8. SINCRONIZAREA MIŞCĂRILOR ROBOȚILOR ÎN TASK-URI COOPERATIVE

8.1 INTRODUCERE

Integrarea roboților industriali pentru obținerea unor sisteme multi-robot implicate în task-uri colective reprezintă tendințele actuale în industrie şi sistemele de producție. Astfel de sisteme sunt de interes din diferite motive:

- Task-urile pot fi prea complexe (sau chiar imposibile) pentru ca un singur robot să le poată realiza, sau se pot obține beneficii de performanță din utilizarea de roboți multipli;
- Utilizarea mai multor roboți mai simpli poate fi mai ieftină, uşoară, flexibilă şi tolerantă la defect decât având un singur robot mai puternic pentru fiecare task separat.

Manipulatoarele robot (cât şi roboții cooperativi), pot găsi, în prezent multe arii de aplicabilitate. Multe beneficii pot fi obținute prin utilizarea lor în producția industrială. Un exemplu tipic în asamblarea flexibilă este acela în care roboții asambleaza două componente formând un singur produs.

Roboții ce lucrează într-o manieră cooperativă pot fi de asemenea folosiți în manipularea materialelor, e.g., transportarea de obiecte a căror greutate depăşeşte capacitatea de transport a unui singur robot. Mai mult, folosirea acestor roboți permite îmbunătățirea calității task-urilor de producție ce necesită o precizie mare.

Această lucrare de laborator este bazată pe doi roboți SCARA de tipul Adept Cobra s600 şi 600TT (Figura 1). Roboții sunt utilizați într-un task de mişcare cooperativă (manipularea de obiecte). Soluția este bazată pe sincronizarea mişcărilor folosind comunicația TCP/IP pe Ethernet şi comunicația folosind liniile de I/E.

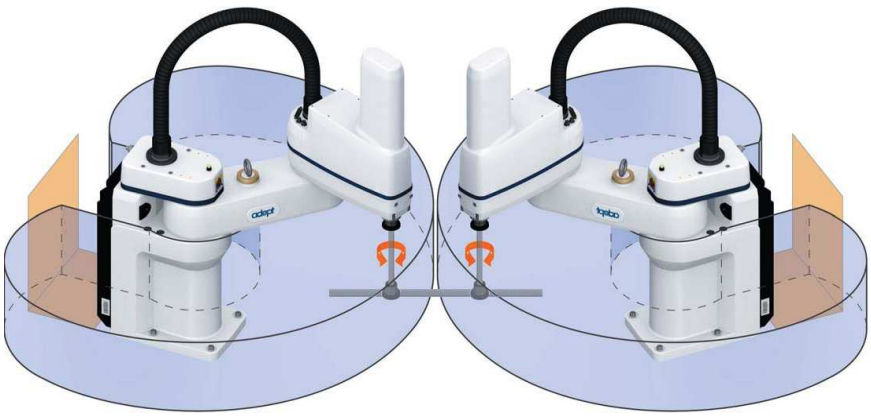

Figura1. Un exemplu de task robot cooperativ.

8.2 SEGMENTE VIRTUALE

Task-urile de manipulare a obiectelor necesită un control precis al forţelor interne. Modelul forţelor interne asociate cu o manipulare cu prinderi multiple (mai mulţi roboţi) poate fi dat de modelul segmentelor virtuale. În acest model, punctele de prindere sunt conectate de un set închis de segmente virtuale ce nu se intersectează.

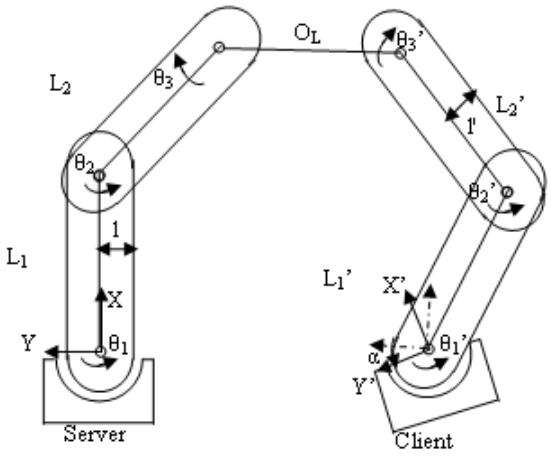

Figura 2. Segmentele virtuale.

Aceeaşi abordare este utilizată şi în aceasta lucrare de laborator, dar în cazul nostru roboţii se găsesc într-o relaţie client-server (sau master-slave) (Figura 2). Pe baza poziţiei spaţiale şi a orientării roboţilor, şi folosind modelele cinematice ale roboţilor, robotul server (sau master) calculeaza traiectoria pentru ambii roboţi şi clientul doar executa mişcările într-o manieră sincronizată. Folosind această abordare, roboţii pot fi controlaţi fără senzorii de forţă deoarece mişcările sunt compuse din segmente mici de mişcări liniare sincronizate folosind liniile de I/E.

Poziţia şi orientarea bazei robotului Client (relativ la sistemul de coordonate World al robotului server) ($X^{'}, Y^{'}, Z^{'}$ şi α) şi dimensiunile obiectului manipulat sunt bine cunoscute, de asemenea poziţiile de prindere sunt învăţate cu precizie.

Utilizând doar mişcări liniare, roboţii au trei tipuri de mişcări cooperative exprimate relativ la obiectul manipulat:

- Translaţii (în acest mod ambii roboţi se mişcă în aceeaşi direcţie utilizând aceeaşi viteză şi realizând un segment de mişcare egal);
- Rotaţii (aici roboţii au diferite direcţii de mişcare, viteze diferite şi realizează segmente de mişcare diferite). În acest caz roboţii încep şi încheie mişcarea în acelaşi timp, astfel că timpul de mişcare este acelaşi în ciuda faptului că distanţele parcurse sunt diferite;
- Mişcări compuse din cele două tipuri de mişcări descrise anterior

In primul caz problema este foarte simplă:

Un segment de mişcare Δx pe axa X a robotului server implică o mişcare a robotului client ce exte compusă din mişcări pe ambele axe:

$$\Delta x \rightarrow \begin{cases} \Delta x' = \Delta x \cos(\alpha) \\ \Delta y' = -\Delta x \sin(\alpha) \end{cases} \qquad (1)$$

Iar un segment de mişcare Δy pe axa Y generează:

$$\Delta y \rightarrow \begin{cases} \Delta x' = -\Delta y \sin(\alpha) \\ \Delta y' = \Delta y \cos(\alpha) \end{cases} \qquad (2)$$

În al doilea caz, rotațiile pot fi executate în jurul oricărui punct ce se găseşte între punctele de prindere (inclus în segmentul O_L) şi nu numai. Notăm cu x punctul în care se va face rotația (ales pe O_L), atunci segmentul O_L este împărțit în două segmente $l_{x_1} = dist(G_S, x)$ şi $l_{x_2} = dist(x, G_C)$ unde G_S şi G_C sunt punctele de prindere ale roboților relativ la obiect: $O_L = dist(G_S, G_C)$. Dacă obiectul trebuie să fie rotit cu $\Delta\theta_L$ în sens trigonometric această rotație va fi generată de o rotație θ_{4_S} şi o mişcare P_{x_S}, P_{y_S} a robotului server:

$$\Delta\theta_L \rightarrow \begin{cases} P_{x_S} = P_{x_{Si}} - l_{x_1} \sin(\Delta\theta_L) \\ P_{y_S} = P_{y_{Si}} + l_{x_1} \cos(\Delta\theta_L) \\ \theta_{4_S} = \theta_{4_{Si}} - \Delta\theta_L \end{cases} \qquad (3)$$

Unde $P_{x_{Si}}$, $P_{Y_{Si}}$ şi $\theta_{4_{Si}}$ este poziția inițială a robotului server şi rotația segmentului 4.

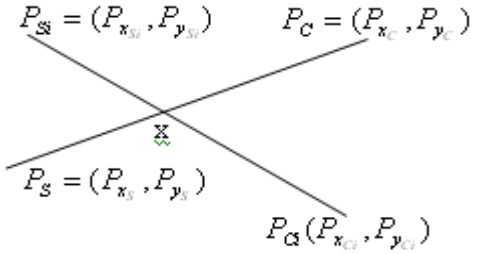

Figura 3. Punctele inițiale şi finale pentru rotația segmentului O_L.

În triunghiurile $\Delta(x, P_S, P_{Si})$ şi $\Delta(x, P_C, P_{Ci})$ (Figura 3) avem relația:

$$\frac{l_{x_1}}{l_{x_2}} = \frac{\Delta x_S}{\Delta x_C} = \frac{\Delta y_S}{\Delta y_C} \qquad (4)$$

unde

$$\begin{cases} \Delta x_S = P_{x_S} - P_{x_{Si}} \\ \Delta y_S = P_{y_S} - P_{y_{Si}} \end{cases}$$

şi (5)

$$\begin{cases} \Delta x_C = P_{x_C} - P_{x_{Ci}} \\ \Delta y_C = P_{y_C} - P_{y_{Ci}} \end{cases}$$

Din (4) şi (5) rezultă:

$$\Delta\theta_L \rightarrow \begin{cases} P_{x_C} = \dfrac{l_{x_1} P_{x_{Ci}} + (P_{x_S} - P_{x_{Si}}) l_{x_2}}{l_{x_1}} \\ P_{y_C} = \dfrac{l_{x_1} P_{y_{Ci}} + (P_{y_S} - P_{y_{Si}}) l_{x_2}}{l_{x_1}} \end{cases} \quad (6)$$

Dar P_{x_C} şi P_{Y_C} sunt exprimate în sistemul de coordonate al robotului server, pentru a calcula coordonatele corecte se inlocuieşte (5) şi (4) în (1) şi (2) rezultând:

$$\Delta\theta_L \rightarrow \begin{cases} P_{x'_C} = P_{x'_{Ci}} - l_{x_2} \sin(\Delta\theta_L + \alpha) \\ P_{y'_C} = P_{y'_{Ci}} + l_{x_2} \cos(\Delta\theta_L - \alpha) \end{cases} \quad (7)$$

Rotația segmentului 4 este:

$$\theta_{4_C} = \theta_{4_{Ci}} - \Delta\theta_L \quad (8)$$

8.3 O ALTERNATIVĂ LA CONTROLUL DESCENTRALIZAT

Abordarea „force control” este de a utiliza o schemă de control cu o arhitectură descentralizată, astfel că momentele de intrare pentru fiecare robot sunt calculate în propriul spațiu al articulațiilor şi ia în considerare constrângerile de mişcare decât să țină cont de dinamica obiectului manipulat.

În cazul unui sistem multi-robot, fiecare robot are acces în timp real doar la propria informație de stare şi poate oferi informații despre forțele celorlalți roboți prin acțiunea lor combinată asupra obiectului. În structura de control descentralizată, specificațiile la nivel de obiect despre task sunt transformate în task-uri individuale pentru fiecare dintre roboții cooperativi. Buclele de control de feedback local sunt dezvoltate la nivelul fiecărui punct de prindere. Transformarea task-urilor şi proiectarea controllerelor locale sunt realizate în concordanță cu modelul segmentelor virtuale. Structura de ansamblu a controlului descentralizat este prezentată în Figura 4, unde F_{force} şi F_{int} sunt forța de intrare şi forțele interne de intrare, f_i şi $f_{s,i}$ sunt forțele din punctul de prindere i şi forțele detectate la punctul de prindere i.

Structura de control prezentată va funcționa corect dacă obiectul este rigid şi dacă nu apar alunecări în zona punctelor de prindere. Alunecarea gripper-ului într-

un sistem real va genera erori în calculul cinematic şi inconsistenţe în modelul segmentelor virtuale.

Pentru a compensa aceste efecte, un anumit nivel de comunicaţie între roboţi este necesar pentru actualizarea stării roboţilor şi modificarea specificaţiilor task-urilor. Rata la care această comunicaţie este cerută este mult mai mică decât rata de control servo. O astfel de comunicaţie poate fi realizată chiar şi la viteze de 10-20 Hz, folosind o linie serială.

În cazul nostru, vom utiliza comunicaţia pentru a deplasa obiectele într-o manieră sincronizată. Această abordare este bazată pe cunoaşterea poziţiei iniţiale a roboţilor, dimensiunile şi poziţia finală dorită a obiectului manipulat.

Structura de control este bazată pe arhitectura client-server. Aici unul dintre roboţi (serverul) cunoaşte toate datele iniţiale, de asemenea cunoaşte şi modelul cinematic al robotului client şi calculeaza off-line calea pe care robotul client trebuie să o urmărească. Calea este împărţită în segmente mici de mişcare iar roboţii sincronizează mişcările utilizând liniile de I/E.

Acest lucru permite rezolvarea următoarelor probleme: nu este necesar ca obiectul să fie rigid (nu se folosesc senzori de forţă), nu apar alunecări la punctele de prindere deoarece se folosesc segmente mici de mişcare sincronizate. De asemenea, datorită faptului că toate calculele (calculul traiectoriei) sunt executate off-line (când roboţii nu au început mişcarea) şi programele de mişcare sunt aproape identice (singura diferenţă o reprezintă valorile stocate în varibilele de tip locaţie), o sincronizare eronată datorată timpului de execuţie al instrucţiunilor este evitată.

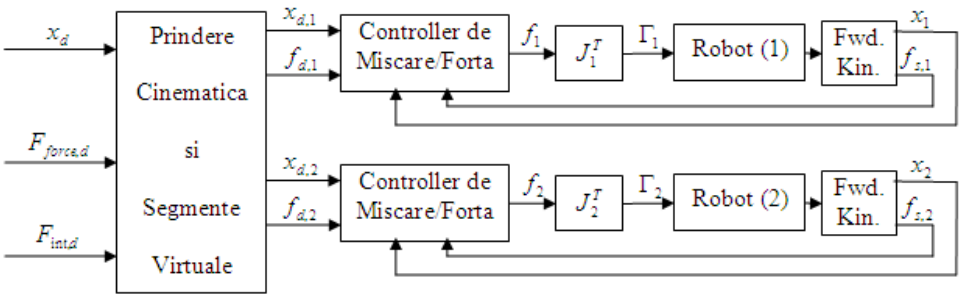

Figura 4. Structura de control descentralizată.

8.4 COMUNICAȚIA

8.4.1 INTERACȚIUNE PRIN MEDIU

Cel mai simplu şi mai limitat tip de interacțiune apare atunci când mediul însuşi devine mediu de comunicație, şi nu apare o comunicație explicită sau interacțiune între roboți. Această modalitate de interacțiune se mai numeşte şi “cooperare fără comunicație”.

Acest tip de comunicație se pretează pentru roboții cu senzori de forță ce sunt conectați prin segmente virtuale utilizând obiectele manipulate. În cazul manipulării cooperative datorită problemelor prezentate mai sus (alunecarea gripper-ului, manipulare obiectelor ce nu sunt rigide) o conexiune de comunicație Serială/Ethernet este de asemenea necesară.

8.4.2 INTERACȚIUNE PRIN COMUNICAȚIE

Această formă de interacțiune implică o comunicație explicită. Abordarea utilizată este de a folosi doar comunicația Ethernet şi I/E pentru a sincroniza mişcările roboților.

În acest caz în care un obiect este manipulat de doi roboți, primul robot calculează offline un set de puncte pe care al doilea robot le urmăreşte, aici, înainte ca mişcarea să înceapă, roboții realizează o cumunicație TCP/IP pentru a trimite/recepționa setul de puncte, în etapa online mişcările sunt sincronizate prin activarea/dezactivarea unei linii de I/E pentru a semnala începerea/terminarea fiecărei mişcări.

8.4.2.1 BAZELE COMUNICAȚIEI PE LINII DE I/E

Unități logice

Toate operațiile de comunicație au ca referință o valoare întreagă numită Număr de Unitate Logica sau LUN. LUN oferă o cale scurtă pentru a identifica ce dispozitiv sau fişier este referit de o operație de I/E.

Verificarea execuției instrucțiunilor de I/E

Spre deosebire de majoritatea instrucțiunilor V+, operațiile de I/E pot să nu se execute corect în anumite cazuri. De exemplu, când se citeşte un fişier, un status este returnat programului pentru a indica dacă s-a ajuns la sfârşitul fişierului. Programul trebuie să trateze acest status şi să continue execuția. În mod similar, o linie serială poate returna un status de eroare pentru o eroare de paritate ce ar trebui să determine programul să reia secvența de transmisie a datelor.

Din acest motiv, instrucțiunile V+ de I/E nu opresc execuția programului când apare o eroare. În schimb, starea execuției instrucțiunii este salvată intern pentru a putea fi accesată de funcția IOSTAT. De exemplu execuția funcției IOSTAT(5) va returna o valoare indicând starea ultimei operații de I/E asupra LUN 5. Valorile returnate de IOSTAT se încadreaza într-una din categoriile:

Valoare	Descriere
1	Operația de I/E s-a efectuat cu succes
0	Operația de I/E încă nu s-a terminat, acest cod apare de obicei în cazul în care se realizeaza o operatie de I/E fără aşteptare
<0	Operația de I/E s-a terminat cu o eroare, codul indică tipul de eroare ce a aparut

Tabelul 1. Valorile returnate de IOSTAT.

În cazul în care valoarea returnată de IOSTAT este mai mică decât 0, atunci funcția $ERROR poate fi folosită pentru a genera mesajul de eroare asociat cu majoritatea erorilor de I/E.

Pentru o bună practică este bine ca IOSTAT să fie folosită după fiecare operație de I/E. În cazul în care se foloseşte funcția GETC nu mai este necesară folosirea funcției IOSTAT deoarece erorile sunt returnate direct de catre GETC.

Ataşarea/Detaşarea Unităților Logice

În general, un dispozitiv de I/E trebuie să fie ataşat folosind instrucțiunea ATTACH înainte ca acesta să poată fi accesat de program. Odată ce un dispozitiv a fost ataşat de un program, dispozitivul respectiv nu mai poate fi folosit de un alt program. Majoritatea operațiilor de I/E vor genera erori dacă dispozitivul asociat cu LUN-ul referit nu este ataşat.

Fiecare task program are propriul set de unități logice. Astfel că mai multe programe pot ataşa acelaşi număr de unitate logică în acelaşi timp fără să apară interferențe.

Un tip de dispozitiv fizic poate fi specificat când unitatea logică este ataşată. Dacă un tip de dispozitiv este specificat, suprascrie dispozitivul predefinit, însă doar pentru unitatea logică ataşată. Tipul de dispozitiv specificat rămâne selectat până când unitatea logică este detaşată.

O cerere de ataşare poate specifica opțional modul imediat. În mod normal, o cerere de ataşare este plasată într-o coadă, iar programul apelant este suspendat dacă un alt program are dispozitivul ataşat. Când dispozitivul este detaşat, următoarea cerere din coadă va fi procesată. În modul imediat, instrucțiunea ATTACH se execută imediat, şi se termină cu o eroare dacă dispozitivul este deja ataşat de un alt program.

În V+, cererile de ataşare pot să specifice de asemenea modul no-wait. Acest mod permite ca o cerere de ataşare să fie introdusă în coadă fără a forța programul să aştepte ca aceasta să fie tratată. În acest caz trebuie folosită funcția IOSTAT pentru a determina când s-a realizat ataşarea.

Dacă un task este deja ataşat la o unitate logică, va primi o eroare dacă va încerca să ataşeze din nou fără ca în prealabil să detaşeze un dispozitiv fără a se ține cont de tipul de aşteptare specificat.

Când un program a încetat a folosi un dispozitiv, trebuie să detaşeze acel dispozitiv folosind instrucțiunea DETACH ceea ce va permite celorlalte programe să proceseze operațiile de I/E cu acel dispozitiv.

Când un program îşi încheie execuţia normal, toate dispozitivele de I/E ataşate sunt automat detaşate. Dacă un program se opreşte anormal, majoritatea dispozitivelor ataşate rămân ataşate. Dacă progranul este reluat şi se reîncearcă ataşarea acestor dispozitive, se vor genera erori datorate faptului că ataşările sunt încă în uz. Comanda monitor KILL forţează un program să detaşeze toate dispozitivele ce au fost anterior ataşate.

Citirea de la un dispozitiv de I/E

Instrucţiunea READ procesează intrările de la toate dispozitivele. Instrucţiunea de bază READ face o cerere către dispozitivul ataşat la LUN-ul indicat şi aşteaptă până când o înregistrare completă de date este recepţionată apoi execuţia programului continuă. (Lungimea ultimei înregistrări citite poate fi obţinută cu funcţia IOSTAT cu al doilea argument egal cu 2.)

Funcţia GETC returnează următorul byte de date de la un dispozitiv de I/E fără să aştepte o înregistrare completă de date. Acestă funcţie este folosită de obicei pentru citirea datelor de la linia serială sau de la terminal. De asemenea poate fi folosită pentru a citi datele dintr-un fisier byte cu byte.

Scrierea la un dispozitiv de I/E

Instrucţiunea WRITE procesează ieşirile către un dispozitiv de I/E. Instrucţiunea WRITE face o cerere către dispozitivul ataşat la LUN-ul indicat, şi aşteaptă până când datele de ieşire au fost scrise, apoi continuă execuţia programului.

Moduri de aşteptare pentru Intrări

În mod normal, V+ aşteaptă până când datele sunt disponibile pentru o instrucţiune de intrare după care continuă execuţia programului. Totuşi instrucţiunea READ şi funcţia GETC acceptă un argument opţional ce specifică un mod fără aşteptare. În modul fără aşteptare, aceste instrucţiuni returnează imediat codul de eroare –526 (No data received) dacă nu sunt date disponibile. Un program poate funcţiona în buclă şi poate folosi aceste operaţii în mod repetat până când se realizează citirea datelor sau până când se recepţionează o anumită eroare.

Moduri de aşteptare pentru Ieşiri

În mod normal, V+ aşteaptă ca fiecare operaţie de I/O să se termine înainte de a trece la următoarea instrucţiune din program. În mod similar instrucţiunile WRITE la liniile seriale vor aştepta ca datele de ieşire să fie scrise înainte de a continua. Această aşteptare nu se realizează dacă formatul de control /N (no wait) este specificat în instrucţiune. În schimb V+ execută imediat următoarea instrucţiune. Funcţia IOSTAT va verifica dacă datele de ieşire au fost scrise şi va returna o valoare de 0 dacă scrierea datelor nu s-a terminat.

Dacă o a doua instrucţiune de ieşire pentru un LUN particular a fost întâlnită inainte ca prima operaţie fără aşteptare să se termine, a doua instrucţiune va aştepta automat până când se va termina prima.

Configurarea I/E pentru liniile seriale

În plus față de selectarea protocolului ce va fi utilizat, programul de configurare al controller-ului permite definirea ratei de transmisie şi formatul de biți pentru fiecare linie serială. Odată ce configurația liniei seriale a fost definită şi sistemul este alimentat, comanda FSET poate fi utilizată pentru a reconfigura temporar liniile seriale. Următoarele formate de biți sunt disponibile:

- Biți de date: 7 sau 8, fără a include biții de paritate
- Unul sau doi biti de stop
- Paritatea activată sau dezactivată
- Paritate impară sau pară

Următoarele rate de biți sunt disponibile:
110, 300, 600, 1200, 2400, 4800, 7200, 9600, 19200, 38400

Exemplu de program ce foloseşte linia serială

Exemplul ataşează prima linie serială şi realizează operații simple WRITE şi READ:

```
.PROGRAM seriala()
      AUTO slun ; Unitate Logica pentru comunicatia pe portul
                ;serial
      AUTO $text
      ;Se face atasarea unitatii logice
      ATTACH (slun, 4) "SERIAL:1"
      IF IOSTAT(slun) < 0 GOTO 100
      ; Se scrie un mesaj pe linia seriala
      WRITE (slun) "Mesaj trimis pe linia seriala 1."
      IF IOSTAT(slun) < 0 GOTO 100
      ;Se citeste o linie de text ce trebuie sa fie incheiata
      ;cu CR/LF
      READ (slun) $text
      IF IOSTAT(slun) < 0 GOTO 100
      TYPE $text
      ;Se afiseaza erorile
   100      IF IOSTAT(slun) < 0 THEN
           TYPE IOSTAT(slun), " ", $ERROR(IOSTAT(slun))
      END
      DETACH (slun) ;Se detaseaza unitatea logica
.END
```

Exercițiu

Folosind scheletul de program de mai sus şi funcțiile (INRANGE, POS, $MID, VAL), realizați un program ce aşteaptă pe linia serială o înregistrare de tipul „x,y,z,” şi va deplasa robotul într-o poziție ce se găseşte la distanța x, y, z, pe axele X, Y, Z (sistem de coordonate World) față de poziția precedentă.

```
.PROGRAM move.serial()
      AUTO slun  ;Unitate Logica pentru comunicatia pe portul
                 ;serial
```

```
      AUTO $text, $xsir, $ysir, $zsir
      AUTO x, y, z, del1, del2, del3
      ;Se face atasarea unitatii logice
      ATTACH (slun, 4) "SERIAL:1"
      IF IOSTAT(slun) < 0 GOTO 100
      ;Se cere o inregistrare
      WRITE (slun) "Introduceti coordonatele sub forma:
x,y,z,"
      IF IOSTAT(slun) < 0 GOTO 100
      ;Se citeste o linie de text ce trebuie sa fie incheiata
cu CR/LF
      READ (slun) $text
      IF IOSTAT(slun) < 0 GOTO 100
      IF $text == "quit" GOTO 110
      del1 = POS($text,",",1)
      del2 = POS($text,",",del1+1)
      del3 = POS($text,",",del2+1)
      $xsir = $MID($text,1,del1-1)
      $ysir = $MID($text,del1+1,del2-del1-1)
      $zsir = $MID($text,del2+1,del3-del2-1)
      x = VAL($xsir)
      y = VAL($ysir)
      z = VAL($zsir)
      IF NOT INRANGE(SHIFT(HERE BY x,y,z)) THEN
          MOVE SHIFT(HERE BY x,y,z)
      END
      ;Se afiseaza erorile
   100      IF (IOSTAT(slun) < 0) THEN
          TYPE IOSTAT(slun), " ", $ERROR(IOSTAT(slun))
      END
   110      DETACH (slun) ; Se detaseaza unitatea logica

.END
```

8.4.2.2 MODELUL SOFTWARE ADEPTNET

Modelul Software AdeptNet descrie arhitectura software a rețelei Adept.

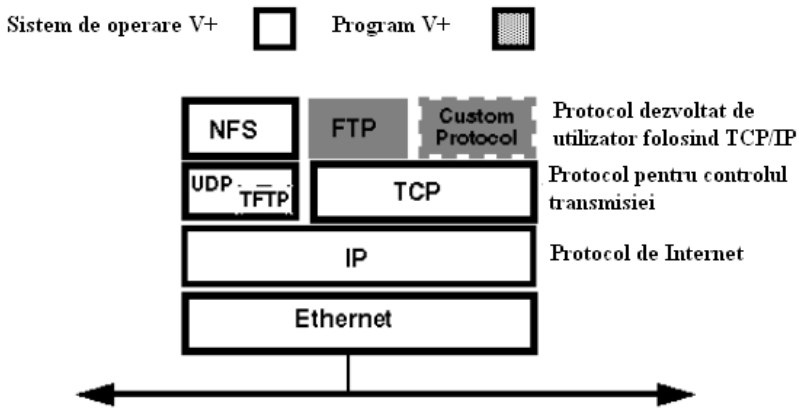

Figura 5. Modelul SoftwareAdeptNET.

În acest model stratifcat, fiecare nivel software foloseşte serviciile furnizate de nivelul inferior. Nivelul inferior este reprezentat de nivelul hardware Ethernet. Acest nivel defineşte conectivitatea fizică a rețelei şi software-ul low-level necesar pentru a transmite şi a recepționa mesaje de-a lungul rețelei.

Deasupra acestui nivel se găseşte un protocol standard numit Internet Protocol (IP). Fiecare nod al rețelei are o adresă IP. Adresa IP identifică în mod unic fie nodul retelei sau LAN-ul (reteaua locală) sau WAN-ul. În cazul transmisiei, nivelul IP realizează o conxiune cu nodul destinație dorit. În cazul recepționării unui mesaj, nivelul IP recunoaşte adresa IP a nodului şi apoi acceptă sau respinge mesajul ce urmează a fi recepționat.

Deasupra nivelului IP se alfă două protocoale diferite: UDP şi TCP. Ca funcționalitate, aceste două protocoale produc aceleaşi rezultate: oferă o interfață consistentă către aplicațiile ce necesită acces la rețea. UDP (User Datagram Protocol) oferă un serviciu nesigur, adică livrarea de mesaje nu este garantată. Acest lucru implică simplitate şi o viteză îmbunătățită la efectuarea operațiilor deoarece nodul destinatar al rețelei nu este obligat să confirme că mesajul a fost recepționat. Din acest motiv, UDP este de obicei folosit pentru aplicații ce nu necesită o fiabilitate absolută, dar şi pentru transmisia către mai multe noduri simultan. O aplicație ce foloseste UDP poate la rândul ei să întreprindă o monitorizare a statusului şi o verificare a erorilor.

TCP (Transmision Control Protocol) furnizează un serviciu de tip point-to-point de încredere. TCP a fost proiectat pentru aplicații ce implică transmisia de date prin rețele WAN (Wide Area Network) cum ar fi Internet-ul, unde comunicațiile sigure sunt esențiale. AdeptNet foloseşte UDP/IP sau TCP/IP în funcție de ce protocol este folosit în aplicație.

La nivelul aplicație, Adept oferă trei pachete: sistemul de fişiere al rețelei (NFS), funcționalitatea de transfer al fişierelor (FTP) şi posibilitatea de a realiza protocoale particulare. Protocoalele de pe cel mai înalt nivel în modelul software

folosesc funcţiile asigurate de nivelurile inferioare. În cazul transiterii de date, fiecare nivel ataşează antetul propriu la mesajul deja existent , iar în cazul primirii unui mesaj, fiecare nivel îndepărtează antetul necesar şi trimite ce a rămas din mesaj către nivelul de mai sus.

Scrierea de Protocoale personalizate folosind AdeptTCP/IP

Suportul TCP/IP în controllerele Adept permite programelor V+ să comunice cu alte dispozitive TCP/IP (controllere Adept sau alte echipamente) fiind fie clienți fie servere. În modul client, controller-ul Adept solicită informații de la un nod particular din rețea, numit server. În modul server, controller-ul Adept răspunde cererilor de la unul sau mai multe noduri client.

Porturi şi Socketi

TCP utilizează adresarea de porturi pentru a furniza informația către aplicații sau programe. Un port este o adresă pe 16 biți. Termenul de socket se referă la gruparea dintre adresa IP a nodului şi un număr de port utilizat de aplicație sau program. Conceptul de deschidere a unui socket într-o aplicație facilitează realizarea de sesiuni multiple pe un singur nod în rețea.

De exemplu, două task-uri V+, sau sesiuni, pe acelaşi controller pot comunica simultan cu un server din rețea, deşi ambele task-uri se execută pe acelaşi nod şi au acelaşi IP. Totuşi, datorită folosirii de socketi, combinația dintre IP şi numărul de port identifică unic task-ul la care TCP trebuie să ruteze informația.

Stabilirea conexiunii TCP

Protocolul TCP este divizat în două părți, partea de server şi partea client. Instrucțiunile programului client diferă de cele ale serverului.

Serverul

Pentru a stabili o conexiune de tip server TCP, un task de tip server V+ TCP informează driverul local TCP că doreşte să accepte toate conexiunile pe un anumit port. Oricărui nod client din rețea îi este acum permis să se conecteze la acest server folosind adresa IP a serverului şi portul specificat.

Serverului nu ii este necesar să ştie IP-ul sau portul clientului ce s-a conectat, în schimb clientul trebuie să ştie IP-ul serverului. Atunci când un client se conecteaza la server, clientului îi este atribuit un control de monitorizare (handle) pe server. Din acest moment, numărul controlului va fi folosit pentru a identifica clientul. Atunci când clientul se deconectează de la server, controlul este eliberat şi este folosit de orice alt client ce doreşte să se conecteze la server.

De multe ori este nevoie de un sistem client-server singular, adică un singur client se conecteaza la un singur server, o astfel de comunicație se numeste comunicație de tip peer-to-peer.

Clientul

Pentru a stabili o conexiune, clientul trebuie să ştie adresa IP şi numărul portului deschis de server. Clientul poate alege portul local pe care va realiza comunicația în mod arbitrar, dar în general acesta este ales de către driverul TCP.

Cu ajutorul acestor informații, clientul poate încerca o stabilire a conexiunii transmițând o cerere către server.

Atunci când serverul TCP primeşte această cerere, verifică dacă portul clientului se potriveşte cu cel de pe server. În momentul in care s+au terminat verificările, se retransmite un mesaj către client. Odată ce conexiunea client-server a fost stabilită, atât serverul cât şi clientul pot începe o transmitere bidirecțională de date.

Exemplu de aplicație de comunicatie pe TCP: Echo Server

Următorul program implementează un server echo. Mesajele primite de la clienți sunt trimise înapoi. Un astfel de program poate fi folosit pentru testarea conexiunilor de rețea.

```
.PROGRAM server.echo()
;-------------------------------------------------------------
;Serever ECHO
;Programul poate fi oprit prin trimiterea mesajului "quit"
;-------------------------------------------------------------

      AUTO handle, lun, do_wait, status
      AUTO $in.str, $out.str, repeat_loop
      ;Constante de initializare
      do_wait = 0
      lun = 7

      ;se ataseaza dispozitivul TCP, mode=4 inseamna ca se
      ;aloca
      ;urmatorul LUN disponibil
      ATTACH (lun, 4) "TCP"
      status = IOSTAT(lun) ; Se verifica executia
                           ;instructiunii ATTACH
      IF status < 0 THEN
          TYPE "Eroare la ATTACH:", $ERROR(status)
          GOTO 110
      END

      ;Se deschide un socket pe portul 1234, ce accepta 5
      ;clienti, avand un buffer de 1024 de bytes
      FOPEN (lun, 16) "/LOCAL_PORT 1234 /CLIENTS 5
/BUFFER_SIZE 1024"
      status = IOSTAT(lun) ;Se verifica executia
                           ;instructiunii FOPEN
      IF status < 0 THEN
          TYPE "Eroare la FOPEN:", $ERROR(status)
          GOTO 100
      END
```

```
      repeat_loop = TRUE
      WHILE repeat_loop DO
          WAIT
          READ (lun, handle, do_wait) $in.str
          status = IOSTAT(lun)
          CASE status OF
            VALUE 1: ;Succes - se trimite sirul de caractere
                     ;inapoi
            IF $in.str == "quit" GOTO 100
            $out.str = $in.str
            TYPE "S-a primit sirul: ", $out.str
            WRITE (lun, handle) $out.str, /N
            VALUE 100: ;S-a deschis o noua conexiune
            TYPE "S-a deschis o noua conexiune. Handle=",
handle
            VALUE 101: ;Conexiune inchisa
            TYPE "Conexiune inchisa. Handle=", handle
            FCMND (lun, 600) $INTB(handle) ; Eliberare handle
            status = IOSTAT(lun) ; Se verifica executia
                                 ;instructiunii FCMND
            IF status < 0 THEN
                TYPE "Eroare la FCMND:", $ERROR(status)
                GOTO 100
            END
            WAIT
            VALUE -526: ;Nu s-au receptionat date
            ANY ;Alte erori
            TYPE "Eroare la citire: ", $ERROR(status)
            GOTO 100
          END
      END

      ;Se inchid toate conexiunile, si se elibereaza unitatea
      ;logica
   100      FCLOSE (lun)
      DETACH (lun)
   110      ;Iesire din program
.END
```

Exercițiu

Realizați un program de tip Client/Server pentru schimbul de mesaje între cei doi roboți SCARA. Mesajele vor fi scrise de la terminalul fiecarui robot.

ROBOTUL COBRA S600

```
.PROGRAM s600.com()
;-------------------------------------------------------------
;Program Comunicatie TCP s600
;Programul poate fi oprit prin trimiterea mesajului "quit"
;-------------------------------------------------------------
```

```
      AUTO handle, lun, wait, status
      AUTO $in.str, repeat_loop

      ;Constante de initializare
      wait = 1
      lun = 7
      $mesaj = ""

      ;se ataseaza dispozitivul TCP, mode=4 inseamna ca se
      ;aloca urmatorul LUN disponibil
      ATTACH (lun, 4) "TCP"
      status = IOSTAT(lun) ; Se verifica executia
                           ;instructiunii ATTACH
      IF status < 0 THEN
          TYPE "Eroare la ATTACH:", $ERROR(status)
          GOTO 110
      END

      ;Se deschide un socket pe portul 1234, ce accepta 5
      ;clienti, avand un buffer de 1024 de bytes
      FOPEN   (lun,   16)   "/LOCAL_PORT   1234   /CLIENTS   5
/BUFFER_SIZE 1024"
      status = IOSTAT(lun) ;Se verifica executia
                           ;instructiunii FOPEN
      IF status < 0 THEN
          TYPE "Eroare la FOPEN:", $ERROR(status)
          GOTO 100
      END

      repeat_loop = TRUE
      WHILE repeat_loop DO
          WAIT
          IF $mesaj == "quit" GOTO 100
          IF $mesaj <> "" THEN
            WRITE (lun, handle) $mesaj, /N
            IF IOSTAT(lun)<0 THEN
                  TYPE   "Eroare   de      comunicatie:   ",
$ERROR(IOSTAT(lun))
                  GOTO 100
            END
            $mesaj = ""
          END
          READ (lun, handle, wait) $in.str
          status = IOSTAT(lun)
          CASE status OF
            VALUE 1: ;Succes - se afiseaza sirul de caractere
            TYPE "S-a primit de la 600TT sirul: ", $in.str
            VALUE 100: ;S-a deschis o noua conexiune
            TYPE  "S-a  deschis  o  noua  conexiune.  Handle=",
handle
            VALUE 101: ;Conexiune inchisa
            TYPE "Conexiune inchisa. Handle=", handle
```

```
            FCMND (lun, 600) $INTB(handle) ; Eliberare handle
            status = IOSTAT(lun) ; Se verifica executia
                                 ;instructiunii FCMND
            IF status < 0 THEN
                TYPE "Eroare la FCMND:", $ERROR(status)
                GOTO 100
            END
            WAIT
            VALUE -526: ;Nu s-au receptionat date
            ANY ;Alte erori
            TYPE "Eroare la citire: ", $ERROR(status)
            GOTO 100
          END
      END

      ;Se inchid toate conexiunile, si se elibereaza unitatea
      ;logica
   100      FCLOSE (lun)
      DETACH (lun)
   110      ;Iesire din program
.END

.PROGRAM send.message()
      WHILE TRUE DO
          PROMPT "Mesaj de trimis pt 600TT:", $mesaj
          IF $mesaj == "quit" GOTO 100
      END
   100
.END
```

ROBOTUL COBRA 600TT

```
.PROGRAM tt600.com()
;-------------------------------------------------------------
;Program de comunicatie TCP 600TT
;Programul poate fi oprit prin trimiterea mesajului "quit"
;-------------------------------------------------------------

      AUTO lun, wait
      AUTO $read_str

      ;Constante de initializare
      wait = 1
      $mesaj = ""

      ;se ataseaza dispozitivul TCP, mode=4 inseamna ca se
      ;aloca
      ;urmatorul LUN disponibil
      ATTACH (lun, 4) "TCP"
      status = IOSTAT(lun) ; Se verifica executia
                           ;instructiunii ATTACH
      IF status < 0 THEN
          TYPE "Eroare la ATTACH:", $ERROR(status)
```

```
        GOTO 110
      END

      FOPEN  (lun,  0)  "s600  /REMOTE_PORT  1234  /BUFFER_SIZE
1024"
      status = IOSTAT(lun) ;Se verifica executia
                           ;instructiunii FOPEN
      IF status < 0 THEN
          TYPE "Eroare la FOPEN:", $ERROR(status)
          GOTO 100
      END

      ;mlun=4
      ;ATTACH(mlun)
      ;FOPEN(mlun)

      repeat_loop = TRUE
      WHILE repeat_loop DO
          WAIT
          IF $mesaj == "quit" GOTO 100
          IF $mesaj <> "" THEN
            WRITE (lun) $mesaj
            IF IOSTAT(lun)<0 THEN
                  TYPE    "Eroare    de    comunicatie:    ",
$ERROR(IOSTAT(lun))
                  GOTO 100
            END
            $mesaj = ""
          END
          READ (lun, wait) $read_str
            IF IOSTAT(lun)<0 THEN
                  TYPE    "Eroare    de    comunicatie:    ",
$ERROR(IOSTAT(lun))
                  GOTO 100
            END
          TYPE "Mesaj de la s600: ", $read_str
      END

      ;Se inchid toate conexiunile, si se elibereaza unitatea
      ;logica
   100      FCLOSE (lun)
      DETACH (lun)
   110      ;Iesire din program
.END

.PROGRAM send.message()
      WHILE TRUE DO
          PROMPT "Mesaj de trimis la s600:", $mesaj
          IF $mesaj == "quit" GOTO 100
      END
   100
.END
```

8.5 APLICAȚIE DE MANIPULARE A UNUI OBIECT FOLOSIND DOI ROBOȚI INDUSTRIALI CU MIŞCĂRI SINCRONIZATE

Se doreşte deplasarea unui obiect folosind doi roboți SCARA. În sensul celor prezentate în preambulul teroretic. Se cunosc pozițiile safe şi p1 (poziția de prindere a obiectului). Robotul Cobra s600 va creea un server TCP şi va trimite coordonatele (calculate în prealabil şi stocate în vectorul $coord[], respectiv pos[]) ce vor descrie mişcarea, către robotul 600TT. După ce coordonatele sunt recepționate se poate executa mişcarea ce este sincronizată folosind liniile de I/E.

Figura 6. Aplicație de manipulare folosind mişcări robot sincronizate.

ROBOTUL COBRA S600

```
.PROGRAM sincro()
;-------------------------------------------------------------
;Program miscare sincronizata Cobra s600
;-------------------------------------------------------------

      AUTO handle, lun, do_wait, status
      AUTO $in.str, semnal1, semnal2

      ;Constante de initializare
      do_wait = 0
      lun = 7
      semnal1 = 1009
      semnal2 = 1

      PARAMETER HAND.TIME = 0.5
      SPEED 100 ALWAYS
      RIGHTY
      MOVET safe, 1
      APPRO p1, 80
      BREAK
      MOVES p1
      CLOSEI
```

```
      ;se ataseaza dispozitivul TCP, mode=4 inseamna ca se
      ;aloca
      ;urmatorul LUN disponibil
      ATTACH (lun, 4) "TCP"
      status = IOSTAT(lun) ; Se verifica executia
                           ;instructiunii ATTACH
      IF status < 0 THEN
          TYPE "Eroare la ATTACH:", $ERROR(status)
          GOTO 110
      END

      ;Se deschide un socket pe portul 1234, ce accepta 1
      ;client, avand un buffer de 1024 de bytes
      FOPEN   (lun,   16)   "/LOCAL_PORT   1234   /CLIENTS   1
/BUFFER_SIZE 1024"
      status = IOSTAT(lun) ;Se verifica executia
                            ;instructiunii FOPEN
      IF status < 0 THEN
          TYPE "Eroare la FOPEN:", $ERROR(status)
          GOTO 100
      END

      TYPE   "Se   asteapta   conectarea   unui   robot   pentru
trimiterea coordonatelor"
    10      READ (lun, handle, do_wait) $in.str
      IF IOSTAT(lun) <> 100 GOTO 10

      TYPE "Se trimit coordonatele"
      WRITE (lun, handle) $ENCODE(n)
      IF IOSTAT(lun) < 0 GOTO 100
      READ (lun, handle, do_wait) $in.str
      IF (IOSTAT(lun) < 0) OR (VAL($in.str) <> n) GOTO 100

      FOR i = 0 TO n
          WRITE (lun, handle) $coord[i]
          IF IOSTAT(lun) < 0 GOTO 100
          READ (lun, handle, do_wait) $in.str
          IF   (IOSTAT(lun)   <   0)   OR   ($MID($in.str,1,
LEN($in.str)-2) <> $coord[i]) GOTO 100
      END
      FCLOSE (lun)
      DETACH (lun)

      DURATION 1 ALWAYS
      FOR i = 0 TO n
          WAIT SIG(semnal1)
          semnal1 = -1*semnal1
          MOVES pos[i]
          SIGNAL semnal2
          semnal2= -1*semnal2
```

```
      END
      OPENI
      DEPARTS 80
      GOTO 110

      ;Se inchid toate conexiunile, si se elibereaza unitatea
      ;logica
   100      FCLOSE (lun)
      DETACH (lun)
   110      ;Iesire din program
.END
```

ROBOTUL COBRA 600TT

```
.PROGRAM sincro()
;-------------------------------------------------------------
;Program miscare sincronizata Cobra 600TT
;-------------------------------------------------------------

      AUTO lun, do_wait, semnal1, semnal2, x, y, z, del1,
      del2, del3
      AUTO $read_str, $xsir, $ysir, $zsir


      ;Constante de initializare
      do_wait = 0
      semnal1 = 57
      semnal2 = 1048

      PARAMETER HAND.TIME = 0.5
      SPEED 100 ALWAYS
      LEFTY
      MOVET safe, 1
      APPRO p1, 80
      BREAK
      MOVES p1
      CLOSEI


      ;se ataseaza dispozitivul TCP, mode=4 inseamna ca se
      ;aloca
      ;urmatorul LUN disponibil
      ATTACH (lun, 4) "TCP"
      status = IOSTAT(lun) ; Se verifica executia
                           ;instructiunii ATTACH
      IF status < 0 THEN
          TYPE "Eroare la ATTACH:", $ERROR(status)
          GOTO 110
      END

      FOPEN (lun, 0) "s600 /REMOTE_PORT 1234 /BUFFER_SIZE
1024"
```

```
      status = IOSTAT(lun) ;Se verifica executia
                           ;instructiunii FOPEN
      IF status < 0 THEN
          TYPE "Eroare la FOPEN:", $ERROR(status)
          GOTO 100
      END

      TYPE "Se primesc coordonatele"
      READ (lun, do_wait) $read_str
      IF IOSTAT(lun) < 0 GOTO 100
      WRITE (lun) $read_str
      IF IOSTAT(lun) < 0 GOTO 100
      n = VAL($read_str)

      FOR i = 0 TO n
          READ (lun, do_wait) $coord[i]
          IF IOSTAT(lun) < 0 GOTO 100
          del1 = POS($coord[i],",",1)
          del2 = POS($coord[i],",",del1+1)
          del3 = POS($coord[i],",",del2+1)
          WRITE (lun) $MID($coord[i],1,del3)
          IF IOSTAT(lun) < 0 GOTO 100
          $xsir = $MID($coord[i],1,del1-1)
          $ysir = $MID($coord[i],del1+1,del2-del1-1)
          $zsir = $MID($coord[i],del2+1,del3-del2-1)
          x = VAL($xsir)
          y = VAL($ysir)
          z = VAL($zsir)
          SET pos[i] = SHIFT(p1 BY x,y,z)
      END
      FCLOSE (lun)
      DETACH (lun)

      DURATION 1 ALWAYS
      FOR i = 0 TO n
          SIGNAL semnal1
          semnal1 = -1*semnal1
          MOVES pos[i]
          WAIT SIG(semnal2)
          semnal2 = -1*semnal2
      END
      OPENI
      DEPARTS 80
      GOTO 110

      ;Se inchid toate conexiunile, si se elibereaza unitatea
      ;logica
   100      FCLOSE (lun)
      DETACH (lun)
   110      ;Iesire din program
.END
```