Korisničko sučelje za upravljanje robotom pomoću programabilnog logičkog sklopa

Šebalj, Karlo

Undergraduate thesis / Završni rad

2018

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:855285

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Karlo Šebalj

Zagreb, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Prof. dr. sc. Bojan Jerbić, dipl. ing.

Student:

Karlo Šebalj

Zagreb, 2018.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru prof. dr. sc. Bojanu Jerbiću što mi je dao priliku raditi na ovom radu te na savjetima i pomoći koju mi je pružio za vrijeme izrade ovog rada. Izuzetno sam zahvalan što mi je omogućio da radim u Laboratoriju za projektiranje izradbenih i montažnih sustava i svoj opremni koja je bila potrebna za izradu istog. Također, zahvaljujem asistentu mag.ing. Josipu Vidakoviću na pomoći, strpljenju i motivaciji koja je mi je uvelike olakšala cijeli posao. Htio bih zahvaliti i laborantu Draženu Buzjaku na iznimnoj pomoći pri svim komplikacijama na koje sam naišao te Mati Kocelju i Luki Rabuzinu na ugodnoj atmosferi u laboratoriju.

Zahvaljujem svojim roditeljima, bratu i djevojci što su mi bili najveća podrška tijekom cijelog studija te svim svojim prijateljima i kolegama koji su bili uz mene i na savjetima koje su mi dali.

Na kraju htio bih zahvaliti pokojnom djedu koji mi je bio veliki uzor za upis na ovaj fakultet.

Karlo Šebalj

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Karlo Šebalj** Mat. br.: 0082049890

Naslov rada na hrvatskom jeziku: **KORISNIČKO SUČELJE ZA UPRAVLJANJE ROBOTOM POMOĆU PROGRAMABILNOG LOGIČKOG SKLOPA**

Naslov rada na engleskom jeziku: **USER INTERFACE FOR ROBOT CONTROL USING PROGRAMMABLE LOGIC CONTROLLER**

Opis zadatka:

U radu je potrebno razviti korisničko sučelje na PC računalu koje je povezano preko programabilnog logičkog sklopa (PLC-a) s robotom. Razvijeno sučelje mora omogućiti udaljeno upravljanje i nadzor programabilnog logičkog sklopa putem kojeg se upravlja robotom. Naredbe robotu se zadaju kao upravljački procesi programabilnog logičkog sklopa čiji se odabir i parametri definiraju korisničkim sučeljem na računalu. Potrebno je povezati sve komponente sustava te izraditi upravljačke algoritme. Za razvijeno rješenje izvršiti laboratorijsku verifikaciju u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Zadatak zadan:

30. studenog 2017.

Rok predaje rada:
1. rok: 23. veljače 2018.
2. rok (izvanredni): 28. lipnja 2018.
3. rok: 21. rujna 2018.

Predviđeni datumi obrane:
1. rok: 26.2. - 2.3. 2018.
2. rok (izvanredni): 2.7. 2018.
3. rok: 24.9. - 28.9. 2018.

Zadatak zadao:

prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva:

Izv. prof. dr. sc. Branko Bauer

SADRŽAJ

SADRŽAJ I
POPIS SLIKA II
POPIS TABLICA III
POPIS OZNAKA IV
SAŽETAK V
SUMMARY VI
1. UVOD 1
2. KORISNIČKO SUČELJE 2
2.1. Povijest korisničkog sučelja 3
2.2. Vrste korisničkih sučelja 6
2.2.1. Komandno korisničko sučelje 6
2.2.2. Tekstualno korisničko sučelje 7
2.2.3. Grafičko korisničko sučelje 7
3. PROGRAMABILNI LOGIČKI SKLOP – PLC 9
3.1. Siemens PLC S7-1200 11
3.2. STEP 7 softver za programiranje 12
4. INDUSTRIJSKI ROBOT FANUC M-10*i*A 15
4.1. Karakteristike robota 16
5. KOMUNIKACIJSKE MREŽE I PROTOKOLI 19
5.1. Industrijski Ethernet 20
5.2. TCP/IP Protokol 21
5.3. PROFINET 22
6. PROJEKTIRANJE I IZRADA SUČELJA 25
6.1. C# Programski jezik 27
6.2. MVVM 28
6.2.1. Model 28
6.2.2. Prikaz (eng. View) 30
6.2.3. Model Prikaza (eng. ViewModel) 31
7. POVEZIVANJE SUČELJA S PLC-OM 33
7.1. Podaci u Data Blokovima 34
7.2. Pisanje programa u Ladder jeziku za kontrolu robota 34
8. POVEZIVANJE PLC-A S ROBOTOM 38
8.1. Programiranje robota 39
9. TESTIRANJE SUČELJA 41
10. ZAKLJUČAK 43
11. PRILOZI 45

POPIS SLIKA

Slika 1. Korisničko sučelje koje je dio Apple-ovog operativnog sustava 2
Slika 2. Prototip miša Douglasa Engelberta 3
Slika 3. Radna stanica XEROX Star-a 4
Slika 4. Korisničko sučelje Lisa kompjutera 4
Slika 5. Izgled grafičkog korisničkog sučelja Windows 95 5
Slika 6. Komandno korisničko sučelje 6
Slika 7. Tekstualno korisničko sučelje 7
Slika 8. Korisničko sučelje koje obuhvaća HMI i GUI 8
Slika 9. Allen-Bradley PLC američke tvrtke Rockwell Automation 10
Slika 10. PLC japanske tvrtke Mitsubishi 10
Slika 11. Siemens S7-1200 s utorima za spajanje 11
Slika 12. Siemens PLC S7-1200 spojen s modulom CSM 1277 12
Slika 13. Primjer bloka u LAD jeziku 14
Slika 14. Primjer bloka u FBD jeziku 14
Slika 15. Primjer koda u SCL jeziku 14
Slika 16. Industrijski robot FANUC M-10*i*A 16
Slika 17. B-kabinet unutar kojega je kontroler R30*i*B 18
Slika 18. Shema komunikacijskih mreža i protokola 19
Slika 19. Prikaz spajanja CNC operacijske jedinice preko raznih protokola 23
Slika 20. Početni izgled sučelja 25
Slika 21. Početni izgled sučelja 26
Slika 22. Tipke za početak/prekid rada sustava 26
Slika 23. Izgled tipki za pomicanje robota u sučelju 27
Slika 24. MVVM koncept 28
Slika 25. Dio koda za čitanje vrijednosti iz PLC-a u sloju model 29
Slika 26. Izgled prikaz sloja 30
Slika 27. Kod u modelu prikaza koji uzima vrijednosti zglobova iz sloja model 31
Slika 28. Primjer koda za dobivanje vrijednosti ako je tipka aktivirana u prikazu 32
Slika 29. Postavke za pristup podacima u PLC-u 33
Slika 30. Izgled Podatkovnog Bloka 2 34
Slika 31. Mreža 1 za pokretanje, prekid i održavanje sustava 35
Slika 32. Mreža 2 za poziv potprograma 36
Slika 33. Mreža 1 unutar potprograma za pretvorbu dobivenih vrijednosti iz robota 36
Slika 34. Dio Mreže 3 s blokovima za tipke „PLUS“ i „MINUS“ zgloba A1 37
Slika 35. Spajanje PLC-a s robotom pomoću GSD datoteke 38
Slika 36. Glavni program u upravljačkoj konzoli 39
Slika 37. Potprogram KARLO_SUCELJE 40
Slika 38. Drugi potprogram KARLO_UPISIVANJE 40
Slika 39. Testiranje tipki Connect, Disconnect, Start i Stop 41
Slika 40. Prikaz normalnog rada sučelja 42

POPIS TABLICA

Tablica 1. Tehničke specifikacije Siemensa S7-1200 klase CPU 1214 DC/DC/DC 11
Tablica 2. Karakteristike robota 17
Tablica 3. Makismalni kut zakreta i kutna brzina pojedinog zgloba 17
Tablica 4. OSI modul naspram TCP/IP modula 21

POPIS OZNAKA

Oznaka	Jedinica	Opis
m	[kg]	Masa
	[mm]	Ponovljivost
L	[mm]	Duljina
φ	[º]	Kut zakreta
ω	[º/s]	Kutna brzina

SAŽETAK

Zadatak završnog rada je razviti korisničko sučelje na računalu za upravljanje industrijskim robotom putem programabilnog logičkog sklopa (PLC, eng. Programmable Logic Controller). Upravljanje robotom preko vanjskih sučelja, umjesto preko robotske konzole omogućava korisniku izravan pristup robotskim funkcijama. Razvijeno sučelje treba omogućiti udaljeno upravljanje te nadzor programabilnog logičkog sklopa putem kojeg se upravlja robotom. U zadatku je trebalo izraditi program koji omogućava zadavanje upravljačkih naredbi na programabilnom logičkom sklopu, koje se potom prenose na upravljačko računalo robota. Sve komponente potrebno je povezati u jedinstven sustav korištenjem odgovarajućih komunikacijskih protokola te oblikovati upravljačke algoritme.

Ključne riječi: PLC, robot, komunikacija, korisničko sučelje, nadzor, upravljanje

SUMMARY

The goal of this task is to develop a user interface on a computer for controlling an industrial robot via programmable logic controller (PLC). Instead of using teach pendant, robot control should be enabled via external interface and allows the user direct access to robotic functions. The developed interface should also allow remote control and supervision of programmable logic controller which is used for controlling the robot. The task was to create a program that allows you to assign control commands in the progremmable logic controller, which are trensferred in to the robot's teach pendant. It is necessary to link all components into a unique system using appropriate communication protocols and to create control algorithms.

Key words: PLC, robot, communication, user interface, supervision, control

1. UVOD

Tema ovog završnog rada je izraditi korisničko sučelje za upravljanje industrijskim robotom preko programabilnog logičkog sklopa (PLC). Korisničko sučelje treba razviti pomoću jednog od programskih jezika na računalu, zatim to sučelje povezati s upravljačkim sklopom te spojiti u jedinstvenu cjelinu kako bi se preko sučelja PLC-a pomicao robot. U današnje vrijeme potreba za ovakvim rješenjima je sve češća jer se tako robotsko upravljanje jednostavno povezuje s okolnom opremom i operaterom. Stoga će sučelje PLC-a biti razvijeno korištenjem mrežnih protokola. Tako će pristup PLC-u, a onda i robotu, biti omogućen putem web preglednika na bilo kojem umreženom računalu ili mobilnom uređaju.

Do povećanog razvoja ovakvih sučelja dolazi zbog potrebe da se na lak i jednostavan način može upravljati robotom. Tome pridonosi nastojanje da se inženjerima omogući kontroliranje radnika preko takvih sučelja što osigurava da ne dođe do kolapsa cijelog sustava ako korisnik napravi neželjenu grešku. Udaljeno upravljanje putem sučelja PLC-a postalo je ključno za automatizaciju procesa i postrojenja. Svaki pogon u industriji koristi određenu vrstu sučelja koja imaju mogućnost povezivanja s ostalim uređajima u proizvodnom procesu. Tako se razvija jedinstveni sustav koji je međusobno povezan s više različitih tehničkih uređaja.

Sučelje je razvijeno pomoću C# programskog jezika u programskom okruženju Microsoft Visual Studio koje je spojeno na odabrani Siemens S7-1200 PLC. PLC je spojen na industrijski robot tvrtke Fanuc, a model robota je M-10iA. Upravljački procesi PLC-a aktivirat će se korištenjem sučelja, a promjena na sučelju dovodi do aktiviranja funkcija robota. Promjene stanja u sustavu praćene su sučeljem na računalu što omogućuje da se nadzor odvija kontinuirano tijekom udaljenog upravljanja.

2. KORISNIČKO SUČELJE

Korisničko sučelje (eng. *User Interface - UI*) definirano je kao sve grafičke, vizualne i tekstualne informacije koje sustav ili program prikazuje korisniku tako da korisnik ima na raspolaganju jednu ili više ulaznih varijabli kao što su tipkovnica, miš ili mikrofon preko kojih obavlja interakciju sa sustavom. U području industrijskog dizajna, interakcija čovjek - računalo predstavlja prostor koji služi kao interakcija između ljudi i strojeva. S ljudskog stajališta cilj ovakve interakcije je osposobiti učinkovito upravljanje i kontrolu stroja. Stroj odnosno robot natrag šalje informacije koje pomažu operateru da donese ispravnu odluku. Primjeri ovakvog koncepta korisničkog sučelja uključuju interaktivne aspekte računalnih operativnih sustava, ručnih alata, kontrolu korisnika i kontrolu procesa. [2]

Slika 1. Korisničko sučelje koje je dio Apple-ovog operativnog sustava

Danas je u svakom sustavu ili uređaju, u kojemu je prisutna digitalna tehnologija, vidljiv nekakav oblik korisničkog sučelja. Počevši od mobitela i računala u kojima je korisničko sučelje jedan od važnijih segmenata, preko automobila do kućanskih aparata kojima se danas može upravljati i glasom. Općenito, cilj dizajna korisničkog sučelja je stvaranje jednostavnog, učinkovitog i ugodnog korisničkog sučelja kojim se može koristiti kako bi upravljanje strojem dovelo do željenog rezultata. To znači da operater mora osigurati minimalni ulaz za postizanje

željenog izlaza koje se prikazuje na sučelju, kao i da sučelje minimalizira neželjene rezultate koji se mogu dogoditi pri korisnikovom unosu podataka.[2]

2.1. Povijest korisničkog sučelja

Povijest grafičkog korisničkog sučelja, shvaćena kao korištenje grafičkih ikona i pokazivačkog uređaja za upravljanje računalom, obuhvaća razdoblje od pet desetljeća inkrementalnih preciziranja, izgrađenih na nekim stalnim osnovnim načelima. Nekoliko dobavljača stvorilo je vlastite sustave prozora na temelju neovisnog koda, ali s osnovnim elementima koji zajednički definiraju WIMP paradigmu; prozor (eng. Window), ikona (eng. Icon), meni (eng. Menu) i uređaj za pokazivanje (eng. Pointing device).[2]

Šezdesetih godina prošlog stoljeća, Douglas Engelbart razvio je mrežni sistem (NLS eng. oN-Line System). Ovo je računalo imalo ugrađeni pokazivač miša i više prozora koji se koriste za rad hiperteksta. Dizajn se temeljio na djetinjastim primitivima za koordiniranje oka, umjesto upotrebe zapovjednih jezika, korisnički definiranih makro postupaka ili automatiziranih transformacija podataka, kao što su kasnije koristili stručnjaci za odrasle.[2]

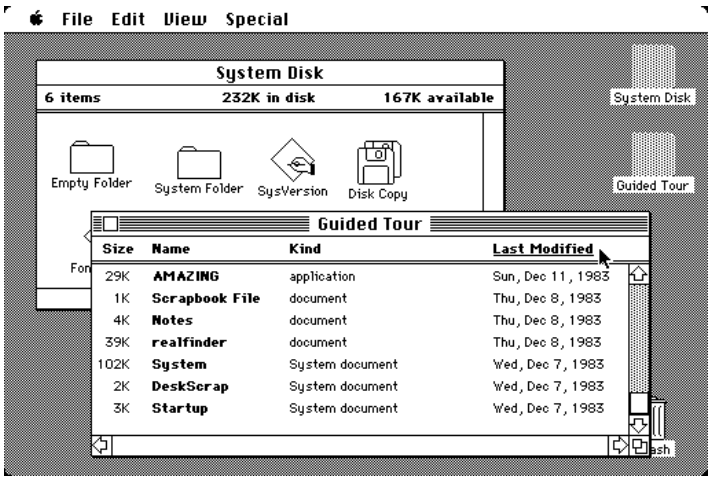
Slika 2. Prototip miša Douglasa Engelberta

Engelbartov rad izravno je doveo do napretka u Xerox PARC-u. Godine 1973., Xerox PARC razvio je Alto osobno računalo. Imao je ekran sastavljen od grupiranih bitova i bio je prvo računalo koje je pokazivalo radnu površinu i grafičko korisničko sučelje (GUI). Nije se radilo o komercijalnom proizvodu, već o nekoliko tisuća jedinica koje su izgrađene i korištene u PARC-u, drugim XEROX uredima i na brojnim sveučilištima. Alto je uvelike utjecao na

dizajn osobnih računala tijekom kasnih 1970-ih i ranih 1980-ih, posebice Tri Rivers PERQ, Apple Lisa i Macintosh te prve radne stanice Sunca.[2]

Slika 3. Radna stanica XEROX Star-a

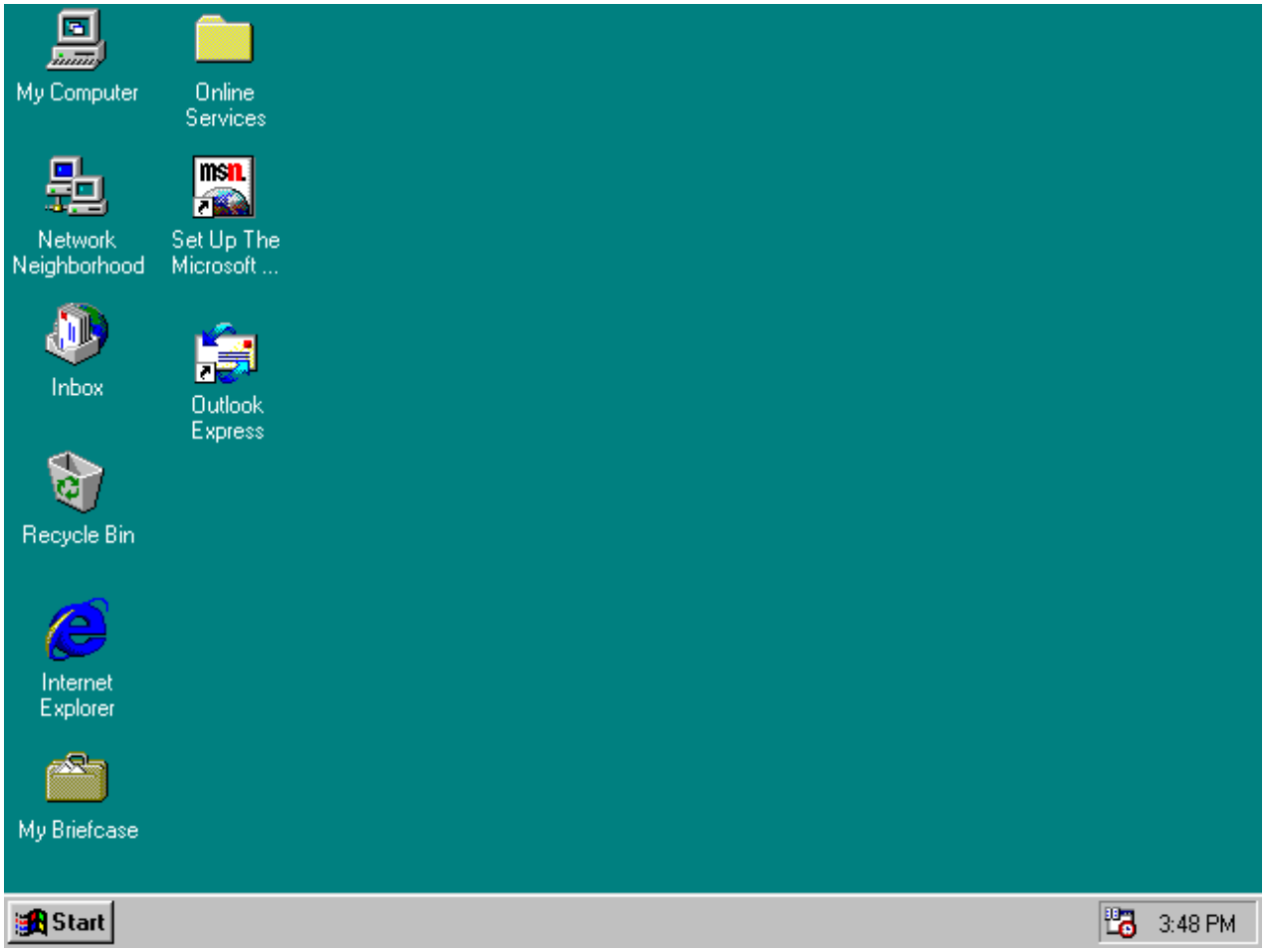
Počevši od 1979. godine, Apple na čelu sa Steve Jobsom nastavlja razvijati takve ideje. Tako su 1983. godine predstavili Lisu, koja je sadržavala grafičko sučelje visoke razlučivosti. Razmjerno pojednostavljeni Macintosh, objavljen 1984. godine i dizajniran da bude jeftiniji, bio je prvi komercijalno uspješan proizvod koji koristi sučelje prozora s više ploča te radnu površinu u kojoj su datoteke izgledale kao komadi papira. [2]



Slika 4. Korisničko sučelje Lisa kompjutera

Nakon računala Lisa tvrtka Apple napravila je računalo koji je imao karakteristike programa i grafičkog sučelja kao i Lisa, ali s manjim ekranom (512x384 piksela), 128 kb memorije, te nemogućnosti korištenja više od jednog programa u isto vrijeme. To računalo, pod imenom

Macintosh, je ostavilo puno značajniji trag na tržištu te je doprinijelo komercijalizaciji osobnih računala. Macintosh i Lisa su prethodili razvoju Microsoftovog korisničkog sučelja gdje je po prvi puta uvedeno da se miš koristi kao obavezna ulazna jedinica. 1985. godine izlazi operativni sustav Windows 1.0 i on zamjenjuje tadašnji sustav MS-DOS, operativni sustav baziran na disketama i naredbenim linijama. Nakon toga izlazi Windows 2.0, koji nije imao puno utjecaja na razvoj ostalih sustava. Tek kada je izašao Windows 3.0 Microsoft je postao jaka konkurencija tvrtki Apple i započinje njihov međusobni uspon. Ubrzo nakon toga izlaze verzije Windows 95. i 98. koji su nam svima poznati i oni postaju osnova današnjih Windows operativnih sustava s grafičkim korisničkim sučeljem koje koriste ljudi na globalnoj razini.[2]



Slika 5. Izgled grafičkog korisničkog sučelja Windows 95

2.2. Vrste korisničkih sučelja

Postoje tri vrste sučelja koja se koriste ovisno o njihovoj potrebi i namjeni, a to su:

1. Komandno korisničko sučelje
2. Tekstualno korisničko sučelje
3. Grafičko korisničko sučelje

2.2.1. Komandno korisničko sučelje

Komandno korisničko sučelje (*CLI, eng. Command Line Interface*) koristi se za interakcije s računalnim programom gdje je korisnik ili operater upisuje naredbe u komandni prozor u obliku uzastopnih linija teksta. Primarno se koristilo kao interakcija s računalnim sustavima na računalnim terminalima tijekom 1970-ih te 1980-ih na osobnim računalnim sustavima uključujući MS-DOS, Apple DOS i ostale sustave. Danas se rijetko koristi jer se većinom upotrebljavaju grafička korisnička sučelja i ostvaruju interakcije s izbornicima. Međutim, mnogi softverski programeri, administratori sustava i napredni korisnici još uvijek ga koriste kako bi učinkovitije obavljali zadatak, pristupili programima i programskim značajkama koje nisu dostupne preko grafičkog sučelja. [4]

Slika 6. Komandno korisničko sučelje

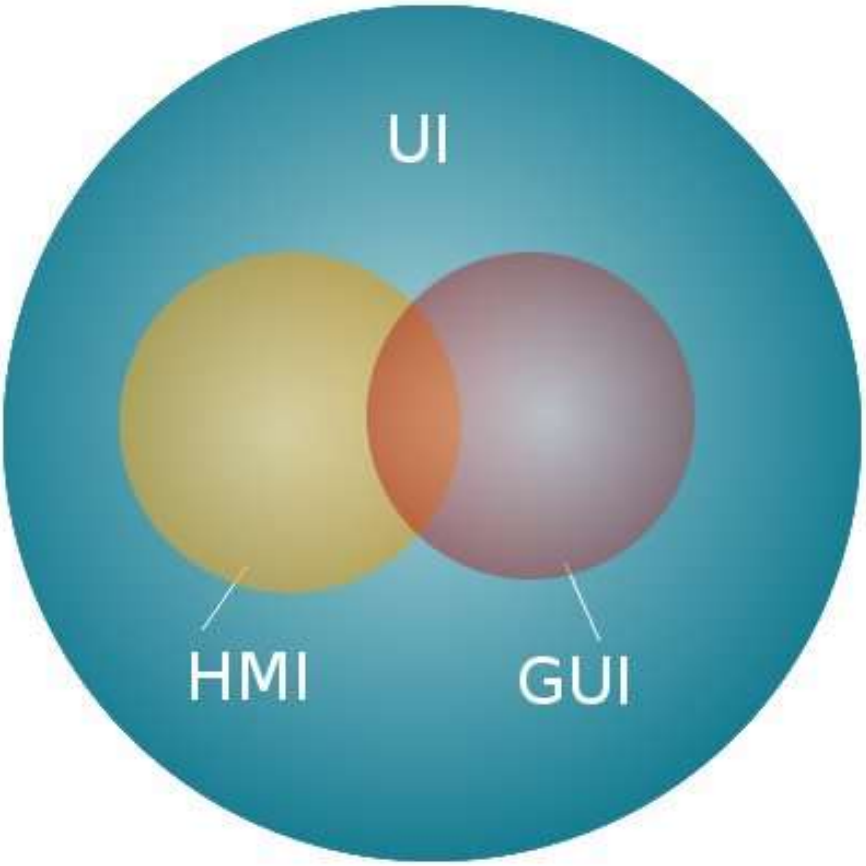
2.2.2. Tekstualno korisničko sučelje

Korisničko sučelje temeljeno na tekstu *(TUI, eng. Textual User Interface*) ne odlikuje grafičkim prikazom, već svugdje koristi tekstualna predstavljanja funkcija. Po načinu korištenja sličnije je grafičkom nego komandnom sučelju gdje se naredbe biraju između postojećih elemenata unutar tog sučelja i prikazuje računalne grafike u tekstualnom kodu. Jedan od primjera gdje se ovakav tip sučelja koristi je BIOS koji je jezgra svakog računalnog sustava.

Slika 7. Tekstualno korisničko sučelje

2.2.3. Grafičko korisničko sučelje

Grafičko korisničko sučelje (*GUI, eng. Graphic User Interface*) koristi grafički prikaz koji se može kontrolirati pomoću različitih uređaja za unos kao što su miš, tipkovnica i drugi. Na GUI sučelju su izbornici, prozori, gumbi, ikone i drugi koji se mogu lakše i prikladnije oblikovati. S druge strane često se kao dio korisničkog sučelja spominje i sučelje između čovjeka i stroja (*HMI, eng. Human Machine Interface*). HMI je sustav za stvaranje pravih tehnoloških funkcija čime se olakšava njihova interakcija. Gotovo sva tehnička rješenja i učinkovitost HMI-a mogu predvidjeti korisnikovo prihvaćanje svih postojećih rješenja. Uglavnom HMI posjeduje suvremeni koncept u industrijskim postrojenjima s načinom komunikacije između uređaja koji pružaju informacije potrebne za planiranje aktivnosti. HMI je sredstvo kojim operator može pristupiti operativnom sustavu u području koji uključuje operacije, održavanje, razvoj i probleme. U ovome radu sučelje je izgrađeno kao sinteza ovih dvaju sučelja. Grafičko korisničko sučelje korišteno je za prikaz svih željenih funkcija, a HMI kao onaj dio sučelja koji se koristi za interakciju između korisnika i robota.



Slika 8. Korisničko sučelje koje obuhvaća HMI i GUI

3. PROGRAMABILNI LOGIČKI SKLOP – PLC

Programabilni logički sklop (*PLC, eng. Programmable logic controller)* je industrijsko digitalno računalo koje je prilagođeno za kontrolu proizvodnih procesa, kao što su linije za montažu, robotski uređaji ili bilo koja aktivnost koja zahtijeva visoku kontrolu pouzdanosti, jednostavnosti programiranja i procesa dijagnoze kvara. U početku su razvijeni za automobilsku industriju kako bi pružili fleksibilne, robusne i lako programirane sklopove za zamjenu ožičenih releja, tajmera i sekvenci, a kasnije i za sve ostale upravljačke procese. Od tada su široko prihvaćeni kao sklopovi za automatizaciju visoke pouzdanosti pogodni za okolinu. PLC je primjer fizičkog sustava u stvarnom vremenu, budući da rezultati izlaza moraju biti proizvedeni kao odgovor na ulazne uvjete u ograničenom vremenu. [7]

Program u PLC-u se izvršava u pet glavnih koraka:

1. Čitanje ulaza
2. Čitanje programa
3. Obrada zahtjeva za komunikaciju
4. Izvršavanje CPU dijagnostike
5. Pisanje izlaza

PLC program kontinuriano se izvodi, tj. izvršava se više puta u petlji sve dok se upravlja sustavom. Na početku svake izvršne petlje, status svih fizičkih ulaza kopira se na područje memorije, ponekad nazvane ulazno/izlaznom (*I/O, eng input/output*) tablicom kojoj procesor može pristupiti. Program potom ide od svoje prve instrukcije pa sve do posljednje. Potrebno je neko vrijeme da procesor PLC-a procijeni sve prepreke i ažurira tablicu ulaza/izlaza.[7]

Kako su PLC-i postali napredniji, razvijene su metode za promjenu slijeda izvršenja programa, a uvedeni su i potprogrami. Ovo pojednostavljeno programiranje može se koristiti za skraćenje vremena izvršenje procesa kod složenih sustava. Na primjer, dijelovi programa koji se koriste samo za pokretanje stroja mogu biti odvojeni od onih dijelova koji su potrebni za rad s većom brzinom. Noviji PLC-i sada imaju mogućnost pokretanja logičkog programa sinkronizirano s I/O skeniranjem. To znači da se I/O učitava u pozadini, a logika očitava i upisuje vrijednosti kao što je to potrebno tijekom razumijevanja logike.[7]

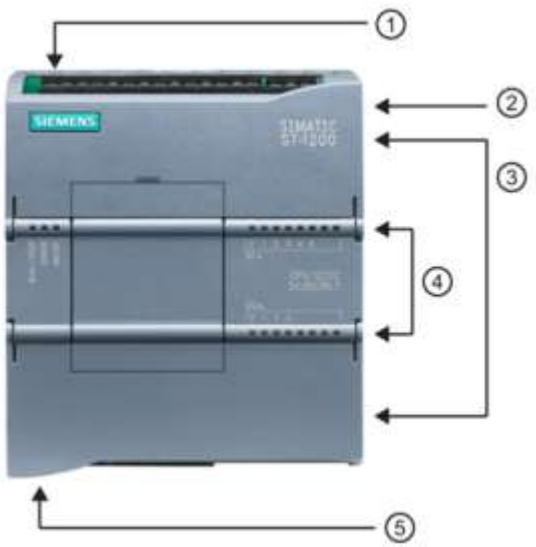
Slika 9. Allen-Bradley PLC američke tvrtke Rockwell Automation

Postoje različite tvrtke koje se bave proizvodnjom PLC-a, a to su Rockwell Automation, ABB, Bosch, Mitsubishi i Siemens. Najpoznatiji proizvođač među njima je tvrtka Siemens čiji PLC-i se najviše koriste u svijetu. Siemens proizvodi PLC-e za kemijsku, petrokemijsku, pomorsku industriju i mnoge druge. Njihovi PLC-i prepoznatljivi su po tome što brzo obrađuju ulaze i prikazuju rezultat koji je vidljiv na izlazu. Negativna strana ovih PLC-a je da tijekom rada ako čak i jedna komponenta ne radi ispravno, sustav postaje nedostupan. Zbog toga stroj prestaje funkcionirati što može dovesti do nepredvidivih grešaka.

Slika 10. PLC japanske tvrtke Mitsubishi

3.1. Siemens PLC S7-1200

Prilikom izrade završnog rada korišten je Siemensov PLC S7-1200. To je kontroler koji pruža fleksibilnost korisniku i moć upravljanja velikim brojem uređaja ovisno o zahtjevima automatizacije. Kako je u ovome radu bilo potrebno spojiti PLC s robotom i računalom, da sve funkcionira kao jedinstvena cjelina, ovaj PLC je odgovarao tome zahtjevu. Njegov procesor (*CPU, eng. Central Processing Unit*) sastoji se od mikroprocesora, integriranog izvora napajanja, ulazno/izlaznih krugova te ulazno/izlazne kontrole kretnje.



① Utor za napajanje
② Utor za memorijsku karticu
③ Odvojivi priključci za ožičenje
④ Status LED dioda ovisno o upaljenim ulazim/izlazima
⑤ PROFINET priključak

Slika 11. Siemens S7-1200 s utorima za spajanje

U sljedećoj tablici navedene su specifikacije Siemensovog PLC-a

Tablica 1. Tehničke specifikacije Siemensa S7-1200 klase CPU 1214 DC/DC/DC

Dimenzije(mm)	110x100x75
Radna memorija	100 Kbajta
Memorija za učitavanje	4 Mbajta
PROFINET priključak	1 priključak
Broj digitalnih ulaza	14
Broj digitalnih izlaza	10
Broj analognih ulaza	2
Dodatni komunikacijski moduli (CM)	3
SIMATIC memorijska kartica	utor:da (kartica:nema)

Za spajanje dodatnih uređaja na PLC, računalo i robot priključen je dodatni modul za komunikaciju. Kako ovaj PLC ima samo jedan PROFINET(Ethernet) priključak, a da bi se povezalo istovremeno na računalo i na robota, spojen je dodatan komunikacijski modul. Korišten je CSM 1277 komunikacijski modul koji omogućava PLC-u da se spoji s još tri dodatna komunikacijska uređaja. Na idućoj slici je prikazano kako je komunikacijski modul CSM 1277 spojen na PLC.

Slika 12. Siemens PLC S7-1200 spojen s modulom CSM 1277

3.2. STEP 7 softver za programiranje

Upravljački sklop Siemens S7-1200 modela CPU1214 DC/DC/DC konfiguriran je pomoću STEP7 Professional V14.0 SP1 inženjerskog softvera u TIA portalu. To je program koji omogućuje da softverski spojimo računalo s PLC-om i tako ostvarimo komunikaciju za prijenos programa iz računala u PLC. Kako TIA služi za slanje jednostavnih programa tako služi i za prijenos onih složenih. U programima se nalazi logika za upravljanje procesom ili sustavom bio to robot, punjenje spremnika, rad stroja ili čitavog pogona u industriji.

Potpuno integrirana automatizacija (*TIA, eng. Totally Integrated Automation*) je strategija (filozofija / arhitektura) u tehnologiji automatizacije, koju je razvio Siemens odjel za Automatizaciju i Pogone 1996. godine. Ova strategija definira interakciju opsežnih pojedinačnih komponenti, alata i usluga kako bi se postigla jedinstvena rješenja za automatizaciju. Interakcija provodi integraciju u četiri razine piramide automatizacije:

1. Razina upravljanja
2. Razina operatera
3. Razina kontrole
4. Područje razvoja

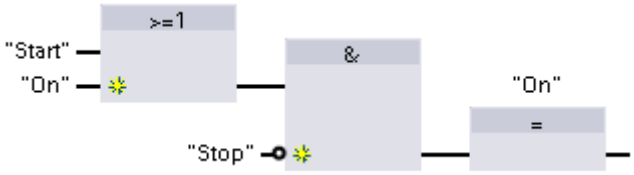
Konzistencija TIA-e pojednostavljuje i smanjuje troškove tvrtkama koje se bave različitim industrijama. [8]

STEP7 je softverski prostor za razvoj. Prilagođen je korisniku za uređivanje i praćenje potrebne logike upravljanja aplikacijom, uključujući alate za upravljanje i konfiguriranje svih uređaja u projektu, kao što su PLC kontroleri i HMI uređaji. Omogućava da se koriste standardni programski jezici za učinkovitost u razvoju kontrolnih programa za aplikaciju. [9]

Programski jezici koji se mogu koristiti unutar ovog softvera su LAD, FBD i SCL. Važno je naglasiti da su ti jezici potpuno slični, a njihova razlika je samo u grafičkom prikazu. Svi jezici imaju jednaku logiku i naredbe kojima se izgrađuje struktura programa.

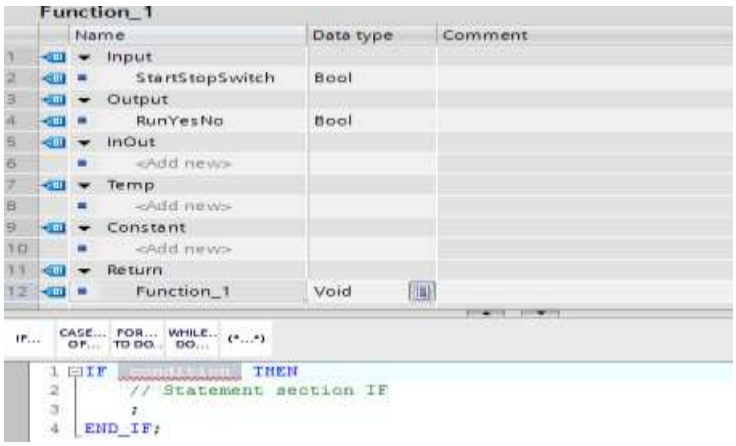
Ljestvičasti dijagram (*LAD, eng. Ladder Diagram*) i Funkcijski blok dijagram (*FBD, eng. Function Block Diagram*) su grafički programski jezici. Pomoću alata iz TIA portala i integrirane funkcionalnosti, kao što je indirektno programiranje, programi se mogu generirati brzinom koja je jednaka brzini generiranja tekstualnih jezika. Kako su to grafički jezici, omogućavaju izvrsnu jasnoću i brzu navigaciju u programskim blokovima. Imaju mogućnost otvaranja i zatvaranja čitavih mreža unutar blok dijagrama, prikazivanja simbola i njihovih adresa, izravnog zumiranja i spremanja izgleda te kopiranja i lijepljenja svih naredba iz jedne mreže u drugu. Zbog svojih karakteristika korišteni su za izgradnju programa i logike u TIA softveru kojime se jednostavno kontrolira ispravnost njihove logike.

Slika 13. Primjer bloka u LAD jeziku



Slika 14. Primjer bloka u FBD jeziku

Strukturirani kontrolni jezik (*SCL, eng. Structured control Language*) je programski jezik temeljen na PASCAL-u za SIMATIC S7 procesore. SCL podržava blok strukturu, kao i grafički jezici, samo u drugačijem obliku. U projektu se mogu uključiti programski blokovi korištenjem bilo koja od navedena tri jezika, a to su SCL, LAD i FBD. Upute SCL-a koriste se standardnim operaterima za programiranje, kao što su pridruživanje (:=) i matematičke funkcije (+ za dodavanje, - oduzimanje, * za množenje i / za dijeljenje). SCL koristi i standardne PASCAL-ove operacije kontrole programa. To su operacije ako-onda-inače, slučaj, ponovi – dokle, idi na i vrati (eng. IF-THEN-ELSE, CASE, REPEAT – UNTIL, GO TO i RETURN). Tajmeri, brojači i mnoge druge instrukcije odgovaraju istim uputama kao i u grafičkim jezicima. [10]



Slika 15. Primjer koda u SCL jeziku

4. INDUSTRIJSKI ROBOT FANUC M-10*i*A

Industrijski robot Fanuc M-10iA je robot sa šest stupnjeva slobode gibanja gdje su svi zglobovi rotacijski. Nazivaju se još i zglobni robot ili zglobna ruka jer po svojem izgledu podsjeća na ljudsku ruku. Tako nazvani se većinom koriste i kada se koriste u ovom kontekstu svi znaju da se radi o industrijskim robotima sa šest osi rotacije. Kada se kaže da robot ima šest osi rotacije misli se na šest stupnjeva slobode gibanja robota. Ovi roboti su vrlo svestrani i koriste se za postupke zavarivanja pa sve do robotskog pomaganja čovjeku gdje robot uzima gotovi dio iz jednog procesa i stavlja ga u sljedeći dio procesa. Još se koriste i u procesima pakiranja, paletizacije, sortiranja te sklapanja različitih dijelova.

Zglobni roboti omogućuju zglobnu rotaciju ili interpolirani pomak u bilo koju točku u prostoru koja je u njegovim granicama.

- Zglob 1 – rotira robota (misli se na bazu robota)
- Zglob 2 – omogućuje pomicanje produžetka donjeg kraka robota naprijed ili nazad
- Zglob 3 – podiže ili spušta gornju ruku robota
- Zglob 4 – rotira gornju ruku robota (ručni valjak)
- Zglob 5 – podiže ili spušta zglob druge ruke robota
- Zglob 6 – rotira ručni dio (zglob na kojem je prihvatnica)

Kretnju robota obavljaju servo motori koji se nalaze u svakom pojedinom zglobu. Kontrolni sustav regulira snagu koja se dovodi svakome motoru i tako se ostvaruje preciznost toga robota. Prednosti ovakvih robota su da ih je lakše uskladiti s referentnim ravninama, jednostavni su za upravljanje i održavanje te ih se jednostavno postavlja za automatski proces injekcijskog prešanja na različitim vrstama i veličinama strojeva za brizganje plastike. S druge strane nedostaci su da mogu biti sporiji od ostalih konfiguracija zbog prirode svojega dizajna.

Slika 16. Industrijski robot FANUC M-10*i*A

4.1. Karakteristike robota

Industrijski robot FANUC M-10*i*A jedan je od najbržih robota u svojoj klasi. Zbog svoje relativno male mase od svega 130kg i njegovog dizajna, ima nosivost od 12kg. To znači da je jako pouzdan i ima odličnu točnost ponavljanja. Kablovi robota integrirani su unutar ruke od trećeg do šestog zgloba zbog čega ne može doći do isprepletanja njegovih kablova. Zbog tankog integriranog držača za kablove unutar njegovih dijelova koristi se na mjestima gdje je radni prostor jako malen, a potreban je brz i precizan kontinuirani rad.

Tablica 2. Karakteristike robota

KARAKTERISTIKE ROBOTA	VRIJEDNOSTI
Broj osi rotacije	6
Ponovljivost [mm]	± 0.03
Masa [kg]	130
Kontroler	R-30*iA*
Integrirani ulazno/izlazni signali	8
Nosivost [kg]	10
Maksimalni raspon [mm]	1422

Robot nije moguće okretati tako da je svaki zglob u maksimalnoj vrijednosti kuta zakreta već postoje određena ograničenja. Prvi zglob A1 može se okretati od -170º do 170º što znači da, iako je to zglob baze, ograničen je kako ne bi došlo do ispreplitanja ili do miješanja kablova koji su unutar ruke robota. Drugi zglob A2 ograničen je kako, robot koji stoji na postolju, ne bi mogao ići ispod postolja. Ako dođe ispod postolja može doći do kolizije gornjeg dijela s postoljem. Zglob A3 je treći zglob i ima široki raspon kuta zakreta jer je to zglob koji okreće prvi krak ruke. Kao i zglob A3 tako i zglob A4 ima široki raspon kuta zakreta. Peti zglob A5 podiže i spušta prihvatnicu te je ograničen kao i drugi zglob A2. I na kraju zglob A6 koji okreće prihvatnicu nema ograničenja već se može okretati od -360º do +360º čime je omogućeno okretanje prihvatnice kako korisnik god to želi. Iduća tablica prikazuje raspon kuta zakreta pojedinog zgloba i maksimalnu brzinu svakog od zglobova.

Tablica 3. Makismalni kut zakreta i kutna brzina pojedinog zgloba

Broj zgloba	Maksimalni kut zakreta [º]	Maksimalna kutna brzina [º/s]
A1	340	225
A2	250	205
A3	445	225
A4	400	420
A5	280	420
A6	720	700

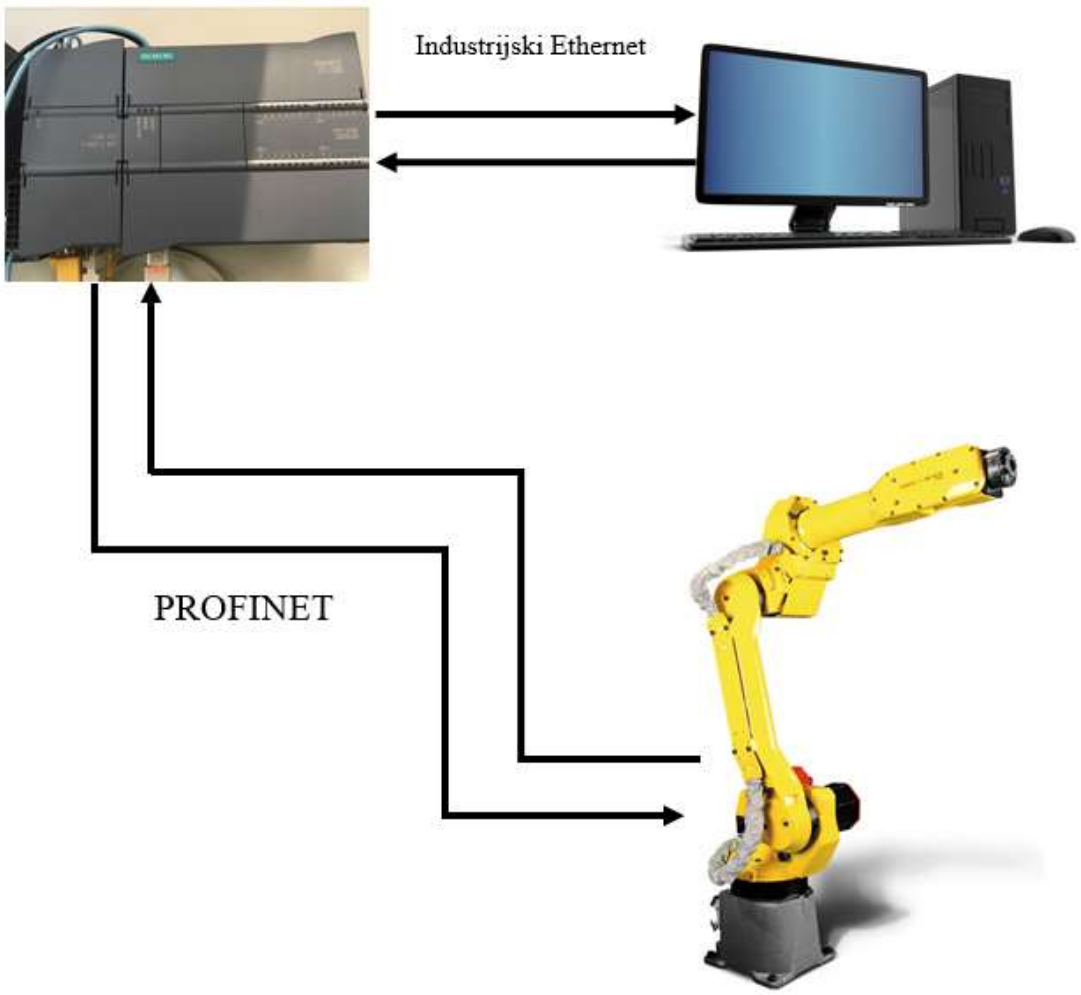
Kontroler robota R-30*i*A smješten je unutar kabineta i povezan s robotom preko industrijskog Etherneta. Iz B-kabineta robot dobiva napajanje, ima dovoljno mjesta za dodatna pojačala ili ulazno/izlazne module. Svrha tog kontrolera je da dobiva potrebne signale iz upravljačke konzole koji pokreću robota odnosno tim signalima se motori motori pale ili gase.

Slika 17. B-kabinet unutar kojega je kontroler R30*i*B

Robotom se upravlja preko robotske konzole (eng. Teach Pendant, iPendant). U nju se upisuju određene funkcije koje robot pretvara u kretnju i izvršava program za gibanje. Na konzoli se nalazi i sigurnosna sklopka koja ako je pritisnuta u bilo kojem trenutku zaustavlja robota. Ovakav način programiranja robota je tzv. „On line“ programiranje gdje korisnik piše program unutar konzole, a robot izvodi kretnju prevođenjem programa.

5. KOMUNIKACIJSKE MREŽE I PROTOKOLI

Komunikacijska mreža je sustav u kojem su računala, strojevi ili PLC-i međusobno povezani i razmjenjuju podatke. Uređaji su najčešće spojeni preko kablova odnosno žičanim putem te koriste zajednički protokol komunikacije kako bi mogli izmjenjivati informacije. Imaju važnu ulogu u svakoj industriji jer niti jedan pogon ne bi funkcioniraju bez da uređaji međusobno ne komuniciraju. Zadatak završnog rada bio je spojiti računalo s PLC-om, te PLC s robotom, odrediti tip komunikacijske mreže i odabrati odgovarajući protokol. Odabran je industrijski Ethernet kao osnovni tip komunikacije između uređaja gdje računalo i PLC komuniciraju korištenjem TCP/IP protokola, a PLC i robot međusobno preko PROFINET-a.



Slika 18. Shema komunikacijskih mreža i protokola

5.1. Industrijski Ethernet

Industrijski Ethernet (IE) je vrsta Etherneta koji se koristi u industrijskom okruženju s protokolima koji omogućuju kontrolu u realnom vremenu. Trenutno je jedna od najraširenijih mreža u sustavima gdje postoji određena automatizacija postrojenja. Protokoli koji se koriste za industrijski ethernet su PROFINET, EtherCAT, EtherNet/IP, MODBUS/TCP. Predstavlja globalnu mrežu koja koristi jedan zajednički industrijski otvoreni protokol (CIP, eng. Common Industrial Protocol).CIP uključuje sve vrste servisa i poruka za primjenu u automatizaciji pri čemu uključuje sinkronizaciju, sigurnost, kontrolu, informacije i konfiguraciju. Može se odnositi i na korištenje standardnih Ethernet protokola sa zaštićenim priključcima i termoizoliranim sklopkama za automatizaciju i kontrolu procesa. Zato je i jedna od najraširenijih mreža jer se može koristiti i u najtežim uvjetima, a da pritom ne dođe do gubitka podataka.

Prednosti industrijskog Etherneta s obzirom na ostale industrijske mreže su:

- Brži prijenos podataka
- Mogućnost udaljenog upravljanja
- Mogućnost korištenja standardnih pristupnih točaka
- Bolja interoperabilnost
- Jednostavni TCP/IP protokol za komunikaciju s uređajima

U današnje vrijeme nemoguće je zamisliti Ethernet bez TCP/IP protokola (eng. Transmission Control Protocol/Internet Protocol). On je osnova internet tehnologije i na njemu se zapravo oživljava čitavi internet i intranet. Korištenjem tih protokola omogućava se uređajima da mogu postojati na globalnoj mreži, ali s druge strane ne garantira da će ti isti uređaji komunicirati međusobno. Zato se i razvio velik broj njihovih inačica kako bi svaki korisnik mogao odabrati svoj način komunikacije između uređaja. Protokoli definiraju pravila komunikacije na mreži te objedinjavaju sve ono što su ljudi nazvali jezikom, bontonom, ali i pismom. Oni su osnova rada svake industrijske mreže i bez njih ne bi bilo komunikacije među uređajima. Osnovne uloge su im da definiraju oblik poruke koja se prenosi, pravila kako, tko i kada može komunicirati na mreži te mehanizme koji su nužni za uspješnu komunikaciju između uređaja i korisnika.

5.2. TCP/IP Protokol

Protokol za kontrolu prijenosa podataka (TCP, eng. Transmission Control Protocol) i internet protokol (IP, eng. Internet Protocol) je skup komunikacijskih protokola koji se koriste za povezivanje mrežnih uređaja koji su povezani internet (Ethernet) kablom. Ovaj paket protokola sastoji se od skupa pravila i postupaka s kojima uređaji komuniciraju. Globalna mreža odnosno internet koristi TCP/IP protokol jer je siguran, pouzdan te se jednostavno uspostavlja i prekida prijenos podatka. Protokoli se služe određenim funkcijama u procesu izmjenjivanja podataka. TCP definira kako se stvaraju kanali komunikacije preko mreže te poruku pretvara u manje pakete prije nego krene njezin prijenos. Tako se poruka sigurno i brzo prenosi na odredište gdje se otpakirava i dobivamo njezinu pravu vrijednost. S druge strane IP definira kako adresirati i usmjeravati svaku poruku prije nego što krene njezin prijenos za osiguranje dolaska na zamišljeno odredište. Zbog toga svako računalo prilikom spajanja s drugim ima vlastitu IP adresu i ona je jedinstvena.

Postavlja se pitanje kako ovaj skup protokola funkcionira? Na temelju modela klijent/poslužitelj dolazi do komunikacije u kojoj korisnik ili stroj pruža uslugu drugom računalu ili stroju odnosno poslužitelju u toj mreži. Ovakav način komunikacije računala s PLC-om ostvaruje uspješnu komunikaciju jer jedan drugome mogu slati podatke. Protokoli se zajedno klasificiraju bez adrese što znači da svaki zahtjev klijenta smatra se novim i on nije povezan s prethodnim zahtjevima. Time se oslobađa prijenosni kanal koji se može neograničeno koristiti.

Tablica 4. OSI modul naspram TCP/IP modula

OSI	TCP/IP
Aplikacijski sloj	Aplikacijski sloj
Prezentacijski sloj	
Sesijski sloj	
Transportni sloj	Transportni sloj
Mrežni sloj	Mrežni sloj
Podatkovni sloj	Sloj mrežnog pristupa
Fizički sloj	

OSI (eng. Open Systems Interconnection) je model mrežne komunikacije koju je sastavio ISO (eng. International Organization for Standardization) 1977. godine. ISO organizacija je stvorena zbog potrebe standardizacijskih tehnologija. OSI model dijeli mrežnu komunikaciju na sedam slojeva od kojih svaki ima određenu ulogu u prijenosu podataka mrežom. Podaci putuju slojevima OSI modela točno određenim redoslijedom. Tako aplikacijski sloj može komunicirati isključivo samo s aplikacijskim slojem na drugom računalu, dok je na istom računalu prezentacijski sloj jedini sloj kojemu može proslijediti podatke. Isto je i s ostalim slojevima. [14]

TCP/IP model nastao je prije OSI modela što je i logično jer je OSI model puno kompliciraniji i detaljniji. Za razliku od OSI modela TCP/IP model je podijeljen u četiri sloja od kojih svaki uključuje određene protokole:

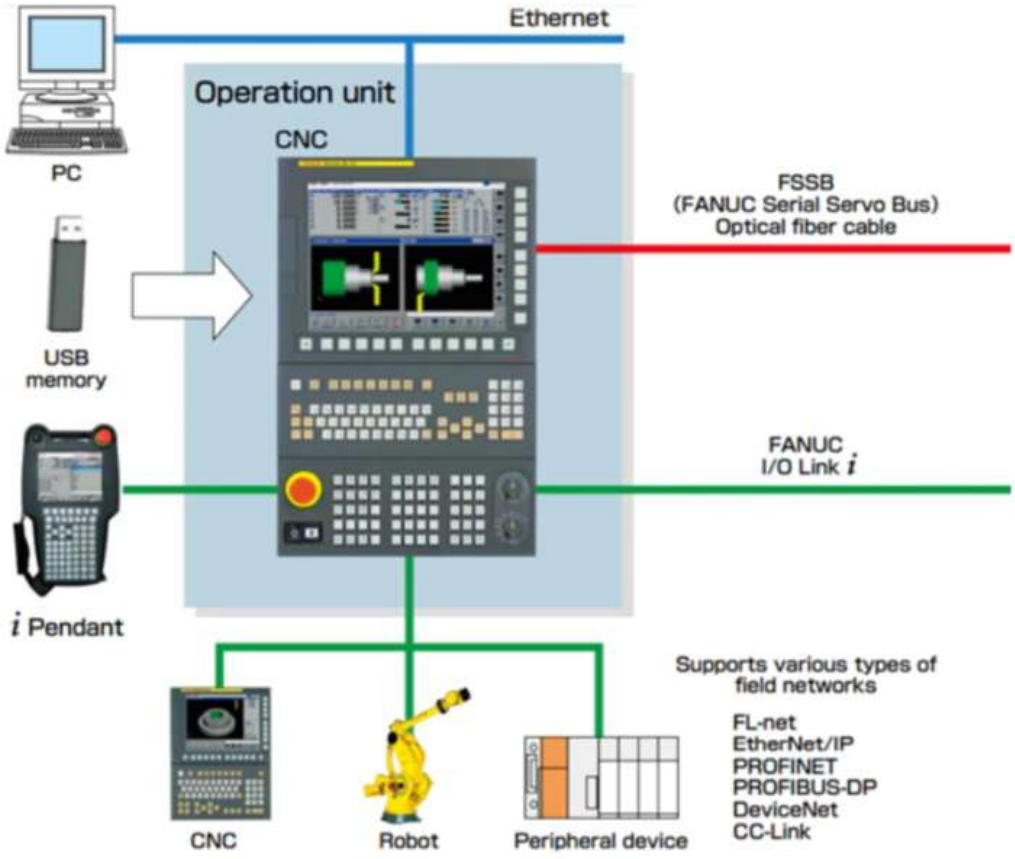
1. Aplikacijski sloj – omogućuje spajanje aplikacije i mrežnog softvera. Tu se nalaze programi koji omogućuju mrežnu komunikaciju.
2. Transportni sloj – odgovoran je za održavanje komunikacije preko mreže. TCP protokol je dio prijenosnog sloja jer upravlja komunikacijom i osigurava kontrolu protoka podatka.
3. Mrežni sloj – naziva se mrežnim slojem jer je IP protokol dio njega, kontrolira pakete podataka te povezuje neovisne mreže za prijenos podataka
4. Sloj mrežnog pristupa – sastoji se samo od protokola koji djeluju na vezi mreža – internet komponenta i ona povezuje čvorove ili domaćine u toj mreži. Dio ovog sloja je protokol za lokalnu mrežu (*LAN, eng. Local Area Network*)

5.3. PROFINET

PROFINET (*eng. PROcess Field Net*) je industrijski standard za podatkovnu komunikaciju preko industrijskog Etherneta. Namijenjen je za kontrolu i prikupljanje podataka iz raznih uređaja u industrijskim sustavima. Ovaj standard održava i regulira tvrtka Profibus and Profinet Industrial koja je smještena u gradu Karlsruhe, u Njemačkoj. Profinet obuhvaća dva protokola za komunikaciju, to su PROFINET IO i PROFINET CBA. PROFINET IO se koristi najčešće u industrijskoj automatizaciji i usredotočuje se na razmjenu podataka

programabilnih sklopova. Zbog toga je za uspostavu komunikacije PLC-a s robotom korišten PROFINET IO. PROFINET CBA je komunikacijski industrijski protokol koji pruža DCOM sustav za organizaciju automatizacijskih sustava u mrežu u kojoj oni automatski izmjenjuju podatke na temelju nekih predefiniranih vrijednosti. [15]

PROFINET IO je vrlo sličan Ethernet Profibusu. Dok Profibus koristi cikličku komunikaciju za razmjenu podataka s programabilnim kontrolerima, PROFINET IO koristi ciklički prijenos podataka za razmjenu podataka preko Etherneta. Kao i kod Profibusa, PLC i drugi uređaj moraju imati prethodno razumijevanje strukture podatka i značenja. U oba protokola podaci su organizirani kao utori koji sadrže module s ukupnim brojem ulaza/izlaza. [15]



Slika 19. Prikaz spajanja CNC operacijske jedinice preko raznih protokola

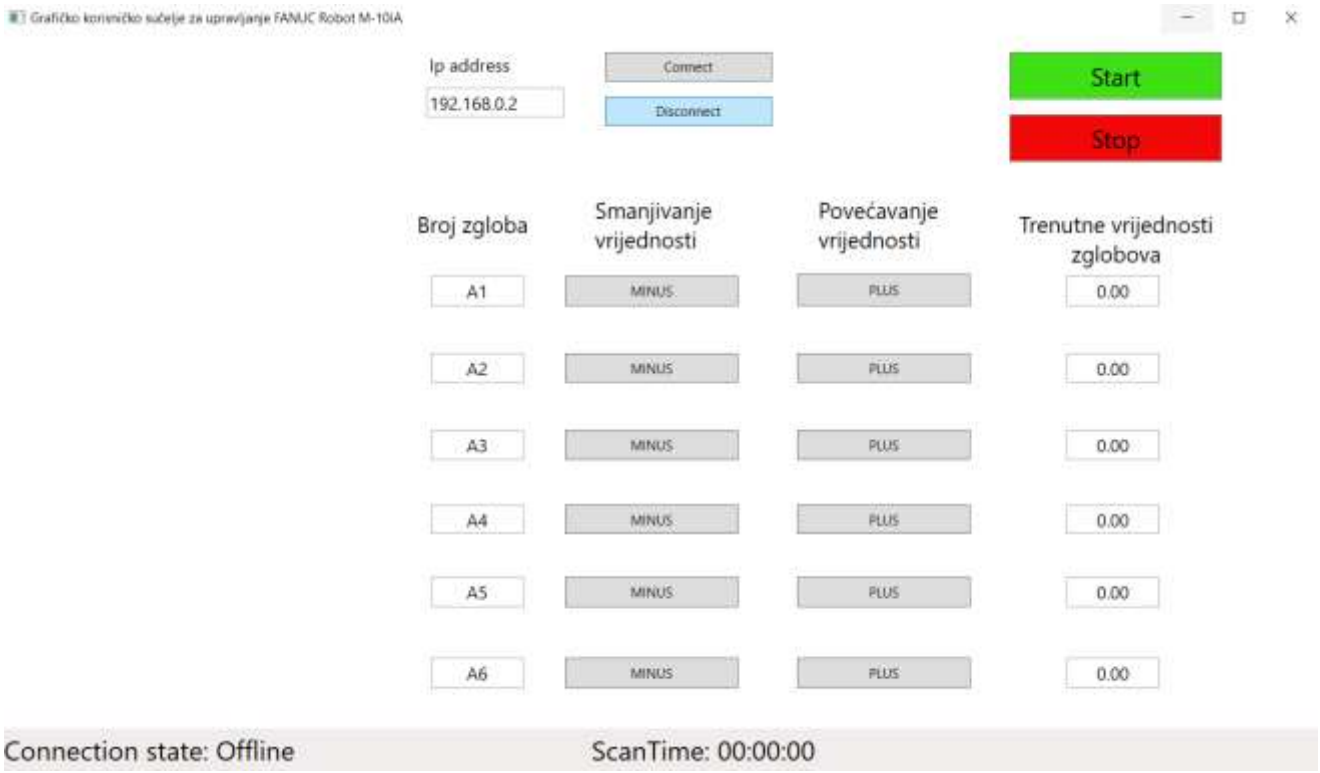
Neke od prednosti PROFINETA su:

- Komunikacijski kanal u stvarnom vremenu omogućuje brzu razmjenu podataka
- Besprijekorna i identična Siemens S7 PLC integracija kao s Profibusom
- Brzo uspostavljanje veze
- Jednostavna instalacija
- Minimalno vrijeme puštanja u pogon i inženjerska podrška

Iako je prvo bilo u planu ostvariti komunikaciju preko Profubusa, odabran je PROFINET. Kako je za ostvarivanje komunikacije robota sa Siemens S7-1200 PLC-om jednaka kao i kod Profibusa korišten je PROFINET. Isto tako zbog brze i jednostavne uspostave veze odabran je PROFINET kao odlična zamjena. Naravno, mogao je biti odabran i bilo koji drugi protokol, ali onda bi došlo do promjena raznih postavki u cijelom sustavu. Na ovaj način odabran je protokol u kojem su spajanje i postavke bile potpuno iste.

6. PROJEKTIRANJE I IZRADA SUČELJA

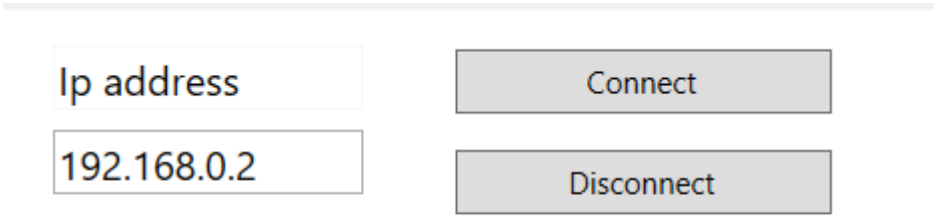
Prvi korak u rješavanju ovog problema bio je dizajniranje korisničkog sučelja. Korisničko sučelje osmišljeno je tako da posjeduje elemente i ima izgled grafičkog korisničkog sučelja, misli se na vizualni prikaz tog sučelja, te da ima elemenata HMI sučelja npr. tipke koje korisnik može aktivirati. Glavni cilj bio je omogućiti udaljenu kontrolu robota preko tipki koje se aktiviraju na sučelju. Kada je određena tipka aktivna, robot mora izvršavati određenu kretnju. Kretnja robota odvija se u smjeru koji mu je korisnik zadao i tako se prepušta potpuna kontrola sučelja korisniku. Jedan od zahtjeva bio je da se robot, koji ima šest stupnjeva slobode gibanja, može preko sučelja kontrolirati svakom trenutku te da se svaki zglob pomiče neovisno o ostalima. Dakle, programski kod treba biti napisan tako da se svih šest tipki mogu aktivirati istovremeno.



Slika 20. Početni izgled sučelja

Sučelje je dizajnirano tako da su na početku napravljene tipke za povezivanje s PLC-om. To su tipke „Connect“ za uspostavu veze te tipka „Disconnect“ kojom se prekida veza i slanje podataka između računala i PLC-a. Ako je komunikacija ostvarena, odnosno mogu se

slati varijable u PLC i čitati njihove vrijednosti na sučelju, u donjem lijevom kutu pojavit će status „Online“ (povezan). Tako će korisnik sučelja znati da je uspostavio vezu s PLC-om. Ako je prikazan status „Offline“ znači da je PLC ili isključen ili je bila pritisnuta tipka „Disconnect“. Ako se pojavi status „ConnectionError“ znači da je došlo do greške u komunikaciji i da upravljanje trenutno nije moguće. Pokraj ovih tipki nalazi se prozor u koji se upisuje Ip adresa PLC-a na koji se spaja. Adresa mora biti jednaka onoj koja je na PLC-u kako bi TCP/IP protokol mogao funkcionirati. Ako je adresa različita prikazuje se stanja „ConnectionError“ i tu komunikacija prestaje.



Slika 21. Početni izgled sučelja

Sljedeće tipke koje je trebalo napraviti su tipke za inicijalizaciju sustava i za prekidanje kretnje robota u bilo kojem trenutku. To su tipke „Start“ i „Stop“. Tipka „Stop“ je sigurnosna kočnica koju korisnik aktivira ako dođe do bilo kakve neželjene kretnje robota gdje bi robot postao opasan za okolinu. Tako je ostvarena sigurnost sučelja da ako korisnik i pogrješi sustav se može zaustaviti, ali isto tako i ponovno pokrenuti.

Slika 22. Tipke za početak/prekid rada sustava

Na temelju izgleda izgled tipki koje se nalaze na upravljačkoj konzoli (eng. Teach Pendant) osmišljeno je dvanaest tipki se koriste za pomicanje zglobova robota. Robot ima šest pomičnih zglobova što znači da svaki zglob treba imati dvije tipke na sučelju. Jedna koja povećava vrijednosti tog zgloba, odnosno pomiče ga u jednu stranu, te druga koja smanjuje vrijednosti, odnosno pomiče ga u drugu stranu. Kako bi korisnik znao vrijednosti svakog

pojedinog zgloba, tj. njegovu poziciju, napravljena su polja za upis vrijednosti pokraj pojedine tipke zgloba (od A1 do A6) koje u svakom trenutku čitaju vrijednosti iz PLC-a i upisuju ih u njegovo odgovarajuće polje.

Slika 23. Izgled tipki za pomicanje robota u sučelju

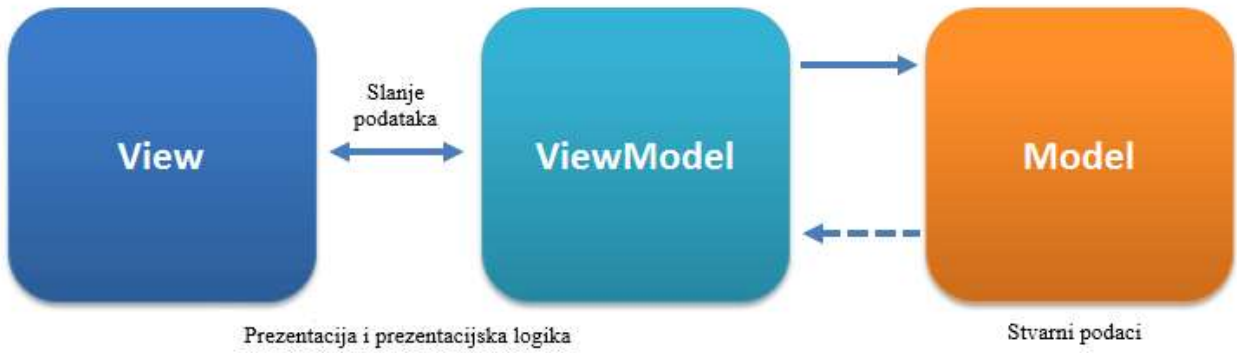
6.1. C# Programski jezik

Za programiranje i izradu sučelja korišten je programsko okruženje Microsoft Visual Studio 2017 u kojemu je programirano koristeći C# programski jezik. C# je objektno orijentiran programski jezik u kojemu svi objekti unutar njega predstavljaju jedan objekt. Objekt predstavlja strukturu koja sadrži neke podatkovne elemente i metode te prikazuje njihovu međusobnu interakciju. Nastao je kao dopuna na nedostatke postojećih Microsoft-ovih jezika C, C++ i Visual Basic. Njegova primjena je raznolika. Koristi se za razvoj softverskih komponenti gdje se lako mogu provjeravati greške u sustavima, prikladan je za razvoj aplikacija za različite uređaje, razvoj raznih web stranica i mobilnih aplikacija te raznih grafičkih i korisničkih sučelja pomoću funkcije WPF (*eng. Windows Presentation Foundation*). WPF oblik je korišten za razvoj ovog sučelja. Iako postoje mnogi drugi jezici u kojima se može razvijati sučelje odabran je C# programski jezik zbog njegove jednostavnosti te lakog implementiranja željenih funkcija u sučelje. Kako bi se povezao vizualni dio sučelja s logičkim korišten je MVVM koncept (eng. framework).

6.2. MVVM

MVVM je zapravo skraćenica riječi Model-Prikaz-ModelPrikaza (eng. *Model-View-ViewModel*) i predstavlja vrsta softverskog koncepta za razvoj i povezivanje sučelja. Ovaj koncept je razvijen u tvrtki Microsoft kako bi se pojednostavilo programiranje korisničkih sučelja temeljenih na nekim događajima. On olakšava odvajanje razvoja grafičkog korisničkog sučelja (GUI) od njegovog logičkog dijela aplikacije. Sastoji se od tri sloja, a to su:

1. Model (eng. *Model*) – definira podatke i logiku
2. Prikaz (eng. *View*) – određuje korisničko sučelje, uključujući sve elemente (gumbe, oznake)
3. Model Prikaza (eng. *ViewModel*) – sadrži logiku koja povezuje model i prikaz



Slika 24. MVVM koncept

6.2.1. Model

Model je ono što označavamo kao objekt neke domene. On predstavlja stvarne podatke ili informacije s kojima se služimo. Najvažnije je da on sadrži samo podatke, ali ne metode i servise koji manipuliraju tim podacima. Nije odgovoran za oblikovanje teksta koji će lijepo izgledati na zaslonu ili dohvaćanje popisa stavki s udaljenog poslužitelja. Model ne zanima slanje podataka već se funkcije za slanje nalaze se u drugim klasama koje rade s modelom. Često je izazov zadržati Model potpuno „čistim“ npr. mogu ostati neki drugi podaci koji su očitani preko modela prikaza i model ne očitava stvarne vrijednosti. [16]

U Modelu je napisan kod koji služi za čitanje vrijednosti iz PLC-a ili da se neke druge vrijednosti upisuju. Model je klasa u kojoj se pozivaju određene funkcije i naredbe iz različitih biblioteka. Za komunikaciju s PLC-om korištena je biblioteka imenom Sharp7.

Sharp7 je implementacija C# sustava koji u sebi sadržava S7Protocol. Implementiran je kao jedinstvena izvorna datoteka koju izravno koristimo u C# projektu za komunikaciju sa Siemens S7 PLC-ovima. Osmišljen je za rad s malim C# hardverom ili čak velikim projektima koji ne zahtijevaju proširene funkcije kontrole. Prednosti biblioteke su da je to potpuno standardni C# kod bez ikakvih ovisnosti o drugim klasama, gotovo svaki hardver s internet adapterom koji može pokrenuti C# program može se spojiti na S7 PLC, potrebna je samo jedna datoteka te nema dodatnih biblioteka za implementaciju. Ova biblioteka također ima već implementirani S7 protokol gdje se koriste TCP/IP protokoli za spajanje na PLC putem internet žice. Odabrana je kako ne bi bilo potrebno raditi vlastite sockete za spajanje na PLC već Sharp7 posjeduje sve u S7Protocolu. Jednostavno pozivom željene funkcije iz Sharp7 biblioteke računalo se spaja na PLC i pomoću naredbi za povezivanje uspostavlja se komunikacija. Iduća slike prikazuju dio koda koji čita vrijednosti iz podatkovnog bloka (eng. Data Block) PLC-a koji je ujedno i dio koda u sloju Model. [17]

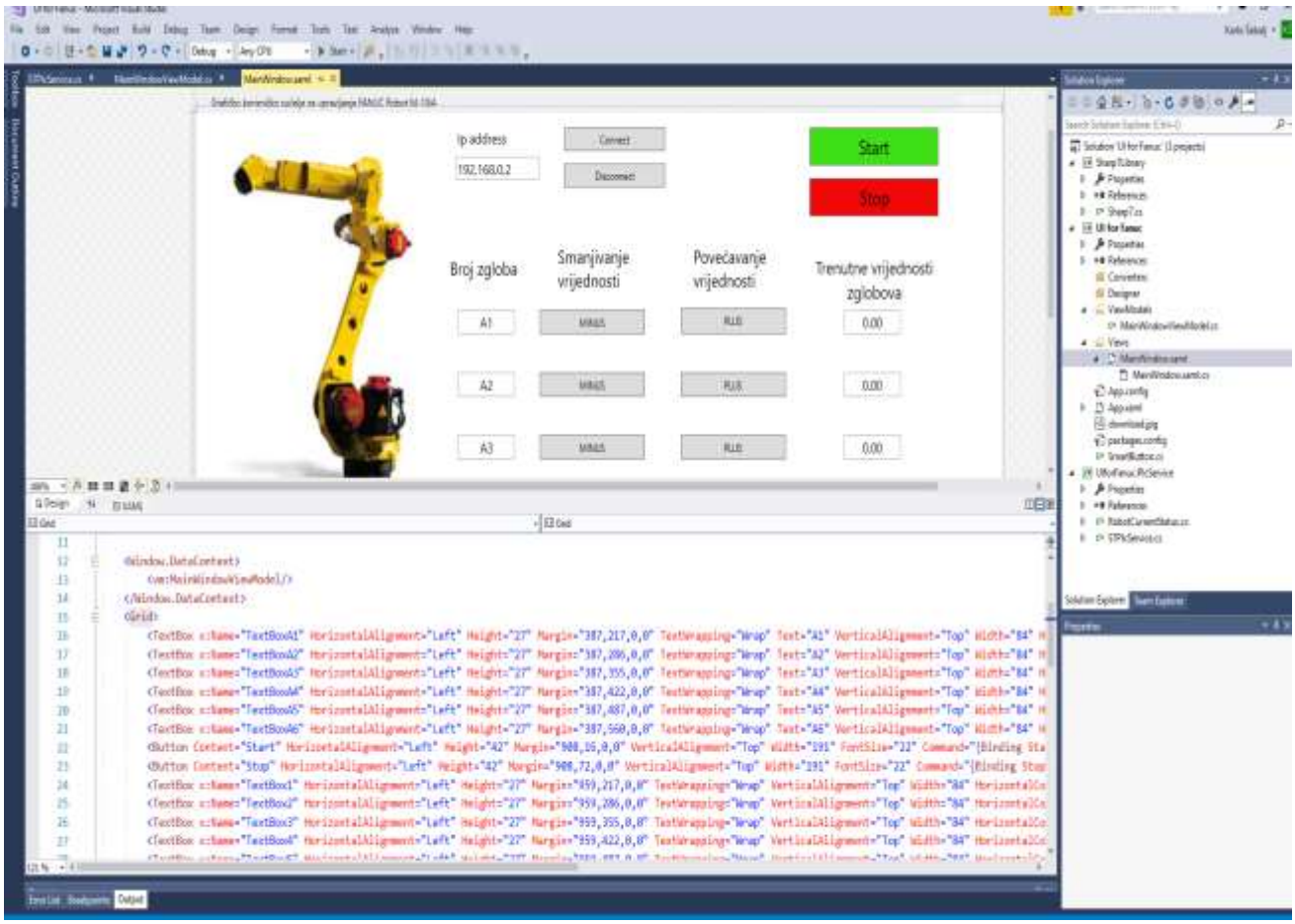
```
private void RefreshValues()
{
    lock (_locker)
    {

        var buffer = new byte[24];
        int result = _client.DBRead(2, 0, buffer.Length, buffer);
        if (result == 0)
        {
            JogA1 = S7.GetRealAt(buffer, 0);
            JogA2 = S7.GetRealAt(buffer, 4);
            JogA3 = S7.GetRealAt(buffer, 8);
            JogA4 = S7.GetRealAt(buffer, 12);
            JogA5 = S7.GetRealAt(buffer, 16);
            JogA6 = S7.GetRealAt(buffer, 20);
        }
        else
        {
            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Read error: " + _client.ErrorText(result));
        }


    }
}
```

Slika 25. Dio koda za čitanje vrijednosti iz PLC-a u sloju model

6.2.2. Prikaz (eng. View)

Prikaz je ono što se zna kada se govori o sučelju i jedina dio s kojim će korisnik biti u interakciji. To je prikaz svih vizualnih elemenata na ekranu pomoću C# aplikacije za sučelje. Prikaz je prostor gdje operateri pokušavaju prikazati vrijednosti u što boljem i ljepšem prikazu kako bi korisniku bilo jasno kako njime upravljati i što te vrijednosti znače. Prikaz također posjeduje metode koje pozivaju određene funkcije, kao što je na primjer korisnikov unos željenih podatka. U ovom sloju korisnik upravlja unosom podataka (pritiskom na tipke, pokretima miša, dodirnim pokretima itd.). Zatim se podaci preko Modela Prikaza šalju iz Prikaza u Model. [16]

U MVVM-u Prikaz je aktivan sloj. Za razliku od pasivnog Prikaza koji nema znanja o modelu, i kojim potpuno kontrolira korisnik, Prikaz u MVVM-u sadrži ponašanja, događaje i veze podataka koji u konačnici zahtijevaju poznavanje sloja Model i Model Prikaza. Iako se ovi događaji i ponašanja mogu grupirati na svojstva, pozive metoda i naredbe, Prikaz je na kraju sam odgovoran za izvršavanje svojih metoda.[16]



Slika 26. Izgled prikaz sloja

6.2.3. Model Prikaza (eng. ViewModel)

Model Prikaza je ključni dio ova tri sloja jer se u njemu nalazi prezentacijsko odvajanje ili koncept koji odvaja Model od Prikaza. Umjesto da je Model svjestan toga da je na Prikazu formatirani izgled vrijednosti zgloba, on samo posjeduje tu informaciju i ne zna što se događa u Prikazu. Model prikaza je kao sklop koji povezuje ta dva sloja. On može uzeti podatke iz Prikaza i prenijeti ih u sloj Model ili obrnuto. On može stupiti u interakciju s Modelom da pretvori neka svojstva koja će poslati na Prikaz. Dakle Model Prikaza uzima vrijednost zgloba koje je očitao iz PLC-a preko sloja Model i šalje taj podatak u Prikaz. U Prikazu dolazi do oblikovanja tog broja idealnim za prikazivanje na sučelju, odnosno kako je određeno s dvije decimale.

Prikaz i Model Prikaza komuniciraju putem podataka koji ih povezuju, poziva na metode, svojstava, događaja i poruka. Model Prikaza ne izlaže samo modele, već druga svojstva i naredbe (poput informacija o stanju PLC-a npr. trenutno je stanje „Online“ tj. uspostavljena je komunikacija). Prikaz obrađuje vlastite događaje koje je korisnik napravio pritiskom na tipku na sučelju, a zatim ih prebacuje na Model Prikaza putem naredbi. Na primjer pritiskom na tipku „Start“ na sučelju model prikaza preko naredbe „iCommand“ prebacuje vrijednost iz Prikaza u Model.

```
private void OnPlcServiceValuesRefreshed(object sender, EventArgs e)
{
    ConnectionState = _plcService.ConnectionState;
    JogA1 = _plcService.JogA1;
    JogA2 = _plcService.JogA2;
    JogA3 = _plcService.JogA3;
    JogA4 = _plcService.JogA4;
    JogA5 = _plcService.JogA5;
    JogA6 = _plcService.JogA6;
    ScanTime = _plcService.ScanTime;

}
```

Slika 27. Kod u modelu prikaza koji uzima vrijednosti zglobova iz sloja model

S7PlcService.cs | MainWindowViewModel.cs | MainWindow.xaml

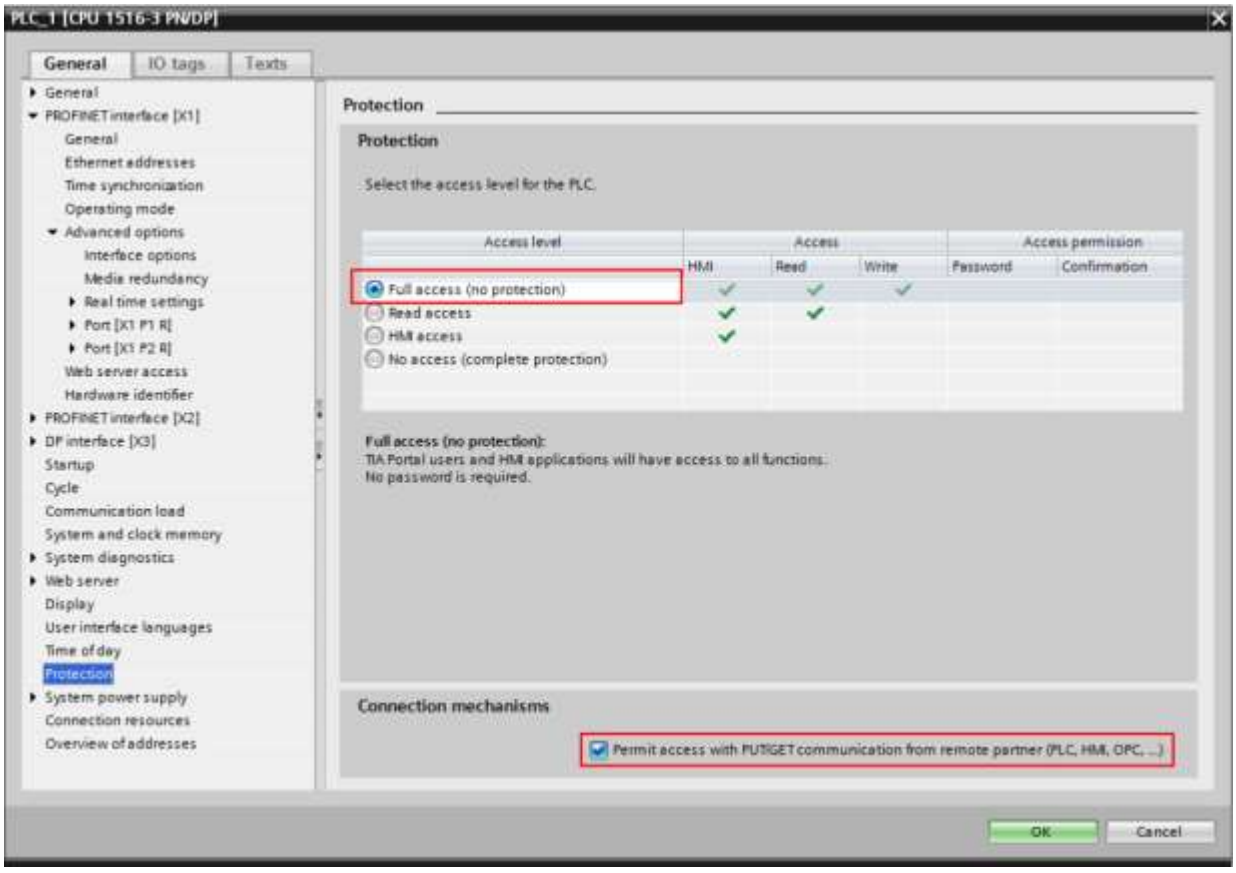
UI for Fanuc — UI_for_Fanuc.ViewModels

```
81              set { Set(ref _scanTime, value); }
82          }
83          private TimeSpan _scanTime;
84
85          public ICommand ConnectCommand { get; private set; }
86
87          public ICommand DisconnectCommand { get; private set; }
88
89          public ICommand StartCommand { get; private set; }
90
91          public ICommand StopCommand { get; private set; }
92
93          public ICommand MinusA1Command { get; private set; }
94
95          public ICommand PlusA1Command { get; private set; }
96
97          public ICommand MinusA2Command { get; private set; }
98
99          public ICommand PlusA2Command { get; private set; }
100
101         public ICommand MinusA3Command { get; private set; }
102
103         public ICommand PlusA3Command { get; private set; }
104
105         public ICommand MinusA4Command { get; private set; }
106
107         public ICommand PlusA4Command { get; private set; }
108
109         public ICommand MinusA5Command { get; private set; }
110
111         public ICommand PlusA5Command { get; private set; }
112
113         public ICommand MinusA6Command { get; private set; }
114
115         public ICommand PlusA6Command { get; private set; }
116
117         S7PlcService _plcService;
```

Slika 28. Primjer koda za dobivanje vrijednosti ako je tipka aktivirana u prikazu

7. POVEZIVANJE SUČELJA S PLC-OM

Drugi korak u rješavanju ovog problema bio je spojiti sučelje s PLC-om kako bi se mogli slati/primati podaci između sučelja i PLC-a. Računalo i PLC povezani su preko internet žice, a protokol za njihovu komunikaciju je TCP/IP. Kako bi taj protokol mogao funkcionirati Ip adrese, na PLC-u i u sučelju, moraju biti potpuno iste. Ako bi bile različite došlo bi do greške u komunikaciji i ona ne bi bila ostvariva. U prethodnom poglavlju opisana je biblioteka Sharp7 koja je korištena u programskom okruženju Microsoft Visual Studio, a služi za povezivanje Siemensovih S7 PLC-a s aplikacijom na računalu. Sharp7 sadrži naredbe koje možemo koristiti za čitanje i upisivanje vrijednosti u Podatkovni Blok (eng. Data Block) PLC-a, za uspostavu i prekid veze, kontrolu PLC-a preko sučelja, informacije o tom PLC-u te mnoge druge. Navedene naredbe korištene su za uspostavu komunikacije. Prije nego se uspostavi komunikacija potrebno je bilo omogućiti pristup svim podacima koji se nalaze u Data Blokovima PLC-a.



Slika 29. Postavke za pristup podacima u PLC-u

7.1. Podaci u Data Blokovima

Data Blokovi u Tia Portalu služe za spremanje podataka koji se kasnije koriste u Ladder dijagramu za programiranje. Sve vrijednosti koje su poslane iz sučelja prvo se spremaju u podatkovne blokove. Prvi podatkovni blok služi za upisivanje vrijednosti koje su istinite (eng. true) ili neistinite (eng. False). Tipke „Start“ i „Stop“ su logičke varijable koje imaju dva stanja. Drugi podatkovni blok služi za spremanje vrijednosti zglobova koje PLC dobiva od robota. To su real vrijednosti veličine četiri bajta. Kako bi se mogle očitati vrijednosti iz sučelja one moraju biti spremljene na ovaj način. Posljednji blok s podacima služi za postavljanje tipki u istinu ili neistinu koje se mijenjaju ovisno o korisnikovom aktiviranju tipke za upravljanje pojedinim zglobom. Na idućoj slici je prikaz Data Blocka 2 (DB2) koji se koristi za spremanje pozicije robota u PLC-u.

Data_block_2

	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static									
2	StatusZglob1	Real	0.0	0.0						
3	StatusZglob2	Real	4.0	0.0						
4	StatusZglob3	Real	8.0	0.0						
5	StatusZglob4	Real	12.0	0.0						
6	StatusZglob5	Real	16.0	0.0						
7	StatusZglob6	Real	20.0	0.0						

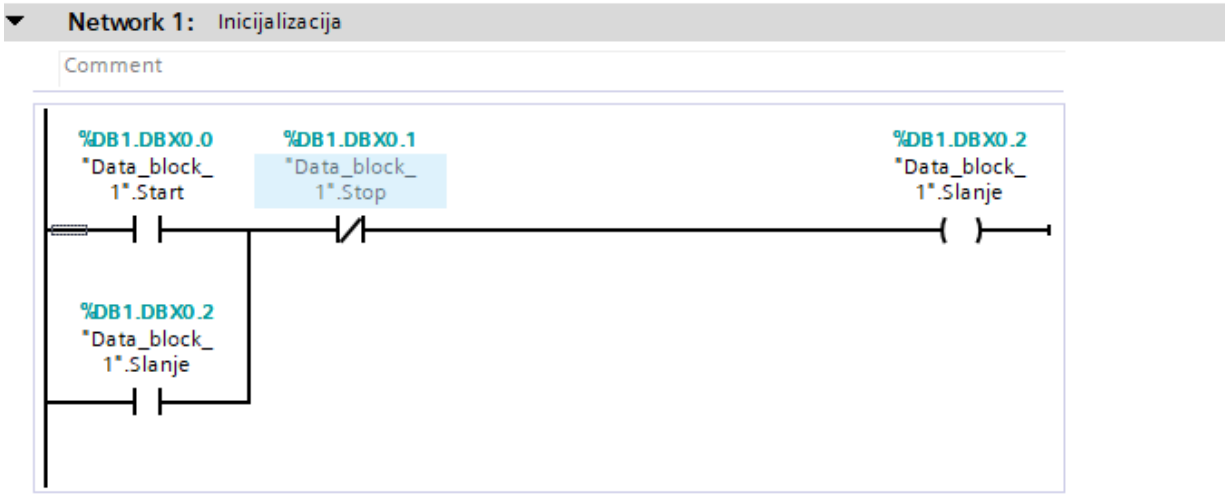
Slika 30. Izgled Podatkovnog Bloka 2

U tablici na slici prikazan je dio koji se zove „Offset“ i to je zapravo najvažniji dio kako bi se mogla očitati vrijednost pomoću Sharp7 biblioteke. Da bi se pojavilo taj stupac, morao se omogućiti pristup registrima PLC-a kako je opisano na stranici prije. Bez te oznake Sharp7 biblioteka ne razumije koje vrijednosti očitava iz podatkovnog bloka i mogu se dobiti krive vrijednosti, koje kad budemo kasnije mijenjali pomoću programa u PLC-u, mogu robota dovesti u neželjenu poziciju.

7.2. Pisanje programa u Ladder jeziku za kontrolu robota

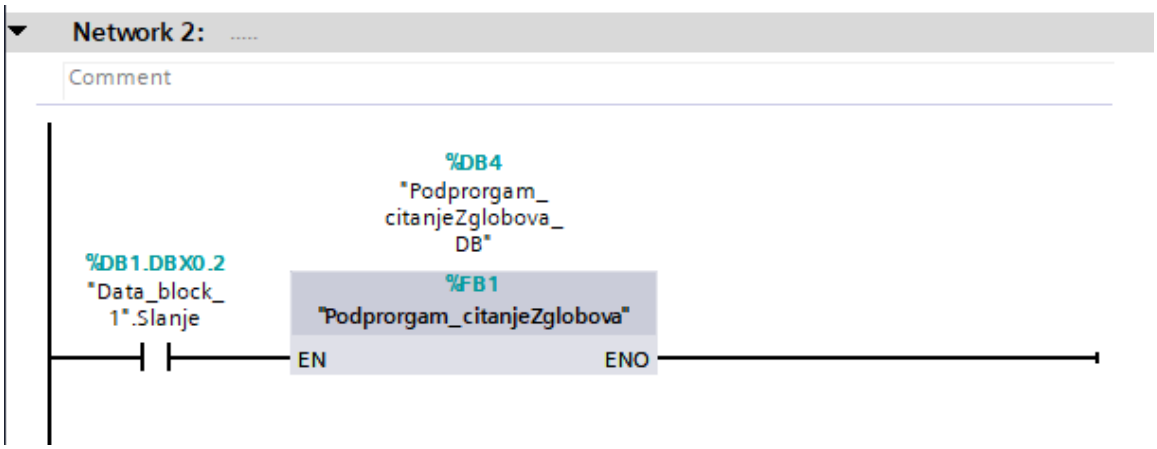
Grupirani podaci u podatkovne blokove bili su pretkorak za izradu programske logike za upravljanje robotom. Programski kod u PLC-u sastoji se od jednog glavnog programa koji ima osam mreža (eng. Networka) i jednog potprograma. Ukratko će biti objašnjeno kako koja mreža radi i koja je njena funkcija u procesu slanja podataka iz sučelja za kretnju robota.

Prva mreža je glavna i ona služi da sustav bude u fazi inicijalizacije. U toj mreži su implementirane tipke „Start“ i „Stop“ za početak rada sučelja. Također je napravljena i najjednostavnija povratna veza koja omogućava da se prilikom pokretanja sustava stanje ne mijenja sve dok ne dođe do prekida sustava pritiskom na tipku „Stop“. Kada se iz sučelja aktiviraju te dvije tipke njihovo stanje se mijenja tj. sučelje šalje vrijednost u PLC da promijeni njihovo logičko stanje iz neistinitog u istinito i obratno. Ovakva ideja gdje postoji jednostavna povratna veza omogućava da sustav bude siguran i održiv. To znači da ako korisnik napravi neželjenu radnju da se sustav odmah zaustavlja i sprječava daljnja nezgoda. Na idućoj slici prikazan je izgled mreže jedan.

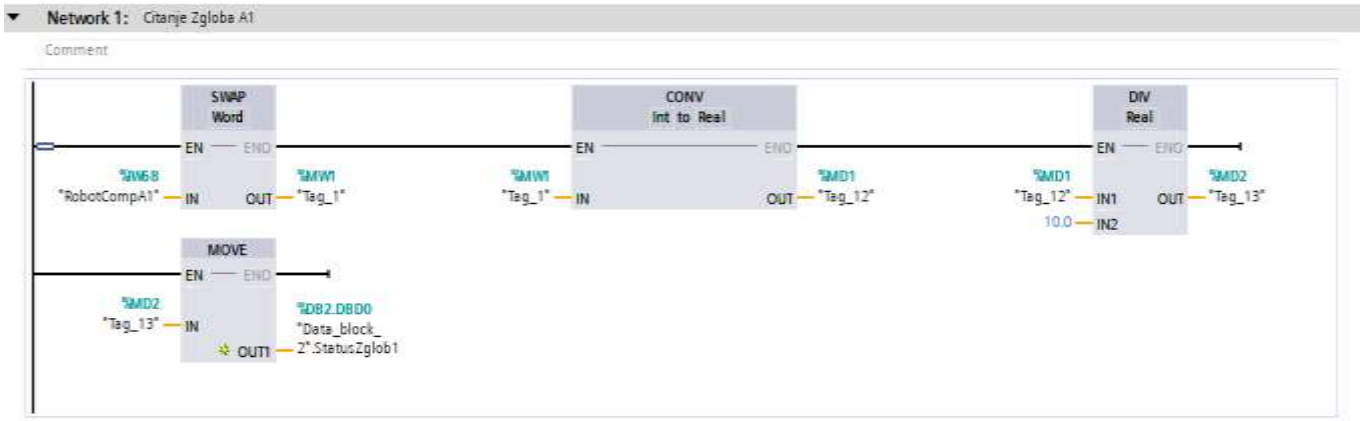


Slika 31. Mreža 1 za pokretanje, prekid i održavanje sustava

Druga mreža poziva potprogram koji služi za očitavanje vrijednosti zglobova iz robota. Za upravljanje varijablama robota potrebno je dobiti stvarne vrijednosti njegovih zglobova. U robotu je napravljen program koji šalje vrijednosti u PLC. Vrijednosti koje PLC dobiva su Integer veličine dva bajta. Kako su vrijednosti u robotu Real, odnosno četiri bajta, prilikom slanja podataka putem PROFINETA dolazi do njihove konverzije i povratne transformacije vrijednosti pomoću blokova u Ladderu. Zato druga mreža poziva potprogram koji pretvara svaku vrijednosti u njezin stvarni iznos koji se dobije iz robota. Potprogram se sastoji od šest mreža, a svaka od njih od četiri bloka. Prvi blok služi za zamjenu redoslijeda znamenki tog broja, drugi za pretvorbu iz Integera u Real vrijednosti, treći za dijeljenje vrijednosti s deset i dobivanje broja s jednom decimalom te zadnji koji pomoću naredbe premjesti (eng. MOVE) sprema vrijednosti u Data Block 2. Sljedeće slike prikazuju poziv potprograma i primjer koda za čitanje zgloba A1.



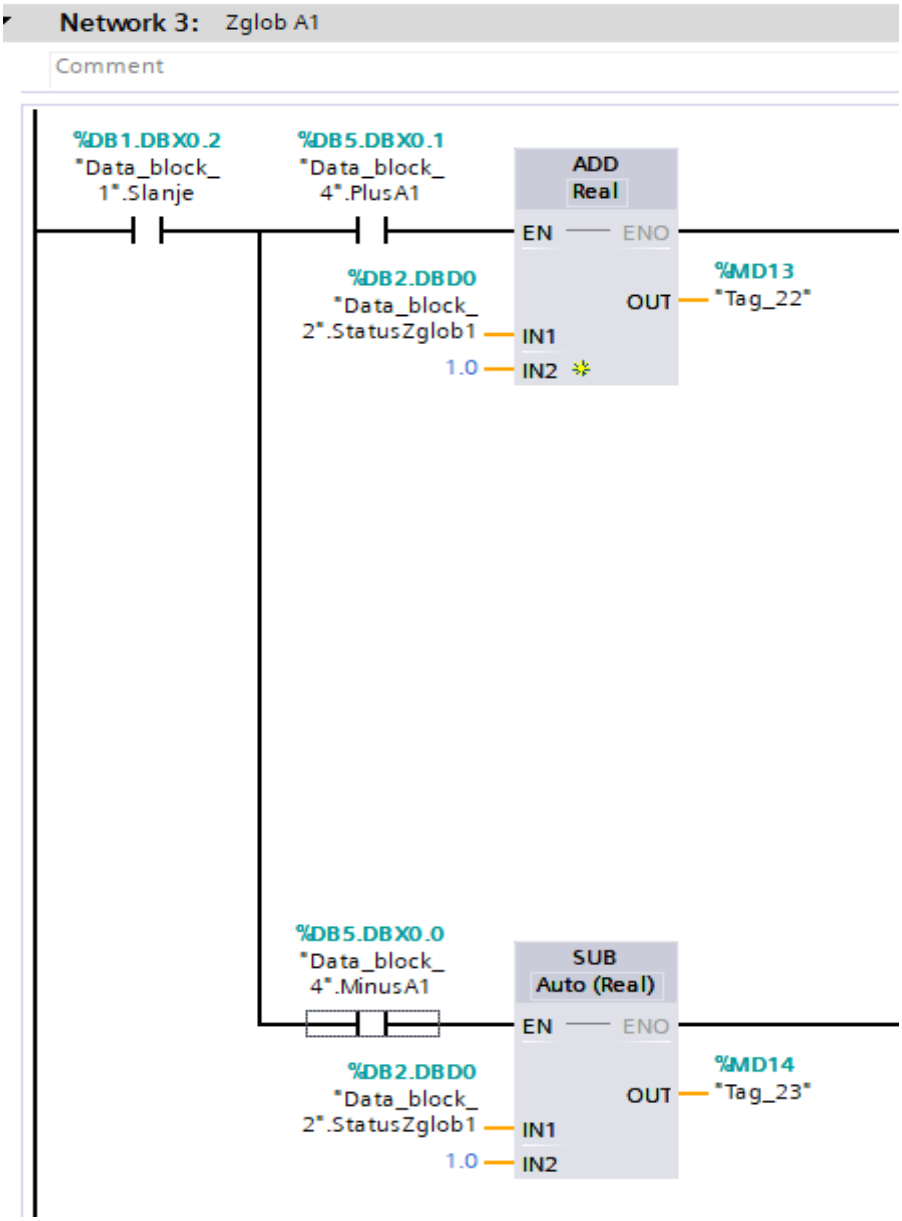
Slika 32. Mreža 2 za poziv potprograma



Slika 33. Mreža 1 unutar potprograma za pretvorbu dobivenih vrijednosti iz robota

Mreže tri, četiri, pet, šest, sedam i osam služe za slanje podataka u robot prilikom čega dolazi do njegovog pomicanja. Ovih šest mreža su iste, no ipak razlikuju se za vrijednosti koje se šalju. Dakle, mreža tri sadrži logiku za prvi zglob odnosno zglob A1, mreža četiri za drugi zglob tj. zglob A2, mreža pet za treći zglob A3 itd. Sve mreže napravljene su istim načinom gdje na početku se inicijalizira sustav aktiviranjem tipke „Start“ te se mreža račva na dva uvjeta. Ako je aktivna tipka „PLUS“, vrijednosti zgloba A1 se povećavaju, a ako je aktivna tipka „MINUS“ dolazi do smanjivanja te iste vrijednosti zgloba A1. Kako se povećavaju odnosno smanjuju vrijednosti zglobova, signali se šalju iz PLC-a u robot i dolazi do pomicanja robota u pozitivnom ili negativnom smjeru ovisno o vrijednosti koja je poslana i tipke koja je aktivna. Jedan od glavnih problema ovoga rada javio se upravo kod logike za pojedinu tipku. Naime, prilikom slanja pozitivnih vrijednosti iz PLC-a u robota sve je funkcioniralo kako treba, ali prilikom slanja negativnih vrijednosti robot je stalno pokazivao grešku i nije se mogao upravljati. Problem je bio u tome da se negativni brojevi drugačije

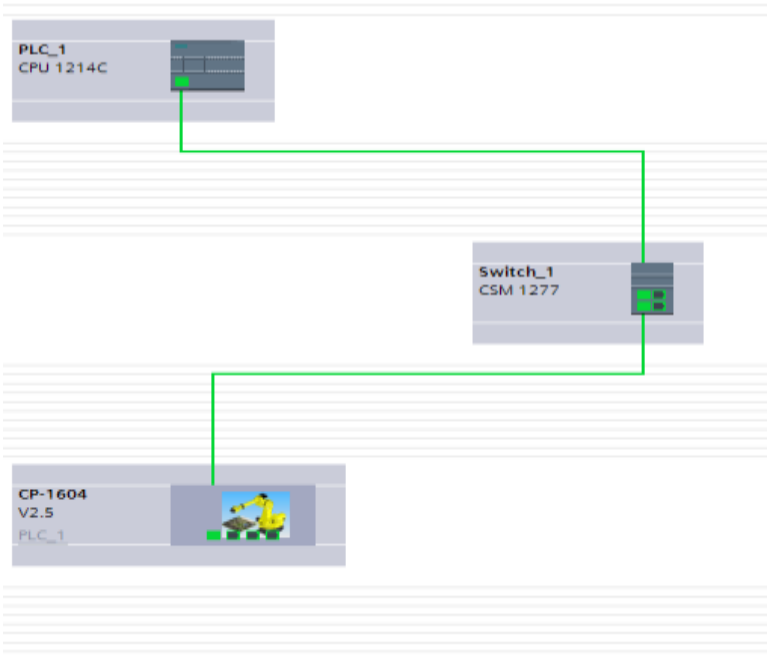
zapisuju nego li pozitivni. Prilikom pretvorbe u PLC-u nije bilo moguće dobiti u robotu prave vrijednosti, a da ne bude greške na upravljačkoj konzoli. Zato je osmišljen poseban način na koji će funkcionirati slanje vrijednosti. Pojedina mreža, nakon što se aktivira tipka „PLUS“ ili tipka „MINUS“, provjerava dva uvjeta. Ako je broj veći ili jednak nuli PLC šalje broj klasično bez dodatnih uvjeta, ali ako je broj manji od nule PLC šalje apsolutnu vrijednost tog broja i aktivira poseban „bit“ u robotu kako će operater znati je li broj pozitivan ili negativan. Ovakav način pokazao se kao dobro rješenje problema i time je riješen problem slanja negativnog broja.



Slika 34. Dio Mreže 3 s blokovima za tipke „PLUS“ i „MINUS“ zgloba A1

8. POVEZIVANJE PLC-A S ROBOTOM

Treći korak u rješavanju ovog problema bio je povezati PLC s robotom preko industrijskog Etherneta te ostvariti komunikaciju između PLC-a i robota preko PROFINET protokola. Robot PLC-u šalje trenutne pozicije zglobova, a PLC robotu uvećane/umanjene vrijednosti te signale u obliku bitova. Kako bi robot i PLC mogli komunicirati instalirana je GSD datoteka koja služi za mrežno povezivanje robota s PLC-om. GSD (*eng. general Station Description*) datoteke su tekstualne datoteke kojima definiramo rad PROFIBUS i PROFINET uređaja. Svaki tehnički uređaj ima vlastitu GSD datoteku, pa tako i robot Fanuc M-10*i*A. GSD datoteka sadrži podatke specifične za uređaj, a to su brzina prijenosa, broj ulazno/izlaznih podataka, dostupni I/O signali itd. Kada je GSD datoteka instalirana potrebno je u TIA portalu u mrežnom prostoru povezati PLC s robotom, označiti PROFINET kao način prijenosa podataka i mapirati ulazno/izlazne podatke. Primjenom ove datoteke omogućuje se kompatibilnost u radu između uređaja različitih proizvođača tj. tvrtke Siemens i tvrtke Fanuc. Na idućoj slici prikazan je način mrežne konfiguracije uređaja u TIA portalu gdje se svakom uređaju postavlja njegova IP adrese. Za razliku od sučelja i PLC-a koji su imali iste IP adrese, postavljanje različitih IP adresa znača da će jedan uređaj u PROFINET komunikaciji biti glavni (eng. *master)* uređaj, a drugi podređeni (eng. *slave)*. PLC je konfiguriran kao glavni uređaj, a robot kao podređeni odnosno onaj koji dobiva naredbe iz PLC-a.



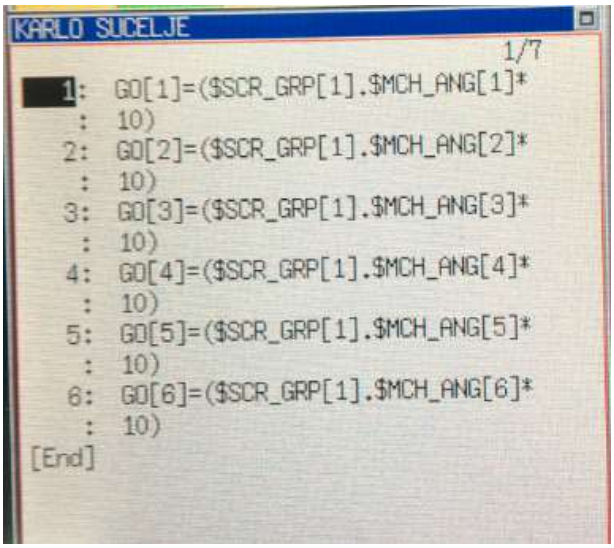
Slika 35. Spajanje PLC-a s robotom pomoću GSD datoteke

8.1. Programiranje robota

Prethodnim koracima omogućeno je da robot prima vrijednosti iz sučelja PLC-a. Kako bi mogli slati trenutne vrijednosti zglobova u sučelje i pomicati robota kada primi određenu vrijednost, potrebno je napisati programe u upravljačkoj konzoli robota. Programski kod upravljačke konzole sastoji se od jednog glavnog programa koji poziva druga dva potprograma. Glavni program je vrlo jednostavan i izvršava se u petlji. Kada se sustav stavi u inicijalni položaj, glavni program se počinje izvršavati i automatski poziva potprograme KARLO_SUCELJE i KARLO_UPISIVANJE. Nakon izvršenja potprogama robot treba promijeniti svoju poziciju ovisno o registru koji se promijenio.

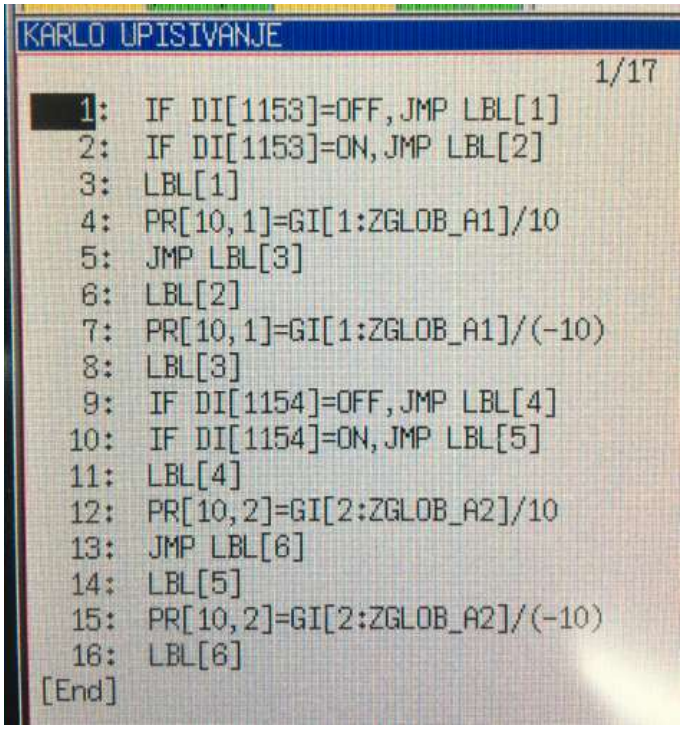
Slika 36. Glavni program u upravljačkoj konzoli

Potprogram KARLO_SUCELJE napisan je da se cijelo vrijeme izvodi u pozadini upravljačke konzole i očitava vrijednosti zglobova robota od zgloba A1 pa sve do zgloba A6. Kada se u bilo kojem trenutku robot pomakne ili njegova pozicija se promijeni program to pošalje u PLC, a PLC u sučelje. Vrijednosti odnosno položaj zglobova tako će biti u svakom trenutku prikazani na sučelju i kako će pojedina tipka za pomicanje robota biti aktivirana tako će se i ta vrijednost automatski prikazivati. Kada se ovaj program ne bi odvijao u pozadini dobivala bi se vrijednost zgloba tek kada bi se robot bio zaustavio. Znači vrijednosti bi ostale iste od početka, pa sve do kraja kretnje robota. Taj način naravno ne bi imao smisla jer je cilj bio da sučelje kontinuirano prati i prikazuje trenutnu vrijednost pojedinog zgloba.



Slika 37. Potprogram KARLO_SUCELJE

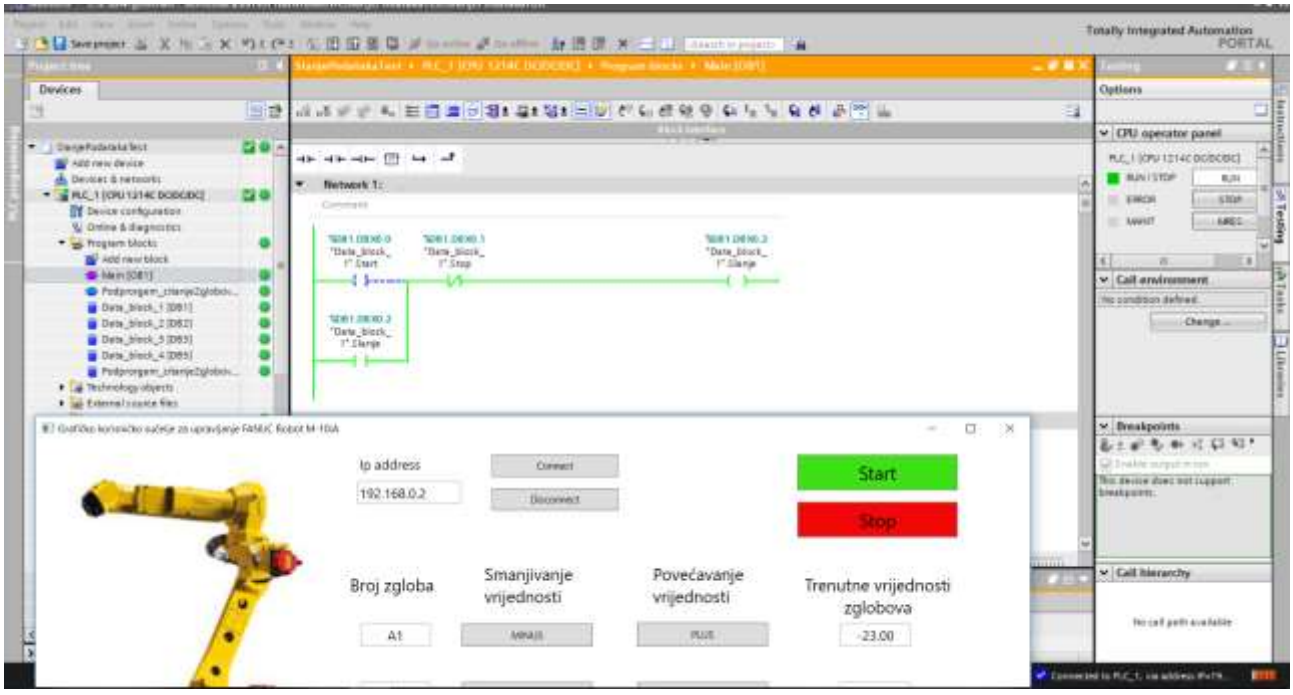
Drugi potprogram KARLO_UPISIVANJE služi za upisivanje vrijednosti, koje su poslane iz PLC-a u robot, u pozicijske registre robota. Kako su vrijednosti koje robot dobije oblika Integer treba pretvoriti u Real vrijednost dijeljenjem s deset ili s minus deset. U prošlom poglavlju opisan je problem koji se javio prilikom slanja negativnih brojeva u robot. Robot dobiva apsolutne vrijednosti i u njegovim ulaznim signalima pali se „bit“ koji PLC aktivira kada je poslan negativni broj. Kao i u programu u PLC-u, svaki zglob ima dva uvjeta. Ako je broj pozitivan dijeli se s deset, a ako je negativan s minus deset. Ovdje je riješen drugi dio problema slanja negativnog broja.



Slika 38. Drugi potprogram KARLO_UPISIVANJE

9. TESTIRANJE SUČELJA

Zadnji korak pri rješavanju problema bio je testirati sustav i povezati sučelje, PLC i robot u jedinstvenu cjelinu. Prva provjera bila je hoće li sučelje aktiviranjem tipke „Connect " povezati se putem TCP/IP protokola na PLC te hoće li trenutni status na sučelju pisati "Online". Isto tako hoće li se, ako se pritisne tipka „Stop" pa „Disconnect" , prekinuti komunikacija. Ovdje je provjerena i tipka „Start" kojom se sustav dovodi u inicijalni položaj. Aktiviranjem tipke „Connect" pa tipke „Start" na PLC-u se umjesto plave boje na mreži pojavljuje zelena i sustav je zbog povratne veze samoodrživ. Tipka „Start" se automatski nakon što je bila aktivirana sama deaktivira kako bi mogla funkcionirati tipka „Stop". Nakon aktiviranja tipke „Stop" došlo je do pojave plave boje na mreži signala i tako je dokazano da logika ovih tipki funkcionira.



Slika 39. Testiranje tipki Connect, Disconnect, Start i Stop

Kako bi sustav funkcionirao neke greške trebalo je namjerno učiniti. Testiranje je provedeno aktiviranjem dvije različite tipke mišem i dovođenjem robota u krajnji položaj pojedinog zgloba. Tako dva zgloba robota su mijenjaju položaj robota, a pritiskom na tipku „Stop" cijeli sustav se prekida i robot se zaustavlja.

Prvi problem, koji se javio prilikom testiranja tipki za pomicanje robota, bio je da ako je aktivirana tipka „MINUS“ za zglob A1, ne mogu se aktivirati ostale tipke za smanjivanje vrijednosti. Koristeći funkciju „await Task.Run“ riješen je problem. Na taj način može se aktivirati bilo koja druga tipka osim aktivirane.

Slika 40. Prikaz normalnog rada sučelja

Druga prepreka bila je kod aktiviranja tipki za povećavanje odnosno smanjivanje vrijednosti. Ako je aktivna tipka za smanjivanje vrijednsoti mogla se aktivirati i tipka za povećavanje. Tako je došlo do greške u logici robota i na upravljačkoj konzoli pojavila se greška. Kako bi se onemogućilo da korisnik tako nešto može napraviti napisan je dodatni uvjet za svaku tipku za povećavanje ili smanjivanje vrijednosti. Koristeći zastavice (eng. flags) unutar sloja model napisana je funkciju koja prolazi kroz dva uvjeta. Prvi provjerava je li suprotna tipka pritisnutoj tipki ugašena, ako je izgašena program nastavlja dalje, ako ta tipka nije izgašena odnosno ona je neistinita tipku je moguće pritisnuti ali se ona neće aktivirati.

10. ZAKLJUČAK

U današnje vrijeme, sve se više izrađuju aplikacije i sučelja za upravljanje sustavima, pogonima i robotima. Većinom se koriste već gotovi industrijski HMI paneli na kojima se razvija sučelje, što omogućuje jednostavnije spajanje i određen je način prijenosa podataka, no moguće je razviti sučelje za korištenje na bilo kojem web pregledniku bez korištenja dodatnih HMI panela, kao što je prikazano u ovom radu. Sučelje je zamišljeno da bude jednostavno i sigurno za korištenje kako bi korisnik mogao njime lakše upravljati. Iako je postupak bez korištenja HMI panela nešto teži i zahtjevniji, naglasak je bio na učenju procesa spajanja i komunikacije PLC sučelja i robota. Na taj način je ostvareno udaljeno upravljanje robota preko upravljačkih procesa programabilnog logičkog sklopa. Programi u logičkom sklopu oblikovani su tako da povezuju upravljačke funkcije sučelja i robota te da osiguravaju sustav ukoliko se pojavi neka greška. Kako bi se dodatno osigurao sustav, programi u upravljačkoj jedinici sadrže logiku kojom se sprječavaju moguće greške u kretnji robota.

Ovaj način izrade sučelja je osobito pogodan u industriji jer ne zahtijeva dodatni industrijski panel niti dodatni protokol za povezivanje uređaja pa se na taj način može uštedjeti. Korišteno je računalo kao osnovno sredstvo za rad te internet kablovi za umrežavanje svih uređaja u sustavu. Programiranjem sučelja omogućena je laka implementacija dodatnih opcija koje se mogu jednostavno promijeniti u bilo kojem trenutku s bilo koje lokacije u kojoj je dostupan internet.

Cilj ovog rada je bio uspostaviti vezu kako bi se preko sučelja moglo upravljati robotom te očitati vrijednosti koje robot šalje preko PLC-a u sučelje, što je i ostvareno. Razvijeno sučelje predstavlja rješenje za velik broj procesa u automatizaciji u kojima se ovakvo sučelje jednostavno može implementirati za upravljanje raznim industrijskim uređajima. Kroz nekoliko godina vidjet će se i napredak ovakvih sučelja koja će u potpunosti zamjeniti upravljačku konzolu.

LITERATURA

[1] Kraut, B.: Strojarski priručnik, Tehnička knjiga Zagreb, 1970.

[2] https://en.wikipedia.org/wiki/User_interface

[3] https://en.wikipedia.org/wiki/History_of_the_graphical_user_interface

[4] https://en.wikipedia.org/wiki/Command-line_interface

[5] https://en.wikipedia.org/wiki/Programmable_logic_controller

[6] https://en.wikipedia.org/wiki/Totally_integrated_automation

[7] https://support.industry.siemens.com/cs/document/107623221/simatic-s7-s7-1200-programmable-controller?dti=0&lc=en-DE

[8] https://w3.siemens.com/topics/mea/en/tia-portal/controller-sw-tia-portal/simatic-step7-professional-v11/iec-programming-languages/pages/default.aspx#LAD_20and_20FBD_20_e2_80_93_20graphic_20programming_20languages

[9] https://www.fanuc.eu/fi/en/robots/robot-filter-page/m-10-series/m-10ia-10m

[10] Jenčić, S: Industrijske računalne mreže, Sveučilište u Splitu, 2015.

[11] https://searchnetworking.techtarget.com/definition/TCP-IP

[12] http://www.umag.hr/sadrzaj/dokumenti/NATJECAJ_informaticki_referent_Uvod_u_racunalne_mreze_Visoko_uciliste_Algebra.pdf

[13] https://www.rtaautomation.com/technologies/profinet-io/

[14] https://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained

[15] Nardella, D.: Sharp7, Reference Manual, 2016

11. PRILOZI

I. Primjer koda u C#

II. Primjer koda u upravljačkoj jedinici Fanuc-a

III. CD-R disc

I. C# kod

1. Kod sloja Model

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Timers;
using Sharp7;


namespace UIforFanuc.PlcService
{
    public class S7PlcService
    {
        private bool MinusA1Flag = false;
        private bool PlusA1Flag = false;
        private bool MinusA2Flag = false;
        private bool PlusA2Flag = false;
        private bool MinusA3Flag = false;
        private bool PlusA3Flag = false;
        private bool MinusA4Flag = false;
        private bool PlusA4Flag = false;
        private bool MinusA5Flag = false;
        private bool PlusA5Flag = false;
        private bool MinusA6Flag = false;
        private bool PlusA6Flag = false;
        private readonly S7Client _client;
        private readonly System.Timers.Timer _timer;
        private DateTime _lastScanTime;
        private volatile object _locker = new object();

        public S7PlcService()
        {
            _client = new S7Client();
            _timer = new System.Timers.Timer();
            _timer.Interval = 100;
            _timer.Elapsed += OnTimerElapsed;
        }

        public ConnectionStates ConnectionState { get; private set; }
        public double JogA1 { get; private set; }
        public double JogA2 { get; private set; }
        public double JogA3 { get; private set; }
        public double JogA4 { get; private set; }
        public double JogA5 { get; private set; }
        public double JogA6 { get; private set; }
        public TimeSpan ScanTime { get; private set; }
        public event EventHandler ValuesRefreshed;

        public void Connect(string ipAddress, int rack, int slot)
        {
            try
            {
                ConnectionState = ConnectionStates.Connecting;
                int result = _client.ConnectTo(ipAddress, rack, slot);
                if (result == 0)
                {
                    ConnectionState = ConnectionStates.Online;
```

```
                    _timer.Start();
                }
                else
                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Connection
error: " + _client.ErrorText(result));
                    ConnectionState = ConnectionStates.Offline;
                }
                OnValuesRefreshed();
            }
            catch
            {
                ConnectionState = ConnectionStates.Offline;
                OnValuesRefreshed();
                throw;
            }
        }

        public void Disconnect()
        {
            if (_client.Connected)
            {
                _timer.Stop();
                _client.Disconnect();
                ConnectionState = ConnectionStates.Offline;
                OnValuesRefreshed();
            }
        }

        public async Task WriteStart()
        {
            await Task.Run(() =>
            {
                int writeResult = WriteBit("DB1.DBX0.0", true);
                if (writeResult != 0)
                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write
error: " + _client.ErrorText(writeResult));
                }
                Thread.Sleep(30);
                writeResult = WriteBit("DB1.DBX0.0", false);
                if (writeResult != 0)
                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write
error: " + _client.ErrorText(writeResult));
                }
            });
        }

        public async Task WriteStop()
        {
            await Task.Run(() =>
            {
                int writeResult = WriteBit("DB1.DBX0.1", true);
                if (writeResult != 0)

                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write
error: " + _client.ErrorText(writeResult));
                }
                Thread.Sleep(30);
                writeResult = WriteBit("DB1.DBX0.1", false);
```

```
                if (writeResult != 0)
                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write
error: " + _client.ErrorText(writeResult));
                }
            });
        }

        public async Task  MinusA1()
        {
            await Task.Run(() =>
            {
                if (PlusA1Flag == false)
                    {
                    if (MinusA1Flag == false)
                    {
                        MinusA1Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.0", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                    else
                    {
                        MinusA1Flag = false;
                        int writeResult = WriteBit("DB5.DBX0.0", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }

                }

            });
        }

        public async Task PlusA1()
        {
            await Task.Run(() =>
            {
                if (MinusA1Flag == false)
                {
                    if (PlusA1Flag == false)
                    {
                        PlusA1Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.1", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                    else
                    {
                        PlusA1Flag = false;
```

```
                        int writeResult = WriteBit("DB5.DBX0.1", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }

        public async Task MinusA2()
        {
            await Task.Run(() =>
            {
                if (PlusA2Flag == false)
                {
                    if (MinusA2Flag == false)
                    {
                        MinusA2Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.2", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                    else
                    {
                        MinusA2Flag = false;
                        int writeResult = WriteBit("DB5.DBX0.2", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }
public async Task PlusA2()
        {
            await Task.Run(() =>
            {
                if (MinusA2Flag == false)
                {
                    if (PlusA2Flag == false)
                    {
                        PlusA2Flag = true;
                        int writeResult2 = WriteBit("DB5.DBX0.3", true);
                        if (writeResult2 != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult2));
                        }

                    }
                    else
                    {
```

```
                        PlusA2Flag = false;
                        int writeResult2 = WriteBit("DB5.DBX0.3", false);
                        if (writeResult2 != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult2));
                        }

                    }
                }
            });
        }

        public async Task MinusA3()
        {
            await Task.Run(() =>
            {
                if (PlusA3Flag == false)
                {
                    if (MinusA3Flag == false)
                    {
                        MinusA3Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.4", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                    else
                    {
                        MinusA3Flag = false;
                        int writeResult = WriteBit("DB5.DBX0.4", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }

        public async Task PlusA3()
        {
            await Task.Run(() =>
            {
                if (MinusA3Flag == false)
                {
                    if (PlusA3Flag == false)
                    {
                        PlusA3Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.5", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
```

```
                    else
                    {
                        PlusA3Flag = false;
                        int writeResult = WriteBit("DB5.DBX0.5", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }

        public async Task MinusA4()
        {
            await Task.Run(() =>
            {
                if (PlusA4Flag == false)
                {
                    if (MinusA4Flag == false)
                    {
                        MinusA4Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.6", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                    else
                    {
                        MinusA4Flag = false;
                        int writeResult = WriteBit("DB5.DBX0.6", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }

        public async Task PlusA4()
        {
            await Task.Run(() =>
            {
                if (MinusA4Flag == false)
                {
                    if (PlusA4Flag == false)
                    {
                        PlusA4Flag = true;
                        int writeResult = WriteBit("DB5.DBX0.7", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }
```

```
                    }
                    else
                    {
                        PlusA4Flag = false;
                        int writeResult = WriteBit("DB5.DBX0.7", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }
                    }
                }
            });
        }

        public async Task MinusA5()
        {
            await Task.Run(() =>
            {
                if (PlusA5Flag == false)
                {
                    if (MinusA5Flag == false)
                    {
                        MinusA5Flag = true;
                        int writeResult = WriteBit("DB1.DBX0.3", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                    else
                    {
                        MinusA5Flag = false;
                        int writeResult = WriteBit("DB1.DBX0.3", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }
                    }
                }
            });
        }

        public async Task PlusA5()
        {
            await Task.Run(() =>
            {
                if (MinusA5Flag == false)
                {
                    if (PlusA5Flag == false)
                    {
                        PlusA5Flag = true;
                        int writeResult = WriteBit("DB1.DBX0.4", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }
```

```
                    }
                    else
                    {
                        PlusA5Flag = false;
                        int writeResult = WriteBit("DB1.DBX0.4", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }

        public async Task MinusA6()
        {
            await Task.Run(() =>
            {
                if (PlusA6Flag == false)
                {
                    if (MinusA6Flag == false)
                    {
                        MinusA6Flag = true;
                        int writeResult = WriteBit("DB1.DBX0.5", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }
                    }
                    else
                    {
                        MinusA6Flag = false;
                        int writeResult = WriteBit("DB1.DBX0.5", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }


        public async Task PlusA6()
        {
            await Task.Run(() =>
            {
                if (MinusA6Flag == false)
                {
                    if (PlusA6Flag == false)
                    {
                        PlusA6Flag = true;
                        int writeResult = WriteBit("DB1.DBX0.6", true);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }
```

```
                    }
                    else
                    {
                        PlusA6Flag = false;
                        int writeResult = WriteBit("DB1.DBX0.6", false);
                        if (writeResult != 0)
                        {
                            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t
Write error: " + _client.ErrorText(writeResult));
                        }

                    }
                }
            });
        }


        private void OnTimerElapsed(object sender, ElapsedEventArgs e)
        {
            try
            {
                _timer.Stop();
                ScanTime = DateTime.Now - _lastScanTime;
                RefreshValues();
                OnValuesRefreshed();
            }
            finally
            {
                _timer.Start();
            }
            _lastScanTime = DateTime.Now;
        }

        private void RefreshValues()
        {
            lock (_locker)
            {

                var buffer = new byte[24];
                int result = _client.DBRead(2, 0, buffer.Length, buffer);
                if (result == 0)
                {
                    JogA1 = S7.GetRealAt(buffer, 0);
                    JogA2 = S7.GetRealAt(buffer, 4);
                    JogA3 = S7.GetRealAt(buffer, 8);
                    JogA4 = S7.GetRealAt(buffer, 12);
                    JogA5 = S7.GetRealAt(buffer, 16);
                    JogA6 = S7.GetRealAt(buffer, 20);
                }
                else
                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Read
error: " + _client.ErrorText(result));
                }
            }
        }

        private void Status()
        {
            lock (_locker)
            {
                var buffer2 = new byte[2];
```

```
                int result2 = _client.DBRead(5, 0, buffer2.Length, buffer2);
                if (result2 == 0)
                {
                    MinusA1Status = S7.GetBitAt(buffer2, 0, 0);

                }
                else
                {
                    Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Read
error: " + _client.ErrorText(result2));
                }
            }
        }
                /// <summary>
                /// upisuje bit na željenu adresu. npr.: DB1.DBX10.2 upisuje bit u db
1, word 10,treći bit
             /// </summary>
                /// <param name="address">Es.: DB1.DBX10.2 upisuje bit u db 1, word
10,treći bit</param>
                /// <param name="value">true or false</param>
                /// <returns></returns>
        private int WriteBit(string address, bool value)
        {
            var strings = address.Split('.');
            int db = Convert.ToInt32(strings[0].Replace("DB", ""));
            int pos = Convert.ToInt32(strings[1].Replace("DBX", ""));
            int bit = Convert.ToInt32(strings[2]);
            return WriteBit(db, pos, bit, value);
        }

        private int WriteBit(int db, int pos, int bit, bool value)
        {
            lock (_locker)
            {
                var buffer = new byte[1];
                S7.SetBitAt(ref buffer, 0, bit, value);
                return _client.WriteArea(S7Consts.S7AreaDB, db, pos + bit,
buffer.Length, S7Consts.S7WLBit, buffer);
            }
        }

        private void OnValuesRefreshed()
        {
            ValuesRefreshed?.Invoke(this, new EventArgs());
        }
    }
}
```

2. Kod sloja Model Prikaza

```
using GalaSoft.MvvmLight;
```

```
using GalaSoft.MvvmLight.Command;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;
using UIforFanuc.PlcService;

namespace UI_for_Fanuc.ViewModels
{
    class MainWindowViewModel : ViewModelBase
    {
        public string IpAddress
        {
            get { return _ipAddress; }
            set { Set(ref _ipAddress, value); }
        }
        private string _ipAddress;

        public bool MinusA1Status
        {
            get { return _MinusA1Status; }
            set { Set(ref _MinusA1Status, value); }
        }
        private bool _MinusA1Status;

        public double JogA1
        {
            get { return _JogA1; }
            set { Set(ref _JogA1, value); }
        }
        private double _JogA1;

        public double JogA2
        {
            get { return _JogA2; }
            set { Set(ref _JogA2, value); }
        }
        private double _JogA2;

        public double JogA3
        {
            get { return _JogA3; }
            set { Set(ref _JogA3, value); }
        }
        private double _JogA3;

        public double JogA4
        {
            get { return _JogA4; }
            set { Set(ref _JogA4, value); }
        }
        private double _JogA4;

        public double JogA5
        {
            get { return _JogA5; }
            set { Set(ref _JogA5, value); }
        }
        private double _JogA5;
```

```
public double JogA6
{
    get { return _JogA6; }
    set { Set(ref _JogA6, value); }
}
private double _JogA6;

public ConnectionStates ConnectionState
{
    get { return _connectionState; }
    set { Set(ref _connectionState, value); }
}
private ConnectionStates _connectionState;

public TimeSpan ScanTime
{
    get { return _scanTime; }
    set { Set(ref _scanTime, value); }
}
private TimeSpan _scanTime;

public ICommand ConnectCommand { get; private set; }
public ICommand DisconnectCommand { get; private set; }
public ICommand StartCommand { get; private set; }
public ICommand StopCommand { get; private set; }
public ICommand MinusA1Command { get; private set; }
public ICommand PlusA1Command { get; private set; }
public ICommand MinusA2Command { get; private set; }
public ICommand PlusA2Command { get; private set; }
public ICommand MinusA3Command { get; private set; }
public ICommand PlusA3Command { get; private set; }
public ICommand MinusA4Command { get; private set; }
public ICommand PlusA4Command { get; private set; }
public ICommand MinusA5Command { get; private set; }
public ICommand PlusA5Command { get; private set; }
public ICommand MinusA6Command { get; private set; }
public ICommand PlusA6Command { get; private set; }

S7PlcService _plcService;

public MainWindowViewModel()
{
    _plcService = new S7PlcService();
    ConnectCommand = new RelayCommand(Connect);
    DisconnectCommand = new RelayCommand(Disconnect);
    StartCommand = new RelayCommand(async () => { await Start(); });
    StopCommand = new RelayCommand(async () => { await Stop(); });
    MinusA1Command = new RelayCommand(async () => { await MinusA1(); });
    PlusA1Command = new RelayCommand(async () => { await PlusA1(); });
    MinusA2Command = new RelayCommand(async () => { await MinusA2(); });
    PlusA2Command = new RelayCommand(async () => { await PlusA2(); });
    MinusA3Command = new RelayCommand(async () => { await MinusA3(); });
    PlusA3Command = new RelayCommand(async () => { await PlusA3(); });
    MinusA4Command = new RelayCommand(async () => { await MinusA4(); });
    PlusA4Command = new RelayCommand(async () => { await PlusA4(); });
    MinusA5Command = new RelayCommand(async () => { await MinusA5(); });
    PlusA5Command = new RelayCommand(async () => { await PlusA5(); });
    MinusA6Command = new RelayCommand(async () => { await MinusA6(); });
    PlusA6Command = new RelayCommand(async () => { await PlusA6(); });
    MinusA1Status = _plcService.MinusA1Status;
    IpAddress = "192.168.0.2";
```

```
        // uvijek refresha vrijednosti na nulu kada se pokrene sučelje
        OnPlcServiceValuesRefreshed(null, null);
        _plcService.ValuesRefreshed += OnPlcServiceValuesRefreshed;
    }
    private void OnPlcServiceValuesRefreshed(object sender, EventArgs e)
    {
        ConnectionState = _plcService.ConnectionState;
        JogA1 = _plcService.JogA1;
        JogA2 = _plcService.JogA2;
        JogA3 = _plcService.JogA3;
        JogA4 = _plcService.JogA4;
        JogA5 = _plcService.JogA5;
        JogA6 = _plcService.JogA6;
        ScanTime = _plcService.ScanTime;

    }
    private void Status(object sender, EventArgs e)
    {
        MinusA1Status = _plcService.MinusA1Status;
    }
    private void Connect()
    {
        _plcService.Connect(IpAddress, 0, 1);
    }
    private void Disconnect()
    {
        _plcService.Disconnect();
    }
    private async Task Start()
    {
        await _plcService.WriteStart();
    }

    private async Task Stop()
    {
        await _plcService.WriteStop();
    }
    private async Task MinusA1()
    {
        await _plcService.MinusA1();
    }
    private async Task PlusA1()
    {
        await _plcService.PlusA1();
    }
    private async Task MinusA2()
    {
        await _plcService.MinusA2();
    }
    private async Task PlusA2()
    {
        await _plcService.PlusA2();
    }
    private async Task MinusA3()
    {
        await _plcService.MinusA3();
    }
    private async Task PlusA3()
    {
        await _plcService.PlusA3();
    }
    private async Task MinusA4()
```

```
        {
            await _plcService.MinusA4();
        }
        private async Task PlusA4()
        {
            await _plcService.PlusA4();
        }
        private async Task MinusA5()
        {
            await _plcService.MinusA5();
        }
        private async Task PlusA5()
        {
            await _plcService.PlusA5();
        }
        private async Task MinusA6()
        {
            await _plcService.MinusA6();
        }
        private async Task PlusA6()
        {
            await _plcService.PlusA6();
        }


    }

}
```

II. Primjer koda u upravljačkoj jedinici Fanuc-a

KARLO_GLAVNIPROG

```
 1:  UFRAME_NUM=0 ;
 2:  LBL[1] ;
 3:  CALL KARLO_SUCELJE    ;
 4:  CALL KARLO_UPISIVANJE    ;
5:J PR[10] 35% CNT50    ;
 6:  JMP LBL[1] ;
```

KARLO_SUCELJE

```
 1:  GO[1]=($SCR_GRP[1].$MCH_ANG[1]*10) ;
 2:  GO[2]=($SCR_GRP[1].$MCH_ANG[2]*10) ;
 3:  GO[3]=($SCR_GRP[1].$MCH_ANG[3]*10) ;
 4:  GO[4]=($SCR_GRP[1].$MCH_ANG[4]*10) ;
 5:  GO[5]=($SCR_GRP[1].$MCH_ANG[5]*10) ;
 6:  GO[6]=($SCR_GRP[1].$MCH_ANG[6]*10) ;
```

KARLO_UPISIVANJE

```
 1:  IF DI[1153]=OFF,JMP LBL[1] ;
 2:  IF DI[1153]=ON,JMP LBL[2] ;
 3:  LBL[1] ;
 4:  PR[10,1]=GI[1:ZGLOB_A1]/10    ;
 5:  JMP LBL[3] ;
 6:  LBL[2] ;
 7:  PR[10,1]=GI[1:ZGLOB_A1]/(-10)    ;
 8:  LBL[3] ;
 9:  IF DI[1154]=OFF,JMP LBL[4] ;
10:  IF DI[1154]=ON,JMP LBL[5] ;
11:  LBL[4] ;
```