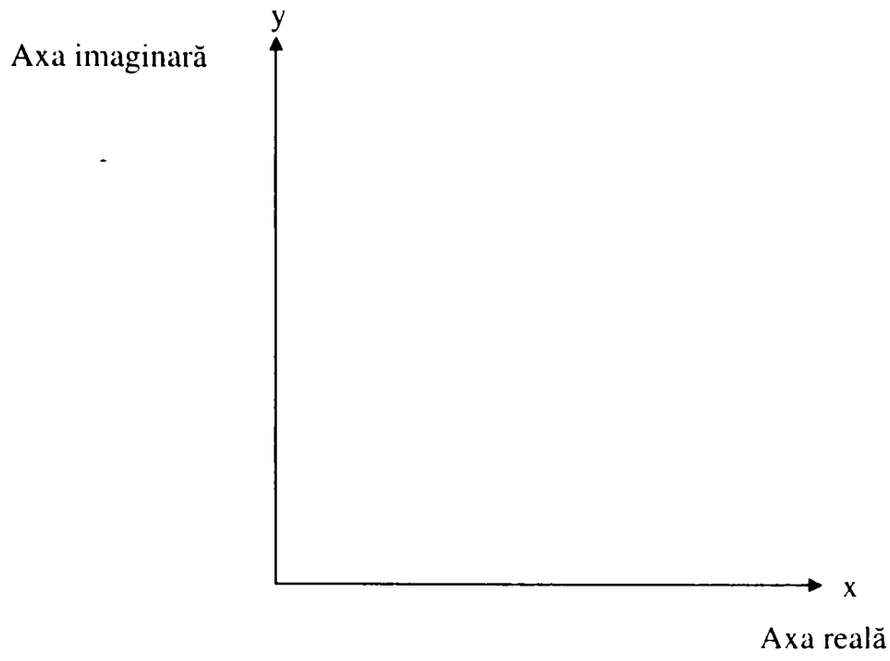


Fig. 2.2.11
Reprezentarea unei regiuni de contur în domeniul de frecvență

Motivul pentru care se face această abordare a problemei este faptul că doar primele câteva componente ale lui $F(u)$,sunt în general necesare pentru a distinge între forme care sunt distincte într-un mod rezonabil.

Transformata Fourier este uşor de normalizat relativ la dimensiune, rotație şi punct de start pentru un contur.

Pentru a schimba dimensiunea unui contur se vor înmulți componentele lui $F(u)$ cu o constantă. Se poate demonstra faptul că deplasarea punctului de start al conturului în domeniul spațial corespunde înmulțirii componentei k a lui $F(u)$ cu e^{ikT} unde T este în intervalul $[0,2\pi]$. Când T se deplasează de la 0 la 2π, punctul de start traversează întregul contur o dată. Această informație poate fi folosită drept bază pentru normalizare.

2.2.2. Descriptorii regiunilor

O regiune de interes poate fi descrisă de forma conturului sau de caracteristicile interne [4], [5], [16], [89], [94], [135], [136], [146], [150].

Astfel, un mare număr de sisteme de vedere artificiale industriale existente sunt bazate pe descriptori regionali care sunt simpli şi astfel au o mare atractivitate din punct de vedere al calculului. Utilizarea acestor descriptori este limitată la situații în

care obiectele de interes sunt foarte distincte încât câțiva descriptori globali sunt suficienți pentru caracterizarea lor.

Aria unei regiuni este definită ca numărul de pixeli conținuți în conturul ei. Aceasta este-un descriptor foarte util când geometria de vizionare este fixă și obiectele sunt întotdeauna analizate aproximativ la aceeași distanță față de cameră. O aplicație tipică este recunoașterea obiectelor ce se mișcă pe o bandă rulantă prin fața camerei video.

Axa mare și ***axa mică*** a unei regiuni sunt definite în funcție de contur și sunt utile pentru determinarea orientării unui obiect. Raportul lungimilor acestor axe, numit ***excentricitatea*** regiuni, este de asemenea un important descriptor global al formei sale.

Perimetrul unei regiuni este lungimea conturului acelei regiuni. Deși uneori el este utilizat ca și descriptor, cea mai frecventă aplicație a sa este stabilirea unei măsuri a ***compactității*** unei regiuni, definită ca perimetru^2/arie. Compactitatea este cantitate adimensională (fiind astfel este insensibilă la schimbări de scală) și este minimă pentru o regiune în formă de disc.

O ***regiune conexă*** este o regiune în care toate perechile de puncte pot fi conectate de o curbă ce se află complet în interiorul regiunii. Pentru un set de regiuni conexe, unele care pot să aibă găuri, este util să considerăm ***numărul lui Euler*** ca pe un descriptor. Numărul lui Euler este definit simplu ca numărul de regiuni conexe minus numărul de găuri. De exemplu, numerele lui Euler ale literelor A și B sunt 0 respectiv -1. În continuare se vor discuta și alți descriptori ai regiunilor [111], [119], [152].

- **Textura**

Identificarea obiectelor sau regiunilor dintr-o imagine poate fi deseori realizată, cel puțin parțial, cu ajutorul descriptorilor de textură [8]. Deși nu există o definiție formală a texturii, se poate imagina acest descriptor ca realizând măsurări cantitative ale proprietăților cum ar fi netezimea, asprimea, și regularitatea.

Principalele metode de abordare ale descriptorilor de textură sunt statistice și de structură. Metodele statistice vor genera caracteristici ale texturii ca fiind netede, aspre, granulare, și așa mai departe. Tehnicile structurale, tratează aranjamentul imaginilor primitive, cum ar fi descrierea texturii bazate pe linii paralele spațiate regular.

Una din cele mai simple abordări pentru descrierea texturii este utilizarea momentelor histogramei de intensitate a unei imagini sau regiuni.

Fie z o variabilă aleatoare care reprezintă intensitatea discretă a imaginii, şi fie $p(z_i)$, $i = 1, 2, ..., L$, histograma corespunzătoare, unde L este numărul de nivele de intensitate distincte. Al n-lea moment al lui z deasupra mediei este definit ca

$$\mu_n(z) = \sum_{i=1}^{L} (z_i - m)^n p(z_i) \tag{2.2.3}$$

unde m este valoarea medie a lui z (intensitate medie a imaginii):

$$m = \sum_{i=1}^{L} z_i p(z_i) \tag{2.2.4}$$

Se observă din relaţia (2.2.3) că $\mu_0 = 1$ şi $\mu_1 = 0$.

Momentul de ordinul 2 este de o importanţă particulară în descriptorii de textură. El este o măsură a contrastului de intensitate care poate fi folosit pentru determinarea descriptorilor de netezime relativă. De exemplu, măsura sa :

$$R = 1 - \frac{1}{1 + \sigma^2(z)} \tag{2.2.5}$$

este 0 pentru zonele de intensitate constantă ($\sigma^2(z) = 0$ dacă toţi z_i au aceeaşi valoare) şi se apropie de 1 pentru valori mari ale lui $\sigma^2(z)$. Momentul de ordinul 3 este o măsură a înclinării histogramei, în timp ce momentul de ordinul 4 este o măsură a planeităţii relative. Momentele de ordinul 5 şi cele mai mari nu sunt atât de simplu de legat de forma histogramei, dar ele pot introduce diferenţe cantitative ulterioare ale texturii.

Măsurile calculate ale texturii folosind histograme, sunt limitate de faptul că nu conţin informaţii referitoare la poziţia relativă a pixelilor unul faţă de altul. Una din modalităţile prin care se poate aduce această informaţie în analiza de textură este luarea în considerare nu doar a distribuţiei intensităţii dar şi a poziţiei pixelilor cu intensităţi egale sau aproape egale.

Fie P un operator de poziţie şi fie A o matrice k x k ale cărei elemente a_{ij} arată de câte ori punctele cu intensitatea z_i apar (în poziţia specificată de P) faţă de punctele cu intensitatea z_j, cu $1 \le i, j \le k$. De exemplu, se consideră o imagine cu trei intensităţi, $z_1 = 0$, $z_2 = 1$, şi $z_3 = 2$:

0 0 0 1 2

1 1 0 1 1

2 2 1 0 0

1 1 0 2 0

0 0 1 0 1

Dacă se defineşte operatorul de poziţie *P* ca "un pixel la dreapta şi un pixel în jos", atunci se va obţine următoarea matrice *A* 3x3:

$$A=\begin{bmatrix}4 & 2 & 1\\ 2 & 3 & 2\\ 0 & 2 & 0\end{bmatrix}$$

unde, a_{11} (colţul stânga sus) arată de câte ori un punct cu nivel de intensitate $z_1 = 0$ apare cu o locaţie de pixel mai jos şi mai la dreapta de un pixel cu aceeaşi intensitate, în timp ce a_{13} (colţul dreapta sus) arată de câte ori un punct cu nivelul $z_1 = 0$ apare mai jos şi mai la dreapta de un punct cu intensitatea $z_3 = 2$.

Este important de remarcat faptul că dimensiunea lui ***A*** este determinată strict de numărul de intensităţi distincte din imaginea de intrare. Problema este realizarea matricei *A* cu dimensiuni cât mai mici.

Fie *n* numărul total de perechi de puncte din imagine care satisface *P* (în exemplul anterior *n*=16). Dacă se defineşte o matrice ***C*** formată prin divizarea fiecărui element a lui ***A*** cu *n*, atunci c_{ij} este o estimare a probabilităţii de unire ca o pereche de puncte satisfăcând *P*, să aibă valorile (z_i, z_j).

Matricea ***C*** este denumită ***matrice de concurenţă a nivelelor de gri***, unde prin nivel de gri se înţelege intensitatea unui pixel monocrom sau a imaginii. Deoarece ***C*** depinde de *P*, este posibil să se detecteze prezenţa unor modele de textură date prin alegerea unui operator de poziţie adecvat. De exemplu, operatorul folosit în exemplul anterior este sensibil la benzi de intensitate constantă ce se întind la -45° (de remarcat că cea mai mare valoarea din ***A*** era $a_{11} = 4$, parţial datorită liniei de puncte cu intensitate 0 ce se află la -45°).

- **Scheletul unei regiuni**

O metodă importantă pentru a reprezenta forma structurală a unei regiuni plane este reducerea ei la un graf [3], [13], [18], [97]. Acest lucru se realizează prin

obținerea ***scheletului*** regiunii prin algoritm de subțiere. Procedurile de subțiere joacă un rol central într-o mare gamă de probleme din vederea artificială.

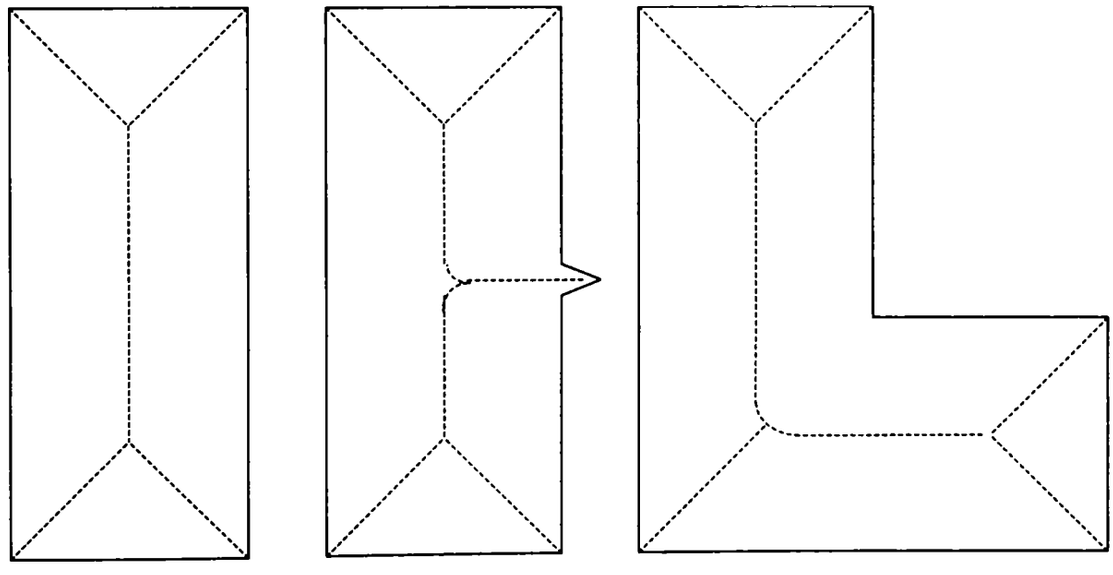
Scheletul unei regiuni poate fi definit cu ajutorul ***transformatei axelor mediane (TAM)***. Transformarea TAM a unei regiuni R cu marginea B se realizează astfel :

Pentru fiecare punct p din R, se va găsi cel mai apropiat vecin din B. Dacă p are mai mulți astfel de vecini, atunci se spune că aparține axelor mediane (scheletului) lui R. Se observă că formularea "cel mai apropiat" depinde de definiția distanței și, astfel, rezultatele unei operații TAM vor fi influențate de alegerea unei metrici. Câteva exemple folosind distanța euclidiană sunt prezentate in figura 2.2.12.

Deşi algoritmul TAM generează un schelet foarte bun, o implementare directă a definiției de mai sus este incompatibilă din punct de vedere al calculului necesar, impunându-se calcularea distanței de la fiecare punct interior la fiecare punct de pe contur.

Au fost propuşi mai mulți algoritmi pentru îmbunătățirea eficienței calculului şi, în acelaşi timp, s-a încercat reprezentarea median axială a unei regiuni date. În mod normal, aceşti algoritmi de subțiere care şterg iterativ punctele de pe marginea unei regiuni pornind de la restricția că ştergând aceste puncte :

1. nu se vor elimina şi punctele finale
2. nu se întrerupe conectivitatea
3. nu se va cauza o eroziune excesivă a regiunii



(a) (b) (c)

Fig. 2.2.12
Axele mediane a trei regiuni simple.

Deşi s-au făcut încercări de a aplica aceşti algoritmi imaginilor cu nivele de gri, totuşi acest tip de reprezentare este de obicei folosit imaginilor binare.

Procedura prezentată în continuare este rapidă, are o implementare liniară şi va genera schelete care sunt, în multe cazuri, superioare celor obţinute cu algoritmi de subţiere.

Algoritmul necesită prin câteva definiţii :

- Se presupune că există date binare, punctele regiunii fiind marcate cu 1 iar fondul cu 0. Acestea vor fi numite puncte ***întunecate*** şi respectiv ***luminoase***
- Un ***punct de margine*** este un punct întunecat care are cel puţin un vecin de tip 4 luminos
- Un ***punct final*** este un punct întunecat care are unul şi numai unul din vecinii de tip 8 întunecat
- Un ***punct de întrerupere*** este un punct întunecat a cărui ştergere va întrerupe conectivitatea.

La algoritmii de subţiere, zgomotul şi orice variaţie eronată de-a lungul conturului poate modifica semnificativ scheletul rezultat (figura 2.2.12,b este un exemplu elocvent). În consecinţă se presupune, a priori, că toate contururile regiunilor au fost netezite anterior de subţiere.

Făcând referire la aranjamentul vecinilor prezentat în figura 2.2.13, algoritmul de subţiere identifică un punct de margine p ca pe un punct ce aparţine unuia sau mai multora dintre tipurile următoare:

1. un ***punct de margine stânga*** ce are vecinul din stânga n_4 luminos
2. un ***punct de margine dreapta*** ce are n_0 luminos
3. **un *punct de margine sus*** ce are n_2 luminos
4. un ***punct de margine jos*** ce are n_6 luminos

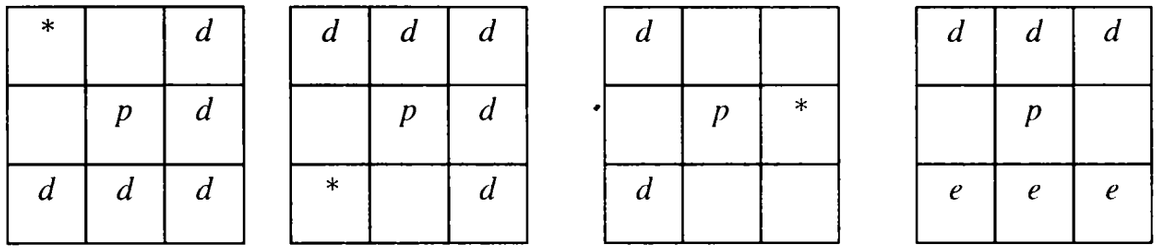
Este posibil ca p să fie clasificat în mai multe din aceste tipuri deodată. De exemplu, un punct întunecat p ce are n_0 şi n_4 luminoşi, va fi un punct de margine dreapta şi stânga simultan.

În continuare procedura începe cu identificarea punctelor de margine stânga care trebuie şterse. Ea va continua în acelaşi mod şi cu celelalte tipuri.

n_3	n_2	n_1
n_4	p	n_0
n_5	n_6	n_7

Fig. 2.2.13
Notarea vecinilor lui p folosită în algoritmul de subțiere.

Un punct de margine este marcat dacă nu este un punct final sau unul de întrerupere sau dacă ştergerea lui va produce eroziune excesivă. Testele pentru aceste condiții sunt realizate prin compararea vecinătăților 8 a lui p cu ferestrele prezentate în figura 2.2.14, unde p şi asterixul sunt puncte întunecate iar d şi e sunt puncte "orice", adică pot fi şi întunecate şi luminoase.



(a) (b) (c) (d)

Fig. 2.2.14
Dacă vecinătatea de tip 8 a unui punct p corespunde uneia din ferestrele de mai sus, punctul p nu va fi marcat. Asterixul înseamnă un punct întunecat, iar d şi e pot fi fie întunecate fie luminoase.

Dacă vecinătatea lui p corespunde cu ferestrele (a) ... (c), pot apărea două cazuri :

1. dacă toate punctele d sunt luminoase, atunci p este un punct final,
2. dacă cel puțin un punct d este întunecat, atunci p este un punct de întrerupere

În ambele cazuri p nu trebuie marcat.

Analiza ferestrei (d) este puțin mai complicată. Dacă cel puțin unul dintre d şi e este întunecat, atunci p este un punct de întrerupere şi nu trebuie marcat.

Trebuie luate în considerare totuşi şi alte aranjamente. Se presupune că toate punctele d sunt luminoase şi că toate punctele e pot fi orice. Această condiție va genera 8 posibilități prezentate în figura 2.2.15.

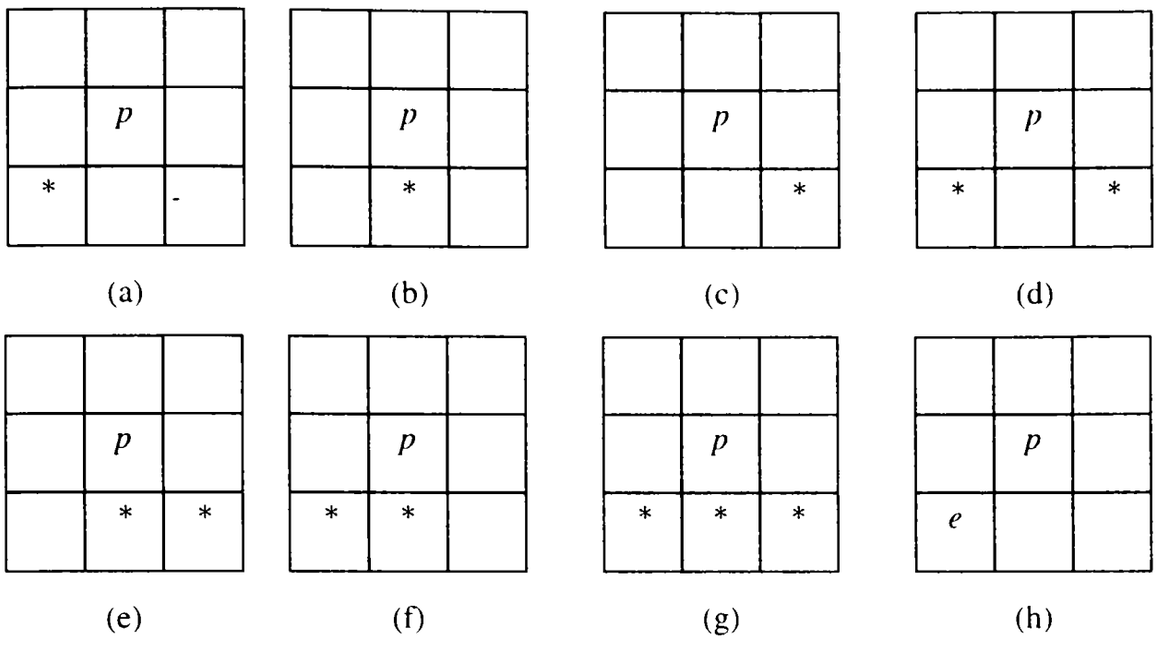


Fig. 2.2.15
Toate configurațiile posibile dacă *d* este luminos în figura 22, iar e poate fi întunecat, *, sau luminos.

Configurațiile (a) ... (c) îl fac pe *p* punct final, iar configurația (d) îl face punct de întrerupere.

Dacă *p* ar fi şters în configurațiile (e) şi (f), este foarte uşor de arătat faptul că ştergerea lui va cauza o eroziune prea mare în regiunile înclinate de lățime 2. În configurația (g), *p* este un punct ce se numeşte ***umflătură***, de obicei datorat unei ieşituri a regiunii. Deoarece se presupune că regiunea a fost inițial netezită, apariția unei umflături în timpul subțierii este considerată importantă pentru formă, şi *p* nu va trebui şters.

În final, dacă toate punctele izolate au fost inițial eliminate, apariția unei configurații ca în (h) pe timpul procesului de subțiere indică faptul că regiunea a fost redusă la un singur punct. Ştergerea lui va elimina ultima bucată existentă de regiune.

Argumente similare se pot aduce dacă rolurile lui *d* şi *e* ar fi interschimbate. Important este că oricare punct de margine stânga *p* a cărui vecinătate 8 coincide cu una din ferestrele din figura 2.2.14, nu trebuie marcat.

Testarea vecinătății 8 a lui *p* cu ferestrele din figura 2.2.14 are o reprezentare booleană particulară şi simplă dată de

$$B_4 = n_0 \bullet (n_1 + n_2 + n_6 + n_7) \bullet (n_2 + \overline{n_3}) \bullet (\overline{n_5} + n_6) \qquad (2.2.6)$$

unde :

- indicele lui B indică faptul că n_4 este luminos (p este un punct de margine stânga)
- "$\bullet$" este operația logică "şi"
- "+" este operația logică "SAU"
- "$\bar{}$" este complementul logic
- n-urile sunt definite ca în figura 2.2.13.

Ecuația (2.2.6) este evaluată prin atribuirea punctelor întunecate, ***punctele nemarcate anterior*** să ia valoarea 1, iar cele luminoase sau marcate să ia valoare 0. Astfel dacă B_4 este 1, se va marca pe p. În caz contrar, p rămâne nemarcat.

Expresii similare se obțin pentru punctele de margine dreapta,

$$B_6 = n_4 \bullet (n_2 + n_3 + n_5 + n_6) \bullet (n_6 + \overline{n_7}) \bullet (\overline{n_1} + n_2) \tag{2.2.7}$$

pentru punctele de margine sus,

$$B_2 = n_6 \bullet (n_0 + n_4 + n_5 + n_7) \bullet (n_0 + \overline{n_1}) \bullet (\overline{n_3} + n_4) \tag{2.2.8}$$

şi pentru punctele de margine jos,

$$B_2 = n_2 \bullet (n_0 + n_1 + n_3 + n_4) \bullet (n_4 + \overline{n_5}) \bullet (\overline{n_0} + n_7) \tag{2.2.9}$$

Utilizând expresiile (2.2.6)-(2.2.9), algoritmul de subțiere va realiza două scanări ale imaginii. Secvența de scanare se realizează fie lungul rândurilor, fie de-a lungul coloanelor, dar alegerea va afecta în general rezultatul final. În prima scanare se vor folosi B_4 şi B_0 pentru a marca punctele de margine stânga şi dreapta; în cea de-a doua scanare se vor folosi B_2 şi B_6 pentru a marca punctele de margine sus şi jos.

Dacă nu au fost marcate noi puncte de margine în timpul celor două scanări, algoritmul se opreşte, iar punctele nemarcate constituie scheletul; în caz contrar, procedura se repetă.

Este de subliniat faptul că punctele marcate anterior sunt tratate ca 0 în evaluarea expresiilor booleene.

O procedură alternativă este setarea oricărui punct marcat la 0 pe timpul execuției algoritmului, astfel rezultând doar scheletul şi punctele de fond în final. Acest procedeu este mai uşor de implementat, dar pierde toate celelalte punctele din regiune.

2.3. Recunoaşterea

Recunoaşterea este un proces de etichetare, ce urmăreşte identificarea fiecărui obiect segmentat dintr-o scenă şi atribuirea unei etichete acelui obiect. În cele mai multe cazuri, etapa de recunoaştere în sistemele industriale de vedere artificială existente, se bazează pe presupunerea că obiectele din scenă au fost segmentate ca unităţi individuale. O altă restricţie firească în aplicaţii impune ca imaginile să fie achiziţionate într-o geometrie de vedere cunoscută (de obicei perpendiculară pe spaţiul de lucru). Acest lucru micşorează variaţia de caracteristici de formă şi simplifică segmentarea şi descrierea reducând posibilitatea suprapunerii. Variaţia în orientarea obiectelor este tratată prin alegerea de descriptori invarianţi la rotaţie sau prin utilizarea axei principale a unui obiect pentru al orienta într-o direcţie predefinită.

Metodele de recunoaştere utilizate în prezent sunt [23], [29], [91], [107], [110], [113], [121], [132], [139], [149], [153], [154] :

- ***de decizii teoretice***
- ***de decizie structurală***

Metodele de decizie teoretică sunt bazate pe descriptori cantitativi (textură statistică), în timp ce metodele structurale se bazează pe descriptori simbolici şi pe relaţiile lor (secvenţa direcţiilor dintr-un contur codat înlănţuit). Cu câteva excepţii, procedurile prezentate în acest capitol sunt în general utilizate pentru recunoaşterea obiectelor într-o reprezentare bidimensională.

2.3.1. Metode de decizie teoretică

Recunoaşterea modelelor prin decizie teoretică este bazată pe utilizarea ***funcţiilor de decizie*** (*discriminatorii*).

Fie $x = (x_1, x_2, ..., x_n)^T$ un ***vector model coloană*** cu componente reale, unde x_i este al i-lea descriptor al unui obiect dat (arie, media intensităţii, perimetru).

Fiind dată o clasă de obiecte M, dată de $w_1, w_2, ..., w_M$, problema de bază în recunoaşterea modelelor prin decizie teoretică este identificarea a M funcţii de decizie, $d_1(x), d_2(x), ..., d_M(x)$, cu proprietatea că :

$$d_i(x^*) > d_j(x^*) \qquad j = 1, 2, ..., M; \qquad j \neq i \qquad (2.3.1)$$

relaţie care este valabilă pentru orice vector model x^* ce aparţine clasei w_i.

Adică, un obiect necunoscut reprezentat de vectorul x* este recunoscut ca aparținând clasei de obiecte *i* dacă, după înlocuirea lui x* în toate funcțiile de decizie, $d_i(x^*)$ produce cea mai mare valoare.

Utilitatea cea mai mare a funcțiilor de decizie în sistemele de vedere artificială industriale este ***potrivirea***. Se presupune că fiecare clasă de obiecte este reprezentată printr-un vector ***prototip*** (sau ***medie***):

$$m_i = \frac{1}{N}\sum_{k=1}^{N} x_k \qquad i = 1, 2, ..., M \tag{2.3.2}$$

unde : x_k sunt mostre de vectori despre care se știe că aparțin clasei w_i.

Fiind dat un vector necunoscut x*, una din modalitățile de a determina apartenența lui la o clasă este să i se atribuie clasa prototipului celui mai apropiat de el. Dacă se folosește distanța euclidiană pentru a determina apropierea, problema se reduce la calcularea mărimii distanței :

$$D_j(x^*) = \| x^* - m_j \| \qquad j = 1, 2, ..., M \tag{2.3.3}$$

unde $\| a \| = (a^T a)^{1/2}$ este norma euclidiană. •

După aceea se atribuie x* clasei w_i dacă $D_i(x^*)$ este distanța cea mai mică. Nu este dificil de arătat că aceasta este echivalentă cu evaluarea funcțiilor :

$$d_j(x^*) = (x^*)^T m_j - 1/2 m_j^T m_j \qquad j = 1, 2, ..., M \tag{2.3.4}$$

selectând valoarea cea mai mare.

Această formulare este în concordanță cu conceptele funcțiilor de decizie, așa cum sunt definite în relația (2.3.1).

O altă aplicație de potrivire este căutarea aparițiilor unei subimagini $w(x, y)$ într-o imagine mai mare $f(x, y)$. La fiecare locație (x, y) a lui $f(x, y)$ se definește ***coeficientul de corelație*** ca fiind

$$\gamma(x,y) = \frac{\sum_s \sum_t \left[w(s,t) - m_w\right]\left[f(s,t) - m_f\right]}{\left\{\sum_s \sum_t \left[w(s,t) - m_w\right]^2 \sum_s \sum_t \left[f(s,t) - m_f\right]^2\right\}^{1/2}} \tag{2.3.5}$$

unde se presupune că $w(s,t)$ este centrată la coordonatele $(x\ y)$. Sumele sunt luate peste coordonatele imaginii comune celor două regiuni, m_w fiind media intensității lui w.

Este de subliniat faptul că, în general, $\gamma(x,y)$ va varia de la o locație la alta și că poate lua valori în intervalul [-1, 1], cu valoarea 1 corespunzătoare unei potriviri perfecte.

Procedura va calcula în continuare $\gamma(x,y)$ la fiecare locație (x,y) și va selecta cea mai mare valoare pentru a determina cea mai bună potrivire a lui w în f.

Calitatea potrivirii poate fi controlată acceptând coeficientul de corelație doar dacă depășește o valoare prestabilită (de exemplu 0,9).

Deoarece această metodă constă în compararea directă a două regiuni, este evident sensibilă la variații ale dimensiunii și orientării obiectelor. Variațiile de intensitate sunt normalizate de către numitor în expresia (2.3.5).

2.3.2. Metode structurale

Tehnicile discutate anterior tratează modelele pornind de la baze cantitative, ignorând orice relație geometrică care poate fi moștenită într-o formă a unui obiect.

Metodele structurale, pe de altă parte, încercă să realizeze o discriminare între obiecte bazându-se pe aceste relații. Punctul central în recunoașterea structurală este descompunerea unui obiect în ***modele primitive*** [3], [6], [11], [14], [20], [21], [86], [92], [93], [101], [105], [116], [120], [145], [148].

Această idee este ușor de explicat cu ajutorul figurii 2.3.1. Partea (a) a acestei figuri prezintă un contur de obiect simplu, iar figura (b) arată un set de elemente primitive de lungime și direcție dată.

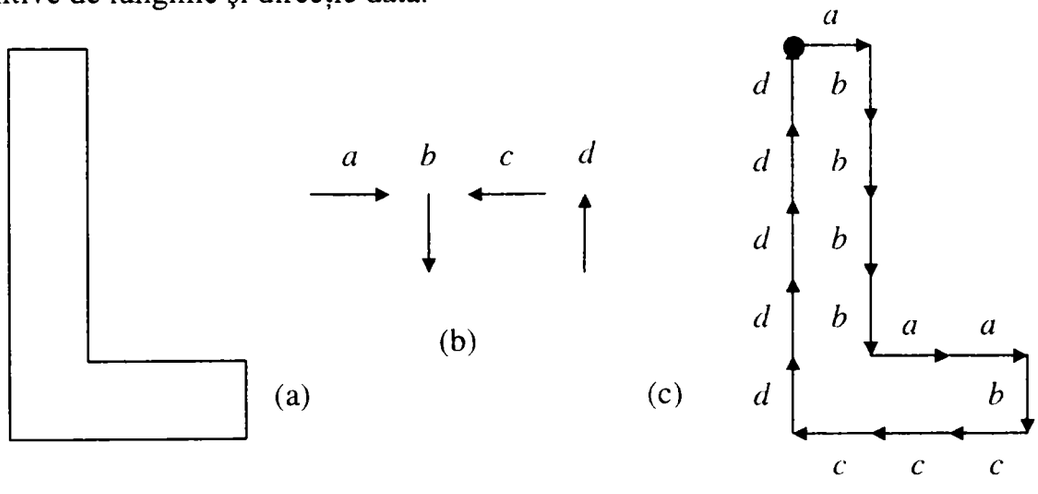


Fig. 2.3.1
Descrierea primitivelor unei forme date
(a) Conturul obiectului.
(b) Primitive.
(c) Conturul obiectului codificat după primitive, rezultatul fiind șirul

Pornind din colțul stânga sus, urmărind conturul obiectului în sens antitrigonometric, și identificând instanțele primitivelor, se obține conturul codificat din figura 2.3.1,c. În final, se reprezintă conturul prin codul *abbbbaabccccddddd*.

Lungimea şi direcţia cunoscută a acestor primitive, împreună cu ordinea în care apar, stabilesc structura obiectului ca pe o reprezentare particulară. Scopul acestui capitol este scoaterea în evidenţă a tehnicilor adecvate pentru tratarea acestor tipuri cât şi a altora de descriptori de modele structurale.

➢ **Potrivirea numerelor de formă**

O procedură similară cu conceptul distanţei minime poate fi formulată pentru compararea a două contururi de obiecte care sunt descrise cu numere de formă [26],[27],[28].

Gradul de asemănare k dintre două contururi de obiecte, A şi B, este definit ca cel mai mare ordin pentru care numerele lor de formă coincid. Adică :

$$s_4(A)=s_4(B),\ s_6(A)=s_6(B),\ s_8(A)=s_8(B),\ \ldots,\ s_k(A)=s_k(B),\ s_{k+2}(A)\neq s_{k+2}(B),$$
$$s_{k+4}(A)\neq s_{k+4}(B),\ \ldots, \qquad (2.3.6)$$

unde :

- s reprezintă numărul de formă
- indicele reprezintă ordinul

Distanţa dintre două forme A şi B este definită ca inversul gradului lor de similaritate:

$$D(A,B)=\frac{1}{k} \qquad (2.3.7)$$

Această distanţă satisface proprietăţile :

- $D(A,B)\geq 0$
- $D(A,B)=0$ dacă $A=B$ (2.3.8)
- $D(A,C)\leq \max[D(A,B),D(B,C)]$

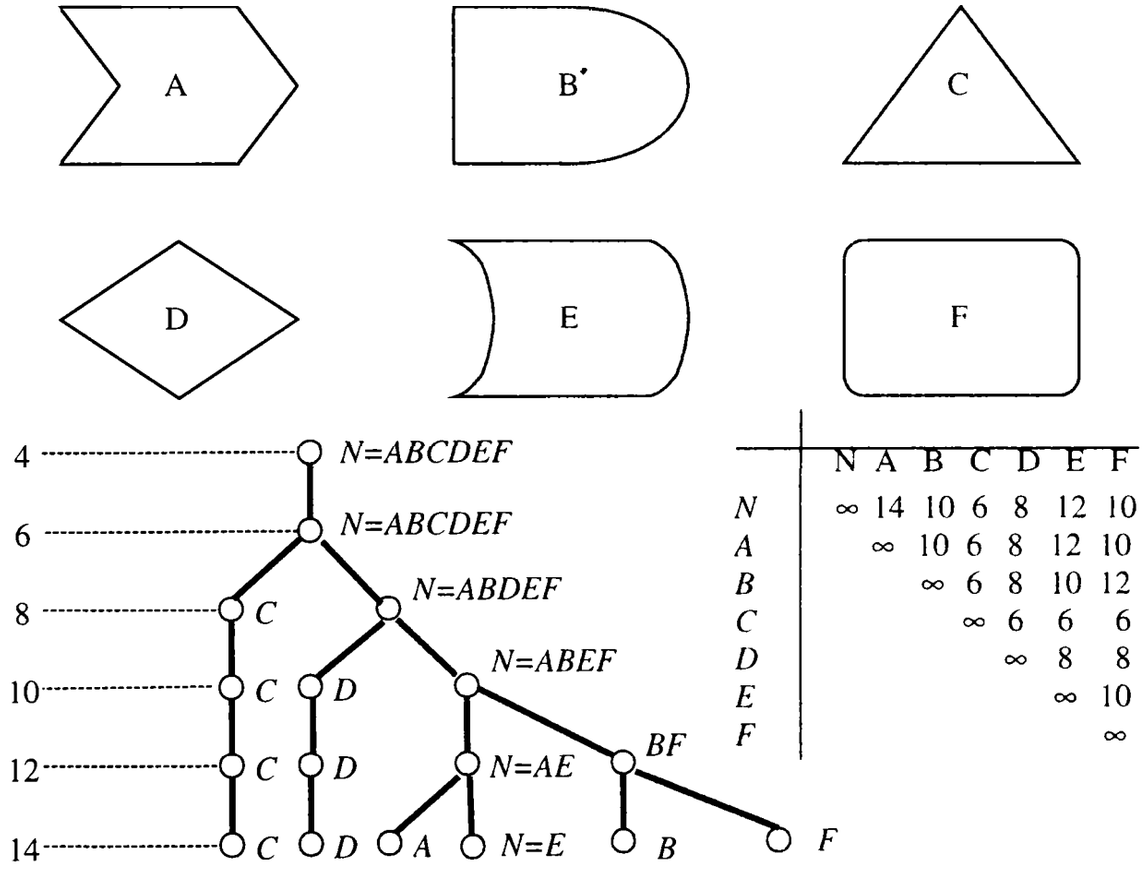
În scopul comparării a două forme, se poate folosi fie k fie D. Dacă se utilizează gradul de asemănare, atunci rezultă din cele de mai sus faptul că, cu cât k este mai mare, cu atât formele sunt mai similare (k este infinit pentru forme identice). În cazul utilizării distanţei problema se pune invers.

Se consideră, spre exemplificare că robotul trebuie să recunoască dintr-o mulţime de contururi pe cel pe care l-a analizat şi căruia i-a atribuit numărul de formă N conform conform figurii 2.2.7 . Dintr-o mulţime de contururi el va trebui să-l selecteze pe cel căutat.

Metoda de căutare este ilustrată în figura 2.3.2,a. Situaţia este analoagă cu problema deţinerii a cinci modele de formă ale căror identităţi sunt cunoscute. Se determină care dintre acestea constituie cea mai bună potrivire cu o formă necunoscută.

Căutarea poate fi vizualizată cu ajutorul ***arborelui de similitudine*** (figura 2.3.2,b). Rădăcina acestui arbore corespunde celui mai mic grad de similitudine considerat, adică 4. Aşa cum se vede în arbore, toate formele sunt identice până la gradul 8, cu excepţia formei C. Astfel, gradul de similitudine al acestei forme în raport cu toate celelalte este 6. Parcurgând descendent arborele se va descoperi că forma *D* are gradul de similitudine 8 în raport cu formele rămase, ş.a.m.d. În acest caz particular, forma *E* rezultă cu cea mai apropiată potrivire de forma N.

Aceeaşi informaţie se poate obţine din ***matricea de similitudine*** din figura 2.3.2,c.



	N	A	B	C	D	E	F
N	∞	14	10	6	8	12	10
A		∞	10	6	8	12	10
B			∞	6	8	10	12
C				∞	6	6	6
D					∞	8	8
E						∞	10
F							∞

Fig. 2.3.2
Construcţia unui arbore de similitudine pentru recunoaşterea formei
(a) Forme, (b) Arborele de similaritate, (c) Matricea de similaritate

➢ **Crearea unei baze de date cu numere de formă**

O altă metodă de abordare a recunoaşterii este construirea unei baze de date formată din numere de formă, care să identifice obiectele de interes, metodă ce se poate folosi-în special în cazul unui număr limitat şi cunoscut dinainte de obiecte, deoarece în acest caz se pot anticipa şi efectele zgomotului.

Reluând exemplul de la descrierea cu ajutorul numerelor de formă, se poate utiliza pentru recunoaşterea figurilor geometrice plane următoarea bază de date, bază care se poate dezvolta pe măsură ce rezultatele experimentale o demonstrează ca necesară :

```
CString Recognition(CString ShapeNumber)
{
 CString Geometric;

 if(ShapeNumber.Compare("00003000030000300003")==0)
  Geometric = "SQUARE";
 else
  if(ShapeNumber.Compare("00000000330000000033")==0 ||
    ShapeNumber.Compare("00000003030000000303")==0 ||
    ShapeNumber.Compare("00000030030000003003")==0 ||
    ShapeNumber.Compare("00000300030000030003")==0)
   Geometric = "RECTANGLE";
  else
   if(ShapeNumber.Compare("00303003130031300313")==0)
    Geometric = "CIRCLE";
   else
    if(ShapeNumber.Compare("00003310310330130133")==0 ||
      ShapeNumber.Compare("00003300300330130133")==0 ||
      ShapeNumber.Compare("00000033003033131033")==0 ||
      ShapeNumber.Compare("00000033030033131033")==0 ||
      ShapeNumber.Compare("00000330030331313133")==0 ||
      ShapeNumber.Compare("00000331310303013133")==0 ||
      ShapeNumber.Compare("00000331031330103133")==0 ||
      ShapeNumber.Compare("00000331310331030133")==0)
     Geometric = "TRIANGLE";
    else
     if(ShapeNumber.Compare("00003000313003130003")==0 ||
       ShapeNumber.Compare("00003000301303130003")==0)
      Geometric = "ROBOT";
     else
      Geometric = "NOMATCH";
 return Geometric;
}
```

Folosind funcţiile implementate se poate crea o funcţie de recunoaştere automatizată care va afişa în final figurile geometrice recunoscute după un cod al culorilor:

```
void CROBOVIEWDoc::RecognizeObjectsByColor(CROBOVIEWDoc          *OriginalDoc)
{
 CROBOVIEWDoc   *DestinDoc;
```

```
CCorp               *CurrentCorp;
POSITION            CurrentPosition;
int                 i,x,y,NumberOfPoints;
struct              BASICRECTANGLE{
                                    RECT   Rect;
                                    int    MajorAxis;
                                    int    MinorAxis;
                                    double Eccentricity;
                                    }BasicRectangle;

DestinDoc = ((CMainFrame*)AfxGetMainWnd())->GetActiveSourceDocument();
memset(&DestinDoc->BitmapMatrix,0xFF,320*240*3);
CurrentPosition = m_CorpList.GetHeadPosition();
while(CurrentPosition)
 {
 BasicRectangle.Rect.left   = 10000;
 BasicRectangle.Rect.right  = 0;
 BasicRectangle.Rect.top    = 10000;
 BasicRectangle.Rect.bottom = 0;
 CurrentCorp = m_CorpList.GetNext(CurrentPosition);
 NumberOfPoints = CurrentCorp->m_pointArray.GetSize();
// Determine bounding rectangle of the given object
 for(i=0;i<NumberOfPoints;i++)
  {
  if(CurrentCorp->m_pointArray[i].x < BasicRectangle.Rect.left)
   BasicRectangle.Rect.left = CurrentCorp->m_pointArray[i].x;
  if(CurrentCorp->m_pointArray[i].x > BasicRectangle.Rect.right)
   BasicRectangle.Rect.right = CurrentCorp->m_pointArray[i].x;
  if(CurrentCorp->m_pointArray[i].y < BasicRectangle.Rect.top)
   BasicRectangle.Rect.top = CurrentCorp->m_pointArray[i].y;
  if(CurrentCorp->m_pointArray[i].y > BasicRectangle.Rect.bottom)
   BasicRectangle.Rect.bottom = CurrentCorp->m_pointArray[i].y;
  }
 for(x=0;x<Xmax;x++)
  for(y=0;y<REALYMAX;y++)
    {
    if(OriginalDoc->BitmapMatrix[x][y][0] == 0 && x>=BasicRectangle.Rect.left-4 &&
               x<=BasicRectangle.Rect.right+4 && y>=BasicRectangle.Rect.top-4 &&
               y<=BasicRectangle.Rect.bottom+4)
     {
     DestinDoc->BitmapMatrix[x][y][0] = GetRValue(GetObjectColor(CurrentCorp->m_Geometric));
     DestinDoc->BitmapMatrix[x][y][1] = GetGValue(GetObjectColor(CurrentCorp->m_Geometric));
     DestinDoc->BitmapMatrix[x][y][2] = GetBValue(GetObjectColor(CurrentCorp->m_Geometric));
      }
     }
   }
 ((CMainFrame*)AfxGetMainWnd())->GetActiveFrame()->RedrawWindow();
}


void CMainFrame::OnUpdateToolsShapeNumber(void)
{
 CROBOVIEWDoc       *SourceDoc=NULL;
 CROBOVIEWDoc       *OriginalDoc=NULL;

 OriginalDoc = ChainCode(4,4,50,TRUE,&SourceDoc);
 SourceDoc->GetShapeNumbers(OriginalDoc);
 SourceDoc->RecognizeObjectsByColor(OriginalDoc);
}
```

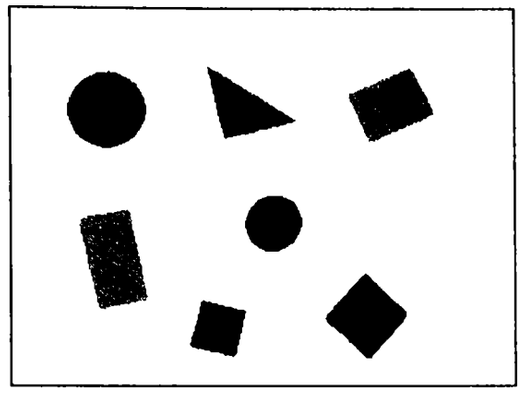


Fig. 2.3.3
Rezultatul recunoaşterii figurilor geometrice şi a unui robot efector, rezultat afişat prin colorarea fiecărei clase de obiecte într-o anumită culoare

Partea III-a
Planificarea mişcărilor roboţilor autonomi pe baza informaţiilor obţinute prin vedere artificială

3.1 Conducerea unui robot într-un spaţiu de lucru populat cu obstacole plane

Experimentul constă din achiziţionarea unei imagini statice a unui mediu format din figuri geometrice plane în cadrul cărora se află un robot de dimensiuni reduse 10 x 10 cm. Camera de achiziţie este plasată deasupra mediului la o distanţă care să permită cuprinderea tuturor obiectelor de interes. Pentru realizarea unui contrast cât mai mare între fundal şi obiecte, fundalul este de culoare albă, iar figurile geometrice sunt de culoare neagră pentru a avea un contrast cât mai mare. Având în vedere scara întregului ansamblu rezultă faptul că robotul efector trebuia să fie de dimensiuni reduse, comparabile cu cele ale figurilor geometrice existente. Confecţionarea artizanală a acestui robot, a urmărit :

- să poată fi uşor comandat printr-o interfaţă existentă la un calculator obişnuit
- să se poată deplasa înainte şi înapoi pe aceeaşi direcţie
- să poată să-şi schimbe direcţia

Odată realizat un robot [88] cu caracteristicile de mai sus procesul de conducere a acestuia printre figurile geometrice constă în recunoaşterea figurilor geometrice şi a robotului efector, stabilirea unei sarcini de lucru, determinarea traiectoriei şi comanda efectivă a robotului pentru parcurgerea acestuia.

3.1.1 Descrierea robotului efector.

Deoarece robotul efector trebuie să îndeplinească condiţiile menţionate, soluţia cea mai simplă a fost realizarea sa prin folosirea a două motoare pas cu pas de dimensiuni reduse, cu patru faze comandate monopolar [88], [7], [106]. Deoarece cele două motoare pas cu pas au nevoie de 8 semnale de comandă, folosirea portului paralel cu cele 8 ieşiri de date ale sale se recomandă de la sine. Deoarece puterea pe ieşirile portului paralel este prea mică, a fost necesară crearea unei interfeţe cu rol de amplificator de curent.

Întregul ansamblu format din motoarele pas cu pas şi interfaţa necesară a fost realizată într-o carcasă construită din bucăţi de circuit imprimat cositorite între ele. Pe axele de rotaţie ale celor două motoare pas cu pas au fost amplasate două roţi dinţate care asigură deplasarea robotului efector. Din cauza aderenţei reduse a roţilor dinţate,

suprafața de deplasare trebuie să aibă un coeficient de frecare ridicat. Cea mai bună soluție care îndeplinește aceste condiții este o bucată de polistiren întrucât și culoarea fondului (albă) este asigurată în acest mod. De asemenea din cauza faptului că circuitul imprimat din care este confecționată carcasa robotului nu este de culoare neagră, a fost necesară confecționarea unui înveliș din carton negru cu reflexie redusă. În aceeași idee a evidențierii conturului robotului efector, cablul paralel de legătură cu calculatorul trebuie mascat printr-o prelungire de culoare albă de data aceasta, a învelișului de carton.

În figura 3.1.1 sunt prezentate câteva imagini ale robotului efector cu și fără învelișul de camuflaj.

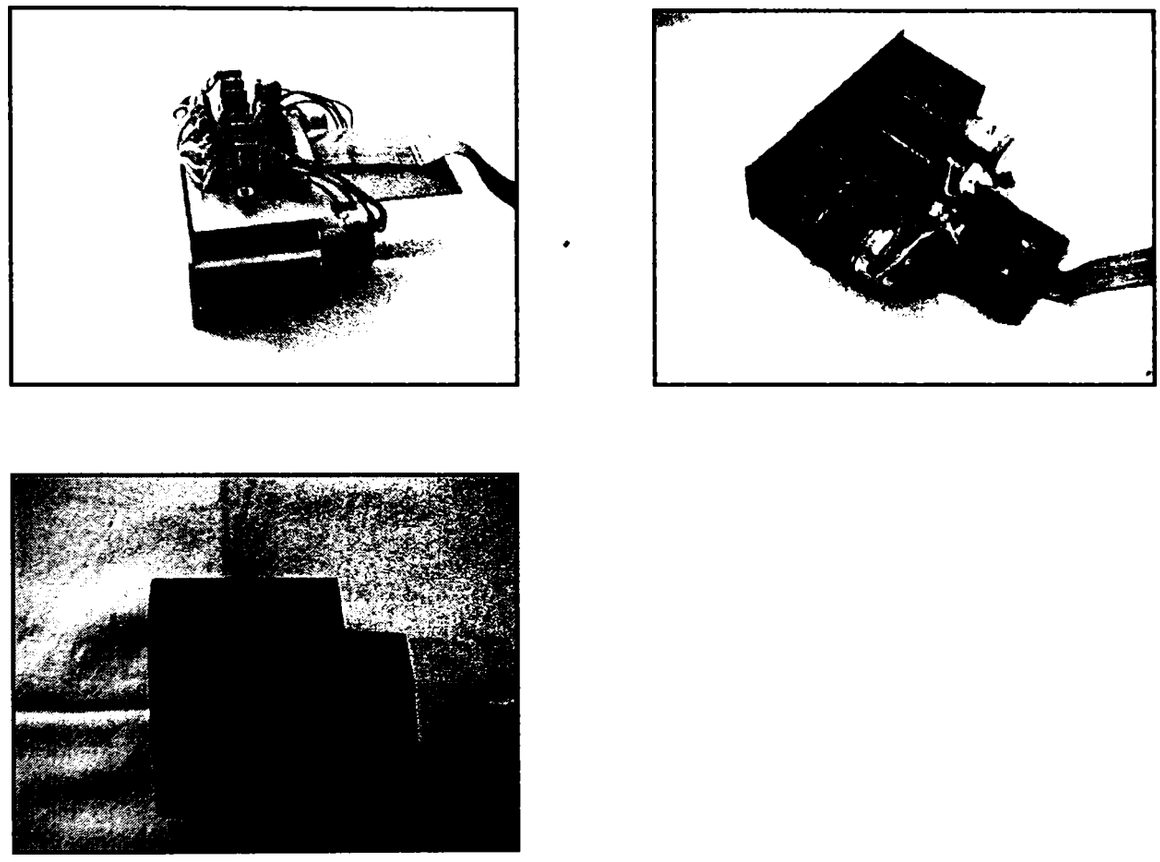


Fig. 3.1.1
(a) Interfața robotului
(b) Partea mecanică de acționare
(c) Robotul îmbrăcat în învelitoarea de carton negru

3.1.2. Detalii constructive

➢ **Principiul de funcționare al motoarelor pas cu pas**

Motoarele pas cu pas reprezintă mașini electrice sincrone modificate ale căror înfășurări de comandă se alimentează cu un sistem m-fazat de impulsuri de tensiune practic dreptunghiulare, rotorul fiind executat fără înfășurări auxiliare de pornire. Tensiunea sub formă de impulsuri aplicată fazelor motorului determină o repartizare discretă a câmpului magnetic în întrefierul mașinii, și ca urmare, mișcarea rotorului constă din deplasări unghiulare elementare succesive [7],[88], [106].

Spre deosebire de motoarele sincrone clasice, motoarele pas cu pas intră în sincronism fără alunecare, iar frânarea se realizează fără ieșirea din sincronism. Datorită acestui fapt ele asigură în domeniul de funcționare porniri bruște, opriri și reversări fără pierderea informației sau fără omisiuni de pas.

Particularitățile utilizărilor cărora le sunt destinate motoarele pas cu pas, au determinat apariția unor tipuri constructive speciale care se deosebesc de mașinile sincrone clasice. Astfel, motoarele pas cu pas se execută cu un diametru rotoric minim și cu un număr mărit de poli, ceea ce permite obținerea unei valori unghiulare mici a pasului.

Înfășurările de comandă ale motoarelor pas cu pas se pot alimenta în mod separat sau în grupuri în diferite combinații. Fiecărei combinații sau tact de comutație îi corespunde o orientare spațială determinată a câmpului magnetic rezultant în întrefierul mașinii.

Ca și motoarele sincrone clasice, motoarele pas cu pas se clasifică în funcție de construcția circuitului magnetic și de numărul înfășurărilor de comandă. Se deosebesc două tipuri de bază:

- Motoare de tip reactiv cu rotorul format numai din jug confecționat din tole.
- Motoare de tip activ cu rotor format dintr-un jug la care se adaugă electromagneți sau magneți permanenți.

Motoarele pas cu pas de tip reactiv au rotorul executat sub forma unui cilindru feromagnetic dințat ce poate fi executat cu un număr mare de poli. Astfel de motoare se execută cu un pas până la un grad, ceea ce este satisfăcător pentru micșorarea erorii unghiulare absolute. La o frecvență ridicată a impulsurilor de comandă (sute de Hz) aceste motoare au viteza de rotație scăzută și prin urmare pot fi utilizate fără reductor.

Motoarele pas cu pas de tip activ au în componența rotorului magneți permanenți sau electromagneți cu înfăşurări de excitație a căror capete sunt scoase la inele colectoare. Motoarele pas cu pas de tip activ se execută cu paşi mari deoarece pasul polar al rotorului cu magneți permanenți sau electromagneți de excitație nu poate fi micşorat prea mult din considerente de ordin constructiv. Din această cauză motoarele pas cu pas cu rotor activ se utilizează în sistemele cu viteze de rotație relativ mari.

Tipurile de bază indicate se mai clasifică în:

- Motoare pas cu pas cu mai multe statoare (polistatorice).
- Motoare pas cu pas polifazate.

Motoarele pas cu pas cu mai multe statoare constau din m sisteme stator-rotor dispuse coaxial în care fiecare sistem, asemănător înfăşurărilor motoarelor polifazate, este rotit unul față de altul. Între motoarele pas cu pas polifazate şi polistatorice nu există deosebire în ce priveşte explicarea funcționării şi tratarea matematică.

Motorul robotului realizat este un motor pas cu pas cu patru statoare (m=3). Cele patru sisteme stator-rotor sunt dispuse pe un ax comun. Circuitul magnetic al fiecărui stator este realizat sub formă de poli aparenți distribuiți uniform pe circumferința statorului.

Cele patru rotoare sunt executate din pachete de tole având pe circumferință acelaşi număr de poli ca şi statoarele.

Polii celor m statoare sunt axiali, pe când polii celor m rotoare sunt decalați cu $1/m$ paşi polari. Înfăşurările polilor unui stator sunt conectate astfel încât prin alimentarea lor cu curent continuu se realizează pe circumferința statorului o succesiune de poli nord-sud.

În poziția inițială este alimentată înfăşurarea statorului 1 şi rotorul este plasat astfel încât reluctanța circuitului magnetic este minimă şi deci polii rotorici se vor aşeza în fața polilor statorici. Prin decuplarea înfăşurării storului 1 şi cuplarea în acelaşi timp a înfăşurării statorului 2, rotorul se roteşte în sens invers acelor de ceasornic cu un unghi θ_p. Pentru executarea următorului pas se decuplează înfăşurarea statorului 2 şi se cuplează înfăşurarea statorului 3, pentru realizarea pasului următor se decuplează înfăşurarea statorului 3 şi se cuplează înfăşurarea statorului 4, iar pentru ultimul pas se decuplează înfăşurarea statorului 4 şi se cuplează înfăşurarea statorului 1.

În felul acesta rotorul s-a rotit cu un unghi total $m\theta_p$ care corespunde unui pas polar şi se află într-o poziţie identică cu cea iniţială. Pentru realizarea unei rotaţii complete succesiunea cuplării înfăşurărilor se repetă de un număr de ori egal cu numărul polilor. În cadrul motorului cu patru statoare şi 16 poli, la o rotaţie completă se realizează 64 de paşi sau în caz general în $2mp$ paşi, unde p este numărul perechilor de poli. Deplasarea rotorului cu un pas are loc sub acţiunea unui cuplu reactiv.

Prin alimentarea înfăşurării unui stator în curent continuu, forma cuplului static care acţionează asupra rotorului la deplasarea acestuia dintr-o poziţie fixă.

În studiul regimurilor de funcţionare ale motoarelor pas cu pas, precum şi pentru aprecierea comparativă a performanţelor diferitelor tipuri de motoare pas cu pas, este necesară cunoaşterea mărimilor lor caracteristice. Acestea sunt :

- ***Unghiul de pas*** (pasul motorului) - reprezintă unghiul de rotaţie al rotorului corespunzător unui tact de comandă. Din punct de vedere cantitativ el depinde de numărul de perechi de poli şi de numărul înfăşurărilor de comandă decalate spaţial una de alta; în cazul motorului pas cu pas de tip activ de numărul dinţilor rotorici iar în cazul motorului pas cu pas de tip reactiv de numărul înfăşurărilor de comandă. De asemenea, unghiul de pas depinde, la ambele tipuri de motoare, de secvenţa de comandă a înfăşurătorilor statorului. Spre exemplu, în cazul unei secvenţe de tipul 1-1, 2-2-2, 3-3... se obţine un pas egal cu jumătate din valoarea corespunzătoare unei secvenţe simple 1-2-3.
- ***Cuplul static sincronizat*** - caracteristica cuplului static sincronizat reprezintă variaţia cuplului electromagnetic dezvoltat de motor în funcţie de unghiul de decalaj al axei magnetice a rotorului faţă de axa fluxului rezultant statoric, în cazul când înfăşurările de comandă sunt parcurse de curent. În general cuplul electromagnetic variază periodic cu unghiul de rotaţie după o funcţie apropiată de o sinusoidă.
- ***Cuplurile limită*** - o deosebită importanţă în alegerea domeniului de funcţionare a motoarelor pas cu pas o are cunoaşterea valorilor maxime ale cuplului aplicat pe arborele motorului. În regim permanent de funcţionare se defineşte un cuplu critic cvasi-staţionar ca fiind valoarea maximă la care poate mări cuplul rezistent, la o frecvenţă de comandă dat, fără a cauza ieşirea din sincronism a motorului (pierderea paşilor). La frecvenţe de comandă mai mari, cuplul critic cvasi-staţionar este mai mic. Variaţia cuplului critic cvasi-

staționar cu frecvența de comandă reprezintă ***caracteristica de sarcină limită*** a motorului pas cu pas. La pornire se definește un cuplu critic de pornire, care reprezintă valoarea maximă a cuplului rezistent la care motorul poate porni cu o frecvență dată fără pierdere de pași. Variația cuplului critic de pornire cu frecvența de comandă se numește caracteristica limită de pornire. În mod similar se definesc cuplurile critice de oprire și reversare.

- ***Frecvențele limită*** - caracteristica de sarcină limită, precum și cele dinamice (de oprire, pornire și reversare), s-au stabilit luând ca reper o frecvență de comandă dată. Astfel, s-au definit cuplurile limită. Dacă se ia ca referință un cuplu rezistent dat se pot defini în mod similar frecvențele limită (critice) la funcționarea în regim static și dinamic (pornire, oprire, reversare). Toate mărimile critice depind în cea mai mare măsură de panta curentului ce se stabilește în înfășurările de comandă și de inerția sistemului de acționare. Caracteristicile statice și dinamice cuplu-frecvență au forme ce se apropie de hiperbolă, prezentând una sau mai multe paliere datorită prezenței fenomenului de rezonanță mecanică. La nivelul superior se situează caracteristica limită de oprire, aceasta din cauză că efectul de oprire este favorizat de sensul cuplului rezistent. Sub caracteristica limită de oprire urmează, în ordine, caracteristicile de sarcină limită, de pornire și de reversare, aceasta din urmă exprimând regimul cel mai greu de funcționare sincronă a motorului pas cu pas. De altfel, caracteristica limită de reversare este acoperitoare în sensul că alegerea domeniului de funcționare fără pierderi de pași în orice regim de funcționare.

❖ **Motoare pas cu pas cu trei sau mai multe înfășurări de comandă**

Din punct de vedere constructiv, motoarele pas cu pas cu trei sau mai multe înfășurări de comandă se pot clasifica astfel:

- Motoare pas cu pas cu mai multe statoare, pe fiecare stator fiind dispusă câte o înfășurare de comandă.
- Motoare pas cu pas cu un stator și mai multe înfășurări de comandă.
- Motoare pas cu pas de tip activ cu două statoare și patru înfășurări de comandă.

În stadiul actual de perfecționare constructivă a motoarelor pas cu pas reactive, cea mai mare răspândire o au motoarele pas cu pas trifazate cu un stator şi trei statoare şi motoarele pas cu pas cu un stator şi patru faze ca cele folosite în acest experiment.

Dinții rotorici care se găsesc sub dinții polilor vecini sunt decalați față de aceştia cu ¼ din pasul polar, adică cu un unghi egal cu θ_p. Prin comutarea alimentării de la înfăşurarea 1 la înfăşurarea 2 rotorul se roteşte din nou cu un unghi corespunzător pasului θ_p şi ocupă poziția corespunzătoare reluctanței magnetice minime a întrefierului de sub polul II.

În realitate la acest tip de motor se pot cupla şi două înfăşurări de comandă succesive, ceea ce duce la mărirea cuplului de rotație. În acest caz fuxul magnetic se închide prin polii vecini, iar poziția rotorului pentru care se obține reluctanța magnetică minimă a întrefierului se deplasează cu $\theta_p/2$ şi coincide cu axa de simetrie a polilor statorici pe care se găsesc înfăşurările alimentate.

Motorul pas cu pas reactiv cu patru înfăşurări de comandă şi cu sistem magnetic nesimetric, în principiu funcționează asemănător cu motoarele pas cu pas reactive având circuitul magnetic simetric. Motorul are numai patru poli aparenți pe care sunt dispuse patru înfăşurări de comandă, fiecare înfăşurare fiind alcătuită dintr-o singură secțiune.

Avantajul acestui motor constă în faptul că dezvoltă un cuplu mai mare pe unitatea de greutate decât în cazul motoarelor cu sistem magnetic simetric. Dezavantajul constă în existența unor cupluri de frecări suplimentare în lagăre şi a unui zgomot în funcționare cauzat de prezența unor forțe de atracție nesimetrice.

Motorul pas cu pas cu opt înfăşurări de comandă şi cu sistem magnetic nesimetric realizează valori unghiulare ale pasului θ_p mai mici, pentru acelaşi număr de dinți statorici şi rotorici. Statorul acestui tip de motor are opt poli aparenți pe care sunt dispuse opt bobine, ce realizează cele opt înfăşurări de comandă. Înfăşurările de comandă se alimentează cu impulsuri de aceeaşi polaritate, iar pentru mărirea cuplului se pot alimenta simultan două sau trei înfăşurări. Şi în cazul acestui tip de motor apar aceleaşi forțe magnetice de atracție nesimetrice, ceea ce la motoarele peste o anumită putere creează un zgomot în funcționare supărător.

În cazul tuturor motoarelor reactive cu un stator şi mai multe înfăşurări de comandă, pentru ca să se asigure o rotație uniformă a rotorului, este necesar ca între dinții rotorici şi cei statorici care aparțin polilor a căror înfăşurări urmează să fie

alimentate să existe un decalaj unghiular de 360/m*zr (unde zr este numărul dinţilor rotorici). Pentru fiecare înfăşurare a dinţilor de comandă, axa câmpului magnetic statoric se roteşte cu un unghi egal cu distanţa polară statorică θ=360/2p unde 2p este numărul polilor statorici. În acest caz rotorul se va roti cu un unghi egal cu un pas θ=360/m*zr.

Se observă că rotorul se roteşte diferit de câmpul magnetic statoric de θ/θ_p ori. Raportul θ/θ_p se numeşte ***coeficientul de reducere electromagnetic***, iar motoarele pas cu pas reactive se numesc ***reductor***.

❖ Probleme specifice cu privire la motoarele pas cu pas

Motorul electric pas cu pas este un convertor electromagnetic, care realizează conversia unui tren de impulsuri de comandă, aplicate înfăşurărilor de fază ale motorului, într-o mişcare de rotaţie ce constă din deplasări unghiulare discrete, de mărime egală. Unghiul de rotaţie pe care îl execută rotorul, la aplicarea unui impuls de comandă, reprezintă unghiul de pas al motorului. Numărul paşilor efectuaţi trebuie să corespundă, în cazul unei funcţionări corecte cu numărul impulsurilor de comandă.

Majoritatea motoarelor pas cu pas sunt bidirecţionale şi permit o accelerare, oprire şi reversare rapidă, fără pierderi de paşi, dacă sunt comandate cu impulsuri a căror frecvenţă de repetiţie este inferioară frecvenţei limită, corespunzătoare unui cuplu rezistent şi un moment de inerţie dat. Pentru extinderea funcţionării motoarelor pas cu pas la viteze mai mari decât viteza corespunzătoare frecvenţei limită, este necesară o accelerare prin creşterea treptată a frecvenţei impulsurilor de comandă.

Motoarele pas cu pas sunt utilizate, în special, în aplicaţiile unde se doreşte realizarea unei mişcări incrementale, folosind sisteme de comandă numerică.

Utilizarea motoarelor pas cu pas conferă, următoarele avantaje:

- Asigură univocitatea conversiei număr de impulsuri-deplasare şi pot fi utilizate în circuit deschis.
- Gama largă de frecvenţe de comandă.
- Precizie de poziţionare şi rezoluţie mare.
- Permit porniri, opriri, reversări, fără pierderea de paşi.
- Memorează poziţia.
- Sunt compatibile cu comanda numerică.

Dezavantajele utilizării motoarelor pas cu pas:

- Unghi de pas, deci increment de rotație, de valoare fixă pentru un motor dat.
- Viteză de rotație relativ scăzută.
- Putere dezvoltată la arbore, de valoare redusă.
- Randament energetic scăzut.
- Necesită o schemă de comandă adaptabilă la tipul constructiv.
- Necesită o schemă de comandă complexă pentru asigurarea unei funcționări la viteze mari.

❖ Modalitățile de comandă a fazelor motoarelor pas cu pas

Principial, fazele motoarelor pas cu pas se alimentează cu impulsuri de curent de amplitudine constantă, care se comută de pe o fază pe alta, în ritmul unui tact de comandă. Modalități de comandă a fazelor motoarelor pas cu pas pot fi asigurate prin:

1. Comanda potențială sau prin impulsuri.
2. Comanda monopolară sau bipolară.
3. Comanda simetrică (simplă sau dublă) sau nesimetrică.

Un circuit concret de alimentare a fazelor unui motor pas cu pas asigură realizarea simultană a 3 (una de tip 1, una de tip 2, şi una de tip 3) dintre cele 6 modalități de comandă prezentate mai sus.

Comanda potențială se referă la durata alimentării unei faze, în raport cu durata între 2 tacturi de comandă 1/f, f fiind frecvența de comandă a motorului pas cu pas. Dacă durata alimentării unei faze este cel puțin egală cu 1/f, atunci comanda este potențială. În acest caz, durata aplicării tensiunii pe o fază variază invers proporțional cu frecvența. Dacă durata alimentării este constantă şi întotdeauna mai mic decât 1/f, atunci este vorba de o comandă prin impulsuri.

Deşi majoritatea schemelor de comutație a fazelor motoarelor pas cu pas sunt cu comandă potențială, s-a consacrat denumirea de comandă prin impulsuri, probabil datorită formei de variație în timp a tensiunilor de alimentare.

Comanda monopolară asigură un sens unic al curentului prin fiecare înfăşurare a motorului, în tot timpul funcționării. Comanda bipolară determină ca, pe parcursul unui ciclu complet de comenzi aplicate înfăşurărilor, sensul curentului prin fiecare înfăşurare să se schimbe succesiv.

Comanda este ***simetrică*** dacă, la un moment dat, sunt alimentate un număr egal de faze şi nesimetrică, dacă numărul fazelor alimentate simultan se schimbă alternativ la fiecare comandă de tact.

Comanda simetrică este numită ***simplă*** sau ***dublă***, după cum sunt alimentate câte una sau două înfăşurări simultan. Prin utilizarea comenzii duble, pentru un motor pas cu pas cu patru faze se măreşte atât cuplul dezvoltat, cât şi amortizarea, dar cresc consumul de putere şi temperatura motorului. În figura 3.1.2 sunt prezentate formele de variaţie în timp ale tensiunilor de alimentare pe faze, pentru comanda monopolară a unui motor pas cu pas cu patru faze, pentru a exemplifica diferite moduri de comandă.

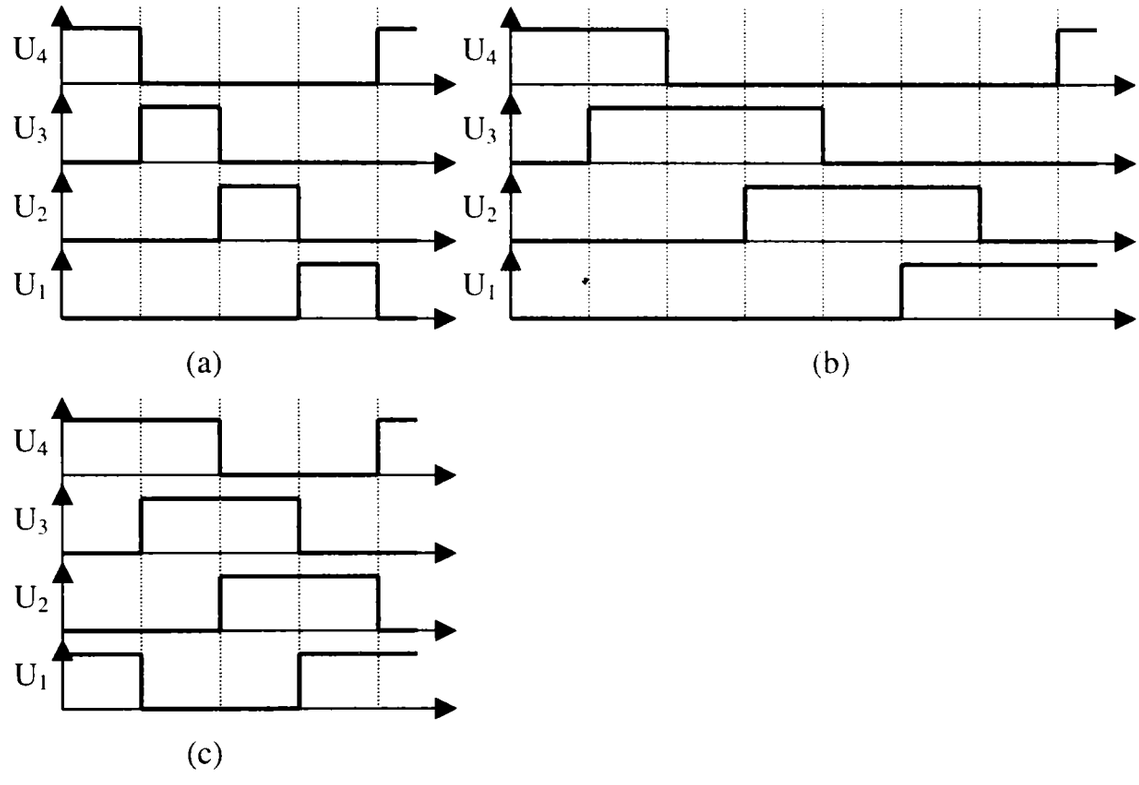


Fig. 3.1.2
Formele de variaţie în timp a tensiunilor de alimentare pe faze,
pentru comanda monopolară a unui motor pas cu pas cu patru faze

În figura 3.1.2 sunt prezentate formele de variaţie în timp a tensiunilor de alimentare pe faze, pentru comanda monopolară a unui motor pas cu pas cu patru faze, pentru a exemplifica diferite moduri de comandă.

Pentru inversarea sensului de rotaţie, impulsurile reprezentate în secvenţele din figura 3.1.2 se aplică în ordine inversă: U_4-U_3-U_2-U_1, etc.

În figura 3.1.3 este prezentat modul de comandă bipolară simetrică dublă a unui motor pas cu pas cu patru faze. Comanda necesită o schemă de comandă complexă, care să asigure circulația curenților din înfăşurări, în ambele sensuri.

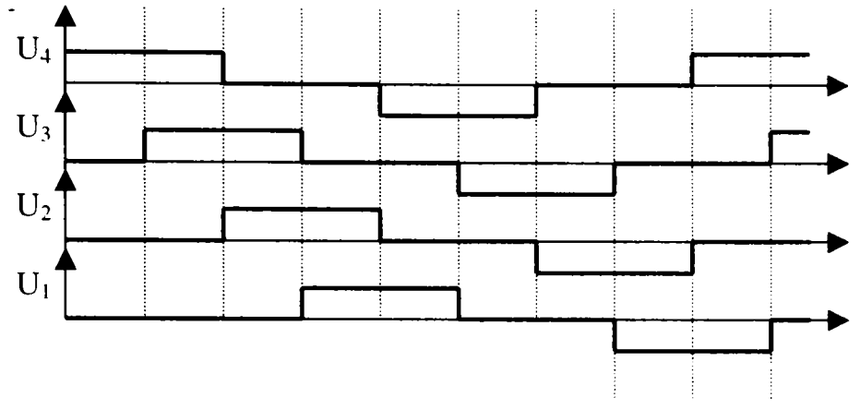


Fig. 3.1.3
Comanda bipolară simetrică dublă a unui motor pas cu pas cu patru faze

❖ **Schema bloc de comandă a motoarelor pas cu pas**

Performanțele unui motor pas cu pas sunt strâns legate de tipul schemei sale de comandă. Astfel, amortizarea, frecvența maximă de mers, cuplul dinamic maxim, precum şi randamentul sau puterea disipată depind în mare măsură de schema de alimentare utilizată. În general, comanda unui motor pas cu pas se face printr-un circuit electronic de putere care alimentează secvențial înfăşurările motorului. Sensul de distribuție al alimentării, ca şi tipul secvenței (simetrică, nesimetrică, număr de ieşiri), precum şi frecvența de comutare a înfăşurărilor sunt realizate prin prelucrare logică secvențială cu circuite integrate, în timp ce curentul în fazele motorului este asigurat printr-un etaj de comutație statică forțată.

Rolul circuitului de comandă este de a prelua semnale standard TTL (impulsuri de comandă şi nivel de sens), a le amplifica şi apoi de a le aplica fazelor motorului. În acest fel, schema de comandă a unui motor pas cu pas îmbină semnale discrete standard cu mărimi electrice de putere.

Schema bloc de comandă generală a unui motor pas cu pas este prezentată în figura 3.1.4 şi reprezintă comanda în circuit deschis (fără reacție) a unui motor pas cu pas. Impulsurile de comandă pot proveni de la un generator propriu, reglabil manual, sau de la un calculator numeric de proces.

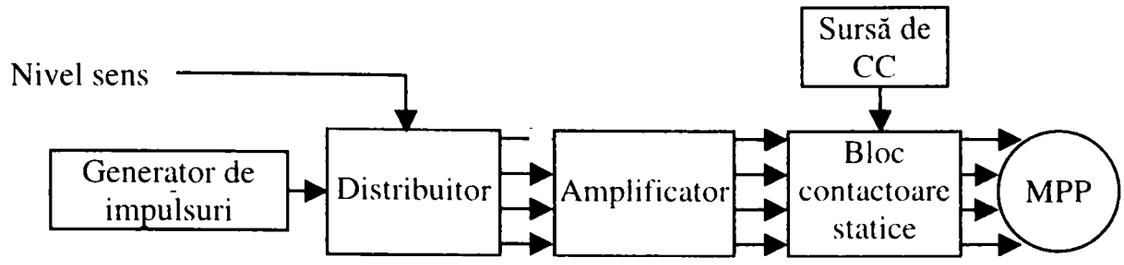


Fig. 3.1.4
Scemă bloc de comandă generală a unui motor pas cu pas

❖ **Blocul contactoarelor statice**

Semnalele de la distribuitor trebuie amplificate pentru a transmite puterea adecvată motorului. În mod uzual, sunt folosite etaje cu tranzistoare de putere cuplate direct sau tiristoare, pentru curenți mai mari. Se deosebesc contactoare statice pentru alimentare unipolară şi contactoare statice pentru alimentare bipolară, de obicei patru tranzistoare de putere în punte.

Înfăşurarea unui motor pas cu pas este o sarcină rezistiv-inductivă, cu inductivitate constantă sau periodic variabilă cu unghiul de rotație. În plus, mişcarea rotorului generează o tensiune electromotoare E, cu sens invers fată de tensiunea de alimentare. O schemă echivalentă este reprezentată în figura 3.1.5.

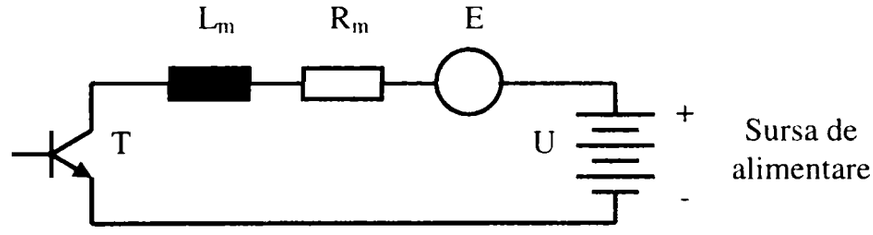


Fig. 3.1.5
Schema echivalentă a unui motor pas cu pas

Rezistența înfăşurării variază datorită toleranțelor de fabricație cu 10%. În plus, rezistența înfăşurării creşte în timpul funcționării motorului, pentru că cele mai multe motoare pas cu pas sunt destinate a funcționa la o temperatură de ordinul a 100° C deci, practic fierbinți. Se acceptă o variație cu temperatura a rezistenței înfăşurării de 25%. Totodată, cele mai multe motoare pas cu pas sunt destinate a lucra în condiții de saturație a circuitului magnetic, mai ales când viteza unghiulară este scăzută, deoarece, în cazurile practice, amplitudinea impulsului de curent prin înfăşurare se

modifică (creşte cu scăderea vitezei de rotaţie). Rezultă, implicit, o variaţie a inductanţei înfăşurării, funcţie de curentul de înfăşurare.

Circuitele de alimentare sunt de curenţi mari, care utilizează tranzistoare de putere capabile să suporte vârfuri de putere. Tranzistoarele se aleg luând în considerare cele mai grele condiţii de lucru. De asemenea, înfăşurările reprezintă sarcini inductive care sunt conectate şi deconectate şi, deci. Tranzistoarele de putere trebuie protejate contra supratensiunilor inductive tranzitorii.

Curentul prin faza motorului are expresia:

$$i(t)=\frac{U}{R_m}\left(1-e^{\frac{1}{T_m}}\right) \tag{3.1.1}$$

unde :

- $T_m=L_m/R_m$ este constanta de timp a fazei
- L_m este inductivitatea proprie medie
- R_m este inductivitatea rezistenţa fazei
- U este tensiunea continuă de alimentare.

Pentru ca motorul pas cu pas să răspundă la frecvenţe cât mai mari este necesar ca timpul de stabilire al curentului la valoarea U/R_m să fie cât mai mic. Blocul contactoarelor statice trebuie să asigure şi forţarea pantei curentului prin fazele motorului.

Tehnicile cunoscute de micşorare a timpului de creştere a curentului prin faze sunt:

- Forţarea prin rezistenţă de serie
- Forţarea prin tensiune
- Forţarea cu rezistenţă şi condensator
- Forţarea tip chopper

Cea mai simplă din punct de vedere constructiv este cea prin rezistenţă serie. Acesta este motivul pentru care această tehnică a fost adaptată în cadrul acestui experimentului.

❖ Forţarea prin rezistenţă serie

Aceasta se bazează pe micşorarea constantei de timp T_m prin inserarea de rezistenţe cu fazele motorului. Constanta de timp se reduce de la L_m/R_m la $L_m/(R_m+R)$,

iar tensiunea de alimentare creşte de la $U=R_m*I$ la $U_1=(R_m+R)*I$, I fiind curentul nominal al fazei.

Dezavantajele metodei constau în randamentul slab, datorat surplusului de putere disipată pe rezistenţa exterioară şi accentuarea oscilaţiilor în răspunsul motorului pas c pas, precum şi a fenomenului de rezonanţă mecanică a motorului.

Schema electrică a interfeţei de comandă a motoarelor pas cu pas utilizată în realizarea practică a robotului efector este prezentată în figura 3.1.6.

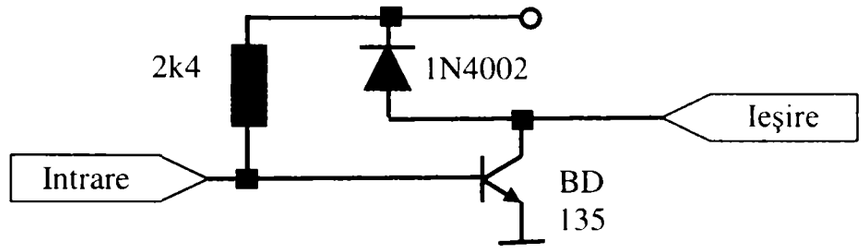


Fig. 3.1.6
Schema electrică a interfeţei de comandă a motoarelor pas cu pas utilizată

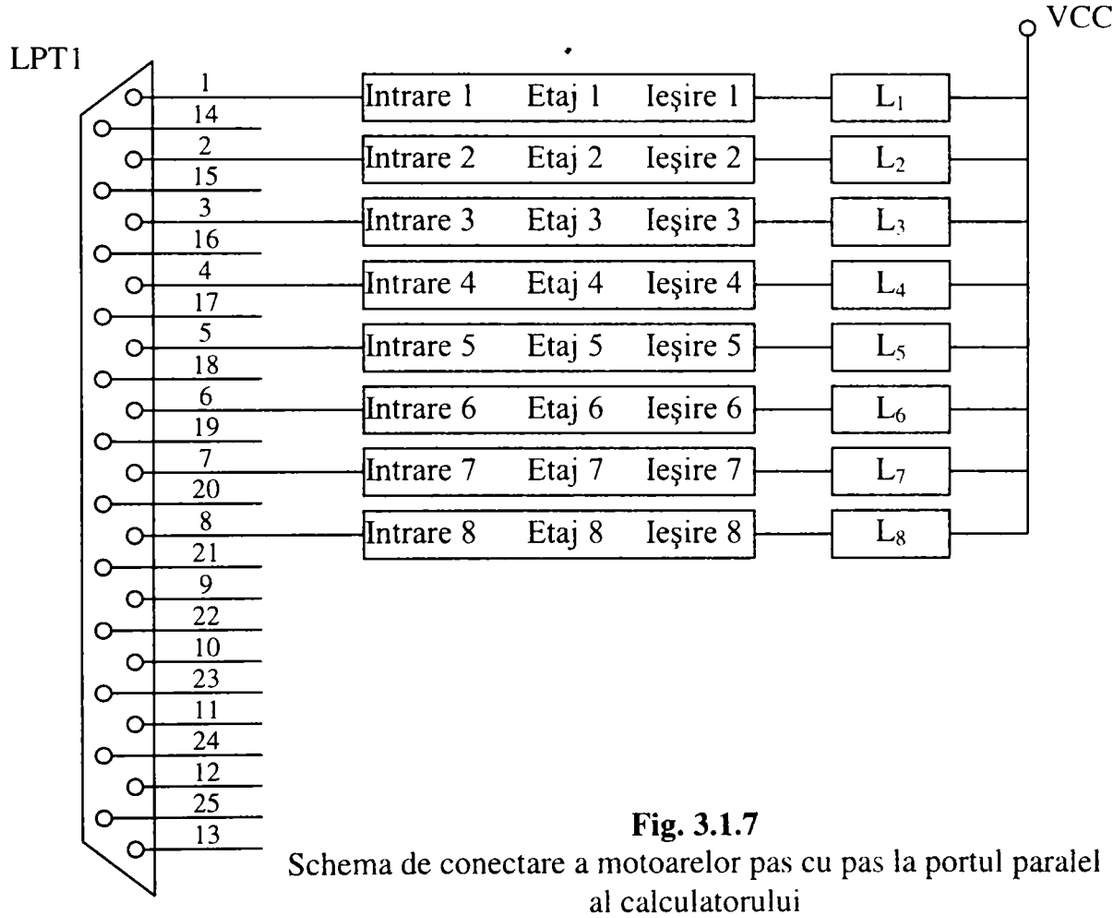


Fig. 3.1.7
Schema de conectare a motoarelor pas cu pas la portul paralel al calculatorului

Schema se compune din 8 blocuri elementare de comandă identice. Borna de intrare a acestor blocuri se leagă la liniile de date ale portului paralel al unui calculator. Rezistorul R are rolul de rezistenţă de colector pentru tranzistorul din portul paralel.

Borna de ieşire se conectează la înfăşurarea corespunzătoare a motorului pas cu pas. Conectarea celor 8 blocuri elementare la portul parale este realizată conform figurii 3.1.7.

3.1.3 Descrierea rutinelor de comandă

Robotul efector trebuie să fie capabil de următoarele mişcări: deplasare înainte, înapoi, viraj la stânga şi la dreapta. Pentru realizarea acestor mişcări se folosesc funcţiile de comandă. Fiind un program Windows este indicat să se utilizeze procedurile de lucru cu fişiere în scopul accesării portului paralel în locul celor stil DOS cu rutinele outp().

Pentru aceasta, trebuie mai întâi deschis şi iniţializat portul paralel:

```
void InitializeLPT1Port(void)
{
 BOOL fSuccess;
 DCB  dcb;

 G_hCom = CreateFile("LPT1",
                     GENERIC_WRITE,
                     0,  // exclusive access
                     NULL, // no security attrs
                     OPEN_EXISTING,
                     0,  // not overlapped I/O
                     NULL
                     );
 if (g_hCom == INVALID_HANDLE_VALUE)
  {
  ShowErrorMessage("LPT1 device cannot be opened\n");
  return;
  }
 fSuccess = GetCommState(g_hCom, &dcb);
 if (!fSuccess)
  {
  ShowErrorMessage("Cannot get LPT1 current configuration\n");
  return;
  }
 dcb.DCBlength = sizeof(DCB);
 dcb.BaudRate = 12358;
 dcb.fBinary = TRUE;
 dcb.fParity = FALSE;
 dcb.fOutxCtsFlow = FALSE;
 dcb.fOutxDsrFlow = FALSE;
 dcb.fDtrControl = DTR_CONTROL_DISABLE;
 dcb.fDsrSensitivity = FALSE;
 dcb.fOutX = FALSE;
 dcb.fInX = FALSE;
 dcb.fRtsControl = RTS_CONTROL_DISABLE;
 dcb.fAbortOnError = TRUE;
 dcb.wReserved = 0;
 dcb.ByteSize = 8;
 dcb.Parity = NOPARITY;
 dcb.StopBits = ONESTOPBIT;
```

```
 fSuccess = SetCommState(hCom, &dcb);
 if (!fSuccess)
  {
  ShowErrorMessage("Cannot set LPT1 new configuration\n");
  return;
  }
}
```

Având în vedere inerția mecanică a părților în mișcare trebuie introduse stări suplimentare de aşteptare pentru a permite motorului să efectueze un pas complet. Funcțiile de rotire al unuia dintre motoare sunt:

```
void RightEngineForward(void)
{
 unsigned char OutChar;
 BOOL          fSuccess;
 DWORD         WrittenBytes;

 OutChar = 0x05;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x09;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x0A;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x06;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}

void RightEngineBackward(void)
{
 unsigned char OutChar;
 BOOL          fSuccess;
 DWORD         WrittenBytes;

 OutChar = 0x01;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x04;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x02;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x08;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}

void LeftEngineForward(void)
{
 unsigned char OutChar;
 BOOL          fSuccess;
 DWORD         WrittenBytes;

 OutChar = 0x50;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x90;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0xA0;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x60;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}
```

```
void LeftEngineBackward(void)
{
 unsigned char OutChar;
 BOOL        fSuccess;
 DWORD       WrittenBytes;

 OutChar = 0x10;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x40;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x20;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x80;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}
```

Pentru mişcarea întregului robot înainte există două posibilităţi: trimiterea comenzilor la fiecare motor separat sau la ambele motoare simultan. Dacă se trimit comenzi separat, mişcarea robotului înainte este generată de următoarea secvenţă:

```
RightEngineForward();
LeftEngineForward();
```

Dacă se comandă simultan ambele motoare se realizează următoarea funcţie:

```
void MoveRobotForward(void)
{
 unsigned char OutChar;
 BOOL        fSuccess;
 DWORD       WrittenBytes;

 OutChar = 0x55;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x99;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0xAA;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x66;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}
```

Aceeaşi situaţie se poate întâlni şi pentru mişcarea înapoi:

```
RightEngineBackward();
LeftEngineBackward();
```

sau

```
void MoveRobotBackward(void)
{
 unsigned char OutChar;
 BOOL        fSuccess;
 DWORD       WrittenBytes;

 OutChar = 0x11;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x44;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x22;
```

```
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x88;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}
```

Pentru rotirea robotului se foloseşte tehnica tancului. Una din roţi se învârte înainte cealaltă înapoi. Şi aici există două variante de comandă: separată şi simultană.

Rotirea la stânga:

```
RightEngineForward();
LeftEngineBackward();
```

sau

```
void RotateRobotLeft(void)
{
 unsigned char OutChar;
 BOOL        fSuccess;
 DWORD       WrittenBytes;

 OutChar = 0x15;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x49;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x2A;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x86;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}
```

Rotirea la dreapta:

```
RightEngineBackward();
LeftEngineForward();
```

sau

```
void RotateRobotRight(void)
{
 unsigned char OutChar;
 BOOL        fSuccess;
 DWORD       WrittenBytes;

 OutChar = 0x51;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x94;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0xA2;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
 OutChar = 0x68;
 fSuccess = WriteFile(g_hCom,&OutChar,1,&WrittenBytes,NULL);
}
```

3.1.4 Recunoaşterea şi maparea spaţiului de lucru

Având o cameră poziţionată deasupra spaţiului de lucru, se realizează achiziţia unei imagini folosind procedurile librăriei Video for Windows. Deoarece imaginea

achiziționată va fi o imagine color, iar procedurile de recunoaştere lucrează pe imagini binare, ea va trebui preprocesată. Astfel, se va converti imaginea color la o imagine cu 256 nivele de gri şi se va filtra median pentru atenuarea zgomotelor.

În continuare se vor aplica paşii descrişi în capitolele de descriere şi recunoaştere (2.2 şi 2.3) ale tezei, paşi care vor genera definirea figurilor geometrice şi robotului prin puncte precum şi identificarea lor.

Având identificarea făcută se poate determina raportul pixeli/cm deoarece se cunoaşte dinainte dimensiunea în centimetri a robotului, iar în urma recunoaşterii de imagini se cunosc şi dimensiunile în pixeli ale acestuia. Astfel, se pot determina cu uşurință dimensiunile în cm ale figurilor geometrice precum şi distanțele între ele. Având descrierea spațiului de lucru, se poate determina drumul ce trebuie să-l parcurgă robotul pentru a rezolva problema propusă.

- **Aspecte ale planificării mişcării**

În general, robotul inteligent trebuie să fie capabil să-şi planifice propriile mişcări, şi anume să decidă automat ce mişcări să execute pentru a realiza o sarcină specificată prin aranjamentul inițial şi final al obiectelor din spațiul de lucru. Crearea roboților autonomi este o sarcină majoră în Robotică. Cu excepția anumitor domenii limitate, nu este real să se anticipeze şi să se descrie explicit toate mişcările pe care robotul trebuie să le execute pentru a realiza sarcina cerută. Chiar şi în cazurile când o asemenea descriere este posibilă, este util să se încorporeze dispozitive de planificare automată a mişcărilor în sistemele de programare off-line ale robotului. Aceasta permite utilizatorului să specifice sarcini stabilind mai degrabă ce doreşte să realizeze decât cum să o facă. Deci robotul trebuie să realizeze mişcarea dorită şi să-şi activeze diferitele mecanisme în acord cu sarcina cerută.

Planificarea mişcărilor unui robot prezintă o varietate neaşteptată de aspecte dificile de calcul. De fapt, inteligența operativă pe care oamenii o utilizează inconştient pentru a interacționa cu mediul înconjurător, necesară percepției şi planificării mişcării, este foarte dificil de reprodus într-un program de calcul. Astfel, o problemă importantă în planificarea mişcărilor este complexitatea algoritmilor de calcul [32], [33], [40], [42], [47], [48], [51], [52], [53], [66], [68], [69], [75], [78], [80], [81], [84], [85], [126], [130] ,[144], [147].

Aşa cum s-a precizat, planificarea mişcării este o problemă în crearea roboților autonomi, dar nu este singura. Aceasta interacționează cu alte probleme importante,

ca de exemplu: analiza cinematică directă şi inversă, modelul dinamic direct şi invers, generarea traiectoriilor, controlul mişcării în timp real, sistemul şi planificarea sarcinii [39], [41], [45], [46], [55], [57],[62],[63],[74], [77], [127].

Scopul definirii problemei de bază a planificării mişcărilor este de a izola anumite probleme centrale şi de a le studia în profunzime înainte de a considera anumite dificultăţi adiţionale.

Cea mai simplă problemă de planificare presupune că robotul este singurul obiect în mişcare în spaţiul de lucru, care nu posedă proprietăţi dinamice evitând astfel problemele temporale. Se consideră de asemenea că robotul nu intră în contact cu obiectele înconjurătoare, evitând astfel probleme legate de interacţiunea mecanică dintre două obiecte fizice. Aceste consideraţii transformă problema planificării "fizice" a mişcării într-o problemă pur geometrică. Mai mult, se consideră că robotul este singurul solid rigid a cărui mişcare este limitată doar de obstacole.

Cu aceste simplificări, problema de bază a planificării traiectoriilor unui robot se poate formula astfel:

- Fie A un singur solid rigid (robotul) care se mişcă într-un spaţiu Euclidian W, numit spaţiu de lucru, reprezentat prin R^N, cu N=2 sau 3; mişcarea lui A nu este limitată de nici o restricţie cinematică.
- Fie $B_1,\ldots,B_q$ obiecte rigide fixe (obstacole) distribuite în poziţii bine determinate în spaţiul de lucru W.
- Cunoscând poziţia şi orientarea, la momentul iniţial, ale robotului, precum şi poziţia finală şi orientarea finală pentru acesta în spaţiul de lucru W, să se genereze o traiectorie T, care să specifice o secvenţă continuă de poziţii şi orientări ale lui A, pornind de la configuraţia (poziţia şi orientare) iniţială, evitând contactul cu obstacolele B_j şi terminând în configuraţia finală.
- Dacă o astfel de traiectorie nu există, trebuie să se raporteze eroare.

Este evident că, deşi problema de bază a planificării este super simplificată, ea este totuşi o problemă dificilă cu mai multe soluţii şi cu extensii directe spre probleme mai complicate [36], [38], [44], [49], [50], [64], [65], [67], [72], [129].

Formularea problemei de bază a planificării traiectoriilor se bazează pe anumite presupuneri care limitează semnificativ utilizarea soluţiilor. Este foarte dificil să se reducă o problemă reală de robotică la problema de bază, să se rezolve această

problemă şi să se adapteze soluţiile obţinute astfel încât să se potrivească condiţiilor problemei reale.

Problema de bază a planificării traiectoriei presupune că robotul parcurge cu exactitate traiectoria generată de planificator. De asemenea, se presupune că, atât geometria robotului, cât şi geometria şi poziţiile obstacolelor sunt cunoscute cu exactitate. În realitate, nici o problemă de planificare nu satisface aceste ipoteze. Mai mult, controlul roboţilor şi modelele geometrice ale acestora nu sunt exacte. În condiţiile în care robotul nu deţine informaţii apriorice despre spaţiul de lucru, acesta trebuie să se bazeze, în timpul execuţie, pe sistemul său senzorial pentru înregistrarea informaţiilor necesare realizării sarcinii. Se impune astfel explorarea spaţiului de lucru şi rezolvarea problemei de planificare în prezenţa incertitudinilor [34], [35], [43]. [54], [56], [82], [83], [128].

- **Modelarea spaţiului de lucru**

Stabilirea unei hărţi de navigare în spaţiul considerat se poate face fie independent de orice acţiune a robotului prin memorarea unei configuraţii date, care de regulă, reprezintă starea iniţială a spaţiului de lucru, fir pe baza informaţiilor înregistrate de sistemul senzorial.

Pe baza cunoştinţelor acumulate despre spaţiul de lucru, acesta se împarte în celule independente. Aceste celule reprezintă zonele admise respectiv interzise pentru robot. Această operaţie se numeşte ***modelarea spaţiului de lucru al robotului.*** Pe o astfel de modelare se bazează procesul propriu-zis de planificare a mişcărilor.

Pentru a se realiza o modelare corecta spaţiului de lucru, unul din principalele obiective care trebui luate în considerare este ansamblul dimensiunilor robotului. Acestea vor modifica dimensiunile obstacolelor, rezultând aşa numitele C-obstacole, după care robotul se poate trata ca un punct în mişcare [36]. Având reprezentate obstacolele în forma în care ele se iau în considerare, se poate proceda la împărţirea spaţiului de lucru în zone accesibile şi zone interzise. Există mai multe metode pentru realizarea modelării spaţiului : metoda grilei omogene, metoda arborelui, metoda grilei neomogene, metoda poligoanelor convexe, metoda punctelor de trecere.

Pentru realizarea prezentelor experimentări s-a considerat metoda grilei neomogene ca fiind adecvată studierii geometrice plane a spaţiului modelat pentru deplasarea robotului.

Această metodă presupune toate obstacolele de formă dreptunghiulară. Dacă ele nu au această formă se vor aproxima printr-un dreptunghi acoperitor. Existând n obstacole în spațiul de lucru, acesta se împarte printr-o grilă rezultată din prelungirea laturilor obstacolelor, obținându-se $(2n+1)^2$ dreptunghiuri sau, altfel spus, $2n+1$ benzi orizontale şi $2n+1$ benzi verticale (figura 3.1.8).

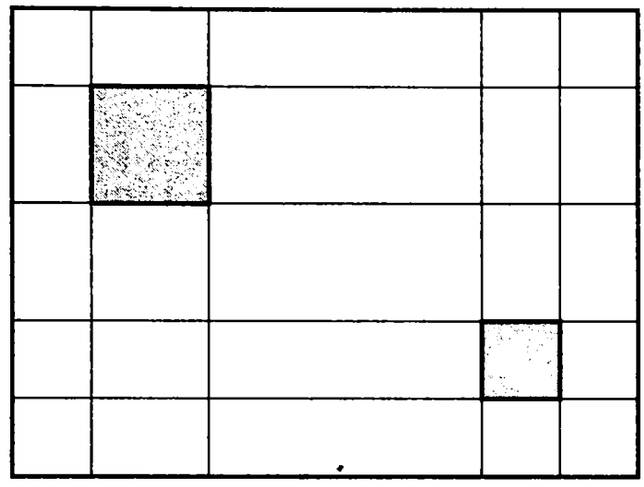


Fig. 1.3.8
Principiul metodei grilei neomogene

Fiecare regiune este reprezentată prin două lanțuri binare de câte $2n+1$ biți, dintre care unul reprezintă poziția relativă x, iar celălalt poziția relativă y. Un bit are valoarea "1" dacă celula este liberă şi "0= în caz contrar pentru primul lanț. Al doilea lanț conține "1" pentru poziția verticală a celulei. Spre exemplu: 01000 00100

- Lanțul 01000 caracterizează situația în care obstacolul este pe poziția a doua pe orizontală, pornind de la stânga;
- Lanțul 001000 caracterizează situația în care obstacolul este al treilea pe verticală, pornind de sus.

De asemenea, pentru a cuprinde mai multe regiuni:

11000 00110

se realizează tablourile echivalente:

10000 00100

01000 00100

10000 00010

01000 00010

Modelarea se realizează în şase etape (un exemplu fiind prezentat în figura 3.1.9):

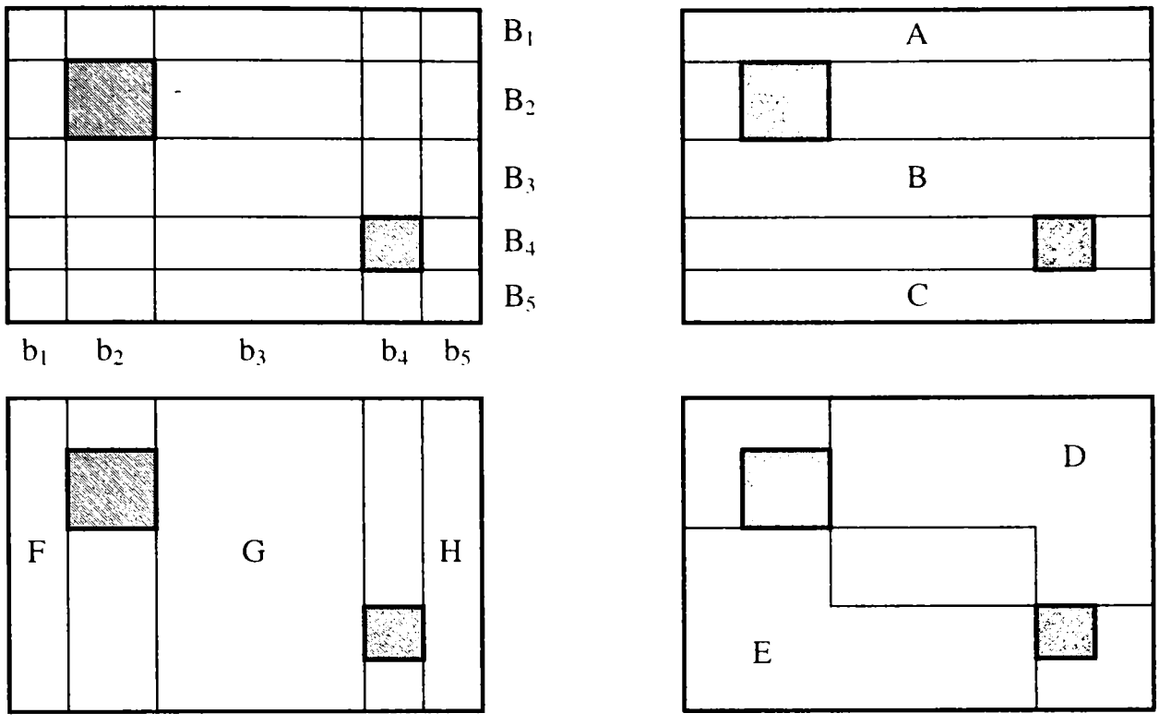


Fig. 1.3.9
Etapele modelării

Etapa 1: Reprezentarea fiecărei benzi orizontale prin $2n+1$ biți. Un bit are valoarea "1" dacă celula este liberă şi "0" în caz contrar. Un al doilea lanț conține un singur "1" care corespunde la poziția verticală a benzii.

Etapa 2: Determinarea tuturor benzilor orizontale continue rezultate din separarea primului lanț în mai multe, fiecare având o suită de "1". De exemplu:

11101 00010

devine:

11100 00010

00001 00010

Etapa 3: gruparea lanțurilor generate în etapa a doua, astfel:

- Lanțurile să fie regrupate în raport cu benzile care le-au generat;
- Grupele să fie ordonate în raport cu poziția verticală a benzilor generatoare.

Etapa 4: Generarea unei noi liste pornind de la cea precedentă conform următoarelor reguli:

- Noua grupă i de lanțuri este generată prin combinarea fiecărui lanț din veche grupă i cu fiecare lanț din vechea grupă $i+1$ (i=1,2,...);

- Două lanțuri sunt combinate printr-un ŞI logic pentru lanțurile din stânga şi SAU logic pentru lanțurile din dreapta; dacă toți biții sunt nuli, lanțul respectiv se elimină, dacă nu, se include pe listă;
- De fiecare dată când un lanț este adăugat la noua listă se elimină lanțurile de pe lista precedentă care sunt acoperite de această adăugare; un lanț S_1 este acoperit de către S_2 dacă un SAU logic între ele conduce la S_2.

Etapa 5: Se repetă etapa 4 dacă noua listă are două sau mia multe grupe.

Etapa 6: Se stabilesc lanțurile neeliminate de pe liste care reprezintă o zonă fără obstacole.

Lista 1 (Etapa 1)

b_1 11111 10000 B_1

b_2 10111 01000 B_2

b_3 11111 00100 B_3

b_4 11101 00010 B_4

b_5 11111 00001 B_5

Lista 2 (Etapa2+3)

11111 10000 A

10000 01000

00111 01000

11111 00100 B

11100 00010

00001 00010

11111 00001 C

Lista 3 (Etapa 4)

10000 11000

00111 11000

10000 01100

00111 01100

11100 00110

00001 00110

11100 00011

00001 00011

Lista 4 (Etapa 5)

10000 11100

00111 11100 D

Lista 5 (Etapa 5)

10000 11110

00100 11110

00001 11110

10000 01110

00100 01110

00001 01110

10000 01111

00100 01111

00001 01111

11100 00111 E

00001 00111

Lista 6 (Etapa 5)

10000 11111 F

00100 11111 G

00001 11111 H

Metodei astfel descrise i se poate asocia un graf ale cărui noduri sunt zonele fără obstacole, aşa cum este prezentat în figura 3.1.10.

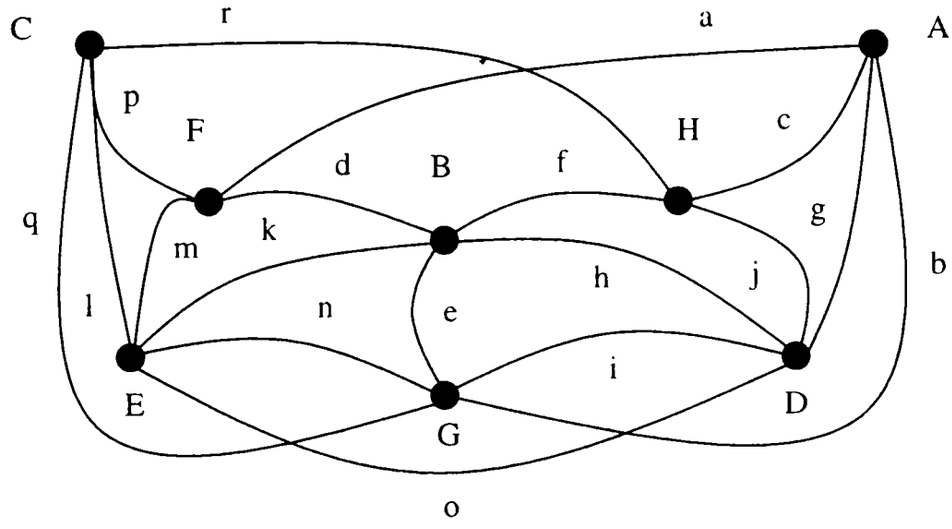
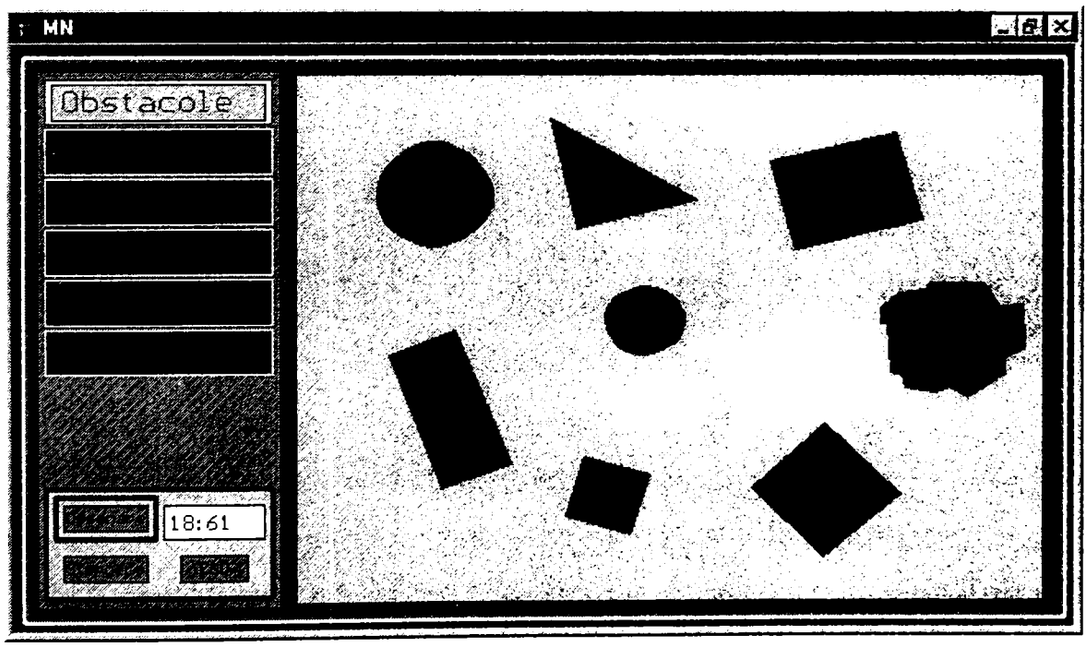


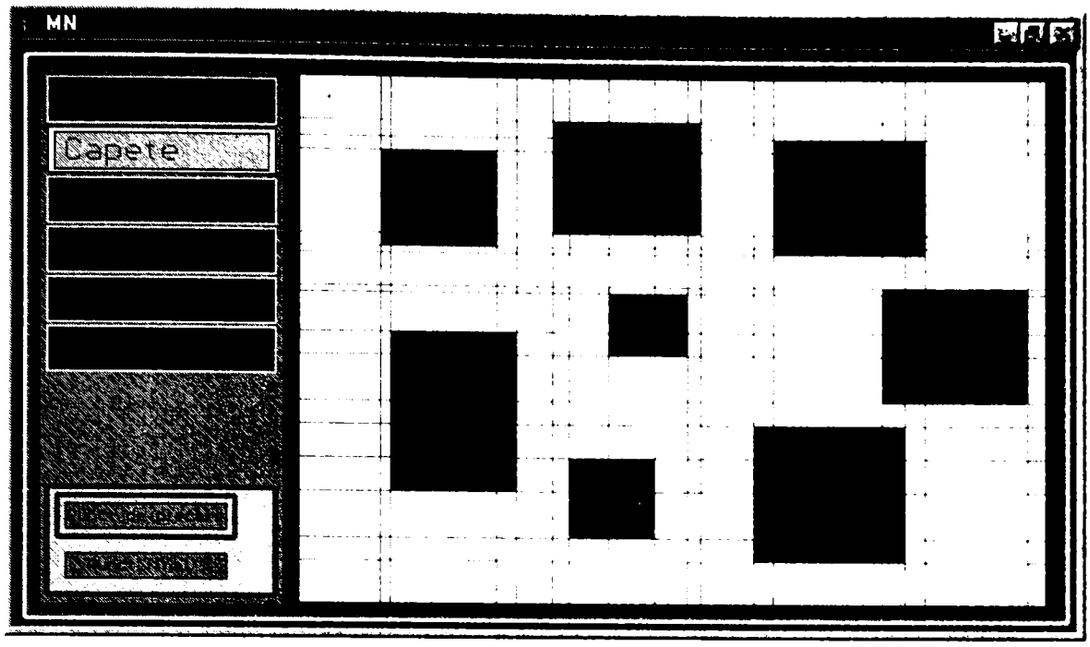
Fig. 3.1.10
Graful metodei grilei neomogene

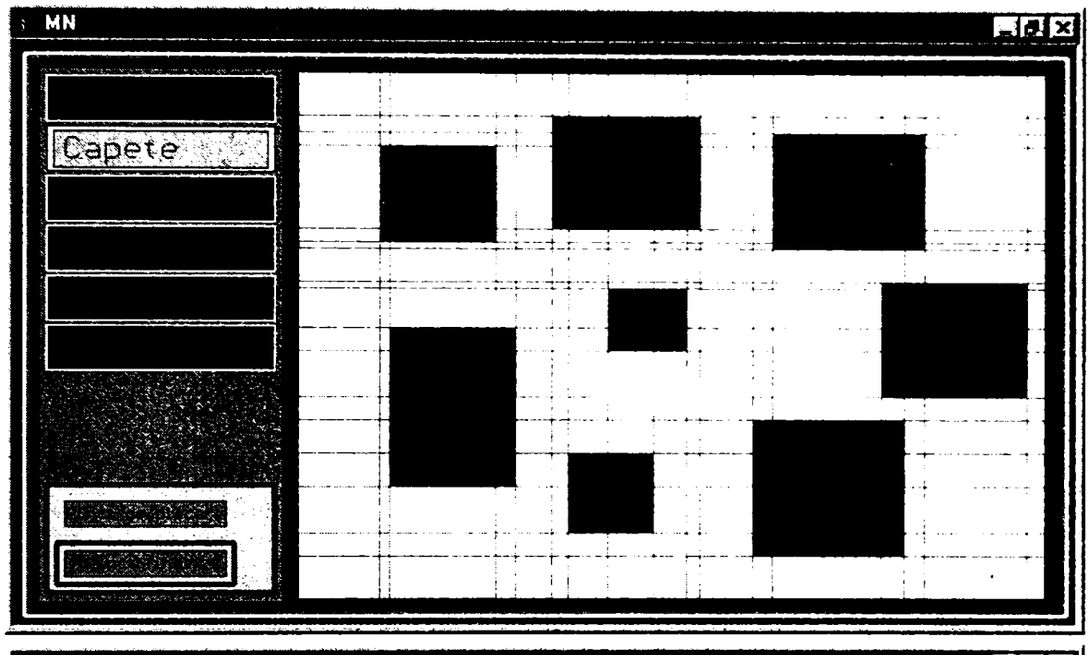
arce	noduri	intersecții
a	AF	10000 10000
b	AG	00100 10000
c	AH	00001 10000
d	BF	10000 00100
e	BG	00100 00100
f	BH	00001 00100
g	AD	00111 10000

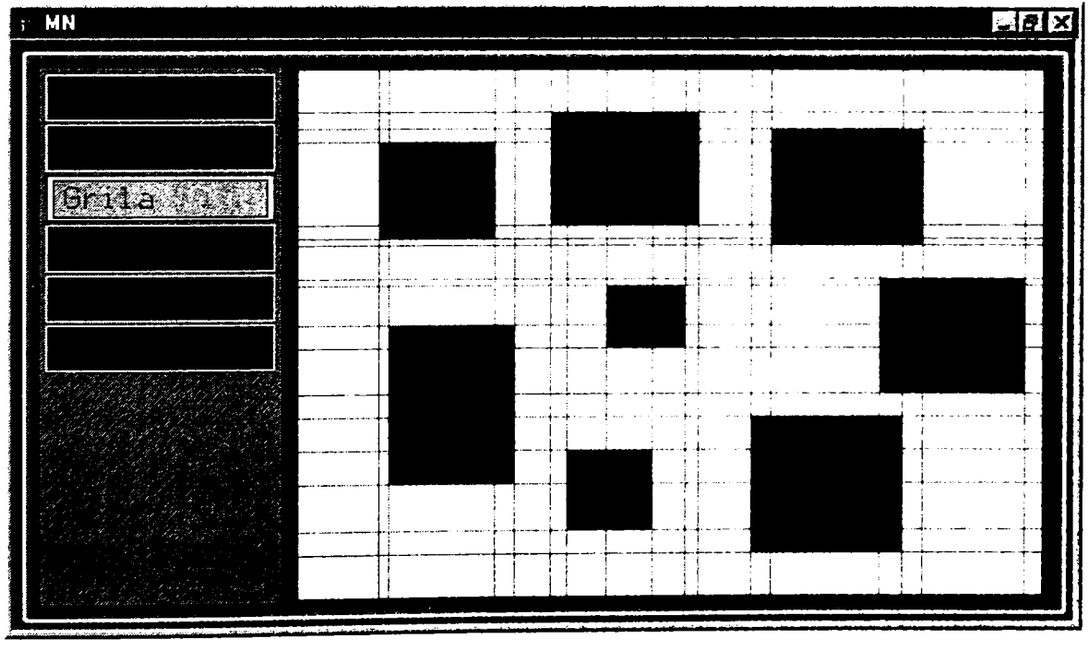
h	BD	00111 00100
i	GD	00100 00111
j	HD	00001 00111
k	BE	11100 00100
l	CE	11100 00001
m	FE	10000 00111
n	GE	00100 00111
o	DE	00100 00100
p	CF	10000 00001
q	CG	00100 00001
r	CH	00001 00001

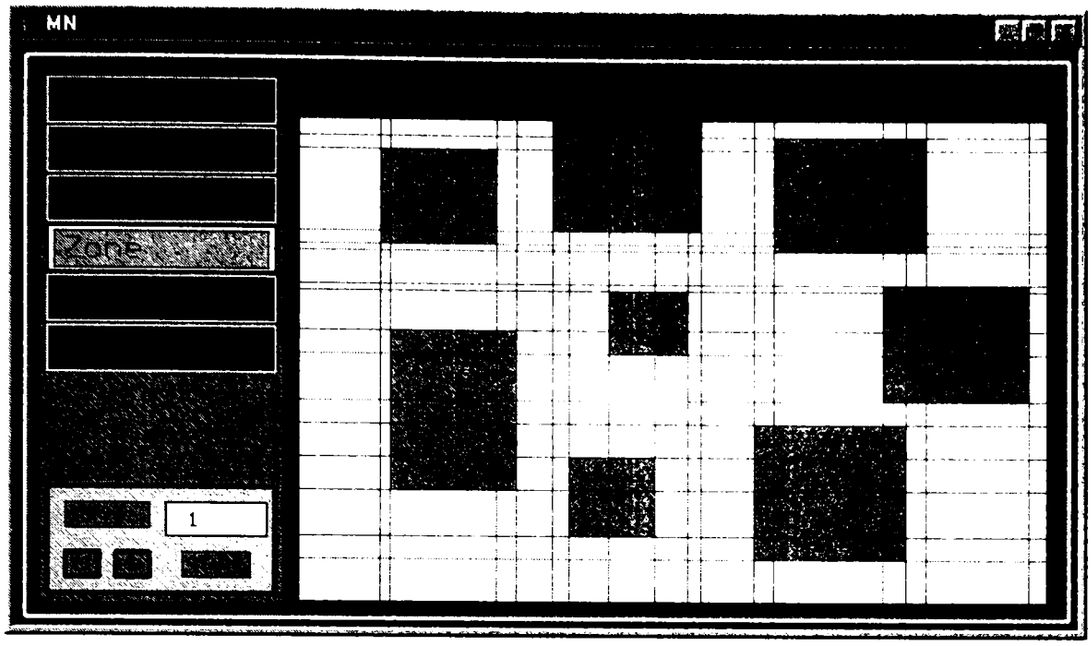
Aplicarea algoritmului pe spațiul de lucru ales este prezentată în figura 3.1.11. S-au eliminat în această figură anumite imagini de detecție de zone pentru că prezentarea integrală a etapelor era prea voluminoasă (35 de imagini diferite). În figură apar în diferite etape ale algoritmului, zone desenate în negru şi care reprezintă zone integral libere şi prin care traiectoria robotului poate trece. Elc corespund nodurilor din graful asociat. Excepție la această observație o constituie prima reprezentare di figura 3.1.11 în care prin culoarea neagră sunt prezentate obstacolele plasate pe placa albă de polistiren

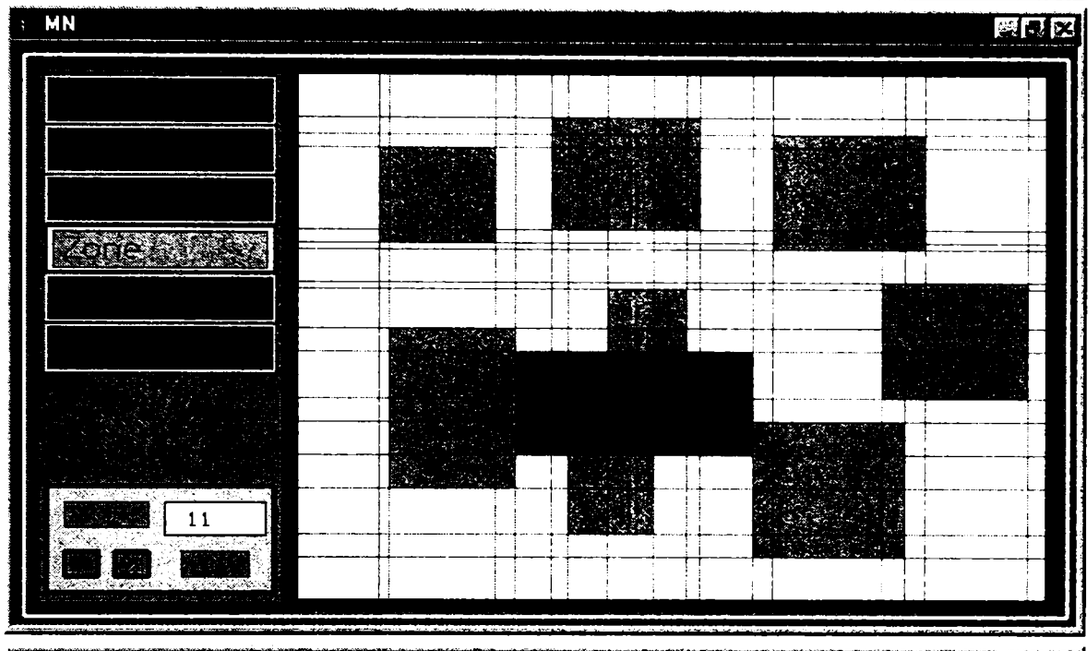


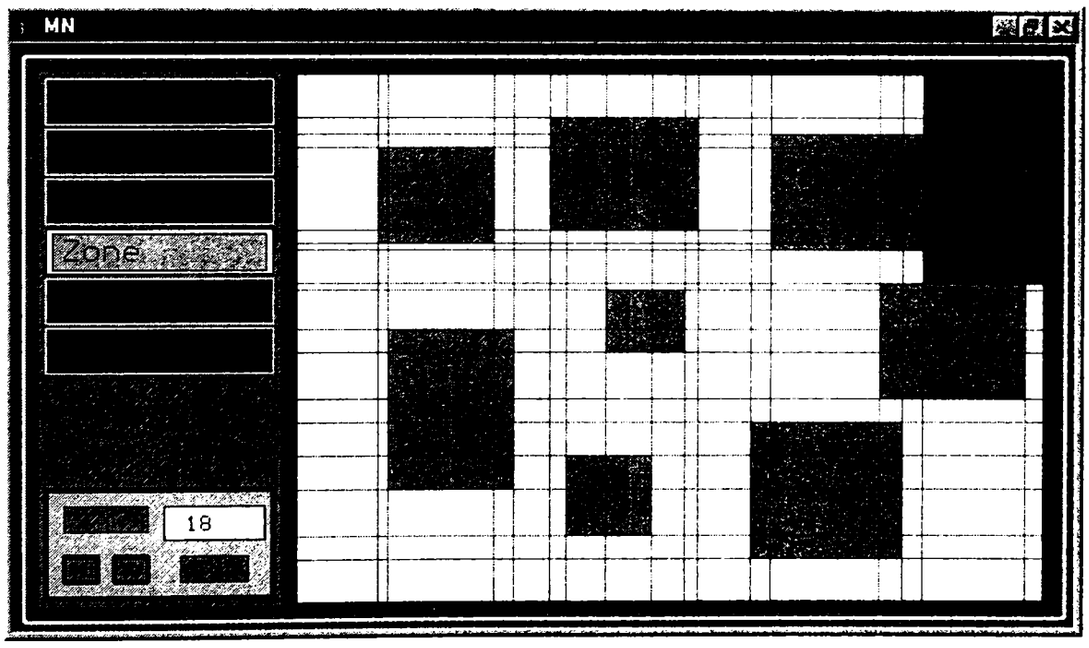












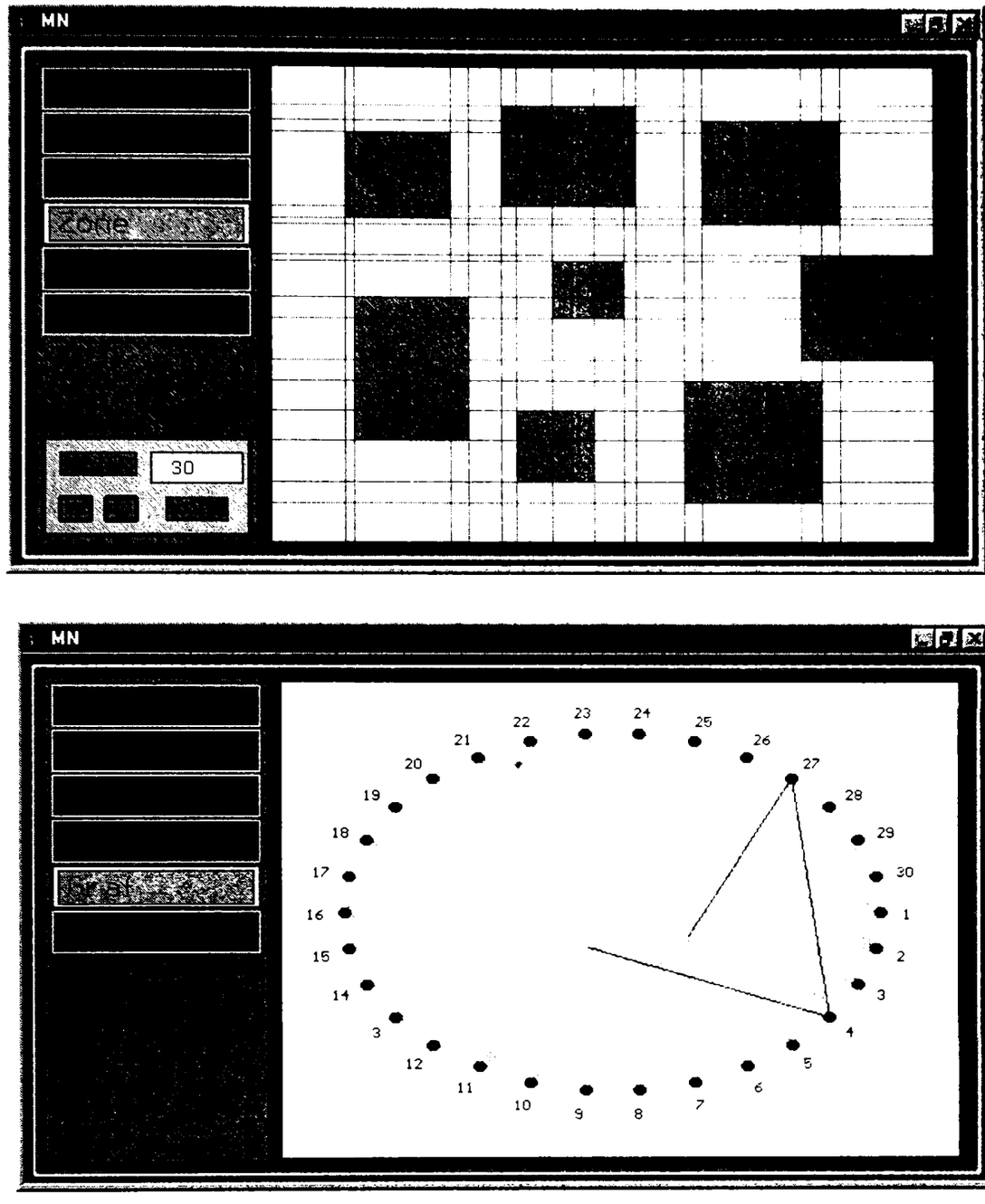


Fig. 3.1.11
Exemplificarea practică a algoritmului metodei grilei neomogene

S-au realizat experimente de deplasare a robotului descris anterior pe mai multe traiectorii posibile conform grafului. Variantele de traiectorie s-au stabilit în funcție de pozițiile inițială şi finală impuse robotului.

➢ **Metode de planificare**

Metoda grilei neomogene deşi este în principiu una de modelare a spaţiului permite şi planificarea succesiunii de celule goale prin care robotul poate trece. Ea poate deci realiza o planificare primară.

Problema planificării propriu-zise a traiectoriilor în prezenţa obstacolelor, poate fi abordată în două moduri (global şi 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, 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 geometrie 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 punct mobil liber care evoluează printre obstacolele transformate [37], [[58], [59], [70], [79].

Astfel, planificarea traiectoriilor unui robot comportă, în general, două etape:

- Transformarea obstacolelor (determinarea C-obstacolelor) astfel încât robotul să poată fi considerat un punct material.
- Cercetarea unei traiectorii, fără coliziune, pentru acest punct.

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

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

Din mulțimea metodelor de planificare cunoscute s-a ales pentru simplitatea algoritmului metoda grafului vizibilității.

Principiul metodei constă din construirea unei traiectorii semilibere ca o linie poligonală ce conectează configurația inițială $\mathbf{C}_{\text{inițial}}$ de configurația finală $\mathbf{C}_{\text{final}}$, trecând prin vârfuri ale spațiului configurațiilor obstacolelor (SCB) [36], [60], [61], [71],[73].

Se consideră un obiect poligonal *A* ce translatează cu orientare fixă în spațiul de lucru bidimensional $W=\mathbf{R}^2$. În acest caz, spațiul configurațiilor lui *A* este $SC=\mathbf{R}^2$, iar regiunea SCB a C-obstacolelor este o regiune poligonală a lui $\mathbf{R}^2$. Spațiul SC_{liber} este completarea lui SCB pe SC.

Dacă există o traiectorie, atunci cu siguranță va exista şi o traiectorie semiliberă *T* de lungime euclidiană minimă între aceste două configurații. Pentru ca traiectoria *T* să fie cea mai scurtă, ea trebuie să fie formată din subtraiectorii care să fie cele mai scurte posibil între cele două configurații date. Aceasta înseamnă că, în orice configurație **C**, traversată de *T*, dacă **C** nu este un vârf al lui SCB, *T* trebuie să aibă curbura nulă. Deci, vârfurile traiectoriei poligonale *T* sunt vârfuri ale lui SCB.

Este evident că pentru a găsi o traiectorie semiliberă între oricare două configurații este suficient pentru un planificator să considere liniile poligonale ce unesc vârfurile lui SCB. Această mulțime de linii poligonale, numită ***graf al vizibilității***, conține cu siguranță şi cea mai scurtă traiectorie, aşa cum este prezentat în figura 3.1.12.

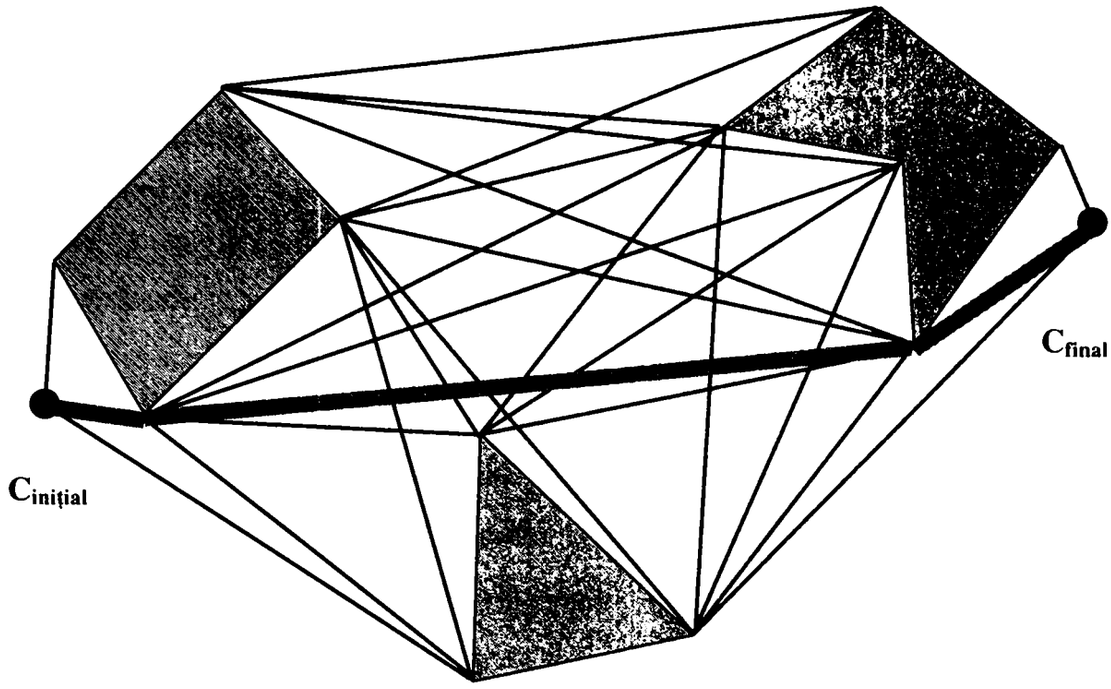


Fig. 3.1.12
Construcţia grafului vizibilităţii

Figura ilustrează graful vizibilităţii în cazul unui spaţiu al configuraţiilor simplu, în care SCB constă din trei regiuni separate. Legăturile grafului include şi muchiile lui SCB. Traiectoria cea mai scurtă între $\mathbf{C}_{iniţial}$ şi $\mathbf{C}_{final}$ este reprezentată cu linie îngroşată.

Din cele precizate rezultă că graful vizibilităţii este graful neorientat (GV), definit astfel :

- Nodurile lui (GV) sunt $\mathbf{C}_{iniţial}$, $\mathbf{C}_{final}$, şi vârfurile lui SCB;
- Două noduri ale lui (GV) sunt conectate printr-o legătură, dacă şi numai dacă, fie segmentul de dreaptă ce le uneşte este o muchie a lui SCB, fie se află în întregime în SC_{liber}, condiţie de la care sunt exceptate cele două capete [111].

❖ **Algoritmul metodei**

Algoritmul metodei grafului vizibilităţii conţine în mod firesc următoarele etape:

- Se construieşte graful vizibilităţii (GV);
- Se cercetează (GV) pentru a găsi o traiectorie între configuraţia iniţială şi cea finală;

- Dacă se găseşte o traiectorie, se comunică aceasta, iar în caz contrar se comunică "eşec".

Un algoritm primitiv de construire a lui (GV) ia în considerare toate perechile de puncte(X, X'), unde X şi X' sunt fie $\mathbf{C_{iniţial}}$ şi $\mathbf{C_{final}}$, fie vârfuri ale lui SCB.

Dacă X şi X' sunt capetele aceleiaşi muchii a lui SCB, atunci nodurile corespunzătoare sunt conectate printr-o legătură în (GV). În caz contrar, se calculează intersecţia dreptei ce trece prin X şi X' cu SCB; nodurile corespunzătoare lui X şi X' sunt conectate printr-o legătură în (GV) dacă şi numai dacă nici o intersecţie nu se află în segmentul deschis ce uneşte cele două puncte.

Algoritmul poate fi îmbunătăţit astfel:

- Pentru fiecare punct X din (GV) se calculează orientarea $\alpha_i \in [0,2\pi]$ a fiecărei semidrepte ce pleacă din X şi trece prin alt punt X_i a lui (GV) şi se sortează orientările lui α_i;
- Se roteşte o semidreaptă ce pleacă din X de la orientarea 0 la orientarea 2π, oprindu-se la fiecare orientare α_i; la fiecare oprire, se reînoieşte intersecţia semidreptei cu SCB, se păstrează urma punctului în intersecţia cea mai apropiată de X şi se verifică dacă segmentul ce conectează X şi X_i intersectează SCB.

❖ Graful vizibilităţii redus

Eficienţa metodei grafului vizibilităţii poate fi crescută observând că unele legături nu sunt necesare. Pentru aceasta este însă necesar să se precizeze câteva noţiuni [32], [33], [77].

Se numeşte ***sector convex*** al unei traiectorii poligonale T în oricare vârf al său, regiunea convexă închisă cuprinsă între două semidrepte trasate prin X şi conţinând segmentele de dreaptă aparţinând lui T şi adiacente în X.

Pentru ca o traiectorie poligonală T să fie cea mai scurtă, fiecare vârf X al lui T trebuie să aibă o vecinătate U astfel încât $SCB \cap U$ să fie în întregime conţinută în sectorul convex al lui T în X.

Dacă $SCB \cap U$ nu este conţinută în sectorul convex al lui T în X, atunci T se poate modifica uşor într-o traiectorie semiliberă mai scurtă.

Astfel, porţiunea de traiectorie din figura 3.1.13a nu satisface această condiţie, deci poate fi scurtată (linia întreruptă), în timp ce porţiunea de traiectorie din figura 3.1.13b verifică această condiţie, şi deci nu poate fi scurtată.

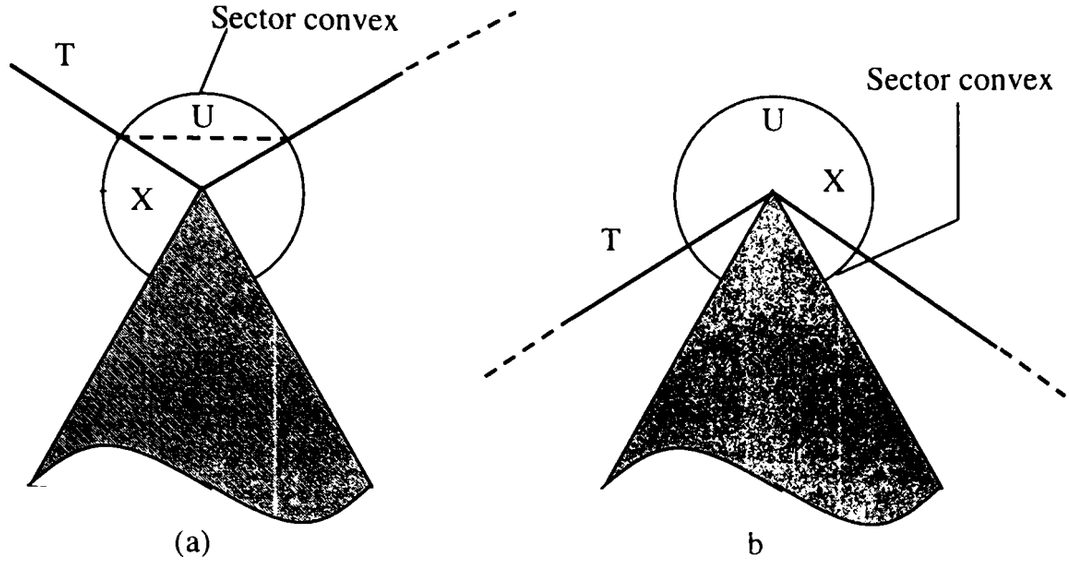


Fig. 3.1.13
Noțiunea de sector convex

Pentru simplificarea suplimentară a grafului vizibilității mai este necesară tratarea noțiunii de ***segment tangent la* SCB *într-un vârf* X**.

Astfel, dacă X este un vârf al lui SCB şi L este un segment de dreaptă ce trece prin X, L este ***tangent*** la SCB în x dacă şi numai dacă într-o vecinătate U a lui X interiorul lui SCB se află în întregime de o singură parte a lui L.

Se consideră două noduri X şi x' ale lui (GV). Segmentul ce uneşte X cu X' se numeşte ***segment tangent*** dacă şi numai dacă:

- X fiind un vârf al lui SCB, L este tangent la SCB în X;
- X' fiind un vârf al lui SCB, L este tangent la SCB în X.

  În figura 3.1.14 sunt prezentate segmentele tangente cu linii continue şi segmentele netangente cu linii întrerupte. Figura ilustrează două proprietăți simple ale segmentelor tangente:
- Între două poligoane convexe disjuncte există exact patru segmente tangente (figura 3.1.14,a) :
  - Două sunt segmente tangente ***"suport"***, cele două poligoane aflându-se de aceeaşi parte a dreptei care conține segmentul;
  - Două segmente tangente ***"separatoare"***, cele două poligoane aflându-se pe părți diferite ale dreptei care conține segmentul.
- Dacă un vârf X este concav, atunci nici un segment tangent nu se termină în X (figura 3.1.14,b).

Se poate proceda astfel la o simplificare a lui (GV) observând că, dacă un segment între X şi X' nu este tangent, el nu trebuie introdus în graful de cercetat.

Subgraful (GV'), obținut din (GV) prin înlăturarea tuturor segmentelor netangente şi a tuturor vârfurilor concave, se numeşte ***graful vizibilității redus***.

De câte ori există o traiectorie semiliberă între $\mathbf{C_{inițial}}$ şi $\mathbf{C_{final}}$, (GV') conține cea mai scurtă t...e ...o.. în.. ac ste ...nfigurații.

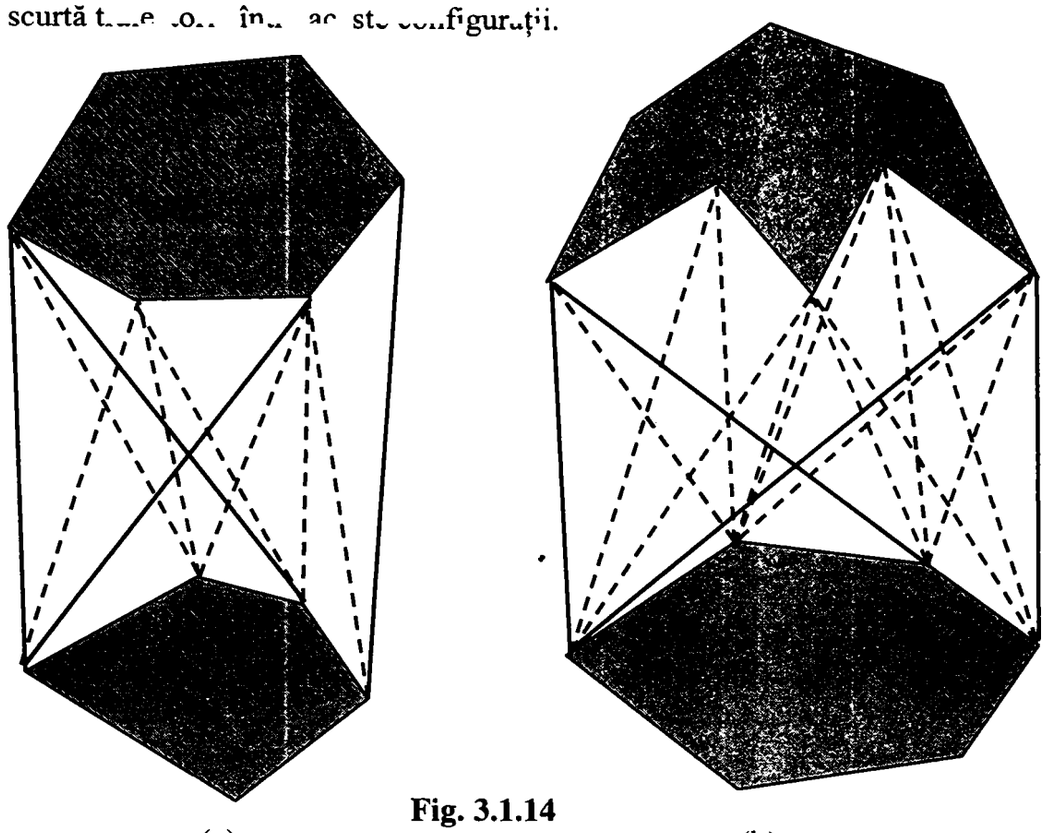


Fig. 3.1.14
(a) Segmente tangente (b)

Figura 3.1.15 prezintă graful vizibilității redus pentru acelaşi spațiu al configurațiilor ca şi figura 3.1.12 şi se observă că el este mult simplificat. Aceasta atrage şi avantajul unui timp de operare redus.

Graful vizibilității redus este graful neorientat (GV') definit astfel:

- nodurile sunt $\mathbf{C_{inițial}}$, $\mathbf{C_{final}}$ şi toate vârfurile convexe ale lui SCB
- legăturile sunt muchiile lui SCB ce conectează vârfurile convexe şi segmentele tangente între $\mathbf{C_{inițial}}$, $\mathbf{C_{final}}$ şi vârfurile lui SCB ce se află în spațiul liber.

Metoda grafului vizibilității poate fi extinsă în cazul în care C-obstacolele sunt poligoane generalizate, adică regiuni mărginite de segmente de dreaptă şi/sau arce de cerc. Astfel de C-obstacole apar când *A* este un poligon generalizat care translatează printre obstacole modelate de asemenea ca poligoane generalizate. Modelarea obiectelor din spațiul de lucru ca poligoane generalizate este mai frecvent utilizată

deoarece caracterizează cu fidelitate conturul acestora şi conduce la o precizie mai bună a metodei de pl n fi .

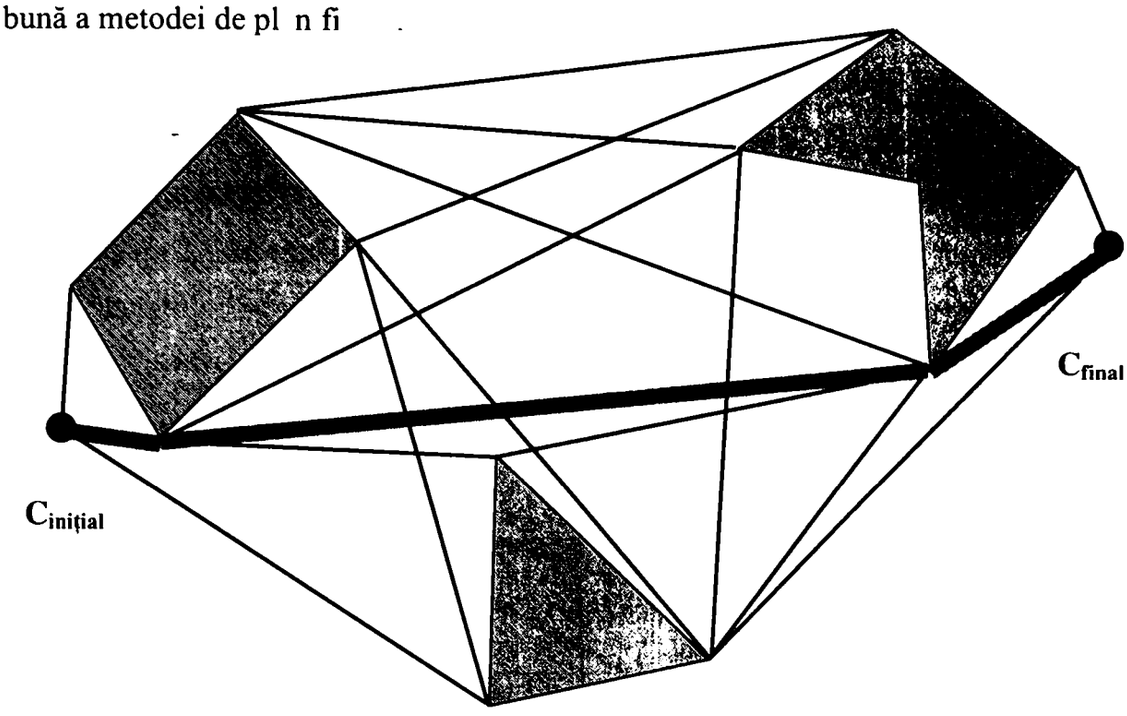


Fig. 3.1.15
Grafului vizibilităţii redus (GV')

Se consideră SC=$\mathbf{R}^2$ populat cu C-obstacole de tip poligoane generalizate, a căror reuniune este SCB.

Graful vizibilităţii (redus) generalizat (GV'G) este definit astfel:

- $\mathbf{C_i}$, $\mathbf{C_r}$ şi vârfurile convexe ale lui SCB sunt nodurile lui (GV'G);
- Fie X şi X' două noduri din cele definite mai sus. Dacă segmentul deschis care uneşte X şi X' se află integral în SC_{liber} şi este un segment tangent la SCB, atunci el este o legătură a lui (GV'G);
- Fie X un nod din cele definite mai sus şi E' o latură circulară a lui SCB. Dacă există un punct X' care aparţine lui E', astfel încât segmentul deschis care uneşte X şi X' se află integral în SC_{liber} şi dreapta suport a acestui segment este tangentă la SCB în X şi X', atunci punctul X' este nod al lui (GV'G) şi segmentul care uneşte X şi X' este o legătură a lui (GV'G);
- Fie E şi E' două laturi circulare ale lui SCB. Dacă există un punct X în E şi un punct X' în E', astfel încât segmentul deschis care uneşte X şi X' se află integral în SC_{liber} şi dreapta suport a acestui segment este tangentă la SCB în X şi X', atunci cele două puncte X şi X' sunt noduri ale lui (GV'G) iar segmentul care le uneşte este o legătură a lui (GV'G);

- Fiecare latură dreaptă a lui SCB, care conectează două vârfuri convexe, este o legătură a lui (GV'G);
- Fiecare două noduri X şi X' ale lui (GV'G), conţinute în aceeaşi latură circulară E a lui SCB. Ele sunt conectate printr-o legătură în (GV'G), dacă nu există un alt nod X'' al lui (GV'G) care se află în E, între X şi X'.

În figura 3.1.16 este prezentat graful vizibilităţii redus generalizat pentru un spaţiu al configuraţiilor SC=$\mathbf{R}^2$, populat cu C-obstacole de tip poligoane generalizate.

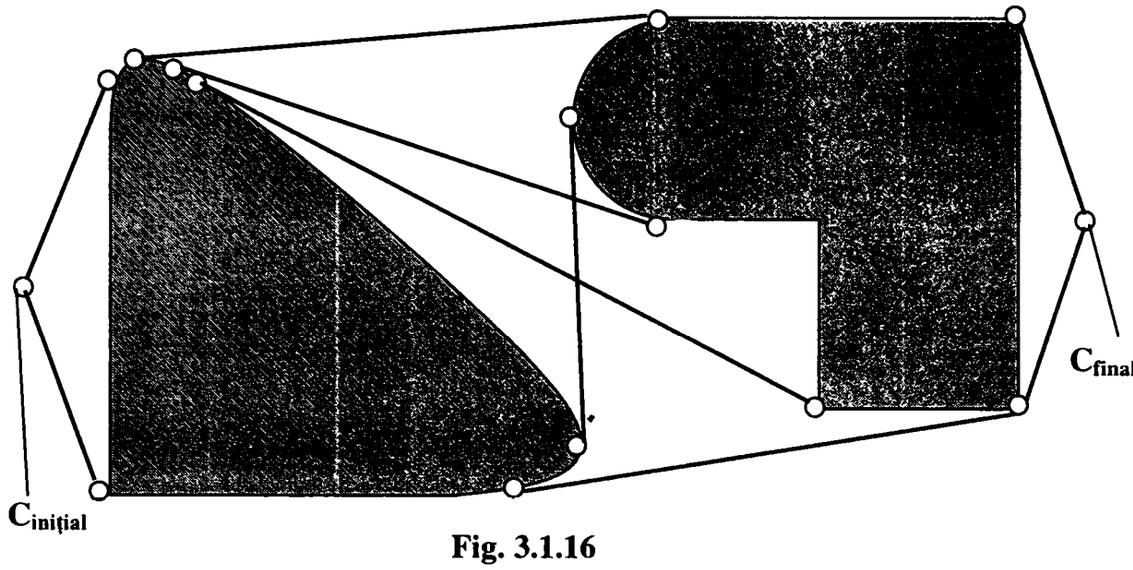


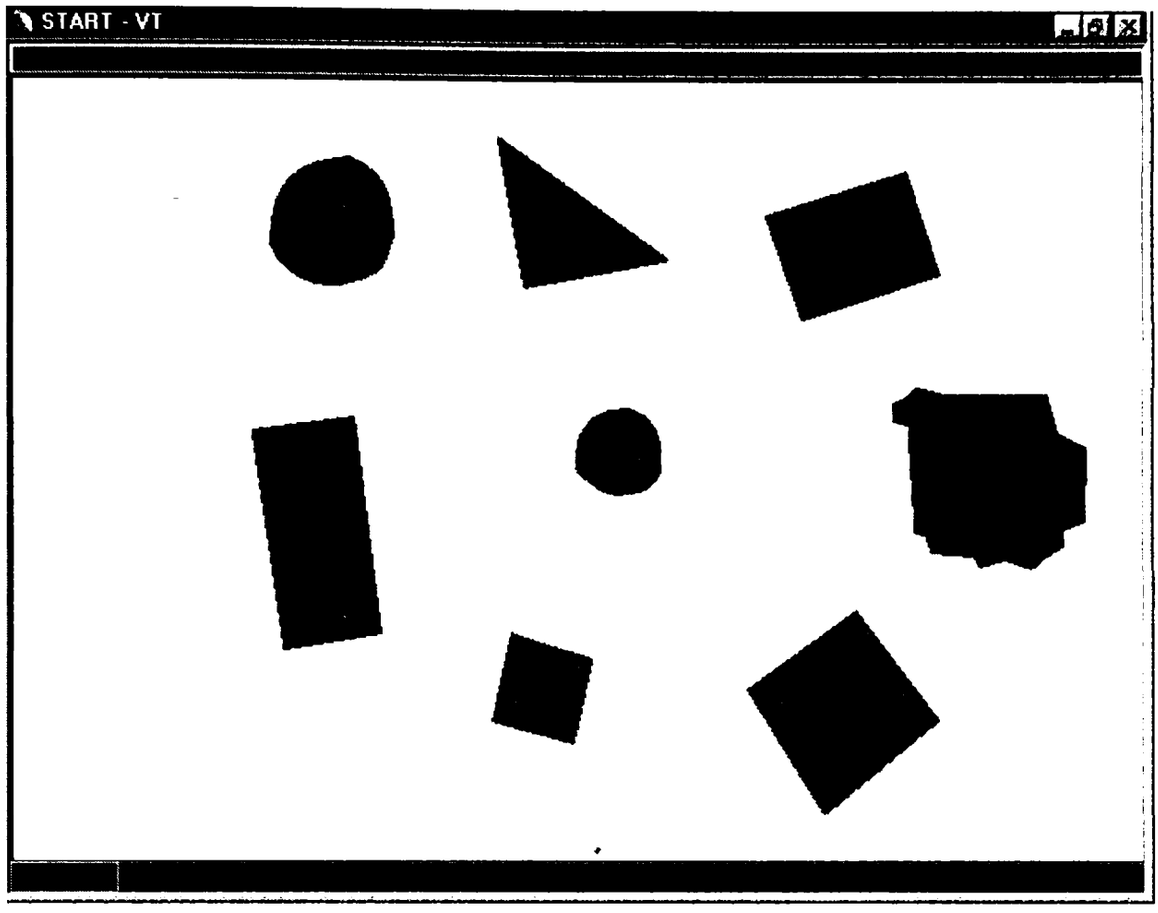
Fig. 3.1.16
Graful vizibilităţii redus generalizat

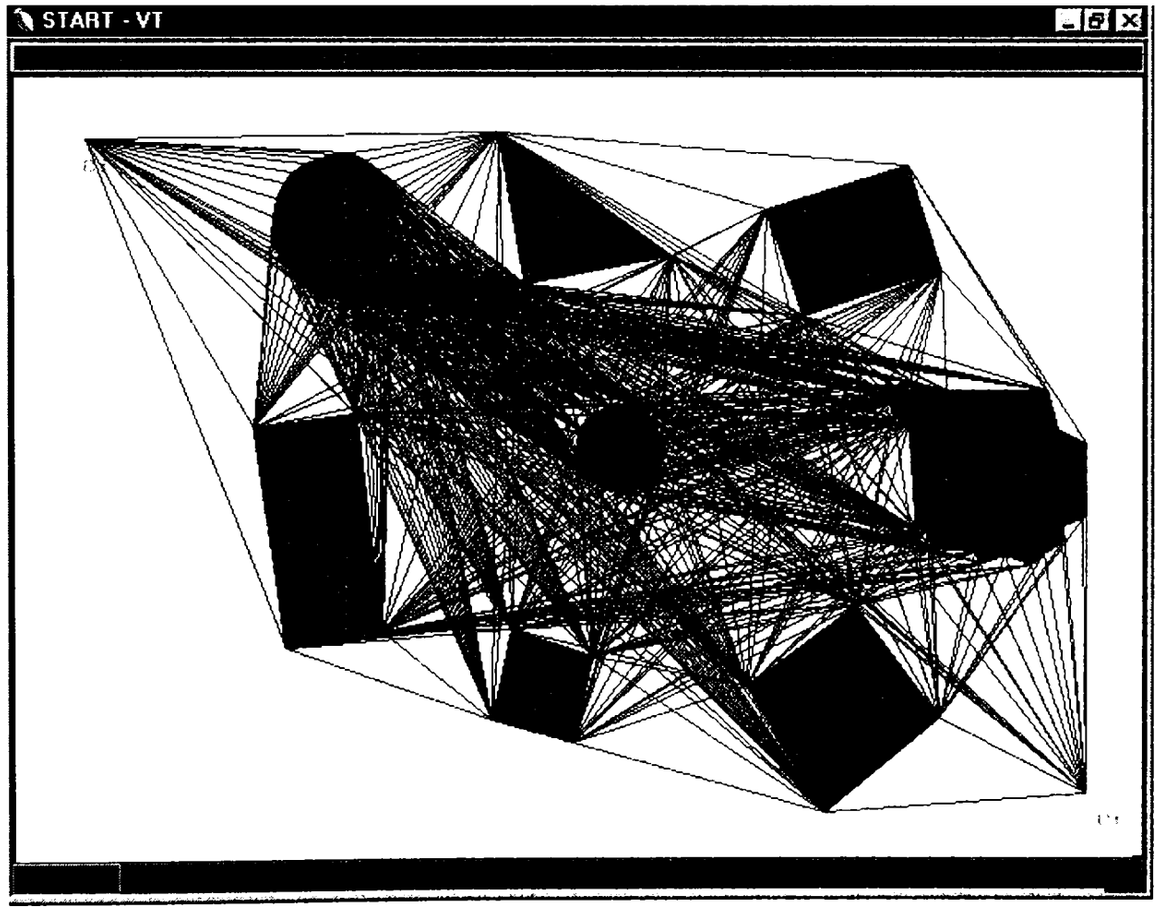
Se poate arăta că există o traiectorie semiliberă între $C_{iniţial}$ şi C_{final} dacă şi numai dacă există o traiectorie între aceste două noduri în (GV'G). În plus, dacă există o traiectorie semiliberă, atunci (GV'G) o conţine pe cea mai scurtă.

Fie n numărul total al vârfurilor lui SCB. Există cel mult două linii care sunt tangente la o latură circulară şi trec printr-un punct dat. Există cel mult patru linii care sunt simultan tangente la două laturi circulare. Pentru fiecare pereche (punct, latură) şi (latură, latură), aceste linii pot fi calculate într-un interval de timp constant. Astfel, numărul nodurilor în (GV'G) este $O(n^2)$ iar numărul legăturilor este de asemenea $O(n^2)$. (GV'G) poate fi construit uşor într-un interval de timp $O(n^3)$, iar cercetarea sa poate fi efectuată într-un interval de timp $O(n^2)$.

Metoda este de asemenea utilă când C-obstacolele poligonale sunt izotropic mărite pentru a se garanta o anumită distanţă între traiectorie şi C-obstacole.

În continuare este prezentat rezultatul aplicării acestei metode pe exemplul spaţiului de lucru în care a evoluat robotul descris [151]:





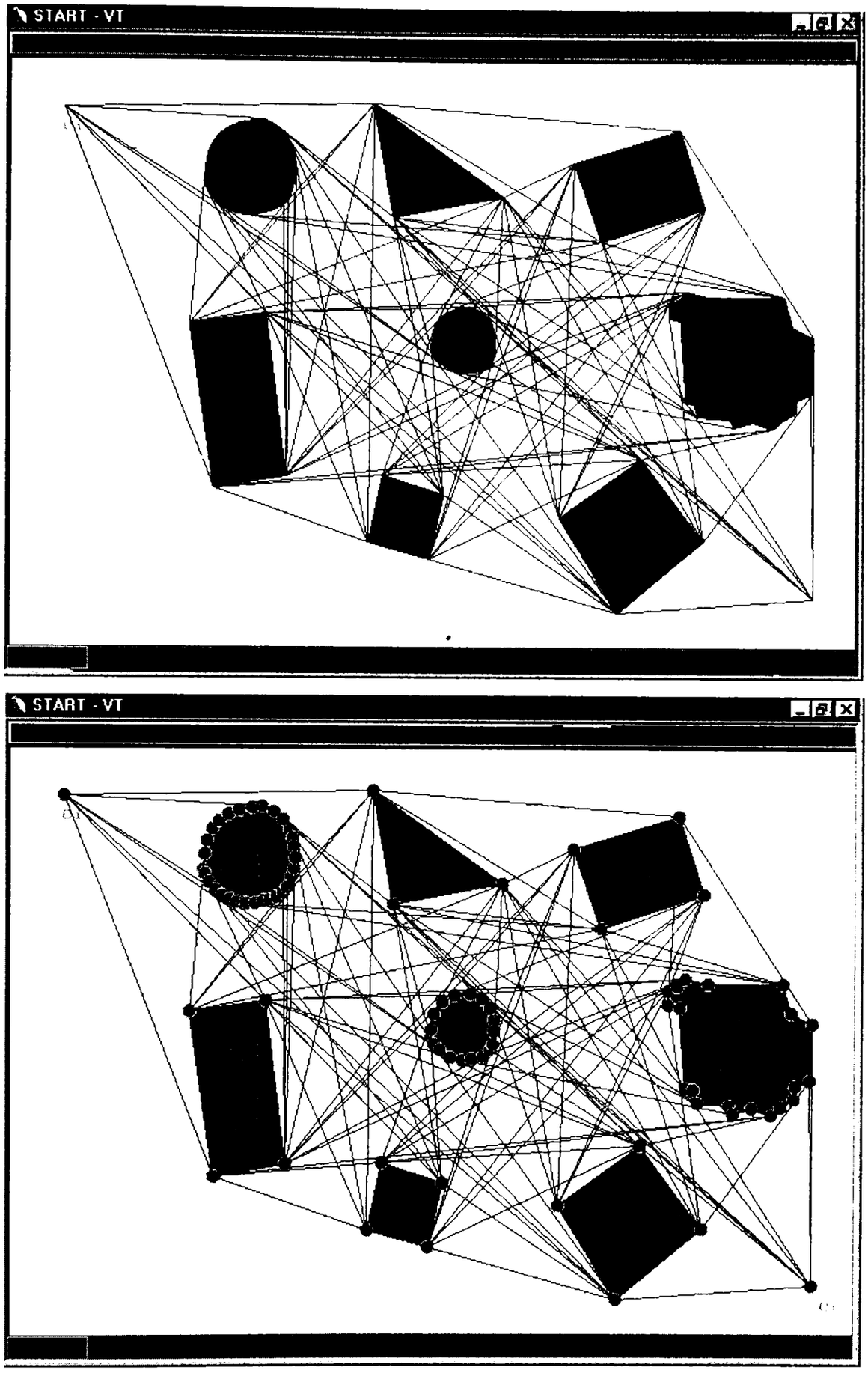


Fig. 3.1.17
Aplicarea metodei grafului vizibilității pe spațiul de lucru ales

Robotul putut fi deplasat pe orice traiectorie realizată din segmentele de dreaptă obținute conform algoritmului descris, între p configurație inițială şi una finală stabilite prealabil.

3.2 Conducerea unui robot cu achiziția imaginii în timp real

Problema conducerii unui robot pe baza achiziţiei imaginii în timp real este una foarte complexă ce necesită un hardware foarte rapid şi un software bine optimizat din punct de vedere al vitezei de calcul.

Experienţele s-au realizat cu un robot autonom (robocar) din dotarea Catedrei de Electronică Industrială a UPT [31], care poate fi comandat de calculator şi care este prevăzut cu o cameră de luat vederi ataşată roţii motoare, ce este în acelaşi timp şi roată directoare. Astfel, camera de luat vederi are direcţia de observare identică cu direcţia de deplasare a robocarului.

În continuare vor fi prezentate câteva imagini ale acestui robot :

Fig. 3.2.1
Robot efector dotat cu cameră video

Ideea principală a experimentului este realizarea unui program care să permită acestui robot, urmărirea unui obiect de o anumită formă. Pentru simplificarea problemei s-au realizat următoarele particularizări:

- Întregul spaţiu de lucru a fost acoperit cu pânze albe pentru eliminarea ţintelor false.
- Pe o bucată de carton alb, s-a lipit o figură geometrică (pentru început un cerc) de culoare neagră.

Deoarece, în spaţiul de lucru există un singur obiect de culoare neagră (cartonul fiind destul de mare, ca persoana care îl ţine să nu intre în cadrul camerei de luat vederi), şi deoarece problema recunoaşterii obiectului nepunându-se în această primă fază, ci doar problema urmăririi obiectului de către robot, cea mai simplă rezolvare a acesteia este alinierea camerei, şi implicit a roţii directoare) după axa verticală a figurii geometrice ce trece prin centrul ei de greutate.

Odată pornit robotul, acesta se află în mişcare atâta timp cât senzorii de proximitate bazaţi pe ultrasunete nu indică apropierea de un obiect. Deci, atâta timp cât obiectul se află în mişcare, iar robotul nu este foarte aproape de el, robotul va trebui să urmărească cartonul cu figura geometrică fără nici o intervenţie exterioară umană.

Robotul a fost realizat ca sistem mecanic mobil şi autonom avându-se în vedere câteva criterii simple :

- costul materialelor şi al manoperei cât mai redus
- performanţe dinamice bune
- greutate redusă
- posibilitatea modificării sau extinderii simple
- posibilitatea montării unui sistem de încărcare descărcare

Sistemul mecanic conţine următoarele elemente :

- şasiul
- mecanismul de deplasare (rulare)
- mecanismul de direcţie
- dispozitivul de protecţie mecanic (bare antişoc)
- dispozitivul de încărcare / descărcare

Soluţiile adoptate pentru sistemul mecanic sunt:

- ***şasiul*** de formă paralelipipedică este uşor de realizat şi oferă o structură rezistentă cu posibilitatea montării simple a tuturor subansamblurilor;
- ***sistemul de rulare cu trei roţi*** poate fi folosit atât la mersul înainte cât şi la mersul înapoi şi prezintă o soluţie simplă în ceea ce priveşte construcţia mecanică. Nu necesită suspensie.
- ***mecanismul de virare*** care acţionează asupra roţii din faţă este simplu, şi prin extinderea spre roţile din spate face posibilă virarea cu rază mică, atât înainte, cât şi înapoi. Nu este necesar un mecanism cu cremalieră.
- ***sistemul de tracţiune pe o roată*** este soluţia ce nu necesită o antrenare diferenţială (electrică sau mecanică). Motorul este montat pe roata de virare prin intermediul reductorului.

➢ **Şasiul**

Acesta este realizat din ţeavă de oţel cu secţiunea dreptunghiulară (20×40) sudată, constituind structura de rezistenţă ce susţine roţile, motoarele, sursele de alimentare, circuitele electronice, etc. (fig. 3.2.2).

Şasiul este acoperit cu tablă de aluminiu cu grosimea 0,8 mm, iar platforma din spate este realizată din tablă de oţel (grosimea 4 mm),

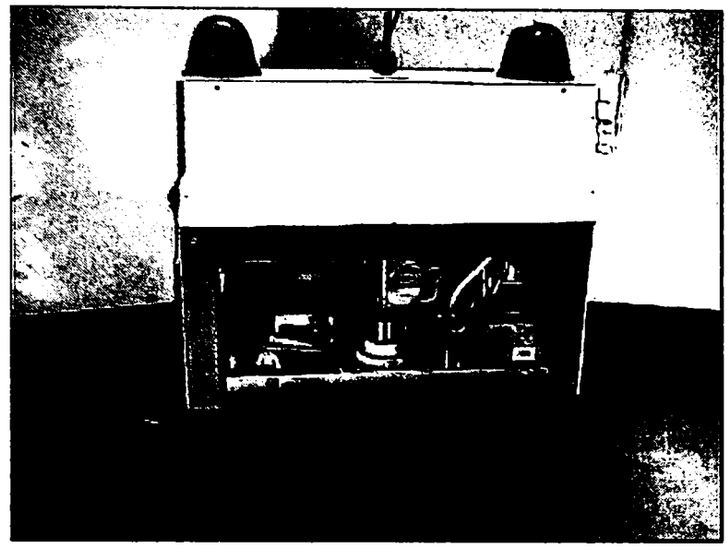


Fig. 3.2.2
Vedere asupra mecanismelor interioare ale robotului

constituind totodată suportul pentru robotul "SCORBOT". În partea de deasupra faţă a cadrului se află un compartiment ce conţine plăcile electronice şi un monitor. Deasupra compartimentului se află două lămpi (avarie şi mers) şi camerele video.

În partea din spate, sub platformă, se montează modulul de comandă al robotului "SCORBOT-ER III". Acesta este un robot articulat vertical, cu cinci grade de libertate. Sistemul SCORBOT-ER III constă în principal într-o structură mecanică şi un controler. Toate cuplele cinematice ale robotului sunt cuple de rotaţie.

Schema cinematică a robotului este prezentată în figura 3.2.3. Cuplele cinematice ale robotului sunt: cupla 1-BASE, cupla 2-SHOULDER, cupla3-ELBOW, cupla 4-PITCH, cupla5-ROLL. Modelul geometric al robotului este determinat de matricea de transformare generală $\mathbf{G}_5^0$ care exprimă poziţia şi orientarea efectorului final în raport cu sistemul de referinţă fix. Pentru calculul matricei de transformare s-a aplicat convenţia Denavit-Hartenberg.

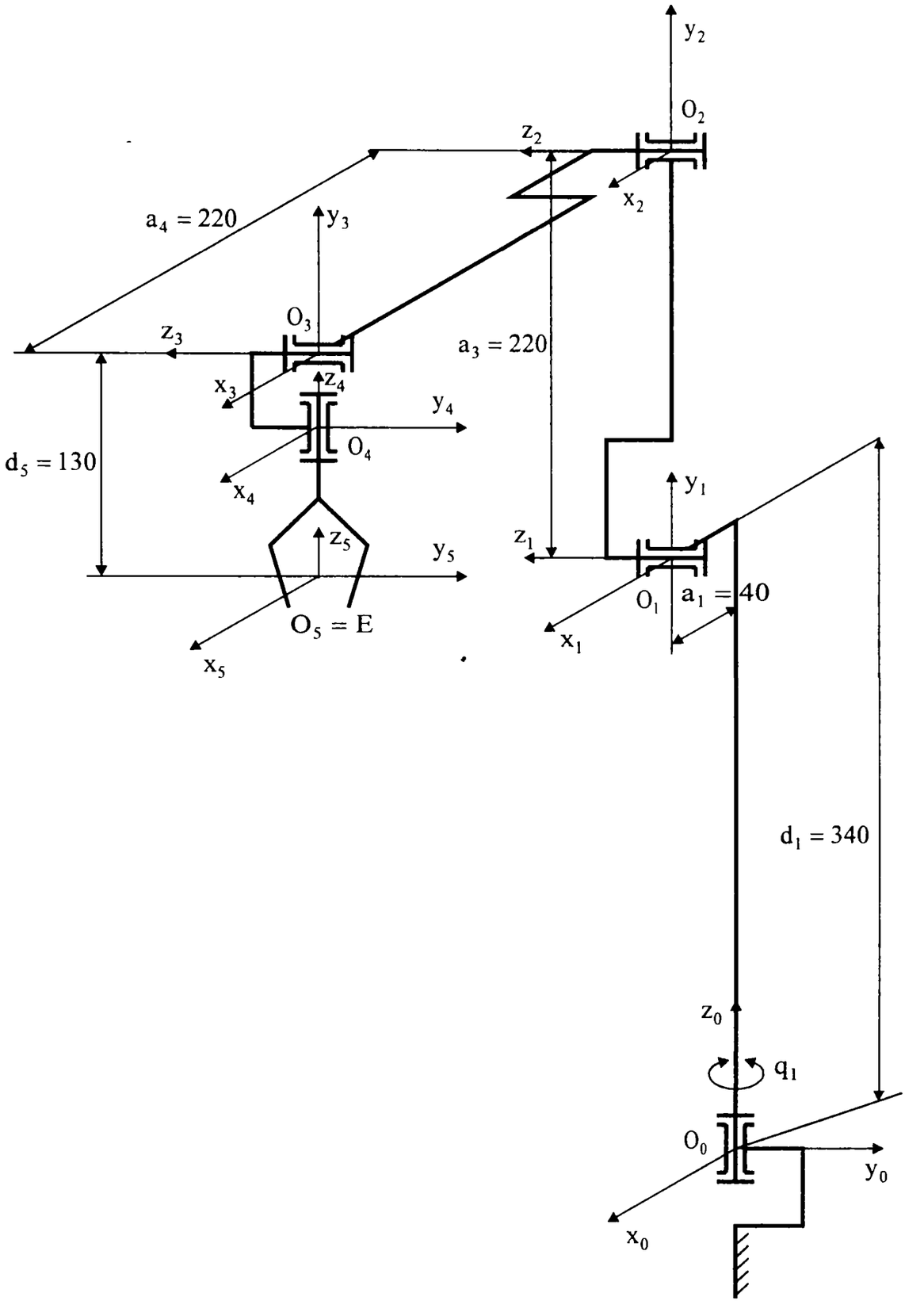


Fig. 3.2.3
Schema cinematică a robotului SCORBOT-ER III

Modelarea geometrică a robotului a condus la ecuațiile cinematice:

$$
\begin{aligned}
n_x &= \cos\theta_1 \cos\theta_5 \cos(\theta_2+\theta_3+\theta_4) - \sin\theta_1 \sin\theta_5 \\
n_y &= \sin\theta_1 \cos\theta_5 \cos(\theta_2+\theta_3+\theta_4) + \cos\theta_1 \sin\theta_5 \\
n_z &= \cos\theta_5 \sin(\theta_2+\theta_3+\theta_4) \\
o_x &= -\cos\theta_1 \sin\theta_5 \cos(\theta_2+\theta_3+\theta_4) - \sin\theta_1 \cos\theta_5 \\
o_y &= -\sin\theta_1 \sin\theta_5 \cos(\theta_2+\theta_3+\theta_4) + \cos\theta_1 \cos\theta_5 \\
o_z &= -\sin\theta_5 \sin(\theta_2+\theta_3+\theta_4) \\
a_x &= -\cos\theta_1 \sin(\theta_2+\theta_3+\theta_4) \\
a_y &= -\sin\theta_1 \sin(\theta_2+\theta_3+\theta_4) \\
a_z &= \cos(\theta_2+\theta_3+\theta_4) \\
p_x &= \cos\theta_1\left[-a_1 + a_2\cos\theta_2 + a_3\cos(\theta_2+\theta_3) - d_5\sin(\theta_2+\theta_3+\theta_4)\right] \\
p_y &= \sin\theta_1\left[-a_1 + a_2\cos\theta_2 + a_3\cos(\theta_2+\theta_3) - d_5\sin(\theta_2+\theta_3+\theta_4)\right] \\
p_z &= d_1 + a_2\sin\theta_2 + a_3\sin(\theta_2+\theta_3) + d_5\cos(\theta_2+\theta_3+\theta_4)
\end{aligned}
\qquad (3.2.1)
$$

Pentru alimentarea robocarului se utilizează două baterii de acumulatoare de 12 V / 45 Ah fiecare, legate în serie. Ele asigură o autonomie de cel puțin 8 ore dacă au fost încărcate la capacitatea maximă.

Acumulatoarele sunt plasate în centrul robocarului, la înălțime de 5 cm față de sol, pentru a coborî centrul de greutate al vehiculului. Încărcarea acumulatoarelor se face de la un redresor ce se cuplează la o mufă plasată în partea stângă a robocarului.

- **Mecanismul de tracțiune.**

Pentru robocarul construit s-a ales soluția cu trei roți (plasate ca în fig.3.2.4), din care două pasive (cele din spate) şi una activă, antrenată de către motorul de deplasare şi rotită în jurul axei verticale de către motorul direcției (roata din față).

Diametrul roților este Φ = 100 mm. Ele sunt construite din oțel şi au un înveliş exterior de cauciuc. Sunt fixate pe ax prin intermediul unui rulment .

Motorul ales pentru asigurarea deplasării este un motor de c.c. cu rotor disc având dimensiuni reduse şi randament ridicat. Tensiunea de alimentare nominală este 24 V, iar curentul consumat de 1,5 A în sarcină. S-a ales acest motor din considerente

energetice. Este preferabil un motor cu randament foarte bun şi caracteristici dinamice bune, pentru a asigura un consum redus şi o eficienţă ridicată la accelerare şi frânare.

Reductorul are gabarit redus şi un raport de transmitere de i = 1 : 34,6. El este cuplat direct cu axul roţii motoare.

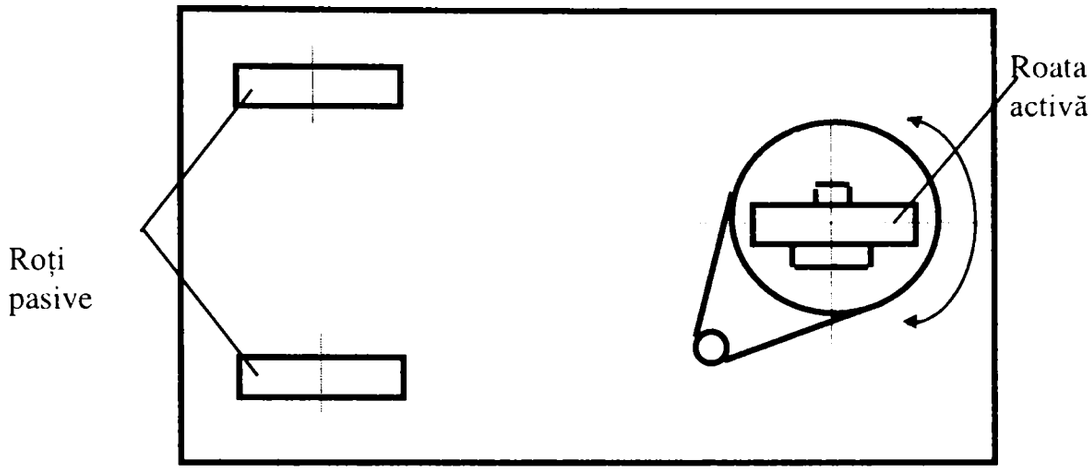


Fig. 3.2.4
Prezentarea soluţiei cu trei roţi a robocarului

- **Achiziţia imaginii în timp real şi conducerea robotului**

Funcţia care realizează întreaga operaţie de captură a imaginilor, de calcul a centrului de greutate şi de conducere a robotului se prezintă astfel :

```
void CMainFrame::OnToolsRealtime()
 {
 CROBOVIEWDoc  *SourceDoc;
 unsigned char       *ClipboardBuffer;
 short                j,x,y;
 MSG                  msg;
 long                 m00,m10;
 POINT                CentruGreutate;
 CDC                 *ActiveFrameDC;
 char                 sir[40];
 RECT                 rect = {10,480,50,500};
 short                TypeOfCapture;
 BYTE                 PixelGrayValue;
 unsigned char        OutChar;
 DWORD                WrittenBytes;
 HANDLE               hCom;
 BOOL                 fSuccess;
 DCB                  dcb;
 short                count =0;
 hCom = CreateFile("COM2",
                      GENERIC_READ | GENERIC_WRITE,
                      0,    // exclusive access
                      NULL, // no security attrs
                      OPEN_EXISTING,
                      0,    // not overlapped I/O
                      NULL
                      );
```

```
 if (hCom == INVALID_HANDLE_VALUE)
  {
  ShowErrorMessage("COM2 device cannot be opened\n");
  return;
  }
 // Omit the call to SetupComm to use the default queue sizes.
 // Get the current configuration.
 fSuccess = GetCommState(hCom, &dcb);
 if (!fSuccess)
  {
  ShowErrorMessage("Cannot get COM2 current configuration\n");
  return;
  }
 // Fill in the DCB: baud=9600, 8 data bits, no parity, 1 stop bit.
 dcb.BaudRate = 9600;
 dcb.ByteSize = 8;
 dcb.Parity = NOPARITY;
 dcb.StopBits = TWOSTOPBITS;
 fSuccess = SetCommState(hCom, &dcb);
 if (!fSuccess)
  {
  ShowErrorMessage("Cannot set COM2 new configuration\n");
  return;
  }
 SourceDoc = GetActiveSourceDocument();
 if(SourceDoc->BitmapBuffer == NULL)
   SourceDoc->BitmapBuffer= (BYTE *)malloc(DIBSIZE);
//--> Capturarea unei singure imagini si determinare carcateristici
 MyEditCopy();
 if(!MyOpenClipboard(NULL))
  {
  MessageBox("Error opening clipboard","Roboview",MB_OK);
  CloseHandle(hCom);
  return;
  }
 if(!(ClipboardBuffer = (unsigned char *)MyGetClipboardData(CF_DIB)))
  {
  MessageBox("Error reading clipboard data","Roboview",MB_OK);
  CloseHandle(hCom);
  return;
  }
 memcpy(SourceDoc->BitmapBuffer,ClipboardBuffer,DIBSIZE);
 CloseClipboard();
 if(SourceDoc->BitmapBuffer != NULL)
  {
  SourceDoc->BitmapBits = GetDibBitsAddr(SourceDoc->BitmapBuffer);
  Xmax = GlobalBitmapInformation.XDimension = GetDibWidth(SourceDoc->BitmapBuffer);
  Ymax = GlobalBitmapInformation.YDimension = GetDibHeight(SourceDoc->BitmapBuffer);
  }
 for(y=Ymax-1;y>=0;y--)
  for(x=0;x<Xmax;x++)
   {
   SourceDoc->BitmapMatrix[x][y][0] = *(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x);
   SourceDoc->BitmapMatrix[x][y][1] = *(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x+1);
   SourceDoc->BitmapMatrix[x][y][2] = *(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x+2);
   }
 for(x=0;x<Xmax;x++)
  for(y=REALYMAX;y<Ymax;y++)
   SourceDoc->BitmapMatrix[x][y][0] = SourceDoc->BitmapMatrix[x][y][1] =
                                      SourceDoc->BitmapMatrix[x][y][2] = 0xFF;
```

```
  TypeOfCapture = GRAYSCALE;
  for(x=0;x<Xmax;x++)
   for(y=0;y<REALYMAX;y++)
    if((SourceDoc->BitmapMatrix[x][y][0]!= SourceDoc->BitmapMatrix[x][y][1]) ||
       (SourceDoc->BitmapMatrix[x][y][0]!= SourceDoc->BitmapMatrix[x][y][2]) ||
       (SourceDoc->BitmapMatrix[x][y][1] != SourceDoc->BitmapMatrix[x][y][2]))
         TypeOfCapture = TRUECOLOR;
//<-- Capturarea unei singure imagini si determinare carcateristici
  TypeOfCapture = SourceDoc->BitmapColorType;
  OutChar = '8';
  fSuccess = WriteFile(hCom,&OutChar,1,&WrittenBytes,NULL);
  if(!fSuccess || WrittenBytes != 1)
   {
   ShowErrorMessage("Cannot write to COM2\n");
   CloseHandle(hCom);
   return;
   }
//Bucla de captura si prelucrare
  for(;;)
   {
   m00 = m10 = 0;
   MyEditCopy();
   if(!MyOpenClipboard(NULL))
    {
    MessageBox("Error opening clipboard","Roboview",MB_OK);
    CloseHandle(hCom);
    return;
    }
   if(!(ClipboardBuffer = (unsigned char *)MyGetClipboardData(CF_DIB)))
    {
    ShowErrorMessage("Error reading clipboard data\n");
    MyDriverDisconnect();
    MyDestroyWindow();
    VideoCapture();
    continue;
    }
   memcpy(SourceDoc->BitmapBuffer,ClipboardBuffer,DIBSIZE);
   if(SourceDoc->BitmapBuffer != NULL)
    SourceDoc->BitmapBits = GetDibBitsAddr(SourceDoc->BitmapBuffer);
   for(y=Ymax-1;y>=0;y--)
    for(x=0;x<Xmax;x++)
     {
     if(TypeOfCapture = TRUECOLOR)
      PixelGrayValue = (BYTE)(0.3*(*(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x)) +
                              0.59*(*(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x+1)) +
                              0.11*(*(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x+2)));
     else
       PixelGrayValue = *(SourceDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x);
     if(PixelGrayValue > 70)
      SourceDoc->BitmapMatrix[x][y][0] = SourceDoc->BitmapMatrix[x][y][1] =
                                         SourceDoc->BitmapMatrix[x][y][2] = 0xFF;
     else
      {
      SourceDoc->BitmapMatrix[x][y][0] = SourceDoc->BitmapMatrix[x][y][1] =
                                         SourceDoc->BitmapMatrix[x][y][2] = 0;
      m00++;
      m10 += x;
      }
     }
   if(m00 == 0)
```

```
  CentruGreutate.x = 160;
else
   CentruGreutate.x = m10/m00;
for(x=0;x<Xmax;x++)
  for(y=REALYMAX;y<Ymax;y++)
    SourceDoc->BitmapMatrix[x][y][0] = SourceDoc->BitmapMatrix[x][y][1] =
                                        SourceDoc->BitmapMatrix[x][y][2] = 0xFF;
SourceDoc->BitmapColorType = BINARY;
EmptyClipboard();
CloseClipboard();
SourceDoc->UpdateAllViews(NULL);
ActiveFrameDC = AfxGetMainWnd()->GetDC();
itoa(160-CentruGreutate.x,sir,10);
ActiveFrameDC->SetTextColor(RGB(255,0,0));
ActiveFrameDC->SetBkMode(OPAQUE);
ActiveFrameDC->ExtTextOut(10,480,ETO_OPAQUE,&rect,sir,strlen(sir),NULL);
if(abs(160 - CentruGreutate.x) > 10)
  {
  if((160-CentruGreutate.x) > 0 /*&& count >= 10*/)
    {
    OutChar = '4';
    count = 0;
    }
  if((160-CentruGreutate.x) < 0 /*&& count >= 10*/)
    {
    OutChar = '6';
    count = 0;
    }
  fSuccess = WriteFile(hCom,&OutChar,1,&WrittenBytes,NULL);
  if(!fSuccess || WrittenBytes != 1)
    {
    ShowErrorMessage("Cannot write to COM2\n");
    CloseHandle(hCom);
     return;
     }
  }
for(j=0;j<100;j++)
  {
  if(PeekMessage(&msg,NULL,0,0,PM_REMOVE))
    {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    if(msg.message == WM_CHAR)
      {
      if(msg.wParam != ' ')
        {
        CloseHandle(hCom);
        return;
        }
      else
        {
        for(;;)
          {
          if(PeekMessage(&msg,NULL,0,0,PM_REMOVE))
            {
            if(msg.message == WM_CHAR)
              {
              if(msg.wParam == ' ')
                {
                CloseHandle(hCom);
```

```
            return;
             }
            OutChar = msg.wParam;
            fSuccess = WriteFile(hCom,&OutChar,1,&WrittenBytes,NULL);
            if(!fSuccess || WrittenBytes != 1)
             {
             ShowErrorMessage("Cannot write to COM2\n");
             CloseHandle(hCom);
             return;
             }
            }
           TranslateMessage(&msg);
           DispatchMessage(&msg);
           }
          }
         }
        }
       }
      else
       break;
     }
   }
 CloseHandle(hCom);
 return;
}
```

Deoarece interfața robotului cu calculatorul comunică printr-un port serial funcția începe prin deschiderea portului COM2 în acces exclusiv. Dacă deschiderea a reuşit, urmează procedurile de inițializare a portului cu valorile necesare comunicației cu hardware-ul de control.

Urmează determinarea tipului de cameră video existent, color sau alb negru. Acest lucru se realizează prin capturarea unei imagini în clipboard şi prin compararea valorilor RGB ale tuturor pixelilor se determină dacă imaginea este color sau în nivele de gri. Dacă imaginea este o imagine color ea va fi convertită la o imagine cu 256 nivele de gri.

Următoarea fază a procedurii va trimite caracterul '8' către interfață, caracter ce înseamnă pornirea motorului şi implicit a robotului. În acest moment în fața robotului trebuie să existe cartonul cu desenul figurii geometrice.

Odată pornit robotul, începe secvența de achiziționare de imagini. Pentru fiecare imagine achiziționată se realizează o binarizare a acesteia cu un prag de 70 nivel gri, şi se determină centrul de greutate al pixelilor rezultați. Odată determinat centrul de greutate, se eliberează clipboard-ul şi se pregăteşte pentru o nouă achiziție. În acelaşi timp se afişează pe ecran poziția determinată a centrului de greutate şi se comandă direcția robotului în funcție de ofsetul centrului de greutate față de linia verticală a ecranului.

Pentru a nu lua în considerare variații foarte mici ale ofsetului și evita mersul șerpuit al robotului, se ignoră ofseturile mai mici de 10 pixeli.

Deci, teoretic roata motoare își va schimba direcția atâta timp cât nu este perpendiculară pe centrul de greutate al figurii geometrice.

Ultima porțiune a funcției asigură transmiterea mesajelor Windows către librăria Video for Windows și o pregătesc pe aceasta pentru o nouă achiziție.

În concluzie robotul va urmări figura geometrică atâta timp cât senzorii de proximitate nu îl vor opri.

- **Recunoaşterea obiectului urmărit.**

Din cauză că algoritmul de recunoaştere este unul mare consumator de tip, pentru acest experiment a fost necesară introducerea unor restricții suplimentare. În primul rând, cea mai mare pierdere de timp se realizează în faza de rotire a corpului de-a lungul laturii mari pentru procesul de recunoaştere. În vederea eliminării acestei operații, figura geometrică va fi desenată în poziție normală cu latura mare orizontală pe bucata de carton. O altă creştere de viteză, se mai poate obține prin apelarea procedurii de recunoaştere din 10 în 10 imagini, în rest orientarea realizându-se după centrul de greutate.

Pe lângă aceste restricții, o ultimă mare problemă o reprezintă reflexia luminii pe figura geometric, reflexie care îi alterează contururile, mai ale deoarece cartonul cu figura geometrică este în continuă mişcare şi nu se poate anticipa o achiziție corectă care să permită o recunoaştere 100% exactă.

- **Planificarea mişcării robocarului**

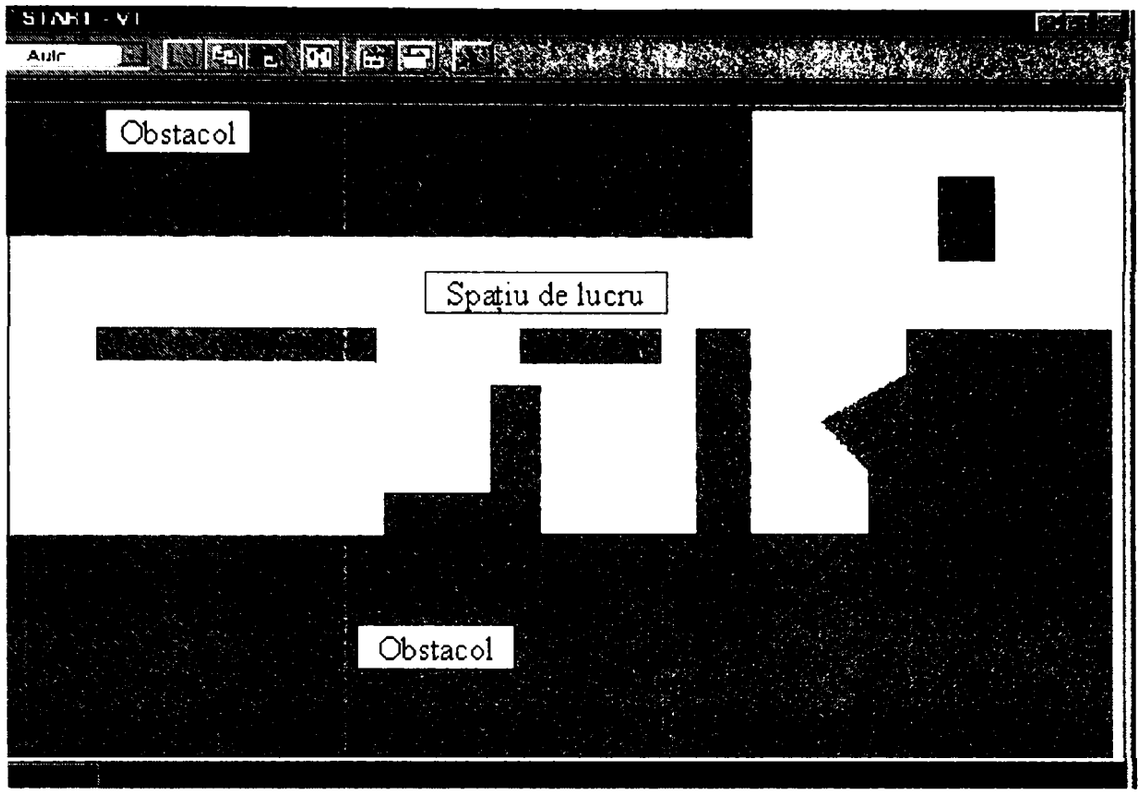
S-au realizat planificări ale mişcării robocarului în spațiul de lucru oferit de Laboratorul de Robotică al Catedrei de Organe de Maşini şi Mecanism al UPT. Pentru că robotul se deplasează pe pardoseaua plană laboratorului, s-a realizat o secțiune plană a spațiului populat cu obstacole şi s-a reprezentat grafic.

Traiectoria planificată pe baza observării spațiului de lucru s-a realizat în mai multe etape succesive prezentate în figura 3.2.5.

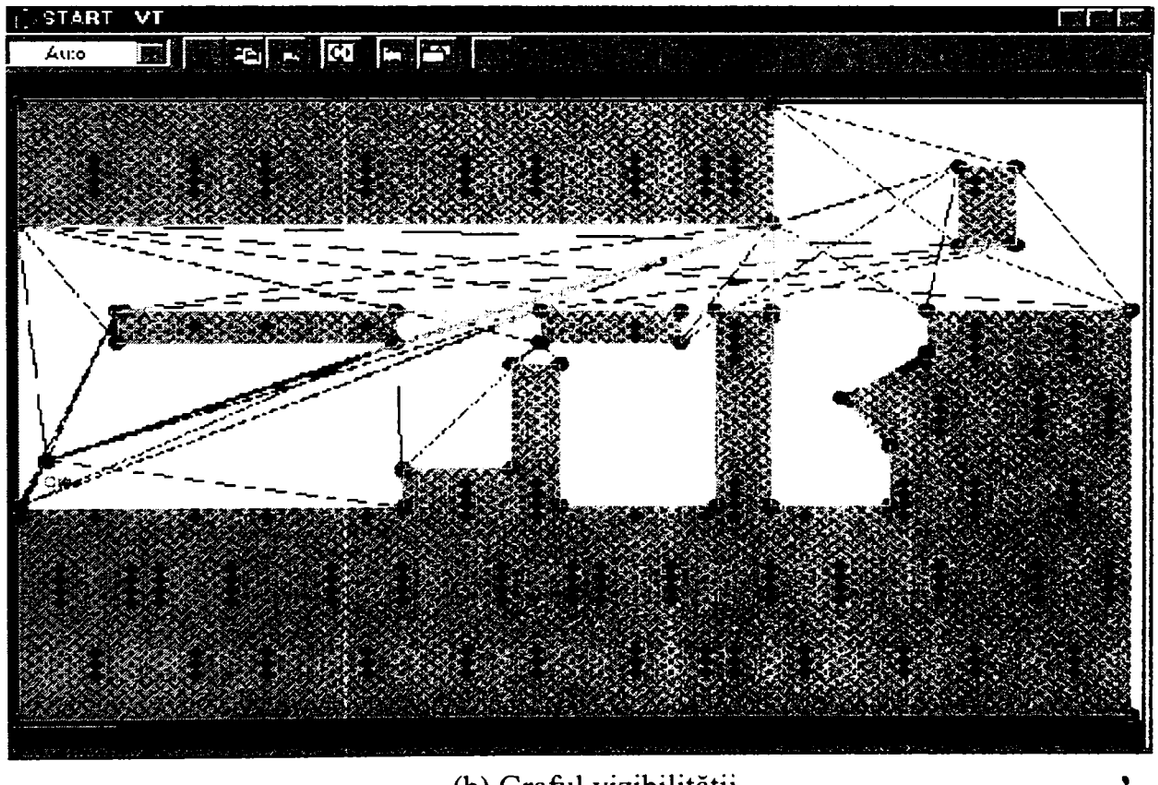
Planificarea traiectoriei robocarului s-a realizat prin metodele grafului vizibilității şi grilei neomogene.

Se observă că pe vehicul s-a montat manipulatorul SCORBOT, acesta îndeplinind rolul de dispozitiv de încărcare-descărcare. Cu ajutorul lui au fost

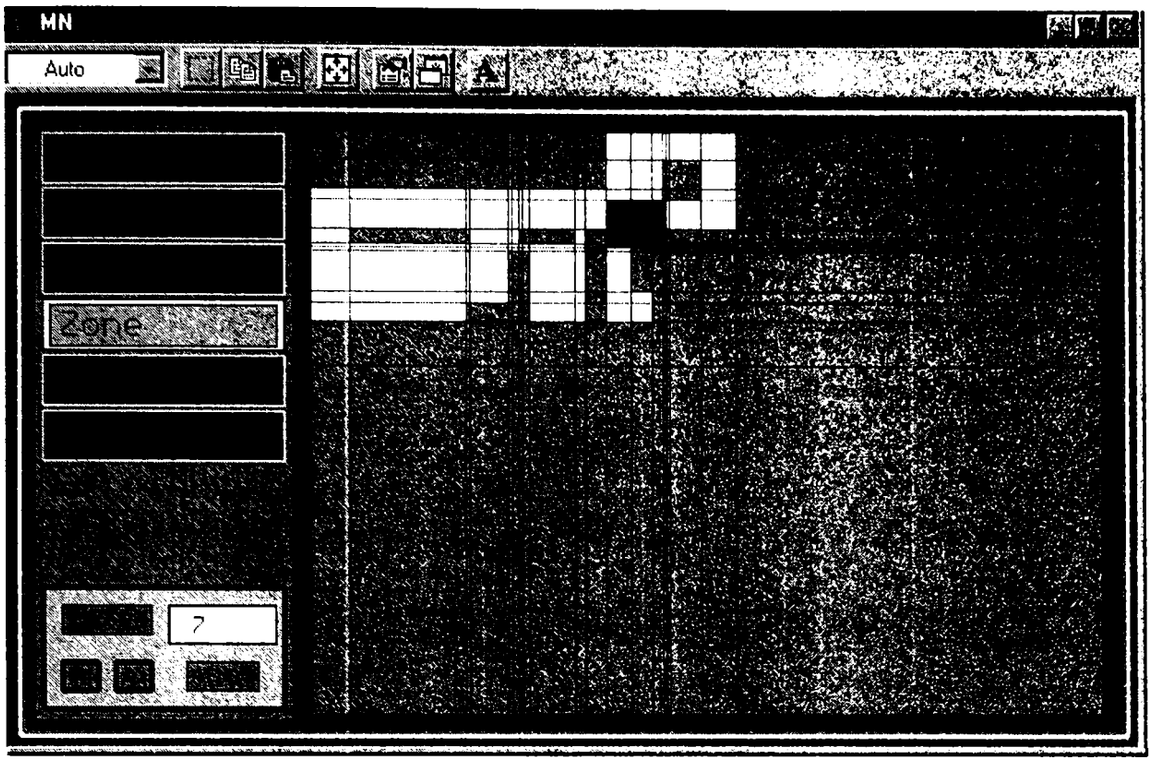
executate operații simple, cum ar fi: preluarea unor obiecte și transportul acestora la o destinație dinainte stabilită, deschiderea unor uși pentru a face posibilă trecerea robocarului, etc.



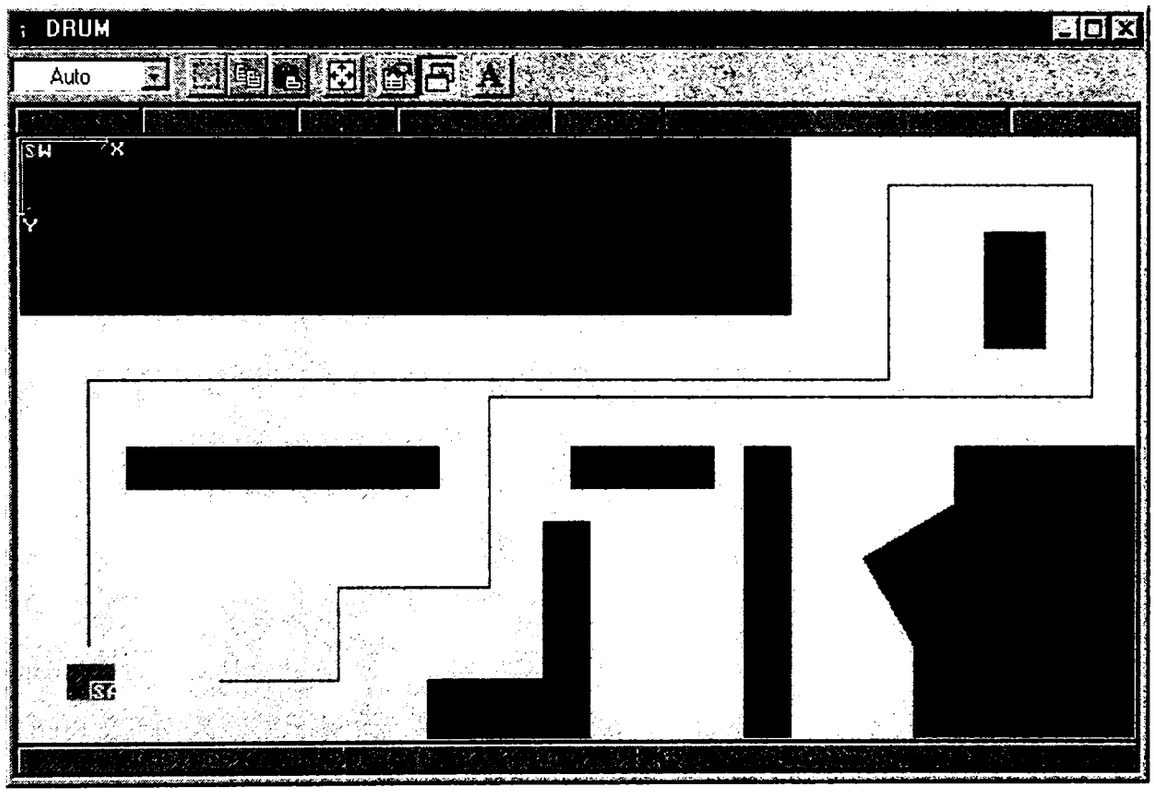
(a) Secțiune plană prin hala laboratorului



(b) Graful vizibilității



(c) O fază din desfăşurarea metodei grilei neomogene



(d) Traiectoria finală planificată

Fig. 3.2.5
Etapele succesive de determinare a traiectoriei

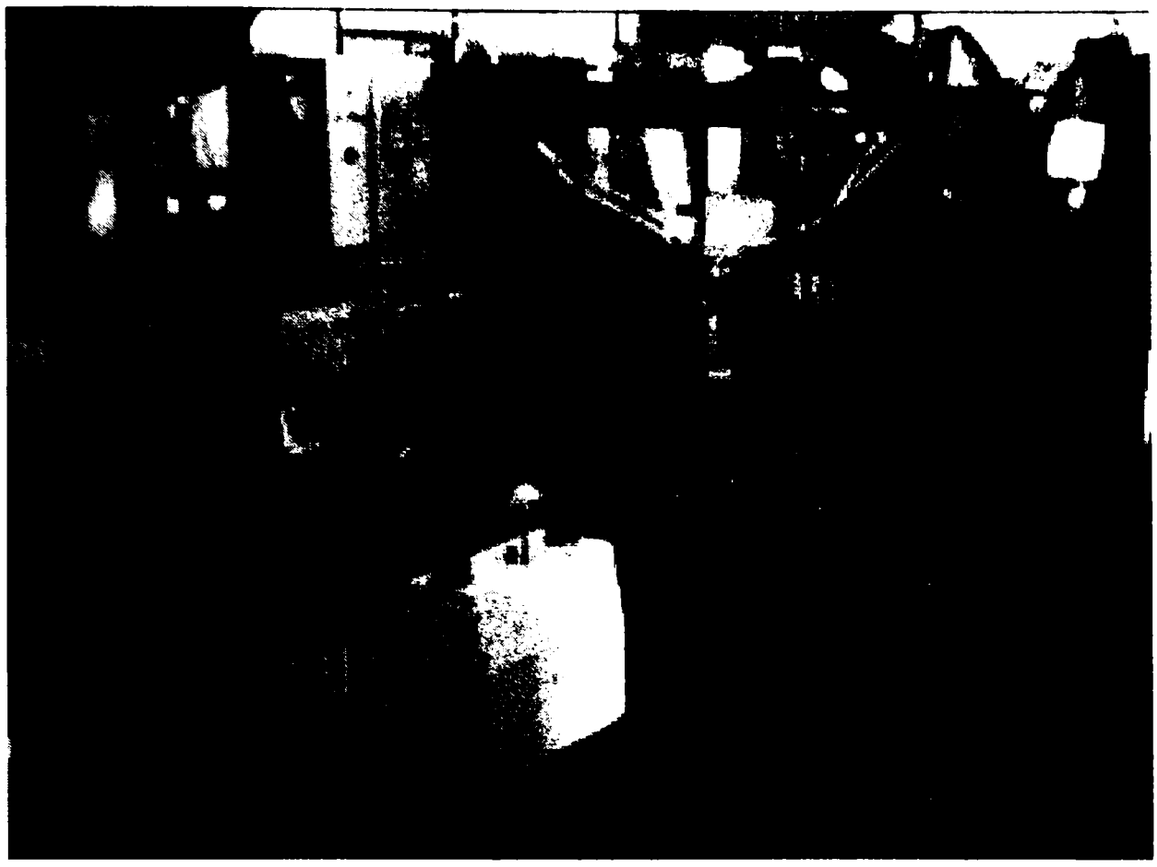


Fig. 3.2.6
Imaginea robocarului şi a manipulatorului
SCORBOT în spaţiul de lucru

3.3. Prezentarea programului principal de achiziție și prelucrare a imaginilor

Având în vedere faptul că algoritmii utilizați necesită lucrul cu diferite obiecte ce diferă ca formă și dimensiuni, pentru o modularizare a proiectului s-a lucrat în Visual C++ 5.0 sub librăria Microsoft Foundation Class (MFC)], limbaj ce oferă cele mai bune metode de programare obiectuală [133], [124]. În vederea maximizării vitezei de prelucrare a imaginilor, scopul final fiind o prelucrare în timp real, se utilizează copii ale imaginilor în memorie în locul interacțiunii directe cu ecranul, care este o procedură mult mai lentă și care nu s-ar preta scopului propus.

Obiectul de bază a întregului program este un document MFC care stochează imaginea recepționată în formatul Windows/OS2 Bitmap cu o adâncime de culoare de 24 de biți, adâncime care permite foarte ușor conversia imaginii la 256 de nivele de gri. Structura acestei clase conține:

- informațiile referitoare la pixeli
- lista corpurilor conținute în cadrul unei imagini
- adâncimea de culoare a imaginii

Structura clasei ce conține o imagine este următoarea:

```
class CROBOVIEWDoc : public CDocument
{
protected:
        CROBOVIEWDoc();
        DECLARE_DYNCREATE(CROBOVIEWDoc)
public:
        BYTE *BitmapBits;
        BYTE *BitmapBuffer;
        BYTE  BitmapMatrix[320][240][3];
        BYTE  BitmapColorType;
        CTypedPtrList<CObList, CCorp*> m_CorpList;

        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        CCorp* NewCorp();
        void DeleteCorps();
        BOOL CROBOVIEWDoc::PixelBelongsToACorp(short x,short y);
        void CROBOVIEWDoc::GetShapeNumbers(CROBOVIEWDoc        *OriginalDoc);
        void CROBOVIEWDoc::RecognizeObjectsByColor(CROBOVIEWDoc*OriginalDoc);
        virtual ~CROBOVIEWDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
        DECLARE_MESSAGE_MAP()
};
```

Pentru afișarea rapidă ca bitmap pe ecran se utilizează variabilele membre BitmapBuffer și BitmapBits, iar pentru parcurgerea imaginii pixel cu pixel se

foloseşte buferul BitmapMatrix care poate stoca o imagine având rezoluţia 320x240 şi cu o adâncime de culoare de 24 de biţi. Variabila membră BitmapColorType permite o identificare uşoară în timpul prelucrării mai multor imaginii a tipului de culoare existent: imagine colorată, nivele de gri sau binară. Variabila membră m_CorpList conţine lista tuturor obiectelor din cadrul imaginii. Funcţiile membre ajută la obţinerea de informaţii referitoare la imagine sau la obiectele care o constituie.

Afişarea rapidă a unui document pe ecran se face funcţia membră OnDraw, astfel:

```
void CROBOVIEWView::OnDraw(CDC* pDC)
{
 HDC hdc;
 short x,y;
 CROBOVIEWDoc* pDoc = GetDocument();
 ASSERT_VALID(pDoc);

  if(pDoc->BitmapBuffer)
  {
  for(y=Ymax-1;y>=0;y--)
   for(x=0;x<Xmax;x++)
     {
     *(pDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x)   = pDoc->BitmapMatrix[x][y][0];
     *(pDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x+1) = pDoc->BitmapMatrix[x][y][1];
     *(pDoc->BitmapBits+3*Xmax*(Ymax-1-y)+3*x+2) = pDoc->BitmapMatrix[x][y][2];
     }
  pDC->SetStretchBltMode(COLORONCOLOR);
  hdc = pDC->GetSafeHdc();
  if(GDI_ERROR==StretchDIBits(hdc, 0, 0, GlobalBitmapInformation.XDimension,
                  GlobalBitmapInformation.YDimension, 0, 0, GlobalBitmapInformation.XDimension,
                  GlobalBitmapInformation.YDimension, (LPSTR)pDoc->BitmapBits,
                  (LPBITMAPINFO)pDoc->BitmapBuffer,DIB_RGB_COLORS,SRCCOPY))
   {
   pDC->TextOut(0,0,"Error opening bitmap",15);
   return;
   }
 }
}
```

Următoarea structură importantă este cea unui obiect din cadrul unei imagini şi ea are următorul conţinut:

```
class CCorp : public CObject
{
 public:
  CCorp();
  CArray<CPoint,CPoint>  m_pointArray;
  CString m_ChainCode;
  CString m_FirstDifference;
  CString m_ShapeNumber;
  CString m_Geometric;
};
```

În această structură un obiect este definit prin punctele care îi descriu conturul în variabila m_PointArray, de codurile ce servesc identificării : codul de înlănţuire,

prima diferență și numărul de formă, cât și de rezultatul recunoașterii acelui obiect care îi descrie forma geometrică în variabila membră m_Geometric.

Programul rulează la o rezoluție de 800 x 600 x 24 sau mai mare, deoarece afișează simultan pe ecran 4 imagini de 320 x 240 pixeli plus fereastra de vizionare a capturii curente.

Funcțiile principale ale programului au fost prezentate pe parcursul tezei, împreună cu exemplele de rularea adecvate. Pe lângă aceste funcții programul mai permite citirea și salvarea de imagini în format Bitmap.

Funcția care realizează citirea și salvarea documentelor este următoarea:

```
void CROBOVIEWDoc::Serialize(CArchive& ar)
{
 short x,y;
 if (ar.IsStoring())
  {
  ar.Write(&GlobalBitmapInformation.FileHeader,sizeof(BITMAPFILEHEADER));
  ar.Write(&GlobalBitmapInformation.InfoHeader,sizeof(BITMAPINFOHEADER));
  ar.Write(BitmapBits,3*320*240);
  }
 else
  {
  BitmapBuffer = ReadDib(&ar);
  if(BitmapBuffer != NULL)
   {
   BitmapBits     = GetDibBitsAddr(BitmapBuffer);
   Xmax = GlobalBitmapInformation.XDimension = GetDibWidth(BitmapBuffer);
   Ymax = GlobalBitmapInformation.YDimension = GetDibHeight(BitmapBuffer);
   }
  for(y=Ymax-1;y>=0;y--)
   for(x=0;x<Xmax;x++)
    {
    BitmapMatrix[x][y][0] = *(BitmapBits+3*Xmax*(Ymax-1-y)+3*x);
    BitmapMatrix[x][y][1] = *(BitmapBits+3*Xmax*(Ymax-1-y)+3*x+1);
    BitmapMatrix[x][y][2] = *(BitmapBits+3*Xmax*(Ymax-1-y)+3*x+2);
    }
  for(x=0;x<Xmax;x++)
   for(y=REALYMAX;y<Ymax;y++)
    BitmapMatrix[x][y][0] = BitmapMatrix[x][y][1] = BitmapMatrix[x][y][2] = 0xFF;
  for(x=0;x<Xmax;x++)
   for(y=0;y<REALYMAX;y++)
    if((BitmapMatrix[x][y][0] != BitmapMatrix[x][y][1]) || (BitmapMatrix[x][y][0] !=
       BitmapMatrix[x][y][2]) || (BitmapMatrix[x][y][1] != BitmapMatrix[x][y][2]))
     {
     BitmapColorType = TRUECOLOR;
     return;
     }
  for(x=0;x<Xmax;x++)
   for(y=0;y<REALYMAX;y++)
    if((BitmapMatrix[x][y][0] != 0) && (BitmapMatrix[x][y][0] != 255))
     {
     BitmapColorType = GRAYSCALE;
     return;
     }
  BitmapColorType = BINARY;
```

```
  }
}
```

Funcția de citire a unui fișier bitmap :

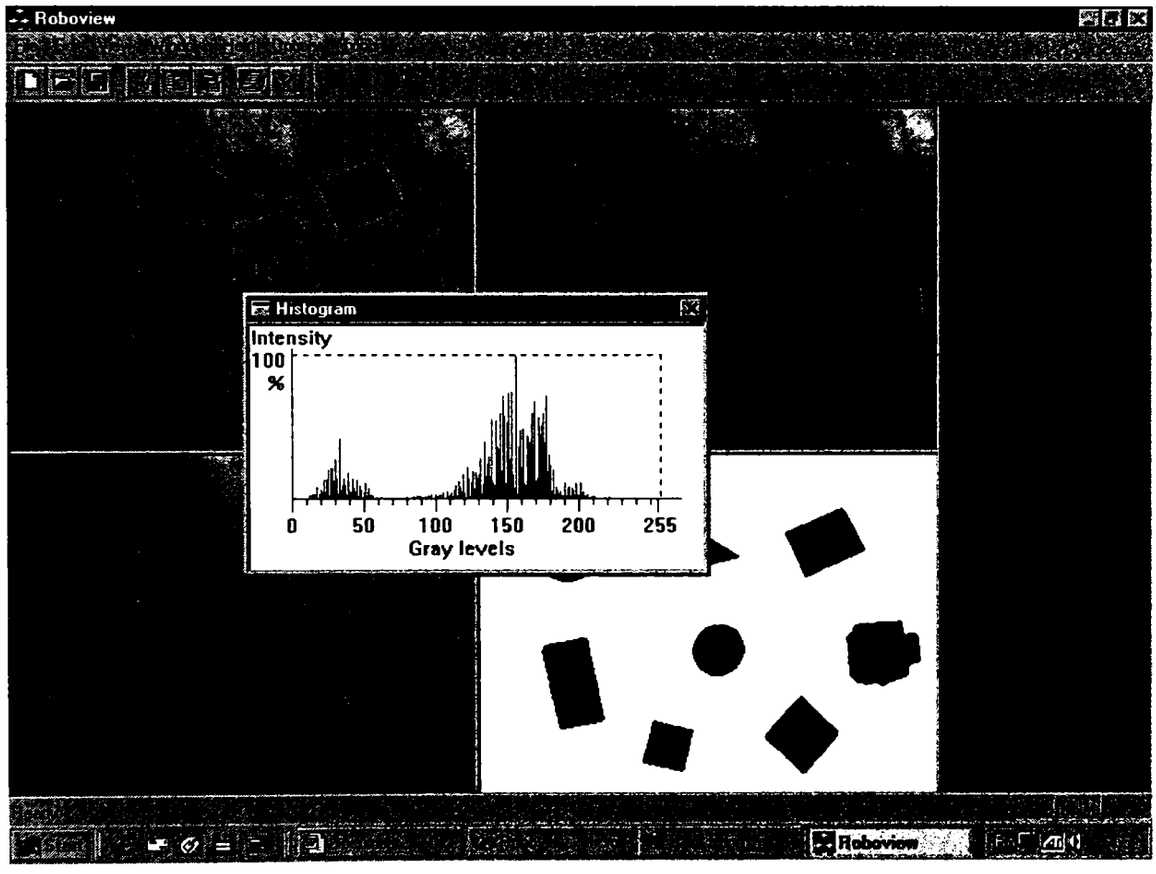
```
BYTE *ReadDib(CArchive *ar)
{
 BITMAPFILEHEADER bmfh;
 DWORD            DibSize,HeaderSize;
 BYTE             *DibBuffer;

 if(ar->Read(&bmfh,sizeof(BITMAPFILEHEADER)) != sizeof(BITMAPFILEHEADER))
  return NULL;
 memcpy(&GlobalBitmapInformation.FileHeader,&bmfh,sizeof(BITMAPFILEHEADER));
 if(bmfh.bfType != *(WORD *)"BM")
  return NULL;
 DibSize = bmfh.bfSize - sizeof(BITMAPFILEHEADER);
 DibBuffer = (BYTE *)malloc(DibSize);
 if(DibBuffer == NULL)
  return NULL;
 if(ar->Read(DibBuffer,DibSize) != DibSize)
  {
  free(DibBuffer);
  return NULL;
  }
 HeaderSize = GetDibInfoHeaderSize(DibBuffer);
 if(HeaderSize < 12 || (HeaderSize > 12 && HeaderSize <16))
  {
  free(DibBuffer);
  return NULL;
  }
 memcpy(&GlobalBitmapInformation.InfoHeader,DibBuffer,HeaderSize);
 return DibBuffer;
}
```

Diverse funcții auxiliare pentru citirea unui fișier bitmap :

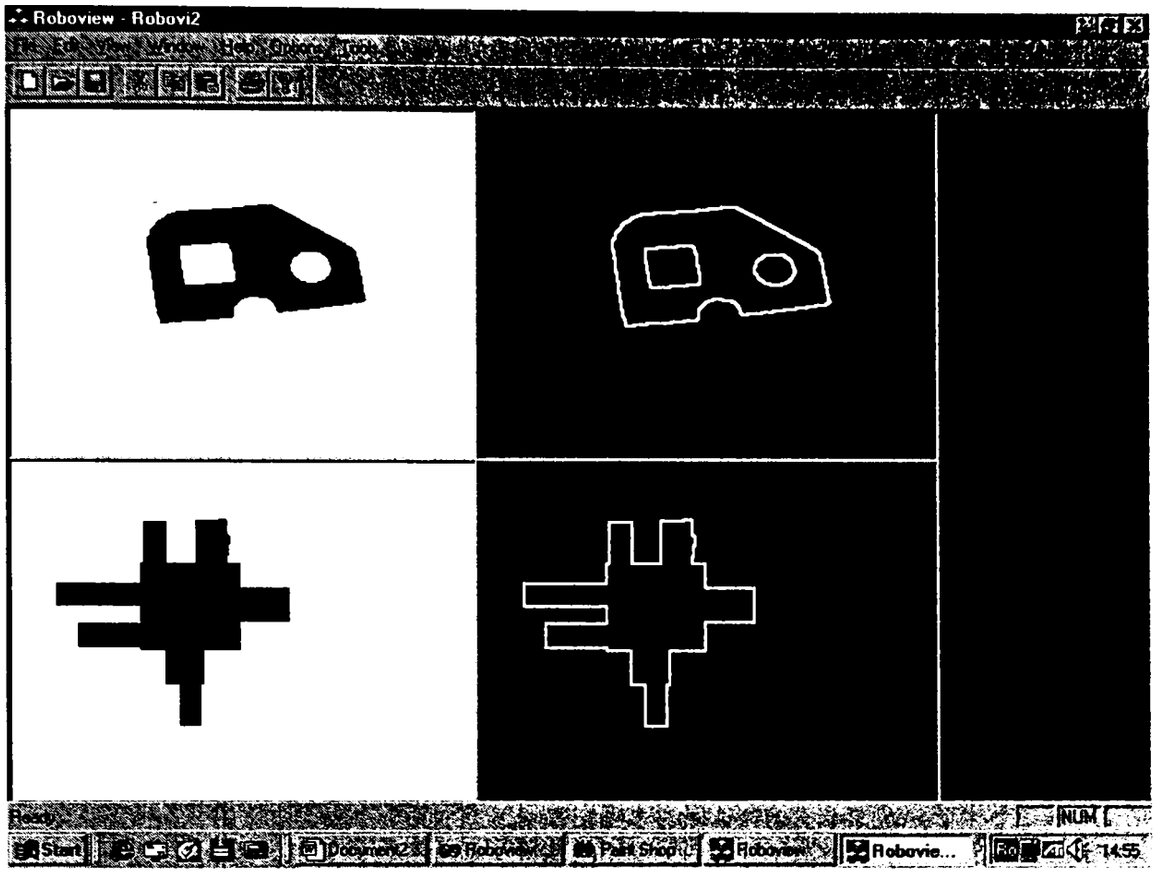
```
BYTE *GetDibBitsAddr(BYTE *DibBuffer)
{
 DWORD  NumColors,ColorTableSize;
 WORD   BitCount;

 if(GetDibInfoHeaderSize(DibBuffer) == sizeof(BITMAPCOREHEADER))
  {
  BitCount = ((BITMAPCOREHEADER *)DibBuffer)->bcBitCount;
  if(BitCount != 24)     NumColors = 1L << BitCount;
  else     NumColors = 0;
  ColorTableSize = NumColors*sizeof(RGBTRIPLE);
  }
 else
  {
  BitCount = ((BITMAPINFOHEADER *)DibBuffer)->biBitCount;
  if(GetDibInfoHeaderSize(DibBuffer) >= 36)
   NumColors = ((BITMAPINFOHEADER *)DibBuffer)->biClrUsed;
  else
   NumColors = 0;
  if(NumColors == 0)
   {
   if(BitCount != 24)  NumColors = 1L << BitCount;
   else  NumColors = 0;
   }
```

```
ColorTableSize = NumColors*sizeof(RGBQUAD);
   }
 return DibBuffer+GetDibInfoHeaderSize(DibBuffer)+ColorTableSize;
}
WORD GetDibWidth(BYTE *DibBuffer)
{
 if(GetDibInfoHeaderSize(DibBuffer) == sizeof(BITMAPCOREHEADER))
  return (WORD)(((BITMAPCOREHEADER *)DibBuffer)->bcWidth);
 else
  return (WORD)(((BITMAPINFOHEADER *)DibBuffer)->biWidth);
}
WORD GetDibHeight(BYTE *DibBuffer)
{
 if(GetDibInfoHeaderSize(DibBuffer) == sizeof(BITMAPCOREHEADER))
  return (WORD)(((BITMAPCOREHEADER *)DibBuffer)->bcHeight);
 else
  return (WORD)(((BITMAPINFOHEADER *)DibBuffer)->biHeight);
}
DWORD GetDibInfoHeaderSize(BYTE *DibBuffer)
{
 return ((BITMAPINFOHEADER *)DibBuffer)->biSize;
}
```

În figura 3.3.1 sunt prezentate câteva imagini din cadrul rulării programului care se referă atât la spațiile de lucru în care au evoluat roboții (fig. 3.3.1,a, b şi d) precum şi alte imagini prelucrate (fig.3.3.1,c şi d) :



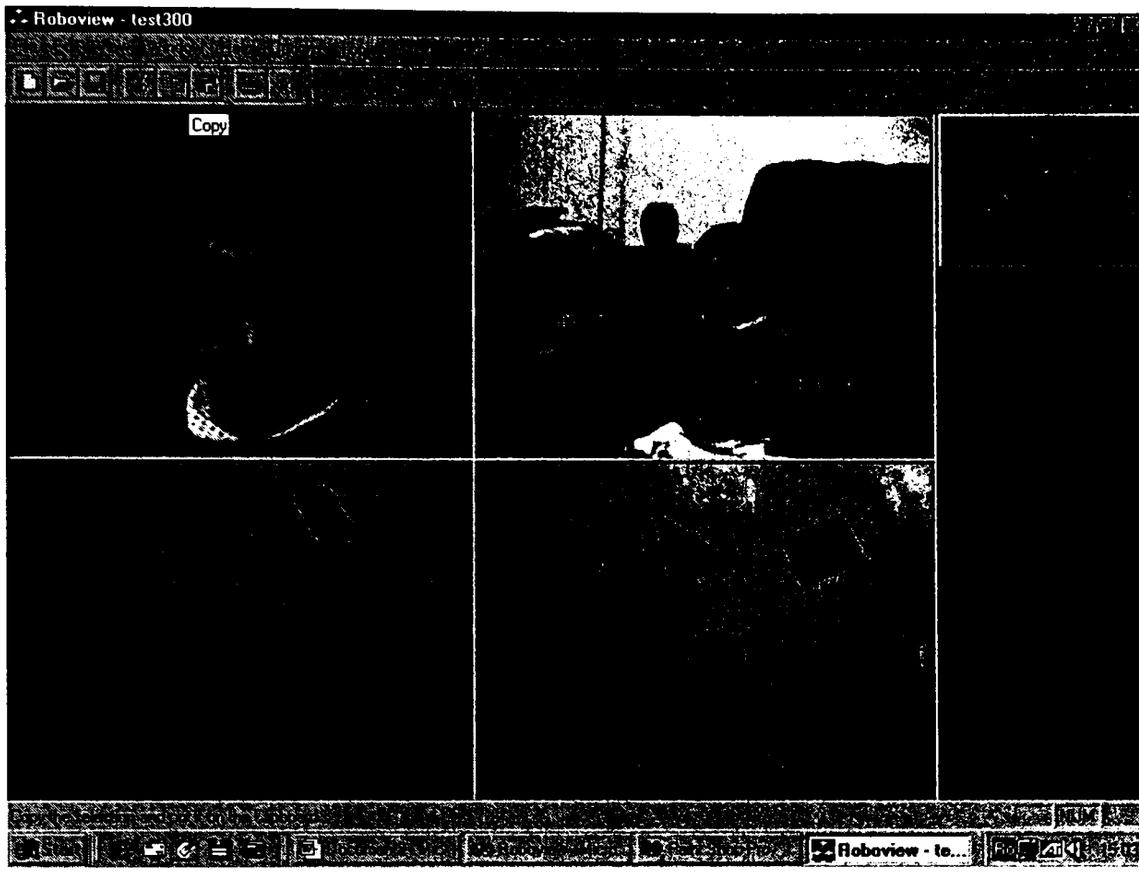
(a)


Roboview - Robovi2
Start
Paint Shop
Roboview
Roboview...
NUM
14:55

(b)


Roboview - Robovi2
File Edit View Window Help Options Tools
Ready
Start
Roboview - R...
NUM

(c)



(d)

Fig. 3.3.1
Exemplificări ale aplicării algoritmilor de recunoaşterea şi prelucrare a imaginilor în cadrul programului principal

Partea IV-a
Concluzii şi contribuţii personale

Principalul scop urmărit în teză este conducerea unui robot mobil printr-un spaţiu populat cu obstacole. Toţi algoritmii şi toate metodele prezentate au fost selectate astfel încât să servească scopului final ales.

La majoritatea capitolelor pe lângă metodele implementate sunt prezentate din punct de vedere teoretic şi alte metode pentru scoaterea în evidenţă a avantajelor şi dezavantajelor metodelor utilizate practic.

Teza este structurată pe trei capitole, fiecare dintre acestea reprezentând cele trei nivele de vedere artificială. Tematica abordată, şi contribuţiile personale a le autorului se pot sintetiza astfel :

- **Partea I-a**

Tratează vederea de complexitate redusă ocupându-se de achiziţia şi preprocesarea imaginii. Această parte este structurată pe 8 capitole, fiecare dintre ele ocupându-se de o anumită ramură din categoria celor de achiziţie şi preprocesare.

Domeniile abordate şi rezolvate în acest cadru sunt :

- ***Achiziţia imaginii***, în care se tratează hardware-ul şi software-ul necesar. Dispozitivele hardware prezentate, sunt camerele video cu tub şi CCD precum şi plăcile de achiziţie utilizate în cadrul experimentelor. Sunt descrise modurile de funcţionare ale camerelor video, avantajele şi dezavantajele celor două tipuri existente în momentul de faţă şi principalele caracteristici ale acestora. Referitor la plăcile de achiziţie sunt prezentate cele două modele personale ale autorului, caracteristicile şi performanţele lor.

  Toată instalaţia experimentală prezentată cu aceste componente a fost realizată integral de autor. S-au încercat şi realizat diferite modalităţi de concepere a instalaţiei şi s-au exprimat în teză concluzii originale cu privire la funcţionare.

  În domeniul software este prezentată librăria Video for Windows, fiind descrise funcţiile, mesajele şi macrourile acesteia. În locurile considerate necesare sunt prezentate exemple de cod în limbaj C pentru înţelegerea modului lor de utilizare şi a scopului deservit de acestea. Sunt prezentate funcţiile realizate în limbaj C, care implementează necesarul de proceduri

pentru captura imaginii utilizate în cadrul experimentelor. *Autorul a realizat în mod original următoarele funcții:*

- *Funcția de enumerare a driverelor de captură existente în sistem*
- *Funcția de conectare a ferestrei de captură cu driverul de captură*
- *Funcția de creare a ferestrei de captură*
- *Funcția principală de captură video*

Pe lângă programul principal, s-au mai făcut experimente cu diferite programe de prelucrare video, cum ar fi:

- Adobe Premiere 4.2
- Digital Video Producer

- ***Prelucrarea imaginilor***, în care sunt exemplificate diferite moduri de procesare în funcție de rezoluție şi nivele de gri. Sunt de asemenea descrise câteva tehnici de iluminare care au ca scop obținerea unor imagini clare, fără umbre şi fără efect de „spot-light". .

Este contribuția autorului această sinteză de tehnici de prelucrare împreună cu toate exemplele de imagini prezentate.

- ***Filtrarea imaginilor***, care constituie prima etapă în orice algoritm de vedere artificială. Aceasta are rolul de a elimina zgomotele datorate conversie analog-numerice. *Autorul a prezentat diferite metode existente în literatura de specialitate cât şi implementarea lor sub formă de funcții în limbajul C++. Întreaga prezentare este în mod original ilustrată* în vederea evidențierii avantajelor şi dezavantajelor oferite de algoritmii utilizați.

Autorul a creat proceduri originale în limbajul C++ pentru următoarele metode de filtrare:

- Metoda mediei vecinătății
- Metoda filtrării mediane
- Metoda mediei imaginilor
- Metoda convoluției şi corelației
- Metoda filtrelor spațiale trece-jos
- Metoda filtrelor spațiale trece-sus
- Metoda filtrelor minim şi maxim

- Metoda filtrării imaginilor binare

Toate aceste metode au fost ilustrate grafic pe imagini prelucrate cu proceduri originale concepute de autor.

În urma experimentelor realizate autorul a constatat că metoda filtrului median şi cea a filtrării imaginilor binare sunt cele mai adecvate vederii artificiale industriale, având cele mai bune rezultate în eliminarea zgomotelor. S-a constatat de asemenea că uneori este necesară aplicarea succesivă a acestor metode pentru eliminarea în totalitate a perturbaţiilor. *Aplicarea acestor metode la recunoaşterea obiectelor este de asemenea o contribuţie originală.*

- ***Îmbunătăţirea imaginii***, care tratează eliminarea efectelor datorate iluminării neuniforme. Au fost prezentate din punct de vedere teoretic trei metode :
  - Metoda egalizării histogramei
  - Metoda specificării histogramei
  - Îmbunătăţirea locală a imaginii

  Deoarece doar prima metodă se poate implementa într-un algoritm ce nu necesită intervenţia utilizatorului, aceasta a fost singura implementată practic. *Funcţia de implementare şi imaginile prelucrate şi prezentate sunt contribuţii originale ale autorului* şi scot în evidenţă eficienţa acestei metode în extragerea detaliilor ascunse în umbra unei iluminări neadecvate.

- ***Detecţia de contur***, care este un prim pas în vederea procesării imaginii, tratând extragerea contururilor dintr-o imagine capturată. Pe lângă partea de teorie referitoare la utilizarea operatorilor de tip gradient şi Laplacian, *s-a prezentat şi implementarea originală a unei funcţii ce utilizează mai mulţi operatori de tip mască :*
  - *Freeman*
  - *Soebel*
  - *Prewitt*
  - *Kirsch*
  - *Vertical*
  - *Orizontal*
  - *45°*

În urma testărilor repetate, autorul a concluzionat că, cele mai bune rezultate experimentale s-au obținut cu operatorii Freeman.

Deoarece ceilalți operatori ofereau rezultate acceptabile în funcție de cazurile particulare studiate, *autorul a realizat programul principal astfel încât să ofere utilizatorului toate tipurile de operatori implementate urmând ca acesta să decidă în funcție de aplicație.*

Pe parcursul tezei au fost prezentate în mod original un mare număr de imagini a căror detecție de contur a fost realizată experimental.

- ***Stabilirea unui prag pentru prelucrarea imaginii***, care tratează o metodă de binarizare a imaginii, metodă necesară deoarece toți algoritmii de procesare utilizați ulterior lucrează cu imagini binare ca imagini de intrare. Metoda construieşte histograma imaginii şi selectează intensitatea de separație între pixelii fondului şi pixelii obiectelor. *Atât funcția de selecție a punctului de prag cât şi cea de calcul şi afişare a histogramei, amândouă în limbajul C++, sunt contribuții originale ale autorului şi au fost prezentate împreună cu exemple concrete în cadrul tezei.*

- ***Domeniile de frecvență***, care prezintă doar un punct de vedere teoretic, pentru a se putea explica avantajele şi dezavantajele metodei domeniilor spațiale. Din cauza necesarului mare de calcul şi timp, aceste metode nu sunt compatibile cu cerințele vederii artificiale industriale.

➢ **Partea II-a.**

Tratează în trei capitole distincte problemele legate de vederea de nivel mediu, adică cele referitoare la segmentarea, descrierea şi recunoaşterea imaginilor. Cea ce autorul a soluționat în această parte a tezei se referă la :

- ***Segmentarea imaginii,*** care se ocupă cu extragerea de obiecte din cadrul unei imagini. Algoritmii de segmentare sunt în general bazați pe două principii importante: discontinuitatea şi similitudinea. Au fost prezentate mai multe metode teoretice de segmentare :

- Conectarea marginilor şi detecţia de contur
- Folosirea pragului pentru segmentarea imaginii
- Segmentarea orientată pe regiuni
- Utilizarea mişcării

Ca o sinteză a tuturor acestor metode autorul a prezentat în finalul capitolului o metodă originală a autorului de segmentare care se bazează atât pe folosirea pragurilor cât şi pe detecţia de contur.

Principala noutate adusă este utilizarea unui rastru, care permite obţinerea unor puncte ce ulterior vor fi utilizate în faza de descriere.

Pe lângă elaborarea teoretică originală a algoritmului, este creată şi o funcţie de implementare originală, care este prezentată la partea de descriere a imaginii.

Prezentarea algoritmului este susţinută şi cu exemple originale obţinute de către autor prin experimentare.

- ***Descrierea***, care se ocupă cu extragerea de informaţii referitoare la obiectele rezultate în urma segmentării, informaţii ce definesc caracteristicile de formă sau textură ale obiectelor. Aceste caracteristici definesc obiectele, şi vor ajuta la distingerea unui obiect faţă de altul. Principalul algoritm urmărit în acest capitol este utilizarea numerelor de formă. Acestea au marele avantaj că pot fi normalizate pentru următoarele situaţii practice :
  - Modificarea orientării
  - Modificarea mărimii
  - Modificarea punctului de iniţializare

  Astfel, corpuri identice ca formă, dar orientate diferit faţă de camera video, având dimensiuni diferite pot fi grupate în aceeaşi clasă de obiecte şi recunoscute ca atare.

  Descrierea pe baza numerelor de formă este una din rezultatele importante ale tezei.

  Având în vedere probleme concrete apărute, legate de figurile geometrice plane, s-a creat un algoritm original de normalizare la orientare.

Întregul set de algoritmi necesari descrieri cu ajutorul numerelor de formă a fost implementat în mai multe funcții C++, care sunt contribuții originale ale autorului.

De asemenea, desfășurarea algoritmului a fost bogat ilustrată cu imagini obținute experimental de către autor urmărindu-se atent fiecare pas al determinării numărului de formă.

Pe lângă acest algoritm au mai fost prezentate din punct de vedere teoretic şi alte metode ca :

Semnături

- Aproximări poligonale
- Coduri de înlănțuire
- Descriptori Fourier
- Descriptorii regiunilor

- ***Recunoaşterea***, se ocupă de etichetarea obiectelor segmentate şi ataşarea lor unor clase cunoscute de obiecte. Din punct de vedere teoretic s-au prezentat două tipuri de metode :
  - Metode de decizie teoretică
  - Metode de decizie structurală

  În cadrul metodelor de decizie structurală s-a implementat un algoritm original bazat pe existența unei baze de date, formată din numere de formă, ce conține descriptori de figuri geometrice plane.

  Crearea bazei de date s-a realizat în mod experimental de către autor, în primă fază, pe corpuri ideale, iar ulterior a fost dezvoltată pe măsură ce rezultatele experimentale o demonstrau necesar.

  Întregul algoritm a fost scris în C++, şi este contribuție originală a autorului, iar rezultatul aplicării lui se reprezenta pin colorarea în diferite culori a diverselor clase de figuri geometrice întâlnite.

➢ **Partea III-a**

Partea a treia tratează procesul de conducere a doi roboți printr-un spațiu de lucru populat cu obstacole. Se realizează o prezentare a roboților ce se deplasează pe baza unor traiectorii planificate. Autorul selectează dintre metodele de planificare

cunoscute, cele care se pretează formelor roboților și obstacolelor, precum și mecanismelor motoare ale roboților. Se prezintă achiziția de imagini în timp real și descrierea generală a programului principal.

Obiectivele urmărite și realizate sunt :

- ***Descrierea primului robot efector***, care prezintă modul de realizare a acestuia, prezentând teoretic modul de funcționare al motoarelor pas cu pas folosite și schema electronică de comandă a acestora. *Autorul a adaptat o sursă comercială de curent, printr-un stabilizator de tensiune în vederea asigurării alimentării cu 5V a robotului.*

- ***Descrierea rutinelor de comandă***, *care au fost create special de autor pentru conducerea robotului prin calculator și care asigură deplasarea robotului prin spațiul de lucru.*

  S-au creat rutine pentru deplasarea robotului înainte și înapoi, pentru virare la stânga și la dreapta. De asemenea s-au folosit operații cu fișiere de intrare ieșire, specifice programelor Windows, în locul tradiționalelor operații de intrare-ieșire din MS-DOS.

  Astfel au fost create în mod original următoarele funcții de comandă:

  - *Funcție pentru inițializarea portului paralel*
  - *Funcție pentru deplasarea înainte a motorului din dreapta*
  - *Funcție pentru deplasarea înapoi a motorului din dreapta*
  - *Funcție pentru deplasarea înainte a motorului din stânga*
  - *Funcție pentru deplasarea înapoi a motorului din stânga*
  - *Funcție pentru deplasarea înainte a robotului*
  - *Funcție pentru deplasarea înapoi a robotului*
  - *Funcție pentru rotirea la dreapta a robotului*
  - *Funcție pentru rotirea la stânga a robotului*

- ***Recunoașterea și maparea spațiului de lucru***, care se ocupă cu aspecte ale planificării mișcării, fiind prezentate două metode de planificare:
  - Metoda grilei neomogene
  - Metoda grafului vizibilității

Ambele metode sunt exemplificate prin număr mare de imagini realizate în mod experimental de către autor, pornind de la spațiul de lucru populat cu obstacole plane, a cărui imagine a fost capturată și procesată anterior.

S-au realizat mai multe experimente originale, de determinare a traiectoriei prin modificarea punctelor de start și final.

- ***Conducerea unui robocar cu achiziția imaginii în timp real*** este o problemă foarte complexă și laborioasă, ce necesită un hardware foarte rapid și un software optimizat pentru cazurile particulare experimentate.

  Este descris pe scurt robocarul utilizat în cadrul acestor experimente, care a fost comandat prin interfața serială a unui calculator și are atașată o cameră video solidară cu roata directoare.

  Ideea aplicată în cadrul acestor experimente este o contribuție originală a autorului.

  Rutina de achiziție și prelucrare are trasate condiții inițiale restrictive, referitoare la spațiul de lucru și la modul de recunoaștere a obiectului.

  Crearea spațiului de lucru a pus probleme complicate datorită necesității eliminării umbrelor și a obiectelor false. De asemenea obiectul urmărit a trebuit realizat dintr-un material mat care nu reflectă lumina, pentru a nu crea efectul de „spot-light".

  Experimentul a reușit urmărirea fără dificultăți a centrului de greutate al obiectului prin spațiul de lucru de către robocar.

  Întregul experiment de conducere a robocarului pe baza imaginii achiziționate și prelucrate în timp real a putut fi realizat pe baza programelor concepute și aplicate de autor.

  Se prezintă un exemplu original de planificare a mișcării robocarului printr-un spațiu de lucru ce reprezintă o secțiune plană a laboratorului de Organe de Mașini și Mecanisme al UPT. Și în acest caz s-au folosit cele două metode de planificare a traiectoriilor utilizate pentru robotul anterior.

  Autorul a realizat diferite experimente pentru recunoașterea în timp real a obiectului, prin impunerea de condiții de simplificare, dar viteza tehnicii de calcul utilizate nu a permis obținerea unor rezultate satisfăcătoare, decât în anumite cazuri.

- ***Prezentarea programului principal*** constituie partea ce *descrie modul original de realizare a programului principal şi obiectele importante ale acestuia. Sunt prezentate în mod original funcţiile necesare în operarea cu imaginile de tip bitmap realizate în limbajul C++ :*
  - *Citirea unei imagini bitmap*
  - *Salvarea unei imagini bitmap*
  - *Afişarea unei imagini bitmap*

Toate funcţiile prezentate în această teză cât şi programul principal sunt contribuţii originale ale autorului.

Bibliografia

1. A.Blake, M.Isard "Active Contours The application of Tehniques from Graphics, Vision , Control Theory ans Statisticsto Visual Tracking of Shapes in Motion". Springer-Verlag New York, 1998
2. A.Craig Lindley: "Practical Image Processing in C". John Wiley & Sons, New York 1991
3. A.D.Greenberg, S.Greenberg : "Digital Images : A Practical Guide". McGraw-Hill, 1995
4. A.Davies : "The Digital imaging A-Z". Butterworth-Heinemann, 1997
5. A.H.Watt, F.Policarpo : "The Computer Image". Addison Wesley Longman, 997
6. A.K.Jain : "Fundamentals of Digital Image Processing". Prentice Hall, 1989
7. A.Kelemen : "Motoare electrice pas cu pas", Curs Lito UPT, 1975
8. A.Khotanzad, R.L.Kashyap : "Texture Classification Using whose Effectiveness can be evaluated a priori". IEEE Trans. Man, Cybernetics, vol. SMC-17, nr. 6, Nov. 1987
9. A.Lambert, N.Le Fort-Piat : "Robot tasks planning integrating uncertainties and local maps". AVCS'98, Amiens, France, 1998
10. A.M.Barry : "Visual Intelligence". State University of New York Press, 1997
11. A.Pugh : "Robot vision". Editions IFS Publications, Springle, New York, NY, 1983
12. A.R.Weeks : "Fundamentals of Electronic Image Processing". SPIE-International Society for Optical Engineering, 1996
13. A.S.Glassner : "Principles of Digital Image Synthesis". Morgan Kaufmann Pubs., 1994
14. Alan V. Oppenheim, Ronald W. Schafer: "Digital Signal Processing". Prentice-Hall, Englewood, 1982
15. Arbib, Michael A. and Hanson, Allen R.: "Vision, Brain, and Cooperative Computation". MIT Press, Cambridge, MA, 1987
16. Azriel Rosenfeld, Avinash Kak: "Digital Picture Processing". Academic Press, Orlando, 1982

17. B. Bouilly, T.Simeon, R.Alami : "A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty". IEEE International Conference on robotics and Automation, 1995
18. B.Jahne : "Digital Image Processing". Springer-Verlag New York, 1997
19. Bart Kosko & J.S. Limm : "Vision as a Causal Activation and Association". S.P.I.E Proceedings, Sept. 1985
20. Blanton Keith : "Image Extrapolation for Flight Simulator Visual Systems", AIAA conference, 1988
21. C. Canudas, H.Khennouf, C.Samson, and O.J. Sordalen "Nonlinear ControlDesign for Mobile Robots". World Scientific Series in Robotics and Automated Systems, Vol. 11, 1993
22. C.Strothotte, T.Strothotte : "Seeing between the Pixels". Springer-Verlag New York, 1997
23. C.T.Leondes : "Image Processing and Pattern Recognition". Academic Press, 1997
24. Christopher Watkins, Alberto Sadun, Stephen Marenka: "Modern Image Processing: warping, morphing, and classical techniques". Academic Press, London 1993
25. CI. Laurgeau & M. Parrent : "Les machines de vision en productique". Editions E.T.A., Strasbourg, 1985
26. D. Drăgulescu, D. Iosif: "Using shape numbers for image description and recognition". Buletinul Şt. şi Tehnic al UTT, seria Automatică şi calculatoare, Tom 42(56), 1997
27. D. Drăgulescu, D. Iosif: "Utilizarea numerelor de formă în descrierea şi recunoaşterea imaginilor". Analele Universităţii din Oradea, Fascicola Mecanică, 1997
28. D. Drăgulescu, D. Iosif: "Utilizarea numerelor de formă în recunoaşterea figurilor geometrice plane". Buletinul AGIR nr. 3, 1998
29. D. Iosif : "Curs de vedere artificială", dat spre publicare la Editura Universităţii Banatului,Tmişoara, 1999
30. D.A.Lyon : "Image Processing in Java". Prentice Hall,1999

31. D.Andreiciuc, cond. şt. T.Mureşan : "Teză de doctorat – Optimizarea conduceri vehiculelor ghidate automat" 1999
32. D.Drăgulescu : "Dinamica Roboţilor". Ed. Didactică şi Pedagogică, Bucureşti, 1997 -
33. D.Drăgulescu, C.Couturier : "Cours de modelisation des robots". Lito. Universite d'Artois, Franţa, 1995
34. D.Drăgulescu, F. Moldovan : "Legile de mişcare ale unui robot RRT cu momente de inerţie variabile". A VIII-a Conferinţă de Vibraţii Mecanice, Timişoara, 1996
35. D.Drăgulescu, F. Moldovan, H. Moldovan "Quantitative and qualitative measurements on populated printed circuits boards". The 4-th International Conference of Electric, Electronic Equipement, Braşov, 1994
36. D.Drăgulescu, F. Moldovan, H.Moldovan : "Metoda de determinare a marginilor unui C-obstacol". Al XII-lea Simpozion Naţional de Roboţi Ind., Timişoara, 1994
37. D.Drăgulescu, F. Moldovan, H.Moldovan "Metoda descompunerii celulare exacte. Curbe critice". Al XII-lea Simpozion Naţional de Roboţi Ind., Timişoara, 1994
38. D.Drăgulescu, F. Moldovan, H.Moldovan: "Considerations about the critical curves at the exact cell decomposition method". International Conference on Technical Informatics, Conti'94, Timişoara, 1994
39. D.Drăgulescu, F. Moldovan, M. Toth-Taşcău "Metode de planificare a traiectoriilor la roboţi. Sinteză". Analele Universităţii Eftimie Murgu, Reşiţa, 1995
40. D.Drăgulescu, F. Moldovan, M. Toth-Taşcău, : "Considerations about the critical curves at the exact cell decomposition method". Buletin Şt. Şi Tehnic al UTT, seria Mecanică, Tom 40 (54), 1995
41. D.Drăgulescu, F. Moldovan, M. Toth-Taşcău, "Soluţionarea problemei cinematice inverse pentru un robot având 6 GL, prin metoda separării matricelor". Analele Universităţii Oradea, 1995
42. D.Drăgulescu, F.Moldovan : "Trajectory control of RRT robot with timevarying inertia links". International Conference on Technical Informatics, Conti'96, Timişoara, 1996
43. D.Drăgulescu, F.Moldovan, M. Toth-Taşcău, C.Crivacucea "Metode de planificare a traiectoriilor la roboţi. Sinteză".Analele universităţii Eftimie Murgu, Reşiţa, anul III, 1996

44. D.Drăgulescu, M. Toth-Taşcău " Determinarea ecuaţiilor de mişcare pe baza modelului dinamic pentru un manipulator cu trei grade de libertate". The 25-th Conference on Production Engineering, Beograd, Jugoslavia, 1994
45. D.Drăgulescu, M. Toth-Taşcău : "Comparative dynamic study of a serial robot". Mecanique-Industrie-Materiaux, vol. 6, Franţa, 1996
46. D.Drăgulescu, M. Toth-Taşcău : "Comparative dynamic study of a serial robot". The 26-th Conference on Mechanical Engineering, Technion city, Israel, 1996
47. D.Drăgulescu, M. Toth-Taşcău : "Contributions to cell decomposition method for a two-dimensional work space". International Conference on Technical Informatics, Conti'96, Timişoara, 1996
48. D.Drăgulescu, M. Toth-Taşcău "Contributions to motion planning using potential field method". International Conference on Technical Informatics, Conti'96, Timişoara, 1996
49. D.Drăgulescu, M. Toth-Taşcău : "Coordonarea cuplelor cinematice ale unui robot prin metoda timpului minimal". Al XII-lea Simpozion Naţional de Robotică, Reşiţa, 1996
50. D.Drăgulescu, M. Toth-Taşcău "Determinarea preciziei de parcurgere a taiectoriei unui robot". A XXVII-a Ses. De Comunicări Ştiinţifice cu Part. Intern., Ac. Tehnică Militară, Bucureşti, 1997
51. D.Drăgulescu, M. Toth-Taşcău : "Dynamic comparative study of a welding robot CLOOS". Third European Conference on Joining Technology EUROJOIN3, Berna, Elveţia, 1998
52. D.Drăgulescu, M. Toth-Taşcău : "Dynamic study of a welding robot". The 25-th Conference on Production Engineering, Beograd, Jugoslavia, 1994
53. D.Drăgulescu, M. Toth-Taşcău : "Generation of six Degrees of Freedom Robot path". Analele Universităţii din Oradea, 1994
54. D.Drăgulescu, M. Toth-Taşcău : "Influence of the robot joints vibrations on the position of the end effector". A V-a Conferinţă de Matematică Aplicată şi Mecanică, Cluj-Napoca, 1996
55. D.Drăgulescu, M. Toth-Taşcău : "Influenţa vibraţiilor din cuplele unui robot asupra poziţiei efectorului final". A VIII-a Conferinţă de Vibraţii Mecanice, Timişoara, 1996

56. D.Drăgulescu, M. Toth-Taşcău : "Interpolarea traiectoriilor unui robot cu funcţii spline şi restricţii cinematice şi dinamice". The VII-th Symposium of Mathematics and its Applic., univ. Politehnica timişoara, 1997
57. D.Drăgulescu, M. Toth-Taşcău : "Kinemati and dynamic study of a welding robot". Second European Conference on Joining Technology EUROJOIN2, Florenţa, Italia, 1994
58. D.Drăgulescu, M. Toth-Taşcău "Kynematics analysis, joint torques and dynamics of a welding robot". Analele Universităţii din Oradea, 1994
59. D.Drăgulescu, M. Toth-Taşcău : "Method to generate an imposed trajectory for a robot having 6 degrees of fredom". International Conference on Technical Informatics, Conti'94, Timişoara, 1994
60. D.Drăgulescu, M. Toth-Taşcău : "Method to study the dynamical behaviour of a complex mechanical structure". International Conference, University of the West, Timişoara, 1997
61. D.Drăgulescu, M. Toth-Taşcău : "Modelarea roboţilor cu structura arborescentă şi lanţ cinematic închis". The VII-th International Conference of Manufacturing Engineering Techno'95, Timişoara, 1995
62. D.Drăgulescu, M. Toth-Taşcău : "Modelarea spaţiului de lucru prin metoda arborelui. Workspace modelling by tree method". Analele Universităţii din oradea, Fascicola Mecanică, 1997
63. D.Drăgulescu, M. Toth-Taşcău : "Modelling of robot vibrations by rigid finite elemnt method". First Romanian-Japanese Joint Seminar on Applied Electromagnetics and Mechanics, RJJSAEM, Neptun, 1996
64. D.Drăgulescu, M. Toth-Taşcău "Motion planning for a movable robot by potential field method". Buletinul Şt. Şi Tehnic al UTT, seria Mecanică, Tom 41(55), 1996
65. D.Drăgulescu, M. Toth-Taşcău "Path planning for a robot by exact cell decomposition method". II-nd International Conference on Advanced Robotics, Viena, 1996
66. D.Drăgulescu, M. Toth-Taşcău : "Planificarea traiectoriei unui robot prin metoda câmpului potenţial". Al XII-lea Simpozion Naţional de Roboţi Ind., Timişoara, 1994

67. D.Drăgulescu, M. Toth-Taşcău : "Position and orientation errors due to robot elemnet deformations". The 30-th International Symposium on Automotive Technology and Automation ISATA'97, Florenţa, Italia, 1997

68. D.Drăgulescu, M. Toth-Taşcău : "Studiul cinematic al unui robot KUKA cu şase grade de libertate". Buletinul Conferinţei Naţionale de dinamica Maşinilor CDM 94, Braşov,1994

69. D.Drăgulescu, M. Toth-Taşcău : "Studiul dinamicii unui robot tip KUKA". The VII-th International Conference of Manufacturing Engineering Techno'95, Timişoara, 1995

70. D.Drăgulescu, M. Toth-Taşcău : "Variaţia elementelor torsorului de reducere din cuplele unui robot RTRTTR". A VIII-a Conferinţă de Mecanica Solidelor, Constanţa, 1994

71. D.Drăgulescu, M. Toth-Taşcău : "Workspace modelling by non-homogeneous grid and tree method". Buletinul Şt. Şi Tehnic al UTT, seria Mecanică, Tom 42(56), 1997

72. D.Drăgulescu, M. Toth-Taşcău, D. Iosif: "Dynamical Study of the Welding Robot CLOOS". Analele Universităţii Eftimie Murgu Reşiţa, 1997

73. D.Drăgulescu, M. Toth-Taşcău, D.Iosif : "Dynamical study of the welding robot CLOOS with rigid links". Buletinul Şt. şi Tehnic al UTT, seria Mecanică, Tom 42(56), 1997

74. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan : "Consideraţii asupra metodei de recunoaştere prin arbore de decizie". A II sesiune de comunicări ştiinţifice cu participare internaţională "Realizări tehnice şi cultural-ştiinţifice pe meleaguri arădene", Arad, 1994

75. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan : "Dinamica robotului KUKA tip 364/10.0". Acta Universitatis Cibiniensis, Vol XiX, Universitatea L. Blaga, Sibiu, 1995

76. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan "Kinematic restrictions in mouvements of robot". Analele Universităţii Oradea, 1995

77. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan "Metoda de planificare a traiectorilor în spaţiul configuraţiilor bidimensional. Metoda grafului vizibilităţii". Analele Universităţii Oradea, 1995

78. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan : "Metodă şi algortim de planificare a triectorilor plane". A II sesiune de comunicări ştiinţifice cu participare internaţională "Realizări tehnice şi cultural-ştiinţifice pe meleaguri arădene", Arad, 1994

79. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan : "Planificarea mişcării roboţilor", Ed. Helicon, Timişoara, 1995

80. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan : "Soluţionarea problemei dinamice a robotului KUKA tip 364/10.0 în condiţii date de mişcare". Analele Universităţii Eftimie Murgu, Reşiţa, 1995

81. D.Drăgulescu, M. Toth-Taşcău, F. Moldovan, H. Moldovan : "Consideraţii asupra unei metode de recunoaşterea formelor". A II sesiune de comunicări ştiinţifice cu participare internaţională "Realizări tehnice şi cultural-ştiinţifice pe meleaguri arădene", Arad, 1994

82. D.Drăgulescu, M. Toth-Taşcău, F.Moldovan : "Soluţionarea problemei dinamice a robotului KUKA tip 346110.0 în condiţii date de mişcare".Analele universităţii Eftimie Murgu, Reşiţa, anul III, 1996

83. D.Drăgulescu, M. Toth-Taşcău, G. Pasco : "Utilizarea matricelor de transfer la calculul forţelor generalizate pentru un robot". Conferinţa Naţională de Dinamica Maşinilor CDM 97, Braşov, 1997.

84. D.Drăgulescu, M. Toth-Taşcău, G.Pasco "Path planning by the exactcell decomposition method". Buletinul Şt. Şi Tehnic al UTT, seria Mecanică, Tom 41(55), 1996

85. D.Drăgulescu, M. Toth-Taşcău: "Studiul mişcării robotului în timp minim, în spţiul operaţional". A II sesiune de comunicări ştiinţifice cu participare internaţională "Realizări tehnice şi cultural-ştiinţifice pe meleaguri arădene", Arad, 1994

86. D.H. Ballard : "Generalizing the Hough Transform to Detect Arbitrary Shapes". Pattern Recognition, vol. 13, Nr. 2, 1981

87. D.H. Ballard and C.M. Brown : "Computer Vision". Prentice-Hall, Englewood Cliffs, N.J., 1982

88. D.Pârvuţi, coord.şt. N. Robu : "Proiect de diplomă – Robot cu vedere artificială", UTT, 1995

89. D.Phillips : "Image Processing in C". R&D Books, 1996

90. D.W. Wloka : "Robotersysteme", Springer-Verlag, Berlin, 1992

91. E.Gose, R.Johnsonbaugh : "Pattern Recognition and Image Analysis" Prentice Hall, 1996

92. Embree, Paul M., and Kimble, Bruce : "C Language Algoritms for Digital Signal Processing". Prentice-Hall, Englewood Cliffs, NJ, 1991

93. Foley, James, van Dam, Andries, Feiner, Steven, and Hughes, John : "Computer Graphics Principles and Practice", 2nd ed. Addisson Wesley, Reading, MA, 1990

94. G. Mezin & B. Bretagnolles :"la Vision Artificielle". Societe ITMI, Measures, Avril 1986

95. G.A.Baxes : "Digital Image Processing: Principles and Applications". John Wiley and Sons Incorporated, 1994

96. G.I. Mitrofan : "Televiziune", 1996

97. G.I.Mitrofan : "Televizinea digitală". 1986

98. G.X.Ritter, J.N.Wilson : "Handbook of Computer Vision Algorithms in Image Algebra". CRC Press Inc. , 1996

99. George Wolberg: "Digital Image Warping". IEEE Computer Society Press Monograph, Loa Alamitos, 1990

100. Glasner, Andrew S. : "An Introduction to Ray Tracing". Academic Press, London , 1989

101. H. Li, M.A. Lavin & R.J. Le Master : "Fast Hough Transform : A Hierarchical Approach". Comput. Vision, Graphics Image Proc, vol. 36, 1986

102. H.E.Burdik : "Digital Imaging: Theory and Applications". McGraw-Hill, 1997

103. H.R.Myler, A.R.Weeks : "The Pocket Hanbook of Imaging Processing Algortims in C". Prentice Hall, 1993

104. H.Stark, Y.Yang : "Vector Space Projections". John Wiley & Sons, 1998

105. Hesse, S. : "Industrie-roboter-peripherie". Verlag Technik, Berlin, 1989

106. I. Boldea, M.Brehui, R.Dumitraşcu : "Transformatoare şi maşini electrice", Curs lito UPT, 1994

107. I. Nicoară, L.Mădăras : "Bazele proiectării transmisiilor mecanice", 1996

108. I.Pitas : "Digital Image Processing Algoritms". Prentice Hall,1993

109. J. Farace : "Digital Imaging : Tips, Tools and Techiques for Photographers". Butterworth-Heinemann,1998

110. J.Blinn : "Jim Blinns's Corner : Dirty Pixels". Morgan Kaufmann Pubs., 1998

111. J.C. Latombe : "Robot motion planning". Kluwer Academic Publisher, 1991

112. J.C.Russ : "The image processing Handbook". CRC Press Inc., 1998

113. J.Farace : "The Digital Image Dictionary". Allworth Press Pubs., 1996

114. J.G. Postaire : "De l'Image a la Decision". Editions Dunod-Informatique, Bordas, Paris, 1987

115. J.Gomes, L. Velho, S.Levy "Image Processing for Computer Graphics". Springer-Verlag New York, 1997

116. J.J. Graig : "Introduction to Robotics, Mechanics and Control", 2nd edition, Addison Wesley, 1989.

117. J.L.Starck, F.Murtagh, A.Bijaoui "Image processing and Data Analysis". Cambridge University Press, 1998

118. J.R.Parker : "Algorithms for Image Processing and Computer Vision". John Wiley & Sons, 1996

119. J.R.Parker : "Practical Computer Vision Using C". John Wiley & Sons, 1993

120. K.R.Castleman : "Digital Image Processing". Prentice Hall, 1995

121. K.R.Pender : "Digital Graphic Design". Butterworth-Heinemann, 1997

122. K.S.Fu, R.C.Gonzalez, C.S.G.Lee: "Robotics - Control, Sensing, Vision, and Intelligence". Academic Press, Boston 1990

123. Keel, William C. "A Simple Photometrically Accurate Algorithm for Deconvolution of Optical Images". Department of Physics and Astronomy, University of Alabama

124. L.D.Şerbănaţi, cond. şt. M. Petrescu : "Teză de doctorat -Implementarea unui limbaj pentru un sistem interactiv a datelor", 1980

125. Liu H., Meusel P., Hirzinger G. : "A Tactile Sensing system for the DLR Three-Finger robot Hand, Proc. Of the ISMCR'95 Conf., 1995

126. M. Toth-Taşcău, D.Drăgulescu : "Generarea mişcării robotului în coordonate operaţionale carteziene". Al XII-lea Simpozion Naţional de Roboţi Ind., Timişoara, 1994

127. M. Toth-Taşcău, D.Drăgulescu : "Generarea traiectoriei unui robot pe baza modelului dinamic". Buletinul Conferinţei Naţionale de dinamica Maşinilor CDM 94, Braşov, 1994

128. M. Toth-Taşcău, D.Drăgulescu : "Minimurile locale în planificarea mişcării prin metoda câmpului potenţial". Al XII-lea Simpozion Naţional de Robotică, Reşiţa, 1996

129. M. Toth-Taşcău, D.Drăgulescu : "Path Planning for a Car-like Robot by Potential Field Method". The 29-th International Symposium on Automotive Technology and Automation ISATA'96, Florența, Italia, 1996

130. M. Toth-Taşcău, D.Drăgulescu : "Precizia cinematică a parcurgerii traiectoriei unui robot". Acta Universitatis Cibiniensis, Vol XiX, Universitatea L. Blaga, Sibiu, 1995

131. M.Awad, J.Kuusela : "Object-Oriented Technology for Real Time Systems". Prentice Hall, 1996

132. M.Sonka, V.Hlavac, R.Boyle "Image Processing, Analysis, and Machine Vision". PWS Publishing, 1998

133. Microsoft Visual C++ & Video for Windows Help.

134. Oppenheim, Allan V., and Schafer, Ronald W. : "Digital Signal Processing". Prentice-Hall, Englewood, NJ, 1985

135. P.E.Mattinson : "Practical Digital Video with Programming Examples in C". John wiley & Sons, 1994

136. P.Soueres, T.Hamel, V.Cadenat, A.Dzual "A Reactive Path Following Controller for Mobile Robots". AVCS'98, Amiens, France, 1998

137. Philippe Coiffet : "Les Robots - Interaction avec l'Environnement". Demand, Paris 1991

138. R.A.Schowenger "Remote Sensing Models and Methods for Image Processing". Academic Press, 1997

139. R.B.MacAllister : "Scanning and Image Manipulation". Delmar Publishers, 1996

140. R.C:Gonzales, R:E:Woods "Digital Image Processing", Addison Wesley Longman, 1992

141. R.Crane, P.S:Hewlett : "A Simplified Approach to Image Procesing". 1996

142. R.M.Haralic, L.G.shapiro : "Computer and Robot Vision". Addison Wesley Logman, 1992

143. R.R.Hendee, N.T.Wells, : "The Perception of Visual Information". Springer-Verlag New York, 1997

144. S.Călin, I.Dumitrache : "Curs Regulatoare automate", 1985

145. S.E. Umbaugh : "Computer Vision and Image Processing". Prentice Hall, 1997

146. Sury J., Remsa V. :" Roboty slouzi cloveku", Nase vojsko, Praha, 1982

147. Th. Borangiu, I. Dumitrache "Intelligent Manufacturing Systems 1995". Elsevier Science, 1997

148. Theo Pavlidis: "Algorithms for Graphics and Image Processing". Computer Science-Press, Rockville, 1982

149. Tzay Y. Young, King-Sun Fu: "Handbook of Pattern Recognition and Image Processing". Academic Press, San Diego, 1986

150. V.Bashkaran, K.Konstantinidies : "Image and Video Compressions Standards". Kluwer Academic Publisher, 1997

151. V.R.Dragomir student, A.Valea student, C.Bratu : "Proiect – Metoda grilei neomogene şi metoda grafului vizibilităţii". Faculatea de Automatică şi Calculatoare

152. W.D.Schwaderer : "Digital Imaging in C and the World Wide Web". Wordware Publishing, 1997

153. W.K.Pratt : "Digital Image Processing". John Wiley & Sons, 1991

154. Watkins, Cristopher D., and Stevens, Roger T. "Advanced Graphics Programming in C and C++". M&T Publishing, San Mateo, CA, 1991

155. William K. Pratt: "Digital Image Processing". John Willey & Sons, New York, 1978

156. Wolberg, George : "Digital Image Warping". IEEE Computer Society Press Monograph, Los Alamitos, CA, 1990

157. Y.Fisher : "Fractal Image Compression". Springer-Verlag Ney York, 1994