Minimizarea timpului de ciclu în mediul industrial

Goța Radu-Cristian
Facultatea de Automatică și Calculatoare
Universitatea Tehnica din Cluj-Napoca
Cluj-Napoca, Romania
radugota6@gmail.com

Abstract - În această lucrare se vor prezenta un sistem complet automatizat format din mai multe stații care realizează diferite procese asupra unei componente, un robot care are rolul de a transporta piesele de la o stație la alta, un computer logic programabil, cu rolul de a coordona întreg procesul și o metodă de îmbunătățire a nivelului producției bazată pe prezicerea următoarei poziții a robotului. Două aplicații vor fi create, una pentru simularea stațiilor cu ajutorul unor timer-e și una pentru coordonarea mișcărilor brațului robotic. Computerul logic programabil ales este Beckhoff CX5140, iar robotul, Stäubli TS2-80. Aplicația pentru plc se va realiza în mediul de programare Twincat 3 folosind limbajele SFC și ST, iar aplicația pentru robot în Stäubli Robotics Suite folosind limbajul VAL3.

I. INTRODUCERE

În mediul industrial se folosesc roboți pentru manipularea diferitelor componente. Controlul acestor roboți se face prin intermediul unui computer logic programabil(engl. Programmable Logic Controller - PLC) care ține evidența tuturor parametrilor și comandă întreg procesul. PLC-ul interacționează cu robotul pe baza unor ordine(engl. Task) pe care le trimite acestuia. Robotul recepționează aceste ordine, execută o secvență de program stabilită după care returnează un *feedback*. Datorită faptului ca plc-ul monitorizează întreg procesul, el reține starea fiecărei stații și a fiecărei piese. De cele mai multe ori, atunci când robotul primește un astfel de ordin, poziția curentă și poziția în care el trebuie să se deplaseze diferă, rezultând un timp mort de așteptare necesar deplasării pentru întreg procesul și automat un timp de ciclu mai mare.

II. CONTROLLER LOGIC PROGRAMABIL BECKHOFF CX5140

Un controler logic programabil este o formă specială de controler bazat pe un microprocesor care stochează instrucțiuni și le aplică ulterior implementând funcții logice, aritmetice etc. în ideea de a controla alte dispozitive. În industrie, plc-ul este unitatea de bază în ceea ce privește controlul. Principalele avantaje ale unui astfel de dispozitiv sunt rezistența în condiții extreme și încorporarea comodă a acestuia împreună cu alte elemente necesare controlului cum ar fi senzori și actuatori.

Arhitectura unui astfel de controler se poate observa în figura 1. El este format dintr-o unitate centrală de control(engl. Central Processing Unit - CPU) care ia datele venite de la intrări, le interpretează în funcție de regulile stabilite și transmite rezultatele ieșirilor, o memorie ce stochează programul scris, un modul de intrare ce colectează datele de la elemente externe și un modul de ieșire prin care se transmit comenzi actuatorilor.

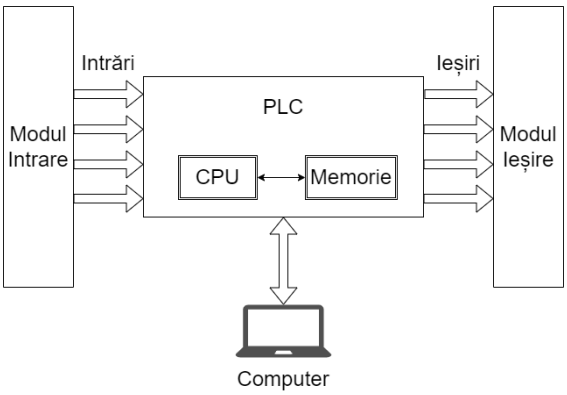

Figure 1: Arhitectura unui plc

Toate modelele acestei familii, CX51x0, au aceeași arhitectură prezentată în figura 2. Unitatea centrală de procesare conține un controler de memorie și unul grafic. Există două interfețe PCIe, USB, DVI, serial, SATA și SD. Controlul comunicației este realizat de două procesoare, unul pentru fiecare port. Interfețele de Ethernet sunt independente și nu există switch-uri integrate. Profilul lor este configurat pentru comunicația EtherCAT. Acest model dispune de o tehnologie numită *1-second UPS* care nu este altceva decât un condensator ce continuă să alimenteze procesorul în momentul unei pane de curent. În acest scurt timp, datele persistente pot fi salvate și menținute până la o repunere în funcțiune.

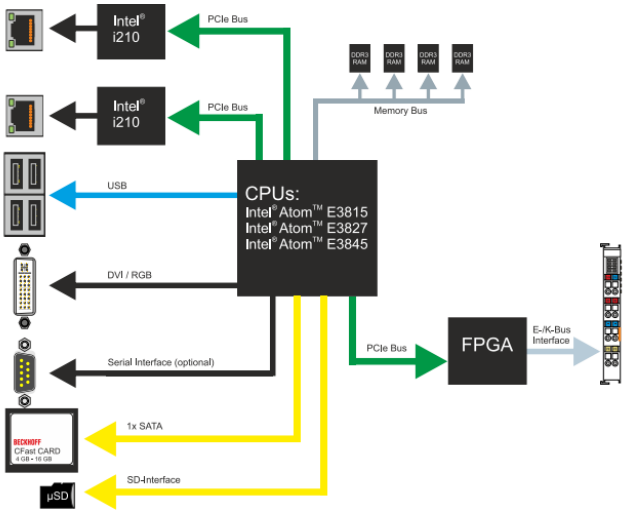

Figure 2: Arhitectura modelului familiei de plc-uri CX51x0 [1]

Programarea unui astfel de plc se face folosind software-ul Twincat 3 ce reprezintă miezul sistemului de control. El se împarte în eXtended Automation Engineering(XAE) și eXtended Automation Runtime(XAR). Este capabil de a

transforma orice calculator personal într-unul cu capacități de timp real. Twincat XAE permite integrarea mai multor tool-uri precum MATLAB, Simulink,C++, iar Twincat XAR oferă un mediu de timp real.

III. ROBOT STÄUBLI TS2-80

Modelul TS2-80 este un braț robotic cilindrid SCARA(Selective Compliance Articulated Robot Arm) folosit în special pentru sarcinile de selectare, ridicare și plasare(engl. Pick and Place) și asamblare. Are o precizie ridicată, fapt pentru care este utilizat preponderent în procesele de fabricație ale circuitelor imprimate(engl. Printed Circuit Board - PCB)

Brațul robotic are nevoie de un controler care este responsabil cu tot ce ține de partea de gândire. Acesta este un computer specializat cu capabilități foarte bune în luarea de decizii și cu o capacitate foarte mare de memorie. Este responsabil pentru activarea elementelor de execuție, pentru efectuarea de calcule complicate atunci când vine vorba de traictorii și viteze, dar și pentru controlul întregului robot, de la preluarea datelor de la senzori, interpretarea lor, luarea de decizii și transmiterea acestora mai departe în sistem.

Programarea unor astfel de roboți se poate face prin mai multe moduri, însă principalul este prin intermediul unui pendant de invățare(engl. Teach Pendant). În domeniu, această practică se numește "teaching". Acest pendant reprezintă legătura dintre operator și memoria robotului. Câteva alte metode sunt prin intermediul mișcării manuale a brațului, prin intermediul unui IDE sau prin intermediul fișierelor CAD [4].

În figura 3 este reprezentat modelul TS2-80. Acesta prezintă trei elemente de rotație și un element de translație. Datorită acestui tip de configurație, poate atinge viteze foarte mari și este preferat în procesele de asamblare.

Figure 3: Brațul robotic TS2-80

IV. PRINCIPIUL DE FUNCȚIONARE AL TEHNOLOGIEI ETHERCAT

Principiul de funcționare al tehnologiei EtherCAT este simplu. Master-ul trimite o telegramă care trece prin fiecare nod legat la rețea. Fiecare dispozitiv de tip slave citește datele care îi sunt adresate în timp ce acestea îl intersectează și își scrie propriile date în cadru(engl. Frame). Dispozitivul de tip master este singurul căruie îi este permis să trimită frame-uri de date, iar nodurile au capabilitatea de a le pasa mai departe către alte noduri precum în figura 4.

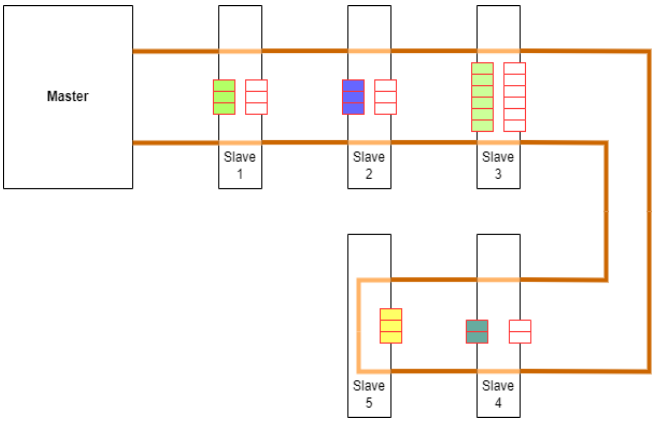

Figure 4: Principiul de funcționare al tehnologiei EtherCAT

Starea fiecărui dispozitiv de tip slave este controlată prin intermediul mașinii de stare. În funție de aceasta, diverse operații sau funcții sunt accesibile sau executabile de către dispozitiv. În figura 5 este descrisă mașina de stare. Se definesc cinci stări: Init, Pre-Operational, Safe-Operational, Operational, Bootstrap(optional), iar permisiunea trecerii dintr-o stare în alta este descrisă prin săgeți.

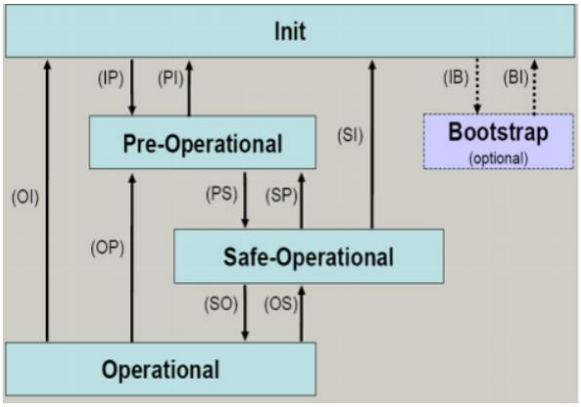

Figure 5: Mașină de stare EtherCAT [2]

După pornire, toate dispozitivele slave trec în starea INIT. În această stare, nu există niciun fel de comunicare. Master-ul inițializează canalele managerului de sincronizare pentru diferitele profiluri de comunicare. În următoarea stare, Pre-Operational, doar comunicația SDO(Service Data Objects) este posibilă, iar master-ul configurează canalele managerului de sincronizare pentru comunicație PDO(Process Data Objects). În același timp, parametrii specifici terminalului, care pot diferi de setările implicite, sunt transferate către master. În timpul tranziției de la Init la Pre-Operational, dispozitivele slave verifică dacă comunicația aciclică a fost inițializată corect. În a treia stare, Safe-Operational, amândouă formele de comunicare sunt posibile. Totuși, dispozitivele slave păstrează ieșirile într-o zonă de siguranță, doar intrările fiind updatate ciclic. În timpul tranziției dintre Pre-Operational și Safe-Operational, dispozitivele slave verifică dacă managerul de sincronizare pentru comunicția PDO și ceasurile distribuite sunt configurate corect. În starea Operational, toate tipurile de comunicare sunt posibile, iar în

Bootstrap, doar comunicația FoE este posibilă, în cazul updatării firmware-ului unui nod slave [2].

V. DESCRIEREA APLICAȚIEI

Ideea aplicației constă în crearea unui proces de pick and place, format dintr-un braț robotic și mai multe stații simulate cu ajutorul unor timer-e. Prin urmare există patru tipuri de stație: Conveyor, Drilling, Test, Metalization. Diagrama fluxului de lucru este reprezentată în figura 6. Componentele sosesc în modulul Conveyor, care mai apoi, sunt transportate la unul dintre modulele de Drilling. După un timp de procesare(diferit în funcție de stație), piesele sunt transportate la unul dintre modulele de Test. La final, fiecare piesă este transportată la modulul Metalization. Transportul pieselor este asigurat de către brațul robotic. În același timp, mai avem nevoie de un modul pentru robot, care este responsabil cu logica din spatele mișcărilor acestuia, dar și cu comunicațiile aferente. Toate aceste stații sunt integrate într-o celulă de lucru.

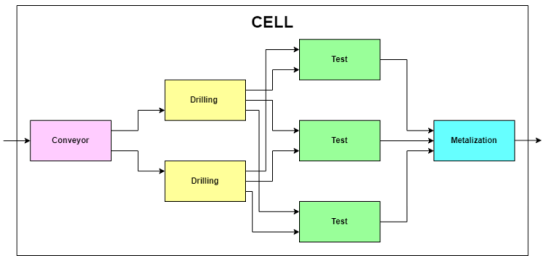

Figure 6: Fluxul de lucru al aplicației

În ceea ce privește partea de logică, fiecare stație în parte are un mod automat, în care acțiunile de procesare ce au loc asupra piesei sunt simulate cu ajutorul unor timer-e. Prin urmare, în momentul în care o piesă ajunge într-o stație de prelucrare, acea stație este semnalată, iar procesul poate începe. Fiecare proces are un timp ce poate fi setat, iar stațiile de test au un timp aleator în intervalul 95-105 secunde. Pentru ca sistemul să nu se blocheze, o primă restricție o reprezintă faptul că robotul nu poate ridica o piesă dintr-un modul dacă următorul modul este ocupat cu alte piese. Ca o piesă să poată fi declarată bună, ea trebuie să treacă printr-o stație de fiecare tip în ordinea corectă.

Algoritmul de decizie, în cazul în care dorim să luăm o piesă dintr-o stație, în vederea transportării acesteia către următoarea stație se bazează pe ordine și este prezentat în mai jos. Pentru celelalte stații procesul este similar, adăugându-se mai multe condiții pentru determinarea scopului deplasării, dacă e pentru a ridica piesa sau pentru a o lăsa.

Algoritmul 1 Calcularea următoarei mișcări

```
1: Intrări: conveyorState, drillingState, pickedOk
2: Ieșiri: nextOrder
3: if conveyorState = Ready
4:         AND drillingState = Empty
5:         AND pickedOk = true
6:         AND nextOrder = noOrder
7:    then
8:         nextOrder ← pickConveyorOrder
9: end if
10: return nextOrder
```

În figura 7 poate fi observat cadrul de lucru aferent aplicației văzut de sus. Aceasta conține limitele de deplasare ale robotului, coordonatele fiecărei stații și traseele de deplasare de la o stație la alta.

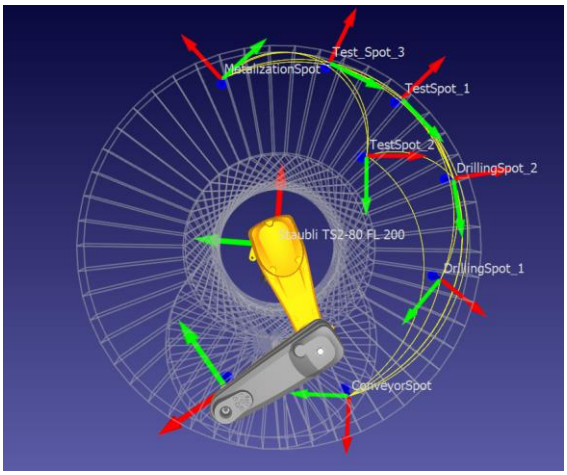

Figure 7: Reprezentarea 2D văzută de sus a spațiului de lucru

VI. OPTIMIZAREA APLICAȚIEI

Primul aspect care a trebuit luat în calcul în ceea ce privește partea de optimizare a fost alegerea limbajului de programare. Deși fiind cel mai rapid datorită asemănării de sintaxă cu codul mașină, Instruction List este un limbaj nerecomandat în standardul IEC-61131 [3]. Următorul limbaj din listă, în ceea ce privește viteza, este Structured Text(ST). Pentru a crea o logică bazată pe mașini de stare (engl. State Machines) se folosește Sequential Function Chart(SFC). Pentru ambele alegeri, codul se execută de sus în jos, de la stânga la dreapta. Acest aspect este important pentru a evita scanarea secțiunilor de cod ce nu sunt executabile. De aceea, într-o diagramă SFC, acțiunile ce au cea mai mare rată de execuție sunt puse cel mai sus și cel mai în dreapta posibil. De asemenea, în cazul utilizării, spre exemplu, a unei instrucțiuni CASE în ST, secțiunile de cod executabile de cele mai multe ori sunt puse primele. Acest lucru, deși nu pare destul de important în zilele noastre datorită puterii de calcul a procesoarelor contemporane, poate salva procesul de câteva µs.

Un al doilea aspect în ceea ce privește scăderea timpului de execuție, este prioritizarea codului important. Pentru asta logica aplicației rulează de 20 de ori mai repede decât actualizarea interfețelor grafice. Secțiunile de cod ce nu necesită rulare continuă sau sunt executate pentru inițializarea diferitelor componente, sunt executate doar în momentele potrivite, datorită condițiilor de oprire ce împiedică scanarea lor [5].

Pentru a prezenta un mod de concepere al acestor aplicații, se folosesc următoarele notații:

- *SCT: Station Cycle Time sau timpul de ciclu al unei stații;*
- *DT: Distance Time sau timpul de deplasare al robotului;*
- *ADT: Adjacent Distance Time sau timpul de deplasare al robotului în punctul adiacent;*
- *RADT: Return Adjacent Time sau timpul de deplasare al robotului către stația din poziția adiacentă;*
- *PCT: Process Cycle Time sau timpul necesar prelucrării piesei;*

Cel mai rău caz într-o astfel de problemă, în care mai multe operații sunt executate asupra aceleiași piese, dar nu simultan, este cazul în care operațiile se execută câte una pe rând. În această situație, timpul necesar executării întregului proces este format din suma timpilor de ciclu ai fiecărea stații plus suma timpilor necesari deplasării de către robot a componentei de la o stație la alta plus timpul necesar deplasării robotului într-o poziție adiacentă, în vederea eliberării spațiului de lucru ca procesul să poată începe plus timpul necesar deplasării robotului din pozitia adiacentă în zona de lucru a stației pentru a ridica piesa. Relația este:

$$PCT = SCT + DT + ADT + RADT \quad (1)$$

Totuși, cea mai mare eficientizare a procesului o reprezintă disponibilizarea modulelor, imediat după ce piesa a fost înlăturată. În cazul anterior, o piesă nouă nu putea intra în proces, decât după ce piesa anterioară era trecută prin toate stațiile. În acest caz, după eliberarea fiecărui modul, el devine disponibil pentru a primi următoarea piesă. Prin urmare, dacă notăm cu M numărul de instanțe de stații, putem avea M piese la un moment de timp în proces.

Problema de optimizare constă în estimarea unui ordin sau a unei poziții de ridicare viitoare a robotului înainte ca acestea să fie transmise. Pentru a putea fi rezolvată, problema trebuie mai întâi formulată, iar pentru asta se folosesc de următoarele notații:

- *Există R un singur robot ce poate muta piese dintr-o stație în alta;*
- *Există $N = \{n_0, n_1, n_3, n_4\}$ un set de tipuri de stații posibile;*
- *Există $M \geq N$, $M = \{m_0, m_1, m_2, m_3, m_4, m_5, m_6\}$ un set de instanțe de stații posibile;*
- *Există un timp de tranziție x de la fiecare componentă a setului M la o altă componentă a setului M, inclusiv de la o stație la ea însăși, timpul fiind 0;*

$$Tm_{ij} = x \quad i \leftarrow 0:M, j \leftarrow 0:M \quad (2)$$

- *Fiecare componentă a setului M are o durată de efectuare variabilă, unde t reprezintă durata de efectuare;*

$$tm_i = x \quad i \leftarrow 0:M \quad (3)$$

- *Există o ordine a operațiilor ce trebuie efectuate, iar stațiile nu pot fi sărite;*

$$m_0 \rightarrow m_1,\ m_2 \rightarrow m_3,\ m_4,\ m_5 \rightarrow m_6 \quad (4)$$

- *Stația m_0 reprezintă o stație generator, iar stația m_6 o stație terminală;*
- *Chiar dacă stațiile sunt simulate, iar timpul lor s-a setat atunci când aplicația a fost creată, pentru partea de optimizare timpul de procesare reprezintă o necunoscută. Prin urmare acesta trebuie măsurat;*

Primul pas în rezolvarea problemei a fost aflarea tuturor necunoscutelor din acest sistem. Acestea sunt Tm_{ij} și tm_i sau timpul de deplasare de la o stație la alta și timpul de procesare al unei piese aferent fiecărei stații. Pentru aflarea timpului de deplasare, a trebuit folosit un mod de a automatiza acest proces, fără a calcula fiecare distanță manual(49 în total). Pentru asta, în interiorul aplicației, s-a creat o funcție de măsurare ce este activată în momentul startării deplasării robotului, adică în momentul trimiterii unui ordin. Algoritmul acestei funcții este prezentat în cele ce urmează.

Algoritmul 2 Calcularea timpului de deplasare între două stații

```
1: Intrări: state                    {variabilă boolean}
2: Ieșiri: timePassed                {variabilă de tip time}
3: rTrigger.CLK ← state              {inițializare trigger pe front crescător clock=true}
4: if rTrigger = ON then
5:     startTime ← currentTime
6: end if
7: fTrigger.CLK ← state              {inițializare trigger pe front descrescător}
8: if fTrigger = ON then
9:     timePassed ← currentTime – startTime
10: end if
11: return timePassed
```

Avand la dispoziție toate aceste date, prima strategie de optimizare poate fi aplicată. Ea constă în monitorizarea continuă a timpilor tututor stațiilor din proces. Atunci când robotul plasează o componentă într-un anumit modul, un timer este activat. În momentul în care robotul este în starea de inactiv, consultă timpul rămas în toate stațiile din jur și se deplasează în imediata apropiere a stației unde timpul rămas este cel mai scurt. Când ordinul de ridicare este receptat de robot, mișcarea este una minimă.

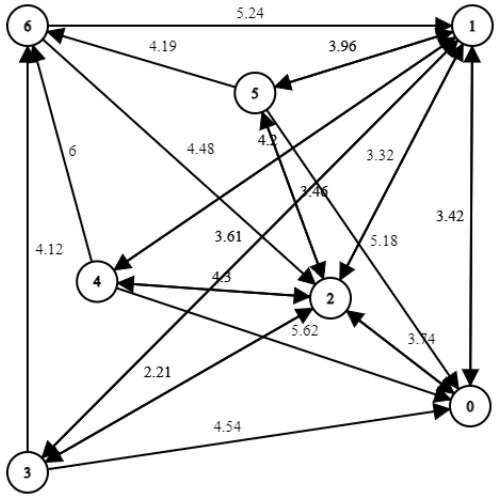

Figure 8: Graful orientat al aplicației

Având aceste detalii la dispoziție, s-a implementat o logică pentru ceea ce s-a discutat mai sus. Ieșirea funcției de optimizare este nextPosition, iar intrările sunt următoarele:

- *job* – o variabilă de tip *Order* prin care se verifică dacă robotul este în starea de repaus;
- *lastPosition* – variabilă de tip *Position* prin care se salvează ultima poziție în care se află robotul;
- *startTime* - variabilă de tip *Time* prin care se memorează timpul la care fiecare stație și-a început procesul;
- *cycleTime* - variabiliă în care se salvează timpul mediu de execuție al fiecărei stații;
- *availableModules* – variabilă prin care se verifică disponibilitatea stațiilor;

Prin urmare, următorul algoritm de calcul a următoarei deplasări a fost folosit:

Algoritmul 3 Optimizare

```
Intrări: job, lastPosition, startTime=[TC, TD1, TD2, TT1, TT2, TT3], cycleTime=[TC, TD1, TD2,
TT1, TT2, TT3], AvailableModules
Ieșiri: nextPosition
if job = noJob then
    for i = 0 to modulesNumber do
        shortestTime[i] ← startTime[i] + movementTime[lastPosition][i] + cycleTime[i]
    end for
    minValue ← getMinOf(shortestTime)
    minIndex ← getIndexOf(minValue)
    finalPosition ← minIndex
    switch finalPosition do
      case conveyorPosition
          if drilling = empty then
              nextPosition = conveyorPosition
          else
              shortestTime[finalPosition] ← maxValue + 1
          end if
      case drillingPosition
          if test = empty then
              nextPosition = drillingPosition
          else
              shortestTime[finalPosition] ← maxValue + 1
          end if
      case testPosition
          if metalization = empty then
              nextPosition = testPosition
          else
              shortestTime[finalPosition] ← maxValue + 1
          end if
    if noPositionAvailable then
        nextPosition ← testPosition
    end if
else
    nextPosition ← NoPositionFound
end if
return nextPosition
```

VII. TESTARE ȘI REZULTATE

Rezultatele pot fi observate în figura 9. Pentru 5 piese, fără funcția de optimizare pornită, timpul total de procesare a fost de 399.26 secunde. Pentru același număr de piese și aceeași ordine a stațiilor, dar de data aceasta cu funcția de optimizare pornită, timpul de prelucrare a ajuns doar la 389.78 secunde. Diferența de 10 secunde este una ideală datorită parametrilor aleși, dar în realitate, ea depinde foarte mult de timpii aleși aleatoriu și de viteza setată a robotului. După un calcul simplu putem estima că producția a crescut cu aproximativ 27 de piese la o durată de funcționare de 24 de ore.

```
'POU' (350): Elapsed time for 5 parts with optimization mode off is: 399.26 s
'POU' (350): Elapsed time for 5 parts with optimization mode on is: 389.78 s
```

Figure 9: Rezultate în urma aplicării optimizării

VIII. CONCLUZII

Conceperea, dezvoltarea și testarea unui astfel de proiect implică o serie de evenimente ce trebuie puse în ordinea corectă încă de la început. Cunoașterea tehnologiilor, dar și a aparaturii *hardware* folosite necesită un volum mare de timp de studiu, însă rezultatele sunt pe măsură. De asemenea, este bine cunoscut faptul că îmbunătățirea unui proces industrial din toate punctele de vedere este aproape imposibilă. De aceea, această lucrare se bazează în principal pe un singur aspect și anume optimizarea secvenței de mișcări a unui robot, în vederea scăderii timpului de ciclu.

REFERINȚE

[1] Beckhoff. [Online]. Available: https://www.beckhoff.com/en-us/.

[2] EtherCAT, "EtherCAT Technology Group," [Online]. Available: https://www.ethercat.org/default.htm.

[3] W. Bolton, Programmable logic controllers, Newnes, 2015, pp. 1-24.

[4] G. F. Rossano, C. Martinez, M. Hedelind , S. Murphy și T. A. Fuhlbrigge, „Easy robot programming concepts: An industrial perspective," în *2013 IEEE international conference on automation science and engineering (CASE)*, 2013, pp. 1119-1126.

[5] Y.-J. Park, G. Ahn și S. Hur, „Optimization of pick-and-place in die attach process using a genetic algorithm," *Applied Soft Computing*, vol. 68, pp. 856-865, 2018. R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.