Izrada aplikacije stvarnog vremena za upravljanje mobilnom robotskom platformom

Ričko, Dominik

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij računarstva

Izrada aplikacije stvarnog vremena za upravljanje mobilnom robotskom platformom

Završni rad

Dominik Ričko

Osijek, 2020

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 25.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Dominik Ričko
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4129, 26.09.2019.
OIB studenta:	16052812268
Mentor:	Izv. prof. dr. sc. Damir Blažević
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Izrada aplikacije stvarnog vremena za upravljanje mobilnom robotskom platformom
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	25.09.2020.
Datum potvrde ocjene Odbora:	30.09.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT

IZJAVA O ORIGINALNOSTI RADA

Osijek, 30.09.2020.

Ime i prezime studenta:	Dominik Ričko
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R4129, 26.09.2019.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Izrada aplikacije stvarnog vremena za upravljanje mobilnom robotskom platformom**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Damir Blažević

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Dominik__Ričko_____, OIB: __16052812268_____, student/ica Fakulteta elektrotehnike,
računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski sveučilišni studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _Izrada aplikacije stvarnog vremena za upravljanje mobilnom robotskom platformom**,**
dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).
Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti

b) studentima/icama i djelatnicima/ama ustanove

c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 30.09.2020._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

1. UVOD

Problem upravljanja robotima na daljinu je količina vremena potrebna da robot primi poslanu naredbu i kašnjenje povratne informacije. U ovom radu se na robotu definiraju načini kretanja, opisuje proces izrade aplikacije koja će uspostavljati vezu sa robotom, upravljati robotom, prikazati video prijenos kamere, te postići što manje kašnjenje pri komunikaciji računala i robota.

U drugom poglavlju ovog rada se opisuje što su mobilne robotske platforme, na koje vrste se dijele, te koja je njihova svrha.

U trećem poglavlju je objašnjeno što su to sustavi stvarnog vremena, koje podjele tih sustava postoje i kako ih razlikujemo.

U četvrtom poglavlju se određuje za koji robot se izrađuje aplikacija, prikazuje se način konfiguriranja robota, opisuje proces izrade aplikacije za upravljanje robotom, te objašnjava zašto se ta aplikacija smatra sustavom stvarnog vremena.

1.1. Zadatak završnog rada

U ovom rada prikazat će se potrebni koraci za izradu *Windows* aplikacije za upravljanje mobilnom robotskom platformom putem bežične mreže. Pri tome, cilj je postići što manje vremensko kašnjenje od izdavanja naredbe na upravljačkoj aplikaciji do izvršenja naredbe na robotu.

2. MOBILNA ROBOTSKA PLATFORMA

Mobilne robotske platforme su računalni sustavi koji se mogu pomicati u prostoru i djelovati na stvarni svijet. Tipično se sastoje od upravljačke ploče (npr. *Raspberry Pi* ili *Arduino*), pokretnih dijelova i aktuatora (električni motori koji pokreću te dijelove).

Mobilne robotske platforme se mogu klasificirati prema načinu kretanja:

- roboti s kretanjem uz pomoć propelera (*dronovi*),
- roboti koji koriste zglobne udove (eng. *articulated limbs*) [1],
- i roboti na kotačima.

Dronovi su najčešće sastavljeni sa četiri propelera, što im omogućava kretanje u zraku. Dva propelera se vrte u jednom smjeru, a druga dva u suprotnom smjeru, time se izjednačava moment inercije *drona*, što ga drži na konstantni kut (misli se na eng. *Yaw*). Smanjenjem brzine vrtnje jednog propelera se *dron* kreće u smjeru „slabijeg“ propelera, a rotaciju vrši mijenjanjem smjerova vrtnje propelera.

Dronovi se koriste za isporuku dostave, u agrikulturi kao način navodnjavanja biljaka, snimanja za filmsku industriju i mjerenja raznih fizikalnih veličina u nedostupnim ili opasnim područjima.

Roboti s zglobnim udovima imaju veću pokretljivost od robota sa kotačima, te najčešće imitiraju pokrete životinja, ljudi i insekata [1].

Roboti na kotačima mogu se sastojati od dva, četiri ili pak većeg parnog broja kotača. Kretanje prema naprijed ili nazad je univerzalan za ovakve robote, neovisno o broju kotača. Roboti na dva kotača se jedino mogu kretati na ravnom pravcu, ali zato se mogu rotirati okretanjem kotača u međusobno suprotnim smjerovima. Roboti na četiri kotača se ne mogu rotirati na jednom mjestu, već kroz kretanje pod nekim kutom prednjih kotača.

Primjeri uporabe robota na kotačima su:

- Samovozeći automobili koji senzorima vide okolinu i oprezno se kreću sa što manjim ljudskim upravljanjem i koriste se za transport ljudi i robe [2],
- autonomni mobilni robot *Roomba* čiji mu je cilj čišćenje prostorija [3],

- *AlphaBot* koji se koristi u svrhu edukacije učenika i studenata o mobilnim robotima.

Osim načina kretanja, roboti se dijele prema načinu navigacije [4].

- Autonomno,
- upravljanje na daljinu.

Autonomni roboti sadrže program koji diktira radnje i kretanja robota. U autonomnu navigaciju spadaju programi kojima roboti prate liniju koja se nalazi na podu, programi za provođenje nasumičnih pokreta, i pametniji programi koji koristeći senzore prepoznaju okolinu, te robota dovode do određenog cilja, bilo to putem prethodno navedene putanje ili pronađene putanje.

Upravljanje na daljinu se postiže bežičnim spajanjem na robota, te slanjem naredbi. Zato se izrađuju upravljačke aplikacije, ili konzole, kojima korisnik može očitati vrijednosti senzora na robotu, vidjeti što robot vidi, te upravljati robotom putem korisničkog sučelja upravljačke aplikacije.

3. SUSTAVI STVARNOG VREMENA I NJIHOVE REALIZACIJE

Sustav stvarnog vremena je skup procesa koji se istovremeno izvode, međusobno djeluju i to u skladu s određenim vremenskim zahtjevima. Vremenski razmak u kojem zahtjev mora biti obrađen određuje okolina, tj. ovisno čime sustav upravlja. [5]

Uz to, sustavima stvarnog vremena su bitne još dvije stavke:

- Logička ispravnost
- i vremenska ispravnost.

Sustav koji na osnovu ulaza prema zadanim postupcima za rješavanje problem izračuna očekivane rezultate se smatra logički ispravnim. Drugim riječima, logički ispravni sustav ne smije raditi krive proračune. [6]

Osim ispravnog rezultata, potrebno je pripremiti taj rezultat u pravom trenutku. Moguće je da izračuni za neke ulaze se izvrše u manjem vremenu nego očekivanom ili zadanom. Ako se ti izračuni pošalju ranije od predviđenog trenutka, sustav može snositi posljedice. Stoga je potrebno odgoditi slanje rezultata na predviđeni trenutak. Sustav koji osigurava da se svi zahtjevi šalju nakon jednakog vremena izvođenja je vremenski ispravan. [6]

Za sustave stvarnog vremena je važnija vremenska ispravnost od logičke ispravnosti, te zbog toga se sustavi klasificiraju prema posljedicama nepoštivanja zadanih vremenskih rokova. Klasifikacije sustava su, sustavi sa strogim vremenskim zahtjevima (eng. Hard real-time systems), sustavi sa ublaženim vremenskim zahtjevima (eng. Soft real-time systems) i postojani sustavi stvarnog vremena (eng. Firm real-time systems) [5]–[7]. Opisi pojedinih sustava dani su u sljedećim podpoglavljima.

3.1. Sustavi sa strogim vremenskim zahtjevima

Sustavi sa strogim vremenskim zahtjevima se koriste kada je potrebno obraditi događaj u zadanom roku. Ukoliko se događaj ne obradi u roku, dogode se teške posljedice na sustav ili okolinu sustava. Primjer ovakvih sustava je pejsmejker (eng. *Pacemaker)*. Pejsmejkerova zadaća je jednostavna,

stvaranje električnih impulsa radi stimulacije mišića srca s ciljem održavanja cirkulacije krvi kroz krvožilni sustav. Samo jedan impuls u krivom trenutku ili nedostatak impulsa u nekom trenutku predstavlja rizik ljudskom životu.

3.2. **Sustavi sa ublaženim vremenskim zahtjevima**

Sustavi na kojima propust roka ne uzrokuje pad sustava, korisnost odziva nakon propuštenog roka opada s vremenom, zbog čega opada kvaliteta usluge koju sustav pruža. Kao primjer ovakvog sustava se navodi grijanje. U jednakim intervalima se mjeri temperatura prostorije. Ako je temperatura prostorije niža od željene temperature, radijatori se uključuju, u suprotnom se isključuju ili ostaju u isključenom stanju. Ako se preskoči jedno mjerenje kada je na granici željene temperature prostorije, sustav će nastaviti grijati prostoriju sve do sljedećeg mjerenje. Time se dobiva veća temperatura od željene, čime opada kvaliteta usluge grijanja, ali ne dolazi do raspada sustava.

3.3. **Postojani sustavi stvarnog vremena**

Sustav koji neće u potpunosti zakazati ako se dogodi pokoji propust roka, ali kvaliteta usluge koju sustav pruža može pasti. Sustav nema koristi od odziva koji bi se dogodio nakon zadanog roka. Primjer ovakvog sustava je video igra. Cilj sustava crtanja grafike igre je crtanje dovoljno okvira u vremenskom periodu (često je to šestdeset okvira po sekundi). Nema smisla prikaza okvira nacrtan nakon vremenskog roka za taj okvir, jer time se dobiva kašnjenje grafike u usporedbi s ostalim elementima igre. Stoga, zakašnjeli okviri nisu od koristi. Kvaliteta pada na način da se zakašnjeli okviri ne prikazuju, što uzrokuje pad broja okvira po sekundi.

4. PRAKTIČNI DIO RADA

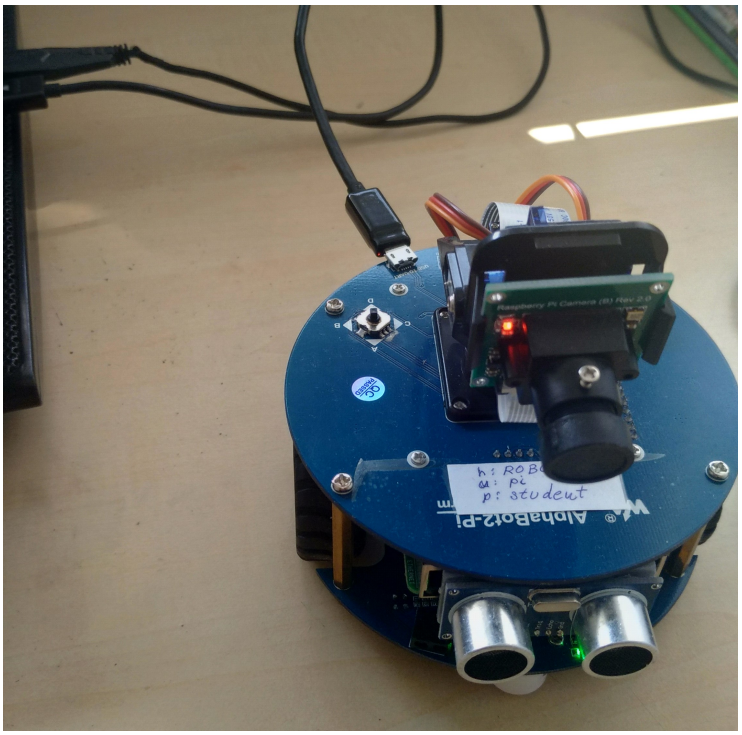
Aplikacija se izrađuje u *Microsoft Visual Studio Community 2019*; besplatni alat za izgradnju modernih aplikacija za Windows operativnog sustava [8]. Aplikacija se izrađuje kao Windows Forma pomoću *C#* jezika, te na mobilnoj robotskoj platformi se koristi *Python* programski jezik. Povezivanje računala s robotom se postiže pomoću *SSH* protokola.

4.1. Konfiguriranje mobilne robotske platforme

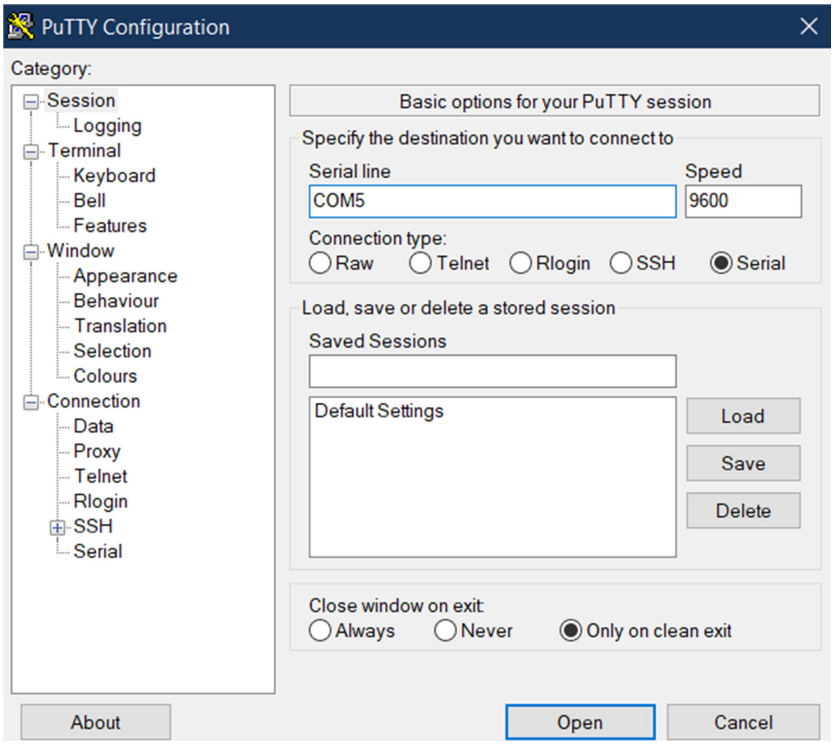
Kako bi se postigla mogućnost upravljanja robotom, potrebno je povezati robot na bežičnu mrežu i instalirati programe potrebne robotu za rad aktuatora kojima se pomiče.

Spajanje robota na bežičnu mrežu

Za konfiguriranje uređaja potrebno je uspostaviti žičanu serijsku komunikaciju s uređajem. Za uspješnu serijsku komunikaciju s robotom, potrebno je spojiti računalo i robota *USB* kabelom kao na Slika 4.1, te na računalu instalirati pogonski program [9]. Softverska komunikacija se uspostavlja uz pomoć PuTTY programa prikazanog na Slika 4.2.

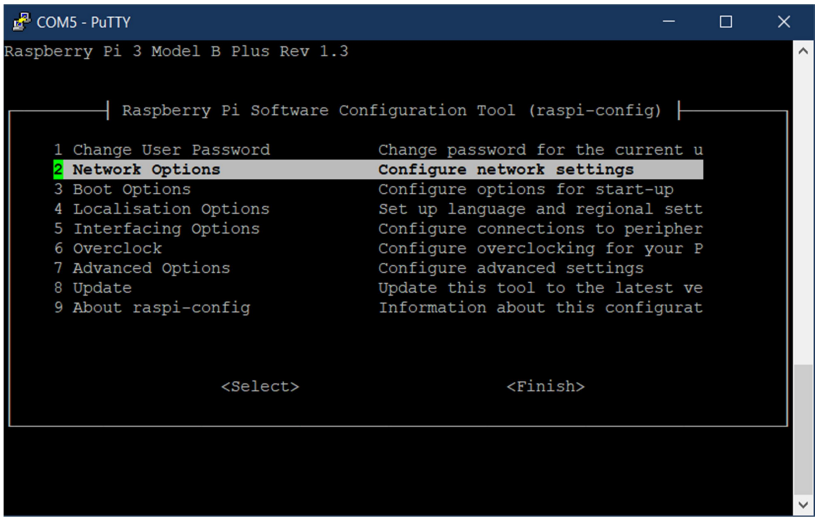


Slika 4.1 *Fotografija robota spojenog na računala USB kabelom*



Slika 4.2 *PuTTY ekran za spajanje*

Nakon otvaranja veze potrebno je pritisnuti nekoliko tipki na tipkovnici kao naznaka robotu da računalo želi uspostaviti serijsku komunikaciju, time se pojavljuje zaslon za prijavu i potrebno je upisati korisničko ime i lozinku za uspješno povezivanje.



Slika 4.3 *Raspi-config meni*

Naredbom *sudo raspi-config* se otvara konfiguracijski meni *Raspberry*-a koji je prikazan na Slika 4.3. Odabirom postavki mreža (engl. *Network Options)* je moguće definirati naziv i lozinku bežične mreže na koju će se *Raspberry* spajati. Bitno je napomenuti da se *Raspberry Pi* ne može povezivati na mreže koje se nalaze u 5GHz frekvencijskim kanalima, samo 2.4GHz.

Instalacija potrebnih biblioteka i skripti

Nakon uspješnog povezivanja robota na mrežu dolazi instalacija programa. Svrha tih programa je omogućiti robotu upravljanje kotačima i čitanje vrijednosti sa senzora. Za te programe su potrebne sljedeće biblioteke naznačene na Slika 4.4.

```
sudo apt-get update
sudo apt-get install ttf-wqy-zenhei
sudo apt-get install python-pip
sudo pip install RPi.GPIO
sudo pip install spidev
sudo apt-get install python-smbus
sudo apt-get install python-serial
sudo apt-get install subversion libjpeg8-dev
sudo pip install rpi_ws281x
```

Slika 4.4 *Popis naredbi za instalaciju biblioteka [10]*

Skripte koje su potrebne za upravljanje robota putem aplikacije na računalu se nalaze u *AlphaBot 2* programu. Preuzimanje i raspakiranje demonstracijskih programa se odvija pozivom naredbi na Slika 4.5.

```
Cd
wget https://www.waveshare.com/w/upload/7/74/AlphaBot2.tar.gz
tar zxvf AlphaBot2.tar.gz
```

Slika 4.5 *Naredbe za preuzimanje programa za AlphaBot 2 [10]*

Skripte za pokretanje robota i kamere

Pokretanje kotača i rotaciju kamere je moguće upravljati uz pomoć dviju *python* skripti koje se nalaze u prethodno skinutom programu. *AlphaBot2.py*, koji sadrži metode za upravljanje kotačima robota, i *PCA9685.py* koji sadrži metode za upravljanje rotacijom kamere na robotu.

Za korištenje tih metodi, potrebne je pokrenuti *python* konzolu, učitati spomenute skripte, te instancirati klase koje se nalaze unutar tih skripti. (Slika 4.6)

```
cd AlphaBot2/python/
python –i
import AlphaBot2
import PCA9685
move = AlphaBot2.AlphaBot2()
camera = PCA9685.PCA9685(0x40, debug = False)
```

Slika 4.6 *Popis python naredbi za inicijalizaciju servo motora.*

Metode klase *AlphaBot2* postavljaju bit-vrijednosti na izlaznim registrima *Raspberrya* pomoću kojih se upravljaju ostale komponente robota. Zbog toga, metode ne zaustavljaju *python* konzolu dok se robot kreče, što omogućuje poziv drugih metoda. Metode koje se koriste za kretanje robota su navedene na Slika 4.7.

```
move.forward()
move.backward()
move.left()
move.right()
move.stop()
```

Slika 4.7 *Popis python naredbi za pokretanje robota*

Za postavljanje rotacije kamere se koristi metoda *setServoPulse* koja ima dva parametra. Prvi parametar predstavlja indeks motora, nula predstavlja motor koji se rotira u smjeru lijevo-desno (eng. Yaw), a jedinica predstavlja motor koji se može rotirati gore-dolje (eng. Pitch). Drugi parametar predstavlja vrijednost rotacije koja se preko zadane frekvencije rada pretvara u postotak trajanja visoke naponske razine u modulaciji širine impulsa (eng. Pulse-width modulation).

Kao zadnji korak, potrebno je uključiti *IP* kameru da snima i prenosi videosnimku uređajima koji se priključe kao gledaoci video prijenosa. (Slika 4.8)

```
cd AlphaBot2/mjpg-streamer/mjpg-streamer-experimental/
nohup ./start.sh &
```

Slika 4.8 *Naredbe za uključivanje kamere*

Korištenjem *nohup linux* naredbe i operatora *&* na kraju naredbe se pokreće skripta bez pisanja odziva programa na konzolu, tj. program se vrši u pozadini.

4.2. **Izrada aplikacije za upravljanje robota**

Aplikacija se izrađuje u *Microsoft Visual Studio Community 2019*; besplatni alat za izgradnju modernih aplikacija za Windows operativnog sustava [8]. Aplikacija se izrađuje kao Windows Forma pomoću *C#* jezika, te na mobilnoj robotskoj platformi se koristi *Python* programski jezik. Povezivanje računala s robotom se postiže pomoću *SSH* protokola.

Aplikacija mora omogućiti uspostavu veze s robotom i upravljanje robota u stvarnom vremenu, stoga se aplikacija sastoji od dva prozora.

Prvi prozor čija je uloga pronalazak uređaja na mreži i uspostava veze s odabranim uređajem. Kako pronalaženje uređaja na mreži nije bitno za temu ovog završnog, rješenje se nalazi među prilozima. Jedna od alternativi je ručno upisati IP adresu robota, ali je pri tome potrebno koristiti metodu *System.Net.Dns.GetHostEntry* jer konstruktoru klase *SshClient* se predaje ime uređaja, a ne IP adresa. Na Slika 4.9 je prikazano kako koristiti dobiti ime uređaja.

```
            IPHostEntry hostEntry = Dns.GetHostEntry(ip);
            string name;
            if (hostEntry != null)
            {
               name = hostEntry.HostName;
            }
```

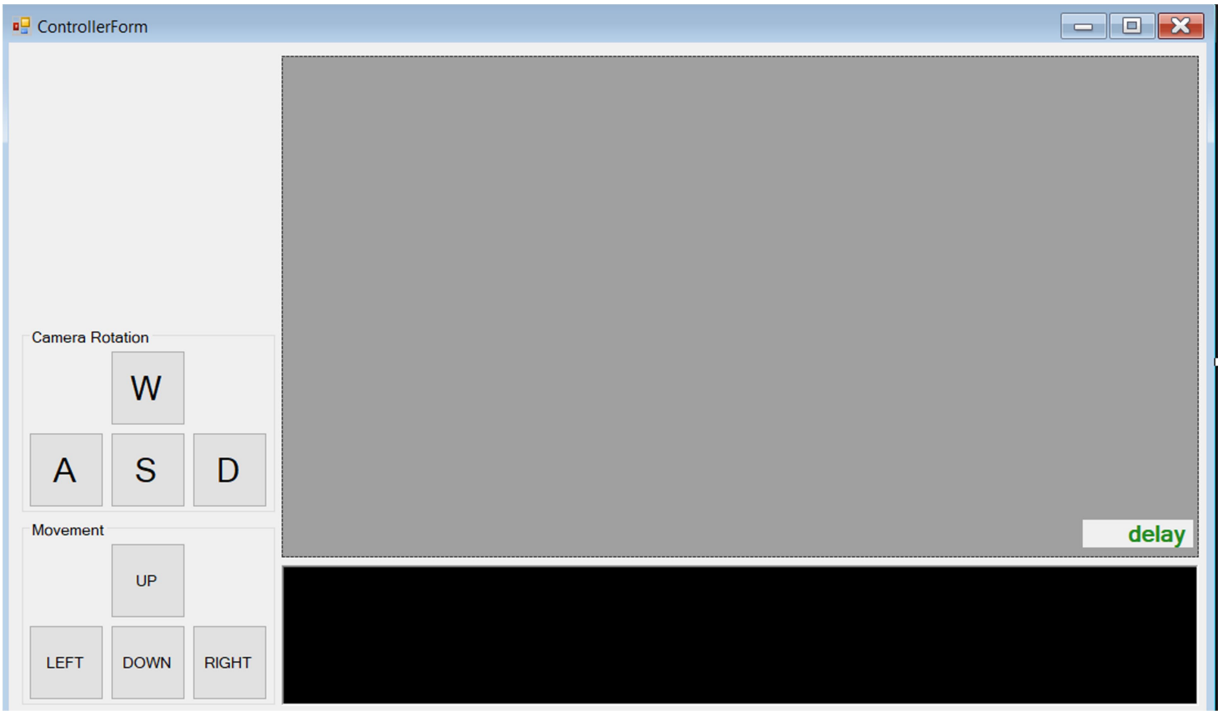
Slika 4.9 *Kod za identifikaciju uređaja*

Za povezivanje su potrebne tri informacije. Naziv uređaja, korisničko ime i lozinka. Predajom tih informacija u konstruktor klase *SshClient* se stvara nova instanca, pozivom metode *Connect* se uspostavlja veza. U slučaju uspješne uspostave veze se otvara novi prozor kojim je moguće upravljati robotom. (Slika 4.10)

```
        private void Connect()
        {
            NetworkDevice host = (NetworkDevice)lstBox_devices.SelectedItem;
            try
            {
                sshClient = new SshClient(host.Name, txtBox_username.Text, txtBox_password.Text);
                sshClient.Connect();
            }
            catch(ArgumentException)
            {
                MessageBox.Show("Username is required.");
                return;
            }
            catch (Exception exception)
            {
                MessageBox.Show(exception.Message);
                return;
            }
            controllerForm = new ControllerForm(sshClient, host);
            btn_Connect.Enabled = false;
            controllerForm.FormClosing += ControllerForm_FormClosing;
            controllerForm.Show();
        }
```

Slika 4.10 *Definicija metode za spajanje*

Prozor za upravljanje je prikazan na Slika 4.11. Vizualno se sastoji od pet dijelova: tipke za upravljanje rotacijom kamere, tipke za kretanje robota, prostor za prikaz video snimke kamere, prikaz vremenskog kašnjenja naredbi, te prostor za ispis naredbi poslane putem *SSH* protokola.



Slika 4.11 *Prozor za upravljanje*

Slanje naredbi i primanje poruka

Za slanje naredbi, potrebno je pozvati metodu objekta *SshClient* klase, *CreateShellStream* koja ima šest parametra: naziv konzole koju stvara, širinu i visinu izraženu preko broj stupaca i redova, širinu i visinu izraženu u pikselima, te veličinu spremnika znakova. Metoda stvara novi objekt tipa *ShellStream* koji se koristi za slanje naredbi povezanom uređaju. (Slika 4.12)

```
public SSHConsole(SshClient sshClient, TextBoxBase outputTextBox, Control delayDisplay)
      {
         this.sshClient = sshClient;
         this.outputBox = outputTextBox;
         this.delayDisplay = delayDisplay;
         this.delayMeasurer = new Stopwatch();
         sshStream = sshClient.CreateShellStream("dumb", 80, 24, (uint)outputTextBox.Width,
(uint)outputTextBox.Height, outputTextBox.MaxLength);
```

```
        sshStream.DataReceived += SshStream_DataReceived;
    }
```

Slika 4.12 *Definicija konstruktora klase SSHConsole*

Za prikaz naredbi i kojekakvih poruka koje bi se prikazale tijekom izvođenja raznoraznih programa na uređaju, potrebno je ažurirati prostor za prikaz naredbi na svaki događaj primanja podataka. Ali, kako se primanje podatak drugoj dretvi, a ne glavnoj dretvi za crtanje grafike aplikacije, potrebno je napraviti to ažuriranje koristeći delegata. (Slika 4.13)

```
    private delegate void delegateUpdate();
    private Stopwatch delayMeasurer;
    private void SshStream_DataReceived(object sender, ShellDataEventArgs e)
    {
        if (outputBox.InvokeRequired)
        {
            outputBox.Invoke(new delegateUpdate(UpdateOutput));
        }
        delay = delayMeasurer.ElapsedMilliseconds;
    }
    public void UpdateOutput()
    {
        if (sshStream.DataAvailable)
        {
            try
            {
                StreamReader reader = new StreamReader(sshStream);
                string outputFrom = reader.ReadToEnd();
                outputBox.AppendText(outputFrom);
                delayDisplay.Text = delay.ToString() + " ms";
            }
            catch (Exception exception)
            {
                MessageBox.Show(exception.Message);
            }
            outputBox.SelectionStart = outputBox.Text.Length;
            outputBox.ScrollToCaret();
        }
    }
```

Slika 4.13 *Dio koda potreban za ažuriranje prostora za prikaz naredbi na prozoru*

Slanje naredbi se odvija pozivom metode na Slika 4.14, ali i time se započinje štopanje vremena kojim se mjeri vremensko kašnjenje od slanja naredbi do primanja potvrde o poslanoj naredbi.

```
public void WriteLine(string line)
      {
         Try
         {
            delayMeasurer.Restart();
            sshStream.WriteLine(line);
         }
         catch (Exception)
         {
            OnSSHException(new EventArgs());
         }
      }
```

Slika 4.14 *Definicija metode WriteLine klase SshConsole*

Problem video prijenosa

AForge je *C#* biblioteka dizajnirana za programere i istraživače u polju računalnog pogleda i umjetne inteligencije, te se koristi za procesiranje slika, neuronske mreže, genetičke algoritme, neizrazitu logiku, strojno učenje i robotiku. [11]

U aplikaciji se koristi biblioteka *AForge.Video* zbog klase *MJPEGStream* koja pruža *NewFrame* događaj i metode za započinjanje i zaustavljanje video prijenosa od strane računala, ili drugim riječima, signalizira *IP* kameri da se računalo želi priključiti kao gledaoc video prijenosa. U konstruktor *MJPEGStream*-a se upisuje *HTTP* putanja odakle odašilje *IP* kamera, te se postavlja metoda koja će se vršiti kada novi okvir video prijenosa dođe na aplikaciju. (Slika 4.15)

```
      public InitializeVideoBox()
      {
         pBox_video.SizeMode = PictureBoxSizeMode.StretchImage;
         camera = new MJPEGStream("http://" + device.IP.ToString() + ":8080/?action=stream");
         camera.NewFrame += new NewFrameEventHandler(Video_newFrame);
         camera.Start();
      }

     private void Video_newFrame(object sender, NewFrameEventArgs e)
      {
```

```
        pBox_video.Image = (Bitmap)e.Frame.Clone();
    }
```

Slika 4.15 *Dio koda kojim se postavlja prostor za prikaz video strujanja IP kamere*

Upravljanje pokretima

Sljedeće rješenje omogućuje upravljanje robota pomoću tipki na prozoru i tipki na tipkovnici. U novoj klasi *AlphaBot2Controller* su definirane metode na Slika 4.16 i Slika 4.17, koje uz pomoć enumeratora *ControlType*, te metode *DetermineControlType* određuju u koji popis izvođenja pokreta će se predana tipka unijeti ili obrisati.

```
    public void RegisterPressedKey(Keys whichKey)
    {
        ControlType controlType = DetermineControlType(whichKey);
        if(controlType == ControlType.CameraXControl)
        {
            if (cameraXPriority.Contains(whichKey)) return;
            cameraXPriority.AddFirst(whichKey);
        }
        else if (controlType == ControlType.CameraYControl)
        {
            if (cameraYPriority.Contains(whichKey)) return;
            cameraYPriority.AddFirst(whichKey);
        }
        else if (controlType == ControlType.MovementControl)
        {
            if (movementPriority.Contains(whichKey)) return;
            movementPriority.AddFirst(whichKey);
            ProcessMovement();
        }
    }
```

Slika 4.16 *Definicija metode registracije pritisnute tipke*

```
public void UnregisterPressedKey(Keys whichKey)
    {
        ControlType controlType = DetermineControlType(whichKey);
        if (controlType == ControlType.CameraXControl)
        {
            if (!cameraXPriority.Contains(whichKey)) return;
            cameraXPriority.Remove(whichKey);
```

```
        }
        else if (controlType == ControlType.CameraYControl)
        {
            if (!cameraYPriority.Contains(whichKey)) return;
            cameraYPriority.Remove(whichKey);
        }
        else if (controlType == ControlType.MovementControl)
        {
            if (!movementPriority.Contains(whichKey)) return;
            movementPriority.Remove(whichKey);
            ProcessMovement();
        }
    }
```

Slika 4.17 *Definicija metode za brisanje tipke sa popisa izvođenja naredbi*

Metoda *ProcessMovement* se koristi za slanje naredbi robotu ovisno o pritisnutoj tipki. Popisi prioriteta izvođenja naredbe postoje zbog mogućnosti pritiska više tipki na tipkovnici. Iako je više tipki pritisnuto, uvijek postoji redoslijed pritisnutih tipki. Pravilo je da svaka zadnje pritisnuta tipka se prva izvodi, stoga povezani popis imitira stog, ali uz jednu razliku, a to je da je moguće obrisati element koji se nalazi negdje u sredini popisa.

Iz popisa prioriteta se uzima prva tipka, ako element povezanog popisa ne sadrži tipku, šalje se naredba za zaustavljanje kretnje. U suprotnom slučaju se gleda koja je to tipka i šalje se pripadajuća naredba za tu tipku. Ova metoda se poziva kad god je neka tipka kretanja pritisnuta ili otpuštena. (Slika 4.18)

```
    Private void ProcessMovement()
    {
        LinkedListNode<Keys> currentKey = movementPriority.First;
        if (currentKey == null)
        {
            console.WriteLine("move.stop()");
            return;
        }
        Keys currentCommand = currentKey.Value;
        switch (currentCommand)
        {
            case Keys.Up: console.WriteLine("move.forward()"); break;
            case Keys.Left: console.WriteLine("move.left()"); break;
            case Keys.Down: console.WriteLine("move.backward()"); break;
            case Keys.Right: console.WriteLine("move.right()"); break;
```

```
            default: console.WriteLine("move.stop()"); break;
        }
    }
```

Slika 4.18 *Definicija metode za slanje naredbi pokretanja robota*

Upravljanje rotacijom kamere se odvija uz pomoć dva povezana popisa, i tajmera. Stvara se jedan povezani popis za svaki smjer rotacije koji je moguć na kameri, u ovom slučaju dva. Tajmer se koristi zbog metode rotiranja na robotu. Metoda *setServoPulse* kao parametar uzima vrijednost koja određuje apsolutni kut na koji će se kamera pomaknuti. Stoga, potrebno je postepeno mijenjati vrijednost *pulse* u jednakim vremenskim intervalima da se dobije sporija, ali preciznija rotacija.

Slanje naredbi za upravljanje rotacijom kamere se odvija svaki put kada tajmer odbroji. (Slika 4.19)

```
private void onTimerElapsed(Object source, ElapsedEventArgs e)
{
    try
    {
        ProcessCameraKey();
    }
    catch (Exception) { }
}
```

Slika 4.19 *Definicija metode koja se izvodi pri okidanje tajmera*

Na isti način kao i za pokretanje robota, uzimaju se prve tipke sa popisa, provjeravaju kojim naredbama pripadaju, te se šalju. Naime ne postoji metoda za zaustavljanje rotacije kamere, već će motori na robotu konstantno držati tu rotaciju kamere. Prije slanja naredbe se provjerava da li je vrijednost *pulse* parametra za pojedini smjer prešao neku definiranu granicu, u tom slučaju se vrijednost *pulse* parametra postavlja na vrijednost granice. (Slika 4.20)

```
Private void ProcessCameraKey()
{
    LinkedListNode<Keys> currentKey = cameraYPriority.First;
    if (currentKey != null)
    {
        Keys currentCommand = currentKey.Value;
        switch (currentCommand)
        {
            case Keys.W:
```

```
                cameraChangeY = true;
                currentY -= 10;
                break;
            case Keys.S:
                cameraChangeY = true;
                currentY += 10;
                break;
        }
    }
    currentKey = cameraXPriority.First;
    if(currentKey != null)
    {
        Keys currentCommand = currentKey.Value;
        switch (currentCommand)
        {
            case Keys.A:
                cameraChangeX = true;
                currentX += 10;
                break;
            case Keys.D:
                cameraChangeX = true;
                currentX -= 10;
                break;
        }
    }
    if (cameraChangeX)
    {
        cameraChangeX = false;
        if (currentX > limitX2) currentX = limitX2;
        if (currentX < limitX1) currentX = limitX1;
        cameraXCommandBuilder.AppendFormat("camera.setServoPulse(0,{0})",
currentX.ToString());
        console.WriteLine(cameraXCommandBuilder.ToString());
        cameraXCommandBuilder.Clear();
    }
    if (cameraChangeY)
    {
        cameraChangeY = false;
        if (currentY > limitY2) currentY = limitY1;
        if (currentY < limitY1) currentY = limitY1;
        cameraYCommandBuilder.AppendFormat("camera.setServoPulse(1,                {0})",
currentY.ToString());
        console.WriteLine(cameraYCommandBuilder.ToString());
        cameraYCommandBuilder.Clear();
    }
}
```

Slika 4.20 *Definicija metode za upravljanje kamerom*

Kako bi sve te naredbe imale nekog utjecaja, potrebno je uključiti *python* konzolu na robotu i učitati i instancirati klase biblioteka *PCA9685.py* i *AlphaBot2.py*. To se odvija slanjem naredbi iz 4.1 podpoglavlja. (Slika 4.21)

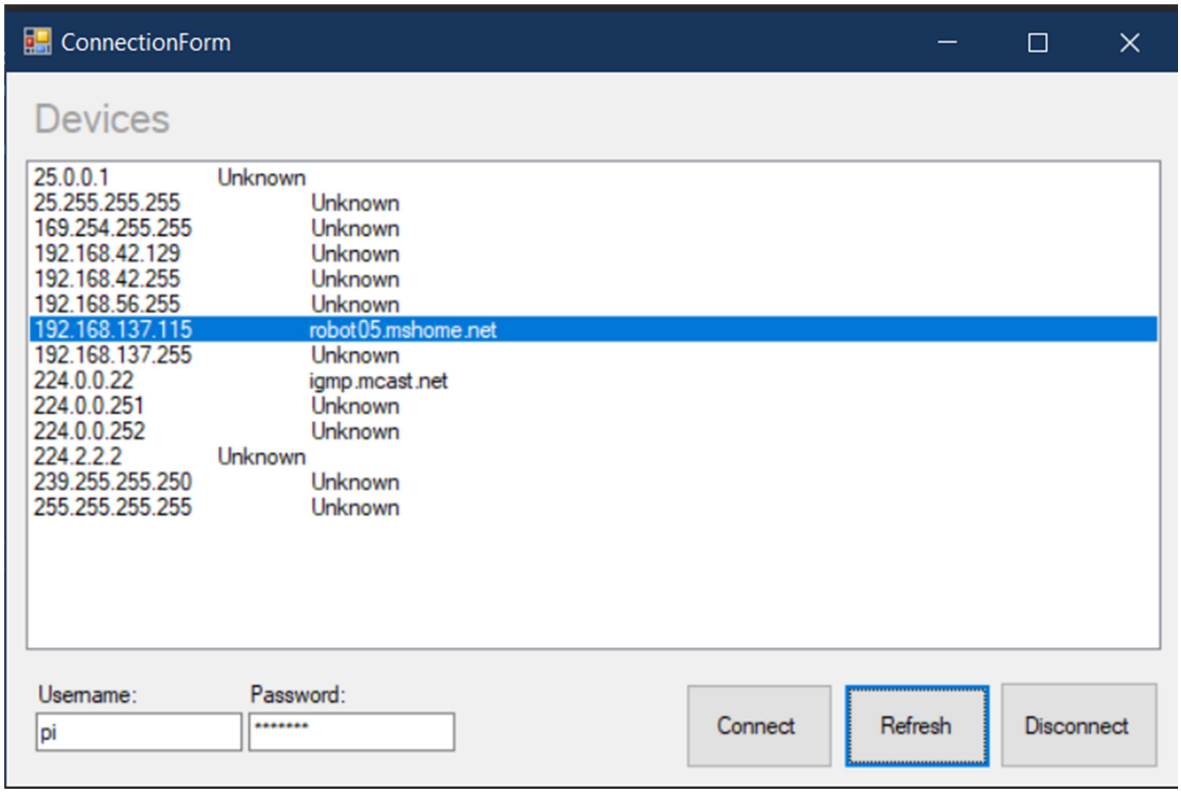
```
public AlphaBot2Controller(SSHConsole console)
    {
        cameraXPriority = new LinkedList<Keys>();
        cameraYPriority = new LinkedList<Keys>();
        movementPriority = new LinkedList<Keys>();
        cameraControlTimer = new System.Timers.Timer();
        cameraXCommandBuilder = new StringBuilder();
        cameraYCommandBuilder = new StringBuilder();
        this.console = console;
        console.WriteLine("cd AlphaBot2/python/");
        console.WriteLine("python -i");
        console.WriteLine("import AlphaBot2");
        console.WriteLine("import PCA9685");
        console.WriteLine("move = AlphaBot2.AlphaBot2()");
        console.WriteLine("camera = PCA9685.PCA9685(0x40, debug = False)");
        console.WriteLine("camera.setPWMFreq(50)");
        cameraControlTimer.Interval = 20;
        cameraControlTimer.Elapsed += onTimerElapsed;
        cameraControlTimer.Start();
    }
```

Slika 4.21 *Definicija konstruktora klase AlphaBot2Controller*

4.3. **Gotova aplikacija i primjena**

Na prozoru za spajanje se vidi, uređaj pod imenom *robot05.mshome.net* time se potvrđuje ispravan rad pronalaženja uređaja na mreži. Zbog *ToString* metode klase *NetworkDevice* se pojavljuje nepravilni prikaz uređaja na popisu. *ToString* metoda ubacuje dva tab razmaka između *IP* adrese i imena uređaja. U slučaju „kraćih“ IP adresa se pomiču imena uređaja prema lijevo. (Slika 4.22)

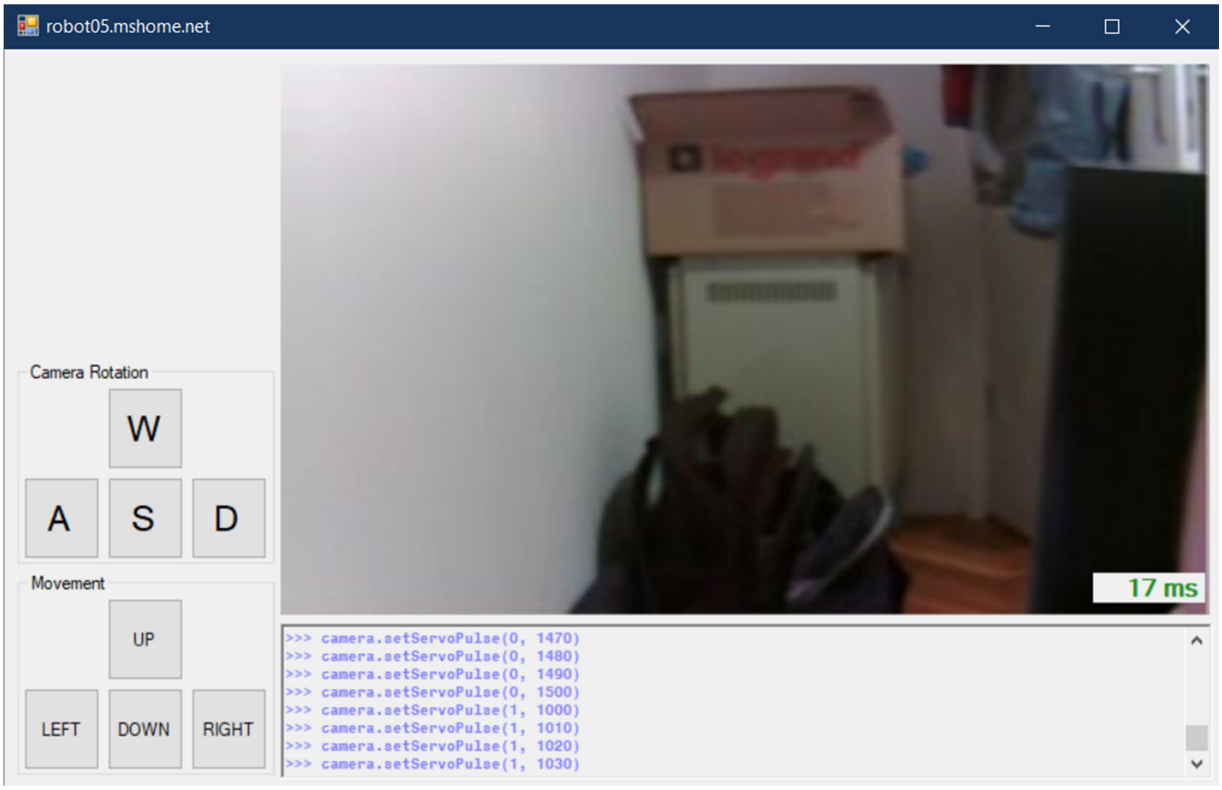
Uz to, ne pripadaju sve *IP* adrese stvarnim uređajima, nego adrese koje su softverski dodane iz kojekakvih razloga u *arp* skladište. Ovo je rješivo ispitivanjem *IP* adresa ping naredbom u pozadini ili postavljanje tajmera na *DNS* zahtjev pri kojem se dobivaju imena tih uređaja. Ako tajmer završi prije odgovora zahtjeva, uređaj se ne dodaje na popis.



Slika 4.22 *Prozor za spajanje na robota*

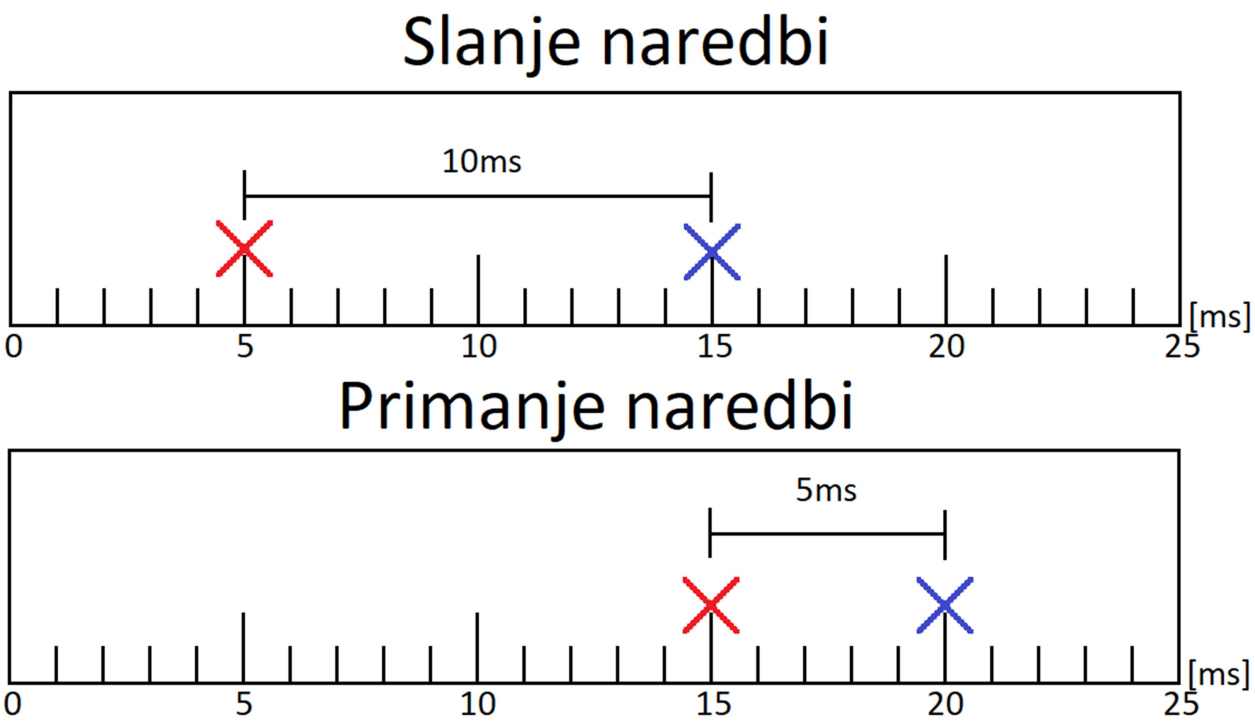
Nakon spajanja na robota je, prema Slika 4.23, vidljivo kako kamera radi. Pritiskanjem tipki za pokretanje se vidi slanje naredbi na konzoli, te na mjerilu kašnjenja naredbe.

Kašnjenje kamere se procjenjuje empirijskom metodom gdje se uz pomoć štoperice određuje količina vremena kada se bačeni objekt zvukom čuje da je pao na robotu vidljivoj površini, i kada se taj događaj vidi na prozoru aplikacije. Procijenjeno je da je kašnjenje video prijenosa između sto pedeset milisekundi i dvjesto milisekundi.



Slika 4.23 *Prozor za upravljanje robota*

Očekuje se od aplikacije da količina vremena između trenutne poslane naredbe i prethodne poslane naredbe bude jednaka količini vremena između trenutne primljene naredbe i prethodne primljene naredbe. U slučaju da se dogodi smetnja pri prijenosu koja će odgoditi poslanu naredbu, te količine vremena poslanih i primljenih naredbi više neće biti jednake, te se posljedica manifestira na način da se robot neće micati za onoliko vremena koliko je bila pritisnuta tipka za kretanje, nego nešto kraće. Slično vrijedi i da je odgođena naredba zaustavljanja kretnje. Tipka je bila pritisnuta neko kraće vrijeme, ali zbog smetnje će robot nastaviti kretanje neko duže vrijeme od trajanja pritiskanja tipke. (Slika 4.24)



Slika 4.24 *Prikaz smanjenja trajanja naredbe zbog nejednakog kašnjenja naredbi pri prijenosu*

Posljedica propuštanja roka isporuke naredbe je gubitak preciznosti kretnje. S većim kašnjenjem naredbe, veća je nepreciznost kretnje i time se aplikacija smatra sustavom sa ublaženim vremenskim zahtjevima.

Promatranjem mjerila vremenskog kašnjenja naredbi pri slanju naredbi se zaključuje da kašnjenje na neopterećenoj mreži varira od pet do trideset milisekundi, s tim da pri uspostavi veze sa robotom kašnjenje doseže do devetsto milisekundi.

5. ZAKLJUČAK

Dovršetkom ovog rada, izrađena je aplikacija koja omogućuje spajanje i kontrolu mobilnog robota, *AlphaBot 2*. Aplikacija otkriva uređaje na mreži te ih prikazuje u popisu iz kojeg se bira uređaj na koji će se aplikacija spojiti. Nakon spajanja se otvara prozor za upravljanje kojim, preko tipki na prozoru ili tipki na tipkovnici se šalju naredbe kretanja robota ili naredbe za rotiranje kamere. Na prozoru se prikazuje video prijenos kamere sa minimalnim kašnjenjem prijenosa koje je procijenjeno između sto pedeset i dvjesto milisekundi.

Problem aplikacije je to što se pri prekidu veze aplikacija zamrzne prije zatvaranja prozora za upravljanje robota. Naime, to je problem biblioteke koja se koristi za spajanje. Ako postoji paket koji se šalje, te se objekt *SSH* veze sa uređajem briše, objekt se neće obrisati sve dok ne dobije odgovor na poslani paket.

Problem samog upravljanja aplikacije leži u prijenosu naredbi. U slučaju velikih zakašnjenja prijenosa, dogodi se da robot bude „van kontrole“ za taj period. Zbog toga bi bilo potrebno robotu napraviti program koji na jednake male vremenske intervale provjerava sa aplikacijom na računalu, da li je spojen. Iako se time ne rješava period „van kontrole“, ublažava se posljedica na način da robot neće otići predaleko.

Problem programa na robotu je što koristi *python* jezik. *Python* u nekim trenutcima pokreće sakupljača smeća što može preopteretiti procesor robota i time opada kvaliteta kretanja ili rotiranja kamere. To je posebice vidljivo kada osim *python* konzole je uključen program za video prijenos pogleda kamere što je uzrok povećanja korištenja procesora na sedamdeset posto. Servo motori za kameru titraju zbog kašnjenja koja se dogode u modulaciji širine impulsa kojom se upravljaju ti motori.

LITERATURA

[1] Wikipedia, “Legged robot,” *Legged robot*. https://en.wikipedia.org/wiki/Legged_robot (accessed Sep. 22, 2020).

[2] “Self-driving car,” *Self-driving car*. https://en.wikipedia.org/wiki/Self-driving_car (accessed Sep. 22, 2020).

[3] Wikipedia, “Roomba,” *Roomba*. https://en.wikipedia.org/wiki/Roomba (accessed Sep. 14, 2020).

[4] Wikipedia, “Mobile robot,” *Mobile robot*, Jul. 08, 2020. https://en.wikipedia.org/wiki/Mobile_robot (accessed Jul. 14, 2020).

[5] G. Martinović, “predavanja s kolegija Računalni sustavi stvarnog vremena, studij: Računalno inženjerstvo, ak. God. 2019/2020.” Accessed: Sep. 22, 2020. [Online]. Available: https://loomen.carnet.hr/pluginfile.php/314464/mod_resource/content/4/Predavanja/PR-1-ff.pdf.

[6] L. Jelenković, “Sustavi za rad u stvarnom vremenu.” zemris.fer.hr, 2019, Accessed: Jul. 13, 2020. [Online]. Available: http://www.zemris.fer.hr/~leonardo/srsv/skripta/_SRSV-skripta.pdf.

[7] Wikipedia, “Real-time computing,” *Real-time computing*, Jul. 06, 2020. https://en.wikipedia.org/wiki/Real-time_computing (accessed Jul. 14, 2020).

[8] Microsoft, “Visual Studio Community 2019,” *Visual Studio Community 2019*. https://visualstudio.microsoft.com/vs/community/ (accessed Aug. 29, 2020).

[9] *CP210x USB to UART Bridge VCP Driver for Windows 10*. Silicon Labs.

[10] WaveShare, “AlphaBot 2-Pi,” *AlphaBot2-Pi*, Dec. 18, 2020. https://www.waveshare.com/wiki/AlphaBot2-Pi (accessed Aug. 29, 2020).

[11] “AForge.NET Framework,” *AForge.NET Framework*. http://www.aforgenet.com/framework/ (accessed Aug. 29, 2020).

SAŽETAK

Tema završnog rada je „Izrada aplikacije stvarnog vremena za upravljanje mobilnom robotskom platformom“. Aplikacija je napravljena pomoću *Microsoft Visual Studio Community 2019* programskog okruženja. Aplikacija se sastoji od dva prozora. Svrha prvog prozora je spajanje na uređaj koji se nalazi na mreži. Aplikacija pronalazi sve uređaje spojene na mrežu, te ih prikazuje na prozor. Spajanje uređaja se započinje klikom na gumb spajanja, te aplikacija započinje proces spajanja na uređaj, očitava koji uređaj je odabran, te upisane vjerodajnice za prijavu na odabrani uređaj. Potom se postavlja SSH veza s uređajem i otvara sljedeći prozor, čija je svrha korisniku omogućiti upravljanje nad robotom. Video prijenos se uključuje skoro odmah i prikazuje se na tom prozoru. Nakon započinjanja video prijenosa, aplikacija šalje naredbe robotu za otvaranje *python konzole* i učitavanje *python* skripti potrebnih za upravljanje rotacijom kamere i kotačima robota. Kako su te skripte dio demonstrativnih programa za *AlphaBot 2*, potrebno ih je preuzeti zajedno sa bibliotekama koje su potrebne za rad tih programa.

KLJUČNE RIJEČI:

AlphaBot, C#, Python, Raspberry Pi, SSH, Upravljanje robota, Wi-Fi, Windows

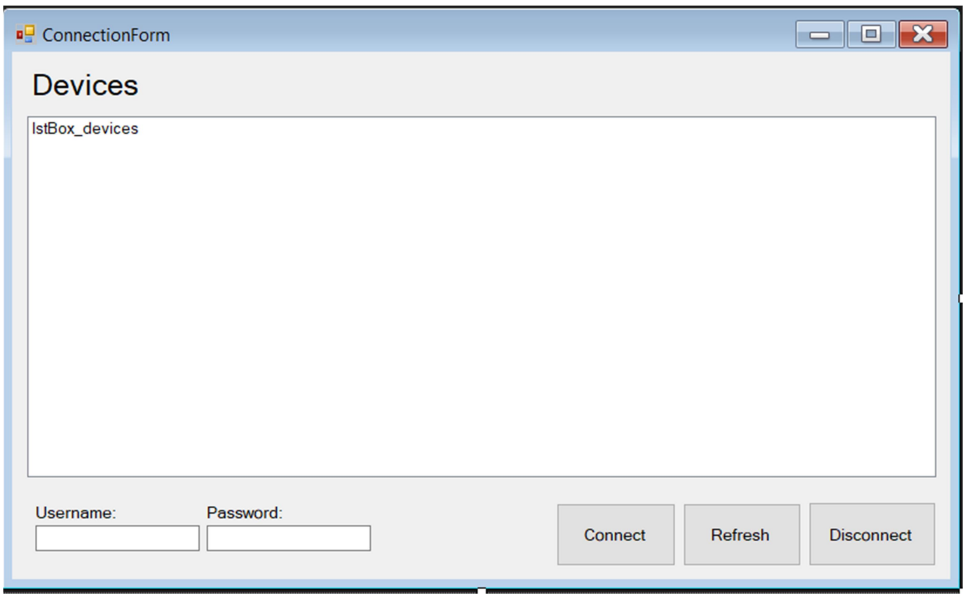
ABSTRACT

This final paper's topic is „Making of a real time application for mobile robot control“. The application is made using *Microsoft Visual Studio Community 2019* development environment. The application is composed of two windows. The purpose of the first window is to connect to a device on a network. The application looks for all devices on the network and displays them on the window. With a press of the button connect, the app begins the process of connecting to the device, and it does so by reading which device has been selected from the list, and reading the login credentials required for said device. Next, the application establishes a SSH connection with the device, and opens the next window, whose purpose is to enable the user to control the robot. The video stream starts almost immediately and it is being displayed on that window. After the start of the video feed, the application also sends commands to the robot to open the python console and to load the python scripts required to control the camera rotation and the robot's wheels. Since these scripts are a part of demonstrative programs for *AlphaBot 2*, it is required to download these along with the libraries required for the programs.

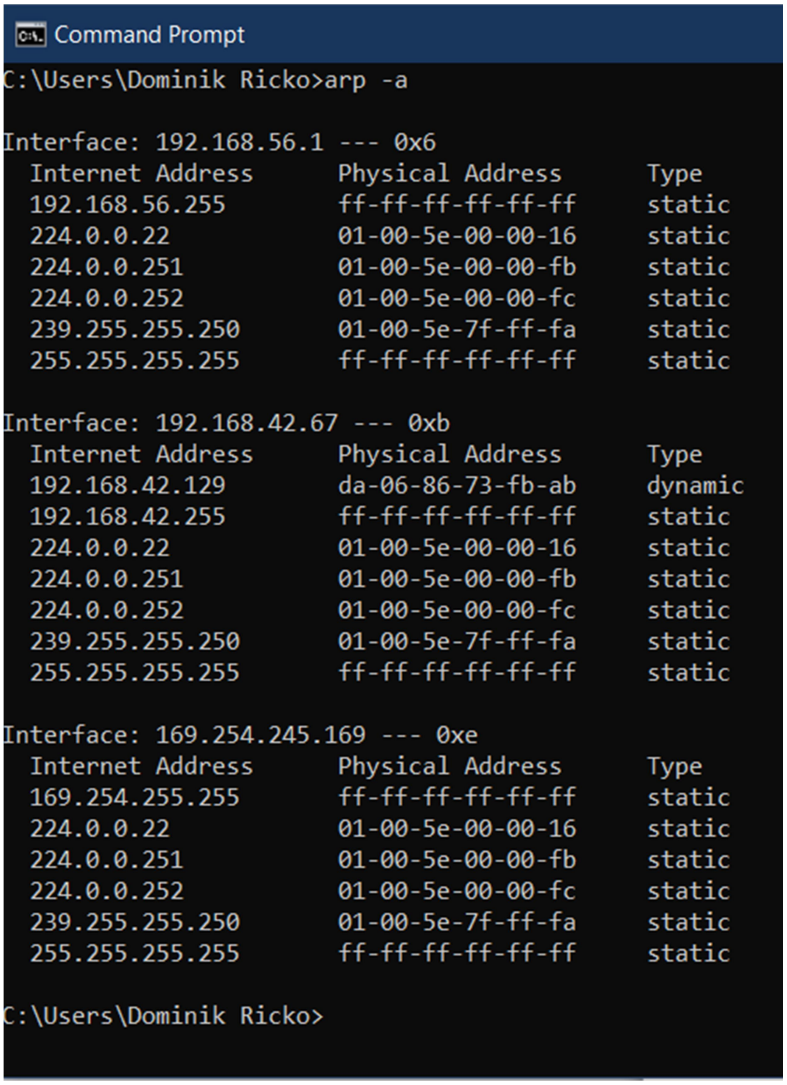
KEYWORDS:

AlphaBot, C#, Python, Raspberry Pi, Robot control, SSH, Wi-Fi, Windows

PRILOZI



Slika 0.1 *Izgled prozora za spajanje*



Slika 0.2 *Ispis naredbe arp -a*

```
public class NetworkDevice
   {
      public string Name { get; set; }
      public IPAddress IP { get; private set; }
      private BackgroundWorker hostEntryResolver;
      private EventHandler onNameChanged;
      public event EventHandler NameResolved
      {
         add
         {
            onNameChanged += value;
         }
         remove
```

```
        {
            onNameChanged -= value;
        }
    }
    protected virtual void OnNameResolved(EventArgs e)
    {
        onNameChanged?.Invoke(this, e);
    }
    public NetworkDevice(IPAddress deviceip)
    {
        IP = deviceip;
        Name = "Unknown";
    }
    public void ResolveName()
    {
        hostEntryResolver = new BackgroundWorker();
        hostEntryResolver.DoWork += worker_Resolve;
        hostEntryResolver.RunWorkerAsync(IP);
    }

    private void worker_Resolve(object sender, DoWorkEventArgs e)
    {
        IPAddress ip = (IPAddress)e.Argument;
        try
        {
            IPHostEntry hostEntry = Dns.GetHostEntry(ip);
            if (hostEntry != null)
            {
                Name = hostEntry.HostName;
                OnNameResolved(new EventArgs());
            }
        }
        catch (SocketException) { }
    }       public override string ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.Append(IP.ToString())
            .Append("\t\t")
            .Append(Name.ToString());
        return sb.ToString();
    }
}
```

Slika 0.3 *Definicija NetworkDevice klase*

```
public static class NetworkDeviceFinder
```

```
{
    public static List<NetworkDevice> KnownDevices { get; private set; }
    private static List<IPAddress> filteredAddresses = null;
    private static IPAddress GetIPFromArpLine(string line)
    {
        string[] segments = line.Split('.');
        List<Byte> ipbits = new List<Byte>();
        foreach (string s in segments)
        {
            byte output = 1;
            bool result = false;
            if(s.Length > 3)
            {
                result = byte.TryParse(s.Substring(0, 3), out output);
                if (!result)
                {
                    result |= byte.TryParse(s, out output);
                }
                if (result)
                {
                    ipbits.Add(output);
                }               }
            else
            {
                byte.TryParse(s, out output);
                ipbits.Add(output);
            }
        }
        return new IPAddress(ipbits.ToArray());
    }
    private static List<IPAddress> GetIPsFromArpLines(List<string> arpLines)
    {
        List<IPAddress> addresses = new List<IPAddress>();
        for (int i = 0; i < arpLines.Count; i++)
        {
            if (arpLines.ElementAt(i).Length <= 1)
            {
                arpLines.RemoveAt(i--);
            }
            else
            {
                string temp = arpLines.ElementAt(i).Trim();
                if (temp.StartsWith("I"))
                {
                    arpLines.RemoveAt(i--);
                }
                else
```

```
            {
                IPAddress ip = GetIPFromArpLine(temp);
                addresses.Add(ip);
            }
        }
    }
    return addresses;
}
public static List<NetworkDevice> Find()
{
    if(KnownDevices == null)
    {
        KnownDevices = new List<NetworkDevice>();
        filteredAddresses = new List<IPAddress>();
    }
    List<NetworkDevice> foundDevices = new List<NetworkDevice>();
    System.Diagnostics.Process Process = new System.Diagnostics.Process();
    Process.StartInfo.FileName = "arp";
    Process.StartInfo.Arguments = "-a ";
    Process.StartInfo.UseShellExecute = false;
    Process.StartInfo.RedirectStandardOutput = true;
    Process.StartInfo.CreateNoWindow = true;
    Process.Start();
    string strOutput = Process.StandardOutput.ReadToEnd();
    string[] lines = strOutput.Split('\n');
    List<string> linesList = lines.ToList<string>();
    List<IPAddress> addresses = GetIPsFromArpLines(linesList);
    foreach(IPAddress ip in addresses)
    {
        if (!filteredAddresses.Contains(ip))
        {
            filteredAddresses.Add(ip);
            foundDevices.Add(new NetworkDevice(ip));
        }
    }
    KnownDevices.AddRange(foundDevices);
    KnownDevices.Sort(new NetworkDeviceIPComparer());
    return foundDevices;
}
}
```

Slika 0.4 *Definicija klase NetworkDeviceFinder*

```
class NetworkDeviceIPComparer : Comparer<NetworkDevice>
{
    Comparer<IPAddress> comparer;
```

```
    public NetworkDeviceIPComparer()
    {
        comparer = new IPComparer();
    }
    public override int Compare(NetworkDevice x, NetworkDevice y)
    {
        return comparer.Compare(x.IP, y.IP);
    }
}
```

Slika 0.5 *Definicija klase NetworkDeviceIPComparer*

```
class IPComparer : Comparer<IPAddress>
{
    public override int Compare(IPAddress x, IPAddress y)
    {
        byte[] xBytes = x.GetAddressBytes();
        byte[] yBytes = y.GetAddressBytes();
        uint totalX = (uint)xBytes[0] * 16777216 + (uint)xBytes[1] * 65536 + (uint)xBytes[2] * 256
+ (uint)xBytes[3];
        uint totalY = (uint)yBytes[0] * 16777216 + (uint)yBytes[1] * 65536 + (uint)yBytes[2] * 256
+ (uint)yBytes[3];
        if (totalX < totalY)
        {
            return -1;
        }
        else if (totalX > totalY)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }
}
```

Slika 0.6 *Definicija klase IPComparer*