Mehanizam za sigurnu komunikaciju robota na računalnoj mreži

Hering, Džuliano

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Džuliano Hering

Zagreb, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

doc. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Džuliano Hering

Zagreb, 2022.

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

Zahvaljujem mentoru doc. dr. sc Tomislavu Stipančiću na pomoći i podršci prilikom izrade ovog rada.

Također zahvaljujem obitelji i prijateljima koji su mi pružali podršku tijekom studiranja i pisanja ovog rada.

Džuliano Hering

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske 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: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Džuliano Hering** JMBAG: **0035217826**

Naslov rada na hrvatskom jeziku: **Mehanizam za sigurnu komunikaciju robota na računalnoj mreži**

Naslov rada na engleskom jeziku: **Mechanism for secure communication of robots on the computer network**

Opis zadatka:

Da bi računala mogla uspješno komunicirati u sklopu računalne mreže ona u konačnici moraju koristiti iste protokole. Kod sinkrone komunikacije prijenos podataka mora biti siguran, pouzdan i brz.

U radu je potrebno:

- istražiti dobre i loše strane Modbus te Socket mrežne tehnologije,
- povezati dva robota koristeći oba principa, usporediti ih te odabrati prihvatljivije rješenje,
- koristeći odabrano rješenje simulirati napad trećim računalom koje se nalazi na istoj mreži te istražiti mogućnost utjecaja tog računala na komunikacijski proces robota,
- razviti svoje rješenje za zaštitu od takvih napada čime bi se ostvarila sigurna i pouzdana razmjena podataka među robotima.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2021.

Zadatak zadao:
Doc. dr. sc. Tomislav Stipančić

Datum predaje rada:
1. rok: 24. 2. 2022.
2. rok (izvanredni): 6. 7. 2022.
3. rok: 22. 9. 2022.

Predviđeni datumi obrane:
1. rok: 28. 2. – 4. 3. 2022.
2. rok (izvanredni): 8. 7. 2022.
3. rok: 26. 9. – 30. 9. 2022.

Predsjednik Povjerenstva:
Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	**Opis**
ASCII	*American Standard Code for Information Interchange*
RTU	*Remote Terminal Unit*
LRC	*Longitudinal Redundancy Check*
TCP	*Transmission Control Protocol*
ARAPNET	*Advanced Research Projects Agency Network*

SAŽETAK

Tema ovog rada je povezati dva robota i osigurati im brzu i pouzdanu izmjenu podataka. U uvodnom dijelu objašnjen je teorijski princip rada modbus protokola i socket mrežne tehnologije te je napravljena njihova usporedba. Zatim slijedi praktično povezivanje robota pomoću obje metode koristeći takozvani virtualni stroj. Sigurna komunikacija se osigurava programiranjem servera u programskom jeziku Python. U zadnjem dijelu opisane su moguće posljedice trećeg računala na proces komunikacije te je iznesen osvrt na eventualna poboljšanja i daljna istraživanja.

Ključne riječi: komunikacija, protokol, modbus, TCP/IP, *socket*, *master* i *slave* uređaj , server, klijent, *Python.*

SUMMARY

The topic of this paper is communication protocols between two robots. In the first part, we will theoretically explain both Modbus and socket communication. Afterwards, we will compare two protocols and choose the more suitable one. Virtual machines have been used for implementing the communication protocols. Secure communication will be established using Python program. In the end, some of the issues that can occur during the communication are described and we give some insights on improvements and future research.

Key words: communication, protocol, Modbus, TCP/IP, socket, master and slave device, server, client, Python.

.

1. UVOD

Prijenos podataka postao je jedan od najvažnijih segmenata u 21. stoljeću te je izrazito bitno osigurati siguran prijenos istih. Jednako je bitno da komunikacija bude brza i pouzdana. Upravo takva svojstva postižemo sa sinkronom komunikacijom. Jedna od njenih bitnijih primjena je upravo u inženjerstvu gdje se svakodnevno manipulira s velikom količinom podataka te je nužno da se njihov prijenos odvija bez većih poteškoća. Navedena komunikacija najčešće se zbiva između dva ili više računala odnosno u konkretnom radu riječ je o interakciji između dva robota. Postoje mnoge metode pomoću kojih roboti mogu komunicirati ali će u okvirima ovog rada zadržati na dvije najrasprostranjenije metode. Prva metoda je takozvani Modbus protokol, a druga metoda se zasniva na socket mrežnoj tehnologiji. Svaka od navedenih tehnologija ima svoje i prednosti i nedostatke koji će biti detaljno razrađeni kroz poglavlja.

U idućim poglavljima vidjet će se da je jedno rješenje ipak bolje od drugih ali kao i sve ostale stvari u inženjerstvu treba sagledati širu sliku i razlog primjene. Bolje rješenje ne znači nužno i uvijek upotrebljivo te treba biti oprezan prilikom odabira komunikacijskog protokola. Komunikacija između dva robota ostvarit će se pomoću virtualnih strojeva koji zamjenjuju fizičku komponentu robota te se njihovom primjenom značajno štedi na vremenu i novcu.

2. KOMUNIKACIJSKI PROTOKOLI

2.1 Modbus Protokol

Modbus je jedan od najpoznatijih serijskih komunikacijskih protokola za podatkovnu komunikaciju između uređaja povezanih na različite sabirnice ili mreže. Prvobitno se koristio za razmjenu podataka između dva ili više programabilna logička kontrolera (PLC-a). Protokol je kreiran od strane Modicon (danas Schneider Electric UK) kompanije 1979. godine. Iako protokol nije službeno standardiziran opće je prihvaćen u industriji radi svoje jednostavne primjene. Kod spomenute serijske komunikacije protokol se bazira na *master-slave* principu, a kada se koristi Ethernet mreža onda se komunikacija odvija između *client-server* načelu. [1] Modbus protokol definira dva glavna načina prijenosa podataka:

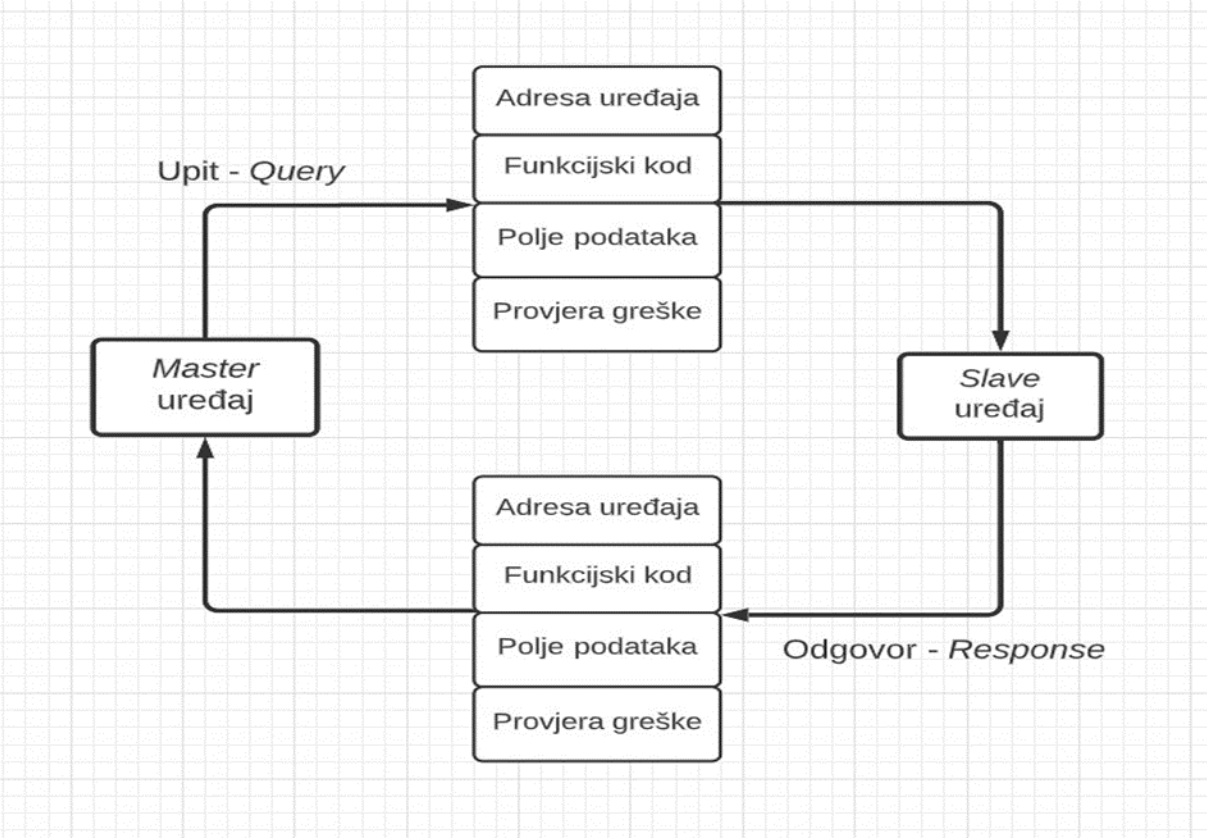
- ASCII (eng. *American Standard Code for Information Interchange*)
- RTU *(*eng. *Remote Terminal Unit*)

Razlika između ova dva načina za prijenos podataka očituje se u načinu spremanja odnosno pakiranja podataka u poruku te njihovo dekodiranje. Postoje i ostale verzije Modbus protokola poput Modbus UDP, Modbus Plus, Pemex Modbus itd. ali oni imaju specijalne namjene te kao takvi neće biti predmet diskusije u ovome radu. [1]

2.1.1 Modbus ASCII

Informacija odnosno poruka koja se prenosi putem Modbus protokola naziva se upit ili naredba (eng. *query*). Prvo ćemo definirati strukturu poruke kod ASCII (eng. *American Standard Code for Information Interchange*) prijenosa podataka. Svaki upit (eng. *query*) *master* uređaja i odgovor (eng. *response*) *slave* uređaja sadrži istu strukturu koja je podijeljena na 4 dijela: adresa uređaja, funkcijski kod, polje podataka te blok koji je zadužen za ispravnost upita ili odgovora odnosno provjeru greške. [2]

Generalno, poruka koju *master* uređaj prosljeđuje *slave* uređaju ima istu strukturu kao i povratna poruka od slave uređaja. Razlikuju se pojedini sadržaji svakog dijela odnosno bloka što će biti opisano sljedećim poglavljima. [3] Grafički prikaz i cijeli jedan ciklus izmjene podataka s korištenim ASCII prijenosom podataka možemo vidjeti na slici. [Slika 2.1]



Slika 2.1 Ciklus Modbus poruke pri ASCII protokolu

Adresa uređaja (eng. *slave adress*) razlikuje se između ASCII prijenosa i RTU prijenosa podataka. Kod ASCII (eng. *American Standard Code for Information Interchange*) prijenosa, adresa uređaja je opisana pomoću polja koje sadrži dva znaka od 00 do FF koja su u heksadecimalnom zapisu. *Master* uređaj upisuje adresu *slave* uređaja kojem želi poslati upit, a *slave* uređaj kada odgovara nazad upisuje također svoju adresu u specifično polje kako bi *master* uređaj znao koji slave uređaj odgovara nazad. [4]

Drugo polje se naziva funkcijski kod (eng. *function code*) koji je također kod ASCII prijenosa opisan s dva znaka od 00 do FF. Funkcijski kod koji master uređaj šalje sadrži specifične informacije koje slave uređaj treba izvršiti. Moguće naredbe su čitanje specifičnog registra, upisivanje brojčane vrijednosti u registar, promjena stanja nekog od ulaza itd. Slave uređaj vraća identičnu poruku ako je u mogućnosti izvršiti naredbu. Ukoliko se dogodi neočekivana greška, *slave* uređaj šalje gotovo identičnu poruku, jedina razlika je promjena najznačajnijeg bita (skroz lijevo) u logičku jedinicu. Funkcijski kod može biti u rasponu od 1-255 ovisno o vrsti uređaja. [4]

Treće polje naziva se polje podataka (eng. *data field*). Kreira se od para heksadecimalnih znakova pri čemu one ovise o načinu prijenosa, kod ASCII prijenosa ono može biti duljine par ASCII znakova. Master uređaj u ovo polje upisuje adrese registara gdje treba izvršiti promjenu te broj traženih podataka. Slave uređaj odgovara s traženim podatcima, odnosno upisuje kod greške zašto nije izvršena tražena naredba.

Zadnje polje, odnosno provjera greške (eng. *error checking field*) provjerava ispravnost poruke. Također razlikujemo dva načina ovisno o načinu prijenosa, kod ASCII prijenosa koristi se LRC (eng. *longitudinal redundancy check*) koji ima ugrađen algoritam koji poruku podijeli na više manjih dijelova. [5]

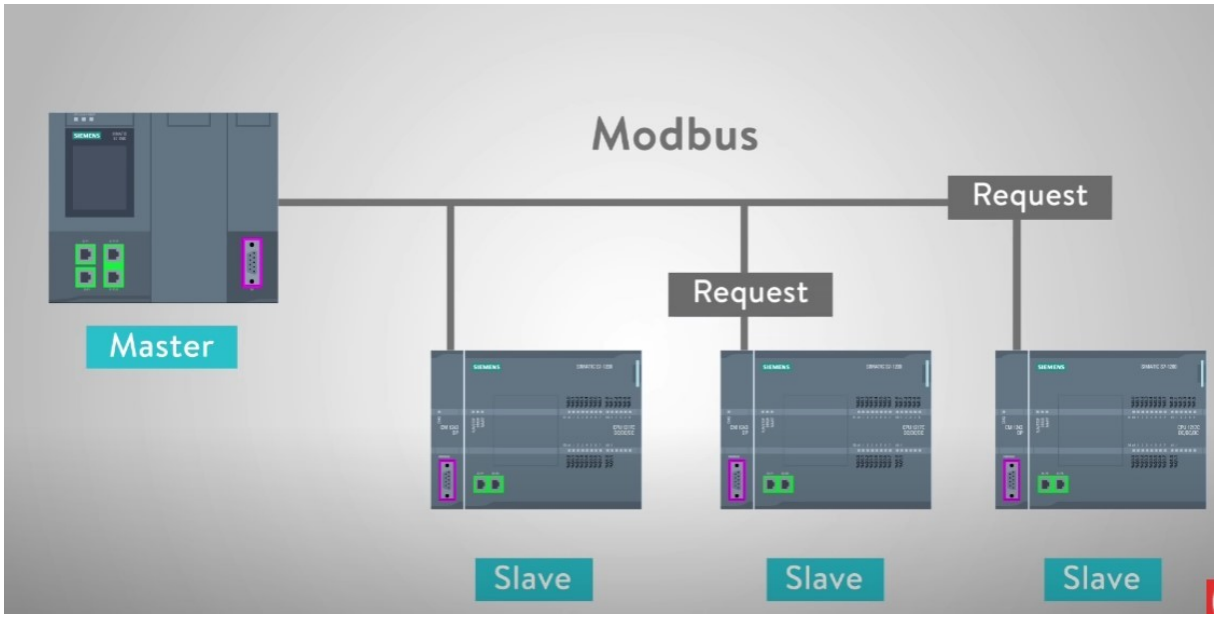
Na samom početku poruke ASCII prijenos generira znak dvotočke (:), zatim slijede četiri opisana polja i na kraju dolazi kraj poruke koji sadrži dva znaka CrLf (eng. *carriage return-line feed*). Znakovi se šalju uzastopno uz mogućnost razmaka do maksimalno jedne sekunde bez očitovanja greške. Strukturu cijele poruke kod ASCII prijenosa možemo vidjeti na slici. [Slika 2.2]

Slika 2.2 Struktura ASCII poruke

Modbus ASCII zasniva se na *master-slave* principu. U pravilu postoji jedan *master* uređaj te jedan ili više *slave* uređaja (najviše njih 247). Prilikom komunikacije *master* uređaj šalje zahtjev (eng. *request*) jednom ili više *slave* uređaja koji zatim uzvraćaju povratnu informaciju u obliku odgovora (eng. *response*). [5]

Važno je napomenuti da kod ASCII prijenosa jedino *master* uređaj može inicirati komunikaciju, a *slave* uređaji mogu isključivo slati povratne informacije (eng. *reponse*). Poruka koju *master* uređaj šalje jednom *slave* uređaju još nazivamo upit ili naredba (eng. *query*). Ukoliko *master* uređaj šalje poruku svim slave uređajima tada se to naziva emitiranje (eng. *broadcast*). *Slave* uređaj će odgovoriti na poruku u obliku potvrde prijema, potvrdom naredbe ili slanjem traženih podataka od strane *master* uređaja. Ako iz bilo kojeg razloga *slave* uređaj ne može provesti upit ili naredbu (primjerice, zagrijati traženi prostor na neodrživu temperaturu ili pomaknuti zglob robota u statički nemoguć položaj) to nazivamo greškom (eng. *exception response*). Greška koju će slave uređaj poslati nazad *master* uređaju sastoji se od adrese *slave* uređaja, naredba ili upit koju je master uređaj tražio te zašto ona nije provedena. Pod uvjetom da *master* uređaj ne primi povratni odgovor (eng. *response*) tada će se ponovno poslati zahtjev (eng. *request*) sve dok se ne dobije povratni odgovor. [6]

Primjer ASCII prijenosa podataka između jednog *master* uređaja i tri *slave* uređaja možemo vidjeti na slici. [Slika 2.3]

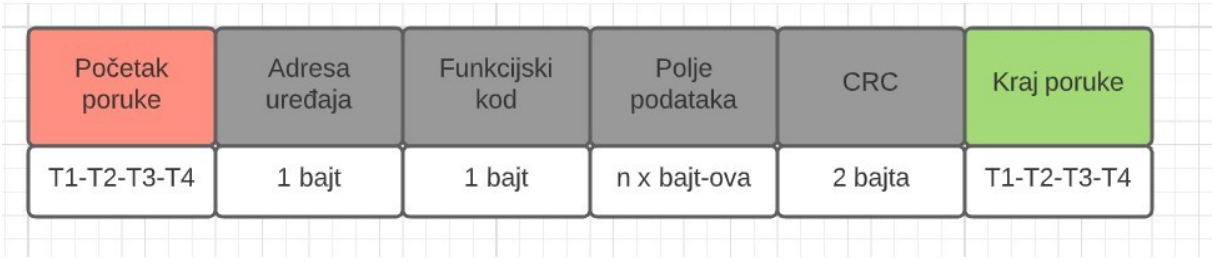


Slika 2.3 ASCII prijenos podataka između više uređaja

2.1.2 Modbus RTU

Modbus TCP/IP je jednostavni Modbus protokol koji je baziran na TCP (eng. *transmission control protocol*) sučelju koji se pokreće na *ethernet* mreži. Prvobitno ethernet mreža bila je zamišljena za uredsko okruženje, a ne za potpuno drukčije industrijsko okruženje. Nekoliko godina kasnije, razvojem industrije i raznih konektora ethernet mreža uspješno je pronašla svoje mjesto u industriji. Drugim riječima, pomoću TCP/IP možemo izmjenjivati binarne podatke između dva računala. Bitno je napomenuti da je TCP/IP transportni protokol i pomoću njega nije definirano što poslani podatci znače niti kako će oni biti protumačeni, naime to odrađuje aplikacijski protokol (eng. *application protocol*), a u ovom slučaju to je Modbus. [7]

Kod Modbus RTU (eng. *remote transmission unit*) protokola struktura poruke je drukčija nego kod ASCII protokola. Sastoji se od 4 različitih polja isto kao i kod ASCII protokola međutim polja se razlikuju u veličini podataka i načinu prijenosa podataka. Strukturu poruke kod Modbus RTU protokola možemo vidjeti na slici. [Slika 2.4]



Slika 2.4 Struktura RTU poruke

Početak poruke koji je definiran kao pauza u komunikaciji. Vremensko trajanje pauze zavisi od brzine Modbus mreže ali minimalni iznos je 3.5 heksadecimalne znamenke. Na slici [Slika 2.4] to je označeno kao T1-T2-T3-T4. Isti protokol ponavlja se i za kraj poruke.

Polje podataka definirano je s *n* brojem bajtova te ono može varirati zavisno kakva je naredba spremljena u poruci.

Kod RTU (eng. *remote terminal unit*) prijenosa podataka svaki bajt se šalje izravno u heksadecimalnom zapisu u obliku dvije znamenke bez konverzije u ASCII (eng. *American standard code for information interchange*) format. Zbog toga brzina komunikacije i prijenos podataka je puno veći nego kod ASCII prijenosa podataka. [8]

2.2. Socket mrežna tehnologija

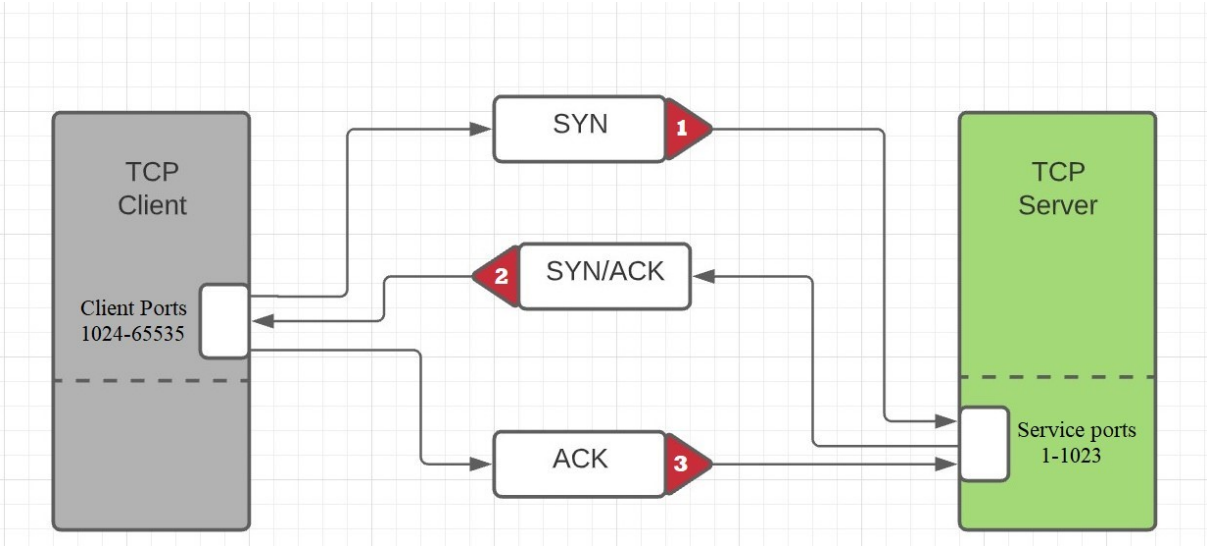
Inačica socket mrežne tehnologije te samog interneta spominje se još 1971. godine prilikom osnivanja ARPANET-a (eng. *Advanced Research Projects Agency Network*). Prvi put se spominje pojam izmjena paketa (*eng. packet switching*) koji je zaslužan za grupiranje podataka odnosno informacija u jedan paket čime je olakšano njegovo prenošenje. Ovako opisani paket se kasnije prenosio putem digitalne mreže. ARPANET organizacija je bila financirana od strane Američke vlade što govori o važnosti projekta i značajnoj tehnologiji koja će se pokazati revolucionarna u desetljećima koja slijede.

Pristupna točka (eng. *socket*) omogućava dvosmjernu izmjenu podataka između servera (eng. *server*) i klijenta (eng. *client*) u realnom vremenu. U konkretnom primjeru koristit će se programski jezik Python zbog toga što radi na principu sinkronosti. Dodatkom dretve (eng. thread) server može izvršavati naredbe koje su mu bile zadane prije te dodatno prihvaćati podatke koji pristižu na server. [9]

Prvi korak je definiranje pristupne točke (eng. socket) koja mora biti spremljena u zasebnu varijablu u kojoj se definira na koji način želimo izvršavati komunikaciju preko pristupnih točaka. U našem slučaju to je TCP/IP komunikacija ali postoje i druge mogućnosti, primjerice UDP komunikacija koja se zasniva na slanju jednog podatkovnog paketa i jednom odgovoru. Nakon traženog odgovora komunikacija se prekida što nije slučaj kod TCP/IP komunikacije.

Objasnit ćemo osnovni način rada socket komunikacije. Početak rada započinje s pristupnom točkom koja je zadužena za očekivanje/slušanje konekcija (eng. *listening socket*) od strane klijenata. Pod uvjetom da jedan od klijenata pošalje zahtjev za povezivanje, server poziva naredbu prihvaćanja (eng. *accept*) kojom se prihvaća povezanost. [10]

Nakon što je server prihvatio određeno spajanje klijenta, slijedi povezivanje klijenta i uspostavljenje veze u tri koraka (eng. *three-way handshake*). Opisani korak je od izrazitog značaja jer on osigurava da je svaki sudionik mreže dostupan, drugim riječima da klijent može komunicirati sa serverom i obratno. Uspostavljanje veze u tri koraka (eng. *three-way handshake*) prikazano je na slici. [Slika 2.5]

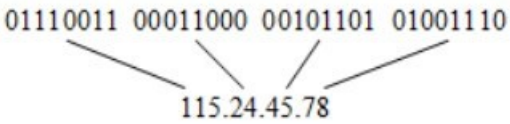


Slika 2.5 Uspostavljanje veze u tri koraka

Nakon uspješnog povezivanja servera i klijenta sljedeći korak je takozvana povratna sekcija (eng. *round-trip section*) koja je zaslužna za izmjenu podataka između servera i klijenta. Poslije izmijenjenih podataka, server i klijent zatvaraju svoje pristupne točke (eng. *socket*). [11]

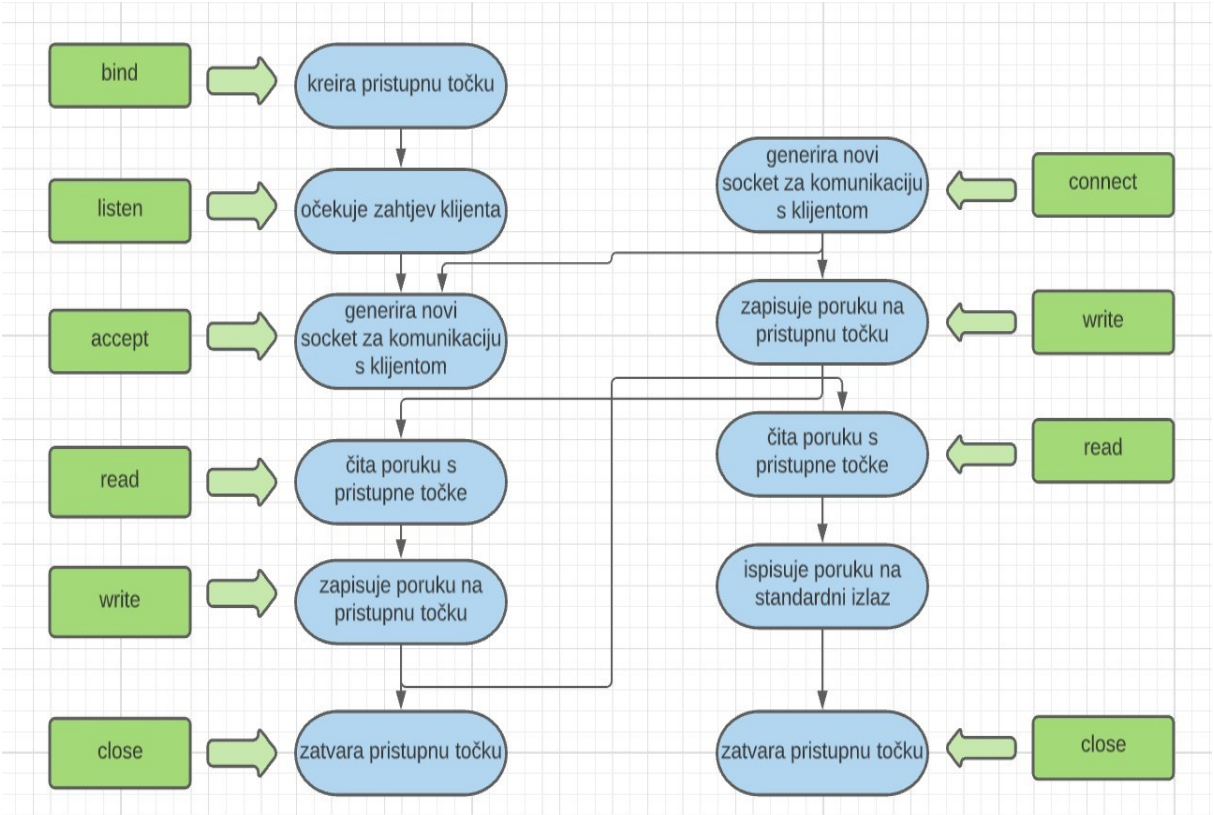
Centralno računalo (eng. *host*) odnosno ono koje pokreće server ima svoje korisničko ime (eng. username) i IP. Ukoliko koristimo IP adresu ona mora biti u IPv4 formatu, a to je danas najrašireniji protokol koji je korišten na računalnoj mreži. IPv4 adresa koja se često skraćeno naziva samo IP je 32-bitni binarni broj. Označavaju se s 4 bajta koja su odvojena decimalnom točkom zbog jednostavnosti u radu. Svaki bajt je predstavljen u decimalnom obliku te prikazuju adresu adresu od 4 broja koja su odvojena točkom.

Navedeni način adresiranja naziva se decimalno adresiranje (eng. *dotted decimal notation*). Radi sigurnosnih nedostataka razvila se naprednija verzija IPv4 protokola koja se naziva IPv6. [12] Takva adresa sadrži 128-bitni binarni broj što znači da automatski može podržavati puno veći broj adresa. Primjer jedne IPv4 adrese prikazan je na slici. [Slika 2.6]



Slika 2.6 Prikaz IPv4 adrese

Primjer jednostavne socket komunikacije možemo vidjeti na slici. [Slika 2.7] U zelenim poljima su definirane i varijable koje se koriste za pojedine naredbe. Naredba *bind* primjerice kreira pristupnu točku (eng. *socket*), naredba *read* primjerice čita poruku s pristupne točke itd.



Slika 2.7 Prikaz jednostavne socket komunikacije

2.3 Usporedba Modbus komunikacije sa socket tehnologijom

U ranijim poglavljima detaljno je objašnjena Modbus komunikacija (ASCII i RTU protokoli) te socket mrežna tehnologija. Pomoću sljedeće tablice [Tablica 2.1] prikazat ćemo glavne prednosti i nedostatke svake od ovih tehnologija. [13]

Modbus protokol	**Socket mrežna tehnologija**
Slobodan izvor (eng. open-source) odnosno tehnologija je javno dostupna svima.	Spor početak prilikom uspostavljanja inicijalne mreže sve dok se ne postigne maksimalni prijenos podataka na mreži.
Jednostavna struktura poruke.	Kompleksna struktura poruke.
Komunikacija između uređaja moguća u jednom smjeru (iznimka su specifični protokoli poput RTU-a).	Komunikacija između uređaja moguća je u oba smjera, server prema klijentu ali i klijent prema serveru.
Jednostavan za korištenje, ne zahtijeva mnogo predznanja (eng. *user friendly*).	Zahtijeva predznanje iz programiranja te je sustav kompleksiniji od Modbus sustava.
Nema zaštitinih mehanizama, poput enkripcije podataka pa je podložan virtualnim napadima.	Posoje zaštitni protokoli koje je moguće implementirati čime se znatno poveća sigurnost podataka na mreži.

Tablica 2.1 Usporedba Modbus protokola i socket mrežne tehnologije

2.4. Izazovi

Načini povezivanja robota objašnjeni su u prethodnim poglavljima te se može zaključiti da svaki protokol ima svoje prednosti i nedostatke pa ovisno o situaciji treba izabrati prihvatljivo rješenje. Ipak, prednost prilikom odabira bih se trebala pružiti socket mrežnoj tehnologiji radi većeg broja prednosti koje posjeduje ali i radi brojnih nedostataka same Modbus komunikacije. Kao njene najveći nedostatak definitivno treba izdvojiti jednosmjernu komunikaciju i nemogućnost nadogradnje sigurnosnog protokola. Iako unazad desetak godina takvi protokoli nisu bili potrebni danas se s razvitkom tehnologije razvila i sve veća potreba za sigurnijom komunikacijom. Korištenjem socket mrežne tehnologije se uvijek mogu implementirati takvi zaštitni protokoli te se njihova nadogradnja praktički beskonačna. U zadnjem poglavlju detaljnije će se opisati sigurnosni protokoli koji se mogu implementirati. [14]

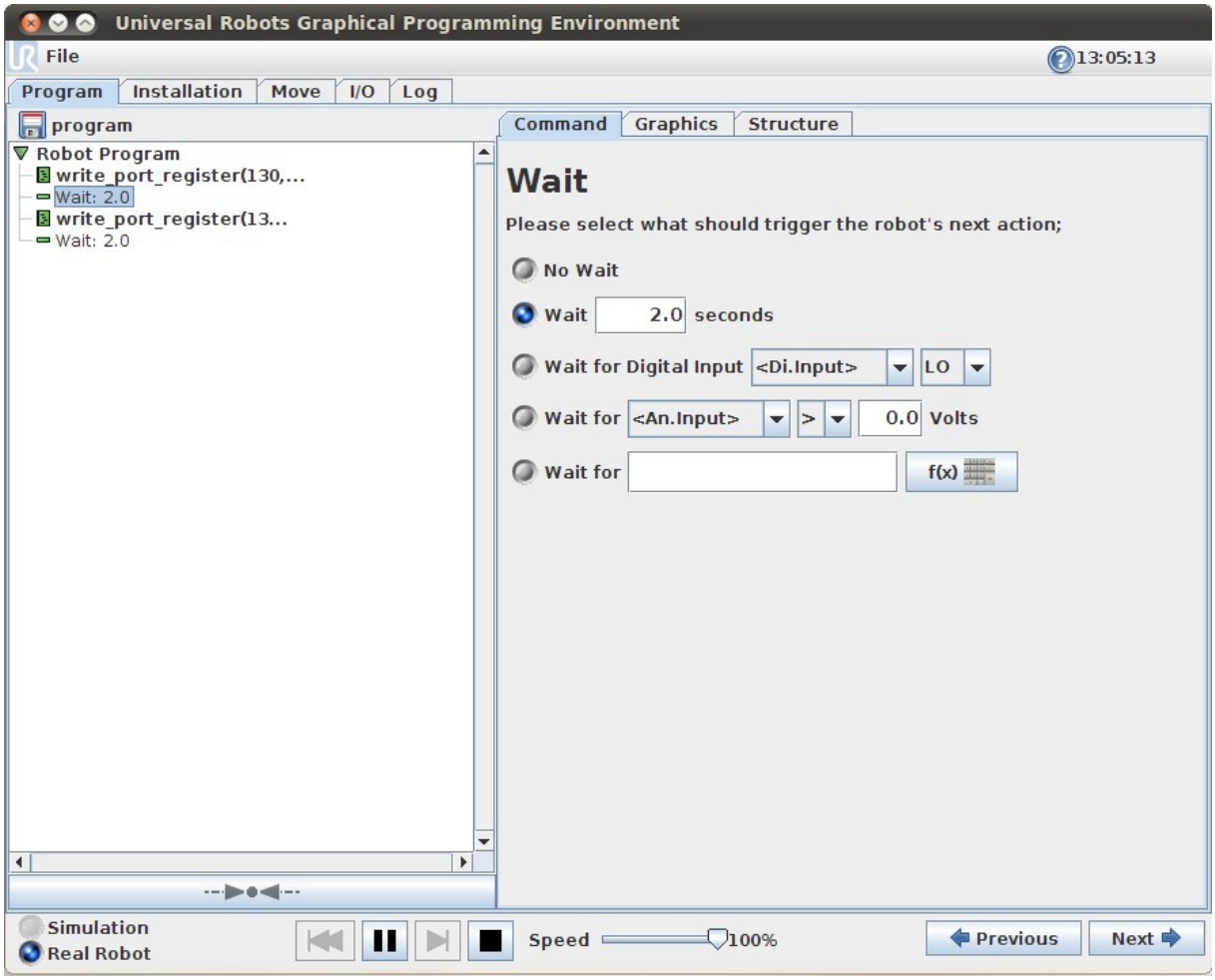
3. POVEZIVANJE ROBOTA

3.1 Povezivanje robota pomoću Modbus protokola

Za ovu prigodu koristit će se takozvani virtualni stroj (eng. *virtual machine*). Njegovo korištenje omogućuje upotrebu zasebnog, novog računala u novom prozoru. U konkretnom slučaju virtualni stroj će zapravo zamijeniti fizičkog robota. Upravo to je i glavna namjena virtualnog stroja, testiranje raznih aplikacija i njihova primjena u mnogo sigurnijem virtualnom okruženju. Koristit će se dva virtualna stroja (eng. *virtual machine*), gdje svaki virtualni stroj simulira ponašanje robota te se tako ostvaruje komunikacija između njih. Za konkretan slučaj koristit će se UR roboti (eng. *universal robots*) odnosno aplikacija za njihovo programiranje Polyscope koja se može opisati kao sučelje za robota koji se programira preko ekrana na dodir. Na slici [Slika 3.1] može se vidjeti programiranje robota u industriji putem sučelja. [15]

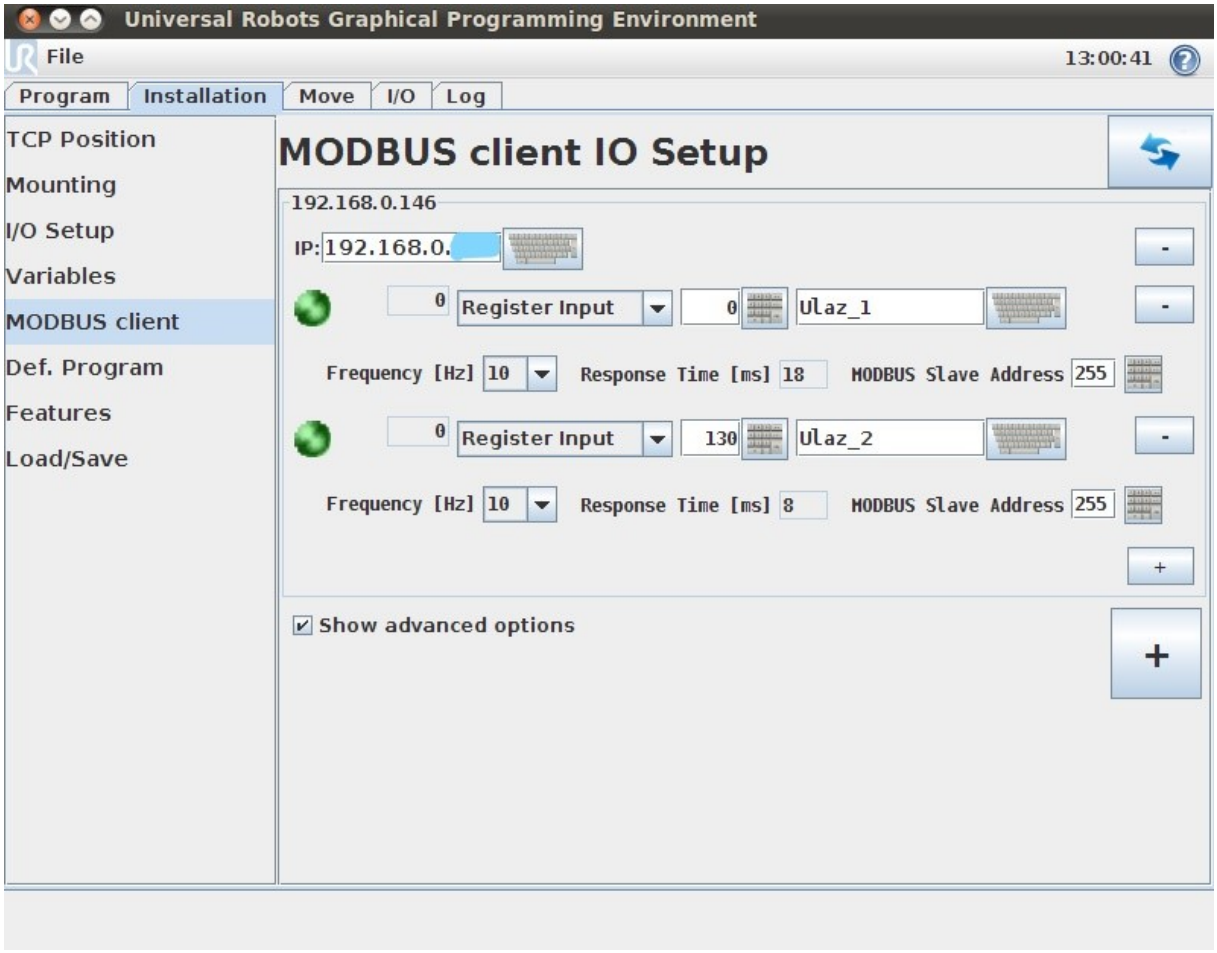
Slika 3.1 Prikaz UR robota i njegovog sučelja za programiranje

Prvi korak prilikom povezivanja dva robota pomoću Modbus protokola je saznati IP adrese od oba robota preko kojih će se izmjenjivati podatci. Kao što je već objašnjeno u prethodnim poglavljima, jedan robot će slati podatke i on se naziva *master* uređaj. Robot koji će primati informacije naziva se *slave* uređaj. Pomoću naredbe *write_port_register* manipulira se vrijednostima registra *slave* uređaja. Odabran je registar pod brojem 130 zato što su upravo registri od 128 do 255 općeniti 16-bitni registri te je u teoriji mogla biti odabrana bilo koja druga adresa u tom rasponu. Ostali registri su rezervirani od strane proizvođača za specijalne namjene. Na slici [Slika 3.2] prikazana je potrebna konfiguracija master uređaja.



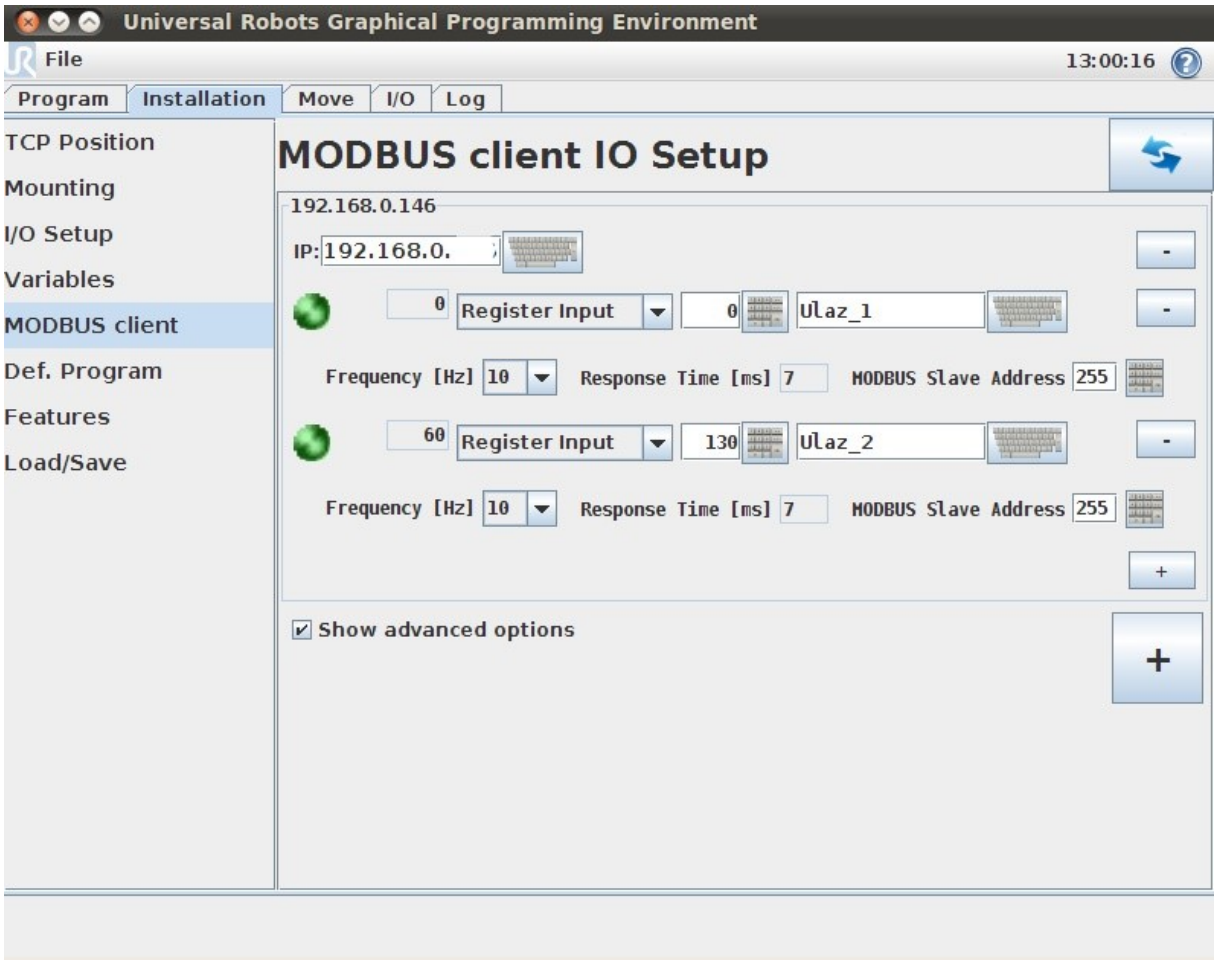
Slika 3.2 Konfiguracija *master* uređaja

Za konfiguraciju *slave* uređaja potrebno je izabrati MODBUS client iz izbornika. IP adresa koju je potrebno unijeti je upravo IP adresa od *master* uređaja tako da *slave* uređaj zna informacije o pošiljatelju. Može se primijetiti da je trenutna vrijednost registra pod nazivom Ulaz_2 jednaka nuli. Takva vrijednost definirana je od strane *master* uređaja. Na slici [Slika 3.3] možemo vidjeti početno stanje vrijednosti registara.



Slika 3.3 Primjer modbus komunikacije

Na sljedećoj slici [Slika 3.4] prikazat će se promjena vrijednosti registra pod nazivom Ulaz_2 koja je postavljena na vrijednost 60 od strane *master* uređaja. Promjenom vrijednosti registra potvrđuje se uspješna komunikacija između dva robota pomoću Modbus protokola. Trenutni program funkcionira u beskonačnoj petlji te će kao takav raditi neograničeno. [16]



Slika 3.4 Odziv modbus izlaza

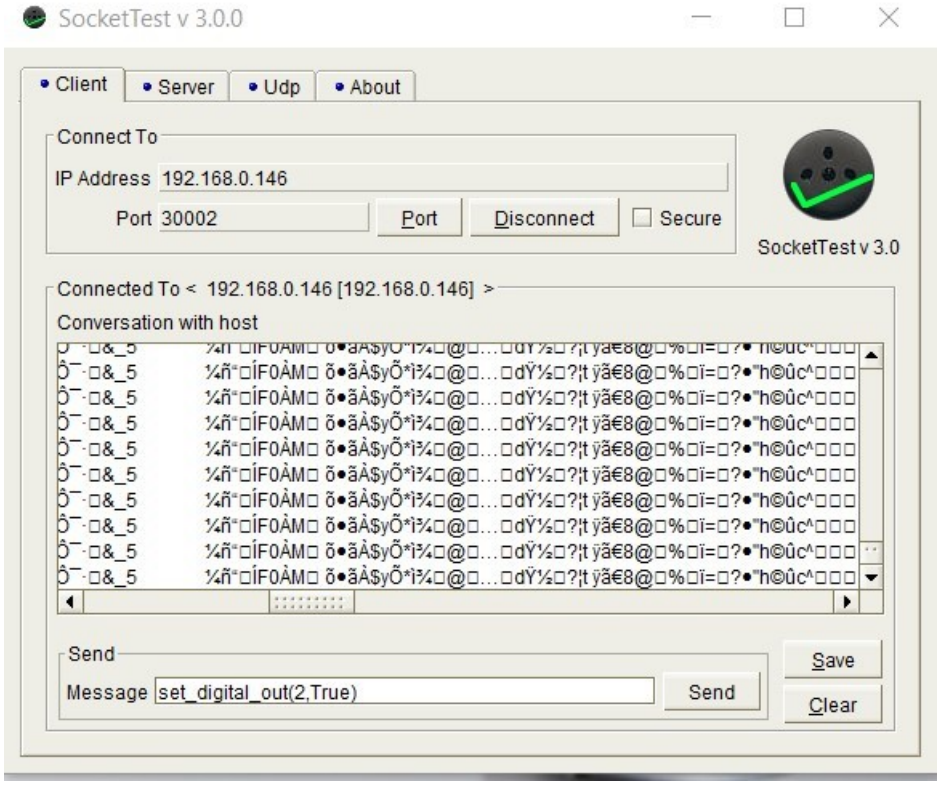
3.2 Testiranje socket mrežne tehnologije

U prethodnim poglavljima objašnjen je princip rada socket mrežne tehnologije, a sada će se on primijeniti na konkretnom primjeru. Prvi korak je testiranje *socket* komunikacije, odnosno odabir pravog mrežnog priključka (eng. *port*). Oni su definirani od strane proizvođača te mogu biti rezervirani za posebne namjene, slično kao što je to slučaj s adresama registara tijekom modbus komunikacije. Na slici [Slika 3.5] prikazani su neki od mrežnih priključaka (eng. *port*) koji će se koristiti za potrebe ovog zadatka. [17]

29998	Internally used
29999	Dashboard server
30001	Primary
30002	Secondary
30003	Real-time
30004	RTDE
30011	Primary read only
30012	Secondary read only

Slika 3.5 Prikaz mrežnih priključaka kod UR robota

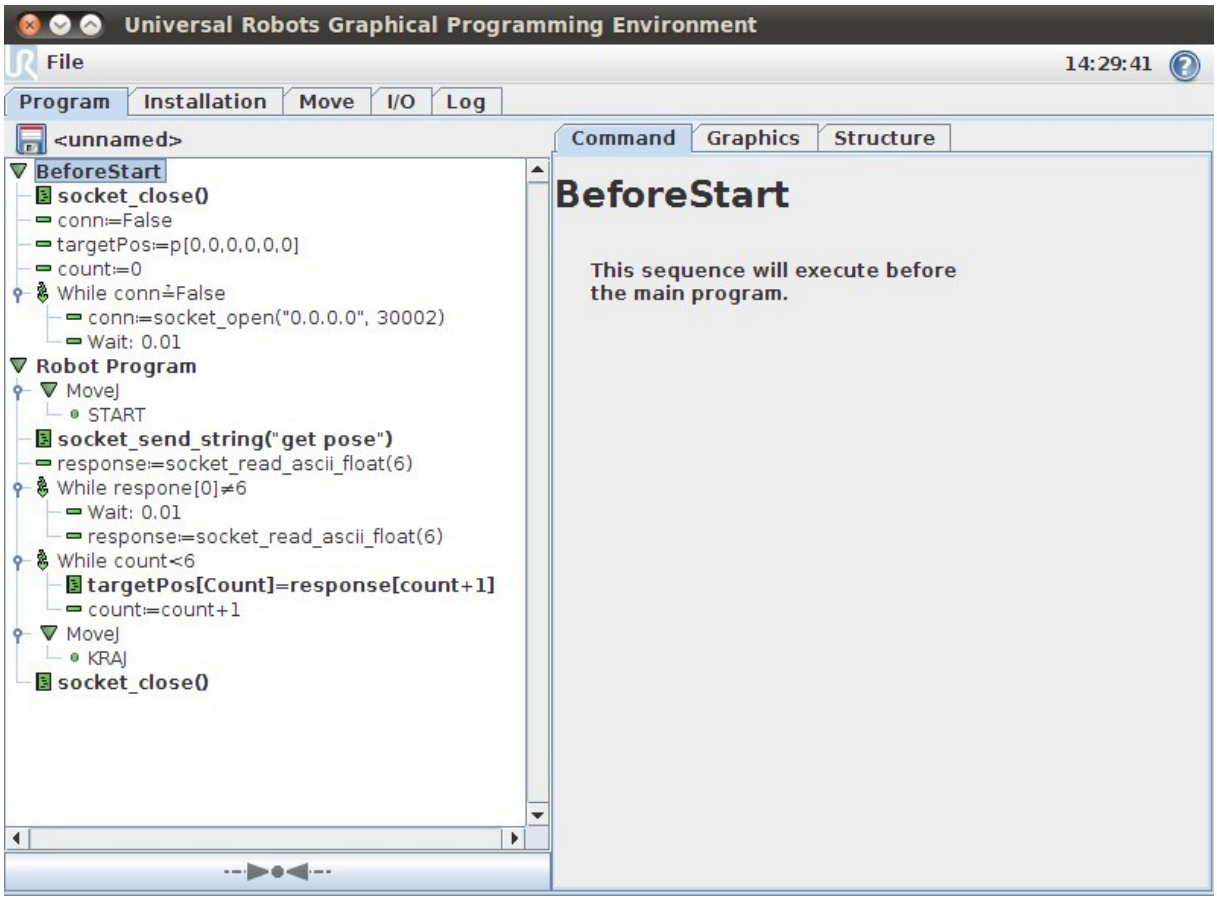
Za testiranje mrežnih priključaka koristit će se aplikacija pod nazivom SocketTest. [18] Često se koristi za testiranje samih mrežnih priključaka kod *socket* komunikacije. Prvi korak je odabir IP adrese robota kojemu se želi poslati naredba. Idući korak je odabir mrežnog priključka (eng. *port*). U konkretnom primjeru odabran je mrežni priključak 30002 jer se pomoću njega može manipulirati digitalnim izlazima. Naredba kojom će se digitalnom izlazu modificirati vrijednost glasi *set_digital_out(2,True).* Konfiguraciju programa i njegovo sučelje možemo vidjeti na slici. [Slika 3.6]



Slika 3.6 Konfiguracija SocketTest-a

Robot koji je korišten u odabranom primjeru ima 8 digitalnih izlaza te dva analogna strujna izlaza. Nakon slanja poruke digitalni izlaz pod rednim brojem dva se uključuje i ostaje aktivan sve dok se na robot ne pošalje takva naredba koja bi omogućila njegovo isključenje. Odabrani UR robot konkretno posjeduje i dodatna dva digitalna izlaza pod brojem 8 i 9 koji imaju specijalne namjene, odnosno koriste se za upravljanje alatom robota. Digitalni izlazi se općenito mogu koristiti u razne namjene, primjerice, mogu predstavljati kapacitivni senzor koji manifestira dolazak predmeta na kraj pokretne trake. Aktiviranje takvog digitalnog izlaza, može biti inicijator za pokretanje robota na zadanu poziciju i njegovo paletiziranje. S druge strane bitno je napomenuti da postoje i analogni izlazi koji se koriste u nešto drukčijim uvjetima. Budući da njihova vrijednost oscilira (konkretno za analogni senzor sa strujnim izlazom vrijednosti su između 4 i 20 miliampera) takvi senzori se najčešće koriste za mjerenje udaljenosti ili sile. [19]

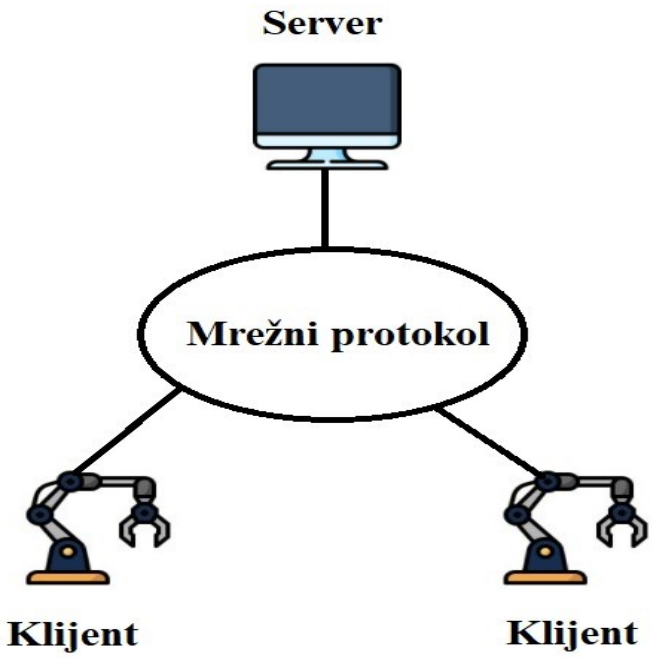
Postoji mogućnost manipuliranja i sa zglobovima robota odnosno njihovo gibanje u odnosu na zadani koordinatni sustav. Princip rada je vrlo sličan, sekvenca pod nazivom *Before Start* zatvara pristupnu točku (eng. *socket*) ukoliko se to iz nekog razloga nije desilo u prijašnjoj komunikaciji. Sljedeći korak je kreiranje varijabli te će *socket_open* aktivirati u trenutku kada se uspostavi komunikacija. Naredba *socket_read_ascii_float* zahtijeva niz od 6 varijabli, svaka od njih definira translaciju i rotaciju oko x,y i z osi. Na slici [Slika 3.7] je prikazan primjer takvog jednostavnog programa koji koristi socket komunikaciju i manipulira robotom između dvije zadane točke.



Slika 3.7 Upravljanje pozicijama pomoću socket komunikacije

3.3 Povezivanje dva robota preko sigurnog komunikacijskog servera

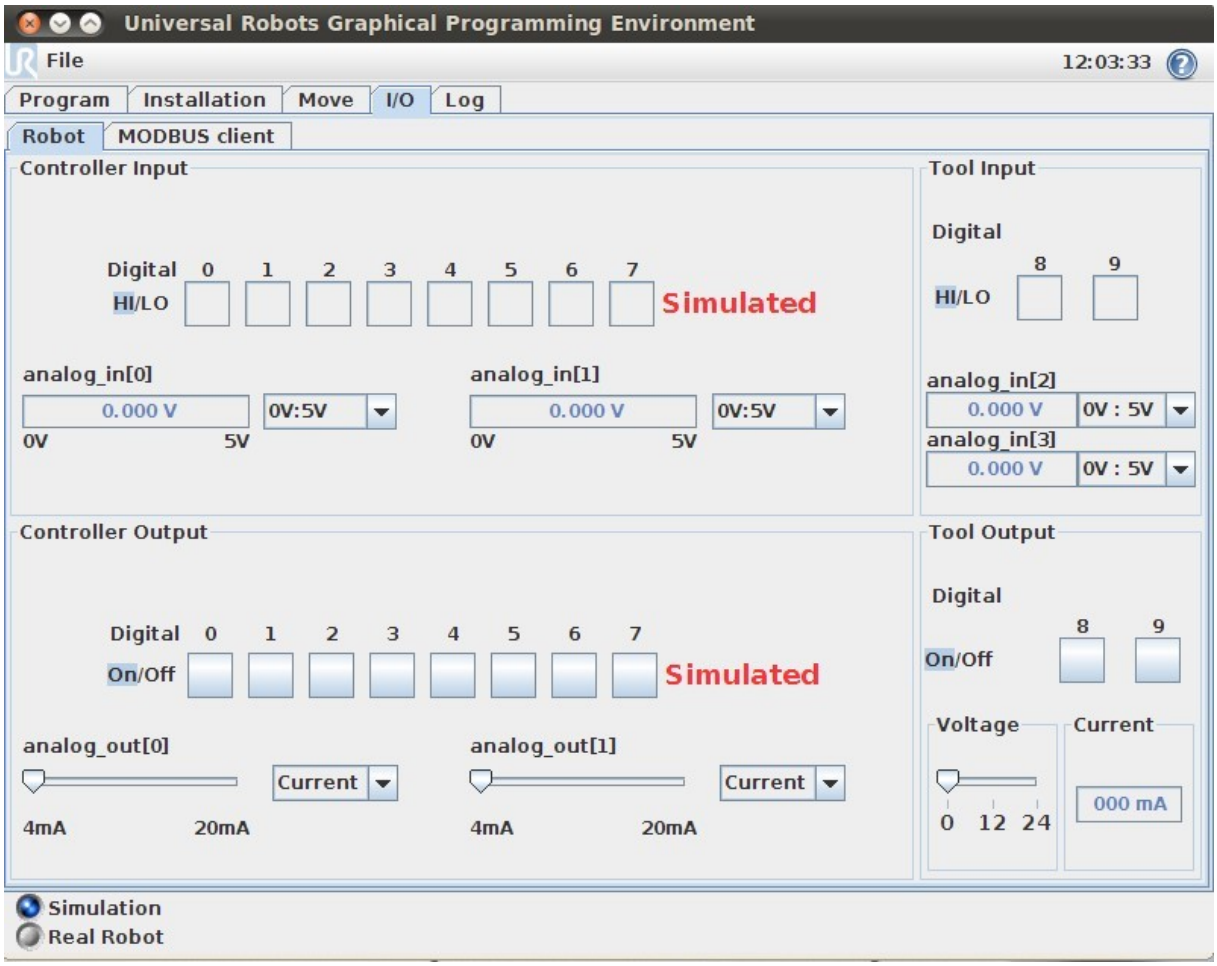
Pokazan je princip rada socket mrežne tehnologije te njegova manipulacija digitalnim izlazima. Naredni korak je izrada komunikacijskog servera koji će komunicirati između dva robota. Budući da smo u prošlim primjerima definirali IP adrese od oba robota one će se kao takve koristiti i sada. Prvi korak je definiranje servera i klijenta. Ovaj slučaj programiran je tako da su roboti zapravo klijenti, a računalo je server. Drugim riječima, računalo (server) je posrednik preko koje roboti (klijenti) izmjenjuju informacije. Shematski prikaz rada između servera i klijenta možemo vidjeti na slici. [Slika 3.8] [20]



Slika 3.8 Shematski prikaz servera i klijenta

Pomoću računalnog programa Python kreiran je komunikacijski server preko kojega će se odvijati sama izmjena podataka. [21] Cilj je omogućiti brzu i pouzdanu komunikaciju. Budući da se u specifičnom primjeru ne prenose velike količine podataka proces komunikacije će se odvijati praktički istovremeno. [22]

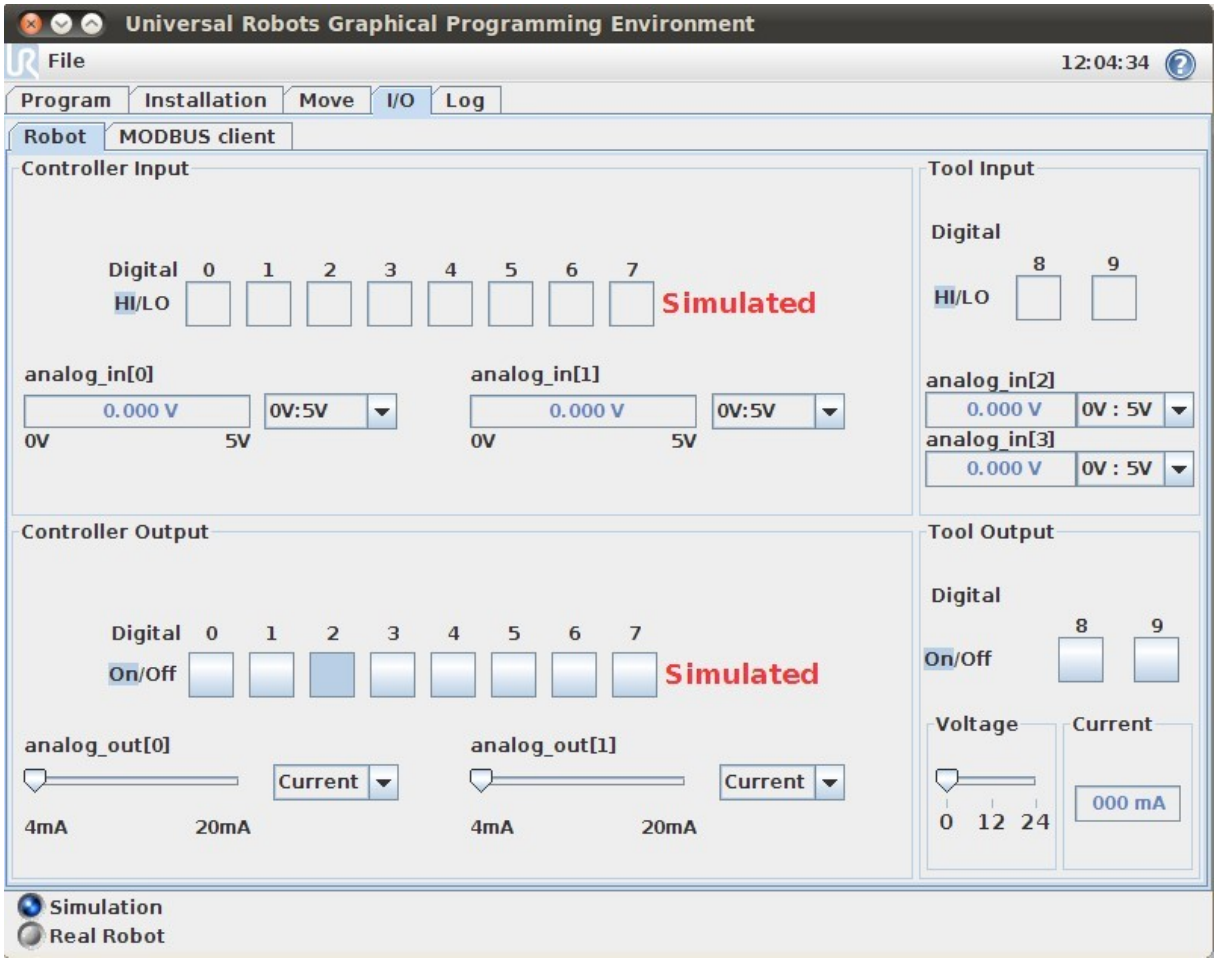
Ako postoji potreba za izmjenom velike količine podataka tada treba obratiti pažnju na konfiguraciju računala te brzinu internet mreže. Na sljedećoj slici [Slika 3.9] je prikazano početno stanje digitalnih.



Slika 3.9 Početno stanje digitalnih izlaza

IP adrese s kojima se treba ostvariti komunikacija spremljene su u *string* listu koja se može opisati kao lista svih IP adresa kojima je dozvoljena komunikacija. Zatim se kreira pristupna točka preko koje će roboti komunicirati. Naredni korak je povezivanje robota na odgovarajući *port* koji je definiran ranije.

Nakon povezivanja robota, šalje se zadana informacija koja u ovom konkretnom slučaju glasi aktiviraj digitalni izlaz dva. Stanje izlaza možemo vidjeti na slici. [Slika 3.10]



Slika 3.10 Konačno stanje digitalnih izlaza

Jasno je vidljivo da je digitalni izlaz dva sada aktiviran, odnosno u logičkoj jedinici. U tom periodu šalje se informacija o zaprimanju poruke te se pristupna točka (eng. *socket*) zatvara. Bitno je naglasiti da iako se pristupna točka zatvara, digitalni izlaz ostaje aktiviran sve dok robot ne dobije drugu naredbu. [23]

Također je važno istaknuti da se digitalni izlazi ne smiju poistovjećivati s pristupnim točkama koje služe samo za komunikacijski proces. Iz svega navedenog može se zaključiti da je socket komunikacija prihvatljivije rješenje koje se može implementirati na razna područja s mogućnošću sigurnije komunikacije.

Iz python skripte koja se nalazi u prilogu vidljivo je da je deklarirana varijabla pod nazivom RACUNALO_3 koja se odnosi na traženo simulacijsko računalo. IP adresa računala se ne nalazi u varijabli MY_ROBOTS te se praktički već time ograničava pristup tom računalu na server. Ako se RACUNALO_3 ili bilo koje drugo računalo pokuša povezati na server to se neće dogoditi upravo zato što se za komunikaciju preko pristupnih točaka ona se prvo treba kreirati. Ukoliko server ne kreira pristupnu točku (eng. socket) za RACUNALO_3 komunikacija jednostavno neće ni početi, a samim time neće postojati ni prijenos informacija odnosno naredbi.

Dodatna razina zaštite moguća je korištenjem programskog paketa JSON [24] koji zapravo predstavlja bazu podataka za svaki robot. Najčešće se koristi u situacijama kada imamo veliki broj objekata odnosno mnogo podataka koje je potrebno raspodijeliti. Implementaciju možemo vidjeti na slici [Slika 3.11].

```
import json

roboti_string = """ # varijabla u koju se spremaju string podatci
{
    "roboti": [
        {
            "korisnicko ime: robot_1"
            "serijski broj: 32044-49584"
            "naziv proizvodaca: Universal Robots"
            "password: robot_12345"
        },
        {
            "korisnicko ime: robot_2"
            "serijski broj: 42584-14789"
            "naziv proizvodaca: Universal Robots"
            "password: robot_98765"
        }
    ]
}"""

data = json.loads(roboti_string) # konverzija u python objekt
```

Slika 3.11 Primjer JSON baze podataka

Na slici [Slika 3.11] se također može vidjeti da se svakom robotu može dodijeliti određeni set podataka. U ovom slučaju definirano je nekoliko vrijednosti, korisničko ime robota, serijski broj, naziv proizvođača i jedinstvena šifra. Ukoliko računalo koje se pokušava povezati ne zadovoljava parametre njegov zahtjev će jednostavno biti odbijen. [25]

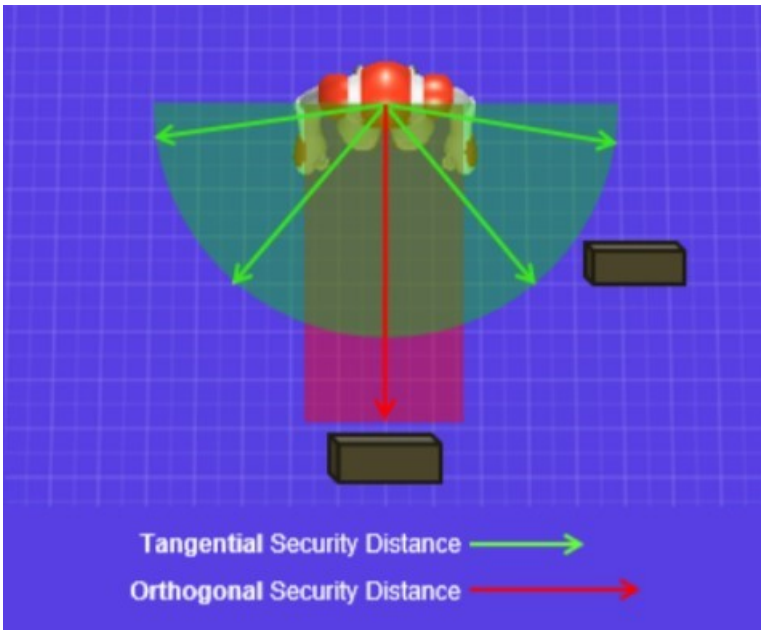
3.5 Posljedice povezivanja trećeg računala na komunikacijski proces

Robot koji prima neželjene naredbe može imati štetne posljedice za organizaciju ili industriju u kojoj se nalazi. Neprimjerene aktivnosti nužno sanirati što je prije moguće kako bi se spriječio neželjeni tog događaja. Objasnit ćemo nekoliko sigurnosnih propusta na različitim robotima.

Robot marke Alpha 1S ima mogućnost komuniciranja preko Bluetooth-a. Moguće se spojiti na robot pomoću mobilnog uređaja i s njime upravljati njegovim funkcijama. Glavni nedostatak takvog načina rada je nekorištenje sigurnosne šifre (PIN) prilikom spajanja što pruža mogućnost bilo kojem uređaju koji se nalazi u dometu uspješno povezivanje.

Sljedeći primjer je robot modela Baxter SDK/RSDK Shell. Ovisno o načinu izrade ovakav robot se može koristiti u industriji ili u istraživačkim radovima. Za uspješnu komunikaciju, a time i upravljanje dovoljno je povezati se na istu mrežu na kojoj se nalazi i robot, što ne predstavlja problem jer sustav ne koristi autentifikaciju. Kada se računalo uspješno poveže na mrežu moguće je upravljati i kontrolirati digitalne izlaze/ulaze, senzore i gibanje robota. Postoji mogućnost ukidanja opcije za detekciju kolizije što znači da robot više neće biti u stanju uočiti koliziju i upozoriti korisnika na eventualni sudar.

Slična situacija kao kod Baxter SDK/RSDK Shell robota može se pojaviti i kod NAO/Pepper robota. Na daljinu je moguće isključiti opciju za detektiranje kolizije ali i za samokoliziju. Glavni cilj isključivanja takvih opcija je oštećenje robota, okruženja ali i ljudi koji se nalaze u blizini. Na slici [Slika 3.12] možemo vidjeti zaštitni mehanizam protiv kolizije u slučaju kada on nije isključen. [26]



Slika 3.12 Sigurnosni mehanizmi Pepper robota

Vidljivo je da NAO/Pepper ima prilično dobro razvijene mehanizme protiv kolizije. Kada se gleda u tangencijalnom smjeru, tada ima manje područje pokrivenosti ali znatno veći kut koji pokriva dok u ortogonalnom smjeru pokriva veće udaljenosti ali i znatno manji kut. Ako dođe do isključenja spomenutih sigurnosnih funkcija zasigurno će doći i do kolizije.

Roboti koji su povezani preko *cloud* servera sačinjavaju veliku bazu robota koja se može sastojati i od nekoliko tisuća robota. Pod uvjetom da dođe do sigurnosnih propusta, korisnik može istovremeno upravljati sa svim robotima koji su povezani na mreži te tako prouzročiti velike materijalne štete. Primjer takvih robota može se naći u topologiji, konkretno robot Kuratas kojeg proizvodi Japanska tvrtka Suidobashi. [26]

4. ZAKLJUČAK

U ovome radu prezentirana su dva osnovna tipa komunikacije između robota, Modbus protokol i komunikacija pomoću socketa. Opisan je teorijski princip rada svake metode iz koje je donesen zaključak za prihvatljivije i samim time i sigurnije rješenje. Iako je vidljivo da Modbus protokol ima svoje prednosti, socket mrežna tehnologija je daleko prihvatljivije rješenje osobito ako je prvi kriterij sigurnost komunikacije.

Sljedeći korak je povezati objekte interakcije, u ovom slučaju robote i uspostaviti komunikaciju. Za to su korištene virtualne replike odnosno virtualni strojevi koji imaju identične karakteristike kao i realni fizički roboti. Implementirana je sigurna komunikacija preko python servera te se trećem računalu onemogućuje povezivanje i njegovo moguće blokiranje komunikacije. Daljnja nadograđivanja na ovu temu su svakako moguća, primjerice korištenje već spomenutog JSON programskog paketa može dodatno povećati sigurnost. Također mogu se implementirati rješenja poput sigurne stijene (eng. *firewall*) ili blockchain tehnologije koja može pronaći svoju svrhu u ovom području.

LITERATURA

[1] https://www.modbus.org/faq.php , dostupno dana 07. siječnja 2022.

[2] https://www.ni.com/en-rs/innovations/white-papers/14/the-modbus-protocol-in-depth.html , dostupno dana 07. siječnja 2022.

[3] https://www.dpstele.com/modbus/index.php , dostupno dana 07. siječnja 2022.

[4] https://ozeki.hu/p_5855-modbus-ascii.html , dostupno dana 07. siječnja 2022.

[5] https://www.modbustools.com/modbus.html , dostupno dana 08. siječnja 2022.

[6] https://realpars.com/modbus/ , dostupno dana 08. siječnja 2022.

[7] https://www.prosoft-technology.com/kb/assets/intro_modbustcp.pdf , dostupno dana 09. siječnja 2022.

[8] https://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf , dostupno dana 13. siječnja 2022.

[9] https://www.geeksforgeeks.org/socket-programming-cc/ , dostupno dana 10. veljače 2022.

[10] https://www.tutorialspoint.com/unix_sockets/client_server_model.htm , dostupno dana 10. veljače 2022.

[11] https://themozak.blogspot.com/2018/09/tcpip-tcp-ip-protokoli-model.html , dostupno dana 12. veljače 2022.

[12] https://www.wpbeginner.com/glossary/ip-address/ , dostupno dana 13. veljače 2022.

[13] https://www.dpstele.com/modbus/index.php , dostupno dana 13. veljače 2022.

[14] https://dev.to/sadarshannaiynar/blockchain-using-nodejs-and-socketio-5gbe , dostupno dana 14. veljače 2022.

[15] https://s3-eu-west-1.amazonaws.com/ur-support-site/77195/99405_UR10e_User_Manual_en_Global.pdf , dostupno dana 15. veljače 2022.

[16] https://www.zacobria.com/universal-robots-knowledge-base-tech-support-forum-hints-tips-cb2-cb3/index.php/modbus-internal-registers-read-and-write/ , dostupno dana 15. veljače 2022.

[17] https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/ , dostupno dana 15. veljače 2022.

[18] https://sourceforge.net/projects/sockettest/, dostupno dana 17. veljače 2022.

[19] https://www.universal-robots.com/articles/ur/interface-communication/connecting-internal-inputs-and-outputs-io-on-the-robots-controller/ ,dostupno dana 17. veljače 2022.

[20] https://medium.com/codex/make-a-client-talk-to-a-local-server-with-python-socket-programming-1-9be3cb4b474, dostupno dana 17. veljače 2022.

[21] https://realpython.com/python-sockets/, dostupno dana 18. veljače 2022.

[22] https://www.geeksforgeeks.org/socket-programming-python/, dostupno dana 18. veljače 2022.

[23] https://www.zacobria.com/universal-robots-knowledge-base-tech-support-forum-hints-tips-cb2-cb3/index.php/ur-script-send-commands-from-host-pc-to-robot-via-socket-connection/, dostupno dana 18. veljače 2022.

[24] https://www.w3schools.com/python/python_json.asp, dostupno 18. veljače 2022.

[25] https://realpython.com/python-json/, dostupno dana 18. veljače 2022.

[26] https://ioactive.com/pdfs/Hacking-Robots-Before-Skynet-Technical-Appendix.pdf, dostupno dana 18. veljače 2022.

PRILOZI

I. CD-R disc

II. Python kod

II. Python kod

```python
import socket

ROBOT_1_IP = "192.168.0.146"
ROBOT_2_IP = "192.168.0.117"
RACUNALO_3 = "192.168.78.1"
MY_ROBOTS = [ROBOT_1_IP, ROBOT_2_IP]
PORT = 30002  # port na kojem se odvija komunikacija


def connect_to_robot(my_socket, ip, port):  # spajanje robota preko socketa
    my_socket.connect((ip, port))


def send_message_to_robot(my_socket, message):
    my_socket.send(message.encode("utf-8"))  # poruka poslana


def receive_response(my_socket):  # primanje odgovora od robota
    response = my_socket.recv(1024)
    print(response)
```

```python
def main():
    message = input("set_digital_out(2,True)" + "\n")

    for robot_ip in MY_ROBOTS:
        my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  # kreiranje socketa
        connect_to_robot(my_socket, robot_ip, PORT)
        send_message_to_robot(my_socket, message)
        receive_response(my_socket)
        my_socket.close()


if __name__ == "__main__":  # prvo se otvara main dio
    main()
```