Upravljanje dvo-osnim manipulatorom

Medovka, Krešimir

FAKULTET STROJARSTVA I BRODOGRADNJE
SVEUČILIŠTE U ZAGREBU

ZAVRŠNI RAD

Krešimir Medovka

Zagreb, 2019.

FAKULTET STROJARSTVA I BRODOGRADNJE
SVEUČILIŠTE U ZAGREBU

ZAVRŠNI RAD

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

Student:
Krešimir Medovka

Zagreb, 2019.

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 pružio priliku raditi na ovako zanimljivom završnom radu te na njegovoj strpljivosti.

Zahvaljujem svojoj obitelji na svim savjetima i podršci tijekom studija te svim prijateljima.

Na kraju zahvaljujem svim članovima obitelji i prijateljima koji više nisu tu, bilo fizički ili psihički, a koji su me inspirirali u raznim područjima života.

Krešimir Medovka

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: **KREŠIMIR MEDOVKA** Mat. br.: 0035203080

Naslov rada na hrvatskom jeziku: **UPRAVLJANJE DVO-OSNIM MANIPULATOROM**

Naslov rada na engleskom jeziku: **CONTROL OF TWO-AXIS MANIPULATOR**

Opis zadatka:

U radu je potrebno oblikovati upravljački program i sučelje za dvo-osni manipulator. Manipulator služi za izuzimanje i odlaganje dijelova termoregulatora između dva transportna sustava. Potrebno je voditi računa o visinskoj razlici i prostornim ograničenima manipulatora. Pri razvoju rješenja osigurati funkcionalnu i upravljačku integraciju s postojećim automatskim montažnim sustavom u laboratoriju. Koristiti postojeću opremu u Laboratoriju za projektiranje izradbenih i montažnih sustava.

Zadatak zadan:
6. svibnja 2019.

Rok predaje rada:
2. rok (izvanredni): 28. lipnja 2019.
3. rok: 20. rujna 2019.

Predviđeni datumi obrane:
2. rok (izvanredni): 2.7. 2019.
3. rok: 23.9. - 27.9. 2019.

Zadatak zadao:

Prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

SADRŽAJ I
POPIS SLIKA III
POPIS TABLICA IV
SAŽETAK V
SUMMARY VI
1. UVOD 1
2. DVO-OSNI MANIPULATOR 2
3. KORIŠTENI SKLOPOVI I ELEMENTI 3
3.1. Programabilni logički sklop (PLC) 3
3.1.1 Siemens S7-1200 3
3.2. Modul za udaljene ulaze i izlaze (eng. *Remote I/O module*) 5
3.2.1. Siemens ET 200SP 5
3.3. Ventilski blok (eng. *Valve terminal*) 6
3.3.1. SMC VQC2201N-51 6
4. KORIŠTENI PROGRAMI 7
4.1. C# programski jezik 7
4.1.1. *Sharp7* 7
4.2. TIA Portal i *Step7* 7
5. KORIŠTENE KOMUNIKACIJSKE MREŽE I PROTOKOLI 9
5.1.Industrijski Ethernet 9
5.2.TCP/IP Protokol 10
5.3.PROFINET 10
6. KORIŠTENE METODE 12
6.1. MVVM 12
6.1.1. Model 12
6.1.2. Prikaz (eng. *View*) 13
6.1.3. ModelPrikaza (eng. *ViewModel*) 13
6.2. Blokirajući signal i Dijagram put-korak 13
6.3. Elektropneumatska taktna metoda i spoj samodržanja 14
7. PROJEKTIRANJE I IZRADA GRAFIČKOG KORISNIČKOG SUČELJA 17
7.1. Pogled 18
7.2. ModelPogleda 18

7.3. Model 18
8. PROJEKTIRANJE I IZRADA UPRAVLJAČKOG PROGRAMA 19
8.1. Main 19
8.2. Dvo-osni manipulator 19
8.3. Statusi 20
8.4. Početni položaj 20
8.5. Manualni mod rada 21
8.6. Automatski mod rada 23
8.6.1. Dijagram tijeka 23
8.6.2. Dijagram put-korak i blokirajući signali 23
8.6.3. Automatski mod rada pomoću elektropneumatske taktne metode simuliran u relejnoj tehnici 24
8.6.4. Automatski mod rada pomoću elektropneumatske taktne metode u Step7 25
8.7. Podatkovni blokovi 26
9. ZAKLJUČAK 27
10. LITERATURA 28
11. PRILOZI 29

POPIS SLIKA

Slika 1. Dvo-osni manipulator u prirodnom okruženju.....2

Slika 2. Siemens S7-1200 s utorima za spajanje.....4

Slika 3. ET 200SP.....5

Slika 4. VVQC2201N-51 na bazi.....6

Slika 5. MVVM koncept.....9

Slika 6. Primjer pneumatske sheme s blokirajućim signalom.....11

Slika 7. Primjer dijagrama put-korak.....11

Slika 8. Elektropneumatska taktna metoda.....12

Slika 9. Spoj samodržanja.....13

Slika 10. Izgled sučelja.....14

Slika 11. Main.....19

Slika 12. Podprogram Dvo-osni manipulator.....20

Slika 14. Deaktiviranje memorijskog bita odnosno podprograma.....21

Slika 15. Spoj samodržanja koji omogućuje slanje podataka.....21

Slika 16. Primanje podataka iz korisničkog sučelja i sprječavanje štetnih naredbi.....22

Slika 17. Dijagram put-korak.....24

Slika 18. Automatski mod rada simuliran pomoću releja u FluidSim Pneumatics.....25

Slika 19. Primjer grane prevedene iz relejne u ljestvičastu logiku.....26

POPIS TABLICA

Tablica 1. Tehničke specifikacije Siemensa S7-1200 klase CPU 1214 DC/DC/DC.....4

SAŽETAK

Zadatak završnog rada je oblikovati upravljački program i korisničko sučelje (eng. *User Interface*, UI) za upravljanje dvo-osnim manipulatorom koji služi za izuzimanje i odlaganje dijelova termoregulatora između dva transportna sustava, putem programabilnog logičkog sklopa (eng. *Programmabale Logic Controller*, PLC). Tijekom oblikovanja potrebno je voditi računa o prostornim ograničenjima manipulatora te o visinskoj razlici dvaju transportnih sustava.

Ključne riječi: manipulator, sučelje, plc, upravljanje

SUMMARY

The goal of this task is to develop control algorithm and a user interface for controling two-axis manipulator which is used for picking and placing components of thermoregulator between two transport systems, using Programmable Logic Controller (PLC). During development it is required to take into account the space limitations of the two-axis manipulator and the height difference of the two transport systems.

Keywords: manipulator, interface, plc, control

1. UVOD

Tema ovog završnog rada je izraditi upravljački program i korisničko sučelje za upravljanje dvo-osnim „pick&place“ manipulatorom.

Upravljački program izvodi se u programabilnom logičkom sklopu (PLC-u) S7-1200 tvrtke Siemens. Pisan je u ljestvičastoj (eng., *Ladder,* LAD) logici u programu *Step*7 integriranom u arhitekturu TIA Portal. Problem blokirajućih signala riješen je uz pomoć elektropneumatske taktne metode.

Sučelje je razvijeno pomoću C# programskog jezika u Microsoft Visual Studio programskom okruženju. Njime se upravlja putem računala, a signali se šalju u PLC koji zatim izvršava zadane naredbe u skladu sa već postojećim upravljačkim programom. Također se pomoću sučelja može vršiti nadzor ispravnosti izvođenja upravljačkog programa s nekog udaljenog mjesta.

Do povećanog razvoja ovakvih sučelja dolazi zbog potrebe da se na lak i jednostavan način može upravljati tehničkim procesima. 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.

2. DVO-OSNI MANIPULATOR

Dvo-osni „uzmi i stavi“ (eng. „*pick&place*“) manipulator u Laboratoriju za projektiranje izradbenih i montažnih sustava jedinstven je i napravljen u sklopu jednog od brojnih završnih i diplomskih radova.

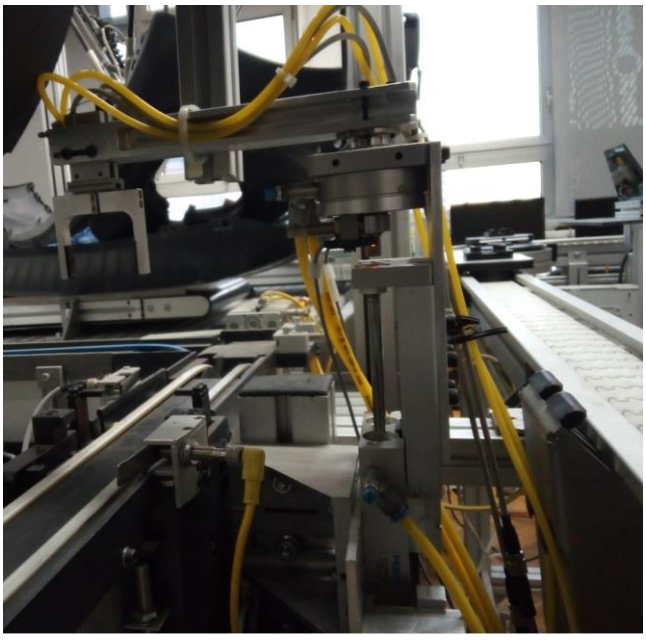
Služi za izuzimanje i dodavanje elemenata termoregulatora između dva transportna sustava.

Manipulator ima dva stupnja slobode gibanja: translaciju po z-osi i ograničene rotacije oko z-osi koju omogućuju pneumatski aktuatori. Ograničenje rotacije u smjeru kazaljke na satu je potpuno, a u smjeru suprotnom od kazaljke na satu ograničenje kuta zakreta je 180°. Također ima jednostavnu pneumatski upravljanu hvataljku sa paralelnim hvatom. Zbog visinske razlike između dva transportna sustava u laboratoriju opremljen je i sa visinskim graničnikom, koji nije ništa drugo već jednostavni dvoradni cilindar sa zavarenim komadom čelika na kraju klipnjače. Visinski graničnik omogućava da translacijsko gibanje iz jednog krajnjeg položaja u drugi uvijek bude isto tj. da visinska razlika između hvataljke i nosača predmeta rada bude ista, odnosno da se manipulator ili transportni sustav ne bi oštetio.

Zbog prostornih ograničenja manipulatora ugrađeni su granični prekidači u obliku reed senzora, koji služe ne samo kako bi ograničili kretanje manipulatora, već i u upravljačke svrhe.

Manipulatorom se može upravljati ili preko računala pomoću korisničkog sučelja koje šalje naredbe PLC-u, ili direktno preko PLC-a, ovisno o modu rada. PLC šalje odgovarajuće izlazne signale blok ventilu, ovisno o modu rada i zadanim ulaznim varijablama, odnosno, ovisno o upravljačkom programu. Blok ventil šalje pneumatski signal određenom segmentu manipulatora.

Na slici 1. prikazan je manipulator u svom prirodnom okruženju.



Slika 1. Dvo-osni manipulator u prirodnom okruženju

3. KORIŠTENI SKLOPOVI I ELEMENTI

3.1. 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.

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

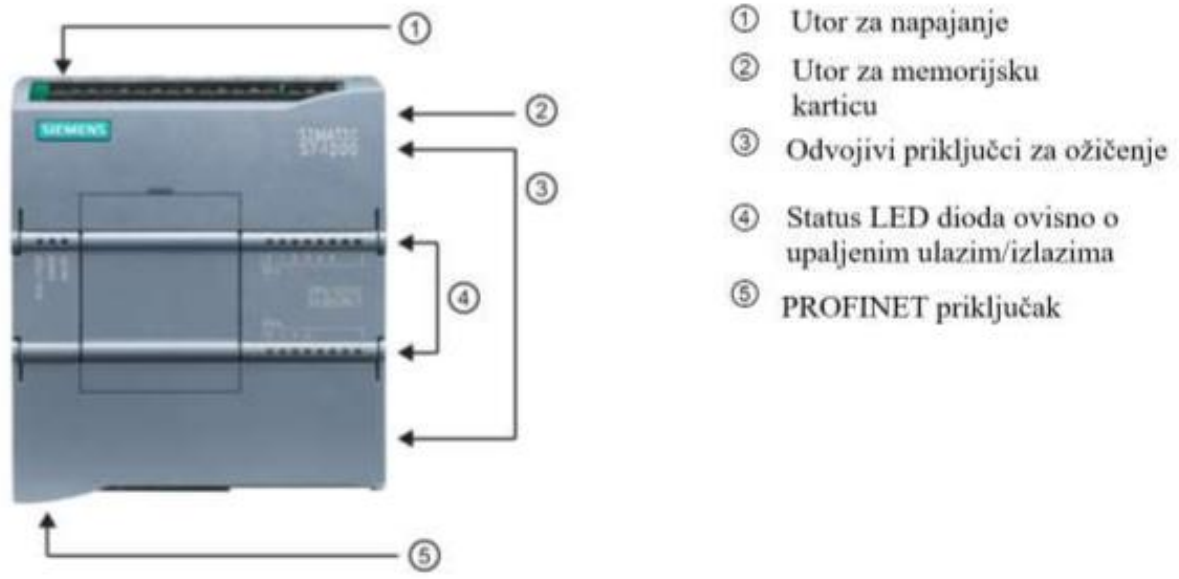
- 1. Čitanje ulaza
- 2. Čitanjeprograma
- 3. Obrada zahtjeva za komunikaciju
- 4.Izvršavanje CPU dijagnostike
- 5.Pisanje izlaza

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. 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 izvršenja 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.

3.1.1 Siemens S7-1200

Prilikom izrade završnog rada korišten je Siemensov PLCS7-1200. 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 računalom i blok ventilom koji šalje pneumatske signale za pokretanje određenih segmenata manipulatora, 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.



Slika 2. 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 priključen je dodatni modul za komunikaciju. Kako ovaj PLC ima samo jedan PROFINET(Ethernet) priključak, a da bi se povezalo istovremeno na računalo, modul za udaljene ulaze i izlaze te ostale već postojeće sustave u laboratoriju,spojen je dodatan komunikacijski modul. Korišten je CSM 1277 komunikacijski modul koji omogućava PLC-u da se spoji s još tri dodatna komunikacijska uređaja.

3.2. Modul za udaljene ulaze i izlaze (eng. *Remote I/O module*)

Moduli za udaljene ulaze i izlaze se koriste kako se u nekom postrojenju ne bi moralo spajati sve ulaze i izlaze direktno na PLC koji se nalazi u kontrolnoj sobi, što je skupo i neučinkovito. Pomoću ovih modula broj veza između kontrolne sobe i postrojenja svodi se na jednu – Ethernet kabel koji povezuje PLC i modul. Glavni nedostatak je što u slučaju bilo kakvog oštećenja tog kabela niti jedan signal neće stići do PLC-a. To se može riješiti ožičavanjem glavnih signala na PLC ili se može spojiti redundantni Ethernet kabel koji će, u slučaju oštćenja prvog, i dalje slati signale PLC-u.

3.2.1. Siemens ET 200SP

U završnom radu korišten je modul ET 200SP tvrtke Siemens. Ovaj modul danas je najtraženiji i najkorišteniji jer ga je lako koristiti, kompaktan je, učinkovit, moduli se lako mijenjaju tijekom rada i može komunicirati preko raznih komunikacijskih protokola, najčešće preko PROFINET protokola.

Konfiguriran je tako da sadrži dva modula za digitalne ulaze (16 ulaza po modulu) i dva modula za digitalne izlaze (16 izlaza po modulu). Na digitalne ulazne module spojeni su senzori graničnih položaja manipulatora.

Slika 3. ET 200SP

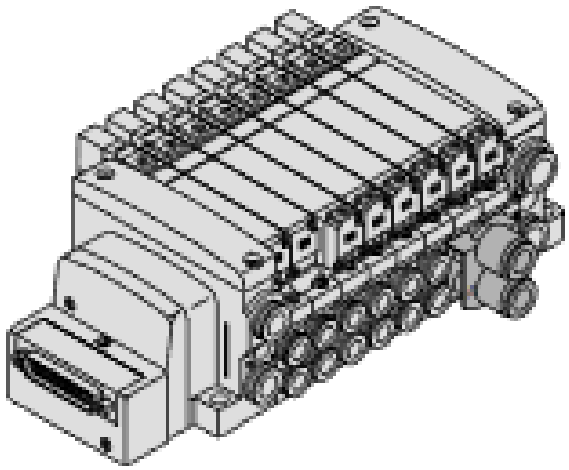
3.3. Ventilski blok (eng. *Valve terminal*)

Ventilski blok je skup modularnih razvodnika koji su montirani na neku bazu, a koriste se kada je potrebno upravljati većim brojem aktuatora.

3.3.1. SMC VQC2201N-51

U završnom radu korišten je ventilski blok VQC2201N-51 tvrtke SMC.

Ovaj ventilski blok koristi elektropneumatske bistabilne 5/2 razvodnike. Razvodnici ove vrstemaju dva stabilna stanja i 5 ulaza od kojih dva služe za pogonjenje aktuatora, a tri za dobavu komprimiranog zraka i odzračivanje, a aktivirani su pomoću električnih signala.



Slika 4. VVQC2201N-51 na bazi

4. KORIŠTENI PROGRAMI

4.1. C# programski jezik

Za programiranje i izradu sučelja korišteno je programsko okruženje Microsoft Visual Studio 2019 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 korišten je za razvoj ovog sučelja. Iako postoje mnogi drugi jezici u kojima se može razvijati sučelje odabranje 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*) koji će biti detaljno objašnjen u slijedećem poglavlju.

4.1.1. *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 spojitina 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.

4.2. TIA Portal i *Step7*

Upravljački sklop Siemens S7-1200 modela CPU 1214 DC/DC/DC konfiguriran je pomoću STEP7 Basic V15.0 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 prijenoson 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.

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.

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 naredbi 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.

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, kaošto 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.

5. KORIŠTENE 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 nebi funkcioniraju bez da uređaji međusobno ne komuniciraju.

Računalo komunicira sa PLC-om preko Industrijskog Etherneta. PLC komunicira s ET 200SP i blok ventilom preko PROFINET-a.

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 nebi 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 funkcijamau procesu izmjenjivanja podataka. TCP definira kako se stvaraju kanali komunikacije preko mreže te poruku pretvara u manje pakete prije nego krene njezinprijenos. 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 protokolafunkcionira? Na temelju modela klijent/poslužitelj dolazi do komunikacije u kojoj korisnik ili stroj pruža uslugu drugom računalu ili stroju odnosno poslužitelju utojmreži. 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.

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 onapovezuje 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. 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. 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.

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

6. KORIŠTENE METODE

6.1. MVVM

MVVM je zapravo skraćenica riječi Model-Prikaz-ModelPrikaza (eng. *Model-View-ViewModel*) i predstavlja vrstu 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 5. MVVM koncept

6.1.1. 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 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 pa prikazai model ne očitava stvarne vrijednosti.

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

6.1.2. Prikaz (eng. *View*)

Prikaz je ono što se zna kada se govori o sučelju i jedini 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 ModelaPrikaza šalju iz Prikaza u Model. 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 ModelPrikaza. Iako se ovi događaji i ponašanja mogu grupirati na svojstva, pozive metoda i naredbe, Prikaz je na kraju sam odgovoran za izvršavanje svojih metoda.

6.1.3. ModelPrikaza (eng. *ViewModel*)

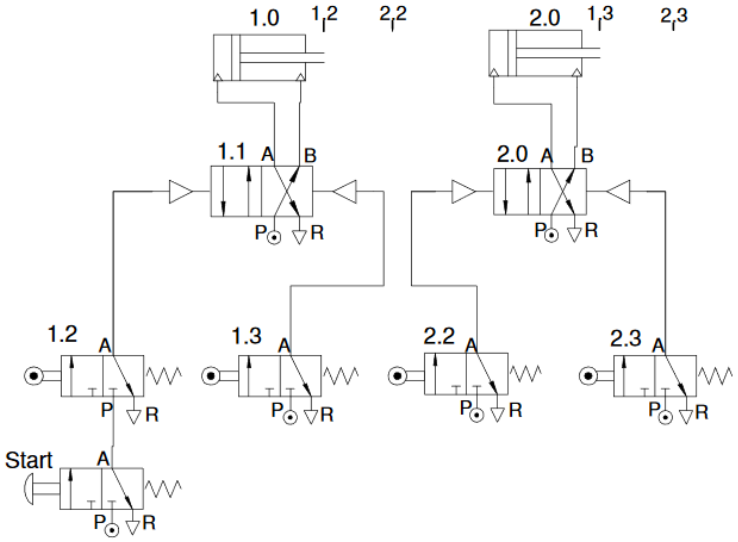
Model Prikaza je ključni dio ova tri sloja jer se u njemu nalazi prezentacijsko odvajanje ili koncept koji odvaja Model od Prikaza. Umjesto da je Model svjestan toga da je u Prikazu formatirano trenutno stanje pojedinog dijela manipulatora, on samo posjeduje tu informaciju i ne zna što se događa u Prikazu. ModelPrikaza 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 ModelPrikaza uzima trentno stanje pojedinog dijela manipulatora koje je očitao iz PLC-a preko sloja Model i šalje taj podatak u Prikaz u kojem se uključuje zeleno svijetlo za odgovarajuće stanje. Prikaz i ModelPrikaza komuniciraju putem podataka koji ih povezuju, poziva na metode, svojstava, događaja i poruka. ModelPrikaza ne izlaže samo modele, već druga svojstva i naredbe (poput informacija o stanju PLC-a npr. trenutnoje stanje „Online“ tj. uspostavljena je komunikacija). Prikaz obrađuje vlastite događaje koje je korisnik napravio pritiskom na tipku na sučelju, a zatim ih prebacuje na ModelPrikaza putem naredbi. Na primjer pritiskom na tipku „Automatski“ na sučelju ModelPrikaza preko naredbe „iCommand“ prebacuje vrijednost iz Prikaza u Model.

6.2. Blokirajući signal i Dijagram put-korak

Kod rada s više cilindara pojavljuje se tzv. blokirajući signal. Sam naziv ukazuje na njegov karakter. On blokira odvijanje daljnjeg programa. Iako se koji put namjerno rabi da bi nešto blokirao, u najvećem broju slučajeva taj je signal nepoželjan. Razrađeno je mnogo metoda, a konstruirani su i neki elementi kojima je cilj eliminacija blokirajućeg signala.

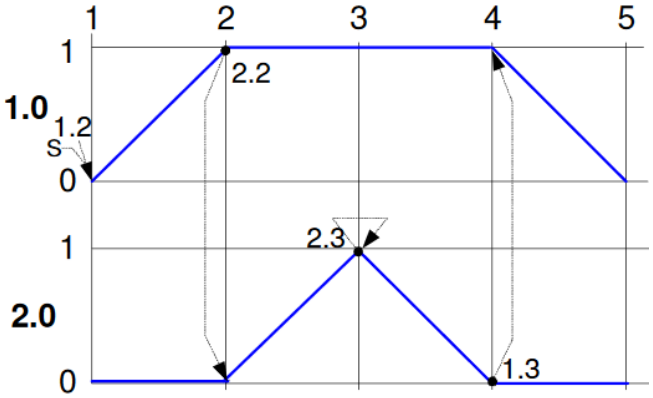
Što je blokirajući signal? Osim gornje općenite oznake, teško je dati univerzalnu oznaku bez nešto podrobnijeg opisa. Na osnovi zamišljenog rada segmenta od dva cilindra, npr. u programskom upravljanju ovisnom o putu, može se objasniti njegov nastanak te utjecaj na blokiranje odvijanja daljnjeg programa. Neka se uzme da po redoslijedu programa klipnjača cilindra 1.0, nakon što je izišla, aktivira kretanje klipnjače cilindra 2.0. To znači da je signal došao na stranu 12(Z) glavnog razvodnika 2.1. Klipnjača cilindra 1.0 npr. po programu treba ostati u izvučenom položaju tako da pritisnuti razvodnik 2.2 trajno daje

signal na priključak 12(Z) razvodnika 2.1. Ako se klipnjača cilindra 2.0 treba po programu uvući, signal koji će doći na stranu 14(Y) neće ga prebaciti u novi položaj jer ga u tome sprječava signal na priključku 12(Z). Taj se signal zove blokirajući signal.



Slika 6. Primjer pneumatske sheme s blokirajućim signalom

Dijagram put-korak služi za lakšu detekciju blokirajućih signala.



Slika 7. Primjer dijagrama put-korak

6.3. Elektropneumatska taktna metoda i spoj samodržanja

U pneumatskom upravljanju postoje tri inženjerska načela eliminiranja blokirajućeg signala i to su: Funkcijsko načelo, Kaskadno načelo i Koračno načelo.

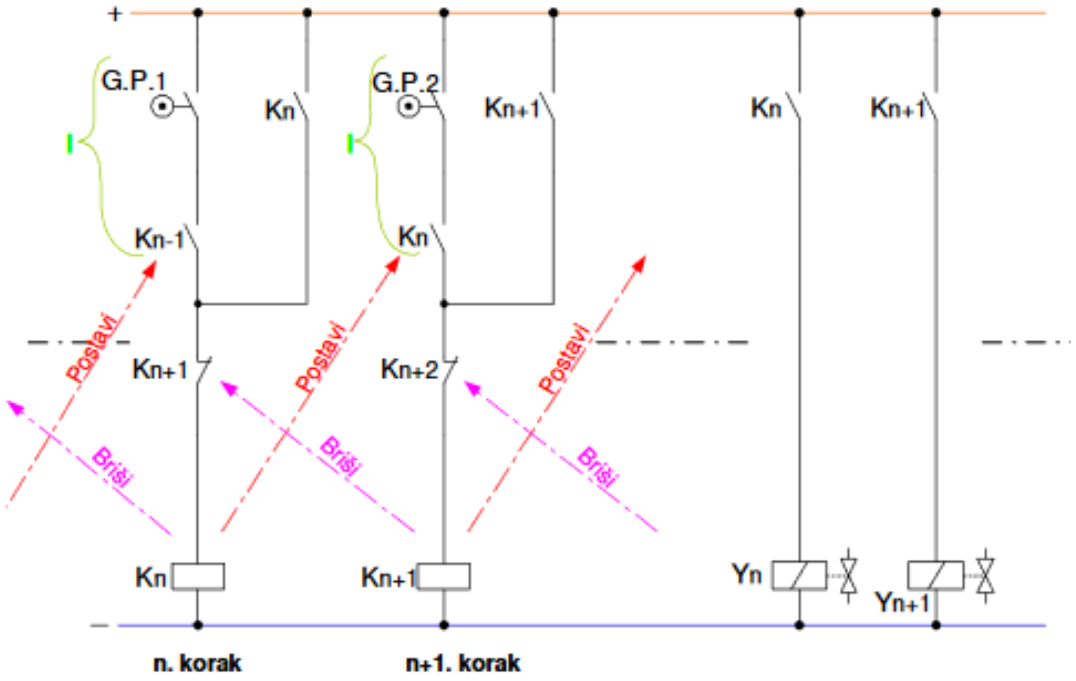
Funkcijsko načelo rabe metode koje se za otklanjanje blokirajućeg signala oslanjanju na specifičnu funkciju rada upotrijebljenoga pneumatskog elementa. To su pneumatski elementi koji, iako trajno aktivirani, na izlazu u sustav daju kratkotrajni signal zbog specifičnog funkcioniranja mehanizma aktiviranja. Najtipičnija metoda za ovu grupu je VDMA (kratica od Verein Deutscher Maschinenbau Anstalten - Udruga njemačkih ustanova za strojogradnju) koja koristi razvodnik 3/2 razvodnik sa zglobnim ticalom.

Kaskadno načelo označuje segmentno (kaskađno) uključivanje dijela graničnih prekidača uporabom dodatnog uvjeta (I funkcija). Taj je uvjet izveden jednim od načina ostvarivanja I funkcije, a posebno je proizveden razvodnicima nazvanim kaskadnim razvodnicima. Oni se uključuju u skladu sa slijedom odvijanja programa. Metode koje se rabe nazvane su po tvrtkama koje su ih plasirale. Uglavnom se razlikuju samo u vrsti kaskadnog

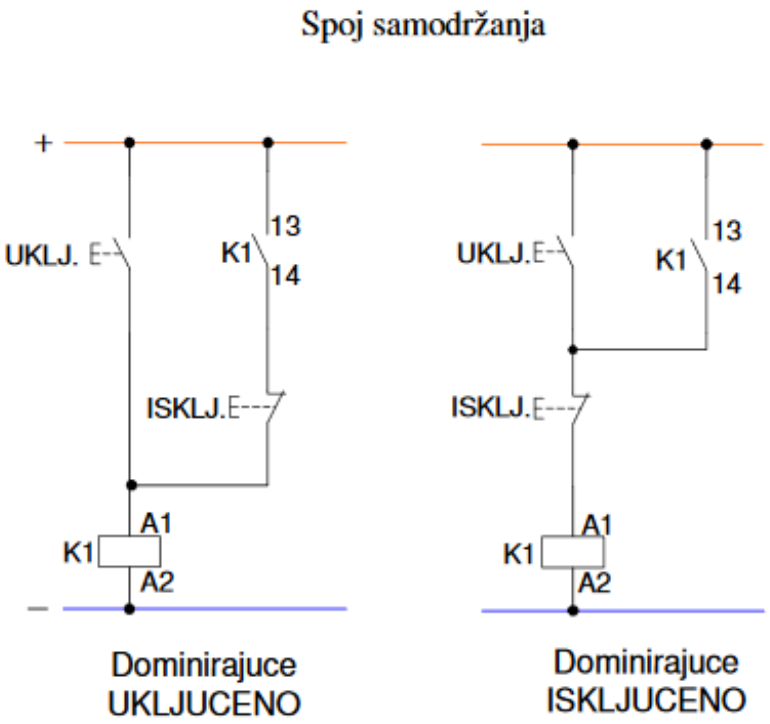
razvodnika koji rabe. Najpoznatija metoda je kaskadna metoda tvrtke Martonair (iako tvrtka više ne postoji), koja je značajnije od drugih rasprostranjena u praksi te se često sve kaskadne metode poistovjećuju s njom.

Koračno načelo temelji se na odvijanju programa ostvarenog određenim načinom povezivanja pneumatskih elemenata, po kojem je u slijed uključen uvijek samo izlaz za jedan korak. Svi ostali izlazi su isključeni. To je jedina sveobuhvatna metoda koja svojim pravilima rada rješava sve slučajeve odvijanja programa. Temeljna metoda ovog načela rada je metoda "korak po korak". Kako ona sadrži znatno više elemenata od prethodnih, znatno je skuplja. Da bi se te riješilo, razvijena je "takt" metoda integracijom elemenata u blokove. Ona se sve više primjenjuje u praksi, jer je sveobuhvatna u načinu rada, a izvedbom blok elemenata postaje i jeftinija.

U elektropneumatskom upravljanju također postoje sva tri načela, a najčešće se koristi elektropneumatska taktna metoda zbog svoje jednostavnosti.
Aktualni korak uspostavlja uvjet (set) za slijedeći, a ukida prethodni (reset). Koristi se spoj samodržanja. Dodatni signal dolazi od krajnjeg (graničnog) prekidača, ili nekog drugog elementa, preko serijskog spoja (I logika).



Slika 8. Elektropneumatska taktna metoda



Slika 9. Spoj samodržanja

Spojevi samodržanja potrebni su u elektropneumatskom upravljanju, kada električni signali moraju biti memorirani. Ako se spojem samodržanja ostvaruje memoriranje signala u električkom dijelu sheme, mogu se tada upotrijebiti pneumatski razvodnici čiji se povrat ostvaruje oprugom (monostabili). Kod češće korištenih upravljanja (taktni lanac) može se funkcija memoriranja , ovisno o elementima, nalaziti u pneumatskom, električkom ili u oba dijela. U tehnici upravljanja govori se o dva spoja samodržanja, dominirajući uključeni (UKLJ.) ili dominirajući isključeni (ISKLJ.)
Tipkalom UKLJ. pobuđuje se relej K1 koji preklapa kontakte. Da bi po otpuštanju tipkala UKLJ. relej K1 ostao privučen, spaja se paralelno tipkalu UKLJ. radni kontakt releja K1.
Ovim paralelnim spojem postiže se držanje releja K1 i u slučaju otpuštanja tipkala UKLJ., jer ga drži paralelni spoj preko njegovog kontakta.
Da bi se spoj mogao srušiti mora se ugraditi tiipkalo ISKLJ.
Kod spoja dominirajući UKLJ. ovo tipkalo ISKLJ. nalazi se u serijskom spoju s kotaktom releja K1 na paralelnoj grani.
Kod spoja samodržanja dominirajući ISKLJ. tipkalo se nalazi u serijskom spoju s tipkalom UKLJ. i svitkom releja K1.
Za dominantnost signala tipkala UKLJ. i ISKLJ. odlučjući je položaj tipkala ISKLJ.
Ako se tipkalo ISKLJ. nalazi u serijskom spoju s radnim kontaktom, onda je uvijek dominirajući signal UKLJ.
Ako se tipkalo ISKLJ. nalazi u serijskom spoju s tipkalom UKLJ., tada je dominirajući signal ISKLJ.

7. PROJEKTIRANJE I IZRADA GRAFIČKOG KORISNIČKOG SUČELJA

Sučelje je osmišljeno na takav način da bude jednostavno za korištenje i učinkovito, a opet da bude estetski privlačno jer u današnje vrijeme bitna je i estetika, a ne samo funkcija.

Programirano je u programskom jeziku C# i korištena je MVVM metoda koja je već ranije opisana.

Sučelje se sastoji od 4 sektora:

- sektor za komunikaciju sa PLC-om
- sektor za odabir moda rada i prikaz stanja na transportnim sustavima
- sektor za promjenu stanja segmenata manipulatora
- sektor sa bitnim informacijama

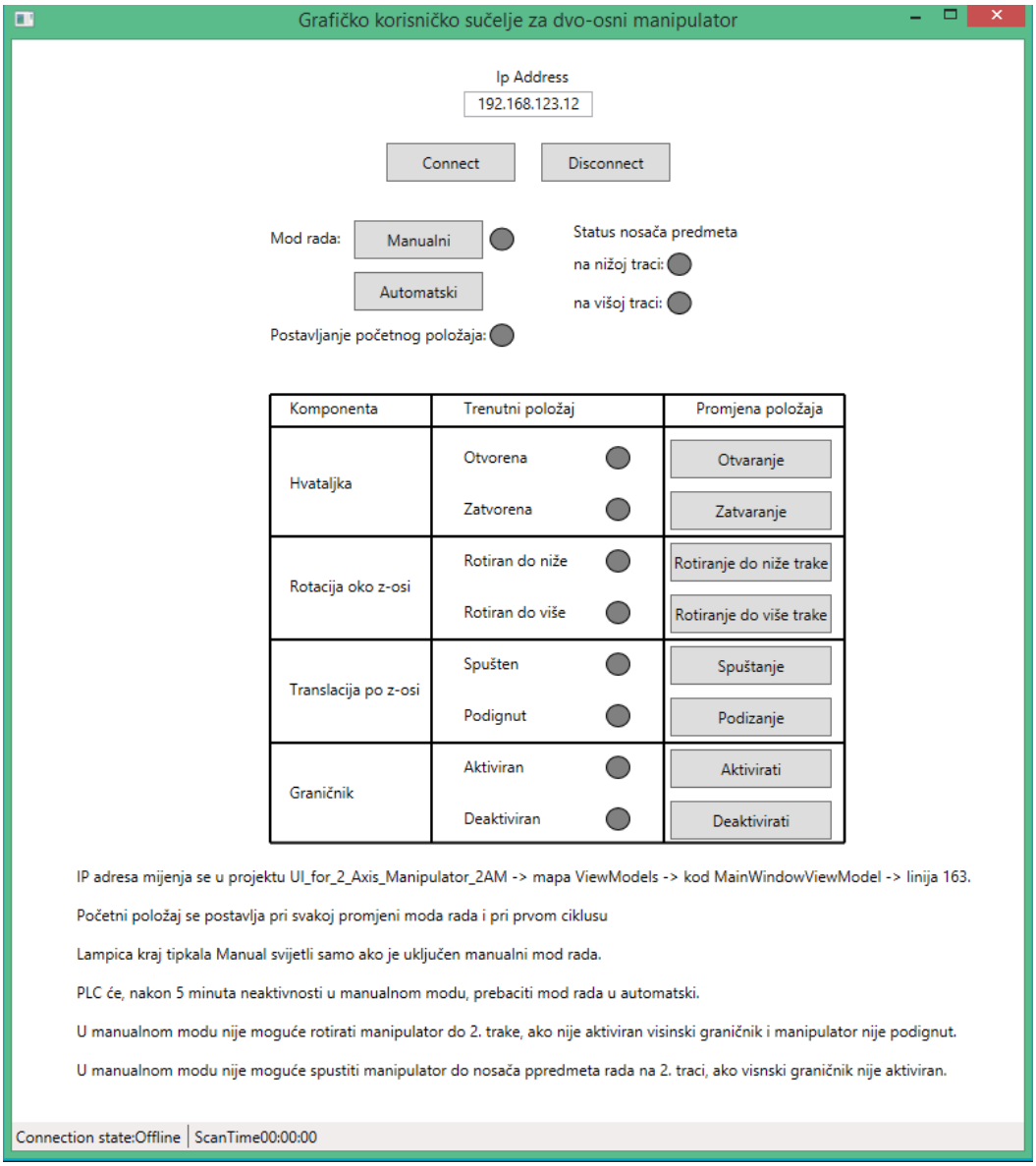
Sektor za komunikaciju sastoji se od IP adrese i tipkala *Connect* i *Disconnect* koja služe za inicijalizaciju komunikacije odnosno prekid komunikacije između sučelja i PLC-a.

Sektor za odabir moda rada i prikaz stanja na transportnim sustavima koristi tipkala Manualni i Automatski za promjenu moda rada manipulatora pri čemu svijetleća lampica označava manualni mod, a ugašena automatski mod rada. Ako je aktiviran manualni mod tada je moguće upravljati manipulatorom pomoću sučelja i vršiti nadzor, a ako je aktivan automatski mod rada moguće je samo vršiti nadzor. Također se u ovom sektoru može vidjeti je li u tijeku postavljanje manipulatora u početni položaj. Prikaz stanja na transportnim sustavima služi kako bi se znalo kada su nosači predmeta rada u poziciji na zaustavnom mjestu.

Sektor za promjenu stanja segmenata manipulatora služi kako bi se u manualnom modu moglo upravljati graničnim položajem određenog segmenta i vidjeti u kojem graničnom položaju je pojedini segment odnosno kako bi se u automatskom modu mogao vršiti nadzor pravilnosti izvršavanja procesa sa nekog udaljenog mjesta.

Sektor sa bitnim informacijama sadrži bitne informacije o načinu manjih izmjena u sučelju, početnom položaju, modovima rada i određenim ograničenjima.

Na slici 10. prikazano je korisničko sučelje.



Slika 10. Izgled sučelja

7.1. Pogled

U sloju Pogled je dizajniran izgled sučelja kako bi bilo što funkcionalnije i logičnije, a opet da oduzima dah kada ga se prvi put ugleda.

7.2. ModelPogleda

Sloj ModelPogleda definira sve značajke koje je potrebno očitavati iz PLC-a, kao i sve naredbe koje se šalju sa sučelja tj povezuje Pogled i Model, odnosno korisničko sučelje i PLC.

7.3. Model

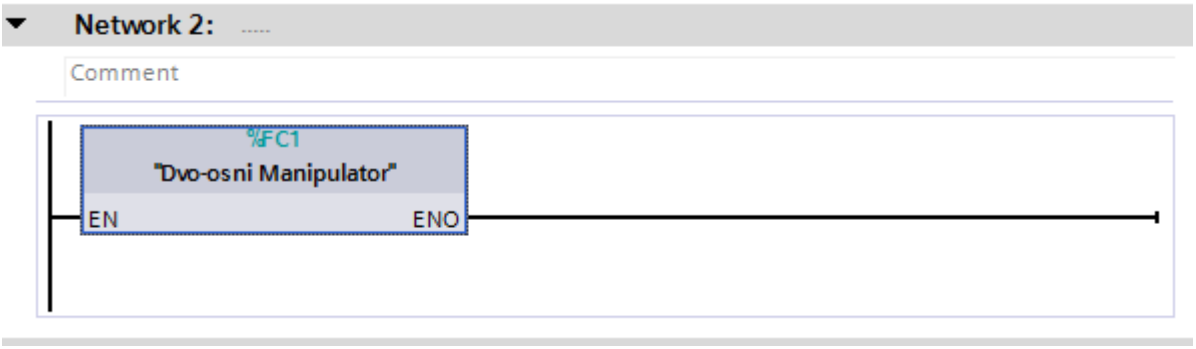
Model sadrži sve potrebne funkcije i protokole kako bi sučelje i PLC uspješno komunicirali, te sve funkcije koje omogućuju čitanje iz podatkovnih blokova PLC-a i pisanje naredbi sa sučelja u PLC.

8. PROJEKTIRANJE I IZRADA UPRAVLJAČKOG PROGRAMA

Upravljački program je osmišljen na način da koristi podprograme (eng. *subroutines*), zbog bolje preglednosti, lakših izmjena i logičnosti izvođenja, i blokove podataka (eng. *Data blocks*) zbog njihove pouzdanosti u odnosu na memorijske bitove, ali i zbog prikazivanja status raznih segmenata na korisničkom sučelju.

8.1. Main

U Main-u se izvršava primarni ciklus skeniranja i tu se nalazi podprogram Dvo-osni manipulator koji je uvijek skeniran jer je priključen direktno na „napajanje“ mreže (eng. *Network*).



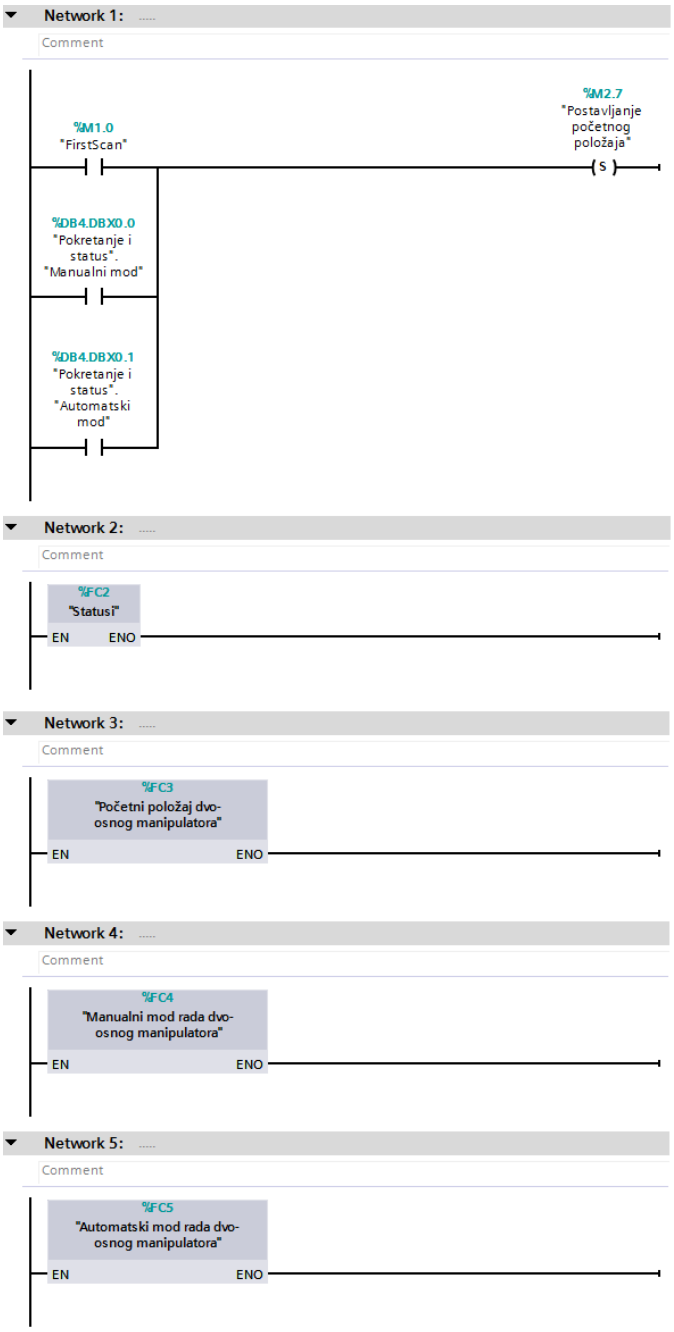
Slika 11. Main

8.2. Dvo-osni manipulator

Dvo-osni manipulator je zapravno glavni sloj upravljačkog programa iz kojeg se izvršavaju svi drugi podprogrami i aktiviranje postavljanja početnog položaja.

Aktiviranje postavljanja početnog položaja se vrši pri prvom skeniranju i promjeni moda rada, aktivacijom odabranog memorijskog bita što prikazuje prva mreža ovog podprograma.

Podprogrami su: Statusi, Početni položaj, Manualni mod rada i Automatski mod rada i mreže dva do pet služe kako bi se kontinuirano skenirali ti podprogrami.



Slika 12. Podprogram Dvo-osni manipulator

8.3. Statusi

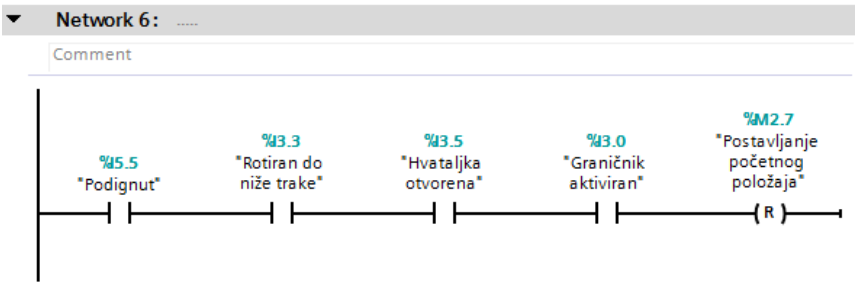
Funkcija podprograma Statusi je bilježenje podataka o postavljanju početnog položaja, statusu nosača predmeta rada i graničnim položajima, a koji se spremaju u podatkovne blokove i šalju sučelju.

8.4. Početni položaj

Podprogram Početni položaj služi kako bi se osiguralo da je, prije početka bilo kakvog rada, manipulator u zadanom početnom položaju, a njegovi segmenti nisu u nekom međupoložaju ili nekom konfliktnom položaju sa okolinom.

Zadani početni položaj je takav da manipulator mora biti translatiran do gornjeg graničnog položaja odnosno podignut (mreža dva), rotiran iznad nižeg transportnog sustava (mreža tri), hvataljka mora biti otvorena (mreža četiri), a visinski graničnik mora biti deaktiviran (mreža pet).

Ovaj podprogram se aktivira pri prvom ciklusu skeniranja ili pri promjeni moda rada iz Automatskog u Manualni i obrnuto te je potrebno aktivirati određeni memorijski bit kako bi se podprogram mogao izvoditi, a deaktiviraju ga granični položaji navedenog početnog stanja (mreža šest).

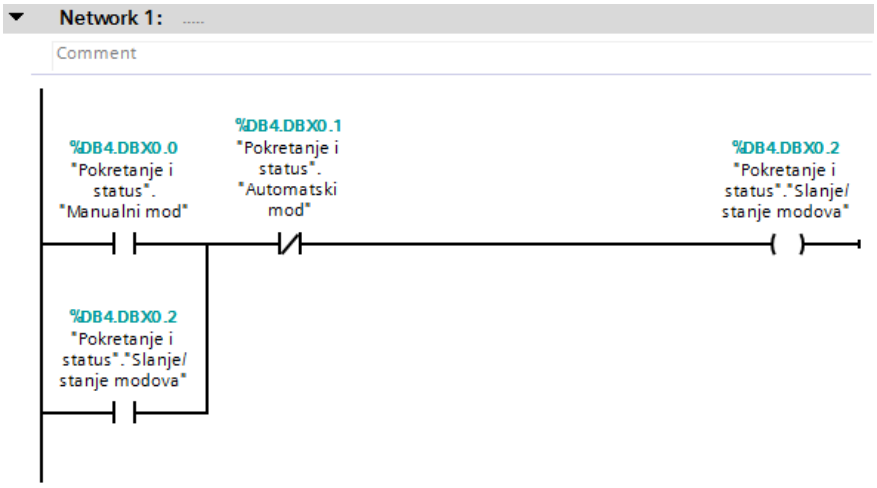


Slika 14. Deaktiviranje memorijskog bita odnosno podprograma

8.5. Manualni mod rada

Manualni mod rada je podprogram kojim se manipulatorom upravlja pomoću korisničkog sučelja.

Kako bi se osiguralo da se Automatski mod ne može izvoditi tijekom izvođenja Manualnog moda, koristi se dominirajuće isključeni spoj samodržanja objašnjen u ranijem poglavlju, a umjesto releja aktivira se „Slanje“ (mreža jedan).



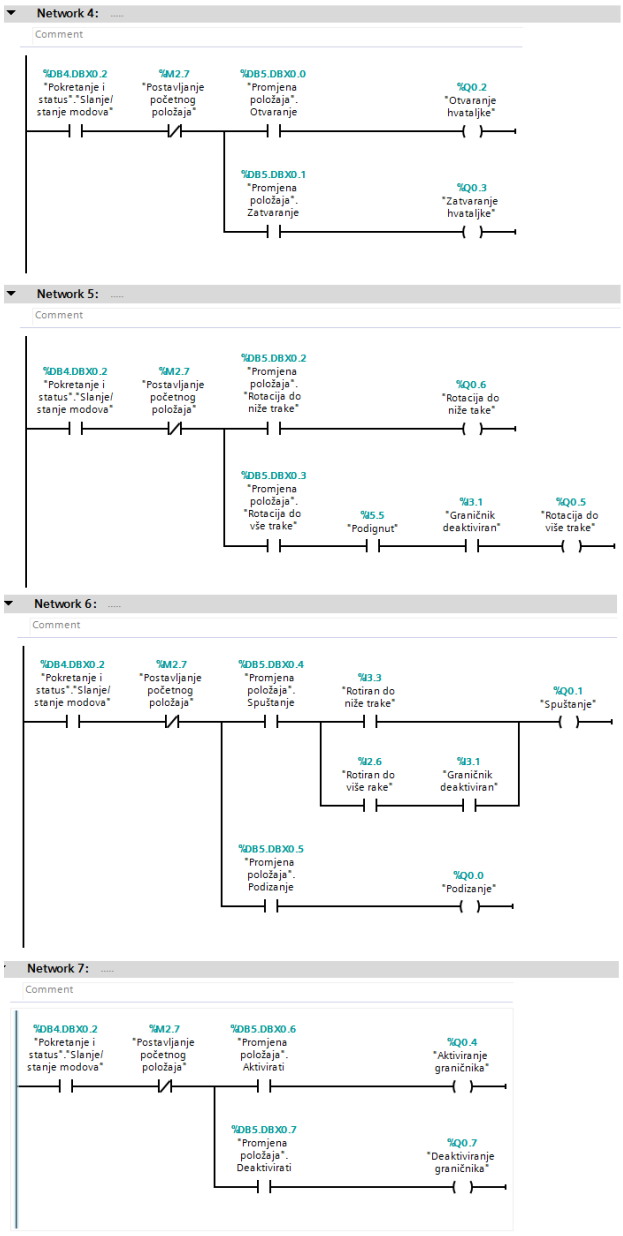
Slika 15. Spoj samodržanja koji omogućuje slanje podataka

Ako je aktivno „Slanje“ i nije aktiviran memorijski bit za postavljanje početnog položaja mogu se slati naredbe sa sučelja.

Također, ako korisnik ne pritisne neko od tipkala u sučelju u vremenskom periodu od 5 minuta, sustav se sam prebacuje u Automatski mod rada (mreže dva i tri). Ovo je postignuto pomoću tajmera s kašnjenjem iskapčanja i tajmera s kašnjenjem ukapčanja s akumulirajućim vremenom.

Mreže četiri, pet, šest i sedam služe za primanje naredbi iz korisničkog sučelja.

Prilikom projektiranja ovog moda rada uzete su u obzir i neke konfliktne naredbe koje korisnik može zadati manipulatoru u smislu da se mnipulator i/ili transportni sustavi mogu oštetiti pa se neke naredbe mogu zadati u točno određenim uvjetima. Te naredbe su Rotiranje do više trake koja je uvjtovana podignutim manipulatorom i aktiviranim visinskim graničnikom i Spuštanje koje se može izvršiti, ako je manipulator rotiran iznad niže trake i graničnik deaktiviran, ili, ako je manipulator rotiran iznad više trake i graničnik je aktiviran.



Slika 16. Primanje podataka iz korisničkog sučelja i sprječavanje štetnih naredbi

8.6. Automatski mod rada

Automatski mod rada je zapravo najvažniji mod rada ovog upravljačkog prorama jer manipulator većinu svog radnog vijeka radi upravo po naredbama iz ovog moda.

Mod je napravljen na način da se prvo napravio logični dijagram tijeka izvršavanja radnji, zatim su identificirani blokirajući signali, potom je iskorišteno znanje elektropneumatske taktne metode i napravljena je električka shema pomoću relejne logike koja je zatim prevedena u ljestvičastu logiku.

8.6.1. Dijagram tijeka

Početna pretpostavka ovog diagrama je da je manipulator postavljen u početni položaj.

Kada dio termoregulatora dođe nižim transportnim sustavom ispod manipulatora, a na višem se nalazi prazni nosač predmeta rad, manipulator translatira u donji granični položaj (spušten), hvataljka se zatvara, manipulator se podiže, aktivira se visinski graničnik, manipulator rotira do višeg transportnog sustava, spušta se do nosača predmeta rada, hvataljka se otvara, maipulator se podiže, zatim rotira do nižeg transportnog sustava te se deaktivira visinski graničnik.

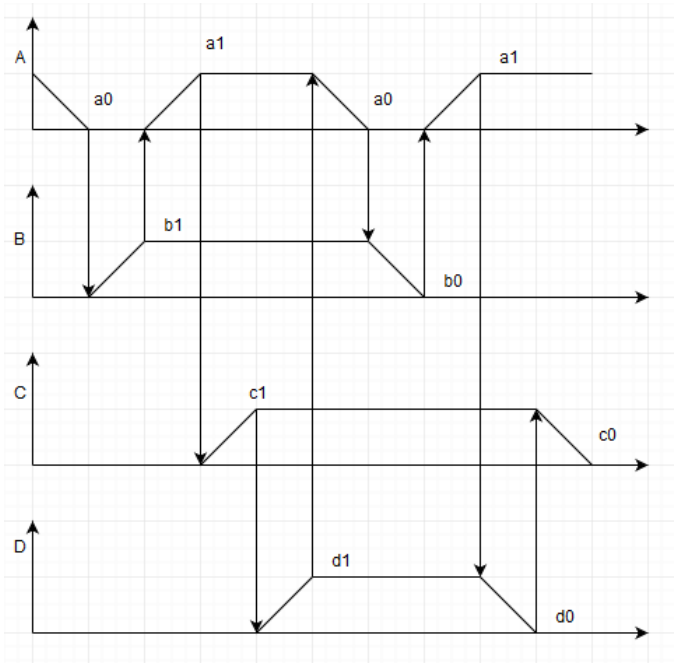
8.6.2. Dijagram put-korak i blokirajući signali

Koristeći prethodno stečena znanja iz pneumatskog i elektropneumatskog upravljanja te prethodni dijagram toka bilo je vrlo jednostavno nacrtati dijagram put-korak i identificirati blokirajuće sigmale.

Potrebno je pojednostaviti manipulator na četiri cilindra: cilindar A čiji je uvučeni položaj ekvivalentan donjem graničnom položaju prilikom translacije, a izvučeni gornjem tj. Spuštenom i podignutom položaju, cilindar B čiji je uvučeni položaj ekvivalentan otvorenoj hvataljci, a izvučeni zatvorenoj, cilindar C čiji je uvučeni položaj ekvivalentan deaktiviranom visinskom graničniku, a izvučeni aktiviranom te cilindar D čiji je uvučeni položaj ekvivalentan rotiranom položaju iznad nižeg transportnog sustava, a izvučeni rotiranom položaju iznad višeg, što je bilo moguće zbog konstrukcijskih značajki manipulatora.

Sada se iz dijagrama tijeka može dobiti redoslijed izvlačenja i uvlačenja cilindara:

A-B+A+C+D+A-B-A+D-C- i prema ovom redoslijedu se crta dijagram put - korak.



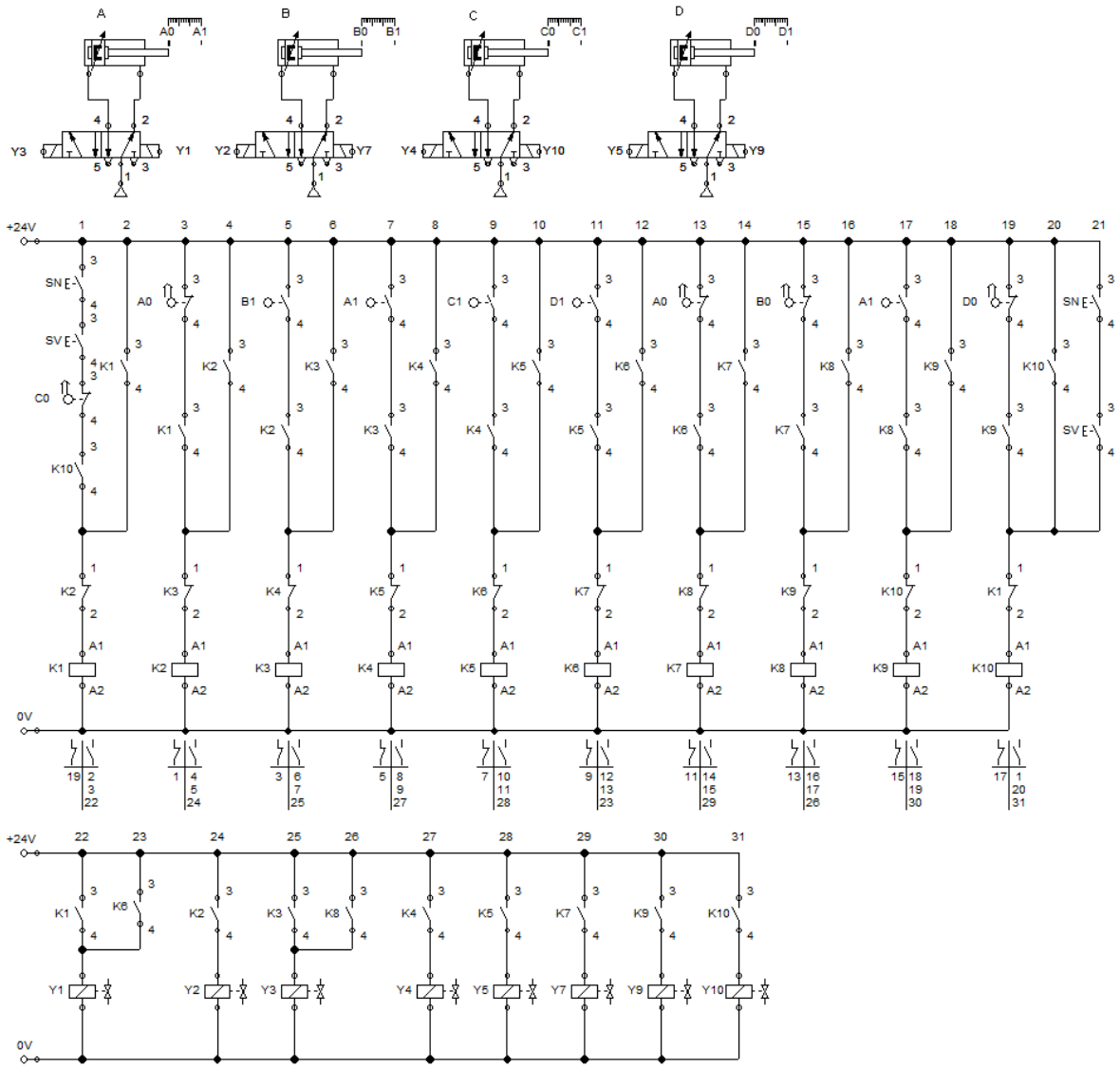
Slika 17. Dijagram put-korak

Blokirajući signali javljaju se prilikom spuštanja i podizanja manipulatora.

8.6.3. Automatski mod rada pomoću elektropneumatske taktne metode simuliran u relejnoj tehnici

Prema pravilima elektropneumatske taktne metode koja su opisana ranije napravljena je pneumatska i električka shema za slijed A-B+A+C+D+A-B-A+D-C-.

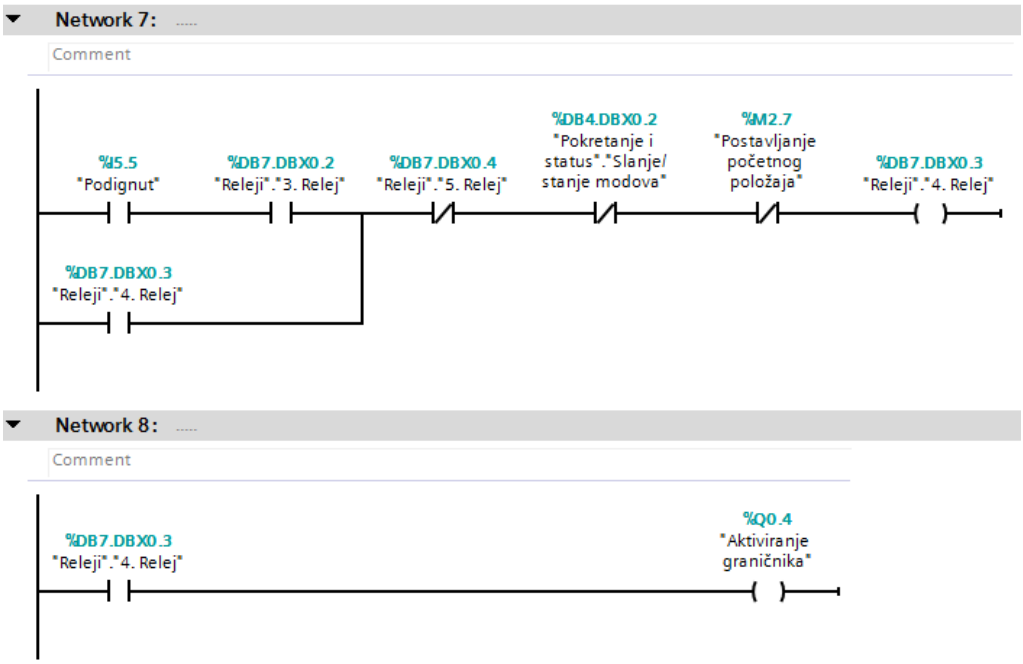
Sheme su prikazane na slici 18.



Slika 18. Automatski mod rada simuliran pomoću releja u FluidSim Pneumatics

8.6.4. Automatski mod rada pomoću elektropneumatske taktne metode u Step7

Program je modeliran prema električkoj shemi elektropneumatske metode. Umjesto releja prvobitno su korišteni memorijski bitovi, ali su oni zamijenjeni podatkovnim blokovima jer su pouzdaniji. Također osim isključivanja pomoću „releja“ neka grana se može isključiti i ako se aktivira „Slanje“ ili ako se aktivira memorijski bit za postavljanje početnog položaja.



Slika 19. Primjer grane prevedene iz relejne u ljestvičastu logiku

8.7. Podatkovni blokovi

Podatkovni blokovi sadrže sva stanja potrebna za nadzor i upravljanje putem korisničkog sučelja, kao i stanja potrebna za izvršenje Automatskog moda rada.

9. 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 učinkovito i jednostavno 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 manipulatora. Na taj način je ostvareno udaljeno upravljanje manipulatora 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, upravljački program sadrži logiku kojom se sprječavaju moguće greške u kretnji manipulatora.

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 Ethernet 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 manipulatorom te očitati vrijednosti koje granični prekidači 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.

10. LITERATURA

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

[2] Nikolić G., Pneumatika i hidraulika: 1. Dio Pneumatika, Školska knjig, 2008.g.

[3] Petrić J., Materijali sa predavanja iz kolegija Pneumatika i hidraulia, FSB, 2016.g.

11. PRILOZI

I. CD-R disk

II. Primjer koda u C#

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 UI_for_2_Axis_Manipulator_2AM.PlcService
{
    public class S7PlcService
    {
        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 bool Otvorena { get; private set; }

public bool Zatvorena { get; private set; }

public bool Rotirandonižetrake { get; private set; }

public bool Rotirandovišetrake { get; private set; }

public bool Spušten { get; private set; }

public bool Podignut { get; private set; }

public bool Aktiviran { get; private set; }

public bool Deaktiviran { get; private set; }

public bool Slanje { get; private set; }

public bool PočetniPoložaj { get; private set; }

public bool SenzorNiža { get; private set; }
```

```
        public bool SenzorViša { 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 WriteManual()
    {
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB4.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("DB4.DBX0.0", false);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " +
_client.ErrorText(writeResult));
            }
        });
    }

    public async Task WriteAutomatic()
    {
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB4.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("DB4.DBX0.1", false);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " +
_client.ErrorText(writeResult));
            }
        });
```

```
    }

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

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

    public async Task WriteRotiranjedonižetrake()
    {
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB5.DBX0.2", true);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " +
_client.ErrorText(writeResult));
            }
            Thread.Sleep(30);
            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 WriteRotiranjedovišetrake()
{
    await Task.Run(() =>
    {
        int writeResult = WriteBit("DB5.DBX0.3", true);
        if (writeResult != 0)
        {
            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " +
_client.ErrorText(writeResult));
        }
        Thread.Sleep(30);
        writeResult = WriteBit("DB5.DBX0.3", false);
        if (writeResult != 0)
        {
            Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " +
_client.ErrorText(writeResult));
        }
    });
}

public async Task WriteSpuštanje()
{
```

```
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB5.DBX0.4", true);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " + _client.ErrorText(writeResult));
            }
            Thread.Sleep(30);
            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 WritePodizanje()
    {
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB5.DBX0.5", true);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " + _client.ErrorText(writeResult));
            }
```

```
            Thread.Sleep(30);
            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 WriteAktivirati()
    {
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB5.DBX0.6", true);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " +
_client.ErrorText(writeResult));
            }
            Thread.Sleep(30);
            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 WriteDeaktivirati()
    {
        await Task.Run(() =>
        {
            int writeResult = WriteBit("DB5.DBX0.7", true);
            if (writeResult != 0)
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Write error: " + _client.ErrorText(writeResult));
            }
            Thread.Sleep(30);
            writeResult = WriteBit("DB5.DBX0.7", 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();
            RefreshValues2();
            OnValuesRefreshed();
        }
        finally
        {
            _timer.Start();
        }
        _lastScanTime = DateTime.Now;
    }

    private void RefreshValues()
    {
        lock (_locker)
        {
            var buffer = new byte[8];
            int result = _client.DBRead(6, 0, buffer.Length, buffer);
            if (result == 0)
            {
                Otvorena = S7.GetBitAt(buffer, 0, 0);
                Zatvorena = S7.GetBitAt(buffer, 0, 1);
                Rotirandonižetrake = S7.GetBitAt(buffer, 0, 2);
                Rotirandovišetrake = S7.GetBitAt(buffer, 0, 3);
                Spušten = S7.GetBitAt(buffer, 0, 4);
                Podignut = S7.GetBitAt(buffer, 0, 5);
                Aktiviran = S7.GetBitAt(buffer, 0, 6);
                Deaktiviran = S7.GetBitAt(buffer, 0, 7);
```

```
            }
            else
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Read error: " + _client.ErrorText(result));
            }
        }
    }

    private void RefreshValues2()
    {
        lock (_locker)
        {
            var buffer = new byte[6];
            int result = _client.DBRead(4, 0, buffer.Length, buffer);
            if (result == 0)
            {
                Slanje = S7.GetBitAt(buffer, 0, 2);
                PočetniPoložaj = S7.GetBitAt(buffer, 0, 3);
                SenzorNiža = S7.GetBitAt(buffer, 0, 4);
                SenzorViša = S7.GetBitAt(buffer, 0, 5);
            }
            else
            {
                Debug.WriteLine(DateTime.Now.ToString("HH:mm:ss") + "\t Read error: " + _client.ErrorText(result));
            }
        }
```

```
    }

    /// <summary>
    /// Writes a bit at the specified address. Es.: DB1.DBX10.2 writes the bit in db 1, word 10, 3rd bit
    /// </summary>
    /// <param name="address">Es.: DB1.DBX10.2 writes the bit in db 1, word 10, 3rd 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 ModelPrikaza

```
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 UI_for_2_Axis_Manipulator_2AM.PlcService;

namespace UI_for_2_Axis_Manipulator_2AM.ViewModels
{
   class MainWindowViewModel : ViewModelBase
   {
      public string IpAddress
      {
         get { return _ipAddress; }
         set { Set(ref _ipAddress, value); }
      }
      private string _ipAddress;

      public bool Otvorena
      {
         get { return _Otvorena; }
         set { Set(ref _Otvorena, value); }
      }
      private bool _Otvorena;

      public bool Zatvorena
      {
         get { return _Zatvorena; }
         set { Set(ref _Zatvorena, value); }
      }
      private bool _Zatvorena;

      public bool Rotirandonižetrake
      {
         get { return _Rotirandonižetrake; }
         set { Set(ref _Rotirandonižetrake, value); }
      }
      private bool _Rotirandonižetrake;

      public bool Rotirandovišetrake
      {
         get { return _Rotirandovišetrake; }
         set { Set(ref _Rotirandovišetrake, value); }
      }
      private bool _Rotirandovišetrake;
```

```
public bool Spušten
{
   get { return _Spušten; }
   set { Set(ref _Spušten, value); }
}
private bool _Spušten;

public bool Podignut
{
   get { return _Podignut; }
   set { Set(ref _Podignut, value); }
}
private bool _Podignut;

public bool Aktiviran
{
   get { return _Aktiviran; }
   set { Set(ref _Aktiviran, value); }
}
private bool _Aktiviran;

public bool Deaktiviran
{
   get { return _Deaktiviran; }
   set { Set(ref _Deaktiviran, value); }
}
private bool _Deaktiviran;

public bool Slanje
{
   get { return _Slanje; }
   set { Set(ref _Slanje, value); }
}
private bool _Slanje;

public bool PočetniPoložaj
{
   get { return _PočetniPoložaj; }
   set { Set(ref _PočetniPoložaj, value); }
}
private bool _PočetniPoložaj;

public bool SenzorNiža
{
   get { return _SenzorNiža; }
   set { Set(ref _SenzorNiža, value); }
}
private bool _SenzorNiža;
```

```
public bool SenzorViša
{
   get { return _SenzorViša; }
   set { Set(ref _SenzorViša, value); }
}
private bool _SenzorViša;

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 ManualCommand { get; private set; }

public ICommand AutomaticCommand { get; private set; }

public ICommand OtvaranjeCommand { get; private set; }

public ICommand ZatvaranjeCommand { get; private set; }

public ICommand RotiranjedonižetrakeCommand { get; private set; }

public ICommand RotiranjedovišetrakeCommand { get; private set; }

public ICommand SpuštanjeCommand { get; private set; }

public ICommand PodizanjeCommand { get; private set; }

public ICommand AktiviratiCommand { get; private set; }

public ICommand DeaktiviratiCommand { get; private set; }

S7PlcService _plcService;

public MainWindowViewModel()
{
   _plcService = new S7PlcService();
```

```
        ConnectCommand = new RelayCommand(Connect);
        DisconnectCommand = new RelayCommand(Disconnect);
        ManualCommand = new RelayCommand(async () => { await Manual(); });
        AutomaticCommand = new RelayCommand(async () => { await Automatic(); });
        OtvaranjeCommand = new RelayCommand(async () => { await Otvaranje(); });
        ZatvaranjeCommand = new RelayCommand(async () => { await Zatvaranje(); });
        RotiranjedonižetrakeCommand = new RelayCommand(async () => { await
Rotiranjedonižetrake(); });
        RotiranjedovišetrakeCommand = new RelayCommand(async () => { await
Rotiranjedovišetrake(); });
        SpuštanjeCommand = new RelayCommand(async () => { await Spuštanje(); });
        PodizanjeCommand = new RelayCommand(async () => { await Podizanje(); });
        AktiviratiCommand = new RelayCommand(async () => { await Aktivirati(); });
        DeaktiviratiCommand = new RelayCommand(async () => { await Deaktivirati(); });

        IpAddress = "192.168.123.12";

        OnPlcServiceValuesRefreshed(null, null);
        _plcService.ValuesRefreshed += OnPlcServiceValuesRefreshed;
    }

    private void OnPlcServiceValuesRefreshed(object sender, EventArgs e)
    {
        ConnectionState = _plcService.ConnectionState;
        Otvorena = _plcService.Otvorena;
        Zatvorena = _plcService.Zatvorena;
        Rotirandonižetrake = _plcService.Rotirandonižetrake;
        Rotirandonižetrake = _plcService.Rotirandonižetrake;
        Spušten = _plcService.Spušten;
        Podignut = _plcService.Podignut;
        Aktiviran = _plcService.Aktiviran;
        Deaktiviran = _plcService.Deaktiviran;
        Slanje = _plcService.Slanje;
        PočetniPoložaj = _plcService.PočetniPoložaj;
        SenzorNiža = _plcService.SenzorNiža;
        SenzorViša = _plcService.SenzorViša;
        ScanTime = _plcService.ScanTime;
    }


    private void Connect()
    {
        _plcService.Connect(IpAddress, 0, 1);
    }

    private void Disconnect()
    {
        _plcService.Disconnect();
    }
```

```
private async Task Manual()
{
   await _plcService.WriteManual();
}

private async Task Automatic()
{
   await _plcService.WriteAutomatic();
}

private async Task Otvaranje()
{
   await _plcService.WriteOtvaranje();
}

private async Task Zatvaranje()
{
   await _plcService.WriteZatvaranje();
}

private async Task Rotiranjedonižetrake()
{
   await _plcService.WriteRotiranjedonižetrake();
}

private async Task Rotiranjedovišetrake()
{
   await _plcService.WriteRotiranjedovišetrake();
}

private async Task Spuštanje()
{
   await _plcService.WriteSpuštanje();
}

private async Task Podizanje()
{
   await _plcService.WritePodizanje();
}

private async Task Aktivirati()
{
   await _plcService.WriteAktivirati();
}

private async Task Deaktivirati()
{
   await _plcService.WriteDeaktivirati();
}
```

```
    }
}
```