Zahvalnica

Ovim putem bih htio zahvaliti mentorici, izv. prof. dr. sc. Ani Sović Kržić, koja me je mentorirala većinu mog boravka na fakultetu. Zahvalan sam na njenom strpljenju te svim prilikama koje mi je pružila tijekom godina studija.

Htio bih se zahvaliti kolegama Ivanu Brčiću, Marku Tomiću i Jani Alajbeg koji su ranije radili na robotu TIOSS te su mi pripomogli kada nisam mogao nešto razumjeti.

I za kraj htio bih se zahvaliti svojim bliskim prijateljima i obitelji koji su mi bili velika podrška i potpora za vrijeme mog boravka na fakultetu.

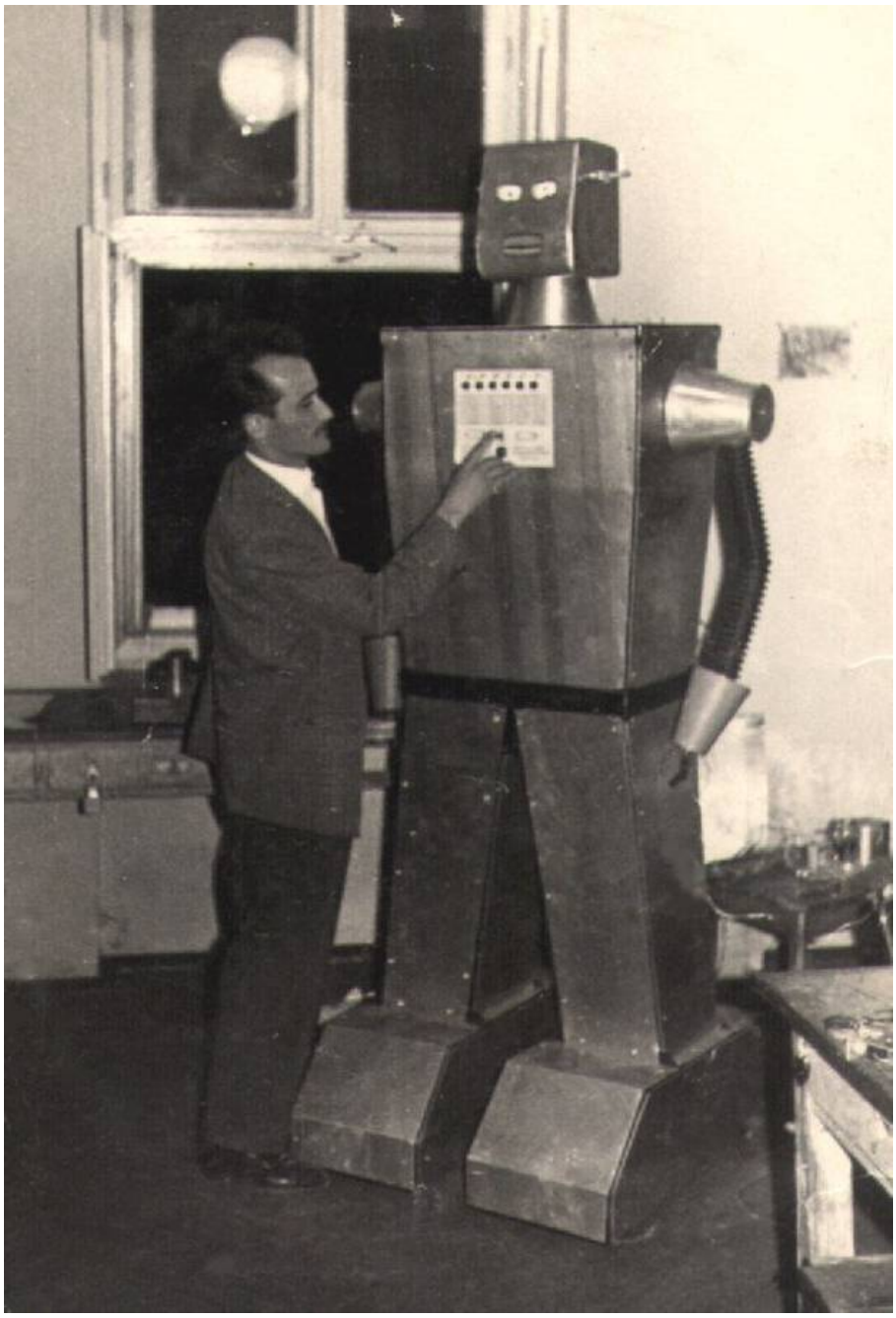
Sadržaj

Uvod

O robotskoj revoluciji se sanjalo još 60 godina prošloga stoljeća kada je, tada student elektrotehnike, Branimir Makanec sa svojim kolegama odlučio napraviti autonomnog robota koji samostalno misli i funkcionira bez daljinskog upravljanja. Zbog tadašnje tehnologije uspjeli su napraviti Teledirigirani izvršni organ samoorganizirajućeg sustava skraćeno TIOSS. TIOSS je humanoidni robot visok 220 cm i težak čak 150 kg. Iako nije mogao sam razmišljati mogao je kretati se te pomicati rukama, čak je mogao okretati glavu za najsvjetlijom točkom. Kasnije opremljen je i radio prijemnikom kako bi se njima moglo upravljati na daljinu. Koristio se za dijeljenje letaka, ali i za promociju znanosti. Kako se puno selio u jednom trenutku izgubio mu se trag. Nakon više od 20 godine 2020. godine pronađen je u brizi kluba mladih tehničara Dubrava. Kada ga je gospodin Makanec pronašao nije bio u dobrom stanju, sve unutarnje komponente su bile izvađene van jedino što je ostalo u njemu su bili motori. Zatim gospodin Makanec je darovao robota TIOSS izv. prof. dr. sc. Ani Sović Kržić koja je pokrenula projekt restauracije kako bi još jednom služio za promociju znanosti. TIOSS je prošao već dvije restauracije kako bi se osposobili njegovi motori i sad je preostalo omogućiti udaljeno upravljanje robotom. Za upravljanje robota odabrano je mikroračunalo VIDI X koji je novi Hrvatski proizvod. Robot i kontroler će komunicirati s pomoću bežične mreže. Ideja sustava je da s pomoću mobitela može postaviti mreža na koju će se spojiti TIOSS i VIDI X. Zatim TIOSS će se javiti svim uređajima u mreži te VIDI X, prepoznavši da se radi o TIOSS-u, spaja se na njega.

1. Robot

Prvi robot napravljen na području bivše Jugoslavije je teledirigirani izvršni organ samoorganizirajućeg sustava ili skraćeno TIOSS. Tada student elektrotehnike Branimir Makanec sa svojim prijateljima je organizirao Grupu kibernetičara i po njegovom nacrtu izrađuju robota [1]. Projekt je počeo 1958. godine, a završen tek 1961. Cilj projekta je bio stvoriti robota koji je potpuno samostalan, no, na žalost, samo su djelomično bili uspješni. Razlog za to vjerojatno leži u činjenici da su dijelove za TIOSS-a skupljali na smetlištima, jer su tada smetlišta bila puna američkih aviona iz Drugog svjetskog rata. Te dijelove su objedinili u metalno kućište koje je izgledalo kako humanoidni robot visok 220 cm i težak 150 kg. Na Slici 1 može se vidjeti njegova stvarna veličina. Kako bi mogao hodati u svaku nogu su mu stavljeni jaki motori. Kretao se tako da je pomicao jednu po jednu nogu i tako simulirao hodanje. Tako hodajući mogao je postići brzinu od 30 km/h. U lijevoj nozi nalazio se punjač akumulatora, a u desnoj se nalazila logika kojom je robot upravljan. Kasnije su omogućili da se robotom upravlja i s pomoću radio valova. Na prsima je imao mikrofon, a u glavi su smješteni zvučnik i dva optička senzora. Mikrofon i zvučnik su omogućili TIOSS-u da razgovara s prolaznicima, odnosno tko god da je uzeo mikrofon sakrio bi se da ga prolaznici ne vide te razgovarao s njima kroz TIOSS-a. Optički senzori su djelovali kao oči s pomoću kojih je, zahvaljujući glavi koja se okreće, mogao pratiti izvor jake svjetlosti. Ruke nisu služile samo za ukras. Imale su hvataljke koje su omogućavale dijeljenje letaka. Robot je postao velika senzacija kada je bio dovršen. Moglo ga se vidjeti na Velesajmu dok dijeli letke, čak je 1968. godine bio na glavnom trgu grada Zagreba. Koristio se i u edukacijske svrhe na fakultetu, ali ga je zbog veličine bilo teško skladištiti. Selili su ga po raznim ustanovama koje su ga izlagale u predvorjima, no dijelovi su s vremenom pokradeni i izgubio mu se svaki trag. Međutim, robot je nedavno pronađen u Klubu mladih tehničara Dubrava koji su pokušali rekonstruirati robota, nažalost neuspješno [2]. Gospodin Branimir Makanec je potom darovao robota izv. prof. dr. sc. Ani Sović Kržić koja je pokrenula proces rekonstrukcije i modernizacije TIOSS-a kako bi opet mogao fascinirati ljude. TIOSS je prošao dva vala restauracije. U prvom su kolege s Fakulteta elektrotehnike i računalstva Marko Tomić i Jana Alajbeg popravili glavu i noge robotu, dok je kolega Ivan Brčić osposobio ruke i hvataljke robota.



Slika 1. Originalni izgled robota[2]

1.1. Prva restauracija

Kolege Marko Tomić i Jana Alajbeg su prvi vidjeli robota nakon njegovog pronalaska. Cilj kolege Marka Tomića je bio osposobiti kretanje robota [3], dok je kolegica Jana Alajbeg restaurirala glavu i njenu funkcionalnost praćenja svjetla [4]. U početku robot nije imao ništa od originalne funkcionalnosti, ali je imao originalne dijelove. Motori su još bili u robotu, ali nisu bili pričvršćeni na nosače, a motor za funkcionalnost glave je bio

pričvršćen za trup, ali nije bio spojen s glavom. Oba motora nogu su bila funkcionalna pa ih se odlučilo očistiti te ponovo ugraditi u robota. Dijelovi motora su elektromotor i pužni prijenos. Spajanjem elektromotora na pužni prijenos uspori se maksimalna brzina, ali se dobije značajno veća snaga. U postupku restauracije bilo je potrebno popraviti zupčanike u pužnom prijenosu jer je nedostajalo nekoliko zuba u zupčaniku što je dovodilo do škljocanja. Popravkom zupčanika škljocanje je nestalo. Elektromotori imaju dva navoja što ukazuje da motor ima dvije brzine. Kolega Tomić je odlučio koristiti manju brzinu motora što rezultira većom snagom, ali isto tako i sporijim kretanjem robota i time sigurnijim upravljanjem samim robotom. I motor glave je građen na isti način, sastoji se od elektromotora i mehaničkog dijela. Mehanički dio čine zupčanik i poluga koja se kreće radijalno kako bi okretala glavu. Dio žica elektromotora je bio pogrešno spojen što je onemogućavalo rad motora pa je i to trebalo prepraviti. Nakon prepravka motor je bio funkcionalan. Na motoru glave postoje graničnici koji mehanički isključe motor kada poluga dođe u krajnji položaj. U robota su dodana i dva akumulatora od 12 V koji su spojeni serijski kako bi napajali robota. One se pune s pomoću punjača koji je smješten u trup robota i spojen izravno na akumulatore. Upravljanje motorima omogućeno je s pomoću releja spojenih u H-most. Za „mozak“ robota izabran je Raspberry Pi 4 Model B [6]. To je odličan odabir jer ima 26 GPIO (eng. General Purpose Input / Output) koji omogućuju spajanje velikog broja dodatnih sklopova ili komponenata. Međutim, tu nije riječ o mikrokontroleru na kojeg se programira određeni program, nego se radi o mikrokontroleru koji ima svoj operacijski sustav Raspberry Pi OS. Raspberry Pi OS je inačica Linux operativnog sustava. Pošto Raspberry PI ima svoj operacijsku sustav možete izabrati proizvoljan programski jezik za izradu upravljačkog koda. Lakše upravljanje robotom omogućava i mogućnost bežičnog spajanja na mreže. Prvom restauracijom se postiglo vraćanje funkcije kretanja robota te mogućnosti robota da prati jaku svjetlost. Na žalost, daljnjim korištenjem robota oštećeni su zupčanici u motorima nogu što ga je ostavilo nepokretnim.

1.2. Druga restauracija

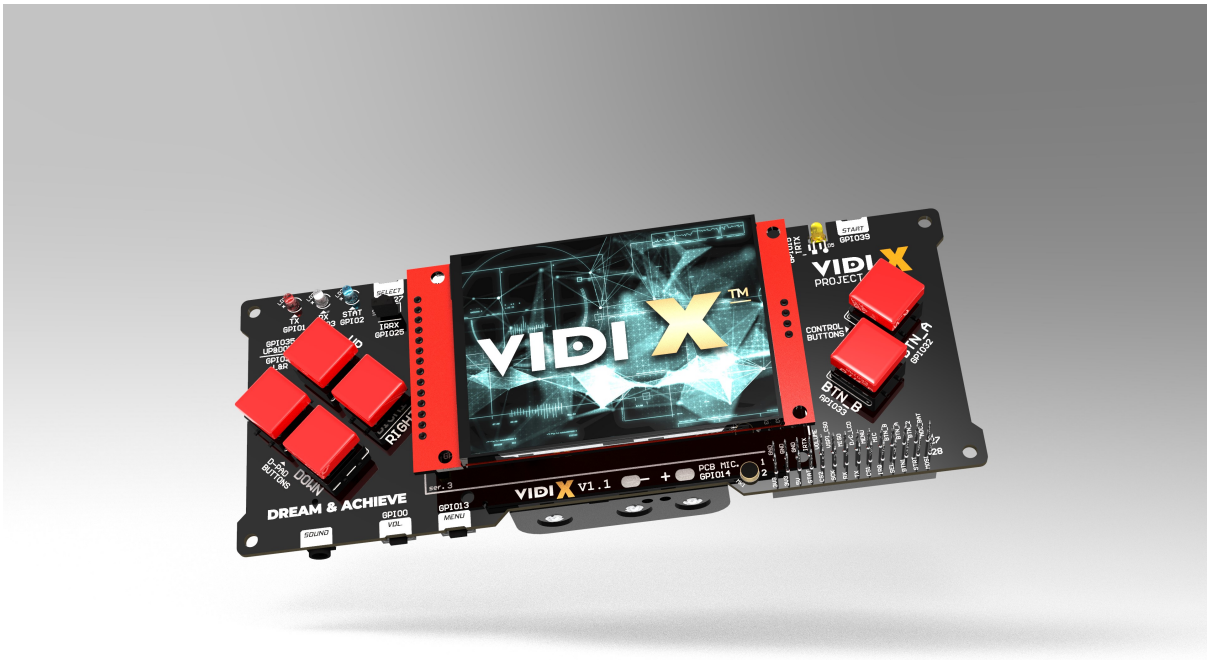
Cilj kolege Ivana Brčića bio je osposobiti ruke i hvataljke robota kako bi se njima moglo upravljati [5]. Motori ruku i hvataljki nisu bili u najbolje stanju. Najprije ih je trebalo izvaditi iz robota i rastaviti. Bili su stari i vidno zapušteni, a četkice motora su bile posve pohabane. Neki dijelovi motora su morali biti zamijenjeni, sve je trebalo dobro podmazati i

ponovno sastaviti. Motori su vraćeni u ruke robota u kojima je su ugrađeni mehanički prekidači koji služe sprečavanju mehaničkih kvarova ruku tako da isključuče motore kada se ruke nađu u krajnjem položaju. Motori hvataljki su bili u boljem stanju pa tu nije bilo toliko puno posla. Hvataljke koriste solenoidne motore koji se otvaraju, odnosno zatvaraju prolaskom struje. Tijekom ove restauracije došlo je do manjih oštećenja motora glave pa je i motor glave uključen u ovaj rad te spojen na upravljački sklop koji je kolega Brčić napravio. Motorima ruku i glave se upravlja tako da se serijski spoji rotor i namot, a smjer vrtnje je određen odabirom namota. Motorima hvataljke se upravlja dovođenjem napona na priključke motora, koja onda otvori, odnosno zatvori ruke. Kako bi mogao pratiti koliko su se podigle ili spustile ruke, odnosno koliko se okrenula glava, kolega Brčić je dodao senzore položaja na motore. Samo upravljanje motorima i očitavanje senzorima je riješeno uz pomoć novog sklopa koji je kolega Brčić izradio te spojio na Raspberry Pi 5.

Slika 1.2. TIOSS nakon druge restauracije [5]

2. Kontroler

Za udaljeno upravljanje robotom postoji više mogućnosti, ali za ovaj rad je odabrano VIDI X mikroračunalo [7]. To mikroračunalo je odabrano prvenstveno zbog toga što je riječ o domaćem proizvodu. VIDI X je rezultat Projekta X tehničkog časopisa VIDI. U cijelosti je osmišljen, projektiran i sastavljen u Hrvatskoj. Jezgra mikroračunala je ESP32 modul s dvije 32-bitne jezgre brzine do 2.5 GHz. Sam modul dolazi s WiFi i bluetooth podrškom te 8 MB flash memorije [8]. Cilj VIDI Projekta X je napraviti platformu sa što više ulazno – izlaznih jedinica na pločici. Zbog toga VIDI X ima 10 tipki, 4 LED diode, 320 × 240 ekran osjetljiv na dodir, zvučnik, mikrofon, infracrveni prijamnik i predajnik, te senzor temperature. VIDI X ima i dodatne pinove za proširenje što omogućava spajanje dodatnih, vlastitih senzora. Ta je funkcionalnost dobro došla u izradi ovog rada jer je omogućilo da svaka akcija ima svoju tipku. Ekran je bio jako koristan za ispisivanje uputa robotu te signaliziranje stanja izvođenja programa.



Slika 2. VIDI X mikroračunalo [7]

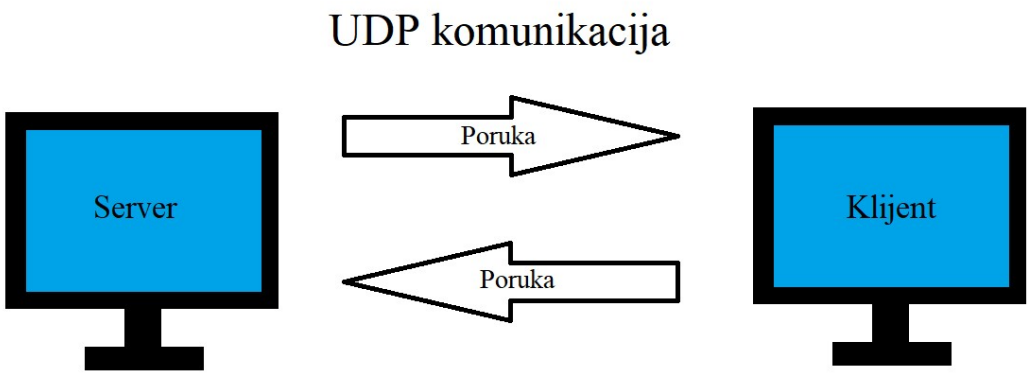
3. Komunikacija

Kako bi se ostvarila komunikacija između kontrolera i robota, odnosno između VIDI X-a i TIOSS-ovog Raspberry Pi potrebno je odabrati način komunikacije koji oba mikroračunala podržavaju. Kako oba mikroračunala podržavaju WiFi i Bluetooth izbor mora biti između ova dva načina komunikacije. WiFi ima dvije glavne prednosti, a to su brzina i domet. Brzina ovisi o WiFi modulima mikroračunala i usmjerivaču koji se koristi. Raspberry Pi podržava brzine do 1300 Mbps, a VIDI X podržava brzine do 300 Mbps. Maksimalna brzina za Bluetooth je 2 Mbps. WiFi ima značajno veću brzinu što može poslužiti za slanje zvuka i videa. Druga bitna prednost je domet. Bluetooth ima mali domet, do desetak metara, dok WiFi može, u teoriji, imati beskonačan domet uz dodavanje dodatnih pristupnih točki. Jedina prednost Bluetooth tehnologije u ovom slučaju je mala potrošnja energije što nije ključno u ovom projektu. Zbog veće brzine i dometa za ovaj rad je odabran WiFi kao način spajanja TIOSS-a i VIDI-X-a. Dodatni benefit korištenja WiFi-a je mogućnost spajanja TIOSS-a na Interrnet što može biti neophodno za neke buduće mogućnosti TIOSS-a, kao što je korištenje LLM-ova. Kako bi TIOSS i VIDI X mogli međusobno komunicirati moraju biti unutar iste mreže. Da bi se to osiguralo oba uređaja su pretprogramirana tako da se spoje na mrežu imena „TIOSLAN“ i unaprijed definirane zaporke. Kako je već veliki broj studenata radio na TIOSS-u ova metoda omogućuje da svatko preko svog mobitela kreira mrežu navedenih parametara bez dodatne intervencije na kodu VIDI X-a i bez potrebe direktnog spajanja na TIOSS-ov Raspberry Pi. WiFi podržava korištenje velikog broja protokola za prijenos informacija. U ovome radu se koriste samo dva osnovna, UDP i TCP.

3.1. UDP protokol

UDP je jedan od osnovnih komunikacijskih protokola mreža baziranih na Internet protokolu [9]. To je protokol transportnog sloja osi modela. Glavno obilježje mu je što ne zahtijeva prijašnju komunikaciju kako bi mogao slati pakete, odnosno ne zahtijeva uspostavu veze. To znači da samo može slati pakete na odredište bez saznanja da li odredište uopće postoji. Negativna strana toga je što ne garantira dolazak paketa na odredište, redoslijed paketa i duplikaciju. Međutim, baš zato što je jednostavan i nema

puno garancija jako je brz te je prikladan za primjene gdje je potrebna brzina ili gdje je moguće detektirati i ispraviti pogreške na razini aplikacije. Postoji mogućnost slanja paketa na sve adrese u mreži ili određenoj podgrupi računala u mreži. Zbog svoje jednostavnosti ima široku primjenu. Koristi se i u drugim protokolima kao što su DNS, DHCP, itd. Također se koristi za prijenos video i audio sadržaja u stvarnome vremenu. U sklopu ovoga rada koristi se zbog svoje brzine i jednostavnosti, ali isto tako i zbog svoje mogućnosti slanja paketa svim uređajima u mreži.

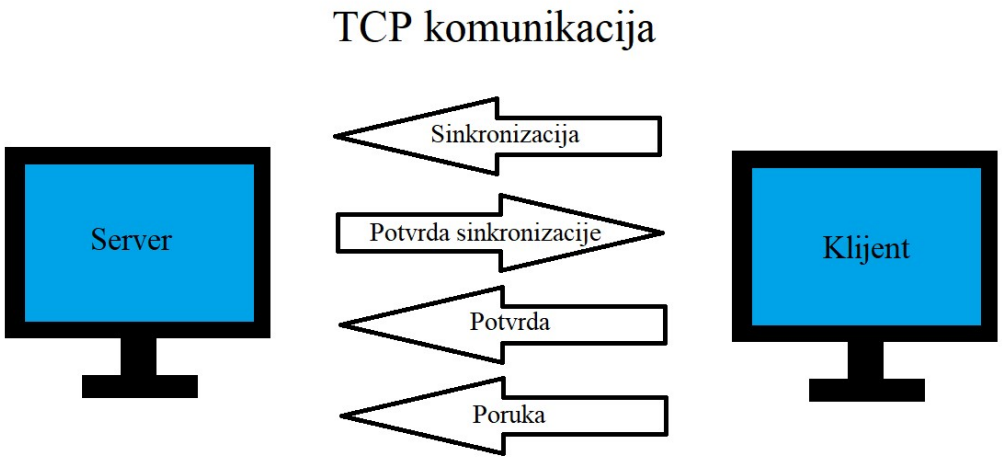


Slika 3.1 Dijagram UDP komunikacije

3.2. TCP protokol

TCP je jedan od glavnih protokola u skupini Internet protokola [10]. Isto kao i UDP i ovaj protokol radi na transportnom sloju OSI modela. Omogućuje dostavu paketa između korisnika, koja je pouzdana, uređena i otporna na greške. TCP zahtijeva uspostavu veze između primatelja i pošiljatelja paketa. Uspostavom veze može garantirati dostavu paketa, točan redoslijed paketa i točnost paketa. Zbog tih pozitivnih kvaliteta jako je popularan i ima mnoge primjene u velikim internetskim programima kao što su World Wide Web, e-mail, slanje datoteka i udaljeno upravljanje računalima. Tu pouzdanost ostvaruje zahtijevanjem prijašnje konekcije s odredištem. Postupak po kojem ostvaruje konekciju te šalje pakete naziva se eng. Three-way handshake. Komunikacija počinje sa serverom. Server osluškuje mrežu i čeka zahtjev za spajanje. Nakon što klijent pošalje zahtjev za spajanje na server, server odgovara klijentu da je primio njegov zahtjev te da prihvaća komunikaciju. Klijent nakon primanja potvrde komunikacije šalje pakete na server i nakon što server primi pakete šalje nazad klijentu potvrdu o primitku paketa. Ako u bilo kojem

trenutku u navedenom procesu jedna od očekivanih poruka ne dođe slijedi ponovno slanje poruke. Kao rezultat ovakvog načina komunikacije TCP može garantirati isporuku paketa na njihovo odredište. Negativna strana ovog procesa je što povećava kašnjenje paketa. TCP koristi i algoritme koji smanjuju brzinu prijenosa u zagušenim mrežama kako bi povećali pouzdanost. Iako TCP ima puno prednosti ima i svojih mana i slabih točki što ga čini podložnim raznim napadima kao što je zabrana usluge (eng. Denial of service) i presretanjem povezivanja (eng. Connection hijacking). U ovom radu TCP se koristi kao pouzdanija alternativa UDP protokolu u slučaju da se robotom ne može pouzdano upravljati.



Slika 3.2. Dijagram TCP komunikacije

4. Server

U sklopu ovog rada riječ server se odnosi na TIOSS-a, odnosno na Raspberry Pi računalo koje služi za upravljanje motorima robota. Spojeno je na motore glave i ruku s pomoću sklopa kojeg je izradio kolega Ivan Brčić za vrijeme druge restauracije robota [5]. Koristi se osam ulazno – izlaznih pinova na Raspberryju kako bi se upravljalo sklopom, odnosno motorima. Dva pina za svaki od motora koji okreću ruke i glavu i po jedan pin za solenoide hvataljki. Specifičnost sklopa na koju se treba osvrnuti je da koristi invertirajući naponski buffer, što znači da se dovođenjem niske naponske razine na sklop pokreću motori, a visoka naponska razina ih zaustavlja.

GPIO pin	*Uloga*
17	Pokretanje desne ruke prema gore.
27	Pokretanje desne ruke prema dolje.
23	Hvataljka desne ruke.
24	Pokretanje lijeve ruke prema gora.
22	Pokretanje desne ruke prema dolje.
26	Hvataljka lijeve ruke.
16	Pokretanje glave u desno.
25	Pokretanje glave u lijevo.

Tablica 4. Korišteni pinovi Raspbery Pia

Velika prednost Raspberry Pi računala je to što ima svoj operacijski sustav što omogućuje odabir bilo kojeg programskog jezika za izradu upravljačkog programa. Za ovaj rad je izabran Python [11]. Ovaj programski jezik je odabran zbog svoje jednostavnosti i bogate podrške biblioteka za veliki broj različitih primjena. U ovom radu izrađena su četiri programa koja se razlikuju u dvije kategorije, po protokolu koji se koristi za komunikaciju s kontrolerom i načinu upravljanja. Protokoli za komunikaciju su UDP i TCP koji su već objašnjeni u dijelu komunikacija, a način upravljanja može biti kontinuiran i diskretan.

Kod programa s diskretnom načinom upravljanja server dobije jednu naredbu koja mu govori s kojim motorom upravlja koliko dugo, dok kod programa s kontinuiranim načinom upravljanja server upravlja motorom samo ako dobiva poruke za upravljanje.

4.1. Opis biblioteka

Biblioteka os je jedna od standardnih biblioteka programskog jezika Python [12]. Omogućava manipulaciju operacijskim sustavom korištenjem programskog jezika Python. U ovom programu koristi se za dobavu mrežne adrese servera. To je jako važno jer omogućuje neovisnost o mreži, odnosno ne mora se ručno upisivati adresa u kod servera i kontrolera.

Biblioteka socket je također jedna od standardnih biblioteka programskog jezika Python [13]. Ova biblioteka omogućava kreiranje socket-a koji predstavljaju programsku strukturu za slanje i primanje poruka preko mreže. Dostupna je na svim modernim Unix, Windows i MacOs operacijskim sustavima.

Biblioteka time je još jedna od standardnih biblioteka programskog jezika Python [14]. Korištenje ove biblioteke omogućava dohvaćanje vremena i datuma. Koristi se u programima s kontinuiranim načinom upravljanja kao pomoć pri upravljanju motorima.

Biblioteka gpiod je Python verzija libgpoid biblioteke [15]. Napravljena je za korištenje ulazno – izlaznih pinova računala koja imaju Linux operacijski sustav. Ova biblioteka omogućuje inicijalizaciju i kontrolu nad ulazno – izlaznim pinovima Raspberry P 5 koji upravljaju TIOSS robotom.

4.2. Programski kod

Sve inačice programskog koda za server slijede isti dijagram. Na početku se nalazi uvoz biblioteka zatim se definiraju globalne varijable i zastavice, iza njih nalaze se sve potrebne funkcije te na kraju nalazi se glavni kod. Program počinje pronalaženjem mrežne adrese servera, zatim kreira potrebne sockete za komunikaciju nakon čega čeka spajanje kontrolera. Jednom kada uspostavi komunikaciju s kontrolerom inicijalizira ulazno izlazne pinove te vrti beskonačnu petlju u kojoj izvršava instrukcije s kontrolera.

4.2.1. Uvoz biblioteka

Uvoze se sve ranije opisane biblioteke kako bi ih se moglo koristiti u programu.

```
import socket
from time import time
import os
import gpiod
```

Dok se biblioteke socket, os i gpoid koriste u svim verzijama programa, biblioteka time se koristi samo u dijelu s kontinuiranim načinom upravljanja.

4.2.2. Definiranje globalnih varijabli i zastavica

Zatim se definiraju konstante i iniciraju zastavice potrebne za rad programa. Među njima su konstante koje definiraju broj socketa, adresu upravljačkog kontrolera za ulazno – izlazne priključnice te pinove na koje su određene funkcionalnosti robota spojene. U niže navedenom isječku nalaze se sve konstante i zastavice koje se koriste kroz sva četiri programa, no ne koriste se sve u svakom programu.

```
UDP_PORT = 5005
TCP_PORT = 5006
left_arm_hold = False
right_arm_hold = False
motor_is_running = False
CHIP_PATH = 'gpiochip4'
HEAD_RIGHT = 16
HEAD_LEFT = 25
HAND_RIGHT_UP = 17
HAND_RIGHT_DOWN = 27
HAND_LEFT_UP = 24
HAND_LEFT_DOWN = 22
HAND_RIGHT_HOLD = 23
HAND_LEFT_HOLD = 26
```

Konstanta UDP_PORT se koristi u svim programima, dok se konstanta TCP_PORT koristi samo u programima koji koriste TCP protokol za komunikaciju.

4.2.3. Funkcija obtainIpLinux()

Funkcija obtainIpLinux() služi za pronalaženje mrežne adrese servera.

```
def obtainIpLinux ():
```

```
gw = os.popen("ip -4 route show default").read().split()
return gw[8]
```

Korištenjem naredbe os biblioteke, os.popen() pošalje se naredba na komandnu liniju operacijskog sustava koja vraća podatke o mreži na koju je računalo spojeno u obliku dugačkog stringa. Kako bi se iz tog stringa izvukla mrežna adresa servera, naredbom .split() string se dijeli na mjestima gdje se nalazi razmak u polje stringova. Mrežna adresa servera se nalazi na devetoj poziciji tog polja što je povratna vrijednost funkcije.

4.2.4. Izrada brodcast adrese

Iz mrežne adrese servera potrebno je napraviti brodcast adresu kojom se adresiraju sva računala u mreži.

```
UDP_BRODCAST = IP_addres.split('.')
UDP_BRODCAST[-1] = '255'
UDP_BRODCAST = '.'.join(UDP_BRODCAST)
```

Dobiva se tako da se mrežna adresa servera razdijeli s pomoću funkcije .split() tamo gdje se nalazi točka. Tako se dobije polje s četiri stringa u kojem svaki string predstavlja jedan oktet mrežne adrese. Zadnji oktet se zamijeni s '255' stringom te se potom to polje spaja naredbom .join().

4.2.5. Izrada socketa

Kako bi program mogao komunicirati preko mreže moraju se pokrenuti socketi. Ovaj dio je različit ovisno o verziji programa. Kod programa koji koriste samo UDP komunikaciju potrebno je stvoriti samo UDP sockete.

```
UDP_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
UDP_socket.bind((IP_addres,UDP_PORT))
UDP_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST,
1)
```

Naredbom socket.socket() stvara se novi socket. Dodavanje opcije socket.AF_INE definira se kao socket internetske mreže, a opcija socket.SOCK_DGRAM definira protokol. U ovom slučaju radi se o protokolu UDP. Iako socket postoji kao objekt pomoću njega se još ne može komunicirati. Za to je potrebno dodijeliti mu mrežnu adresu i broj priključnice. To se radi naredbom .bind() gdje IP_address predstavlja mrežnu adresu servera, a UDP_PORT

konstantu koja određuje o kojem se programu radi na serveru. Na kraju je potrebno omogućiti slanje, broadcast, poruka s pomoću naredbe .setsocketopt(). Opcija socket.SOL_SOCKET označava da se želi primijeniti postavka samo na trenutni socket, a opcija socket.SO_BRODCAST označava želju za primjenom brodcast opcije socketa, dok je 1 vrijednost na koju se opcija postavlja.

I za kodove koji korist TCP način komunikacije potrebno je pokrenuti novi socket s drugim brojem priključnice. Postupak je sličan, ali postoje razlike u kodu.

```
TCP_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
TCP_socket.bind((IP_addres,TCP_PORT))
TCP_socket.listen()
```

Za stvaranje TCP socketa koristi se ista naredba socket.socket() kao i kod UDP socketa, ali se za opciju odabira protokola koristi naredba socket.SOCK_STREAM. Zatim se isto kao i kod UDP protokola naredbom .bind() dodaje mrežna adresa socketu i drugi broj priključnice kako bi see UDP i TCP socketi mogli razlikovati. Za kraj koristeći naredbu .listen() TCP socket je spreman za komunikaciju.

4.2.6. Pronalaženje kontrolera pomoću UDP-a

Ovaj dio koda služi kako bi kontroler saznao mrežnu adresu servera te kako bi server znao da je kontroler prisutan u mreži.

```
while (True):
    sock.sendto(b"ThisIsTIOSS", (UDP_BRODCAST, UDP_PORT))
    try:
        data, addr = sock.recvfrom(32)
        if (addr[0] == UDP_IP):
            data, addr = sock.recvfrom(32)
        data = data.decode()
        if (data == "HelloTIOSS"):
            print("Found controler")
            break
    except:
        print("Couldnt find connection!")
```

Naredbom .settimeout() UDP socket se postavlja u način čekanja. Ukoliko se koristi naredba .recvfrom() program će stati i jednu sekundu slušati mrežu za nadolazeće pakete,

nakon čega će javiti exeption. Zatim program ulazi u beskonačnu petlju koja sadrži funkciju za traženje kontrolera. Najprije se naredbom .sendto(b“ThisIsTIOSS“,(UDP_BRODCAST,UDP_PORT)) šalje poruka sadržaja ThiisIsTIOSS na brodcast adresu, odnosno svim računalima u mreži i broj priključnice UDP_PORT. Nakon toga se u try-except bloku pokreće naredba .recvfrom(32) koja jednu sekundu čeka poruku maksimalne veličine od 32 byte da dođe na UDP socket. Ukoliko poruka dođe naredba vrati njen sadržaj i adresu pošiljatelja. Kako je prva naredba slanje svim uređajima u mreži ta ista poruka se pošalje i serveru. Zato se naredbom if pogleda dolazeća poruka te ako je adresa pošiljatelja adresa servera čeka se nova poruka. Nakon što je zaprimljena poruka koja ne dolazi od samog servera njen sadržaj se pretvara iz byte u string tipa podataka u UNICODE string tip podataka naredbom .decode(). Ako je sadržaj poruke HelloTIOSS to znači da je kontroler našao adresu servera i da se može nastaviti izvođenje programa, a ako nije nastavlja se petlja. Isto se događa ako nije zaprimljena poruka unutar jedne sekunde od početka naredbe .recvfrom(), tj. naredba baca exception i petlja se samo nastavlja.

4.2.7. Pronalaženje kontrolera pomoću TPC-a

Ovaj dio koda služi kako bi kontroler saznao mrežnu adresu servera te kako bi server znao da je kontroler prisutan u mreži.

```
def lookingForConnection():
    TCP_socket.settimeout(1)
    while True:
        UDP_socket.sendto(b"ThisIsTIOSS", (UDP_BRODCAST,
UDP_PORT))
        try:
            client_socket, client_addres =
TCP_socket.accept()
            print("Client connected")
            return client_socket
        except:
            print("Client not found")
            continue
```

U ovom slučaju naredbom .settimeout() postavlja se TCP socket u način čekanja do jedne sekunde. Zatim se ulazi u beskonačnu petlju gdje se naredbom .sendto(b"ThisIsTIOSS", (UDP_BRODCAST, UDP_PORT)) šalje poruka preko UDP soketa svim uređajima u

mreži isto kao i kod čiste UDP komunikacije. No sada u try-except bloku naredbom TCP_socket.accept() započinje se čekanje od jedne sekunde dok kontroler ne započne spajanje na TCP socket. Ukoliko se unutar te jedne sekunde kontroler spoji na server prekida se petlja i vraća se client_socket. Ako se unutar te jedne sekunde, kontroler ne pošalje zahtjev za povezivanje, naredba .accept() baca exception i petlja se ponavlja sve dok se kontroler ne poveže sa serverom.

4.2.8. Inicijalizacija ulazno-izlaznih pinova

Nakon što server i kontroler uspostave komunikaciju bilo s pomoću UDP ili TCP protokola u svim verzijama programa na isti način se inicijaliziraju ulazno-izlazni pinovi i postavljaju u početna stanja.

```
pin_head_right = gpiod.Chip(CHIP_PATH).get_line(HEAD_RIGHT)
pin_head_right.request(consumer = "PIN",
type=gpiod.LINE_REQ_DIR_OUT)
pin_head_right.set_value(1)
```

Naredbom .Chip() adresira se kontroler koji upravlja ulazno-izlaznim pinovima, a konstanta CHIP_PATH adresira kontroler Raspberry Pi. Kako bi se odabrao specifičan pin kontrolera potrebno je koristiti naredbu .get_line() koja odabire pin s kojim se želi upravljati. Konstanta HEAD_RIGHT odgovara broju pina koji okreće glavu robota u desno. Potrebno je postaviti tip pina u izlazni što radi naredba .request(). Prije nastavka programa potrebno je postaviti početnu izlaznu vrijednost pina u visoku odnosno 1 što se radi naredbom .set_value(). Ovaj postupak se ponavlja za sve pinove iz Tablice 5.

4.2.9. Kod za diskretni način upravljanja

Ovim kodom definira se ponašanje robota nakon primanja instrukcije.

```
client_socket.settimeout(None)

while (True):
    try:
        data = client_socket.recv(32)
        data = data.decode()
        data = data.split('.')

        if (data[0] == 'GlavaLeft'):
```

```
                print("Glava u lijevo")
                pin_head_left.set_value(0)
                customWait(int(data[1]) * 2)
                pin_head_left.set_value(1)
                continue
            .
            .
            .
        except:
            continue
```

Kod diskretnog načina upravljanja prije beskonačne petlje naredbom .settimeout() i vrijednosti None naredba .recv() postaje blokirajuća. To znači da će program stati kada se ta naredba pozove. Nakon toga se ulazi u beskonačnu while petlju i pozove se naredba .recv() te se program blokira. Bitno je napomenuti da naredbe .settimeout() i .recv() ne ovise o vrsti protokola koji se koristi za slanje, tj. iste su za oba protokola. Zatim server čeka dok kontroler ne pošalje poruku. Kada server dobije poruku koja je formata GlavaLeft.1 poruka se dekodira naredbom .decode() te dijeli naredbom .split() na GlavaLeft i 1. GlavaLeft govori o kojoj se naredbi radi, odnosno koji motor i u kojem smjeru se uključuje. Broj 1 se odnosi na to koliko dugo motor radi. S pomoću if naredbe odabire se odgovarajući pin te se motor uključuje naredbom .set_value(0), odnosno postavljanjem odgovarajućeg pina na nisku naponsku razinu. Zatim se poziva funkcija customWait() s parametrom koliko dugo se motor treba vrtjeti.

```
def customWait(wait_time):
    client_socket.settimeout(wait_time)
    try:
        data = client_socket.recv(32)
        data = data.decode()
        if (data == "Cancle"):
            print("Cancled")
            client_socket.settimeout(None)
            return
    except:
        print("Gotovo cekanje")
        client_socket.settimeout(None)
```

Umjesto funkcije customWait() moguće je koristiti funkciju sleep() koja bi blokirala program na proizvoljno vrijeme. Funkcija customWait() je korištena kako bi se moglo prekinuti izvođenje odabrane naredbe. Prekid se izvodi tako da se naredbom .settimeout()

postavi socket u način rada u kojemu čeka poruku, za duljinu vremena čekanja koristi se vrijednost s kojom je funkcija customWait() pozvana. Nakon toga se naredbom .recv() unutar try-except bloka blokira program na traženi broj sekundi. U slučaju da dođe poruka sadržaja Cancle socket se naredbom .setimeout(None) vraća u blokirajući način rada te se iz funkcije izlazi prije isteka vremena. Ukoliko prođe zadano vrijeme ulazi se u except dio bloka socket se vraća u blokirajući način rada i izlazi iz funkcije.

Nakon povratka iz customWait() funkcije naredbom .set_value(1) postavlja se pin u visoko stanje i time se zaustavlja motor. Bitno je napomenuti da motori imaju analogne graničnike koji isključuju napon i automatski zaustavljaju motor kada dođu u krajnji položaj, što znači da program ne mora voditi računa o položaju. Ovaj postupak je isti za sve motore.

Upravljanje hvataljkama je malo drukčije zato što one mogu biti ili uključene ili isključene, odnosno otvorene ili zatvorene.

```
if (data[0] == 'RArmHold'):
            if (not right_arm_hold):
                print("Desna ruka pusti")
                right_arm_hold = True
                pin_hand_right_hold.set_value(0)
                customWait(int(data[1]))
                continue
            if (right_arm_hold):
                print("Desna ruka stisni")
                right_arm_hold = False
                pin_hand_right_hold.set_value(1)
                customWait(int(data[1]))
```

Kako bi se moglo pratiti je li hvataljka otvorena ili zatvorena koriste se pomoćne zastavice, jedna za svaku hvataljku, kojima je početna vrijednost False. Kada dođe naredba za promjenom stanja hvataljke najprije se s pomoću zastavice provjeri u kojem je stanju hvataljka, zatim joj se naredbom .set_value() promijeni stanje. Za hvataljku desne ruke stanje 0 predstavlja otvoreno stanje, dok stanje 1 predstavlja zatvoreno stanje. Kod hvataljke lijeve ruke stanja su obrnuta. Istovremeno se osvježi vrijednost zastavica kako bi pratile u kojem je stanju hvataljka. Kao mjera opreza nakon promjene stanja dodano je čekanje od jedne sekunde kako bi se hvataljka stigla otvoriti, odnosno zatvoriti te kako bi se zaštitio solenoid.

4.2.10. Kod za kontinuirani način upravljanja

Ovim kodom definira se ponašanje robota nakon primanja instrukcije.

```
client_socket.settimeout(0)

while (True):
    try:
        data, addr = client_socket.recvfrom(32)
        data = data.decode()

        if (data == 'GlavaLeft'):
            if (time() - rememberTime < 0.07):
                rememberTime = time()
                continue
            rememberTime = time()
            motor_is_running = True
            pin_head_left.set_value(0)
            print("Glava u lijevo")
            continue
    except:
        if (time() - rememberTime > 0.2 and
motor_is_running):
            print("Ugasi")
            motor_is_running = False
            stopMotors()
        continue
```

Kod kontinuiranog načina upravljanja prije beskonačne petlje socket se postavlja u neblokirajući način rada naredbom .settimeout(0), odnosno svaki puta kada se pozove naredba .recvfrom() ako u tom trenutku ne dobije poruku naredba baca exception. Za razumijevanje ovog koda bitno je razumjeti ulogu varijable rememberTime. Ta varijabla se koristi kako bi se znalo vrijeme kada je motor zadnji puta uključen. U beskonačnoj petlji se nalazi try-except blok koji počinje s pozivom naredbom .recvfrom() te ako naiđe na poruku onda se dekodira naredbom .decode(). Zatim se s pomoću if naredbi odredi koji je motor potrebno uključiti. Najprije se if naredbom provjeri je li prošlo manje od 70 milisekundi od zadnje primljene poruke. Ukoliko je to tako tada se varijabli rememberTime pridružuje trenutno vrijeme pozivom funkcije time()) te se nastavlja s novim prolazom kroz beskonačnu petlju. U slučaju da je prošlo više od 70 milisekundi od zadnje poruke varijabli

rememberTime se pridružuje trenutno vrijeme pozivom funkcije time() te se zastavica motor_is_runnning postavlja u True i naredbom .set_value(0) motor se uključi.

Motor radi sve dok pristižu poruke od kontrolera, no kada prestanu stizati poruke odvija se kod u except dijelu. Kada ne stigne poruka s pomoću if naredbe se provjerava je li prošlo više od 200 milisekundi od zadnje poruke i je li neki od motora uključen. Ako je uvjet if naredbe istinit, onda se mijenja vrijednost zastavice u False poziva se funkcija stopMotors() koja samo naredbom .set_value(1) zaustavlja sve motore.

Upravljanje hvataljkama je isto kao i u programu s diskretnim načinom upravljanja. Jedina razlika je u mjeri opreza, odnosno u brizi da se solenoid stalno ne otvara, odnosno zatvara.

```
if (data == 'RArmHold'):
            if (time() - rememberTime < 0.2):
                continue
            rememberTime = time()
            if (not right_arm_hold):
                print("Desna ruka pusti")
                right_arm_hold = True
                pin_hand_right_hold.set_value(0)
                continue
            if (right_arm_hold):
                print("Desna ruka stisni")
                right_arm_hold = False
                pin_hand_right_hold.set_value(1)
                continue
```

Za zaštitu se koristi if naredba koja provjerava je li prošlo više od 200 milisekundi od prijašnje poruke. Ukoliko je prošlo više od 200 milisekundi osvježava se varijabla rememberTime te mijenja stanje hvataljke. Ako nije prošlo više od 200 milisekundi kreće izvođenje sljedeće iteracije beskonačne petlje.

5. Klijent

U sklopu ovoga rada klijent je kontroler odnosno VIDI X mikroračunalo. S pomoću njega će se slati instrukcije na server. Odlična stvar kod VIDI X mikroračunala je što su sve periferne jedinice već spojene ma mikrokontroler tako da je samo potrebno iščitati na koje su pinovi spojene. Za potrebe ovoga rada potrebno je znati pinove deset tipki te pinove na koje je spojen LCD ekran. Pošto se ne koristi osjetljivost na dodir ekrana potrebni su samo pinovi za kontrolu ispisa na ekran.

GPIO pin	*Uloga*
5	Odabir ekrana za komunikaciju.
21	Kontrolni pin LCD ekrana.
35	Tipke UP i DOWN.
34	Tipke LEFT i RIGHT.
32	Tipka BTN_A.
33	Tipka BTN_B.
13	Tipka MENU.
27	Tipka SELECT.
39	Tipka START.
0	Tipka VOLUME.

Tablica 5 Korišteni pinovi VIDI X-a

Za programiranje VIDI X mikroračunala koristi se programski jezik C++. Komplementarno serveru napravljena su četiri programa za klijenta. Razlikuju se po načinu komunikacije i načinu upravljanja. Program s diskretnim načinom upravljanja šalje jednu poruku kada je pritisnuta tipka za slanje željene instrukcije dok kod kontinuiranog načina upravljanja pritiskom na tipku šalju se poruke sve dok je tipka stisnuta. Specifičnost kod programa za mikrokontrolere su samopozivajuće funkcije setup i loop. Funkcija setup

se izvodi samo jednom i služi kako bi priredila sklop za rad. Dok funkcija loop predstavlja beskonačnu petlju u kojoj se nalaze naredbe koje mikrokontroler stalno ponavlja.

5.1. Opis biblioteka

Biblioteka WiFi.h je biblioteka Arduino IDE razvojnog okruženja [16]. Omogućava spajanje mikroprocesora na bežične mreže i omogućuje slanje i primanje poruka s pomoću UDP i TCP protokola. Ova biblioteka je neophodna za rad programa jer s njom se kontroler spaja na bežičnu mrežu te s pomoću nje se šalju instrukcije na server.

Biblioteka WiFiUDP.h je dio biblioteke WiFi.h s pomoću koje se obavlja komunikacija UDP protokolom.

Biblioteka string.h je standardna biblioteka programskog jezika C++ [17]. Sadrži funkcije za manipulaciju stringova i nizova. U ovom programu ova biblioteka je potrebna za usporedbu dolazećih poruka.

Biblioteka Adafruit_ILI9341.h je proizvod Adafruit Industries [18]. To je biblioteka otvorenog koda koja omogućuje jednostavnu komunikaciju s ekranima koje prodaje Adafruit stranica. LCD koji se nalazi na VIDI X-u je kompatibilan s ovom bibliotekom.

Biblioteka Adafruit_GFX.h je biblioteka otvorenog koda koju Adafruit Industries održava [19]. Napravljena je primarno za korištenje s ekranima koji su prodavani na njihovim stranicama. Olakšava ispis teksta, crtanje oblika te prikaz slika na ekranima. Ova biblioteka je kompatibilna s LCD ekranom koji se nalazi na VIDI X mikroračunalu.

5.2. Programski kod

Programski kod za VIDI X sastoji se od tri glavne cjeline: ispisa na LCD ekran, mrežne komunikacije te odabira instrukcija za upravljanje robotom. U nastavku ispis na LCD ekran će biti opisan samo na jednom primjeru te sve ostale funkcije koje rade ispis na LCD ekran će biti samo spomenute, fokus je na funkcijama koje obavljaju mrežnu komunikaciju i odabir instrukcija. Program je koncipiran tako da se spoji na bežičnu mrežu, zatim pronađe server te daje izbor instrukcija za upravljanje robotom. Na LCD ekranu se iscrtava svaki korak izvođenja programa.

5.2.1. Uvoz biblioteka

Uvoz prije navedenih biblioteka. Kroz sve inačice programa koriste se iste biblioteke.

```
#include <WiFi.h>
#include <WiFiUdp.h>
#include <string.h>
#include <Adafruit_ILI9341.h>
#include <Adafruit_GFX.h>
```

5.2.2. Zastavica i globalne varijable

Zastavicama i globalnim varijablama definiraju se postavke i početni parametri programa.

```
const char* ssid = "TIOSLAN";
const char* password = "suzasuza";
const int CONNECTION_TIMEOUT = 10;
```

Pomoću prve tri varijable definiraju se parametri za spajanje na bežičnu mrežu. Varijabla ssid predstavlja ime mreže, varijabla password predstavlja zaporku mreže, a varijabla CONNECTION_TIMEOUT određuje koliko puta će se mikrokontroler pokušati spojiti na bežičnu mrežu.

```
WiFiUDP Udp;
char packet_buffer[255];
uint8_t  reply_buffer[] = "HelloTIOSS";
const char tioss_message[] = "ThisIsTIOSS";
IPAddress tioss_ip;
unsigned int udp_port = 5005;
```

S pomoću ovih varijabli definira se UDP komunikacija. Za razliku od kodiranja u Pythonu ne mora se eksplicitno dodjeljivati mrežna adresa objektu, jer to sam odrađuje. Varijabla udp_port sadrži broj priključnice preko koje se odvija komunikacija bitno je pobrinuti se da je taj broj isti i na serveru i na klijentu. Varijabla packet_buffer služi za zapis dolazećih poruka na UDP priključnicu. U varijabli tioss_message nalazi se poruka koju kontroler očekuje od servera, dok varijabla reply_buffer sadrži poruku koju server očekuje od kontrolera. Kada kontroler prepozna server onda njegovu mrežnu adresu zapiše u varijablu tioss_ip. Sve do sad navedene varijable se nalaze u svim inačicama programa.

```
WiFiClient tcp_client;
int tcp_port = 5006;
```

Ove varijable se nalaze samo u TCP inačicama programa. Objekt tcp_client se koristi za komunikaciju s pomoću TCP protokola dok varijabla tcp_port predstavlja broj priključnice koji se koristi za komunikaciju s pomoću TCP protokola.

S pomoću zastavica definiraju se pinovi na koje su tipke i LCD ekran spojeni. Sve zastavice su iste neovisno o inačici programa.

```
#define TFT_CS 5
#define TFT_DC 21
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
```

Za upravljanje LCD ekranom potrebna su samo dva pina TFT_CS i ATFT_DC. Nadalje u kodu LCD ekran će se zvati tft. Kako bi se moglo ispisivati stvari na LCD ekran stvara se objekt tft pozivom naredbe Adafruit_ILI9341(TFT_CS, TFT_DC). Kao opcije naredbe prosljeđuju se pinovi na koje je LCD ekran spojen, nije potrebno odvojeno inicijalizirati pinove za LCD ekran.

```
#define BTN_UP_DOWN  35
#define BTN_LEFT_RIGHT 34
#define BTN_A  32
#define BTN_B  33
#define BTN_MENU 13
#define BTN_SELECT 27
#define BTN_START  39
#define BTN_VOL  0
```

Ove zastavice predstavljaju na koje su pinove spojeni koji gumbi. Može se primijetiti da za gumbe RIGHT i LEFT te UP i DOWN ima samo jedan gumb po paru. To je zato što su ti gumbi spojeni na analogni ulaz.

5.2.3. Setup funkcija

Ova funkcija je prva funkcija koja se izvodi kada se uređaj pokrene. S pomoću nje se postavi uređaj za rad.

```
void setup()
{
  Serial.begin(115200);
  tft.begin();
  connectToWiFi();
  connectedWIFI();
  Udp.begin(udp_port);
```

```
  findTIOSS();
  connectedTIOSS();
  initializePins();
  startMainMenu();
}
```

Serial.begin() pokrene se serijska veza za komunikaciju s Arduino IDE alatom koja služi za pomoć pri testiranju rada programa. Broj naveden unutar zagrada je brzina komunikacije. Naredba tft.begin() inicijalizira LCD ekran i priprema ga za korištenje. Zatim se poziva funkcija connectToWiFi() koja spaja VIDI X na bežičnu mrežu. Kada se VIDI X spoji na bežičnu mrežu poziva se funkcija connectedWiFi() koja na ekranu postavi pozadinu u crno i u gore lijevi kut zelenim slovima ispiše da je kontroler spojen na bežičnu mrežu. Potom naredbom Udp.begin(udp_port) omogućuje se komunikacija s pomoću Udp protokola. Pozivom funkcija findTIOSS() kontroler osluškuje mrežu kako bi se povezao sa serverom. Ova funkcija ima dvije varijante a varijanta koja se koristi ovisi o komunikacijskom protokolu. Funkcija connectedTIOSS() ispisuje u gornji lijevi kut, ispod poruke o spajanju na bežičnu mrežu, da je kontroler pronašao server. Zvanjem funkcije initializePins() inicijaliziraju se pinovi koji su spojeni za gumbe kako bi ih se moglo koristiti u ostatku programa. Za kraj zove se funkcija startMainMenu() koja na tft ekran iscrtava početi zaslon. Na početnom zaslonu u gornjem lijevom kutu zelenim slovima je ispisano da je kontroler spojen na bežičnu mrežu i na server te se u sredini ekrana nalazi izbornik s pomoću kojega se može izabrati želi li se upravljati motorima glave ili ruku robota. Na dnu LCD ekrana nalazi se uputa kako izabrati jednu od opcija izbornika.

5.2.4. Spajanje na bežičnu mrežu

Pozivom funkcije connectToWiFi() mikrokontroler se spaja na bežičnu mrežu.

```
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
```

Naredbom WiFi.mode(WIFI_STA) mikrokontroler postaje uređaj koji se spaja na bežičnu mrežu. Zatim pozivanje naredbe WiFi.begin(ssid, password) počinje spajanje na bežičnu mrežu definiranu varijablama ssid i password. Nakon toga na LCD ekran se ispisuje poruka da se mikrokontroler spaja na bežičnu mrežu.

```
tft.fillScreen(ILI9341_BLACK);
tft.setTextSize(4);
tft.setRotation(3);
```

```
tft.setTextColor(ILI9341_RED);
tft.setCursor(40,80);
tft.print("Connecting");
tft.setCursor(10,120);
```

Prvo se cijeli ekran postavi u crno naredbom tft.fillScreen(ILI9341_BLACK). Zastavica ILI9341_BLACK predstavlja heksadecimalni kod za crnu boju. Umjesto nje se može upisati bilo koja heksadecimalna vrijednost. Zatim naredbom .setTextSize(4) definira se veličina slova. Naredbom .setRotation(3) postavlja se orijentacija ispisa slova i pozicija početka ispisa. Opcijom 3 početna točka se nalazi u gornjem lijevom kutu. S pomoću naredbe .setTextColor(ILI9341_RED) mijenja se boja teksta koji će se ispisati. Isto kao i kod naredbe .fillScreen() zastavica ILI9341_RED predstavlja heksadekadsku vrijednost crvene boje. Dodatno dodavanje zareza u opciji naredbe .setTextColor() može se definirati i boja pozadine slova. Ta opcija nije korištena u ovom dijelu, ali koristi se kasnije. Kako bi se odabrala lokacija ispisa teksta koristi se naredba .setCursor() koja pomakne točku ispisa za 40 pixela u desno i 80 pixela prema dolje u odnosu na gornji lijevi kut. I na kraju naredbom .print() ispituje se željeni tekst na sam ekran.

```
int dot_counter = 0;
int timeout_counter = 0;

while(WiFi.status() != WL_CONNECTED)
{
  tft.print(".");
  delay(100);
  dot_counter++;
  timeout_counter++;
  if (dot_counter > 11)
  {
    tft.fillRect(10,120,300,40,ILI9341_BLACK);
    tft.setCursor(10,120);
    dot_counter = 0;
  }
  if(timeout_counter >= CONNECTION_TIMEOUT * 5)
    ESP.restart();
}
```

Pošto treba neko vrijeme kako bi se mikrokontroler spojio na bežičnu mrežu pokreće se while petlja koja provjerava je li se mikrokontroler uspio spojiti na bežičnu mrežu. To radi negacijom, odnosno uvjet će biti istinit sve dok mikrokontroler nije spojen na bežičnu

mrežu. Prolazak kroz while petlju događa se svakih 100 milisekundi što je napravljeno naredbom delay(). Isto tako svakim prolazom kroz while petlju na LCD ekran se ispisuje točka kako bi se signaliziralo korisniku da program još uvijek radi. Kada se na ekranu ispiše 12 točaka naredbom .fillRect() se brišu točke. Naredba .fillRect() iscrtava ispunjeni pravokutnik s početnom točkom (10,120) pixela širine 300 pixela i visine 10 pixela, popuno crne boje. Za kraj u s pomoću varijable timeout_counter broji se koliko je vremena prošlo te ako je prošlo više od 5 sekundi ponovno pokreni mikrokontroler. Kada se mikrokontroler spoji na bežičnu mrežu uvjet while petlje više nije istinit te se pelja prekida i izlazi se iz connectToWiFi() funkcije.

5.2.5. Funkcija findTIOSS() za UDP protokol

Ova funkcija služi kako bi kontroler saznao mrežnu adresu servera. Radi tako da čita sve pakete koji su mu poslani te kada pročita poruku čiji sadržaj odgovara prije dogovorenoj frazi zna da je tu poruku poslao server te zabilježi mrežnu adresu pošiljatelja te poruke. I na kraju šalje serveru poruku kojom potvrđuje da je našao njegovu mrežnu adresu.

Funkcija počinje tako da na LCD ekran ispiše poruku korisniku da kontroler trenutačno traži server odnosno robota TIOSS. Naredbe koje se koriste za to su iste one koje su se koristile kod funkcije connectToWiFi(). Zatim ulazi u beskonačnu while petlju.

```
int packet_size = Udp.parsePacket(); // Procitaj paket
if (packet_size) // ako paket nije prazan
{
  int len = Udp.read(packet_buffer, 255);
  if (len > 0)
  {
    packet_buffer[len] = 0;
  }
  if (!strcmp(packet_buffer, tioss_message))
  {
    tioss_ip = Udp.remoteIP();
    sendMessage(reply_buffer);
    return;
```

U petlji na početku naredbom .parsePacket() provjerava se da li je na UDP priključnicu došao neki paket. Ako je došao novi paket vrijednost varijabla packet_size je veća od 0. Naredbom if provjerava se je li varijabla packet_size veća od 0 i ako je čita se sadržaj paketa. Naredbom .read() pročita se nadolazeći paket i spremi se u prije definiranu

varijablu packet_buffer. Druga opcija naredbe .read() definira maksimalnu veličinu paketa koju naredba može pročitati. Rezultat poziva naredbe .read() pridružuje se numeričkoj varijabli len onda služi kako bi se na kraj znakovnog niza poruke mogao dodati null terminator koji označuje kraj znakovnog niza. U zadnjoj if naredbi provjerava se dali sadržaj poruke odgovara frazi koju šalje server. U slučaju da se znakovni nizovi podudaraju funkcija strcmp() vraća 0. Prepoznajući da je server taj koji je poslao poruku u prije definiranu varijablu tioss_ip sprema se mrežna adresa servera naredbom .remoteIP(). I prije izlaska iz funkcije pozivom naredbe sendMessage(reply_buffer) šalje se odgovor na server.

5.2.6. Funkcija findTIOSS() za TCP protokol

Funkcija obavlja istu svrhu bez obzira na komunikacijski protokol koji se koristi. Jedina razlika je što kada prepozna da nadolazeće poruke dolaze od servera umjesto da pošalje odgovor na server koristi se svojstvo TCP protokola o uspostavljanju veze kako bi se uspostavila komunikacija. Kod ove funkcije izgleda skoro identično kao i u dijelu s UDP komunikacijom s izmjenama samo na kraju beskonačne petlje.

```
while (1)
  { int packet_size = Udp.parsePacket();
    if (packet_size)
    {
      int len = Udp.read(packet_buffer, 255);
      if (len > 0)
      { packet_buffer[len] = 0; }
      if (!strcmp(packet_buffer, tioss_message))
      {tioss_ip = Udp.remoteIP();
        break;}
   }}
  if (tcp_client.connect(tioss_ip,tcp_port))
    return;
```

U zadnjoj if naredbi beskonačne petlje kada se zabilježi mrežna adresa servera umjesto da se šalje odgovor, naredbom break se izlazi iz petlje. Potom naredbom connect() se uspostavlja komunikacijski kanal sa serverom. Kao opcije naredbe .connect() potrebno je navesti adresu servera te broj priključnice na koju se spaja. Uspostavljanjem komunikacijskog kanala između kontrolera i servera izlazi se iz funkcije findTIOSS().

5.2.7. Funkcija sendMessage() za UDP protokol

Ova funkcija postoji zbog razlika u načinu slanja paketa između protokola. Za poslati poruku s pomoću UDP protokola potrebno je pozvati nekoliko naredbi stoga je taj postupak odvojen u svoju funkciju.

```
void sendMessage(uint8_t *packet)
{
  int messegeSize = 0;
  while (packet[messegeSize] != '\0')
  { messegeSize++; }
  Udp.beginPacket(tioss_ip, tioss_port);
  Udp.write(packet, messegeSize);
  Udp.endPacket();
}
```

Kako bi se poslao paket s pomoću UDP protokola potrebno je znati veličinu paketa. U tu svrhu koristi se varijabla messegeSize i while petlja. While petlja povećava vrijednost varijable messegeSize za jedan sve dok ne dođe do null-terminatora u nizu koji se želi poslati. Nakon što se odredi veličina paketa počinje se slanje paketa naredbom .beginPacket() u kojoj se navede mrežna adresa odredišta i priključnica za UDP komunikaciju. Potom naredbom .write() se odredi koji niz se šalje te njegova veličina. Za kraj se slanje završi pozivom naredbe .endPacket(). Vrijedi napomenuti da se u varijabli messegeSize nalazi duljina niza bez null-terminatora zato što kada poruka stigne na server, programski jezik Python dodaje svoj null-terminator. I zbog toga ako se pošalje niz s null-terminatorom usporedba nizova u Pythonu neće raditi.

5.2.8. Funkcija sendMessage() za TCP protokol

Funkcija sendMessage() je znatno jednostavnija kod rada s TPC protokolom. Sastoji se samo od jedne linije.

```
void sendMessage(char *packet)
{
  tcp_client.write(packet);
}
```

Razlika između funkcija se može uočiti već pri samoj definiciji. Dok kod UDP-a znakovni niz mora biti tipa uint8_t, kod TCP-a se koristi tip char. Zatim se pozivom naredbe .write() pošalju poruke. Naredba .write() može poslati nizove tipa char i byte. U sklopu WiFi.h

biblioteke postoje još dvije funkcije za slanje poruka. One su .print() i .println(). S pomoću njih se mogu slati i drugi tipovi podataka, ali unutar naredbi se pretvaraju u tip char pa nisu pretjerano korisne.

5.2.9. Funkcija initializePins()

Kako bi se mogle koristiti tipke na VIDI X-u potrebno je inicijalizirati pinove ne koje su oni spojeni.

```
void initializePins()
{
  pinMode(BTN_UP_DOWN, INPUT_PULLUP);
  pinMode(BTN_LEFT_RIGHT, INPUT_PULLUP);
  pinMode(BTN_A, INPUT_PULLUP);
  pinMode(BTN_B, INPUT_PULLUP);
  pinMode(BTN_MENU, INPUT_PULLUP);
  pinMode(BTN_SELECT, INPUT_PULLUP);
  pinMode(BTN_START, INPUT_PULLUP);
  pinMode(BTN_VOL, INPUT_PULLUP);
}
```

Naredbom pinMode() inicijalizira se odabrani pin. Pin koji se želi inicijalizirati se proslijedi kao prva opcija naredbe dok druga opcija određuje u kojem načinu rada je pin. U ovom slučaju svi pinovi se postavljaju kao ulazni pinovi u eng. pullup načinu rada. To znači da su pinovi početno spojeni na visoku naponsku razinu te da pritisak tipke spaja na nisku naponsku razinu. Ovaj trik se koristi kako bi se garantiralo očitanje pritisnute tipke. S programske strane to znači da prilikom očitavanja stanja pojedinačnog pina ako se nalazi u niskom stanju to znači da je tipka pritisnuta.

5.2.10. Funkcija loop()

Ovo je glavna funkcija programa te služi kao početak programa s kojim korisnik može ostvariti interakciju. Prije samog poziva ove funkcije na LCD ekran se iscrtava glavni izbornik koji daje korisniku pregled s kojim funkcijama robota može upravljati. Specifično u ovome radu radi se o izboru između glave i ruku robota. Trenutačni odabir predstavljen je bijelom pozadinom i crnim slovima. Korisnik može tipkama UP i DOWN mijenjati odabir te pritiskom na tipku SELECT potvrditi svoj odabir što poziva funkciju koja obrađuje odabranu funkcionalnost.

```
void loop() {
```

```
  delay(120);
  if (analogRead(BTN_UP_DOWN) > 4000)
  {…}
  if (analogRead(BTN_UP_DOWN) > 1800 and
analogRead(BTN_UP_DOWN) < 2000)
  {…}
  if (digitalRead(BTN_SELECT) == LOW)
  {…}
}
```

Prije same funkcije definira se pomoćna varijabla menu_item cjelobrojnog tipa. Pomoćna varijabla menu_item koristi se za prepoznavanje trenutačno izabrane opcije u izborniku. Zbog velike brzine izvođenja instrukcija potrebno je usporiti program kako bi se mogao registrirati samo jedan pritisak tipke. Program se usporava dodavanjem čekanja od 120 milisekundu naredbom delay(). Za prepoznavanje tipke koja je pritisnuta koristi se if naredba koja u svojem uvjetu ima provjeru stanja pina tipke. Većina tipki su digitalnog tipa što znači da mogu samo poprimiti nisko i visoko stanje te se njihovo stanje provjerava naredbom digitalRead() i kao opciju se pošalje broj pina (primjer tipke SELECT). Međutim tipke UP i DOWN kao i LEFT i RIGHT su spoje na jedan analogni pin čija vrijednost se dobiva naredbom analogRead(). Naredba analogRead() ne vraća samo nisko ili visoko stanje već vraća neku vrijednost iz raspona 0 do 4095. Kako očitanje nije uvijek savršeno uvode se tolerancije. Tako da kada se naredbom analogRead() s pina označenog zastavicom BTN_UP_DOWN pročita vrijednost veća od 4000 tada je pritisnuta tipka UP, ako je očitana vrijednost unutar raspona 1800 do 2000 onda je pritisnuta tipka DOWN.

```
if (analogRead(BTN_UP_DOWN) > 4000)
  {
    menu_item = abs((menu_item - 1) % 2);
    updateManinMenu(menu_item);
  }
```

Pritiskom na tipku UP ili DOWN mijenja se koju opciju želimo izabrati na glavnome izborniku. To se radi tako da kada je detektiran pritisak tipke prvo se računa nova vrijednost varijable menu_item tako da se doda ili oduzme 1 zatim se nađe modulo 2 te se od tog broja uzme apsolutna vrijednost. Traži se modulo s brojem dva jer su samo dva izbora, ako bi bilo više izbora onda se ova vrijednost mora promijeniti. Ovakvim računanjem varijable menu_item omogućuje se prolaženje kroz opcije izbornika bilo kojim redoslijedom. Nakon promijene vrijednosti varijable menu_items poziva se funkcija

updateManinMenu() kojoj se prosljeđuje varijabla menu_items. Funkcija updateManinMenu() zatim ponovo ispisuje opcije glavnog izbornika ali sada bijelom pozadinom će biti označena druga opcija.

```
if (digitalRead(BTN_SELECT) == LOW)
  {
    if (menu_item == 0)
      glavaLogic();
    if (menu_item == 1)
      rukeLogic();
    menu_item = 0;
  }
```

Kada se odabere kojim dijelom robota se želi upravljati pritiskom na tipku SELECT se poziva funkcija koja je zaslužna za upravljanjem tim djelom robota. Na kraju kada se vrati u glavni izbornik varijabla menu_item se postavlja na 0. Funkcije za upravljanje različitim dijelovima robota su skoro identične samo se razlikuju u opcijama koje pružaju tako da će biti samo biti objašnjena funkcija za upravljanje s glavom. Ove funkcije također se razlikuju ovisno o načinu upravljanja robotom.

5.2.11. Funkcija glavaLogic () za diskretno upravljanje

Ova funkcija pokazuje opcije kako se može upravljati glavom robota TIOSS. Zato što se upravlja njome diskretno dodatno će s desne strane ekrana biti iscrtan stupac s pomoću kojega se može odrediti koliko dugo se izvodi željena instrukcija. Svaki pravokutnik stupca predstavlja dvije sekunde. Na dijelovima funkcije pokazna je logika kojom funkcija radi.

```
void glavaLogic(){
  int power_bar = 0;
  glavaScreen();
  drawPowerBar(power_bar);
  while(1){
    delay(100);
    if (digitalRead(BTN_A) == LOW){
      if(power_bar < 2)
        power_bar++;
      drawPowerBar(power_bar);
      continue; }}}
```

Prvo se definira pomoćna varijabla power_bar s pomoću koje će se određivati koliko dugo korisnik želi da instrukcija traje. Varijabla power_bar može poprimiti cjelobrojne vrijednosti od 0 do 2. Zatim pozivom funkcije glavaScreen() se na LCD ekranu ispišu naredbe koje se mogu slati na robota te funkcijom drawPowerBar() se iscrtava stupac s desne strane LCD ekrana koji radi vizualnu reprezentaciju željenog vremena trajanja.

Unutar beskonačne while petlje pomoću if naredbi detektira se pritisak tipki. U slučaju pritiska tipke BTN_A povećava vrijednost varijable power_bar, za 1, samo ako je manja od 2, analogno tipka BTN_B smanjuje vrijednost varijable power_bar, za 1, samo ako je veća od 0. Prije nastavka while petlje pozivanjem funkcije drawPowerBar() iscrtava se nova vrijednost varijable power_bar.

```
if (analogRead(BTN_LEFT_RIGHT) > 4000)
 {
      uint8_t buff[20] = "GlavaLeft";
      sendMessagePower(buff, power_bar + 1);
      waitScreen((power_bar + 1) * 2);
      glavaScreen();
      drawPowerBar(power_bar);
      continue;
    }
```

Pritiskom na jedu od tipki koje predstavljaju funkcionalnost robota npr. glava u lijevo. Definira se varijabla buff koja sadrži definiranu frazu te instrukcije. Poziva se funkcija sendMessagePower() s parametrima koji predstavljaju sadržaj poruke i trajanje instrukcije te funkcija waitScreen() s parametrom power_bar njemu dodana jedinica i pomnožen s 2. Ove dvije funkcije su jedinstvene za diskretni način upravljanja i bit će objašnjene u nastavku. Nakon izvršavanja tih dviju funkcija ponovno se iscrtava prikaz opcija glave i stupac s desne strane s pomoću funkcija glavaScreen() i drawPowerBar().

Ukoliko se pritisne tipka MENU na ekran se iscrtava glavni meni i izlazi se iz funkcije glavaLogic (). Analogno ovoj funkciji funkcija rukeLogic radi na isti način.

```
void sendMessagePower(uint8_t *packet, int num)
{
  uint8_t c = num + '0';
  int len = strlen((char *)packet);
  packet[len] = '.';
  packet[len + 1] = c;
  packet[len + 2] = '\0';
```

```
  sendMessage(packet);
}
```

Funkcija sendMessagePower() kao parametre prima sadržaj poruke koja se šalje te duljinu izvođenja instrukcije. Zatim na kraj poruke dodaje točku i samo trajanje instrukcije te pozivom funkcije sendMessage() pošalje instrukciju na server.

```
void waitScreen(int wait_time){
  int count = 0;
  connectedTIOSS();
  …
  while(wait_time > 0){
    delay(100);
    if (count == 9){
      wait_time--;
      tft.setCursor(130,100);
      tft.setTextSize(10);
      tft.setTextColor(ILI9341_WHITE, ILI9341_BLACK);
      tft.print(wait_time);
      count = 0;}
    if (digitalRead(BTN_START) == LOW){
      uint8_t buff[] = "Cancle";
      sendMessage(buff);
      return; }
    count++;}
  delay(100);
}
```

Funkcija waitScreen() služi za postavljanje programa u način čekanja dok se zadana instrukcija ne izvrši. Funkcija na ekranu brojčano signalizira korisniku koliko je vremena ostalo do kraja izvođena zadane instrukcije te dozvoljava korisniku slanje prekida izvođenja instrukcije. Na početku definira se pomoćna varijabla count koja će brojati da li je prošla jedna sekunda. Varijabla wait_time predstavlja broj sekundi koje korisnik mora čekati kako bi instrukcija završila. U uvjetu while petlje provjerava se da li je vrijeme čekanja veće od nula i dok je izvodi se petlja. Naredbom delay() između svakog prolaza while petlje prođe 100 milisekundi što znači da svakih deset prolaza treba smanjiti vrijednost varijable wait_time i na LCD ekran ispisati novi broj. Ukoliko se za vrijeme izvođenja ove funkcije pritisne tipka START na server se pošalje poruka „Cancle“ te se izlazi iz ove funkcije.

5.2.12. Funkcija glavaLogic () za kontinuirano upravljanje

Funkcije koje se koriste za odabir instrukcija u kontinuiranom načinu upravljanja su puno jednostavnije. Imaju istu funkciju kako i u diskretnom načinu rada samo puno manje stvari zapravo trebaju obavljati. Isto ispišu opcije upravljanja robotom i zatim prate koje opcije su odabrane tako da prate koje su tipke pritisnute.

```
void glavaLogic(){
  glavaScreen();
  while(1){
    delay(50);
    if (analogRead(BTN_LEFT_RIGHT) > 4000) // Lijeva tipka {
      char buff[20] = "GlavaLeft";
      sendMessage(buff);
      continue;}
      }
}
```

Tu pri pozivu funkcije jedina stvar koja se iscrtava na LCD ekran je naznaka koja je opcija odabrana te koje sve instrukcije su podržane. Beskonačna while petlja je usporena funkcijom delay() na izvođenje svakih 50 milisekundi. Pritiskom tipke jednih od opcija definira se varijabla buff koja sadrži frazu odabrane instrukcije. Zatim pozivom funkcije sendMessage() pošalje se instrukcija na server. Pritiskom na tipku MENU pozivaju se funkcije connectedTIOSS() i startMainMenu() za iscrtavanje grafika početnog izbornika te se izlazi iz funkcije.

6. Rezultati

Sve verzije programa su testirane u laboratorijskim uvjetima i radile su bez greške. To nije bilo očekivano ponašanje programa. U prijašnjeg susreta s VIDI X mikroračunalom, kada bi se koristio program koji komunicira pomoću UDP protokola, dolazilo je do gubitka paketa. Iz tog razloga očekivalo se da nakon pritiska tipke, ništa ne dogodi ne zato što VIDI X nije uspješno poslao paket već jer se paket „izgubio“. Međutim to je lako objašnjivo s činjenicom da su svi programi bili testirani u laboratorijskim uvjetima tako da u okolini s više smetnji očekuje se gubitak paketa. Isto tako gubljenje paketa se neće osjetiti u kontinuiranom načinu upravljanja nego bit će više primjetno u diskretnom načinu upravljanja. Isto tako bitno je napomenuti da za vrijeme testiranja „poklopac“ koji prekriva prednju stranu trupla TIOSS robota je bio skinut. U jednom slučaju pokušano je „zatvoriti“ robota, ali „poklopac“ nije do kraja sjeo na svoje mjesto već je nakošeno stajao i oko njega je bila rupa i do 2 centimetra na dijelovima. Međutim to nije imalo utjecaja na upravljanje s robotom. Ta činjenica je iznenađujuća, ali i dalje je preporučeno spojiti antenu na Raspberry Pi računalo i izvući ju iz kućišta ili ostaviti kućište otvorenim.

Slika 6. Izvođenje programa na VIDI X

Zaključak

Ovim radom omogućeno je daljinsko upravljanje robotom TIOSS pomoću VIDI X mikroračunala. U sklopu rada izrađeno je 8 programa odnosno 4 para programskih kodova za upravljanje robotom. Kako bi se upravljalo robotom prvo je potrebno postaviti bežičnu mrežu, najlakše je s pomoću mobitela napraviti hotspot imena TIOSLAN i dogovorene zaporke. Zatim se upali TIOSS te se pokrene program koji se želi koristit. Potom se na VIDI X prenese komplementarni program te se može upravljati s robotom TIOSS.

Ovim radom uspješno su povezani VIDI X i TIOSS te je istraženo nekoliko načina spajanja te upravljanja robotom. Nadogradnja na ovaj rad može ići u više smjerova. Među jednostavnim primjerima je modifikacija inačice programa TCPdiskretno kako umjesto pokretanja akcije robota na određeno vrijeme može se zadati pozicija u koju se ruka ili glava želi postaviti te to omogućiti s pomoću potenciometara. Još jedna ideja je modifikacija jedne od inačica s kontinuiranim upravljanjem da se istovremeno može upravljati s više motora istovremeno, itd. Drugi neki smjer je koristiti već napravljene programe za Raspberry Pi te napisati komplementarne programe za neki drugi mikrokontroler ili android uređaj.

Originalno kada je napravljen robot TIOSS je služio za istraživanje mogućnosti tehnologije i ovaj rad je jedan, u nadi od mnogih, koji se koristio njime za istraživanje mogućnosti tehnologije.

Literatura

[1] Stela Lechpammer, Životni intervju: Branimir Makanec, datum nastanka 22. studenog 2020., 'Otac je ušao u sobu, poljubio me i više ga nikada nisam vidio. Ne znam ni gdje mu je grob' , https://www.vecernji.hr/vijesti/moj-robot-robi-je-68-bio-prava-senzacija-setao-se-zagrebackim-ulicama-i-dobacivao-curama-1447912, pristupljeno 2. rujna 2024

[2] Željka Krmpotić, Robi spašen s otpada, datum nastanka 20. svibnja 2020., Prvi hrvatski robot sreo 'tatu' nakon 60 godina: 'Kakav dar!', https://www.24sata.hr/lifestyle/moj-robot-je-po-zagrebu-hodao-jos-1958-robi-je-bio-pravi-hit-693180, pristupljeno 2. rujna 2024.

[3] Tomić, M. Modernizacija kretanja robota TIOSS. Diplomski rad. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, lipanj 2023.

[4] Alajbeg, J. Modernizacija kretanja robota TIOSS korištenjem robotskog vida i govora. Diplomski rad. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, lipanj 2023.

[5] Brčić, I. Upravljanje rukama humanoidnog robota TIOSS. Diplomski rad. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, lipanj 2024.

[6] Raspberry Pi 4 Model B tehničke specifikacije, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, pristupljeno 2. rujna 2024.

[7] VIDI X mikroračunalo, https://hr.vidi-x.org/sto-je-vidi-x/, pristupljeno 2. rujna 2024.

[8] ESP32 modul, https://www.espressif.com/sites/default/files/documentation/esp32-wrover-b_datasheet_en.pdf, pristupljeno 2. rujana 2024.

[9] UDP protokol, https://en.wikipedia.org/wiki/User_Datagram_Protocol, pristupljeno 2. rujna 2024.

[10] TCP protokol, https://en.wikipedia.org/wiki/User_Datagram_Protocol, pristupljeno 2. rujna 2024.

[11] Python, https://www.python.org/, pristupljeno 2. rujan 2024.

[12] Biblioteka os, https://docs.python.org/3/library/os.html, pristupljeno 2. rujna 2024.

[13] Biblioteka socket, https://docs.python.org/3.10/library/socket.html, pristupljeno 2. rujan 2024.

[14] Biblioteka time, https://docs.python.org/3/library/time.html, pristupljeno 2. rujna 2024.
[15] Biblioteka gpiod, https://pypi.org/project/gpiod/, pristupljeno 2. rujna 2024.
[16] Biblioteka WiFi.h, https://www.arduino.cc/reference/en/libraries/wifi/, pristupljeno 2. rujna 2024.
[17] Biblioteka string.h, https://cplusplus.com/reference/cstring/, pristupljeno 2. rujan 2024.
[18] Biblioteka Adafruit_ILI9341.h, https://github.com/adafruit/Adafruit_ILI9341, pristupljeno 2. rujna 2024.
[19] Biblioteka Adafruit_GFX.h, https://github.com/adafruit/Adafruit-GFX-Library, pristupljeno 2. rujan 2024.

Sažetak

Udaljeno upravljanje robotom TIOSS

Teledirigirani izvršni organ samoorganizirajućeg sustava ili skraćeno TIOSS je prvi hrvatski robot napravljen 60 godina prošlog stoljeća. Od tada je izgubio dosta svojih mogućnosti i dijelova. Kroz nekoliko diplomskih radova restauriran je i može pomicati svojim udovima. Kroz ovaj rad prvi hrvatski robot će se spojiti s novim hrvatskim mikroračunalom, VIDI X, kako bi se njima moglo daljinski upravljati. TIOSS i VIDI X komunicirat će preko bežične mreže. Za komunikaciju se koriste TCP i UDP protokol. Robotom se može upravljati na dva načina: diskretno i kontinuirano. Diskretnim načinom zadaje se robotu instrukcija te se čeka da ju izvrši, dok kod kontinuiranog načina robot se kreće samo dok je jedna od tipki pritisnuta.

Ključne riječi: TIOSS, udaljeno upravljanje, VIDI X, Raspberry Pi, ESP32, WiFi, bežična mreža, UDP, TCP

Summary

Remote control of TIOSS robot

Teledirigirani izvršni organ samoorganizirajućeg sustava or TIOSS for short. It is first croatian robat built in the 60ts. Since then, it has seen a lot of use and with that slowly fell apart. Few students took the opportunity to fix up the TIOSS robot as their master theses. This thesis focuses on remotely controlling the first Croatian robot using a new Croatian microcomputer, VIDI X. The TIOSS robot and VIDI X microcomputer will communicate over the wireless local area network. To enable the communication, UDP and TCP protocols will be used. TIOSS robot can be controlled in two ways, discreetly and continually. When controlled discreetly, the user will send the instruction to the robot and wait until the robot preforms the instruction. On the other hand, while controlled continually, the user will have to hold down the button to control the robot.

Key words: TIOSS, remote control, VIDI X, Raspberry Pi, ESP32, WiFi, WLAN, UDP, TCP