MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

2022

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

OPTIMIZAREA ADAPTIVĂ A EXECUȚIEI MIȘCĂRILOR ROBOȚILOR INDUSTRIALI

PROIECT DE DIPLOMĂ

Autor:

Conducător științific:

2022

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Vizat,

DECAN
Prof.dr.ing. Liviu MICLEA

DIRECTOR DEPARTAMENT AUTOMATICĂ
Prof.dr.ing. Honoriu VĂLEAN

Autor: **Radu-Cristian GOȚA**

OPTIMIZAREA ADAPTIVĂ A EXECUȚIEI MIȘCĂRILOR ROBOȚILOR INDUSTRIALI

1. Enunțul temei: În această lucrare se prezintă modul de funcționare al roboților in mediul industrial, integrarea acestora împreună cu un controler logic programabil folosind o tehnologie de comunicație utilizată în practică și conceperea unei strategii de scurtare a timpilor necesari pentru procesarea unei piese/componente, exemplificată prin intermediul unei aplicații de tipul "Pick and Place" ce simulează stațiile de prelucrare.

2. Conținutul proiectului: Pagina de prezentare, Declarație privind autenticitatea proiectului, Sinteza proiectului, Cuprins, Introducere, Studiu bibliografic, Aparatură *hardware* utilizată, Analiză, proiectare și implementare, Concluzii și Direcții de dezvoltare, Bibliografie, Anexe.

3. Locul documentației: Universitatea Tehnică din Cluj-Napoca, Departamentul de Automatică

4. Data emiterii temei:

5. Data predării:

Semnătura autorului

Semnătura conducătorului științific

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Listă de figuri

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SINTEZA

proiectului de diplomă cu titlul:

Optimizarea adaptivă a execuției mișcărilor roboților industriali

Autor:

Conducător științific:

1. Cerințele temei: conceperea, crearea și simularea unui proces industrial de fabricație, integrarea acestuia cu un robot care asigură transportul componentelor produse și dezvoltarea unei metode de îmbunătățire a nivelului producției.

2. Soluții alese: crearea și implementarea a două aplicații, una prin care stațiile de lucru sunt simulate în mediul de dezvoltare integrat *Twincat 3* pentru un computer logic programabil *Beckhoff CX5140* folosind limbajul *Structured Text* și una pentru coordonarea mișcărilor robotului *Stäubli TS2-80* folosind mediul de programare *Stäubli Robotics Suite* folosind limbajul *VAL3.* Dezvoltarea unui algoritm de optimizare prin care se minimizează timpul de ciclu.

3. Rezultate obținute: un sistem complet automatizat format din mai multe stații simulate de procesare prin care piesele/componentele trebuie să treacă secvențial, o strategie de transport a acestora bazată pe manevrarea unui braț robotic și o metodă de comunicare care să faciliteze schimbul de date dintre computerul logic programabil și robot. Un algoritm de optimizare a secvenței de mișcări a robotului prin care timpul de ciclu a fost îmbunătățit.

4. Testări și verificări: testarea modului automat, în care procesul se execută independent, fără intervenție asupra lui, testarea eficienței algoritmului de optimizare aplicat și al soluției de comunicare implementată.

5. Contribuţii personale: realizarea ambelor aplicaţii necesare pentru simulare, a comunicaţiilor aferente dintre acestea şi dezvoltarea algoritmului de optimizare.

6. Surse de documentare: articole ştiinţifice şi cărţi de specialitate din domeniul de comunicaţii, al computerelor logice programabile şi al roboţilor industriali; manuale de utilizare şi documentaţie al programelor *Twincat 3*, *Stäubli Robotics Suite.* Standarde şi norme impuse în ceea ce privește programarea elementelor industriale.

Semnătura autorului

Semnătura conducătorului ştiinţific

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Cuprins

1 Introducere

1.1 Scurt istoric al automatizării industriei

În ultimele secole industria a continuat să se dezvolte masiv, plecând de la fabrici în care toata munca era realizată de oameni si ajungând, în momentul actual, la fabrici complet automate în care intervenţia umană este minimă. La aceste modificări drastice au contribuit, în mare parte, apariţia utilajelor ce foloseau apa sau aburul ca şi combustibil, iar mai apoi a curentului ca primă sursă de energie. Următorul pas al acestui progres de proporţii este reprezentat de internet, ce faciliteaza comunicarea inteligentă şi continuă fără a putea fi întreruptă nici măcar de barierele geografice.

1.2 Context general

În mediul industrial se folosesc roboţi pentru manipularea diferitelor componente sau pentru realizarea anumitor operaţii. Controlul acestor roboţi se face prin intermediul unui *PLC* (engl. *Programmable Logic Controller - PLC*) care, în acelaşi timp, ţine evidenţa tuturor parametrilor şi comandă întreg procesul, precum in figura 1.1.

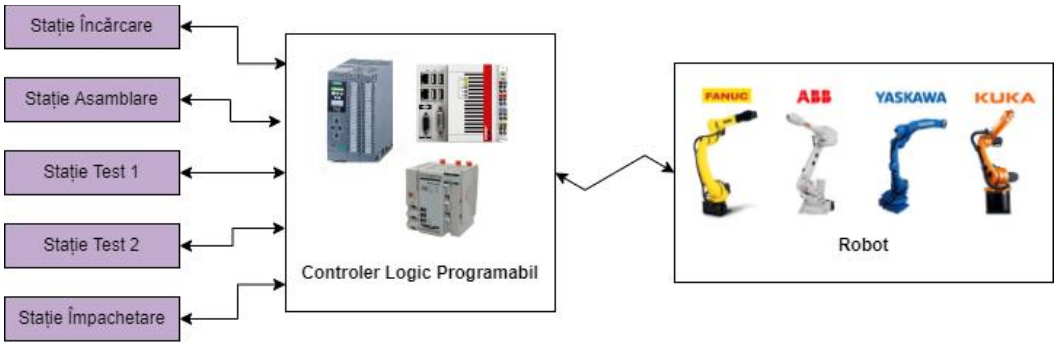

Figura 1.1: Integrarea PLC-ului cu robotul şi celelalte staţii

PLC-ul interacţionează cu robotul pe baza unor ordine (engl. *Job*, *Order*, *Task*) pe care le trimite acestuia. Când robotul recepţionează aceste ordine, el execută o secvenţă de program stabilită după care returnează un feedback. Cum întreg procesul este comandat de către un *PLC*, acesta reţine starea fiecărei staţii şi a fiecărei piese. El trimite un ordin robotului pentru a muta o piesă de la o staţie sau punct de prelucrare la o alta, în vederea executării unei noi modificări asupra ei, urmând procesul tehnologic specific. Un exemplu al unui astfel de ordin ar fi "ia piesa din staţia de asamblare şi plaseaz-o în staţia de test 1". În realitate, robotul se află intr-o oarecare poziţie care, în cele mai multe

dintre cazuri, diferă față de poziția în care a fost trimis deci, prin urmare, între momentul în care robotului i-a fost asignuit *task*-ul și momentul în care robotul execută acel *task*, există un oarecare timp de deplasare sau timp mort în care unele stații stagnează în așteptarea robotului.

În cadrul unui proces, timpul calculat din momentul în care materia primă suferă prima modificare până în momentul în care avem un obiect finalizat sau mai pe scurt, această trecere prin tot procesul, se numește timp de ciclu (engl. *Cycle-time*). Timpul de ciclu este în cea mai mare parte influențat de parametrii prelucrării, de natura obiectului pe care vrem să îl obținem și diferă de la proces la proces. Scurtarea acestui timp de execuție duce la îmbunătățirea producției, ceea ce înseamnă automat creșterea profiturilor companiei. Această imbunătățire a producției se poate face în două moduri și fiecare are avantajele și dezavantajele ei.

Prima îmbunătățire, cea mai simplă, dar nu și cea mai logică ar fi, bineînțeles, achiziționarea unor unelte mai rapide și mai performante (*hardware*). A doua îmbunătățire se aplică atunci când resursele *hardware* sunt limitate. De cele mai multe ori fiind și cea mai potrivită, ea constă în perfecționarea aparaturii deja disponibile (*software*).

1.3 Obiective

În vederea exemplificării unei astfel de îmbunătățiri de natură *software*, scopul principal al acestei lucrări este de a crea un proces simulat și de a găsi o metodă de optimizare care să scadă timpul de ciclu pe cât de mult posibil prin minimizarea sau chiar eliminarea timpului necesar mișcării între două puncte diferite în spațiu a robotului. În același timp, soluția trebuie să fie adaptabilă pe orice tip de proces controlat de un *PLC Bekhoff* în care acțiunea de tipul pick and place este realizată de un robot *Stäubli* cu 4 axe.

Pentru îndeplinirea acestui scop următoarele obiective au fost alese:

- alegerea unui proces care poate fi simulat și care implică mai multe acțiuni asupra aceleiași componente, dar nu simultan;

- conceperea unei aplicații pentru *PLC Beckoff CX5140-0155* folosind mediul de programare *Twincat 3* împreuna cu *Visual Studio* care să reprezinte în detaliu și să simuleze stațiile necesare unei viitoare optimizări ai timpului de ciclu, dar si mișcările robotului;

- dezvoltarea unei aplicații pentru un robot *Stäubli TS2-80 SCARA* cu 4 axe folosind mediul de programare *Stäubli Robotics Suite* care să faciliteze integrarea robotului in procesul respectiv;

- crearea unei comunicații între *PLC Beckoff CX5140-0155* și robotul *Stäubli TS2-80* folosind o tehnologie de comunicare de tip *EtherCAT*;

- integrarea robotului in mediul de programare *Twincat 3*, alături de staţiile simulate şi crearea logicii folosite pentru coordonarea procesului;

- analizarea comportamentului procesului şi colectarea datelor de trebuinţă pentru implementarea optimizării timpilor de ciclu;

- întocmirea unui algoritm de optimizare care să îmbunătăţească timpii de execuţie şi să mărească nivelul producţiei;

- colectarea datelor după ce algoritmul de optimizare a fost aplicat, analizarea rezultatelor şi revizuirea acestuia în cazul în care rezultatele sunt nule;

Un al doilea scop al acestui proiect este de a uşura munca necesară unei viitoare implementări şi testări sau a unei îmbunătăţiri a acestei idei, această aplicaţie putând servi ca model.

1.4 Specificaţii

Primul pas spre atingerea scopului ales este setarea unor reguli stricte în ceea ce priveşte construirea şi dezvoltarea aplicaţiilor. Ideea de bază de la care se pleacă este că procesul trebuie să aibă timpi morţi, mai precis, timpi în care robotul nu are nimic de făcut, pentru ca aceştia să fie eliminaţi în urma optimizării. O a doua cerinţă este simplitatea. Deoarece dorim ca aplicaţia să poată fi folosită ca model în vederea următoarelor implementări, aceasta trebuie să fie cât se poate de compactă şi bine organizată. Adăugarea staţiilor trebuie să se facă rapid, scoaterea lor la fel, iar orice modificare trebuie să se poată face uşor.

Limbajul de programare trebuie ales cu grijă pentru a respecta toate cerinţele de mai sus. Pentru că într-un astfel de domeniu, al fabricaţiei, fiecare milisecundă contează, structurile de date cu care lucrăm trebuie să fie cât mai necomplicate.

În ceea ce priveşte partea de optimizare, metodele alese şi algoritmii folosiţi trebuie să aibă un timp mic de execuţie şi să nu fie consumatori mari de memorie, deoarece timpul disponibil pentru astfel de calcule este unul minim.

Întrucât comunicarea între *PLC*, robot şi celelalte staţii din proces se face permanent în timpul producţiei, se recomandă ca datele schimbate să fie cât mai primitive.

Datorită faptului că staţiile sunt simulate, pentru o mai bună înţelegere a proiectului, se recomandă crearea unei interfeţe vizuale în care se pot observa stările fiecărei staţii.

Pentru manevrabilitate, dar şi pentru a facilita analiza şi testarea rezultatelor este lesne înţeles faptul că va fi nevoie de o interfaţă de control care să permită pornirea, oprirea şi eventual resetarea poziţiei robotului. În acelaşi timp, programul care comandă şi monitorizează mişcarea, dar şi logica din spatele acesteia trebuie să fie simple.

2 Studiu bibliografic

2.1 Controler Logic Programabil

Un controler logic programabil(engl. *Programmable Logic Controller - PLC*) este o formă specială de controler bazat pe un microprocesor care stochează instrucţiuni şi le aplică ulterior implementând funcţii logice, aritmetice, etc. în ideea de a controla alte dispozitive.

În indrustrie, *PLC*-ul este unitatea de bază in ceea ce priveşte controlul. Rapiditatea si precizia cu care efectuează calculele, uşurinţa de integrare şi configurare, costul mic şi varietatea modelelor fac ca el să fie numit la nivel global "cal de lucu" (engl. *Work horse*) [1]. Principalele avantaje ale unui astfel de dispozitiv sunt rezistenţa in condiţii extreme şi încorporarea comodă a acestuia împreună cu alte elemente necesare controlului, cum ar fi senzori şi actuatori. Există o gamă largă de astfel de produse faimoase printre care cele mai cunoscute ar fi: *Rockwell Automation*, *Siemens*, *Omron*, *Bekhoff*, *Mitsubishi Electric*, *Schneider Electric* etc.

2.1.1 Arhitectura unui *plc*

Elementele consolidate ale unui *PLC*, pe baza figurii 2.1 sunt după cum urmează:

- unitatea centrală de control (engl. *Central Processing Unit - CPU*) care are rolul de a prelua datele venite de la intrări, de a le interpreta în funcţie de regulile care au fost programate în memorie şi mai apoi de a transmite deciziile luate ieşirilor;
- memoria care are rolul de a stoca programul scris ce conţine regulile de control venit de la o sursă externă, cum ar fi un computer, dar şi datele venite de la intrări şi datele modificate pentru ieşiri;
- modulul de intrare care colecteaza datele de la elemente externe şi le trimite mai apoi la memorie. Aceste date sunt de forma a unor semnale care pot fi discrete, analogice şi digitale. Un exemplu de semnal discret este întrerupătorul care poate fi doar închis sau deschis. Semnalele digitale reprezintă o serie de semnale discrete, iar semnalele analogice sunt direct proporţionale cu variabila pe care o măsoară;
- modulul de ieşire preia semnalele discutate mai sus şi le transmite mai departe către actuatori. Un exemplu ar putea fi un motor a cărui turaţie este controlată cu ajutorul unui semnal analogic provenit de la *PLC* în urma unor calcule;

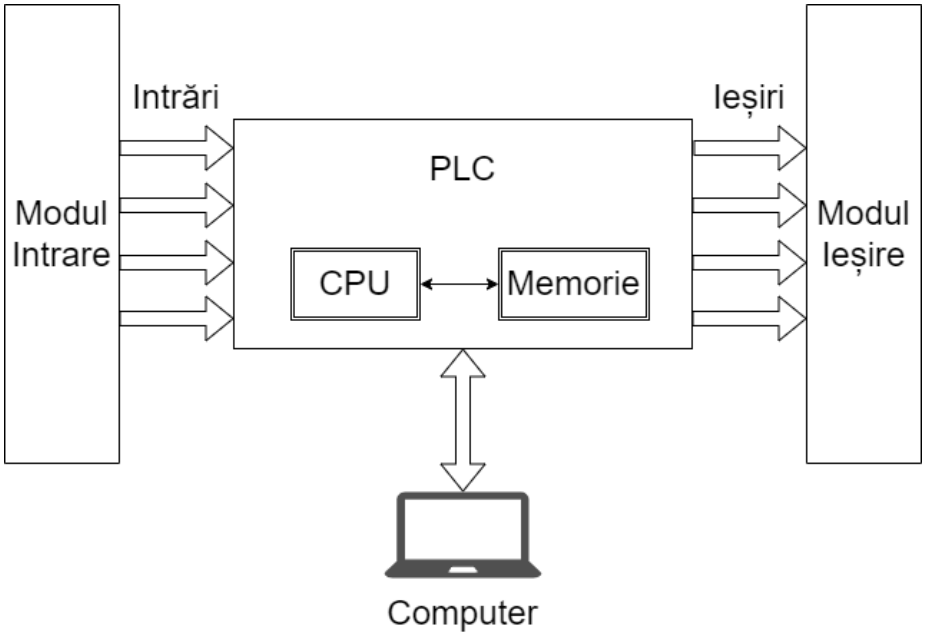

Figura 2.1: Arhitectura unui plc

- computerul sau dispozitivul de programare este folosit pentru a scrie programul ce urmează a fi încărcat în memoria *PLC*-ului. În practică se folosesc o multitudine de limbaje de programare printre cele mai folosite fiind textul structurat (engl. *Structured Text - ST*), lista de instrucţiuni (engl. *Instruction List - IL*) şi diagrama scară (engl. *Ladder Diagram - LD*) [2];

2.1.2 Standardul *IEC 61131*

De-a lungul anilor multe standarde au fost propuse în ceea ce priveşte programarea *PLC*-urilor. Deşi fiecare companie dezvoltatoare de *PLC*-uri are propriul ei mediu de programare şi propriile metode de configurare a dispozitivului, prin folosirea acestui standard, care poate fi sau nu acceptat de către o companie, tot procesul necesar creării unei aplicaţii este şablonizat din momentul gândirii acesteia până la rezultatul final. Odată cu adaptarea la acest standard, munca inginerilor devine mult mai simplă, ei putând să se focuseze mai mult pe crearea tehnicilor si algoritmilor de control şi mai puţin pe partea de implementare a acestora. Totodata, eforturile companiilor în instruirea angajaţilor devin minime, partea de depanare, mentenanţă şi consultare se face mai intuitiv, iar erorile sunt din ce în ce mai uşor de depistat. În acelaşi timp trebuie înteles faptul că adoptarea acestui standard nu este obligatoriu să fie făcută în totalitate deoarece acest lucru ar duce la imposibilitatea devansării dezvoltatorilor între ei. Un echilibru trebuie găsit în ceea ce priveşte acest aspect [3].

2.1.3 Securitatea unui sistem distribuit

În trecut, sistemele industriale de control erau conectate la o reţea locală, izolată, ele neavând interacţiuni cu internetul vast. Datorită prosperului acestei industrii şi necesităţii de a controla mai multe procese deodată aflate la distanţe geografice considerabile, aceste sisteme au fost conectate la internet sau intranet. Odată cu aceste racordări, noi posibilităţi de penetrare a acestora au fost posibile.

Sistemele de control au fost concepute pentru a lua decizii in situţii critice într-un timp scurt, pentru a fi preemptive şi nicidecum pentru a fi capabile să respingă sau să evite atacuri cibernetice. Majoritatea lor sunt formate din mai multe componente printre care se găsesc *PLC*-uri, interfeţe om-maşină(engl. *Human Machine Interfaces - HMI*), unităţi terminale principale (engl. *Master Terminal Units - MTU*), unităţi terminale la distanţă (engl. *Remote Terminal Units - RTU*). Toate acestea pot fi o ţintă deoarece toate sunt conectate în reţea. În acelaşi timp, protocoalele de comunicare dintre ele, care la fel au fost gândite pentru viteză şi rate mari de transfer, pot facilita accesul atacatorilor. Pentru a înţelege mai bine, în figura 2.2 este descrisă arhitectura unui astfel de sistem [4].

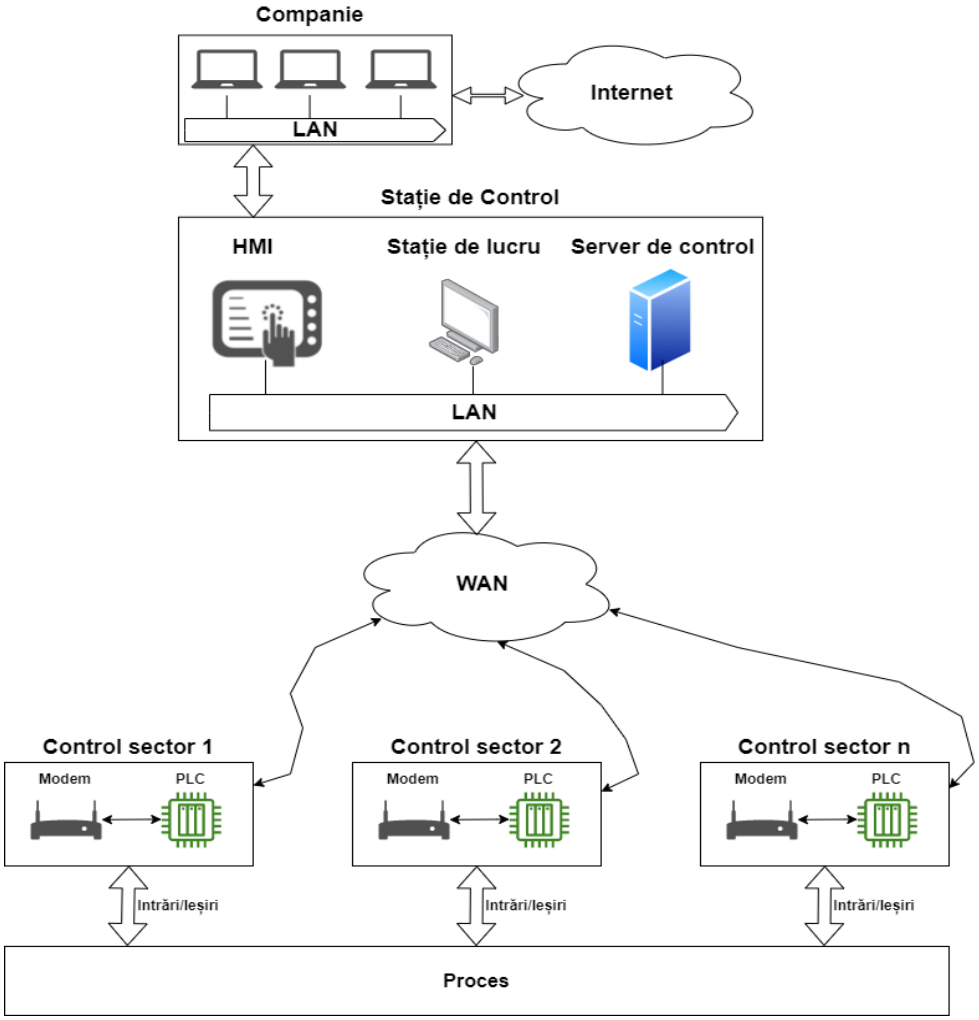

Figura 2.2: Prezentarea generală a unui sistem de control distribuit

În lucrarea [5] este prezentat şi analizat un astfel de atac asupra unei reţele de energie electrice din Ucraina în care un utilizator terţ a pătruns în sistemele *SCADA*(*Supervisory Control and Data Acquisition*) şi a oprit aprovizionarea cu energie electrică a aproximativ 80.000 de clienţi, dezactivând şapte substaţii generatoare de electricitate. Părţi suplimentare ale reţelei au fost afectate şi operatorii au fost forţaţi să treacă în modul manual. Acest lucru a fost posibil datorită unui e-mail ce conţinea un ataşament prin care atacatorii au reuşit să fure credenţiale prin care au obţinut controlul de la distanţă asupra staţiei de lucru şi a *HMI*-ului şi au oprit toate substaţiile, au blocat

accesul operatorilor la întreg sistemul şi au supraîncărcat reţeaua de telefonie mobilă pentru a limita transmiterea de informaţii în interiorul companiei.

În vederea protejării unor astfel de sisteme şi a evitării unor astfel de atacuri, antiviruşii pot fi folosiţi ca măsură de prevenţie. Totuşi ei nu pot ţine mereu pasul cu avansul rapid al *software*-ului rău intenţionat şi pentru asta, se recomandă supravegherea continuă a reţelei şi raportarea imediată a neregularităţilor către experţi [6].

2.2 Roboţi industriali

Această categorie de roboţi este una bine cunoscută pe piaţă şi în acelaşi timp foarte diversificată. În ultimele decenii, roboţii cu schimbat cu totul faţa industriilor. Aceştia au fost preferaţi în locul oamenilor pentru muncile repetitive, dar şi pentru cele nocive. În zilele noastre există roboţi pentru orice fel de sarcină, de la operaţii de tăiere, sudare, vopsire la transport, inspecţie şi mutare. Sunt folosiţi în toate industriile printre care cea farmaceutică, de fabricare dar şi în domeniul medical.

Principalele avantaje când vine vorba de astfel de roboţi este munca continuă, fără întrerupere, uşurinţa programării şi întreţinerea minimă.

2.2.1 Caracteristici generale

Un prim aspect general atunci când vine vorba de roboţi industriali este gradul de manevrabilitate al acestuia. Este important ca robotul să poată servi mai multor scopuri dacă este nevoie şi să nu fie limitat în îndeplinirea unei singure sarcini (engl. *Task*). Prin urmare se definesc mai multe tipuri de roboţi în funcţie de mişcarea pe care o pot realiza:

- braţ robotic articulat care poate fi definit de 3 până la 10 articulaţii de rotaţie şi oferă manevrabilitate totală în întreg planul tridimensional. Aceşti roboţi sunt folosiţi pentru operaţii de asamblare, împachetare, vopsire, sudare, unde spaţiul de obicei este unul limitat şi punctele de lucru sunt specifice şi/sau greu accesibile;
- braţ robotic paralel ce constă în mai multe lanţuri ataşate unei baze fixe şi una mobilă, folosiţi în special în sarcinile unde precizia şi viteza sunt absolut necesare. Acestea sunt atinse deoarece, spre deosebire de roboţii seriali, greutatea actuatorilor nu influenţează aproape deloc mişcarea. Un exemplu îl reprezintă braţul robotic Delta ce este format din 3 lanţuri cinematice simetrice;
- braţ robotic cilindric *SCARA*(*Selective Compliance Articulated Robot Arm*) folosit în special pentru sarcinile de selectare, ridicare şi plasare(engl. *Pick and Place*) şi asamblare. Are o precizie ridicată, fapt pentru care este utilizat preponderent în procesele de fabricaţie ale circuitelor imprimate(engl. *Printed Circuit Board – PCB*);
- braţ robotic macara cu toate articulaţiile liniare, folosit în special în sarcinile de paletare unde sarcina este grea;

2.2.2 Componentele unui robot industrial

Principalele componente ale unui robot industrial sunt: braţul robotic, senzorii, efectorul final, controlerul şi elementele de execuţie. Toate acestea împreună cu o strategie bună de control au revoluţionat procesele de fabricaţie din întreaga lume.

Ne putem gândi la braţul robotic ca la un braţ uman format dintr-o bază(engl. *Base*), un umăr(engl. *Shoulder*), un cot(engl. *Elbow*) şi o încheietură(engl. *Wrist*). Senzorii sunt dispozitive care detectează anumite modificări şi declanşează anumite răspunsuri la acestea. Importanţa acestora atunci când vine vorba de roboţi este dată de noţiunile de siguranţă, pentru a evita coliziunile cu mediul înconjurător, de precizie, pentru a respecta în exactitate cerinţele impuse, dar şi de măsurare, pentru eventualele analizări de comportament. Câteva tipuri de senzori folosiţi în producerea unui robot, în funcţie de scopul şi mediul pentru care este dezvoltat sunt: senzori de atingere şi forţă, de temperatură, de sunet, senzori chimici şi de lumină, iar cei mai importanţi, de poziţie(potenţiometru sau encoder).

Alături de un robot industrial puternic stă un controler şi mai puternic responsabil cu tot ce ţine de partea de gândire. Acesta este de fapt un computer specializat legat de robot cu capabilităţi foarte bune în luarea de decizii şi cu o capacitate foarte mare de memorie. Este responsabil pentru activarea elementelor de execuţie, pentru efectuarea de calcule complicate atunci când vine vorba de traictorii şi viteze, dar şi pentru controlul întregului robot, de la preluarea datelor de la senzori, intrepretarea lor, luarea de decizii şi transmiterea acestora mai departe în sistem. În acelaşi timp el memorează codul care descrie mişcarea robotului, venit din surse externe, îl transformă în parametrii pe care mai apoi îi trimite la motoare.

Elementele de execuţie sunt responsabile cu mişcarea robotului. În funcţie de tipul de mişcare pe care îl execută, dar şi de procesul asupra căruia este supus, ele pot fi electrice, pneumatice şi hidraulice.

- actuatorii electrici sunt motoare ce convertesc energie electrică în mişcare de translaţie sau de rotaţie. Câteva tipuri de motoare sunt motoarele în curent continuu, servo-motoarele, motorul pas cu pas, motorul *BLDC*. Principalele avantaje ale acestora sunt precizia ridicată, feedback-ul instant necesar controlului şi mentenanţei, programarea uşoară, sunetul redus şi relaţia bună cu mediul înconjurător. Totuşi există şi multe dezavantaje printre care supraîncălzirea şi lipsa manevrabilităţii. Având parametrii fixi, un motor nu poate fi modificat decât dacă este înlocuit. În acelaşi timp, acest tip de actuatori nu poate fi folosit în orice mediu;
- un al doilea tip de actuatori sunt cei hidraulici, folosiţi în special atunci când sarcina de lucru este una mare. În comparaţie cu cei electrici, aceştia pot produce forţe foarte mari capabile să ridice greutăţi pe măsură. Sunt utilizaţi acolo unde precizia, viteza mare si stabilitatea sunt necesare. Avantajele acestora sunt uşurinţa cu care se controlează, sunetul redus, simplitatea şi acurateţea, iar dezavantajele le reprezintă preţul ridicat, mentenanţa atentă, iar dacă ceva se defectează, scurgerile de lichid pot cauza daune mediului;

- elementele de execuţie pneumatice sunt folosite când igiena este mai importantă decât precizia. Fiind foarte gălăgioase, sunt printre cele mai rar întâlnite;

În funcţie de sarcina care îi este atribuită, robotului industrial i se pot ataşa o multitudine de efectori finali. Aceştia pot varia de la un simplu sistem de prindere controlat electric până la mecanisme complexe de inspecţie, diferite aparate pentru sudare, lipire, vopsire etc sau chiar senzori pentru detectarea anumitor suprafeţe şi sunt independenţi de robot. Totodată, fără astfel de efectori, în cele mai multe dintre cazuri, robotul devine inutil.

2.2.3 Programarea unui robot industrial

Principalul mod prin care un robot industrial poate fi programat este prin intermediul unui pendant de invăţare(engl.*Teach Pendant*). În domeniu, programarea robotului folosind această metodă poartă numele de "*teaching*". Pendantul reprezintă calea sau mai bine spus legătura între operator şi memoria robotului. Prin intermediul lui, robotul poate fi mişcat manual şi poate invăţa puncte fixe. Metoda este destul de intuitivă, deoarece interfeţele pendantelor au devenit foarte uşor de înţeles. În figura 2.3 este reprezentat un astfel de pendant. Există o varietate mare de astfel de dispozitive în funcţie de producător şi în majoritatea dintre cazuri, ele vin ataşate de robot.

Figura 2.3: Teach Pendants

Avantajele găsite la acest punct stau în siguranţă, precizie şi intuinţă. Deoarece robotul nu poate fi mişcat decât pas cu pas, rata de coliziune este minimizată, iar mişcarea poate fi întreruptă instant prin folosirea butonului de stop. Precizia este asigurată de faptul ca punctele şi mişcările sunt trecute direct în memoria robotului, iar intuinţa constă în uşurinţa folosirii aparatului. Programarea, însă, poate deveni destul de anevoioasă atunci când este nevoie de secvenţe de mişcări complexe, iar în unele cazuri, programele complicate nu pot fi scrise decât prin intermediul altor metode. În acelaşi timp, *pendant*-ul poate fi foarte folositor pentru întelegerea modului de funcţionare al unui robot, dar şi pentru exersarea anumitor poziţii şi mişcări.

Un alt mod prin care un astfel de robot poate fi programat este prin îndrumarea acestuia. Metoda constă în mişcarea manuală a braţului în modul în care vrem să se efectueze deplasarea şi prin punctele prin care acesta dorim să treacă. Tot acest proces este înregistrat în memoria robotului şi reparcurs la fiecare ciclu. Spre exemplu, un operator poate mişca braţul robotic prin diferite puncte, pe o anumită traictorie pentru a monta o componentă. În momentul pornirii automate, robotul reparcurge traseul memorat de câte ori este lăsat. Principalul avantaj al acestui mod de programare este intuitivitatea, deoarece robotul este ghidat fizic exact pe traseul pe care dorim să-l parcurgă. Nu există nicio problemă de siguranţă atât în ceea ce priveşte programatorul sau operatorul, cât şi în ceea ce priveşte coliziunea robotului cu alte obiecte, ea fiind imposibilă. Acest tip de programare este preferat atunci când robotul şi oamenii colaborează în îndeplinirea unui scop, iar suprafaţa de lucru este comună.

Un al treilea mod de programare este programarea *offline*. Ea constă într-un alt dispozitiv precum un laptop şi într-un mediu de dezvoltare integrat(*IDE*) în care se scriu instrucţiuni. După ce programul a fost creat, el este testat pe un model virtual, de obicei 3D, pentru a se vedea eficienţa şi corectitudinea lui. Dacă standardele au fost îndeplinite, el este încărcat mai apoi în memoria robotului, de unde poate fi accesat şi utilizat în mod real. Avantajele unei astfel de metode stă în complexitatea aplicaţiilor care pot fi dezvoltate folosind un limbaj de programare corespunzător, dar şi în optimizarea şi eficienţa operaţiilor. Totodată, acest mod de programare poate fi făcut doar de personal calificat care înţelege pe deplin specificaţiile tehnice şi cele *low-level* în ceea ce priveşte roboţii.

Cu timpul, datorită nevoii de independabilitate, alte metode au luat naştere precum programarea prin pictograme făcută prin cu drag and drop prin intermediul unei interfeţe de utilizator. Prin acest tip, toate erorile de sintaxă sunt evitate. Programarea se face uşor şi intuitiv, dar din nou, complexitatea este mică. Programarea prin intermediul diagramelor de flow este destul de răspândită şi poate fi observată mai ales în mediul de lucru LabView unde utilizatorii pot lega şi configura blocuri funcţionale care descriu mişcări fizice. Cel mai intuitiv mod de programare existent pe piaţă în acest moment, este însă programarea prin fişiere CAD. Au fost dezvoltate o multitudine de software-uri în care nu doar că putem crea şi simula în acelaşi timp mişcările robotului, ci putem crea întreg procesul. Un astfel de exemplu este Factory IO care, în acelaşi timp, poate fi considerat un joc, în care procesele sunt simulate. Majoritatea programelor de acest gen au o proprietate prin care, în funcţie de aplicaţia făcută, poate fi generat cod automat gata pentru implementat în realitate [7].

2.2.4 Sistemul de control al unui robot industrial

În figura 2.4 este descris modul de control al unui robot industrial. Prin intermediul pendantului, acesta primeşte de la operatori comenzi pe care mai apoi le trimite la interfaţa de mişcare. Acestea sunt procesate şi transmite la rândul lor mai departe către efectori.

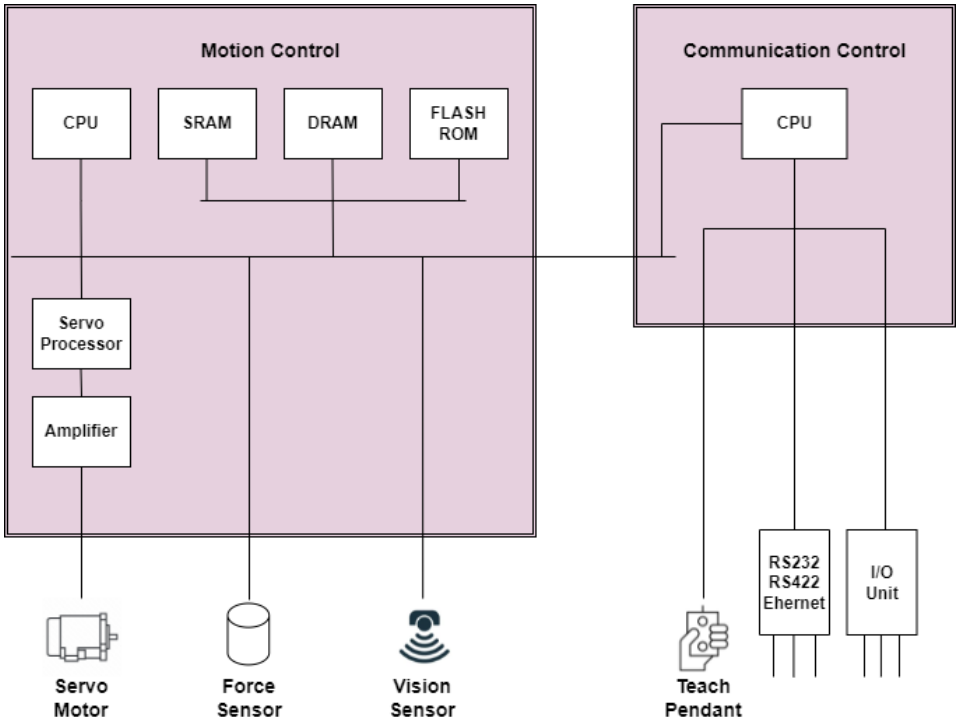

Figura 2.4: Sistemul de control al unui robot industrial

2.3 Protocoale de comunicaţie

Datorită nevoii mari de a creşte viteza şi de a mări producţia în mai toate domeniile, comunicaţiile au avut şi ele parte de schimbări. S-au completat sau chiar înlocuit complet sistemele proprii de comunicare din sistemele *SCADA* cu sisteme *Fieldbus*. În zilele noastre, comunicaţia prin fir de tip Fieldbus este standardizată şi folosită în majoritatea sistemelor de control, favorizând schimbul de date dintre senzori, intermediari şi actuatori [8].

Pentru a înţelege mai bine de ce metodele de comunicaţie sunt importante, mai jos sunt descrise cele mai mari cerinţe în domeniul industrial:

Comportamentul în timp real

Sistemele de timp real se bazează într-o proporţie foarte mare pe timpul de ciclu a pachetelor de date. Prin urmare, pe baza acestui timp, pot fi definite patru clase ce definesc timpul real:

- *Soft Real-Time*, unde timpul de ciclu este variabil, iar nerespectarea termenelor limită nu duce la probleme severe;
- *Hard Real-Time*, unde timpul de ciclu variază doar în intervalul 1-10 ms, folosit în sistemele de control;
- *Isochronous Real-Time*, unde timpul de ciclu variază între 250 µs şi 1 ms, iar *jitter*-ul este mai puţin de 1 µs;
- *Non Real-Time*, unde timpul nu conteaza, folosit pentru schimbul de date în vederea analizării sau observării lor;

Siguranţa funcţională

Protecţia împotriva defecţiunilor sau a funcţionării incorecte la nivelul reţelei împiedică schimbul eronat de date sau chiar întreruperea acestuia.

Securitatea

Datorită evoluţiei masive a internetului, protecţia datelor a devenit un principal factor de luat în considerare atunci când o reţea este configurată [9]. Majoritatea sistemelor de comunicaţii industriale se bazează pe sisteme de operare comerciale despre care este binecunoscut faptul că au vulnerabilităţi. Pentru a diminua rata de succes a atacurilor, mai multe obiective au fost definite:

- Confidenţialitatea: împiedicarea scurgerilor de date către persoane neautorizate. Aici pot intra date referitoare la performanţele sistemelor, planuri de dezvoltare, parole etc.
- Integritatea: prevenirea persoanelor neautorizate în schimbarea parametrilor proceselor, valori de senzori etc.
- Valabilitatea: împiedicarea persoanelor neautorizate în a bloca accesul utilizatorilor. Pe lângă problemele economice ce pot apărea, stoparea operatorilor din a avea acces la datele şi controlul sistemelor poate duce la probleme de siguranţă;
- Autentificarea: este responsabil cu validarea indentităţii utilizatorilor;
- Autorizarea: se ocupă cu restricţionarea accesului utilizatorilor asupra anumitor părţi din sistem. Nerespectarea acestuia poate duce la probleme de siguranţă;

2.3.1 Reţele industriale prin fir

Pentru a garata permisiunea asupra datelor din diferitele părţi ale unui proces industrial, a trebuit să se creeze o legătură fizică între diferitele straturi ale acestuia. De la conectarea utilizatorilor între ei până la facilitarea accesului asupra întregului sistem, incluzând aici senzori, elemente de execuţie etc.

Atunci când vine vorba de o reţea de comunicare, există o serie de performanţe ce trebuie luate în considerare, în funcţie de tipul şi rolul acesteia şi anume: dimensiunea reţelei, numărul de dispozitive care pot fi conectate, lăţimea de bandă, timpul de răspuns, perioada de eşantionare, dar şi sarcina ce trebuie transferată. Pentru a înţelege mai bine legătura dintre componentele unui proces industrial, în figura 2.5 este exemplificată arhitectura unei astfel de reţele.

2.3.1.1 Reţeaua Senzor-Actuator

Principalele protocoale găsite la acest nivel sunt ***HART***(*Highway Addressable Remote Transducer*) ce transpune un semnal digital peste semnalul analogic 4-20 mA, având o viteză de 1200 bps şi ***ASi***(*Actuator Sensor Interface*) ce reprezintă o conexiune între actuatori şi senzori prin două fire. El constă într-un master căruia îi pot fi atribuiţi până la 31 dispozitive de tip slave cu o rată de scanare de 5-10 ms/IO.

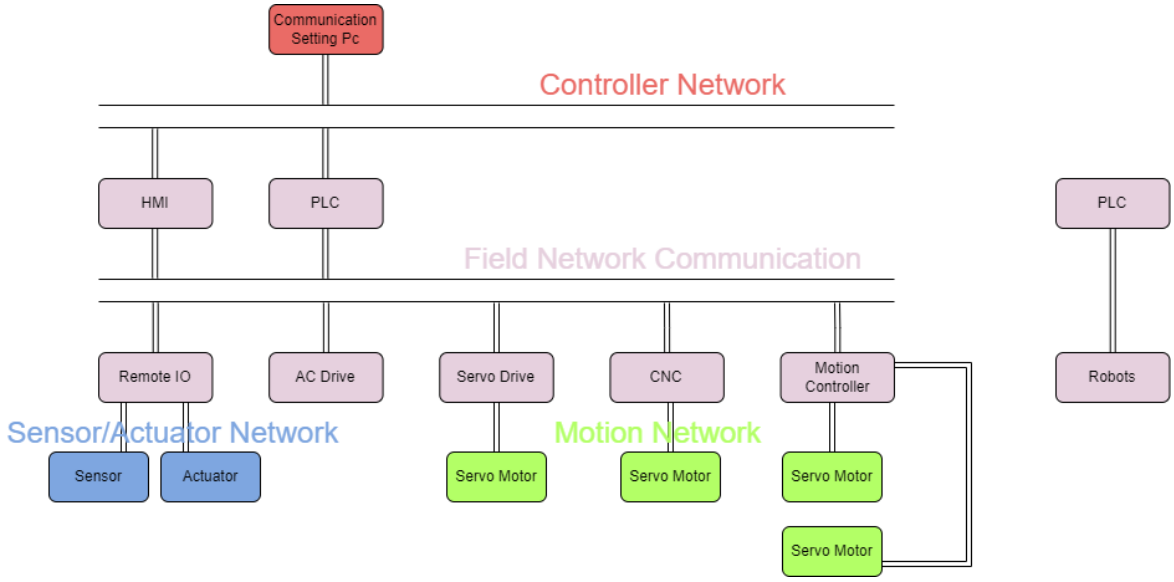

Figura 2.5: Arhitectura unei rețele industriale

2.3.1.2 Rețeaua Fieldbus

Există o multitudine de concepte care se întâlnesc la acest nivel, fapt datorat existenței pe piață a mai multor producători. Principalele protocoale sunt:

Profibus: Se găsește în mai toate industriile și suportă nenumărate configurații. Principalul avantaj al acestui protocol este ușurința folosirii și flexibilitatea. Necesită puțin *hardware* și cablaj, iar mentenanța se face rapid. Deoarece dispozitivele de tip *slave* sunt pasive, adică lucrează doar în momentul solicitării de către master, consumul de energie se minimiează. Permite diferite rate de transfer 9.6/19.2/45.45/93.75/187.5/500/1500/3000/6000/12000 kbps. Există doua tipuri de *master*, primul reprezentând un controler central ce schimbă informații la un interval de timp definit cu toate dispozitivele de tip *slave*, iar al doilea având ca scop monitorizarea, mentenanța sau diagnosticarea rețelei. Se împarte în două categorii, în funcție de locul unde se folosește: *Profibus DP*(*Decentralized Peripherals*) reprezintă varianta ce atinge vitezele maxime descrise mai sus, utilizat în automatizarea fabricilor și *Profibus PA*(*Process Automation*) care are o viteză de 31.25 kpbs și este utilizat în automatizarea proceselor. Deși există această împărțire, comunicația *Profibus* este *half-duplex*, ceea ce înseamnă că un singur dispozitiv comunică la un moment de timp. Ca și topologii, putem întâlni configurații de tip stea, inel sau serie.

DeviceNet: Este un protocol care are ca scop principal conectarea controlerelor industriale cu dispozitivele de intrare/ieșire, dar și între ele. Favorizează dispozivitele care nu necesită multă energie electrică să poată fi activate direct prin intermediul cablului de rețea. Folosește o topologie de tip magistrală, unde fiecare nod este reprezentat de către un dispozitiv. Datorită acestei configurații, defectarea unui nod nu afectează deloc celelalte noduri. Suportă până la 64 de noduri formate dintr-un singur master și 63 *slave* în același timp pe o singură rețea, cu o rată de transfer de 125/250/500 kbps;

ProfiNET: Spre deosebire de *Probibus* care este un protocol de tip *half-duplex*, *ProfiNET* este *full-duplex*, ceea ce înseamnă că comunicația se face în ambele sensuri în

acelaşi moment de timp. În acelaşi timp, timpul de ciclu poate fi ajustat pentru fiecare dispozivit în parte în funcţie de necesităţi. Viteza la care ajunge poate atinge 100 Mbps cu uşurinţă sau chiar 1 Gbps şi mai mult. Fiind un protocol bazat pe *Ethernet*, configuraţia se face foarte uşor, singurele condiţii fiind ca *switch*-urile să poată ţină pasul cu viteza acestuia şi să fie *full-duplex*. Principalele topologii sunt linie, arbore, stea, inel. *ProfiNET* acceptă şi dispozitive conectate prin *wireless* sau *bluetooth*, unde confidenţialitatea mesajelor este asigurată de *PROFIsafe* care nu este decât un strat de siguranţă aplicat peste protocolul principal.

***CAN*:** Este folosit pentru în special pentru transportul mic de date, dar foarte rapid care satisface cerinţele de timp real. Poate atinge viteze de 1 mbps şi este folosit preponderent în maşini, avioane etc. Este o tehnologie *half-duplex* cu mult superioară lui *RS232* şi mult mai ieftină. Deşi un protocol precum *TCP/IP* poate atinge viteze de sute de mbps, *CAN* se descurcă mult mai bine pe distanţe scurte, cu date de dimensiuni mici. Deasupra acestui strat a apărut tehnologia *CANopen* folosită în industriile medicale, de căi ferate, în agricultură, marină etc şi nu este decât un set de reguli de limbaj pentru datele transmise pe căile *CAN* [8].

Tabelul 1 conţine o comparaţie între cele descrise mai sus.

Tabel 1:Comparare protocoale de comunicaţie prin fir

Protocol	Mediu	Link	Reţea	Transport	Viteză	Nr. Noduri
Etheret/Ip	Ethernet	Ethernet	IP	TCP/UDP	10/100/1000 mbps	
PROFIBUS	RS-485, fibră optică	PROFIBUS Fieldbus			31.25-12000 mbps	32 normal sau 126 prin fibră optică
DeviceNet	CANbus	CAN bus	DeviceNet	DeviceNet	1 mbps	64
PROFINET	Ethernet	Ethernet	IP	TCP/UDP	10/100/1000 mbps	
HART	Fire de cupru	Conexiune electrică		Segmentare automată a datelor		
Modbus TCP/IP	Ethernet	Ethernet	IP	TCP port 502	10/100/1000 mbps	254
EtherCAT	Ethernet	Ethernet + Ethercat	IP+timing	TCP/UDP	10/100/1000 mbps	65535

2.3.1.3 Reţeaua Controller

Protocoalele de la acest nivel sunt de calitate înaltă şi se pot distinge în funcţie de clasele de timp real. Pentru clasa *soft real-time*, majoritatea se bazează pe *TCP(UDP)/IP* şi diferă în funcţie de producător: *MODBUS TCP/IP (Schneider), Ethernet/IP (Rockwell), High Speed Ethernet-HSE (Fieldbus Foundation), P-Net/Ip, Profibus (Siemens)* etc. În clasa a doua, *hard real-time*, găsim *PROFINET DP, Time-Critical Control Networ(Tcnet,*

Toshiba), iar în a treia clasă, *isochronous real-time Powerlink* care a fost dezvoltat pentru control de mişcare, *EtherCAT(Beckhoff), Profinet IO, Sercos III*.

2.3.2 Reţele industriale *wireless*

Reţelele industriale *wireless* se află într-un proces de adaptare şi de adoptare din ce în ce mai mare datorită uşurinţei de folosire, dar mai ales datorită evoluţiei industriei la *Industry 4.0.* Spre deosebire de reţelele prin fir, acestea, nu necesită o configuraţie foarte promptă, o întreţinere adecvată, majoritatea costurilor de cabluri şi arhitecturi dispar, iar consumul de energie devine substanţial mai mic. În acest context, au fost dezvoltate reţele de sensori fără fir(engl. *Wireless Sensor Network-WSN*) [10].

Reţeaua *WSN* a devenit un subiect de interes pentru mai multe domenii printre care agricultura, cel militar, sau medical etc. Intr-o reţea *WSN*, există mai multe noduri şi fiecare are o funcţie specifică de citire fie de parametrii de la sensori de umiditate, temperatură, presiune, fie de performanţele procesului în general. Deasupra acestui strat s-a dezvoltat reţeaua industrială de senzori fără fir(engl. *Industrial Wireless Sensor Network-IWSN*). Aceasta a moştenit multe aspecte de la reţeaua precedentă printre care cele mai importante find protocoalele de comunicaţie. Principalele diferenţe între *WSN* şi *IWSN* sunt următoarele:

Latenţa: Datorită timpului rapid de răspuns care este necesar într-un astfel de domeniu, latenţa trebuie să fie mică înspre inexistentă. Deoarece aceste reţele sunt responsabile cu monitorizarea parametrilor unui proces şi comanda lor în timp real, propagarea de informaţii trebuie să fie cât mai rapidă. Acest beneficiu vine, însă, şi cu un dezavantaj. Dispozitivele care se bazează pe acest tip de comunicaţie sunt bazate pe baterii şi se plasează, de obicei, în locuri greu accesibile. Prin urmare, viaţa bateriilor trebuie să fie cât mai lungă, ceea ce duce la o latenţă ridicată.

Mobilitate: Pentru a spori flexibilitatea şi mobilitatea, această reţea poate dispune de noduri mobile precum roboţi mobili sau vehicule automate. Acest aspect este în contrast cu reţeaua *WSN*, unde nodurile sunt fixe.

Mediul: Un alt aspect foarte important în ceea ce priveşte diferenţele dintre *IWSN* şi *WSN* este mediul în care sunt folosite. Spre deosebire de *WSN*, unde mediul este unul prietenos, reţeaua *IWSN* este întâlnită în medii ostile cu vibraţii şi temperaturi foarte mari, cu mult praf şi umiditate crescută. Prin urmare, aceste tipuri de mediu pot afecta într-o măsură ridicată, care poate duce la ulterioare erori, semnalele transmise. Pentru a asigura siguranţa împotriva interferenţelor, strategii care să asigure integritatea datelor trebuie aplicate.

Capacitatea: În domeniul industrial, fiecare dispozitiv emite si recepţionează o cantitate mare şi continuă de date. Reţeaua care facilitează acest transport, trebuie să atingă cerinţele necesare. Nodurile comunică nu doar cu vecinii, dar şi cu alte dispozitive conectate pentru atingerea scopurilor în general şi pentru a rezolta probleme complexe precum procesarea de date. Prin urmare, nodurile din reţelele *IWSN* sunt consumatoare mai mari de energie, au capacităţi mult mai mari de calcul şi memorie şi sunt de obicei mai deştepte decât nodurile dintr-o reţea *WSN*[7].

2.3.2.1 ZigBee

Este o rețea de tip mesh, unde fiecare nod este conectat cu unul sau mai multe noduri în același timp. Se bazează pe standardul *IEEE 802.15.4* care este axat în special pe controlul și monitorizarea parametrilor din mediul industrial, construcția și mentenanța caselor inteligente, dar și automatizarea sistemelor energetice. Principalele avantaje ale unei astfel de rețele sunt consumul redus de energie și suportul pentru configurarea acesteia în diferite topologii. Totuși, dacă numărul de noduri ajunge la un număr foarte mare, atingerea obiectivului în ceea ce privește timpul de ciclu poate deveni imposibilă [11]. Există trei tipuri de dispozitive: Coordonator, care este rădăcina rețelei și conține informații cu legătură la chei de securitate, iar în același timp este responsabilă cu conectarea rețelei la alte rețele; Router, care transmite date la alte dispozitive; Dispozitiv coadă(engl. *End device*), care reprezintă senzorul și comunică doar cu *router*-ul sau coordonatorul. Acesta nu poate comunica cu alte dispozitive din rețea. O variantă îmbunătățită a rețelei admite gruparea dispozitivelor vecine și stocarea de date. Viteza maximă la care poate ajunge este de 250 kbps, într-o configurare de până la 30 de metri.

2.3.2.2 Wireless HART

Este o extensie a protocolului *HART* special concepută pentru monitorizarea proceselor și controlul acestora. Aceasta se bazează pe standardul *IEEE 802.15.4* și are o configurație de tip *mesh* în care fiecare dispozitiv poate să transmită atât datele proprii cât și datele provenite de la alte dispozitive mai departe. Putem avea mai mule rețele *HART* în aceeași arie, deoarece fiecare mesaj transmis are un identificator unic. Fiecare nod are o listă cu vecinii lui care poate fi actualizată în momentul detectării unui defect. Securitatea acestui protocol este dată de criptarea datelor făcută cu o cheie simetrică de 128 de biți

2.3.2.3 UWB

Banda Ultralargă(engl *Ultrawideband-UWB*) este o tehnologie *wireless* de transmitere a datelor pe mici distanțe bazată pe transmiterea unor impulsuri emise în secvențe periodice. Nu se recomandă utilizarea acestui tip de rețea pe distanțe mari sau pentru măsurarea și monitorizarea parametrilor în medii foarte ostile, datorită undelor transmise.

2.3.2.4 IETF 6LoWPAN

Acest tip de comunicare se bazează în întregime pe *IP* versiunea 6(*IPv6*). Avantajele unei astfel de rețele sunt abilitatea de a comunica direct cu alte dispozitive conectate prin *IP*, dar și suportul care există pe piață în acest moment [10].

Tabelul 2 oferă o explicaţie mai concretă a specificatiilor fiecărei tip de reţea.

Tabel 2: Comparare Protocoale de comunicare Wireless

Protocol	Distanţă	Frecvenţă	Viteză	Caracteristici	Aplicaţii
ZigBee	~75m	2.4Ghz	250kbps	Suportă topologii de tip stea, arbore şi mesh	întrerupătoare wireless, echipamente industriale
WirelessHART	Max 225m	2.4 GHz	250kbps	Bazat pe 802.15.4, autoorganizator, sincronizat în funcţie de timp, arhitectură mesh	Dispozitive inteligente responsabile cu controlul proceselor industriale
6LoWPAN	Max 75m	2.4 Ghz ISM,915 MHz(NA),868 MHz(EU)	250kbps	Pot fi conectate multe dispozitive în funcţie de IP	Senzori wireless, case inteligente, producţie automată
UWB	Max 200m	3.1-10.6 GHz	110mbps	Acurateţe mare	Localizare

2.4 Aplicaţii întâlnite în industrie

Datorită gradului mare de manevrabilitate de care sunt capabili, roboţii industriali se întâlnesc în mai toate industriile din zilele noastre. Printre primele operaţii la care au fost supuşi se enumeră sudatul, manipularea materialelor, vopsitul, paletarea, mutatul obiectelor, asamblarea diferitor componente, tăierea, polişarea sau lipirea. În toate aceste aplicaţii, controlul milimetric al robotului este absolut necesar, iar prin urmare, trebuie gândite strategii care să îndeplinească aceste condiţii. Este aproape imposibil să prezicem fiecare poziţie prin care robotul să treacă până să îndeplinească *task*-ul cerut, iar de aceea, robotul trebuie echipat cu inteligenţă suficientă care să faciliteze întreg procesul. Aici poate fi vorba de senzori de poziţie, de atingere sau chiar de camere inteligente care să îndrume robotul pe traictoriile necesare. Luând ca exemplu un proces de sudare a două componente între ele, pe o traictorie fixă, putem programa robotul să execute o mişcare liniară pe marginea lor.Dar ce facem dacă dimensiunea pieselor se schimbă? Va trebui să reprogramăm robotul la fiecare mică modificare a componentelor. Acest aspect se poate observa în mai toate procesele enumerate mai sus, unde o mică modificare a materialelor folosite poate genera o problemă uriaşă în ceea ce priveşte controlul robotului. Pentru a evita astfel de situaţii şi a combate aceste restrângeri, în cazul exemplului explicat, o cameră inteligentă poate fi folosită. Aceasta scanează componentele care necesită modificări şi transmite mai departe parametrii de mişcare robotului. Ca rezultat, robotul va ştii mereu între ce puncte trebuie să se deplaseze, dar şi cu ce viteză. Totodată folosirea unei camere inteligente vine cu dezavantajele ei, deoarece ea trebuie calibrată, unghiul din care se ia

imaginea trebuie să fie perpendicular pe piese, iar dacă nu, datele obţinute trebuie modificate, lumina trebuie să fie bună pentru a avea măsurători cât mai precise etc.

O altă aplicaţie întâlnită într-o măsură foarte mare în industrie, este operaţia de paletare, în care robotul trebuie să sorteze, să mute şi să adune materialele de acelaşi tip la o viteză foarte ridicată deoarece timpul este foarte important în astfel de procese. De obicei aceste materiale vin pe o bandă transportoare, iar sortarea lor se face în funcţie de greutate, dimensiuni, culori etc. Toate aceste caracteristici pot fi delimitate cu ajutorul senzorilor sau a camerelor. În majoritatea aplicaţiilor în care materialele mânuite se schimbă des, trebuie aplicaţi algoritmi complexi de mişcare pentru satisfacerea condiţiilor.

2.4.1 Aplicaţii de tipul *Pick and Place*

Reprezintă cea mai răspândită aplicaţie a roboţilor industriali în zilele noastre şi se realizează cu ajutorul unui braţ robotic montat de o suprafaţă plană ce ia materialul dintr-o parte şi îl mută în altă parte. Materialul se poate afla într-o poziţie fixă ceea ce face ca punctul de ridicare să poată fi invăţat de către robot sau poate fi în mişcare, de exemplu pe o bandă transportoare, caz în care este nevoie de dispozitive inteligente care să ghideze robotul la poziţia exactă din care se face ridicarea. Integrarea roboţilor în astfel de operaţii duce la maximizarea productivităţii, unii dintre ei procesând chiar şi 200 de produse pe minut. Aplicaţiile de tipul *pick and place* nu reprezintă doar mutarea obiectelor dintr-o parte în alta, ci asamblarea diferitelor componente, sortarea, împachetarea şi despachetarea acestora. Aceste operaţii nu ar fi, însă posibile fără un element de prindere potrivit. În funţie de materialele cu care avem de a face, trebuie să alegem un end-effector corespunzător care să respecte condiţiile de finețe sau manevrabilitate ale elementelor.

2.5 Optimizarea aplicaţiilor de tipul *Pick and Place*

Cea mai simplă astfel de aplicaţie întâlnită în practică este ridicarea şi mutarea materialelor dintr-un singur loc, într-un singur loc. Astfel robotul nu va trebui să execute decât o deplasare între două puncte, iar în cazul în care spaţiul este restrâns, câteva mişcări intermediare vor fi adăugate pentru evitarea coliziunilor. În procesele mai complicate există mai multe puncte de *pick* şi mai multe puncte de *place*, iar în funcţie de domeniul în care se lucrează pot exista mai multe variante de optimizare care să crească nivelul producţiei. Câteva metode de optimizare le reprezintă folosirea algoritmilor euristici în care se ia în vedere minimizarea timpului sau a distanţei parcurse de către robot. Totodată aceste tehnici nu duc întotdeauna la o soluţie perfectă, ci mai degrabă la găsirea unui minim local. În vederea perfecţionării soluţiei, algoritmi metaeuristici pot fi aplicaţi. Aceştia vin cu o îmbunătăţire faţă de euristicii clasici deoarece pot fi aplicaţi pe orice problemă de optimizare, unde datele cunoscute sunt limitate sau puterea computaţională este mică. Câţiva algoritmi de genul sunt algoritmii genetici(engl. *Genetic Algorithm*), optimizarea cu colonii de furnici (engl. *Ant Colony Optimization*),optimizarea cu roiuri de particule (engl. *Particle Swarm Optimization*) etc.

În lucrarea [12] este dezbătută o astfel de problemă de optimizare folosindu-se un algoritm metaeuristic de optimizare bazat pe căutare locală iterată(engl. *Iterated Local Search*) şi pe programare întreagă (engl. *Integer Programming*) pentru un proces de fabricare al *PCB*-urilor. Scopul este minimizarea distanţei parcurse în vederea ridicării si mutării componentelor prin găsirea unei secvenţe optime şi minimă ca durată de *pick* şi *place*. Pentru a înțelege mai bine ideea, în figura 2.6 este prezentată o reprezentare grafică a intrepretării unui *PCB*. Punctele constituie locuri în care urmează să se monteze componente.

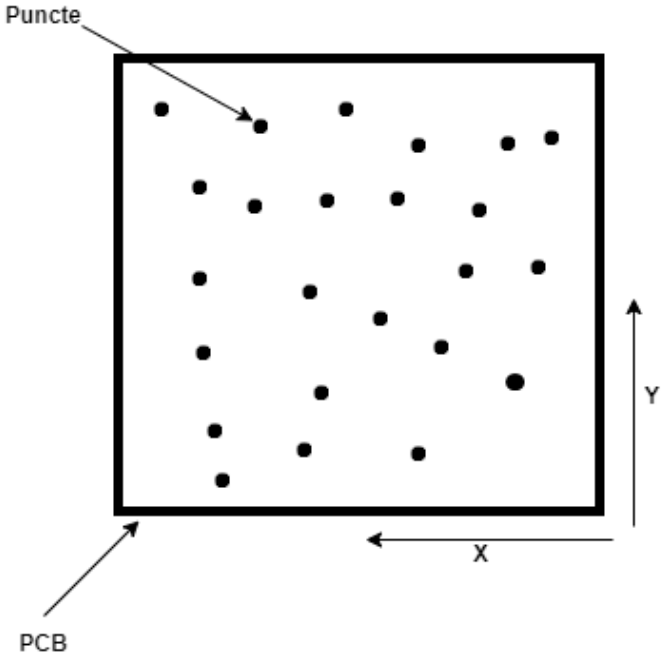

Figura 2.6: Localizarea punctelor de plasare a componentelor pentru minimizarea distanţelor parcurse de robot

PCB-ul este fixat pe o masă de lucru, iar coordonatele fiecărui punct sunt încărcate în sistem. Mai departe, un element de execuţie ia piese de pe un alimentator şi le potrivește în dreptul fiecărui punct. Acest pas se repetă până fiecare componentă este aşezată la locul ei. Se pot ridica mai multe decât o singură componentă în acelaşi timp.

Pentru a reprezenta rezultatele într-un mod concis, au fost aleşi şi aplicaţi alţi doi astfel de algoritmi: genetic şi memetic. Rezultatele arată, însă, că algoritmul de optimizare bazat pe căutare locală iterată s-a descurcat cel mai bine, scoţând cel mai bun timp şi dovedindu-se capabil să lucreze cu structuri de date foarte mari.

3 Aparatură hardware utilizată

3.1 Controler Logic Programabil *Beckhoff CX5140-0155*

Controlerul *CX5140-0155* face parte din familia computerelor încorporate *CX5100*. Numele acestui model este dat de configuraţia *hardware*, dar şi *software* de care dispune. Acesta vine echipat cu un procesor *Intel Atom* de *1.9 GHz* cu *4* miezuri ce îi oferă o putere mare de calcul. Sistemul de operare ce rulează este un *Windows 10 IoT Enterprise, LTSB (64-Bit)* special conceput pentru astfel de sisteme întegrate, iar aplicaţiile concepute pot fi încărcate, rulate şi administrate direct în mediul integrat *Twincat 3 XAR(eXtended Automation Runtime)*, independent de sistemul de operare.

3.1.1 Privire de ansamblu asupra arhitecturii

Toate modelele ale acestei familii, *CX51x0*, au aceeaşi arhitectură prezentată în figura 3.1. Unitatea centrală de procesare mai conţine un controler de memorie şi unul grafic. Memoria *RAM* poate fi de *2GB* sau de *4GB DDR3* şi nu poate fi modificată după ce modelul a fost produs. Există două interfeţe *PCIe* pentru comunicaţii, patru *USB*, o *DVI*, o interfaţă serială, una *SATA* pentru memorie şi una *SD*. Controlul comunicaţiei este realizat de două procesoare *Intel i1210*, câte unul pentru fiecare port.

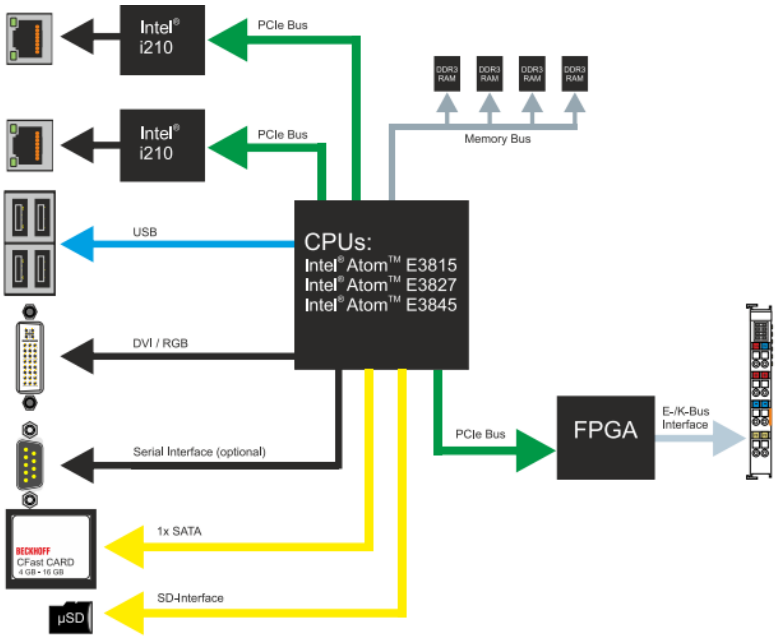

Figura 3.1:Arhitectura modelului familiei de plc-uri CX51x0 [13]

Cele patru intrări *USB* sunt folosite în vederea conectării unui mouse sau a unei tastaturi. Interfețele de *Ethernet* sunt independente și nu există *switch*-uri integrate. Profilul acestor interfețe poate fi modificat în multe moduri, dar în fabrică, ele se configurează pentru comunicația *EtherCAT*. Amândouă interfețele pot atinge viteze de 10/100/100 mbps. Primul led indică starea comunicării, dacă există o conexiune, culoarea lui este verde. În momentul transferului de date, acesta pâlpâie. Al doilea led indică viteza transferului de date. Dacă este 10 mbps, nu este aprins, la 100 mbps este verde, iar la 1000 mbps culoarea devine roșie. Înterfața *DVI* este potrivită pentru conectarea unui monitor. Rezoluția depinde de distanța la care se află, iar maximul este de 5 m. Totuși există și extensii care pot duce această distanță până la 50 m.

3.1.2 1-second UPS

1-second UPS este un condensator ce continuă să alimenteze procesorul în momentul unei pane de curent. În acest interval scurt de timp, datele persistente pot fi salvate și menținute până la eventuala revenire a curentului, când vor fi incărcate în sistem, iar procesul poate continua. Flow-ul unui astfel de eveniment este descris în figura 3.2

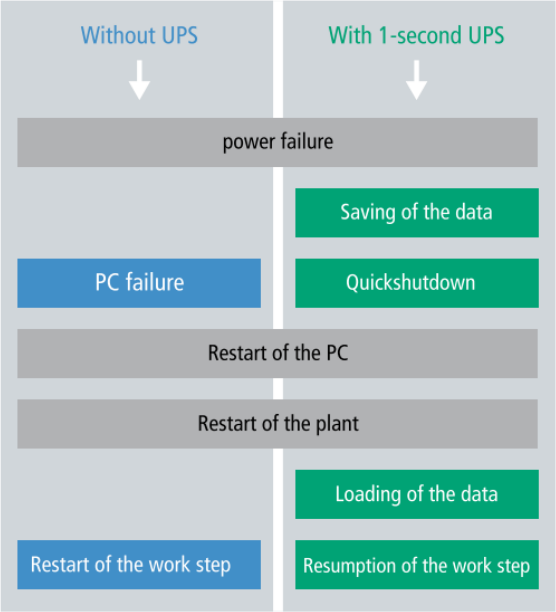

Figura 3.2: Comportamentul plc-ului în cazul unei pene de curent [13]

Acest sistem este construit pentru a dura întreaga viață a controlerului, iar pe durata acestui timp, condensatorul se va degrada. Prin urmare, este recomandată salvarea a maxim 1 MB de date. Variabilele persistente sunt declarate în momentul creării aplicației ce urmează să fie încărcată în memoria *plc*-ului și sunt salvate pe cardul de memorie aferent, de unde sunt și reîncărcate la repornirea sistemului.

3.1.3 Mediul de dezvoltare integrat Twincat 3

Datorită complexității mari a aparaturii moderne și în același timp a necesității de a reduce timpul dedicat controlării lor, programarea modulară este din ce în ce mai folosită în practică. Ea constă în declararea mașinilor, a funcțiilor sau a ansamblurilor ca module, fiecare fiind cât se poate de independente și structurate ierarhic. Scopul acestei

practici este, bineînteles, reutilizarea codului, uşurinţa scalării acestuia, dar în acelaşi timp şi întelegerea lui. *Software*-ul *Twincat 3* reprezintă miezul sistemului de control şi permite o astfel de programare el împărţindu-se în *eXtended Automation Engineering(XAE)* şi *eXtended Automation Runtime(XAR)*. El poate transforma aproape orice calculator personal într-un computer capabil de operaţii în timp real.

Twincat XAE permite simplificarea ingineriei *software* prin integrarea a altor limbaje de programare sau *tool*-uri precum *MATLAB, Simulink, C++, C, C#*. Spre exemplu, utilizatorii pot realiza sisteme de control folosind *Matlab*, transformate mai apoi automat în module pentru *Twincat*. În acelaşi timp, *Twincat* este integrat ca o extensie în mediul de dezvoltare integrat (engl. *Integrated Development Environment-IDE) Microsoft Visual Studio*.

Twincat XAR oferă un mediu în timp real unde modulele pot fi încărcate, executate şi administrate. Modulele nu trebuie create folosind acelaşi compiler, dar totuşi ele sunt tratate la fel. Aceste module sunt apelate ciclic prin intermediul *task*-urilor sau prin intermediul altor module. Limita numărului de *task*-uri este 65000, dar în general, depinde de resursele dispozitivului. Un alt aspect important al acestui *software*, este capabilitatea de a împărţi *task*-uri pe diferite miezuri ale procesorului (acolo unde resursele *hardware* permit)[13].

3.2 Robot *Stäubli TS2-80*

3.2.1 Controlerul *CS9*

Controlerul *CS9* este un computer specializat în controlul tuturor roboţilor produşi de *Stäubli*. O privire de ansamblu asupra lui este reprezentată în figura 3.3. Acesta este format din 3 componente principale.

Figura 3.3: Controler CS9 [14]

Principala componentă reprezintă partea de alimentare a braţului robotic. Aceasta conţine o placă de control numită *STARC9* ce acţionează ca un translator între

limbajul de programare şi semnalele codificatoarelor pentru mişcarea braţului. În acelaşi timp el conţine şi amplificatoare pentru fiecare motor.

A doua componentă este computer-ul propriu zis ce conţine unitatea centrală de procesare, dar şi mai multe interfeţe de comunicare printre care *Robot Safety Interface RSI9*, folosită pentru selectarea modului de lucru şi securizarea intrărilor şi al ieşirilor, o interfaţă pentru conectarea pendantului capabilă de o viteză de 100 mbps, o interfaţă *USB* şi una de *Ethernet*.

A treia componentă conţine alimentarea de la reţea, un *switch* de pornire/oprire, mai multe siguranţe şi câteva leduri pentru indicarea stării actuale a controlerului.

3.2.2 Braţul robotic *TS2-80*

Braţul robotic este format dintr-o bază, partea superioară a braţului, cot, antebraţ şi şurubul cu bile, precum în figura 3.4. Acestea reprezintă trei elemente de rotaţie şi un element de translaţie. Datorită acestui tip de configuraţie, poate atinge viteze foarte mari şi este preferat în procesele de asamblare.

Figura 3.4: Braţul robotic TS2-80

Unele limitări când vine vorba de astfel de roboţi stau în raza de acţiune pe care pot lucra şi în greutatea pe care o pot manvevra, ea nedepăşind 10 kg.

3.2.3 Teach Pendant pentru braţul robotic TS2-80

Pendant-ul robotului *Stäubli TS2-80* dispune de o interfaţă grafică cu ecran tactil şi de mai multe butoane de control precum în figura 3.5. Butoanele din partea stângă sunt folosite pentru nagivarea prin interfeţele dispozitivului(acasă, aplicaţii, I/O etc.), în timp ce cele din dreapta sunt folosite pentru controlul robotului(pornirea mişcării, modificarea vitezei şi controlul fiecărui ax). Acesta mai dispune de un buton de oprire de urgenţă care, odată apăsat, opreşte orice mişcare, iar pe partea din spate de un buton de activare a dispozitivului. Interfeţele *pendant*-ului permit selectarea profilurilor,

selectarea aplicaţiilor ce dorim să ruleze, observarea şi analizarea erorilor, controlul ieşirilor, mişcarea manuală şi automată, setarea limbii, vizualizarea detaliilor *hardware* şi *software* ale robotului(sistem de operare, timp de ciclu, parametrii de siguranţă etc.) şi controlul frânelor fiecărui ax. În acelaşi timp el permite şi programarea robotului prin metoda "*teaching*", prin care se salvează în memorie poziţia anumitor puncte prin care robotul trece. Există şi o funcţie de *back-up* a datelor, dar şi a aplicaţiilor.

Figura 3.5: Teach Pendant [14]

3.2.4 Mediul de dezvoltare integrat Stäubli Robotics Suite

Stäubli Robotics Suite este un mediu de programare ce rulează pe sistemul de operare *Windows* şi simplifică gestionarea robotului. El permite dezvoltarea, simularea şi depanarea aplicaţiilor ce urmează să fie încărcate în memoria robotului şi utilizate în producţie. Suportă evaluarea conceptelor de automatizare a roboţilor într-un mediu intuitiv prin utilizarea interfeţelor 3D, chiar şi fără experienţă în programare şi simularea diferitelor scenarii întâlnite în producţie. Programarea se face utilizând limbajul *VAL3* care este dedicat roboţilor *Stäubli*. Acesta este un limbaj de nivel înalt ce combină funcţionalităţile de bază ale unui limbaj de programare în timp real cu funcţionalităţi specifice roboţilor industriali [14].

3.3 Tehnologia de comunicaţie *EtherCAT*

EtherCAT(Ethernet Control Automation Technology) este un protocol de comunicare bazat pe *Ethernet*, dezvoltat de compania *Beckhoff Automation* special pentru controlul industrial şi standardizat în *IEC 61158*. Acesta foloseşte modelul *master/slave*, în care există un singur dispozitiv de tip *master* şi mai multe dispozitive de tip *slave*, iar comanda se face unidirecţional. Deoarece este fondat pe *Ethernet*, aceasta foloseşte cadrele şi stratul fizic descrise în *IEEE 802.3 Ethernet Standard*. Principalele îmbunătăţiri aduse sunt minimizarea timpului de răspuns, acurateţea datelor, uşurinţa integrării şi costul redus al implementării.

3.3.1 Principiul de funcţionare al tehnologiei EtherCAT

Principiul de funcţionare al tehnologiei *EtherCAT* este simplu. *Master*-ul trimite o telegramă care trece prin fiecare nod legat la reţea. Fiecare dispozitiv de tip *slave* citeşte datele care îi sunt adresate în timp ce acestea îl intersectează şi îşi scrie propriile date în cadru(engl. *Frame*). Dispozitivul de tip *master* este singurul căruia îi este permis să trimită *frame*-uri de date, dar nodurile au capabilitatea de a le pasa mai departe către alte noduri precum în figura 3.6.

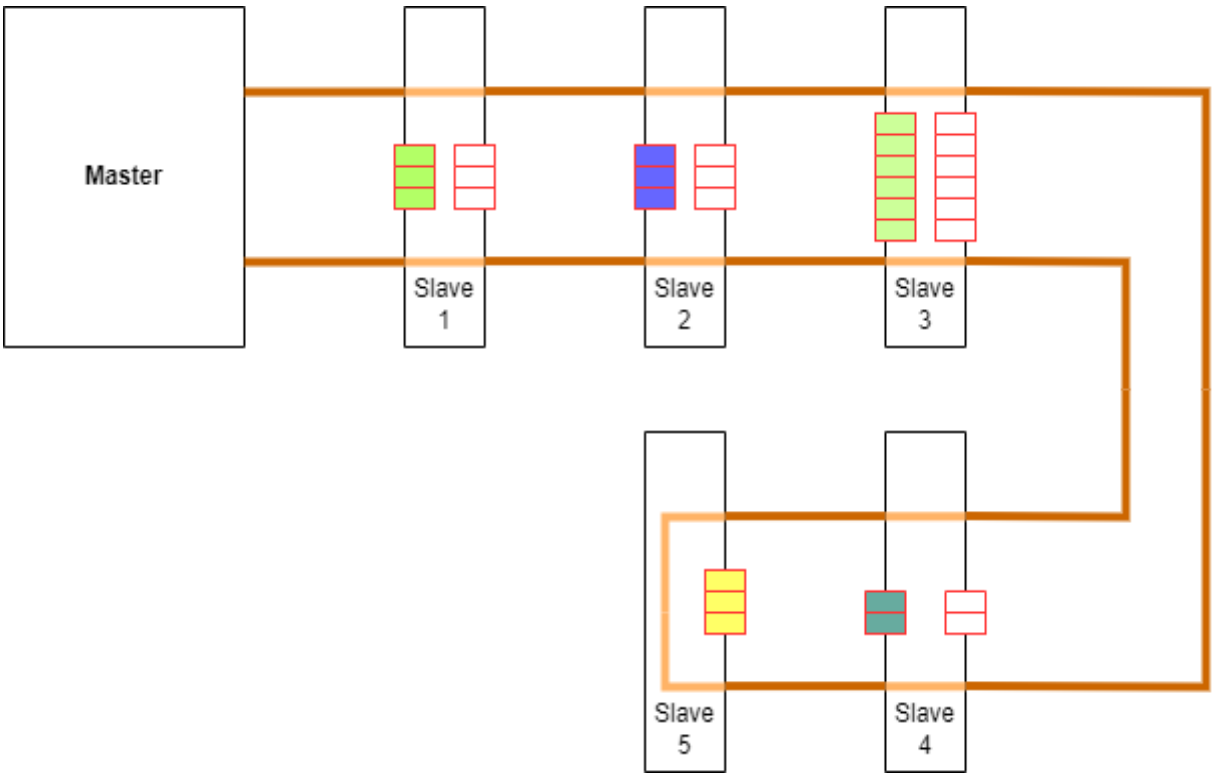

Figura 3.6: Principiul de funcţionare al tehnologiei EtherCAT

Dispozitive dintr-o reţea *EtherCAT* au de obicei două porturi *Ethernet*: la primul fiind conectat cablul de la nodul precedent, iar la al doilea cablul către nodul următor. Ultimul nod detectează terminarea reţelei datorită lipsei conectiviăţii nodului următor şi întoarce datele către *master*. Această abordare asigură operare în timp real şi evită întârzierile. Totodată, acest aspect poate fi privit ca un dezavantaj, deoarece foarte multe dispozitive aflate pe piaţă în momentul de faţă nu dispun de resursele necesare pentru a ţine pasul cu timpul de ciclu foarte mic al tehnologiei *EtherCAT*. Prin urmare, este posibil ca reţeaua să fie încetinită pentru ca fiecare dispozitiv să poată face schimbul de date necesar.

Master-ul foloseşte doar un controler de acces media(engl. *Media Access Controller-MAC*) fără altă aparatură *hardware*, ceea ce face ca el să poată fi reprezentat de către orice dispozitiv care dispune de un port *Ethernet*. Dispozitivele de tip *slave* au nevoie de un controler adiţional numit *Ethercat Slave Controller(ESC)* pentru a primi şi a trimite date către *frame*, în timp ce acesta este în mişcare.

Într-o rețea *EtherCAT* pot fi conectate până la 65535 noduri, sub diferite topologii precum stea, inel, linie, magistrală etc. prin intermediul terminalelor de joncțiune. În acelaşi timp, distanţa pe care o astfel de rețea se poate întinde nu este o problemă, deoarece prin aceste joncțiuni se poate trece de la cabluri normale de *Ethernet* la cabluri de fibră optică ce se pot întinde până la 500 km. Viteza unei astfel de rețele este influențată bineînțeles de numărul de noduri, iar limita superioară atinge 100 mbps.

3.3.2 Ceasuri distribuite

Într-un proces complicat, care este format din mai multe componente(elemente de execuție, senzori etc.) având ca scop controlul sau acolo unde acțiuni simultane trebuie efectuate, este important ca nodurile să opereze având o legătură strânsă de timp între ele, adică să fie sincronizate. Un exemplu constă în setarea a mai multor ieşiri simultan, independent de momentul în care staţia primeşte acele date. Termenul de simultan este relativ, în funcţie de proces. La unele acesta poate însemna o gamă de 0-10 ms, în timp ce la altele, limita este 0-100 μs .

Sincronizarea nodurilor într-un astfel de proces se efectuează cu ajutorul ceasurilor distribuite(engl. *Distributed Clocks-DC*). Această metodă nu necesită *hardware* adiacent şi minimiează *jitter*-ul până la valori mult mai mici de 1 μs. În timp ce structura de date trece prin fiecare nod, în momentul primirii de către acesta, el returnează o înregistrare a timpului curent ce se ataşează de cadru. Acest proces se întâmplă şi în momentul în care datele se întorc către *master*. Aceste înregistrări sunt mai apoi procesate de *master*, iar rezultatul reprezintă cantitatea te timp ce trebuie adăugată pentru a compensa întrârzierile fiecărui nod. Dacă toate nodurile sunt sincronziate, ele pot culege date provenite de la intrări şi pot modifica ieşiri în acelaşi timp.

3.3.3 Profile de comunicare

Există doua tipuri de comunicare ce se pot realiza într-o rețea *EtherCAT*. Prima dintre ele este comunicarea ciclică(*PDO communication*) dintre *master* şi toate dispozitivele *slave* în timp real, iar a doua se bazează pe mesaje transmite aciclic între dispozitive(*SDO communication*). *Master*-ul trimite o comandă nodurilor, iar acestea trimit înapoi un răspuns.

Pentru a permite conectarea unei mari varietăți de dispozitive, au fost stabilite câteva profile de comunicare: *CoE(CANopen over EtherCAT), SoE(Servo Drive over EtherCAT), FoE(File Access over EtherCAT) și EoE(Ethernet over Ethercat)*. Aceste profile permit implementarea protocoalelor *CANopen, Sercos, Ethernet* într-o rețea *EtherCAT*. Mai multe profile pot exista în acelaşi sistem, iar un nod slave nu trebuie să suporte toate aceste profile. El poate alege care i se potrivește mai bine şi anunță master-ul printr-un fişier de descriere(engl. *Device Description File*).

3.3.4 EtherCAT State Machine

Starea fiecărui dispozitiv de tip *slave* este controlată prin intermediul maşinii de stare. În funție de aceasta, diverse operaţii sau funcţii sunt accesibile sau executabile de către dispozitiv. În figura 3.7 este descrisă maşina 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.

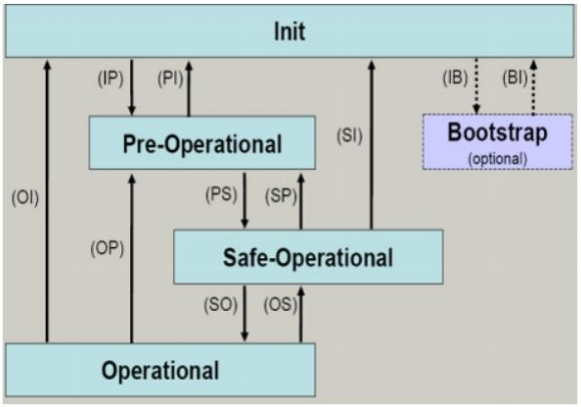

Figura 3.7: Maşină de stare EtherCAT [15]

După pornire, toate dispozitivele slave trec în starea *INIT*. În această stare, nu există niciun fel de comunicare. *Master*-ul iniţializează canalele managerului de sincronizare pentru diferitele profiluri de comunicare. În următoarea stare, *Pre-Operational*, doar comunicaţia *SDO(Service Data Objects)* este posibilă, iar *master*-ul configurează canalele managerului de sincronizare pentru comunicaţie *PDO(Process Data Objects)*. În acelaşi timp, parametrii specifici terminalului, care pot diferi de setările implicite, sunt transferate către *master*. În timpul tranziţiei de la *Init* la *Pre-Operational*, dispozitivele slave verifică dacă comunicaţia aciclică a fost iniţializată corect. În a treia stare, *Safe-Operational*, amândouă formele de comunicare sunt posibile. Totuşi, dispozitivele *slave* păstrează ieşirile într-o zonă de siguranţă, doar intrările fiind updatate ciclic. În timpul tranziţiei dintre *Pre-Operational* şi *Safe-Operational*, dispozitivele slave verifică dacă managerul de sincronizare pentru comunicţia *PDO* şi ceasurile distribuite sunt configurate corect. În starea *Operational*, toate tipurile de comunicare sunt posibile, iar în *Bootstrap*, doar comunicaţia *FoE* este posibilă, în cazul updatării *firmware*-ului unui nod *slave* [15].

4 Analiză, proiectare, implementare

În vederea atingerii obiectivelor setate în capitolul de introducere al acestei lucrări şi pentru a exemplifica detaliile teoretice discutate în capitolele 2 şi 3, am ales să creez o aplicaţie de tipul *pick and place* care să reprezinte într-un mod intuitiv modul de configurare şi funcţionare al computerelor logice programabile şi al roboţilor din mediul industrial. La modul general, aplicaţia este compusă din două programe principale, unul creat pentru *plc*-ul *Beckhoff CX5140-0155*, în mediul de programare *Twincat 3*, iar celălalt creat pentru robotul *Stäubli* cu 4 axe, în mediul *Stäubli Robotics Suite*.

4.1 Crearea şi configurarea aplicaţiei în *Twincat 3*

Pentru a putea folosi acest program, un fişier executabil trebuie mai întâi descărcat de pe *website*-ul dezvoltatorilor. Acesta reprezintă un *wizzard* care ghidează utilizatorii prin tot procesul de instalare.

După instalarea şi lansarea programului, o interfaţă precum cea din figura 4.1 va fi deschisă.

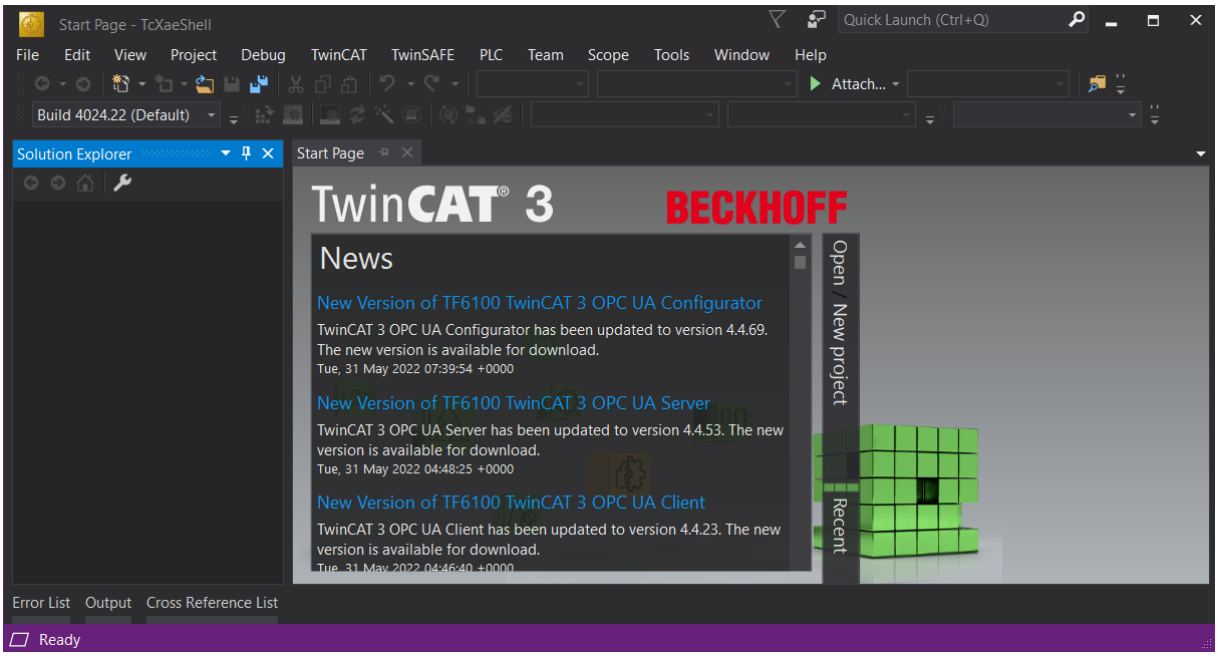

Figura 4.1: Interfaţă de pornire Twincat 3

Crearea unui proiect nou se face accesând *tab*-ul *File->New->Project* sau prin folosirea *shortcut*-ului *Ctrl+Shift+N*. Mai departe alegem din meniul din stânga *TwinCAT*

Projects şi selectăm singura variantă disponibilă, dăm un nume, alegem directorul în care dorim să salvăm şi apăsăm *OK* precum în figura 4.2.

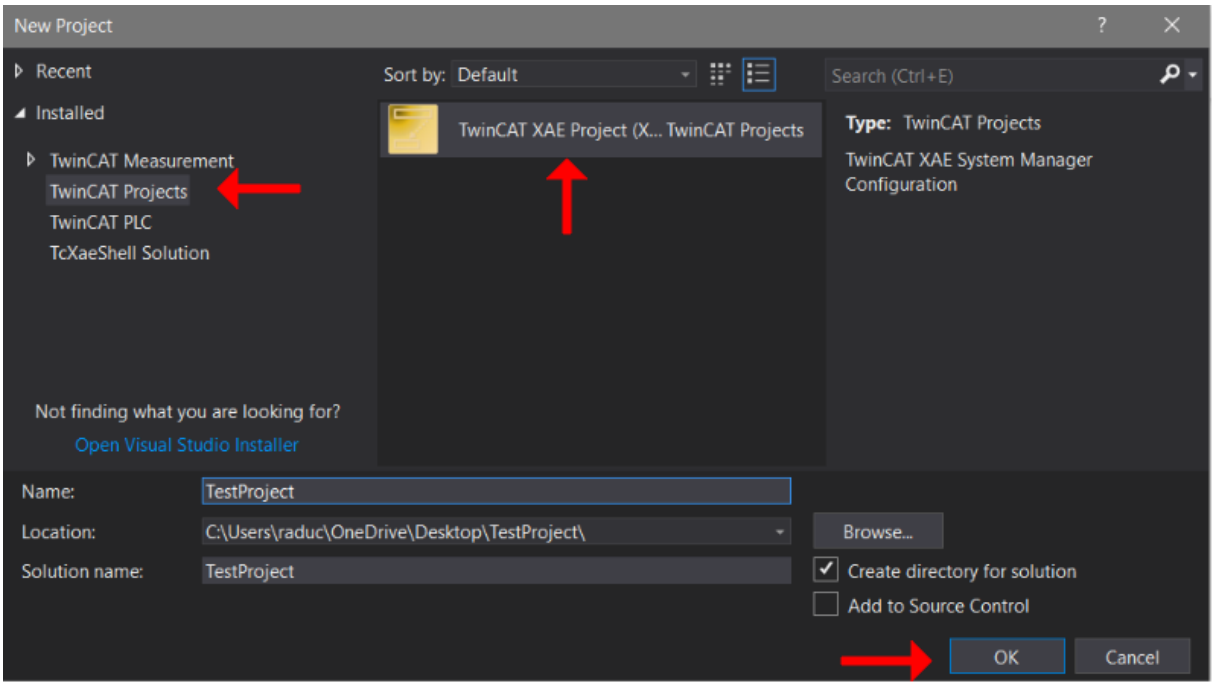

Figura 4.2: Crearea unui nou proiect nou în Twincat 3

Pentru ca aplicaţia să fie funcţională şi să poată rula, câteva setări trebuie efectuate în ceea ce priveşte partea de conexiune cu *PLC*-ul, configurarea comportamentului în timp real şi administrarea licenţelor folosite. Aceste setări pot fi efectuate accesând *tab*-ul *SYSTEM* din figura 4.3.

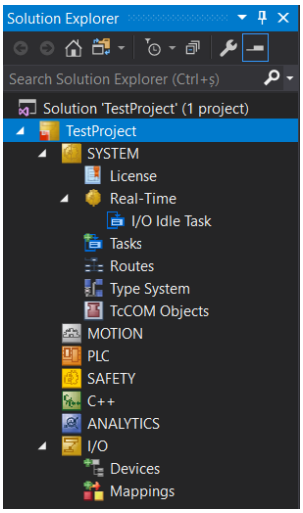

Figura 4.3: Solution Explorer în programul Twincat 3

Datorită componentei *XAR(eXtended Automation Runtime)* a programului *Twincat 3*, aplicaţia poate fi rulată atât local, pe computerul propriu, cât şi pe un *plc*. Conexiunea dintre computerul de lucru şi *plc* se realizează printr-un cablu de *Ethernet* direct în portul X000 al *plc*-ului. După conectarea fizică dintre acestea, o legătură *software* trebuie efectuată. Asta se face accesând *tab*-ul *SYSTEM* şi apăsând butonul *Choose Target..* din figura 4.4

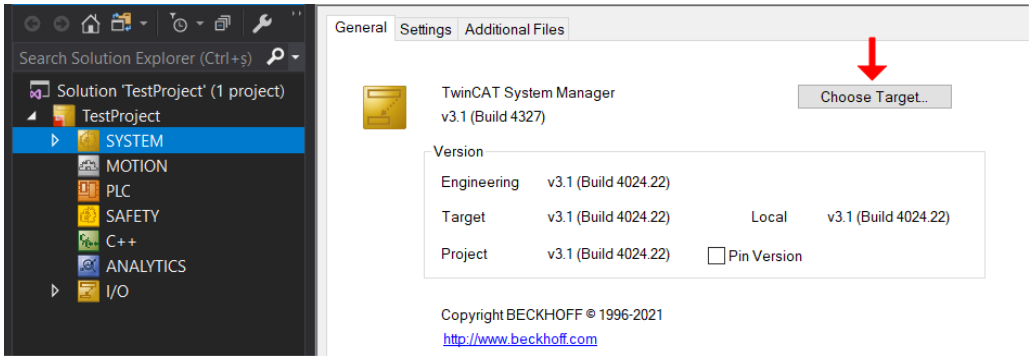

Figura 4.4: Tab-ul SYSTEM din cadrul Solution Explorer

După apăsarea butonului afişat, se va deschide o interfaţă pe care vom da *click* pe butonul de *Search(Ethernet)..->Broadcast Search* şi selectăm adaptorul pe care dorim să se realizeze căutarea(*wireless* sau prin cablu). În urma căutării *plc*-ul va apărea în lista din figura 4.5. Putem vedea numele, adresa, versiunea de *Twincat* care rulează şi sistemul de operare. Pe coloana *Connected*, dacă în dreptul numelui apare un x, înseamnă că conexiunea a fost realizată cu succes, iar în acest moment aplicaţia noastră poate fi descărcată şi rulată direct pe *plc*.

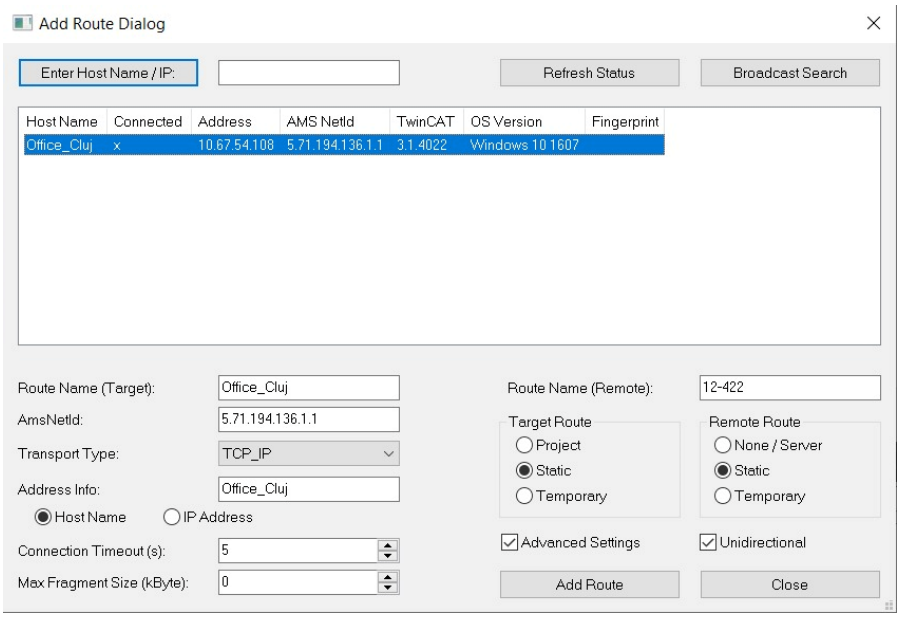

Figura 4.5: Adăugarea unei rute noi şi conectarea computerului local cu plc-ul

În *tab*-ul *License* se pot genera diferite licenţe pentru *toolbox*-urile de care avem nevoie în funcţie de complexitatea şi scopul aplicaţiei. Un exemplu îl reprezintă figura 4.6. Generarea licenţelor se realizează pe o perioadă de 7 zile şi se pot reînnoi.

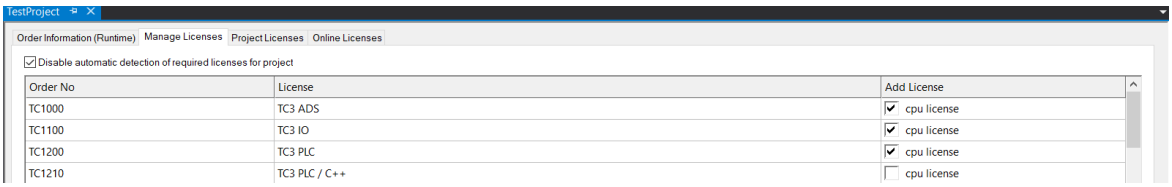

Figura 4.6:Selectarea licenţelor dorite pentru aplicaţia din Twincat 3

În *tab*-ul *Real-Time* se configurează proprietăţile de timp real ale aplicaţiei. În funcţie de posibilităţile *hardware* ale echipamentelor, avem posibilitatea să setăm pe ce miezuri ale procesorului dorim ca *task*-urile aplicaţiei să ruleze. Acest aspect este evidenţiat în figura 4.7. Se utilizează funcţia *Read From Target*, care detectează automat, în funcţie de sistemul ales(local sau *plc*) numărul de miezuri ale procesorului. În acelaşi timp, atunci când aplicaţia rulează pe propriul calculator, se recomandă utilizarea unui miez izolat(engl. *Isolated*). Putem izola un miez folosind propretăţile de *share/isolated* din secţiunea *Available Cores*.

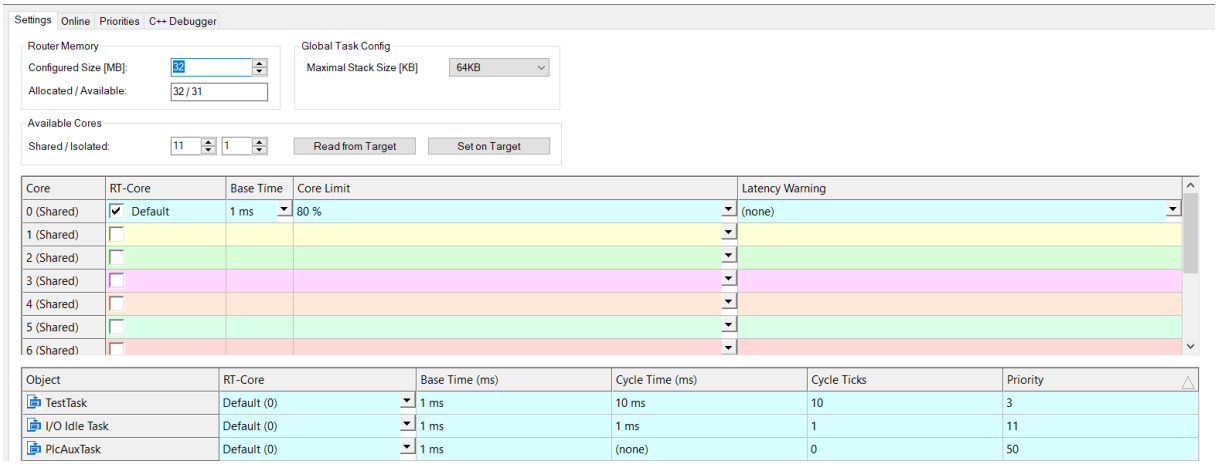

Figura 4.7: Configurare proprietăţilor de timp real ale aplicaţiei

În secţiunea *Tasks* putem configura numele *task*-urilor, pe ce porturi rulează, timpul de ciclu, dar şi prioritatea acestora. Aceste aspecte apar în partea de jos a figurii anterioare.

Următorul pas este crearea programului propriu-zis în care se realizează logica funcţionării aplicaţiei. Acest lucru se face dând *click* dreapta pe *tab*-ul *PLC->Add New Item* sau prin folosirea *shortcut*-ului *Ins*. O interfaţă precum cea din figura 4.8 va fi deschisă. Aici alegem *Standard PLC Project*, dăm un nume aplicaţiei şi apăsăm *Add*.

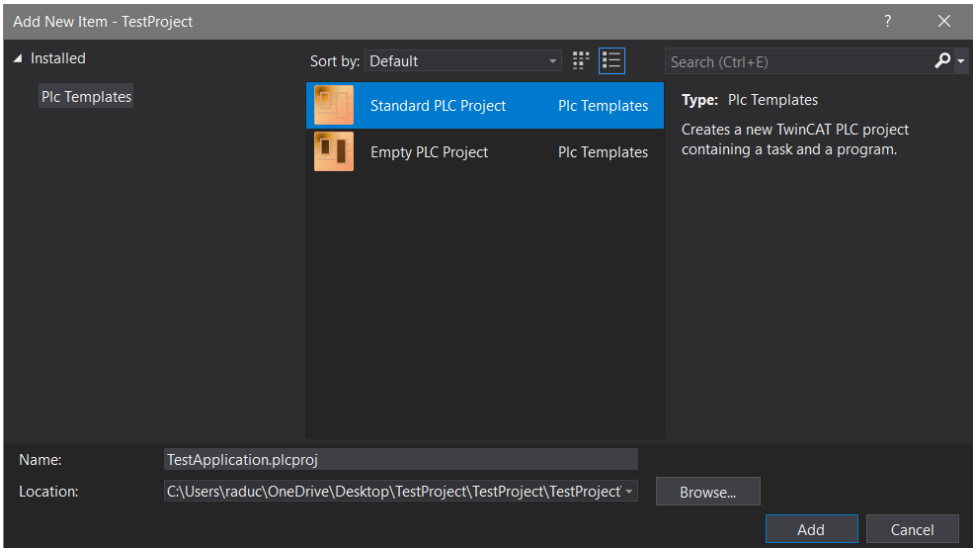

Figura 4.8: Crearea unei aplicaţii pentru plc folosind Twincat 3

După crearea aplicației, structura acesteia va arăta precum în figura 4.9.

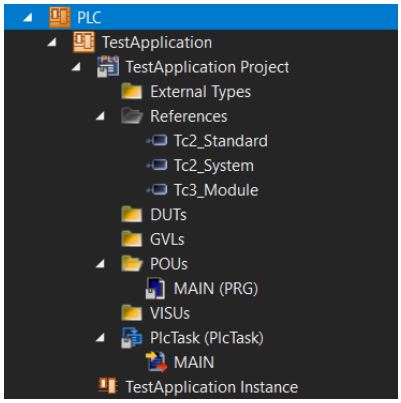

Figura 4.9: Structura unui Program plc

În directorul *References* găsim trei librării standard necesare oricărui proiect. Acolo pot fi adăugate și altele în funcție de necesitățile fiecărei aplicații în parte. Directorul *DUTs* conține tipuri de date, *GVLs* liste de variabile globale, *POUs (Program Organization Units)* programele dezvoltate, *VISUs* vizualizări, iar *PlcTask* reprezintă *task*-ul pe care aplicația rulează.

Tab-ul *I/O* conține dispozitivele conectate. Pentru a ne folosi de tehnologia de comunicare *EtherCAT* un *master* trebuie adăugat. Acest lucru se face dând click dreapta pe *Devices->Add New Item*. Se selectează *EtherCAT->EtherCAT Master* precum în figura 4.10 și se apasă *Ok*.

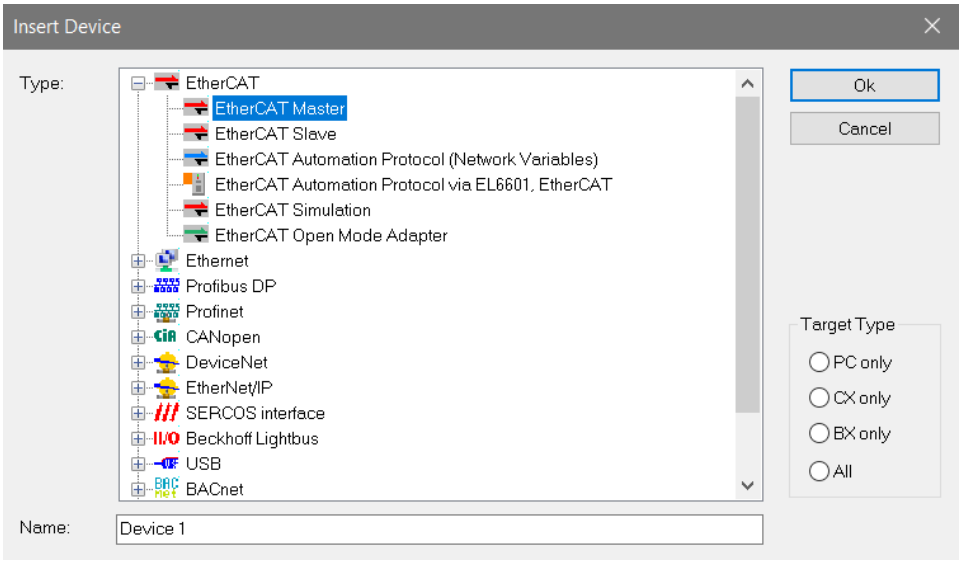

Figura 4.10: Adăugarea unui dispozitiv EtherCAT Master

După înserarea dispozitivului de tip *master*, pentru a găsi toate dispozitivele de tip *slave* din rețea se dă *click* dreapta pe *Device 1 (EtherCAT)* și se selectează *Scan*. *Twincat* efectuează o căutare și returnează o listă cu toate componentele găsite în figura 4.11. Fiecare *Term* reprezintă o cartelă atașată fizic *plc*-ului având diferite scopuri. *Term 11* reprezintă terminalul prin care se realizează comunicația dintre *plc* și robotul *Stäubli*. Totodată, nu toate dispozitivele de tip *slave* pot fi găsite prin această căutare, iar prin urmare ele trebuie adăugate manual. În 4.12 sunt cartelele fizice.

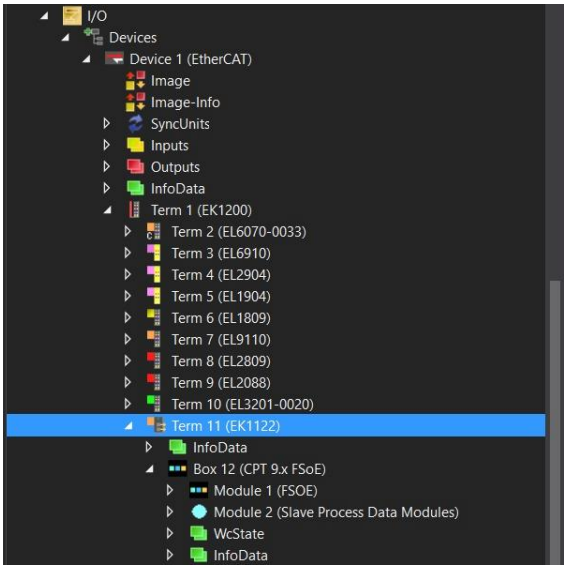

Figura 4.11: Rezultatul scanării dispozitivelor de tip slave din Twincat 3

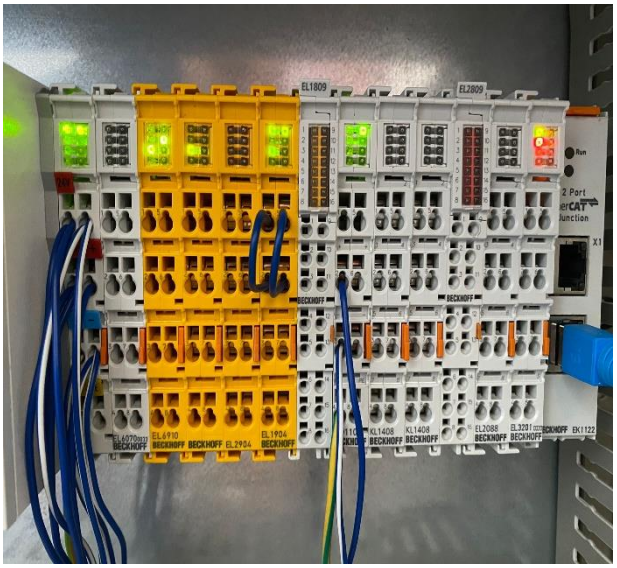

Figura 4.12: Terminale conectate fizic precum în figura 3.6

Conectarea robotului la *plc* se face cu ajutorul joncțiunii *EtherCAT* la care se leagă controlerul dispozitivului prin intermediul interfeței *Ethernet*, portul *J207/J208* descris în capitolul 3. Acesta este cofigurat ca dispozitiv *slave*. Configurarea intrărilor și al ieșirilor se face printr-un fișier *ESI(EtherCAT Slave Information)* ce descrie modul în care datele sunt schimbate între controlerul robotului și *plc*. Un exemplu de fișier *ESI* este în figura 4.13.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <EtherCATModule xmlns:xsi="http://www.w3.org/2001/XMLSc
3    <Vendor>
4      <Id>#x00000584</Id>
5      <Name>Stäubli Faverges SCA</Name>
6      <ImageData16x14>424D360300000000000036000000280000
7    </Vendor>
8    <Modules>
9      <Module>
10       <Type ModuleClass="Staeubli_CPT9x_customer_module
11       <Name>CPT 9.x RE/ECS V4.2</Name>
12       <!-- SRC_IOS -->
13       <RxPdo Sm="2" Fixed="true" Mandatory="true">
14         <Index>#x1601</Index>
15         <Name>1 Slave Process Data RxPDO</Name>
16         <Entry>
17           <Index>#x7D00</Index>
18           <SubIndex>1</SubIndex>
19           <BitLen>1</BitLen>
20           <Name>1 Byte Out (0)_Byte_0_Bit_0</Name>
21           <DataType>BIT</DataType>
22         </Entry>
23         <Entry>
24           <Index>#x7D00</Index>
25           <SubIndex>2</SubIndex>
26           <BitLen>1</BitLen>
27           <Name>1 Byte Out (0)_Byte_0_Bit_1</Name>
28           <DataType>BIT</DataType>
29         </Entry>
```

Figura 4.13: Fișier EtherCAT Slave Information(ESI) pentru Robot

Acest fișier se salvează în directorul *C:\TwinCAT\3.1\Config\Io\EtherCAT* și este selectat în momentul configurării modului de comunicare dintre *plc* și robot. Un exemplu îl reprezintă figura 4.14, în care variabile din program sunt legate de variabile descrise în fișierul *ESI*.

>Name		Online	Type	Size	Address	In/Out	User ID	Linked to
1 Byte Out (0)_Byte_0_Bit_0	X	0	BIT	0.1	82.0	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_0 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_1	X	0	BIT	0.1	82.1	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_1 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_2	X	0	BIT	0.1	82.2	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_2 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_3	X	0	BIT	0.1	82.3	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_3 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_4	X	0	BIT	0.1	82.4	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_4 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_5	X	0	BIT	0.1	82.5	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_5 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_6	X	0	BIT	0.1	82.6	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_6 . POU Outputs . Setup Instanc
1 Byte Out (0)_Byte_0_Bit_7	X	0	BIT	0.1	82.7	Output	0	PRG_Cell.Instance_Robot.Out_0_Byte_0_Bit_7 . POU Outputs . Setup Instanc
1 Byte Out (1)	X	0x00	BYTE	1.0	83.0	Output	0	PRG_Cell.m_ResetDone . POU Outputs . Setup Instance . Setup

Figura 4.14: Legătura dintre variabilele din program și fișierul ESI

În acest moment, toate configurațiile *hardware* și *software* sunt efectuate corect și putem trece la partea de implementare a aplicației.

4.1.1 Organizarea proiectului

Ideea aplicației, precum am explicat și în primul capitol, constă în crearea unui proces de *pick and place*, format dintr-un braț robotic și mai multe stații simulate cu ajutorul unor *timer-e*, care au funcții predefinite. Prin urmare, în ceea ce privește stațiile de lucru, am ales să creez patru tipuri de stații numite: *Conveyor, Drilling, Test* și *Metalization*. Diagrama fluxului de lucru este reprezentată în figura 4.15. Componente 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.

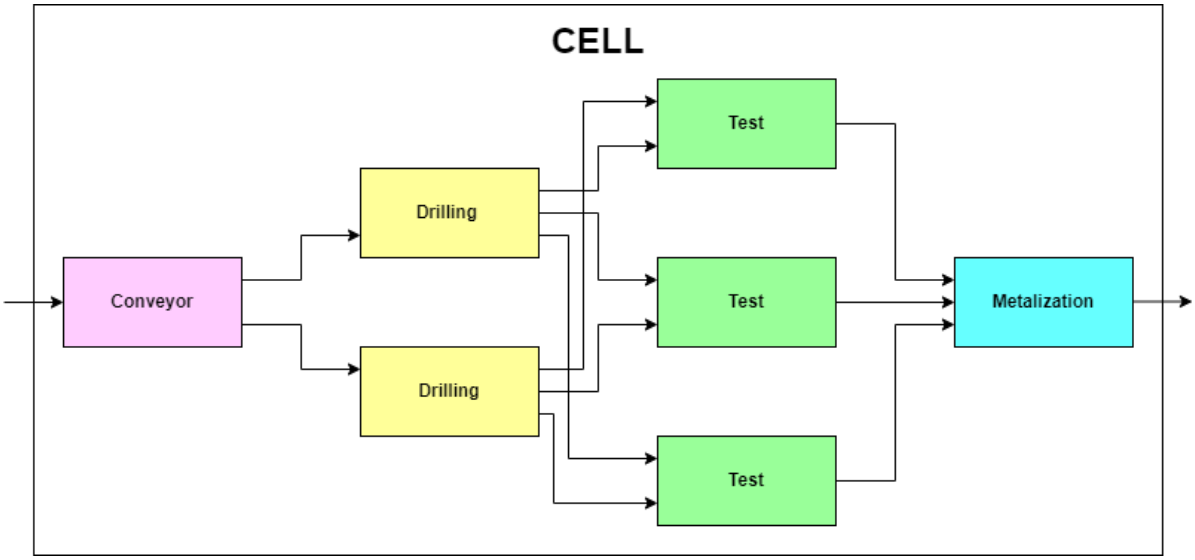

Figura 4.15: Fluxul de lucru al aplicației

Pentru programarea acestor stații am ales să folosesc limbajele de programare *Structured Text(ST)* și *Sequencial Function Chart(SFC)* datorită flexibilității și vitezei de executare a comenzilor. Fiecare stație este construită pe baza programării modulare descrise în capitolul 3 și se bazează pe funcții bloc pentru a implementa funcționalitatea ei. În figura 4.16 sunt arătate principalele componente ale aplicației.

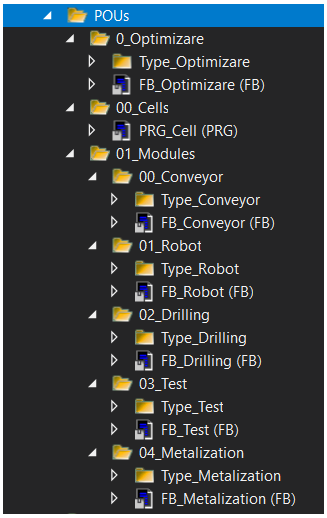

Figura 4.16: Componentele aplicației pentru plc

Directoarele cu numele *Type* reprezintă structuri de date folosite pentru a realiza comunicațiile dintre funcțiile bloc aferente. Înserarea unei funcții bloc noi se face dând *click* dreapta pe directorul de care dorim ca aceasta să aparțină *Add->POU*. Dăm un nume reprezentativ, alegem limbajul folosit și apăsăm *Ok* precum în figura 4.17.

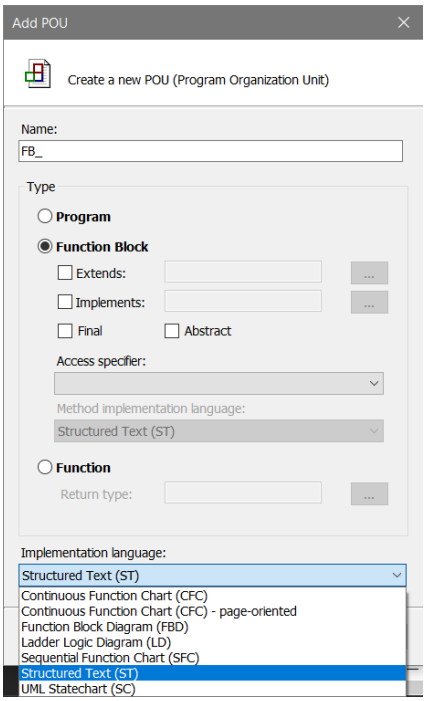

Figura 4.17: Adăugarea unei noi funcții bloc în programul plc-ului

4.1.2 Logica aplicației dezvoltate

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. După cum am spus și mai sus, transportul componentelor dintr-o stație de lucru în alta este asigurat de brațul

robotic. Pentru ca sistemul să nu se blocheze, o primă restricţie 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 staţie de fiecare tip în ordinea corectă. În figura 4.18 este reprezentat modul de organizare al staţiilor şi legăturile care există între acestea. Funcţiile bloc *FB_Conveyor, FB_Drilling, FB_Test, FB_Metalization* comunică cu *FB_Robot* care la rândul ei comunică cu robotul *Stäubli* folosind tehnologia *EtherCAT* discutată în capitolul 3.

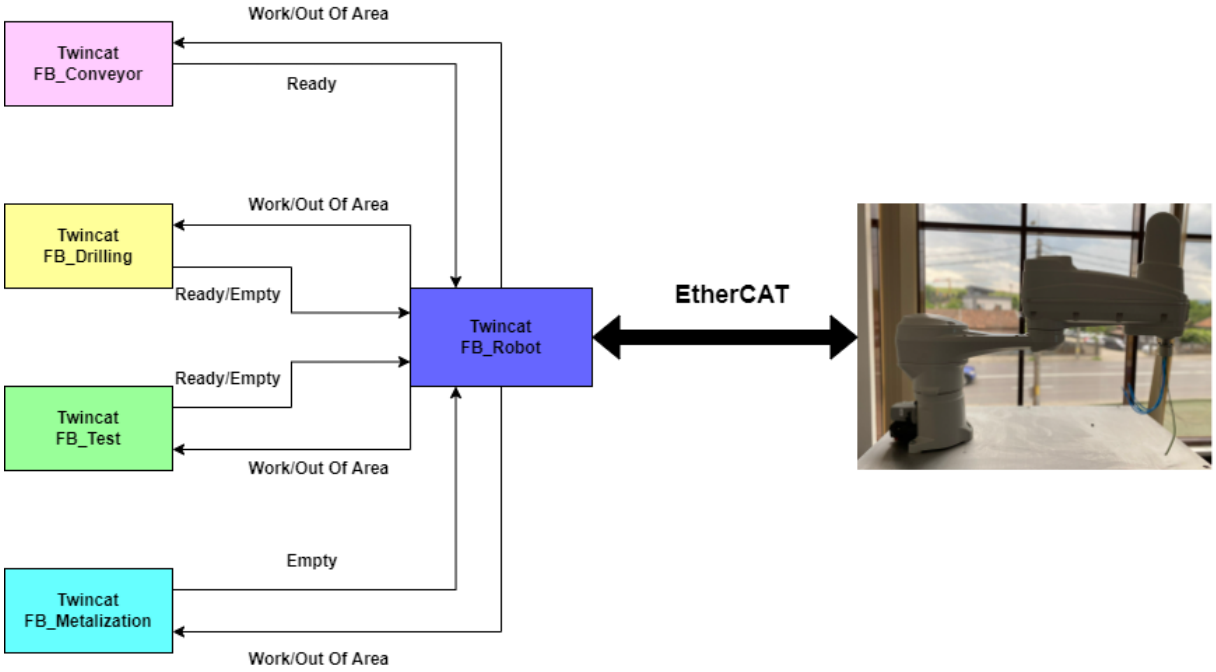

Figura 4.18: Organizarea aplicaţiei şi modul de comunicare

Primul pas după pornirea sistemului este iniţializarea staţiilor de lucru. Acestea verifică poziţia robotului folosindu-se de variabilele *Work/Out of Area*, iar dacă acesta nu se află în zona lor de acţiune, modulul *Conveyor* poate începe să aducă piese. Toate celelalte module intră într-o stare de *stand-by* până când robotul va aduce componentele ce necesită prelucrări în raza lor de acţiune, iar acestea vor fi detectate. Atunci când modulul *Conveyor* are o piesă gata, el semnalează robotul prin variabila *Ready*. Robotul vine, ia piesa şi o transportă la următorul modul, *Drilling*. De la *Drilling* la o staţie de *Test*, iar de la *Test* la modulul final, *Metalization*, după care piesa iasă din sistem. Staţiile pot lucra în acelaşi timp, ceea ce înseamnă că putem avea maxim numărul de staţii piese concomitent în sistem. Legătura dintre variabilele de comunicare este făcută prin intermediul celulei de lucru în care staţiile sunt instanţate şi pornite. Modul de lucru al unei staţii este bazat pe limbajul de programare *SFC(Sequencial Function Chart)*, iar un exemplu îl reprezintă figura 4.19, în care este descrisă diagrama de flux a funcţiei bloc *FB_Conveyor*. Toate celelalte staţii se bazează pe acelaşi principiu. Fiecare pas reprezintă o acţiune, în cazul ăsta, scrisă în *Structured Text(ST)* şi are un nume reprezentativ pentru partea de care se ocupă. Toate aceste acţiuni rulează continuu, mai puţin acţiunile de intrare/ieşire care, în figura aferentă, sunt reprezentate de un „X”.

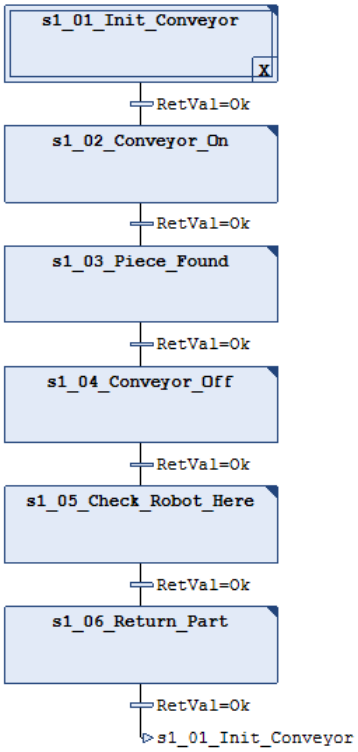

Figura 4.19: Modul de lucru al funcţiei bloc FB_Conveyor

4.1.3 Funcţia bloc *FB_Robot*

Această funcţie este cea mai importantă din toată aplicaţia, deoarece ea se ocupă cu coordonarea robotului *Stäubli* şi monitorizarea tuturor staţiilor din sistem. Stările fiecărui modul sunt monitorizate în timp real şi, în funcţie de acestea, decizii cu privire la locaţiile de unde se vor ridica sau lăsa piese sunt luate. Algoritmul de decizie, în cazul în care dorim să luăm o piesă dintr-o staţie, în vederea transportării acesteia către următoarea staţie se bazează pe ordine şi este prezentat în mai jos. Pentru celelalte staţii procesul este similar, adăugându-se mai multe condiţii pentru determinarea scopului deplasării, dacă e pentru a ridica piesa sau pentru a o lăsa.

Algoritmul 4.1 Calcularea următoarei mişcări

```
1: Intrări: conveyorState, drillingState, pickedOk
2: Ieşiri: 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
```

Comunicarea dintre *plc* şi robotul fizic se face prin intermediul unor semnale digitale şi analogice. Funcţia *FB_Robot* comunică cu robotul fizic prin intermediul acestor semnale descrise în fişierul *ESI*. Pentru a anunţa robotul că o piesă este gata într-un anumit modul, o variabilă de tip *boolean* se face *true*. Robotul citeşte acea variabilă şi ştie

că trebuie să meargă în poziția aferentă. În momentul în care acesta ajunge în poziția respectivă, semnalează *plc*-ul prin intermediul unei variabile *boolean* că a ajuns. *Plc*-ul dă permisiunea că poate să ridice piesa, acesta o ridică, anunță și pornește către următorul modul.

Aplicația dispune de două moduri principale:

- *Automat*: În modul automat tot procesul se desfășoară independent, fără a fi nevoie de intervenția din afară asupra lui și este pornit prin intermediul unei interfețe grafice;
- *Reset*: În modul reset, atunci când este activat, robotul așteaptă ca piesele din fiecare stație să fie procesate, iar mai apoi le preia și le transportă într-o locație intermediară. De asemenea, el este pornit prin intermediul interfeței grafice;

În cadrul acestei funcții, pentru o calitate ridicată a aplicației, am ales să folosesc variabile de tip *flag*. Acestea sunt predefinite în mediul de programare *Twincat 3*. Principalele *flag*-uri folosite sunt *SFCInit, SFCReset, SFCError, SFCCurrentStep. SFCInit* este o variabilă de tip *boolean* care atunci când devine adevarată, diagrama *SFC* corespunzătoare funcției bloc în care *flag*-ul a fost declarat trece în pasul de inițializare și rămâne blocată acolo, iar toate celelalte *flag*-uri sunt resetate. Procesul se reia din pasul de început în momentul dezactivării variabilei. *SFCReset* se bazează pe același principu, exceptând partea de blocare a procesului. El doar duce sistemul în pasul de început. Pentru o bună funcționalitate a codului se obișnuiește ca pe *step*-urille dintr-o diagramă *SFC* să se introducă limite de timp, minime și maxime. Dacă aceste limite sunt trecute sau avem o pauză în sistem, o variabilă de tip *SFCError* se activează. Odată activată, ea trebuie resetată dacă dorim să ne putem folosi din nou de ea. *SFCCurrentStep*, după cum îi spune și numele, este o variabilă de tip *string* ce ține minte numele *step*-ului activ la un moment de timp. Totuși, atunci când avem ramuri paralele, se afișează mereu numele pasului cel mai din dreapta.

4.1.4 Crearea unei vizualizări în Twincat 3

Pentru a facilita înțelegerea programului conceput, am ales să creez o vizualizare care să reprezinte cu exactitate ceea ce se petrece în proces. Ea conține două led-uri pentru fiecare mod în parte, *automat* și *reset* și încă unul aferent funcției de optimizare a procesului. Acestea sunt aprinse dacă variabilele *boolean* corespunzătoare sunt adevărate și sunt stinse în caz contrar. Mai conține șase indicatoare pentru fiecare instanță de stație. Ele ne arată *step*-ul activ al fiecărui modul. Atunci când o piesă este gata pentru a fi ridicată într-o anumită stație, culoarea indicatorului devine verde. Poziția robotului este indicată printr-o săgeată deasupra fiecărui indicator de stare aparținând unei stații. Interfața vizualizării poate fi observată în figura 4.20. Ea conține și două imagini reprezentative și un indicator pentru numărul de componente realizate de-a lungul funcționării procesului. Adăugarea acestor componente a fost făcută prin intermediul *toolbox*-ului pus la îndemână de către dezvoltatorul programului *Twincat 3*.

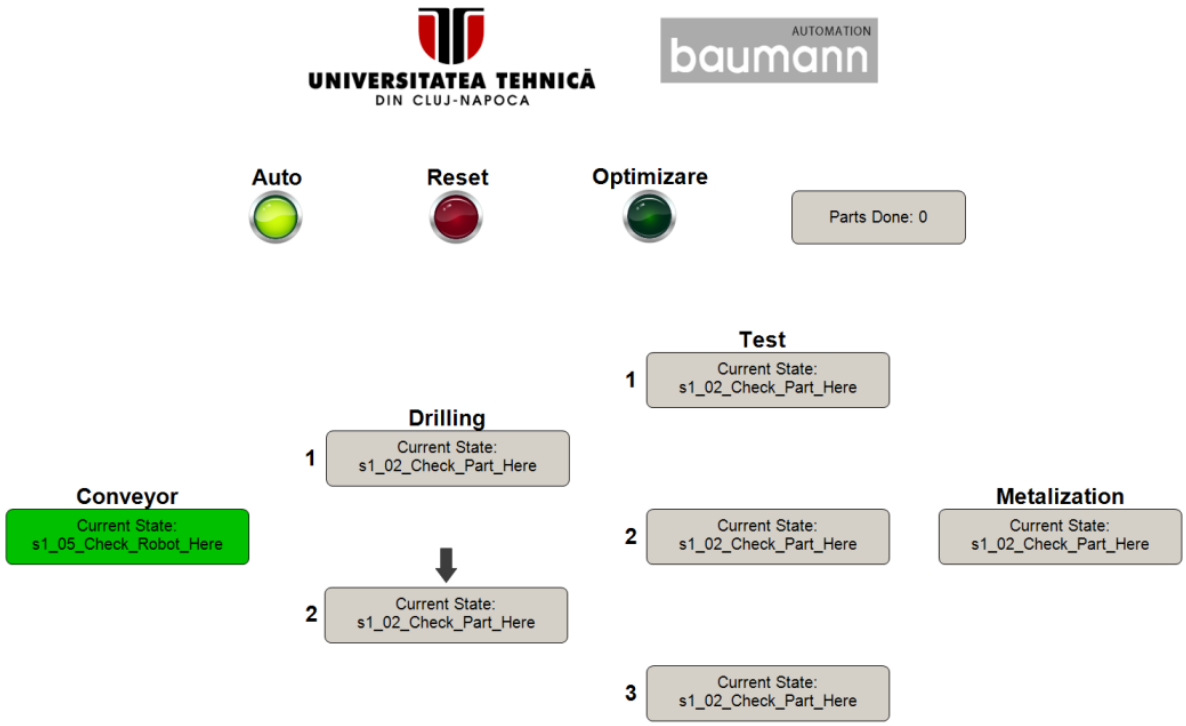

Figura 4.20: Interfață grafică Twincat 3

Variabilele folosite pentru controlul afișării acestei vizualizări se actualizează la un timp de ciclu de aproximativ 200 ms datorită existenței unui *task* separat de *task*-ul principal al aplicației, pe care aceasta rulează. Prioritatea acestuia este, bineînțeles, mai mică decât prioritatea *task*-ului principal.

Crearea unei vizualizări se face dând *click* dreapta pe directorul de care dorim ca aceasta să aparțină *Add->Vizualization*. Mai departe dăm un nume reprezentativ și bifăm căsuța corespunzătoare instalării librăriei necesare pentru a dispune de obiecte vizuale predefinite(led-uri, butoane, casete text etc.), iar mai apoi apăsăm *Open* precum în figura 4.21.

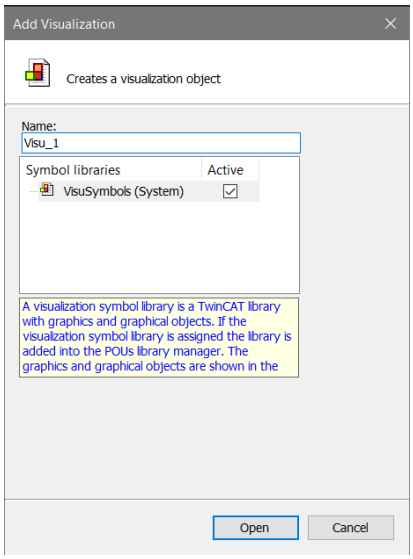

Figura 4.21: Crearea unei vizualizări în Twincat 3

Odată cu crearea unei vizualizări, se creează şi un *Vizualization Manager*, care este responsabil cu setările generale petru toate obiectele de tip vizualizare. Vizualizarea poate fi de trei tipuri, în funcţie de locul în care rulează: Vizualizare *integrată*-pe computerul pe care aceasta a fost creată, *PLC HMI*-pe un computer terţ, *PLC HMI Web*- în *browser*. Toate obiectele folosite în cadrul interfeţei sunt declarate în standardul *IEC6131-3* ca fiind funcţii bloc. Pentru ca vizualizarea să ruleze pe un computer terţ sau pe *web*, dăm click dreapta pe managerul de vizualizări şi selectăm *TargetVisualization* pentru primul mod, iar *WebVisualization* pentru cel de al doilea. Indiferent de cazul ales, numele vizualizării create trebuie adăugat în secţiunea *Start Visualization*. Totodată putem alege dacă interacţiunea cu obiectele din vizualizare să se facă prin intermediul unui ecran tactil sau prin intermediul tastaturii.

4.1.5 Erori întâlnite în decursul implementării

Erorile prezentate în figura 4.22 apar datorită configurării greşite a proprietăţilor de timp real atunci când aplicaţia este rulată local, dar nu numai. Rezolvarea lor constă în efectuarea unei scanări a numărului de miezuri ale procesorului şi configurarea unui miez izolat pe care *task*-urile aplicaţiei să activeze, în cazul în care aplicaţia rulează local.

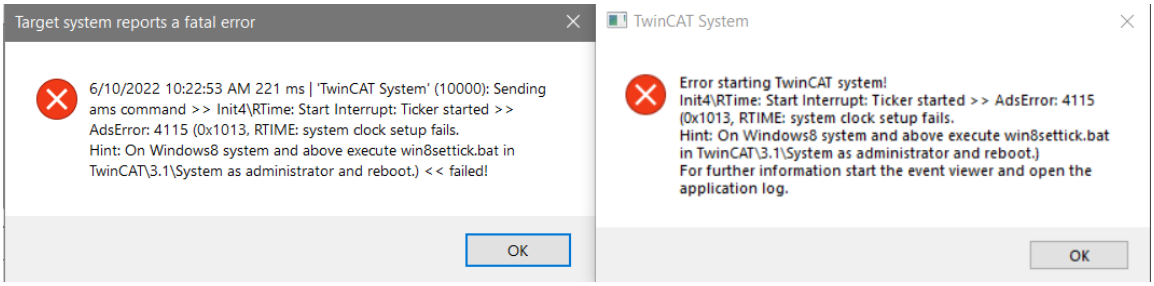

Figura 4.22: Eroare Target System

Eroarea din figura 4.23 apare în momentul în care noi am configurat un dispozitiv *EtherCAT master*, dar în reţeaua noastră nu exsită unul fizic. Rezolvarea acestei erori constă în dezactivarea dispozitivului dând *click dreapta->Disable* atunci când lucrăm local şi evident nu avem un *master* sau prin adăugarea lui fizic.

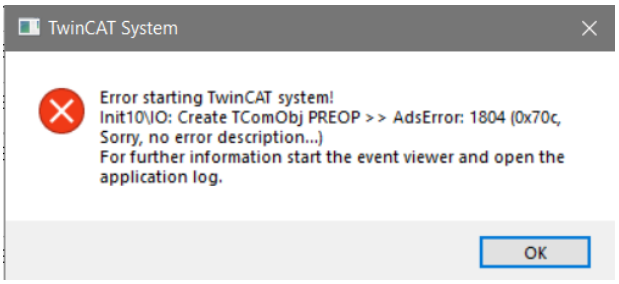

Figura 4.23: Eroare lipsă master

4.2 Crearea şi configurarea aplicaţiei în *Stäubli Robotics Suite*

Deoarece avem nevoie de o mişcare mai complexă a robotului, programarea acestuia nu poate fi făcută prin intermediul *pendant*-ului. Prin urmare, tehnica de programare offline discutată în capitolul anterior trebuie aplicată. Aceasta se face

folosindu-ne de un mediu de dezvoltare integrat(*IDE*) numit *Stäubli Robotics Suite* în care scrierea codului aferent se face în limbajul *VAL3*.

Crearea unui program nou se face accesând meniul *File->New->New Cell Wizard*. Acolo dăm un nume aplicației și selectăm directorul în care se salvează. Următorul pas este configurarea controlerului, iar pentru asta selectăm *Add a local controller*. În continuare se va deschide o interfață precum cea din figura 4.24 în care trebuie să configurăm brațul de robot cu care vom lucra. Aici alegem familia și modelul, modul în care se va face prinderea, în cazul nostru pe podea, dacă dorim să dispunem de un model 3D sau doar 2D, tipul de efector final, lungimea și diametrul *joint*-ului 3 și câteva detalii referitoare la mediul în care acest robot va lucra. După ce am selectat toate aceste aspecte și apăsăm *Next*, se va deschide o interfață în care trebuie să selectăm tipul de controler cu care acest braț va lucra și să-i dăm un nume. Ultima interfață deschisă în crearea programului ne arată tipul de braț robotic și tipul de controler selectate, împreună cu configurațiile fiecăruia.

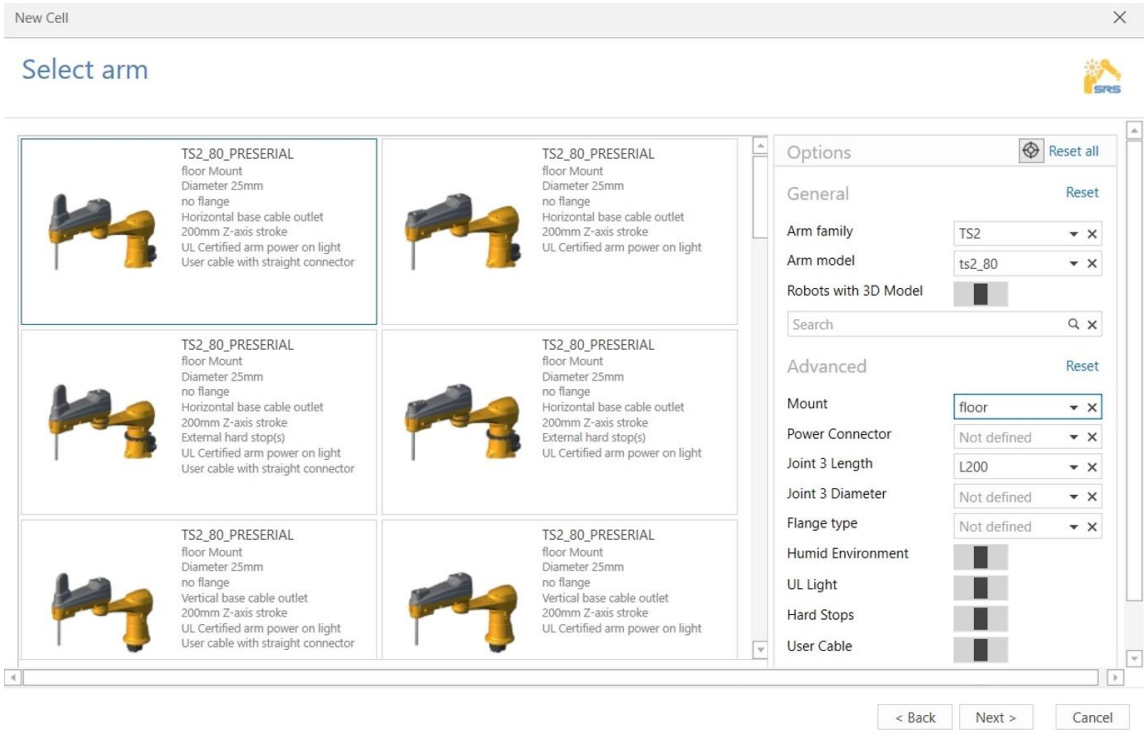

Figura 4.24: Configurarea brațului robotic TS2-80

Structura programului va arăta ca în figura 4.25. Pentru a crea o aplicație în limbajul *VAL3*, se va da *click* dreapta pe *Controller1->New Application* sau se va folosi *shortcut*-ul *CTRL+N+A.* Acolo selectăm numele fișierului și directorul în care dorim să se salveze. Programul va conține două funcții principale *start* și *stop*, un *tab* de referințe și un *tab* pentru crearea unei interfețe grafice.

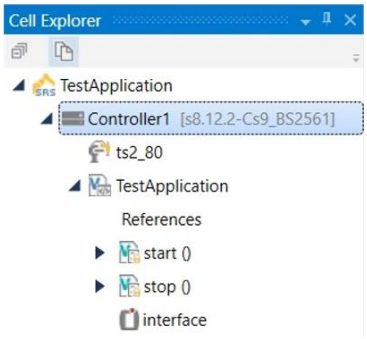

Figura 4.25: Structura programului în Stäubli Robotics Suite

Adăugarea unei funcţii noi se face dând *click* dreapta pe aplicaţia principală apoi pe *Add->New Data* sau prin intermediul *shortcut*-ului *CTRL+N+P*. Fiecare funcţie conţine o secţiune de parametrii şi una de variabile locale. Parametrii pot fi consideraţi ca fiind intrări şi ieşiri, iar variabilele locale sunt variabile utilizate strict în funcţia respectivă. Apelarea funcţiei se face utilizând instrucţiunea "*call*", urmată de numele acesteia. Funcţiile *start/stop* sunt două funcţii rulate de *task*-uri cu priorităţi mari. În start se configurează logica aplicaţiei, se descriu mişcări şi se apelează alte funcţii ce dorim să ruleze în momentul lansării în execuţie al programului. Aceasta este apelată automat în momentul în care o rulăm. Funcţia *stop* este apelată automat în momentul opririi aplicaţiei.

4.2.1 Comunicaţia cu *plc*-ul

Pentru a realiza comunicaţia dintre *plc* şi robot, mai întâi robotul trebuie configurat ca *slave* în reţeaua *EtherCAT*. Acest lucru se face accesând meniul *Physical IOs* din meniul principal şi selectarea *tab*-ului *Add IO board*. Acest lucru duce la configurarea portului *J207/J208* precum în figura 4.26

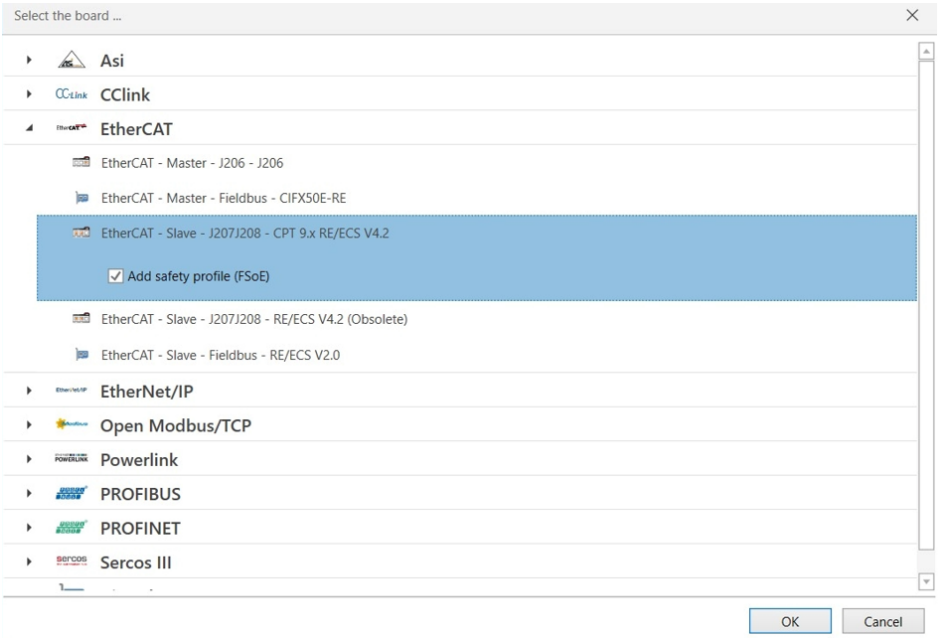

Figura 4.26: Configurarea portului J207/J208 din controlerul CS9

În interiorul aplicației, pentru a avea acces la semnalele digitale și analogice trimise din *Twincat 3*, dar și pentru a trimite la rândul nostru semnale către *plc*, trebuie să declarăm date de tip *dio* pentru semnale digitale, respectiv *aio* pentru semnale analogice. Aceste sunt legate mai departe de intrările/ieșirile fizice, prin intermediul fișierului *ESI*. Un exemplu de astfel de legătură este prezentat în figura 4.27, unde variabilele *bit_OUTPUT* sunt legate de ieșiri.

Figura 4.27: Legătura dintre variabile și intrări/ieșiri din programul Stäubli Robotics Suite

Citirea și scrierea variabilelor de acest fel se realizează prin folosirea funcțiilor *aioGet*, *aioSet* în ceea ce privește semnalele analogice. Semnalele digitale pot fi stocate în memorie prin atribuirea variabilei *dio* unei variabile *boolean* folosind operatorul *"="*. Scrierea acesteia se realizează la fel, prin atribuirea valorii 0/1 sau *false/true* variabilei *dio*.

4.2.2 Interfața grafică din cadrul *Stäubli Robotics Suite*

Pentru o ușoară manevrabilitate a programului, am ales să creez o interfață grafică ce o să fie afișată pe ecranul pendantului de instruire al robotului în momentul rulării aplicației. Interfața dispune de cinci butoane: *Start, STOP, Reset, On/Off*(pentru funcția de optimizare) și *Exit*. Acestea au fost adăugate folosind *toolbox*-ul conținut în programul *Stäubli Robotics Suite* ce se basează pe *Html, CSS* și *JavaScript*. Butonul de *Start* este responsabil pentru modul automat. Odată apăsat, stațiile descrise în subcapitolul anterior se inițializează, iar procesul pornește. Opusul acestuia este butonul de *STOP*, care odată activat oprește tot sistemul, nu doar mișcările robotului. Butonul de *Reset* modifică mișcarea robotului, acesta trasportând piesele într-o locație adiacentă și apoi deplasându-se într-un punct implicit, după care procesul se oprește. Pentru a observa mai bine în ce constă termenul de optimizare, am ales să adaug un buton prin care putem activa/dezactiva funcția, în vederea fizică a diferențelor și beneficiilor. Butonul de *Exit* oprește aplicația robotului.

Aceste butoane sunt legate de variabile din program trimise mai apoi prin *EtherCAT* către variabilele din *Twincat 3*. Un *task* pe care această interfață rulează este responsabil cu modificarea și transmiterea parametrilor într-un timp util. Am mai adăugat două imagini reprezentative și o etichetă pentru explicarea funcționalității unui buton. Toate aceste elemente sunt încadrate în interiorul unei rame, pentru a putea păstra dimensiunile și pozițiile corecte.

Interfața grafică este prezentată în figura 4.28.

Figura 4.28: Interfață grafică Stäubli Robotics Suite

4.2.3 Învățarea pozițiilor robotului prin intermediul *pendant-ului*

Pentru a reprezenta cât mai intuitiv modul prin care pozițiile robotului pot fi învățate prin intermediul *pendant*-ului, am ales să mă folosesc de emulatorul pus la dispoziție de *software*-ul folosit pentru programarea acestuia. O imagine reprezentativă o constituie figura 4.29, în care dispunem de șase *tab*-uri.

Tab-ul *VAL3* cuprinde o metodă de accees asupra memoriei controlerului în care sunt stocate aplicațiile scrise. Din acest *tab* putem rula diferite aplicații, dar le putem și modifica. În același timp, în momentul în care un program rulează, putem observa numele, starea și prioritatea fiecărui *task*. Putem să îl activăm/dezactivăm, să îl oprim și chiar să vedem ce linie de cod este în executare la un moment în timp.

Tab-ul *IO* conține interfața de acces asupra intrărilor și al ieșirilor disponibile. De aici putem configura și observa modurile de comunicare, putem citi valori ale sistemului precum temperaturi ale procesoarelor sau a diferitelor componente ale controlerului, starea axelor brațului, frâne active și bineînțeles intrările și ieșirile aplicației.

În *tab*-ul *Robot* dispunem de câteva unelte pentru calibrarea robotului sau înregistrarea unei mișcări în vederea analizării ei. De aici avem acces asupra informațiilor din sistem precum versiunea robotului, modul în care s-a făcut prinderea,

modul în care este adăugat în rețea(*EtherCAT slave*) sau chiar limitele de mișcare în care poate opera în condiții de siguranță.

În *Settings* putem modifica limba afișată, data și ora sistemului sau alte aspecte în ceea ce privește profilul sau rețeaua.

În *Logger* sunt afișate informații, erori și mesaje de avertizare cu privire la sistem. Acesta poate fi exportat pe un computer și folosit mai târziu pentru analiză.

Figura 4.29: Interfața pendant-ului de programare al robotului

Secțiunea *Jog* cuprinde informații referitoare la direcția de rotație a axelor. De aici putem adăuga/șterge puncte, *end-effectori* și putem configura spațiul de lucru. Primul pas în invățarea unui punct este crearea acestuia în interiorul aplicației. Acest lucru poate fi făcut prin intermediul mediului de dezvoltare integrat *VAL3* sau prin intermediul *pendant*-ului. Pentru a folosi *pendant*-ul, aplicația trebuie descărcată local, precum în figura 4.30.

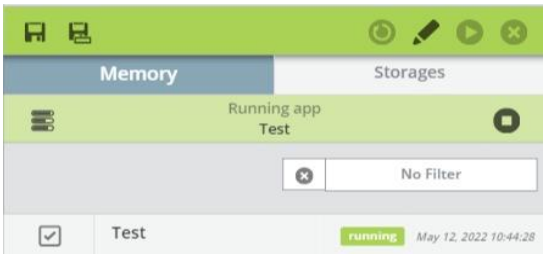

Figura 4.30: Descărcarea și rularea aplicației pe robotul fizic

Următorul pas este accesarea *tab*-ului *Jog* și selectarea aplicației căruia dorim să-i aducem modificări. După selectare, o să ne apară toate punctele declarate în program. Pentru a adăuga unul nou apăsăm pe butonul de plus disponibil, introducem un nume, o dimensiune, dacă să fie public sau privat și selectăm *Ok*. Fie că a fost adăugat prin

intermediul *pendant*-ului sau prin intermediul mediului de dezvoltare *VAL3*, coordonatele se modifică selectând punctul și apăsând butonul din figura 4.31.

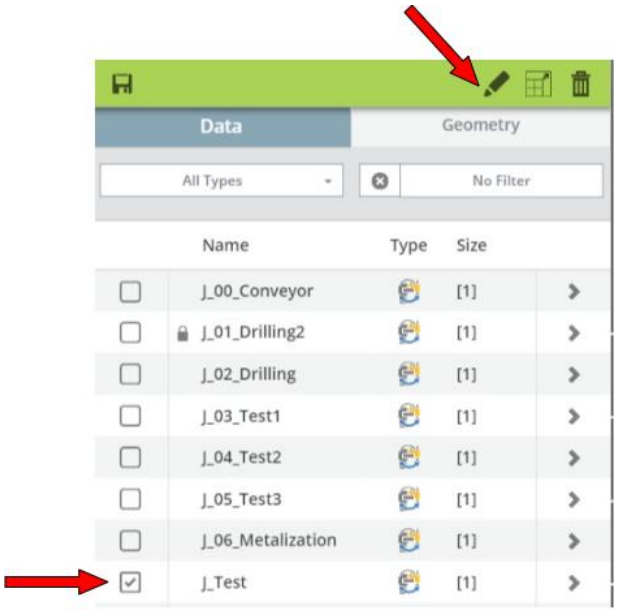

Figura 4.31: Modificarea coordonatelor unui punct prin intermediul pendant-ului

În continuare se va deschide o interfață precum cea din figura 4.32, în care coordonatele actuale ale poziției robotului pot fi memorate în varibila punct declarată. Apăsăm *Ok*, iar mai apoi salvăm.

Figura 4.32: Memorarea coordonatelor poziției robotului

Odată adăugate, punctele sunt gata pentru a fi folosite. Acestea se vor modifica, iar noile coordonate vor fi disponibile și în mediul de programare *VAL3*.

4.3 Problema de optimizare a secvenței de mișcări a robotului

Pentru a oferi o descriere cât mai amplă și mai precisă asupra unui astfel de proces destul de întâlnit în practică și a modului în care el poate fi îmbunătățit, noțiunea de optimizare a trebuit luată în calcul încă din momentul conceperii aplicației. Scopul principal al acesteia este realizarea unei piese sau componente într-un timp cât mai scurt posibil, fără adăugare de echipament *hardware* în plus. Deoarece stațiile sunt

simulate, acestea sunt reprezentate de puncte în spaţiu. În figura 4.33 poate fi observat cadrul de lucru aferent aplicaţiei văzut de sus. Aceasta conţine limitele de deplasare ale robotului, coordonatele fiecărei staţii şi traseele de deplasare de la o staţie la alta.

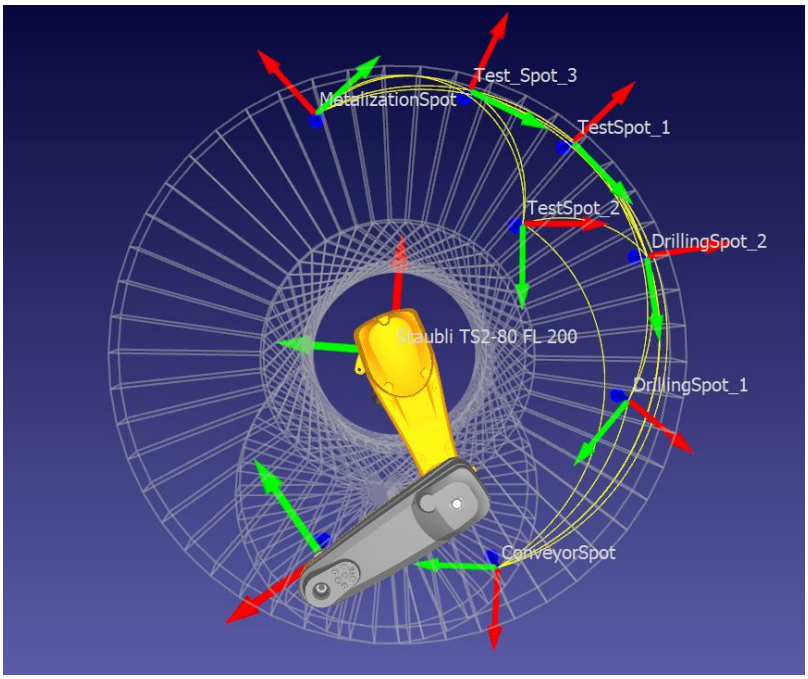

Figura 4.33: Reprezentarea 2D văzută de sus a spaţiului de lucru

Pentru a crea o mică corelaţie între o staţie reală şi staţia simulată, am ales ca punctele de ridicare respectiv de plasare a componentelor să difere. Astfel le-am declarat la înălţimi diferite, fapt observabil în figura 4.34. Atunci când robotul ajunge în *SignalPoint*, staţia este anunţată că robotul a ajuns în drept cu ea. În cazul acţiunii de ridicare, dacă piesa este gata, aceasta poate fi luată din *PickPoint*, iar în cazul plasării, aceasta poate fi lăsată în *PlacePoint*.

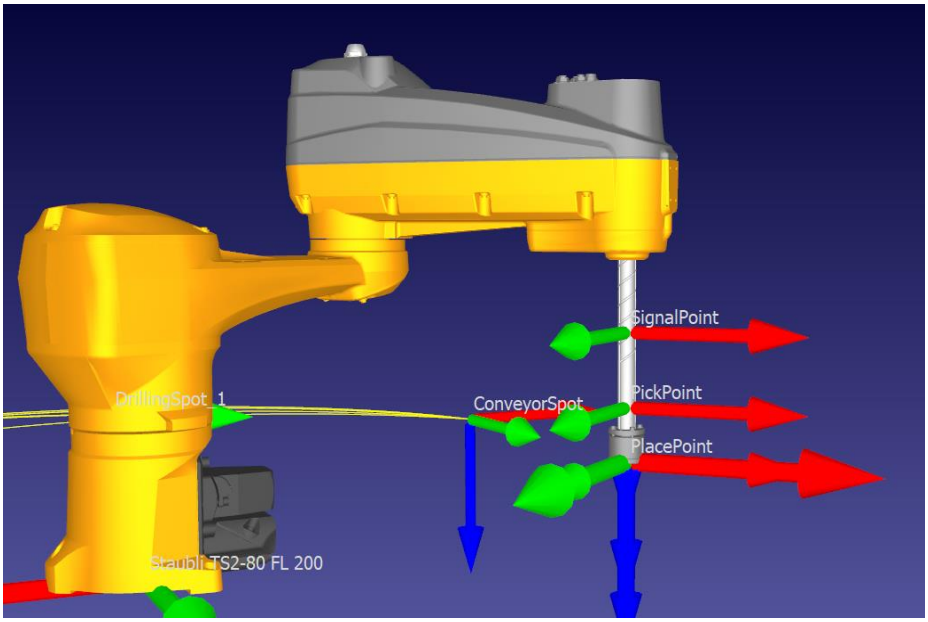

Figura 4.34: Reprezentarea 3D a locaţiilor de ridicare/plasare a pieselor

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*. Următorul limbaj din listă, în ceea ce privește viteza, este *Structured Text(ST).* Datorită capabilităților de portare, manipulare de date complexe și implementare a programării orientate pe obiecte, am ales să folosesc acest limbaj în dezvoltarea aplicației. Pentru a crea o logică bazată pe mașini de stare (engl. *State Machines*), pentru simplitatea înțelegerii codului, dar și a obiectelor create, am ales să folosesc *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 am ales ca logica aplicației să ruleze 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.

Ținând cont de aceste detalii, implementarea aplicației poate fi realizată. Pentru a prezenta un mod de concepere al acestor aplicații, o să mă folosesc de 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 stație 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:

$$PCT = SCT + DT + ADT + RADT \tag{4.1}$$

O primă îmbunătățire ce poate fi efectuată asupra acestei abordări este eliminarea punctului adiacent. Prin urmare, toate deplasările efectuate pentru a elibera spațiul de lucru sunt eliminate. Totuși, avem nevoie de o strategie de evitare a zonei de lucru după ce piesa a fost plasată, iar prin urmare, punctul adiacent eliminat anterior se transformă de fapt în câte un punct adiacent asociat fiecărei stații. Exemplificând acest concept pe figura 4.34, punctul adiacent aferent fiecărei stații îl reprezintă orice pozitie a axei 3, deasupra punctului *SignalPoint*. Pe scurt, atunci când axa 3 se află deasupra punctului *SignalPoint*, indiferent de poziția celorlalte axe, procesul poate începe și se consideră că se efectuează în condiții de siguranță.

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.

Deoarece aplicația constă în comunicarea continuă dintre *plc* și robot, ea trebuie să se realizeze cât mai simplu și mai eficient. Pentru asta, mișcările robotului sunt comandate prin intermediul unor biți de control transmiși doar la nevoie de mișcare, nu continuu. Un exemplu de comunicare este reprezentat în figura 4.35.

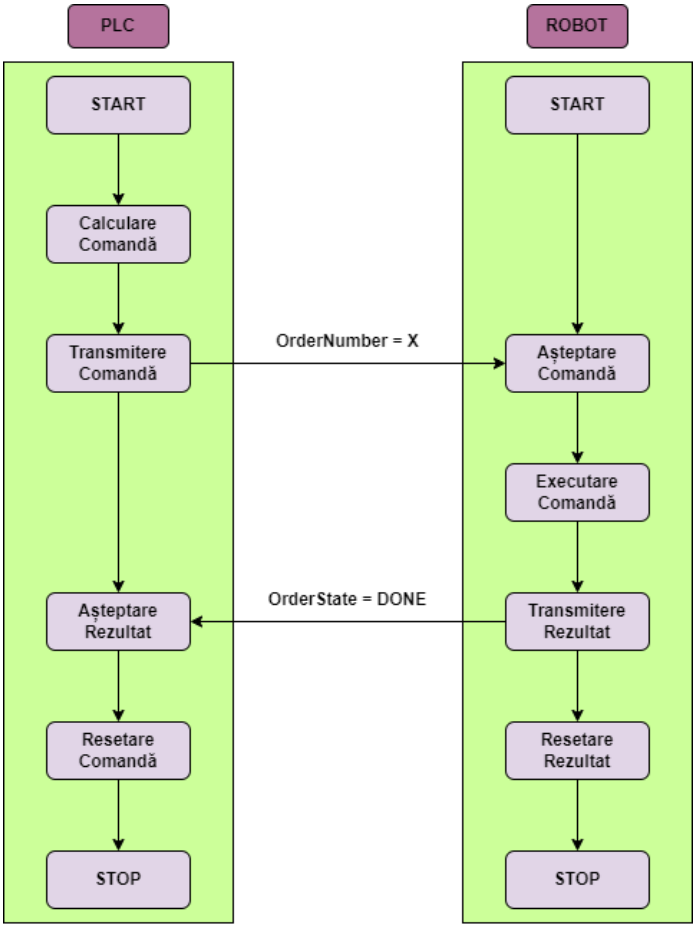

Figura 4.35: Principiul de comunicare dintre plc și robotul Stäubli

4.3.1 Optimizarea adaptivă a secvenței de mișcări a robotului

Ultima, dar şi cea mai importantă parte în ceea ce priveşte optimizarea unui astfel de proces o reprezintă optimizarea adaptivă. În acest caz, termenul de optimizare se referă la integrarea pe cât de mult posibil a timpilor necesari deplasării braţului robotic, în vederea mutării unei piese de pe o staţie pe alta, sincron cu timpul necesar prelucrării acesteia. Termenul de adaptiv constă în independabilitatea eficientizării procesului în contrast cu variaţia parametrilor externi ai staţiilor sau ai robotului. Un exemplu îl reprezintă defectarea unei staţii sau variaţia timpului procesului care are loc în cadrul ei.

După cum am explicat şi în capitolul de introducere, *plc*-ul interacţionează cu robotul prin intermediul unui mod de comunicare bazat pe ordine. Problema de optimizare constă în estimarea unui ordin sau a unei poziţii de ridicare viitoare a robotului înainte ca acestea să fie transmise. Pentru a putea fi rezolvată, problema trebuie mai întâi formulată, iar pentru asta am ales să mă folosesc de următoarele notaţii:

- *Există R un singur robot ce poate muta piese dintr-o staţie în alta;*
- *Există N = {n_0, n_1, n_3, n_4} un set de tipuri de staţii posibile;*
- *Există M ≥ N, M = {m_0, m_1, m_2, m_3, m_4, m_5, m_6} un set de instanţe de staţii posibile;*
- *Există un timp de tranziţie x de la fiecare componentă a setului M la o altă componentă a setului M, inclusiv de la o staţie la ea însăşi, timpul fiind 0;*

$$Tm_{ij} = x \qquad i \leftarrow 0:M, j \leftarrow 0:M \tag{4.2}$$

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

$$tm_i = x \qquad i \leftarrow 0:M \tag{4.3}$$

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

$$m_0 \rightarrow m_1,\ m_2 \rightarrow m_3,\ m_4,\ m_5 \rightarrow m_6 \tag{4.4}$$

- *Staţia m_0 reprezintă o staţie generator, iar staţia m_6 o staţie terminală;*
- *Chiar dacă staţiile sunt simulate, iar timpul lor l-am setat atunci când aplicaţia 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} şi tm_i sau timpul de deplasare de la o staţie la alta şi timpul de procesare al unei piese aferent fiecărei staţii. Pentru aflarea timpului de deplasare, a trebuit să mă gândesc la un mod de a automatiza acest proces, fără a calcula fiecare distanţă manual(49 în total). Pentru asta, în interiorul aplicaţiei, mi-am creat o funcţie de măsurare ce este activată în momentul startării deplasării robotului, adică în momentul trimiterii unui ordin. Algoritmul acestei funcţii este prezentat în cele ce urmează.

Algoritmul 4.2 Calcularea timpului de deplasare între două staţii

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

Practic acelaşi semnal trimis către robot în vederea pornirii mişcării este trimis şi către funcţia de calculare al distanţei. Primind acest semnal *boolean*, *trigger*-ul se activează, iar timpul curent este memorat în variabila *startTime*. Atunci când robotul ajunge în punctul dorit, trimite un semnal către funcţie tot prin intermediul aceleiaşi variabile *boolean*. *Trigger*-ul se activează din nou şi se calculează diferenţa de timp dintre timpul curent şi *startTime*. Diferenţa o reprezintă timpul de mişcare.

Acelaşi principiu este aplicat şi pentru aflarea timpilor de execuţie al fiecărei staţii. În momentul în care robotul plasează o piesă într-un modul şi se ridică într-un punct sigur discutat în secţiunile anterioare, un *trigger* pe front crescător este activat şi timpul curent este salvat. Atunci când piesa este gata, *trigger*-ul pe front descrescător este activat, iar diferenţa de timp este calculată şi returnată, ea reprezentând timpul de execuţie al staţiei.

Având la dispoziţie toate aceste date, prima strategie de optimizare poate fi aplicată. Ea constă în monitorizarea continuă a timpilor tututor staţiilor 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 staţiile din jur şi se deplasează în imediata apropiere, deasupra punctului de *SignalPoint*, a staţiei unde timpul rămas este cel mai scurt. Când ordinul de ridicare este receptat de robot, mişcarea este una minimă. Datorită timpilor staţiilor care sunt variabili, această estimare nu este întotdeauna una corectă. În urma testelor efectuate, am ales să creez un nou *task* cu o prioritate mai mică decât *task*-ul principal pe care această funcţie de optimizare să ruleze. Pe durata mişcării executate de către robot în urma semnalării unei noi estimări din partea funcţiei, dacă un ordin nou, diferit de cel în execuţie, este semnalat, robotul se va deplasa urmând comanda cu prioritatea cea mai mare.

Această abordare poate fi însă îmbunătăţită prin adăugarea în ecuaţie a timpilor de deplasare. Să presupunem următorul exemplu prezentat în figura 4.36. Avem două staţii, *Module 1* şi *Module 2* cu timpul estimat(în funcţie de media timpilor de ciclu) de terminare aferent, *7* pentru *Module 1* şi *9* pentru *Module 2*(unitatea de măsură nu este relevantă atâta timp cât scara este aceeaşi pentru toate mărimile) şi un robot *Robot*. Pe desen sunt trecuţi timpii necesari deplasării pentru fiecare staţie, *15* pentru *Module 1* şi *5* pentru *Module 2*. Aplicând algoritmul conceput anterior, bazat pe timpul de terminare al fiecărei staţii, robotul va fi comandat să meargă la modulul care va fi gata primul, adică *Module 1*. Făcând un calcul simplu, putem observa că în momentul estimării

următoarei mișcări, plecând din timpul curent, adică *0*, vom ajunge în dreptul modulului 1 la timpul *15*, iar piesa va fi gata. Calculând pentru stația 2, plecând tot din timpul *0*, constatăm că ajungem în dreptul ei la timpul *5* și mai asteptăm *4* timpi ca piesa să fie gata, în total *9*. Se dovedește astfel că dacă luăm în calcul și timpii necesari deplasării, estimarea bazată doar pe timpul de terminare al stațiilor nu va fi mereu cea mai eficientă.

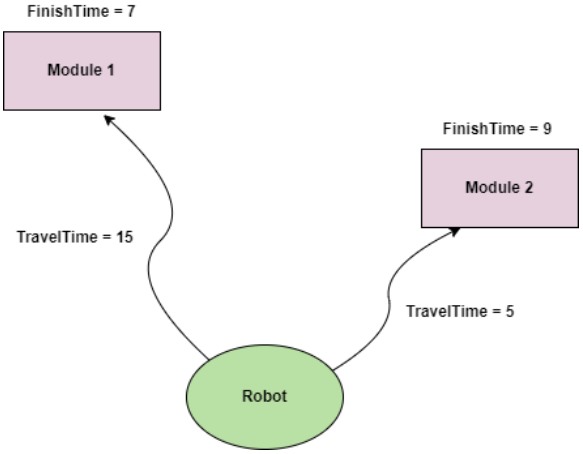

Figura 4.36: Exemplu de îmbunătățire al aplicației

Această strategie poate fi aplicată prin intermediul unui graf orientat în care nodurile sunt reprezentate de stații, iar muchiile reprezintă costul aferent, în timp, asociat fiecărei deplasări. Graful acestei aplicații poate fi observat în figura 4.37.

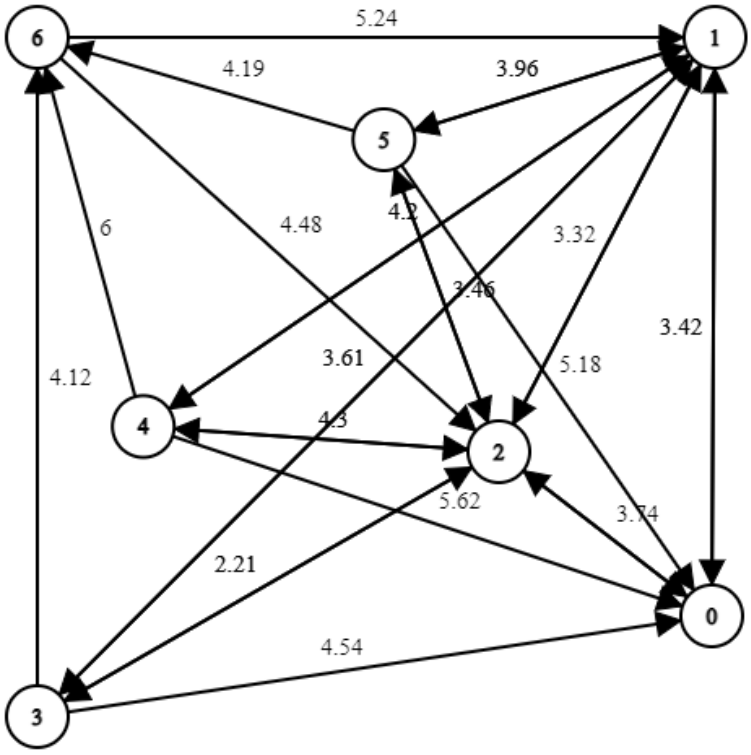

Figura 4.37: Graful orientat al aplicației

Matricea de cost aferentă grafului este:

$$\begin{array}{c} \\ Cnv \\ Drl1 \\ Drl2 \\ Tst1 \\ Tst2 \\ Tst3 \\ Mtz \end{array} \begin{array}{c} \begin{array}{ccccccc} Cnv & Drl1 & Drl2 & Tst1 & Tst2 & Tst3 & Mtz \end{array} \\ \begin{bmatrix} 0 & 3.42 & 3.74 & X & X & X & X \\ 3.42 & 0 & 3.32 & 3.61 & 4.2 & 3.96 & X \\ 3.88 & 3.5 & 0 & 2.21 & 4.3 & 3.46 & X \\ 4.54 & 3.61 & 2.21 & 0 & X & X & 4.12 \\ 5.62 & 4.2 & 4.3 & X & 0 & X & 6 \\ 5.18 & 3.96 & 3.46 & X & X & 0 & 4.19 \\ X & 5.24 & 4.48 & X & X & X & 0 \end{bmatrix} \end{array} \quad (4.5)$$

Conţinutul matricii îl reprezintă timpul de transport al unei piese dintr-o staţie în alta calculat în secunde. Bineînţeles, timpul corespunzător de transport de la o staţie la ea însăşi este *0*. Datorită restrângerilor aplicate prin care interzicem unei componente să treacă haotic prin toate staţiile, acolo unde transportul nu este posibil, este trecut *X*. Un alt aspect interesant descoperit în momentul construirii acestei matrici, este diferenţa timpilor măsuraţi pe aceeaşi traictorie, dar în sens invers. Spre exemplu timpul de la *Conveyor* la *Drilling_2* este *3.74s*, iar de la *Drilling_2* la *Conveyor 3.88s*. Acest fapt se datorează ori orientării braţului robotic ori timpului de scanare al codului.

Având aceste detalii la dispoziţie, am putut implementa o logică pentru ceea ce am discutat mai sus. Ieşirea funcţiei de optimizare este *nextPosition*, iar intrările sunt următoarele:

- *job* – o variabilă de tip *Order* prin care verific dacă robotul este în starea de repaus;
- *lastPosition* – variabilă de tip *Position* prin care salvez ultima poziţie în care se află robotul;
- *startTime* - variabilă de tip *Time* prin care memorez timpul la care fiecare staţie şi-a început procesul;
- *cycleTime* - variabiliă în care salvez timpul mediu de execuţie al fiecărei staţii;
- *availableModules* – variabilă prin care verific disponibilitatea staţiilor;

Primul pas pe care această funcţie îl execută este culegerea datelor necesare calculelor. Datele sunt mai apoi salvate local şi modificate. În cazul în care nu există niciun ordin disponibil, ea este apelată. Se efectuează calcule precum cele exemplificate în cadrul prezentării figurii 4.36 şi se returnează o estimare a poziţiei viitoare. Această estimare nu este întotdeauna cea corectă, iar prin urmare ea trebuie examinată. Pentru asta verificăm starea următorului modul în secvenţă după staţia rezultată în urma aplicării funcţiei. Dacă acesta este liber, ieşirea funcţiei o reprezintă estimarea făcută. Dacă modulul următor nu este liber, se recalculează o nouă estimare ignorând-o pe prima şi se repetă procesul. Dacă nicio estimare nu este bună din cele cinci posibile, înseamnă că toate staţiile procesului nostru sunt pline, deci prin urmare, robotul trebuie dus pe poziţia modulelor de test care preced modulul de ieşire din sistem. Pseudocodul aferent acestui algoritm este prezentat mai jos, însă pentru o explicare cât mai simplă am ales să tratez staţiile de acelaşi tip ca fiind o singură instanţă.

Algoritmul 4.3 Optimizare

```
1: Intrări: job, lastPosition, startTime=[TC, TD1, TD2, TT1, TT2, TT3], cycleTime=[TC, TD1, TD2,
   TT1, TT2, TT3], AvailableModules
2: Ieşiri: nextPosition
3: if job = noJob then
4:     for i = 0 to modulesNumber do
5:         shortestTime[i] ← startTime[i] + movementTime[lastPosition][i] + cycleTime[i]
6:     end for
7:     minValue ← getMinOf(shortestTime)
8:     minIndex ← getIndexOf(minValue)
9:     finalPosition ← minIndex
10:    switch finalPosition do
11:        case conveyorPosition
12:            if drilling = empty then
13:                nextPosition = conveyorPosition
14:            else
15:                shortestTime[finalPosition] ← maxValue + 1
16:            end if
17:        case drillingPosition
18:            if test = empty then
19:                nextPosition = drillingPosition
20:            else
21:                shortestTime[finalPosition] ← maxValue + 1
22:            end if
23:        case testPosition
24:            if metalization = empty then
25:                nextPosition = testPosition
26:            else
27:                shortestTime[finalPosition] ← maxValue + 1
28:            end if
29:    if noPositionAvailable then
30:        nextPosition ← testPosition
31:    end if
32: else
33:    nextPosition ← NoPositionFound
34: end if
35: return nextPosition
```

4.4 Testarea algoritmului implementat

Pentru partea de testare am ales să măsor timpul necesar prelucrării a cinci piese cu funcţia de optimizare atât pornită, cât şi oprită pentru a putea observa rezultatele. Pentru asta m-am folosit de aceeaşi funcţie de măsurare bazată pe *trigger*-e. Am ales ca testul să se realizeze pe un caz ideal, în care timpii de procesare aleatoriu ai staţiilor sunt fixi, pentru a avea o mai bună reprezentare a efectelor aplicării unei astfel de optimizări. Rezultatele pot fi observate în figura 4.38. Pentru 5 piese, fără funcţia de optimizare pornită, timpul total de procesare a fost de 399.26 secunde. Pentru acelaşi 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.

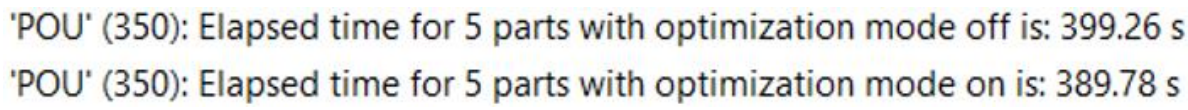

Figura 4.38: Rezultate în urma aplicării optimizării

5 Concluzii și Direcții de dezvoltare

5.1 Concluzii

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

Cel mai anevoios moment în crearea acestei lucrări a fost gândirea unui proces pentru care o metodă de optimizare poate scoate în evidenţă într-un mod cât mai precis şi intuitiv importanţa bunelor practici în ceea ce priveşte programarea, compunerea şi configurarea unui proces industrial, în care robotul este principalul element de execuţie. Alegerea limbajului de programare s-a făcut rapid datorită singurei specificaţii impuse, viteza. Chiar dacă staţiile sunt simulate, ele reprezintă cu un procent ridicat de precizie principalele caracteristici ale unei staţii reale: tehnici de comunicare, metode de programare modulară, flux de evenimente etc. Integrarea robotului într-un astfel de proces implică lucruri precum crearea comunicaţiilor aferente, definirea poziţiilor staţiilor, evitarea obstacolelor şi simularea acestuia într-un mediu 3D pentru dovedirea siguranţei. Fluxul aplicaţiei a fost gândit şi creat pentru a simula în exactitate caracteristicile unui proces real.

În urma testelor efectuate se observă că metoda de optimizare dă roade în ceea ce priveşte acest proces, dar nu numai. Datorită capabilităţilor de adaptare la diferiţi parametrii în cadrul acestei aplicaţii, precum timpul de prelucrare al pieselor, dar şi la o structură diferită de proces prin scoaterea sau adăugarea unor staţii, aplicaţia poate servi ca punct de plecare pentru o implementare mai amplă în vederea optimizării altor procese industriale, unde atenţia este focusată asupra unui braţ robotic.

Aplicaţia poate fi rulată şi analizată având un minim de cunoştinţe datorită interfeţelor vizuale create şi puse la dispoziţia utilizatorilor atât pentru aplicaţia de *plc*, cât şi pentru programul robotului. Prin utilizarea acestor interfeţe, utilizatorii se pot focusa mai mult pe partea de secvenţă de mişcări şi mai puţin pe partea de implementare a acestora.

Consider că am reuşit să acopăr toate obiectivele propuse în capitolul de introducere şi să scot în evidenţă aspecte importante, de multe ori ignorate, cu privire la modul de integrare al roboţilor în mediul industrial.

5.2 Direcţii de dezvoltare

Deoarece am dorit încă din primul moment al creării acestei lucrări ca aplicaţia să poată servi ca model în vederea altor implementări, o primă direcţie de dezvoltare este crearea unui manual în care este explicat pas cu pas fiecare aspect de la crearea programului propriu-zis la lansarea acestuia în execuţie. Manualul trebuie să conţină date cu privire la: modul de gândire al comunicaţiilor, al staţiilor şi al robotului. Fiind un domeniu atât de vast şi de răspândit, echipamentul folosit este utilizat, probabil, într-un procent mic. Prin urmare, pentru a facilita implementarea aplicaţiilor şi pe alte echipamente, o altă direcţie de dezvoltare posibilă este crearea pseudocodului aferent atât programului de *Twincat 3*, cât şi programului din *Stäubli Robotics Suite*. Acest lucru duce la capabilitatea de folosinţă pe scară largă, indiferent de modelul de *plc* sau de robotul folosit.

Întreg fluxul aplicaţiei se bazează pe faptul că o singură componentă poate fi transportată, la un moment de timp, de la o staţie la alta. O îmbunătăţire vizibilă este adăugarea unui element de prindere ce poate lua două sau chiar trei piese în acelaşi timp. Prin acest lucru se creează cu un efort minim încă o poziţie auxiliară în care o piesă poate staţiona în cazul în care modulul următor nu este încă liber. Bineînţeles, acest aspect duce la eliberarea mai rapidă a staţiilor curente, indiferent de disponibilitatea staţiilor următoare şi la posibilitatea completării lor cu alte componente în aşteptare.

O altă imbunătăţire ce poate fi adusă acestei aplicaţii este introducerea unui alt robot în sistem care să folosească acelaşi spaţiu de lucru. Pentru asta, în viitorul apropiat, doresc să concep o strategie de colaborare între ei care să crească nivelul producţiei. Un prim avantaj al acestei abordări este faptul că distanţele parcurse de fiecare robot se înjumătăţesc, iar mai mult, deplasările se desfăşoară concomitent. În vederea atingerii acestui obiectiv, doresc să proiectez o metodă de evitare a coliziunilor dintre aceştia, care să menţină sau chiar să îmbunătăţească timpul de ciclu aferent procesului, bazată pe tehnici şi tehnologii folosite în domeniul inteligenţei artificiale cu date rezultate în urma implementării acesteia într-un mediu 3D simulat.

6 Bibliografie

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

[2] D. H. Hanssen, Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS, John Wiley & Sons, 2015.

[3] M. G. Hudedmani, S. K. Kabberalli, R. M. Umayal şi R. Hittalamani, „Programmable logic controller (PLC) in automation," *Advanced Journal of Graduate Research,* vol. 2, nr. 1, pp. 37-45, 2017.

[4] I. Ahmed, S. Obermeier, S. Sudhakaran şi V. Roussev, „Programmable logic controller forensics," *IEEE Security & Privacy,* vol. 15, nr. 6, pp. 18-24, 2017.

[5] D. U. Case, „Analysis of the cyber attack on the Ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC),* vol. 388, pp. 1-29, 2016.

[6] E. N. Yilmaz şi S. Gönen, „Attack detection/prevention system against cyber attack in industrial control systems," *Computers & Security,* vol. 77, pp. 94-105, 2018.

[7] G. F. Rossano, C. Martinez, M. Hedelind , S. Murphy şi 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.

[8] S. Y. Nof, Springer handbook of automation, Springer, 2009.

[9] D. Dzung , M. Naedele, T. P. Von Hoff şi M. Crevatin, „Security for industrial communication systems," *Proceedings of the IEEE,* vol. 93, nr. 6, pp. 1152-1177, 2005.

[10] X. Li, D. Li, J. Wan , A. V. Vasilakos, C.-F. Lai şi S. Wang, „A review of industrial wireless networks in the context of industry 4.0," *Wireless networks,* vol. 23, nr. 1, pp. 23-41, 2017.

[11] V. C. Gungor şi G. P. Hancke, „Industrial wireless sensor networks: Challenges, design principles, and technical approaches," *IEEE Transactions on industrial electronics,* vol. 56, nr. 10, pp. 4258-4265, 2009.

[12] Y.-J. Park, G. Ahn şi S. Hur, „Optimization of pick-and-place in die attach process using a genetic algorithm," *Applied Soft Computing,* vol. 68, pp. 856-865, 2018.

[13] Beckhoff. [Online]. Available: https://www.beckhoff.com/en-us/. [Accessed 17 05 2022].

[14] Stäubli, "Stäubli," [Online]. Available: https://www.staubli.com/ch/en/corp.html. [Accessed 22 05 2022].

[15] EtherCAT, "EtherCAT Technology Group," [Online]. Available: https://www.ethercat.org/default.htm. [Accessed 19 05 2022].