Sustav za posredno korištenje ulaznih uređaja

Kruljac, Luka

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

Sveučilišni diplomski studij

SUSTAV ZA POSREDNO KORIŠTENJE ULAZNIH UREĐAJA

Diplomski rad

Luka Kruljac

Osijek, 2021

Sadržaj

1. **UVOD** **1**

1.1 Zadatak rada 1

2. **SUSTAVI ZA POSREDNO KORIŠTENJE ULAZNIH UREĐAJA** **4**

2.1 Postojeća rješenja 4

2.2 Usporedba postojećih rješenja s novim rješenjem 4

3. **RAZVOJ APLIKACIJE ZA POSREDNI UREĐAJ** **6**

3.1 WPF Programski okvir 6

3.2 MVVM Struktura 7

3.3 RAW ulazna biblioteka 10

3.4 Tijek rada aplikacije 10

3.5 Rezultat i korištenje 10

4. **UPRAVLJAČKI PROGRAM ZA SUSTAVE BAZIRANE NA UNIX-U** **12**

4.1 Razvojno okruženje 12

4.2 Glavna funkcija upravljačkog programa 14

4.3 Parsiranje argumenata glavne funkcije 15

4.4 Dretva za komunikaciju 16

4.5 Dretva za izvršavanje 17

4.6 Korištenje upravljačkog programa na ciljanoj platformi 17

5. **DEMO PROJEKT** **19**

5.1 Sklopovlje 19

5.2 Programska podrška 20

6. **REZULTATI TESTIRANJA SUSTAVA** **22**

6.1 Rezultati mjerenja 24

7. **ZAKLJUČAK** **26**

LITERATURA **27**

POPIS I OPIS UPOTRJEBLJENIH KRATICA **28**

SAŽETAK **29**

Ključne riječi 29

ABSTRACT **30**

Key words 30

ŽIVOTOPIS **31**

PRILOZI **32**

ELEKTRONIČKA VERZIJA RADA **33**

1. Uvod

U današnje vrijeme moderan čovjek okružen je velikim brojem raznih uređaja, tehničkih i operacijskih sustava. Većina takvih sustava dizajnirana je da u nekim trenutcima zahtijeva interakciju s korisnikom. Kako bi se takva interakcija i ostvarila dizajnirani su posebni uređaji, koji se međusobno razlikuju ovisno o svojoj namjeni, a koji se grupno nazivaju ulaznim uređajima, dok se uređaji odnosno platforme kojima ulazni uređaji upravljaju nazivaju ciljani uređaji odnosno ciljane platforme.

Postoje situacije kada nije moguće na konvencionalni načini povezivati ulazne uređaje s ciljanom platformom, odnosno ciljanim uređajem putem PS/2, USB, *Bluetooth* ili neke slične veze kao na prikazu sa slike 1.1. Razlozi za takve probleme su razni, a neki od njih nabrojani su u nastavku.

- Integrirani ulazni uređaji

  Ako se za primjer promatra prijenosno računalo, njegovu integriranu tipkovnicu nije moguće (barem ne na jednostavan način) od spojiti od prijenosnog računala te ju zatim spojiti i koristit na nekom drugom uređaju.

- Ciljani uređaj je mobilan

  Ukoliko je ciljani uređaj „na kotačima“, potrebno je pomjerati i ulazni uređaj. Primjerice učenički projekt automobila pokretan RPI (engl. *Raspberry Pi*) računalom, a koji se upravlja tipkovnicom ili kontrolerom (engl. *gampad*).

- Ciljani uređaj je dislociran ili fizički nedostupan

  Ponekad nije moguće spojiti ulazni uređaj jer je korisnik udaljen od njega odnosno nema fizički pristup, ili ako ima pristup, nema mogućnost spajanja, primjerice, zauzeti su svi USB portovi

- Ulazni uređaj u fizičkom obliku nije dostupan

  Aplikacija na posrednom računalu nudi i tzv. zaslonske uređaje, gdje kliktanje po sučelju simulira pritisak odgovarajuće tipke pa posjedovanje ulaznog uređaja nije nužno.

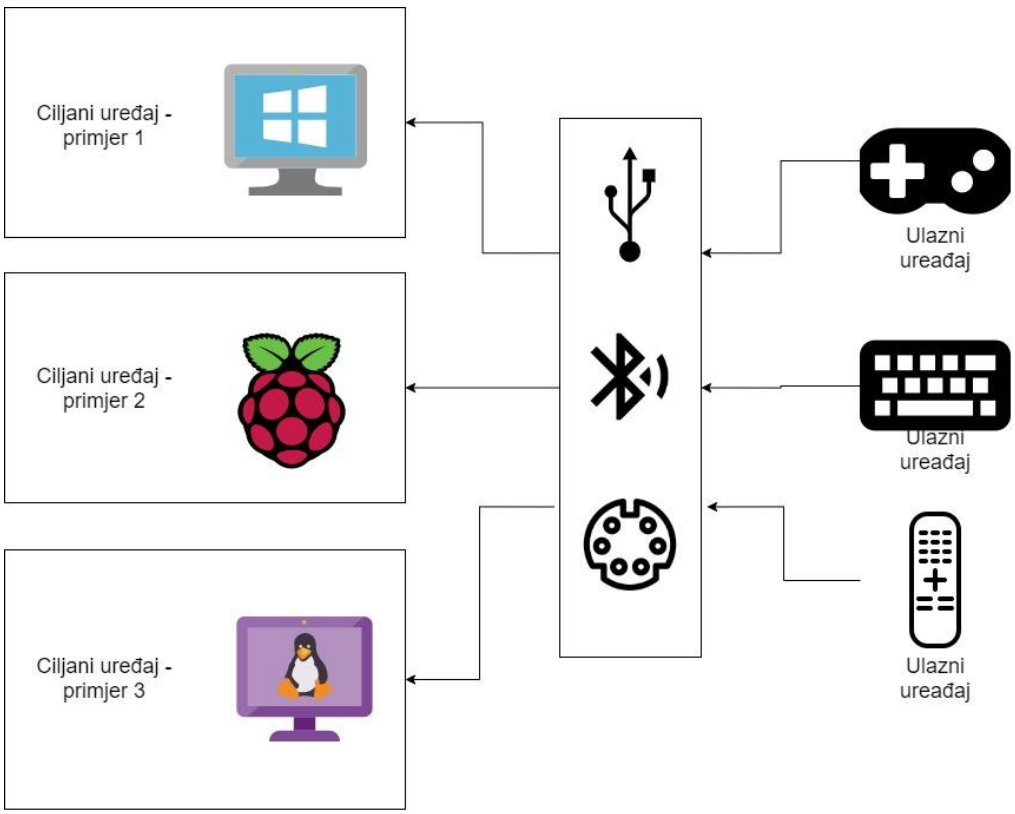
1.1 Zadatak rada

Zadatak ovog diplomskog rada je razviti programsku podršku (engl. *software*) za posredno korištenje ulaznih uređaja (engl. *using input devices indirectly*). Programska podrška obuhvaća čitavo rješenje (engl. *solution*) u obliku više projekata koji u cjelini omogućuju korištenje određenog ulaznog uređaja na ciljanom uređaju (engl. *target device*), bez da su uređaji direktno

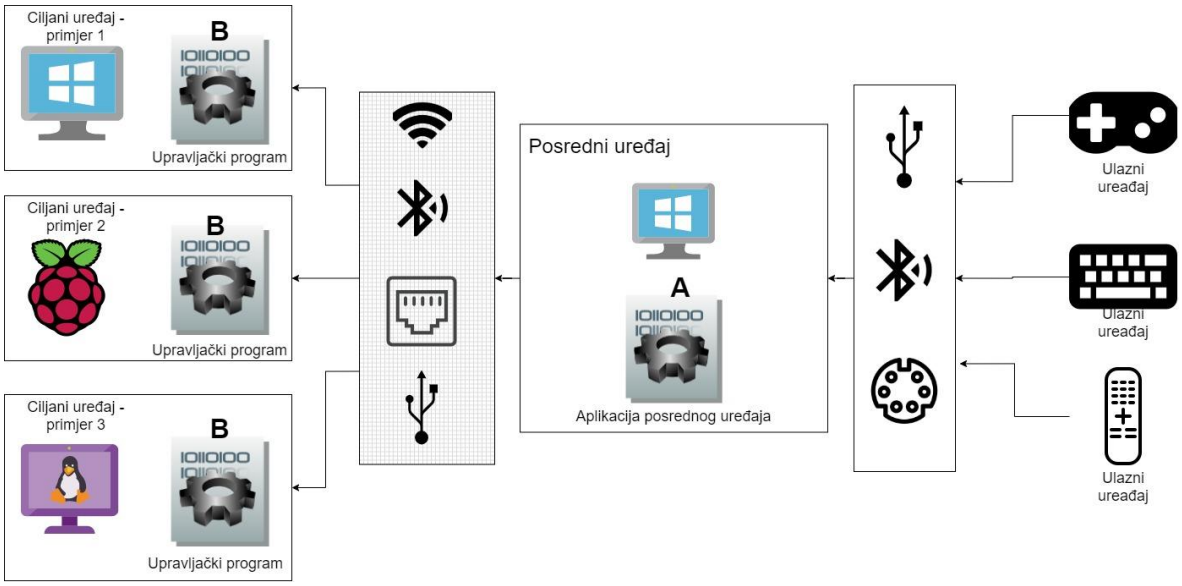
spojeni. Veza između njih uspostavljena je preko posrednog uređaja koje je sa oba spomenuta uređaja direktno povezan određenom vezom. Za uspješan rad ovakvog koncepta potrebno je izraditi upravljačku programsku podršku (engl. *driver*) na strani ciljanog uređaja, odnosno više njih, ovisno o platformama ciljanih uređaja koje se žele koristiti. Također je potrebno izraditi i aplikaciju koja se izvršava na posrednom uređaju, a koja za ulogu ima uspostavljenje komunikacijskog puta između ulaznog i ciljanog uređaja odnosno pruža sučelja zaslonskih virtualnih uređaja. Ovaj rad obuhvaća cjelokupni proces razvoja sustava sa slike 1.2, od analize problema, usporede postojećih rješenja, preko razvoja do testiranja. Kroz drugo poglavlje predstavljene su postojeće tehnologije koje imaju sličnu ulogu kao i njihove prednosti i mane u odnosu na koncept sustava iz naslova ovog rada. U poglavljima tri i četiri opisani su postupci razvoja sustava te funkcioniranje dvije najvažnije komponente sustava:

1. Aplikacija na posrednom uređaju, obrađeno kroz treće poglavlje.
2. Upravljački program za ciljanu platformu, obrađeno kroz četvrto poglavlje.

Također, u sklopu rada pripremljen je i demo sustav čija je izrada predstavljena u poglavlju pet, dok su rezultati i analiza provedenog testa opisani u šestom poglavlju ovoga rada.



Slika 1.1 Uobičajen način povezivanja uređaja



Slika 1.2 Željeni način povezivanja uređaja

2. Sustavi za posredno korištenje ulaznih uređaja

Sustavi za posredno korištenje ulaznih uređaja razlikuju se po funkcionalnostima koje nude te njihovom načinu izvedbe. Te razlike odnose se na tipove platformi koje podržavaju, načinu povezivanja između uređaja, podrške za određene ulazne uređaje, načinu postavljana sustava te preduvjetima za korištenje istih. Kroz ovo poglavlje bit će predstavljeni neki od postojećih sustava, odnosno neke od već dostupnih aplikacija koje se bave sličnom tematikom.

2.1 Postojeća rješenja

Računarstvo u virtualnim mrežama (engl. *Virtual Network Computing* - VNC) alat iz rada [1] rješava sve probleme koji su navedeni u prvom poglavlju, uz još nekoliko drugih funkcionalnosti poput prikazivanja zaslona ciljanog uređaja na posrednom uređaju, prosljeđivanje zvuka itd. Međutim, ovo rješenje ima određena ograničenja po pitanju zahtijeva, odnosno jedan od nedostataka je što ciljani uređaj mora imati spojen ili virtualno montirani zaslon (eng. *display*). Osim toga ovo podrazumijeva isključivo TCP/IP vezu između ciljanog i posrednog uređaja. Osim VNC tu je i sigurna ljuska (engl. *Secure shell* - SSH) rješenje iz rada [2] koji također pruža puno više mogućnosti i ima puno širu primjenu. Ovaj program rješava samo neke od navedenih problema – moguće je proslijediti događaje tipkovnice ali ne i *gampeada*. Zbog ograničenja koje je postavljeno na vezu, između posrednog i ciljanog uređaja moguće je komunicirati isključivo putem TCP/IP ili UART-a. Također, postoje problemi i kod prosljeđivanja događaja tipkovnice, primjerice pritisak više tipki istodobno nije podržan. Rješenja ovih problema nudi i *Microsoft* sa svojim *Mouse Without Borders* alatom iz [3]. Alat je namijenjen za korištenje u okruženjima s više uređaja na kojima je isključivo *Windows* operacijski sustav. Osim toga, dizajniran je na način da je moguće povezati samo uređaje koji se spojeni u istu lokalnu mrežu.

2.2 Usporedba postojećih rješenja s novim rješenjem

Za razliku od gore spomenutih rješenja, sustav predložen u ovom radu pruža mogućnost istovremenog korištenja jednog uređaja na više različitih uređaja neovisno pripadaju li istoj platformi. Za razliku od VNC-a i *Mouse Without Borders* alata, moguće ga je koristiti i na uređajima koji nemaju ugrađeni zaslon, odnosno moguće je uštedjeti resurse na ciljanoj platformi jer nije potrebno postavljati virtualni zaslon. Također, ukoliko je konkretno sklopovlje uređaja nedostupno, moguće je koristiti virtualne uređaje, te je programska podrška dizajnirana tako da je u budućnosti moguće dodavati i druge vrste ulaznih uređaja. Uspoređujući ovo rješenje s SSH rješenjem prednost je što je koristeći ovaj sustav moguće poslati odzive uređaja različitih od tipkovnice, te je moguće slati više pritisaka tipki tipkovnice istovremeno. Međutim, za postavljanje

sustava u određenim slučajevima i dalje je potrebno koristiti SSH kada želimo pokrenuti upravljački program na ciljanoj platformi. Zaključak je da postojeća rješenja ne rješavaju sve nabrojene probleme u jednoj cjelini ili ih ne rješavaju na efikasan način, odnosno postavljaju dodatne preduvjete koji možda ne mogu biti ispunjeni u svakoj situaciji kada je potrebno upravljati ciljanom platformom.

3. Razvoj aplikacije za posredni uređaj

Na slici 1.2 prikazano je da u tako postavljenom sustavu posredni uređaj zahtijeva programsku podršku (komponenta označena slovom A), koji će omogućiti uređaju da postane posrednik između ulaznog uređaja i ciljane platforme. Da bi programska podrška odradila svoju ulogu postoji nekoliko zahtijeva među kojima su tri ključna:

1. Osluškivanje strujanja podataka (engl. *data stream*) ulaznih uređaja
2. Osiguranje komunikacijskog kanala s ciljanom platformom
3. Generiranje i slanje poruke kroz komunikacijski kanal

Osim tri ključna zahtijeva, zadani su zahtjevi poput filtriranja određenih uređaja kako bi korisnik između više istovrsnih ulaznih uređaja mogao izabrati čije događaje želi prosljeđivati. Na primjer, spojene su dvije tipkovnice na posrednom uređaju međutim na ciljani uređaj prosljeđivat će se događaji samo s odabrane tipkovnice. Također, jedan od zahtijeva je i omogućavanje zaslonskih virtualnih uređaja, poput zaslonske tipkovnice (engl. *On Screen Keyboard*).

U sklopu ovog prototipa posredni uređaji ograničeni su na uređaje s *Windows* operacijskim sustavom. S obzirom da *Windows* operacijski sustav omogućava grafičko korisničko sučelje (engl. *Graphical User Interfernce* - GUI) te uz činjenicu da velika većina korisnika preferira GUI aplikacije u usporedbi s aplikacijama sa sučeljem naredbenog retka (engl. *Command Line Interface*), logičan izbor za tip ove komponente, odnosno aplikacije koja očekuje interakciju s korisnicima je *Windows* desktop aplikacija.

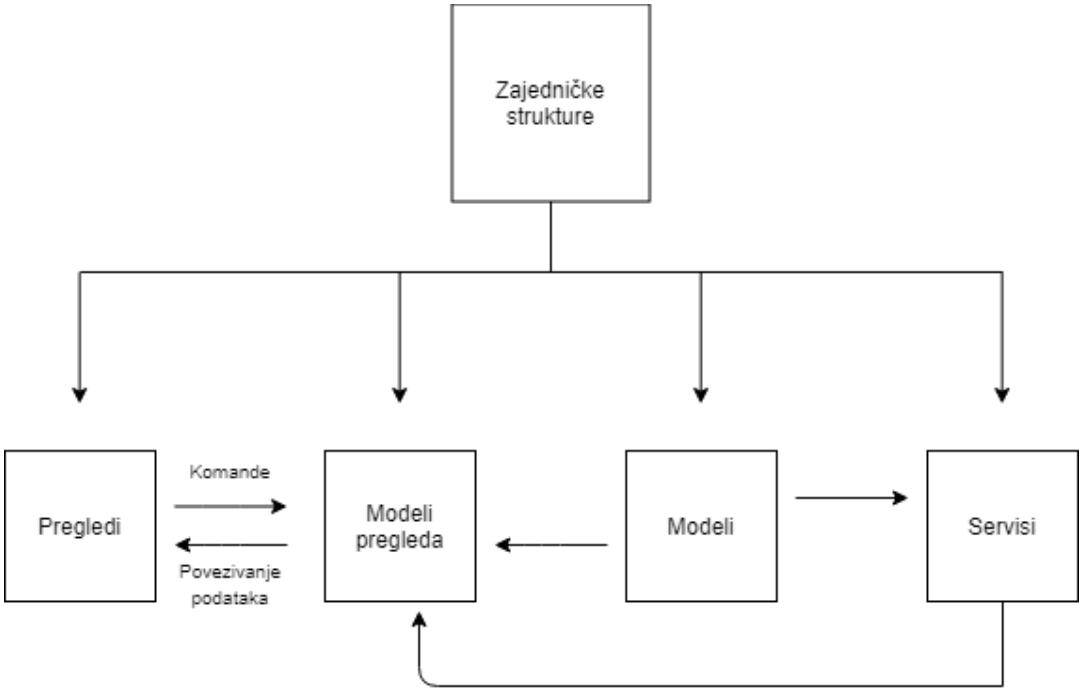
3.1 WPF Programski okvir

Za izradu *Windows* desktop aplikacije u ovom projektu korišten je radni okvir (engl. *framework*) otvorenog koda (engl. *open source*) *Windows* prezentacijski temelj (engl. *Windows Presntation Foundation* - WPF*)* iz [4]. Ovaj radni okvir temeljen je na vektorskom prikazu koji uz široki set značajki pomoću proširenog aplikacijskog jezika za označavanje (engl. *Extensible Application Markup Language* - XAML) omogućuje poprilično jednostavnu izgradnju GUI-a neovisnu o rezoluciji, te popratnog pozadinskog koda (engl. *backend code*) u jezicima iz .NET okruženja (C#, F#, VB). Neke od ključnih značajki WPF-a jesu da koristi *DirectX*, odnosno sklopovsko ubrzanje za iscrtavanje grafičkih elementa, omogućuje povezivanje podataka (engl. *data binding*) pomoću čega se lako odvajaju cjeline prikaza (engl. *view*) od cjelina podataka (engl. *models*). S obzirom da je *Miscrosoft* kreator radnog okvira, *Visaul Studio* je zasigurno najbolje integrirano razvojno okruženje (engl. *Integrated Development Environment* - IDE) za razvoj WPF projekta te je on i korišten tijekom razvoja. Od programskih jezika, korišten je C# za pozadinski

kod kao i neke predkompajlirane (engl. *precompiled*) dinamičke biblioteke koje su razvijene C/C++ jezikom.

3.2 MVVM Struktura

Pri razvoju aplikacije, za odvajanje programskih blokova ispravno je koristit neki od konstrukcijskih uzoraka (engl. *design patern*) kao što su Model-Pregled-Kontorler (engl. *Model-View-Controler* - MVC), Model-Pregled-PregledModel (engl. *Model-View-ViewModel* - MVVM), Model-Pregled-Prezenter (engl. *Model-View-Presenter* – MVP) itd.. Korištenje konstrukcijskih uzoraka olakšava i ubrzava sam proces razvoja, omogućuje lakše testiranje i pojednostavljuje potencijalne izmjene u idućim nadogradnjama. Za ovaj projekt odabrana je MVVM struktura te su uz osnovne MVVM komponente dodane i komponente servisa (engl. *Services*) te zajedničke komponente (engl. *Common*). Ovaj set predstavlja čitavo rješenje aplikacije, prikazano na slici 3.1. Komponente Zajedničke strukture (*Common*), Modeli (*Model*), Servisi (*Services*) i Modeli pregleda (*ViewModel*) predstavljaju .NET *Framework Class Library* projekte čiji su produkti datoteke tipa *.dll*, dok je komponenta Pregleda (*View*) *Windows* aplikacija na dnu hijerarhije čiji je produkt izvršna .exe datoteka.



Slika 3.1 Hijerarhijski prikaz projekata

Common projekt sadrži dvije klase koje omogućuju korištenje MVVM strukture te korištenje *data bindinga*. Datoteka *BaseView.cs* prikazan na kodu 3.1 je apstraktna klasa koju nasljeđuju sve *model* i *viewmodel* klase. Nasljeđivanjem ove klase omogućuje se izvršavanje

funkcije upravljanja događajima (engl. *event handler*) koji će reagirati na promjene vrijednosti odnosno bit će pozvan u *Set* metodi svojstava koji se prikazuju na *View* strani prikazano na kodu 3.2. Ovaj mehanizam omogućuje da svaka promjena svojstava u pozadinskom kodu bude automatski promijenjena i u GUI-u. Datoteka *DelegateCommand.cs* definira objekt korišten unutar onih *ViewModela* čiji *View* sadrži neke komande te tako omogućuje povezivanje određene funkcije iz pozadinskog koda s određenom interaktivnom GUI komponentom. Sama datoteka sadrži dvije različite klase koje se razlikuju po tipu odnosno omogućuju povezivanje funkcija void tipa ili bilo kojeg generičkog tipa.

```
public abstract class BaseViewModel : INotifyPropertyChanged
{
      public event PropertyChangedEventHandler PropertyChanged;
      protected void RaisePropertyChangedEvent(string propertyName)
      {
            var handler = PropertyChanged;
            if (handler != null)
            {
                  handler(this, new PropertyChangedEventArgs(propertyName));
            }
      }
}
```

Kod 3.1 Upotreba metode u model klasi

```
public string Status
{
     get { return _Status; }
     set
     {
           _Status = value;
           RaisePropertyChangedEvent("Status");
     }
}
```

Kod 3.2 Prikaz upotrebe u klasi modela

Unutar model projekta definirane su klase koje opisuju objekte te enumeracije korištene u *ViewModel* i *Services* klasama. Konkretno, definirani su modeli mogućih linkova te mogućih ulaznih uređaja i apstraktne klase koje uređaji odnosno linkovi nasljeđuju. Na primjeru sa koda 3.3 vidljivo je kako je definiran konektor kojeg kasnije neka konkretna realizacija konektora i nasljeđuje. Dodavanjem novih linkova komunikacija odnosno konektora potrebno je definirati klasu koja će naslijedi klasu *Connector* te definirati metode *SendMessage* i *Connect*. Ovim je postignuta jednostavnost uvođenja novih načina povezivanja između posrednog uređaja i ciljane platforme. Na sličan način definirani su i modeli ulaznih uređaja.

```
public abstract class Connector : BaseViewModel
{
    #region Properites
    private string _Status;
    public string Status
    {
        get { return _Status; }
        set { _Status = value; RaisePropertyChangedEvent("Status"); }
    }
    private bool _IsConnected;

    public bool IsConnected
    {
        get { return _IsConnected; }
        set
        {
            _IsConnected = value;
            RaisePropertyChangedEvent("IsConnected");
            RaisePropertyChangedEvent("ConnectionStatus");
        }
    }
    private string _ConnectionStatus;
    public string ConnectionStatus
    {
        get
        {
            if (IsConnected){ _ConnectionStatus = "Connected"; }
            else { _ConnectionStatus = "Disconnected"; }
            return _ConnectionStatus;
        }
        set
        {
            _ConnectionStatus = value;
            RaisePropertyChangedEvent("ConnectionStatus");
        }
    }
    #endregion
    public abstract void SendMessage(byte[] data);
    public abstract void Connect();
}
```

Kod 3.3 Definicija apstraktne klase konektor s primjenom *BaseViewModel* klase

ViewModel sadrži klase koje opisuju pozadinski kod pojedinog *viewa*. Za svaku stranicu odnosno kontrolu koja se može pojaviti u grafičkom sučelju aplikacije definiran je odgovarajući *viewmodel*. *Viewmodel* kreira objekte modela, poziva metode servis klasa na određene aktivnosti u grafičkom sučelju poput otvaranje prozora, pritisak tipke, selektiranje potvrdnog okvira (engl. *checkbox*) i sl.. *View* dio definira izgled svih prozora, stranica i kontrola na pojedinim stranicama. Također, ovaj projekt je polazna točka(engl. *startup*) čitave aplikacije pa je ondje definirana i

startup funkcija, a osim toga unutar klase aplikacije definirana je i veza između pojedinog *Viewa* te pripadajućeg *ViewModela*.

Da bi rezultat čitavog rješenja mogao biti dostavljen u jednoj izvršnoj datoteci potrebno je pri izgradnji *View* projekta u izlaznu izvršnu datoteku ugraditi korištene biblioteke, odnosno i izgrađene izlazne .dll datoteke ostalih projekata. Da bi se tako nešto postiglo dovoljno je koristiti gotove pakete *Fedora* i *Costura* unutar *View* projekta.

3.3 RAW ulazna biblioteka

Najvažniji dio ove aplikacije je detektiranje događaja koje kreiraju ulazni uređaji. Za realizaciju ovog zahtijeva koristeći C# okruženje moguće je koristiti *RAW input API* iz [6]. *RAW input* omogućuje aplikacijama stabilan i robustan način za prihvaćanje neobrađenih ulaznih podataka od bilo kojeg korisničkog ulaznog uređaja (engl. *Human Input Device* - HID).

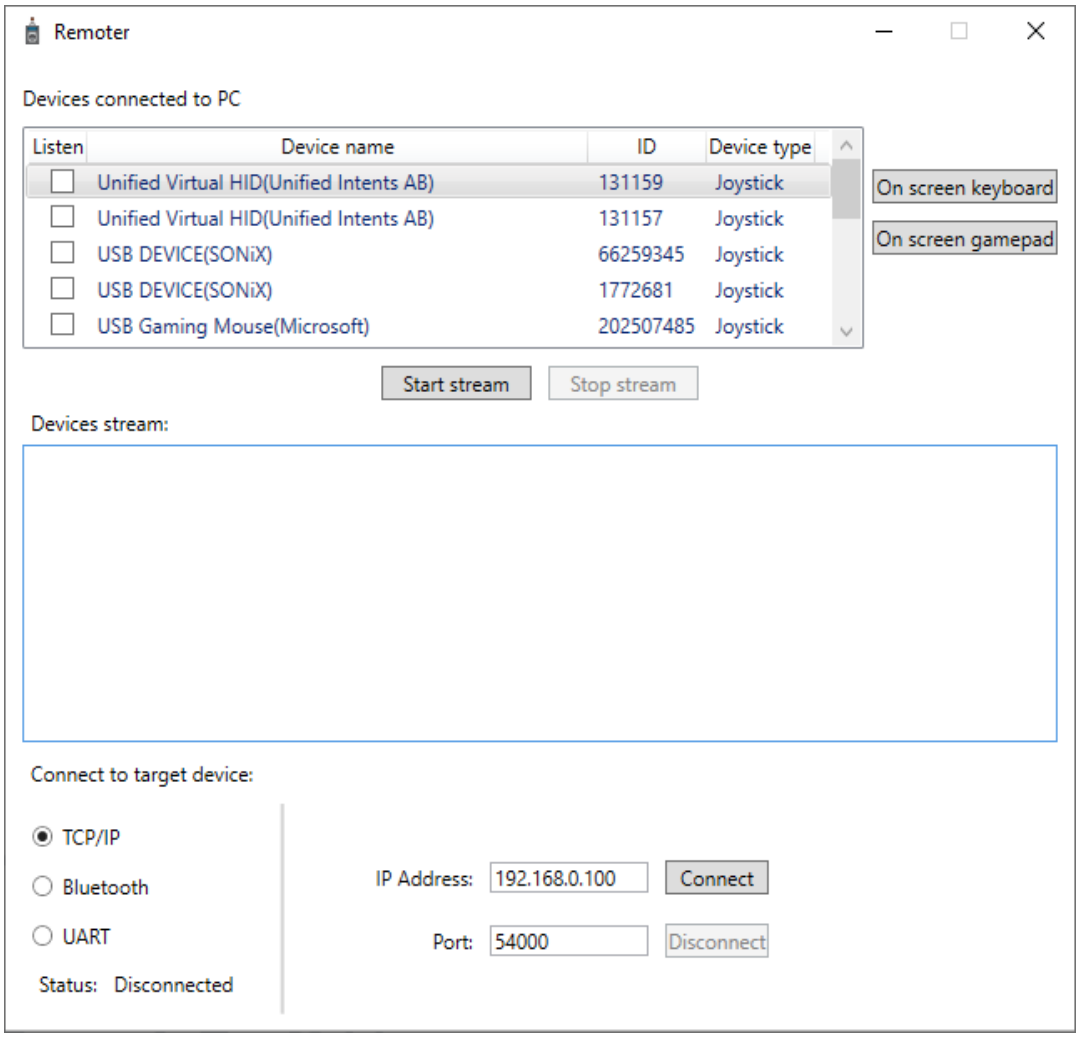
3.4 Tijek rada aplikacije

Kao početni korak u pozadinskom kodu ove aplikacije je prikaz glavnog prozora te izlistanje svih korisničkih ulaznih uređaja na njemu. Zatim se od korisnika očekuje uspostavljanje komunikacije s ciljanim uređajem. Da bi se definirao status veze, očekuje se da ciljani uređaj pošalje odzvanjajuću (engl. *echo*) poruku. Pri ostvarivanju te veze izgradit će se strujanje (engl. *stream*) koje će kasnije biti korišteno u povratno pozivnim funkcijama za slanje poruka. Nadalje, kada je komunikacija uspostavljena, očekuje se od korisnika da odabere uređaje čije događaje želi proslijedit na ciljani uređaj. Označavanjem uređaja te pritiskom na tipku *Start Stream* za svaki označeni uređaj kreira se vlastita dretva (engl. *thread*) koja osluškuje RAW input tog uređaja, osim toga pri kreiranju dretve, prosljeđuje se i povratno pozivna funkcija koja će se izvršavati na svaku pojavu novog događaja pripadajućeg uređaja. U ovom slučaju korištene povratno pozivne funkcije su metode iz klase pojedinog uređaja te je u tim metodama definiran postupak slanja poruka te ažuriranje podataka na glavnom prozoru.

3.5 Rezultat i korištenje

Gradnjom (engl. *Building*) gore opisanih komponenti konačni rezultat je aplikacija s glavnim prozorom prikazanim na slici 3.2. U ovom prozoru moguće je uspostaviti link između otvorene aplikacije na posrednom uređaju te upravljačkog programa na ciljanom uređaju. Aplikacija će korisnika obavijestiti o statusu veze te ukoliko je veza uspostavljena moguće je krenuti na idući korak. Aplikacija će pri svom pokretanju izlistati sve pronađene HID uređaje te je na korisniku da odabere koji od njih je potrebno proslijediti na ciljani uređaj. Označavanjem

uređaja te pritiskom na tipku *Start stream* svi događaji generirani od strane selektiranih uređaja bit će proslijeđeni prema upravljačkom programu na ciljanom uređaju. Također, moguće je otvoriti i virtualne uređaje poput *On screen keyboard* i *On screen Gamepad*.



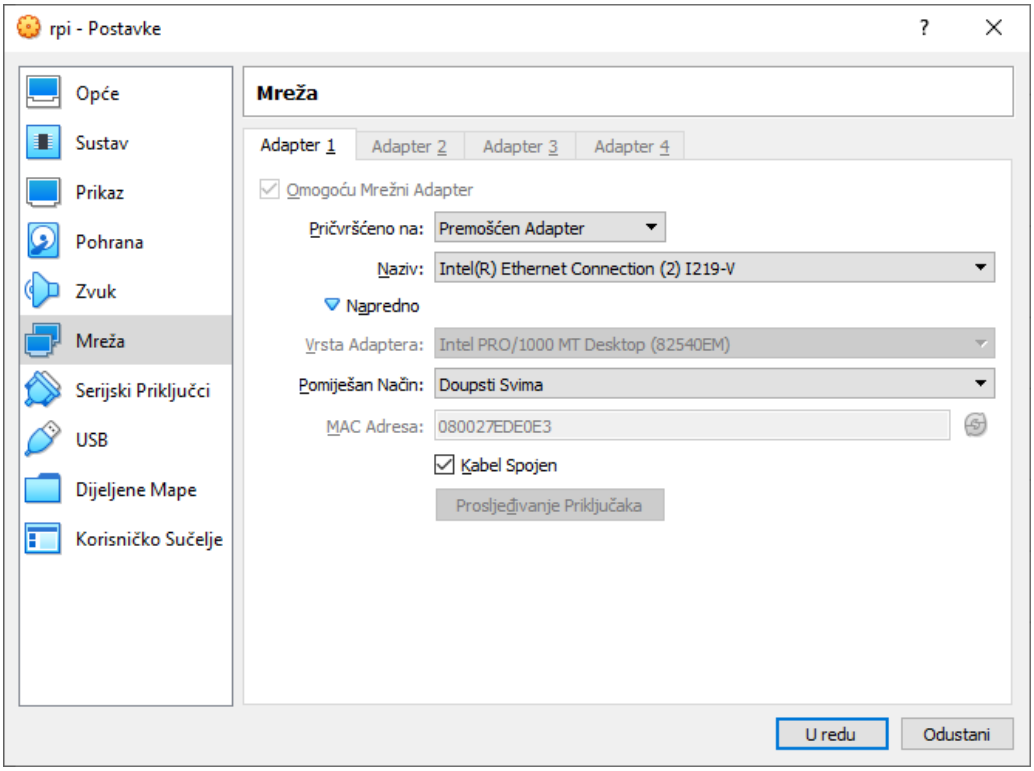
Slika 3.2 Glavni prozor *Windows* aplikacije

4. Upravljački program za sustave bazirane na UNIX-u

Kako bi poruke generirane na strani desktop aplikacije bile pravilno detektirane te kako bi operacijski sustav na ciljanom uređaju kreirao događaje na temelju takvih poruka, potreban je upravljački program na strani ciljanog uređaja. U sklopu ovog prototipa kreirana je aplikacija namijenjena za izvršavanje na UNIX arhitekturi.

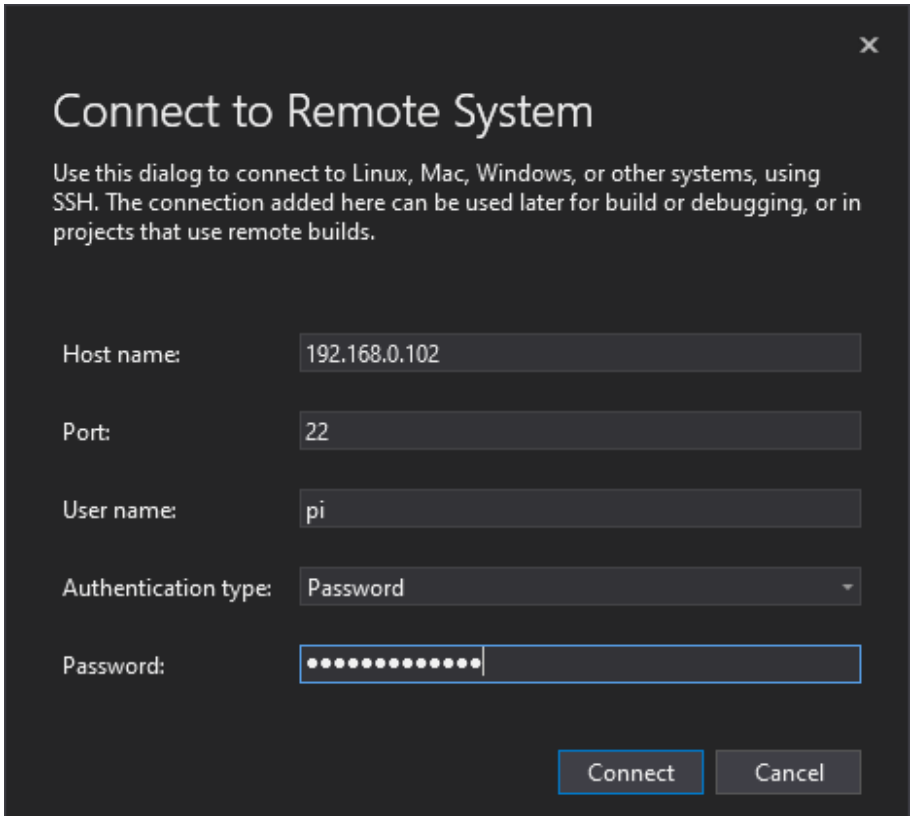
4.1 Razvojno okruženje

Za razvoj ovog upravljačkog programa korišteni su C i C++ programski jezici. Od alata za razvoj korišteni su *MS Visual Studio* kao integrirano razvojno okruženje, *Oracle VM Virtaul Box* za simulaciju ciljane platforme. Unutar *Visaul Studia* korištena je značajka za unakrsno kompajliranje (engl. *Cross Compile*) s obzirom da je program razvijan na *Windows* operacijskom sustavu, a namijenjen je za UNIX arhitekturu. S obzirom da je cilj ovog projekta bio da dvije aplikacije komuniciraju, između ostalog i preko TCP/IP-a, potrebno je bilo učiniti još neka postavljanja poput prosljeđivanja porta. Naime, u TCP/IP komunikaciji potrebna su 2 parametra, IP te port. S obzirom da virtualna mašina i njezina adresa nije vidljiva iz domene *Windows* aplikacije prema zadanim postavkama, potrebno je na mrežnim postavkama postaviti mrežni adapter u premošteni način rada kao na slici 4.1.

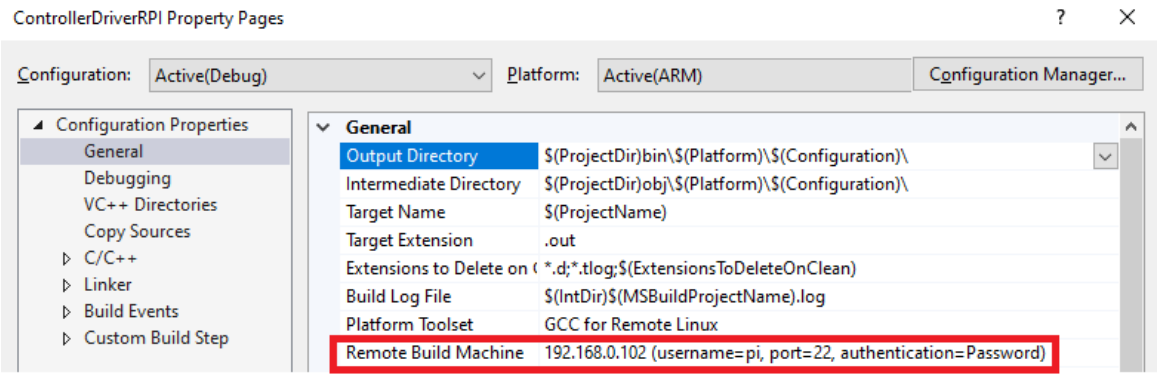


Slika 4.1 Postavke mrežnog adaptera

Nadalje, da bi se projekt mogao izgraditi (engl. *Build*) te debugirati potrebno je pripremljenu virtualnu mašinu dodati u IDE. U postavkama za unakrsno kompajliranje, prema [7] koristeći *connection manger* potrebno je dodati podatke s ranije kreirane virtualne mašine kao na slici 4.2. Nakon toga, ovaj dodani sustav je potrebno dodati i u postavkama projekta pod parametre *Remote Build Machine* prikazano na slici 4.3 te na isti način odraditi i za *Remote Debug Machine*.



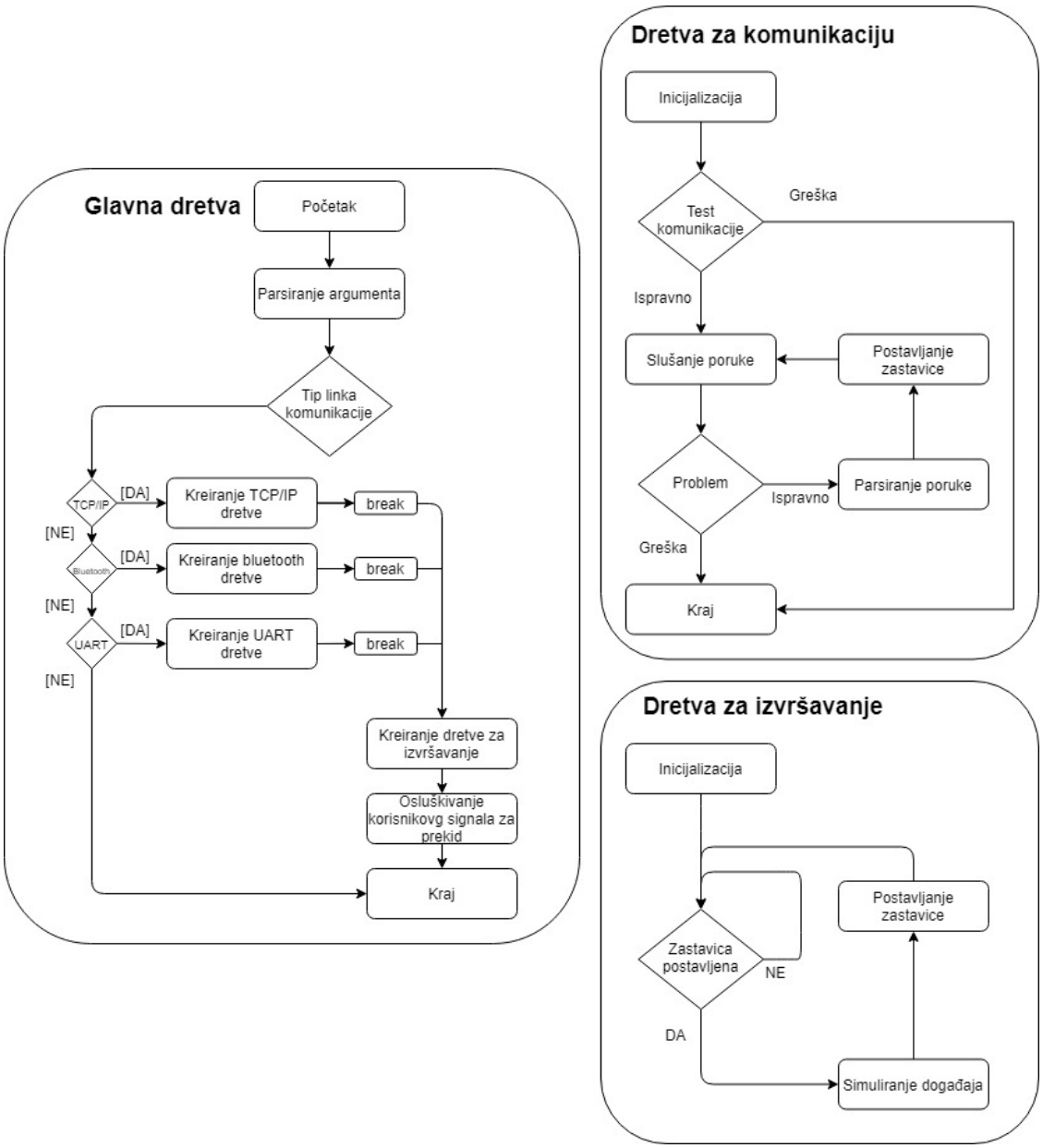
Slika 4.2 Dodavanje novog udaljenog sustava



Slika 4.3 Dodavanje udaljenog sustava u parametar projekta

4.2 Glavna funkcija upravljačkog programa

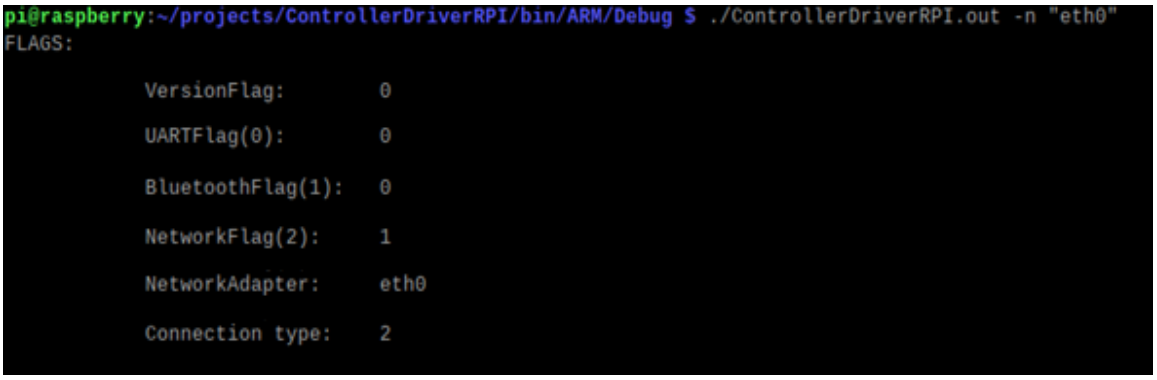
Glavna funkcija programa ima dvije uloge. S obzirom da program podržava argumente pri svom pozivu odnosno pokretanju, ujedno to automatski znači da i glavna funkcija prima argumente, pa je prva uloga primanje i obrada istih. Druga je uloga kreiranje dretvi. Jedna dretva služi za osluškivanje poruka odnosno za kreiranje strujanja između ovog upravljačkog programa i desktop aplikacije na posrednom uređaju, dok je druga zadužena za simuliranje događaja ulaznih uređaja. Tijek obje dretve prikazan je na slici 4.4.



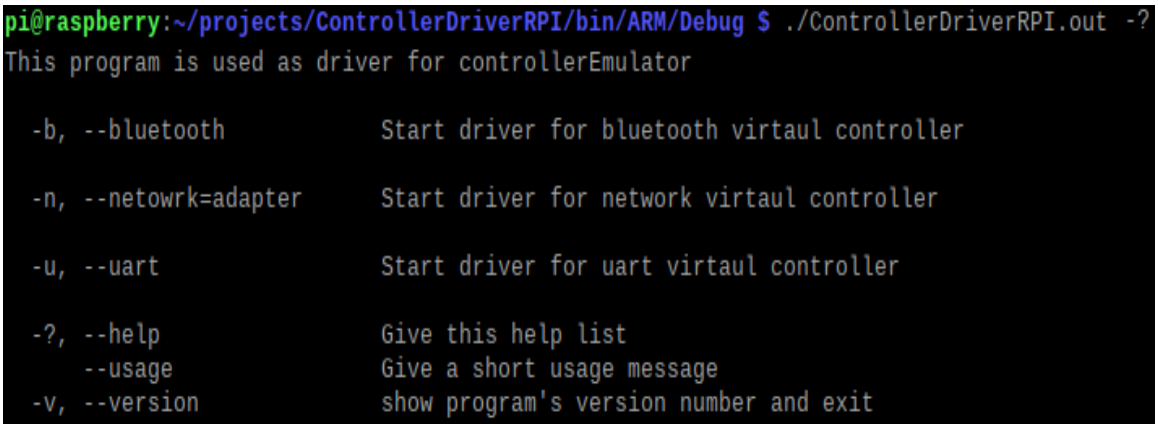
Slika 4.4 Dijagram tijeka upravljačkog programa

4.3 Parsiranje argumenata glavne funkcije

S obzirom da je glavna funkcija parametarska funkcija, potrebno je izvući korisne informacije iz samog poziva funkcije odnosno poziva upravljačkog programa. Argumenti glavne funkcije su cjelobrojni broj *argc* (engl. *Argument Count*) te dvostruki pokazivač tipa *char argv* (engl. *Arguemnt Values*). Prvi argument govori koliko je argumenata predano, dok drugi u obliku niza stringova nosi informaciju o argumentima. Za potrebe ovog parsiranja kreirana je jednostavna biblioteka temeljena na postojećoj kompleksnijoj *argp* biblioteci iz [8], odnosno kreirana je omotačka (engl. *Wraper*) biblioteka. Unutar biblioteke definiran je novi strukturni tip *ARGUEMTNS* na način da u njega mogu biti pohranjeni podaci iz poziva programa. Temeljna funkcija ove biblioteke je *ArgpParser*, koja prima dva parametra glavne funkcije (*argc* i *argv*) te adresu na već alociranu strukturu tipa *ARGUMENTS* gdje će rezultati parsiranja biti pohranjeni. Za pravilno funkcioniranje biblioteke također je potrebno definirati vrijednosti određenih varijabli unutar biblioteke. Ovisno u prihvatljivim argumentima odnosno opcijama potrebno je unutar *argumentParser.cpp* datoteke izmijeniti vrijednosti polja strukture *options*, odnosno potrebno je definirati ključni znak, ključnu riječ, tip opcije (parametar, zastava, alias i sl.) i opis za svaku moguću opciju programa izuzev opcije pomoći i opcije dokumentacije koje su već ugrađene. *ArgpParser* funkcija za početak postavlja inicijalne vrijednosti elemenata strukture *arguments*, zatim analizira niz stringova iz *argv* te na pojavu oznake iz opcije izvršava *case* blok definiran za tu opciju te ondje postavlja vrijednosti strukture arguments. Osim samog parsiranja, ovom bibliotekom omogućen je ispis pomoćne poruke te dokumentacije argumenata, također definirana je funkcija koja ispisuje rezultate parsiranja. Na slici 4.5 prikazana je snimka zaslona terminala pri pokretanju upravljačkog programa uz parametre *-n „eth0“* koja označava da će se koristiti TCP/IP link, odnosno network adapter imena *„eht0“* što je i vidljivo na ispisu parsiranja. Na slici 4.6 vidljivo je kako ispisati pomoćnu poruku te što je ondje opisano, odnosno opisi pojedenih argumenata upravljačkog programa.



Slika 4.5 Ispis rezultata parsiranja



Slika 4.6 Poruka pomoći upravljačkog programa

4.4 Dretva za komunikaciju

Nakon što se iz argumenata utvrdi koji se link komunikacije treba koristiti, potrebno je i pokrenuti strujanje na tom linku. U prototipu ovog upravljačkog programa definirana je dretva koja koristi TCP/IP način komunikacije. Prije pokretanja dretve uspostavlja se komunikacija. Dohvaćaju se podaci o IP adresi uređaja na kojem je program pokrenut, zatim se otvara unaprijed definiran port, a nakon toga se osluškuje prva poruka upućena na IP adresu na zadanom portu. Ukoliko čvor s druge strane, odnosno posredni uređaj odgovori porukom istog sadržaja koja je i poslana zaključuje se da je veza uspostavljena te se u tom slučaju može kreirati i započeti dretva za komunikaciju. Za registraciju dretve potrebno je definirati njezinu startnu funkciju. Startna funkcija, ujedno i glavna funkcija dretve sadrži jednu beskonačnu petlju unutar koje se konstantno osluškuje poruka na ranije uspostavljanom komunikacijskom linku. Vrše se provjere veličine pojedinog djela poruke, te ukoliko ne odgovara očekivanoj veličini petlja se prekida, zatim dretva te na kraju i sam upravljački program jer je došlo do neočekivane poruke izazvane problemom u komunikaciji. S obzirom da se radi o jednostavnim porukama, unaprijed definiranog formata, pripremljena je i jednostavna funkcija koja iz strujanja linka za komunikaciju izvlači zadani broj bajtova te ga zapisuje na predanu adresu. Nakon što je poruka na taj način parsirana rezultat je struktura tipa *Message* koja sadrži podatke vidljive u tablici 4.1 te u kodu 4.1. Za kraj jedne iteracije u petlji postavlja se zastavica *recivedEventFlag*, odnosno vrijednost globalne varijable se mijenja te na osnovu toga iduća dretva mijenja svoje ponašanje. U primjeru iz tablice 1 prikazana je poruka pritiska tipke Y na tipkovnici iz scan koda, međutim posredno računalo ima postavljen različit raspored tipkovnice (engl. *Layout*) pa je prema tome kod tipke u operacijskom sustavu zapravo Z.

```
typedef struct Message
{
    int deviceType;
    int eventType;
    int keyCode;
    int scanCode;
    int size;
    uint8_t* content;
}Message;
```

Kod 4.1 Prikaz strukture poruke

Device type				Event type				Key code				Scan code			
00	00	00	00	00	00	00	01	00	00	00	2C	00	00	00	15
Keyboard				Pressed				key Z				key Y			

Tablica 4.1 Prikaz poruke u kanalu

4.5 Dretva za izvršavanje

Nakon što je kreirana dretva za komunikaciju, potrebno je kreirati i dretvu za izvršavanje događaja ulaznih uređaja koji će biti opisani svakom porukom u komunikacijskom kanalu. Ponovno, za kreiranje dretve potrebna je startna funkcija. *DeviceThread* iz vlastite *inputDevice* biblioteke je startna funkcija ove dretve. Funkcija na svojem početku inicijalizira datoteke mogućih ulaznih uređaja, a zatim u beskonačnoj petlji provjerava stanje *recivedEventFlag* zastavice spomenute u prethodnom ulomku. Ukoliko je zastavica postavljena, odnosno ukoliko je njezina vrijednost različita od „0“ poziva se funkcija traženog uređaja kojem se predaje poruka iz kanala. Svaki od podržanih ulaznih uređaja ima funkciju koja se poziva na primljenu poruku, ukoliko je poruka vezana za taj uređaj. Unutar te funkcije, poruka se rekonstruira prema parametrima potrebnim za *uinput* biblioteku, primjerice potrebno je rekonstruirati stanje kontrolera iz *RAW input* podataka u podatke kompatibilne za *uinput* pisanje opisane u dokumentaciji biblioteke prema [9].

4.6 Korištenje upravljačkog programa na ciljanoj platformi

Za korištenje ovog upravljačkog programa potrebno je imati pristup terminalu ciljanog uređaja te na njemu imati izvršnu datoteku ovog projekta. Ukoliko ne postoji direktna veza, moguće je koristiti SSH kako je i spomenuto u poglavlju 2. Kako bi upravljački program imao

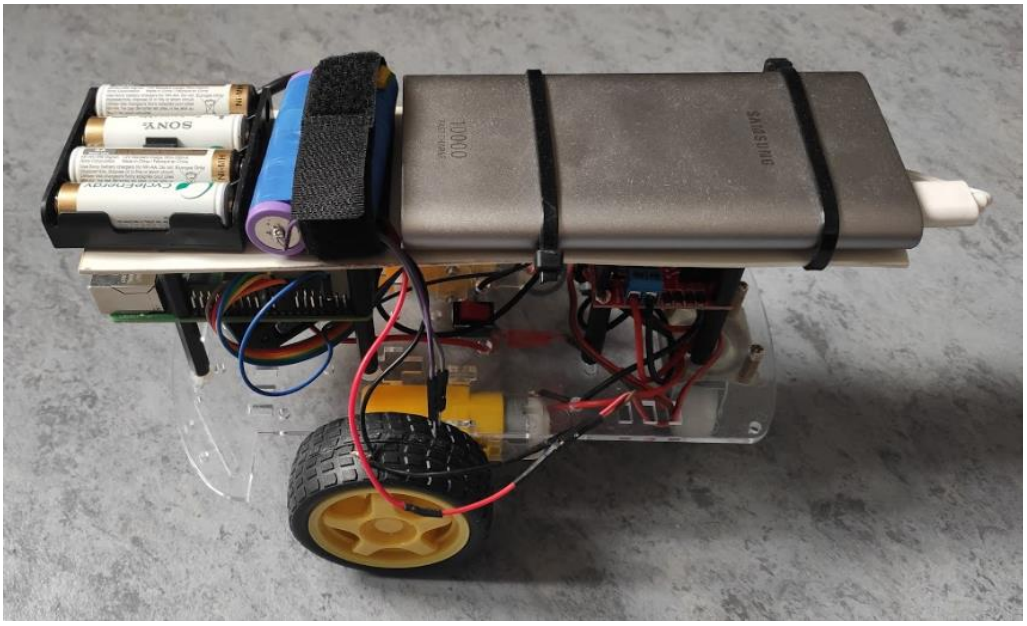
ovlasti kreirati datoteku ulaznog uređaja isti je potrebno pokrenuti sa *super user* privilegijama, odnosno koristeći ključnu riječ *sudo*. Ukoliko to nije moguće činiti svaki puta, onda je potrebno samo jednom, tj. prije prvog pokretanja postaviti ovlasti za direktoriji uređaja koristeći navedeni kod 4.2. Nadalje, na slici 4.6 vidljivo je koje parametre treba predati uz pokretanje upravljačkog programa.

```
user@machine: chmod -R 777 /dev
```

Kod 4.2 Komanda za promjenu ovlasti mape uređaja

5. Demo projekt

U sklopu ovoga rada napravljen je demo projekt na kojem je prototip testiran, vidljivo na slici 5.1. Ideja demo projekta bila je modelom automobila upravljati pomoću *gamepada* bez da je isti direktno priključen na sklopovlje modela. Za postizanje tog cilja, glavni kontroler ovog modela imao je osiguranu konstantnu internetsku vezu te je putem TCP/IP-a bio povezan s aplikacijom na posrednom uređaju.



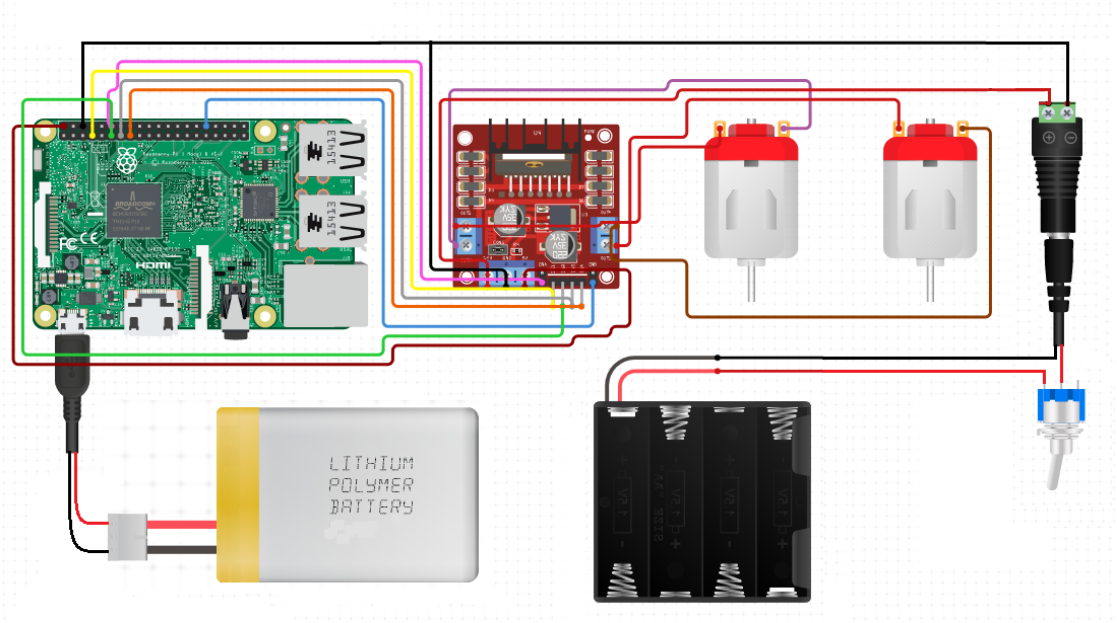
Slika 5.1 Model automobila

5.1 Sklopovlje

Za izradu ovog modela korišteno je iduće sklopovlje:

- Rasppbery Pi 4
- MicroSD card
- DC motor x 2
- L298N – upravljač DC motora
- 5V Lithium Ion prijenosna baterija s USB portom
- USB A to USB C kabal
- Prekidač
- Battery holder + Lithium Ion punjive baterije
- Vodići
- Plastična konstrukcija s kotačima i odstojnicima

RPI u ulozi glavnog kontrolera napajan je prijenosnom 5V Li-Ion baterijom preko USB kabla te ja na taj način omogućena autonomija samog modela. Nadalje, RPI upravlja s L298N kontrolerom prema [10] te su međusobno povezani sa 7 vodiča – po jedan za pulsno širinski modulacijski (engl. *Puls Width Modulation* - PWM) signal pojedinog motora te po 2 logička kontakta za određivanje smjera rotacije pojedinog motora, preostali vodič služi za izjednačavanje potencijala u svrhu osiguranja jednakog tumačenja logičke razine na oba kontrolera, odnosno ovaj vodič spaja GND kontakt RPI s GND kontaktom L298N kontrolera. Osim 7 vodiča spojenih na RPI, L298N je posredno preko prekidača spojen na izvor koji se koristi za napajanje motora. S obzirom da L298N radi na napajanju između 5V i 12V te da 2 DC motora mogu ostvariti potrošnju preveliku za klasične AAA baterije, za napajanje su korištene dvije 18650 Li-Ion baterije spojene u seriju te tako omogućile napon od 9V te dovoljnu snagu za korištenje 2 DC motora istovremeno na 100% snage. Opis ovog spoja prikazan je na slici 5.2, te je tako spojen spreman za korištenje, naravno uz pripadajuću programsku podršku.



Slika 5.2 Dijagram ožičenja

5.2 Programska podrška

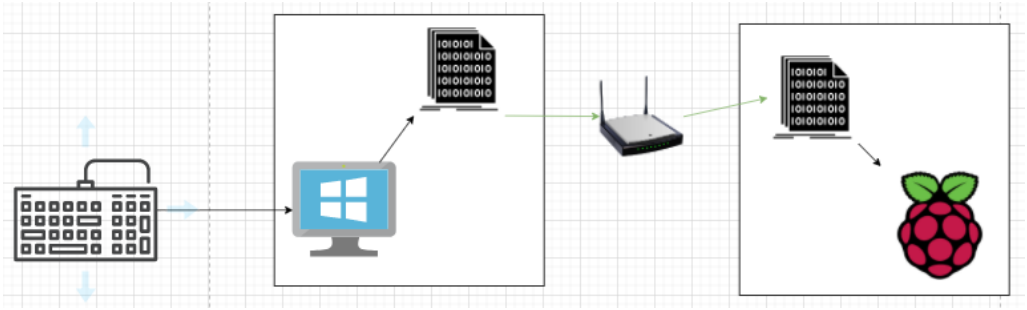
Za operacijski sustav RPI mikro računala instaliran je *Raspberry Pi operating system* također poznat pod imenom *Raspbian operating system*, što je verzija *Debian* operacijskog sustava. Zajedno s ovim operacijskim sustavom instaliran je i *python* interpretator. *Python* je potreban za izvršavanje programa koji upravlja RPI GPIO kontaktima na koje je spojen L298N. Ovaj program pri pokretanju pokušava pronaći ulazni uređaj tipa *gamepad* te u slučaju uspjeha

nastavlja osluškivati strujanje uređaj. Ovisno o podacima koji se pojave u strujanju, konkretno položaju analognih usmjerivača, program će izmjenjivati logičke vrijednosti na GPIO kontaktima kako bi odredio smjer i brzinu rotacije pojedinog kotača. Više detalja o ovom programu moguće vidljivo je u prilogu **P.2**. Naravno, kako bi prototip ovog sustava bio omogućen i na demo projektu, potrebno je instalirati i pokrenuti upravljački program iz poglavlja broj 4.

Za postavljanje programske podrške potrebno je preuzeti .iso datoteku *Raspbian* operacijskog sustava te na formatiranu *microSD* karticu prenijeti sadržaj .iso datoteke. Ovo je moguće napraviti automatski koristeći neki od alata poput *Raspberry PI Imager*. Nadalje, potrebno je RPI spojiti na mrežu. Najlakši način je spajanjem s *ethernet* kabelom direktno u usmjerivač ili preklopnik. Zatim, spajanjem na upravljačko sučelje usmjerivača saznati dodijeljenu IP adresu. Za slučaj da pristup sučelju usmjerivača nije dostupan, skeniranjem mreže pomoću *Advanced IP scanner* alata može se također saznati IP adresa traženog uređaja. Kada je poznata IP adresa uređaja moguće je spojiti se pomoću SSH koristeći primjerice *putty*. Ukoliko je konekcija preko SSH uspješno ostvarena, programska podrška je kompletno postavljena.

6. Rezultati testiranja sustava

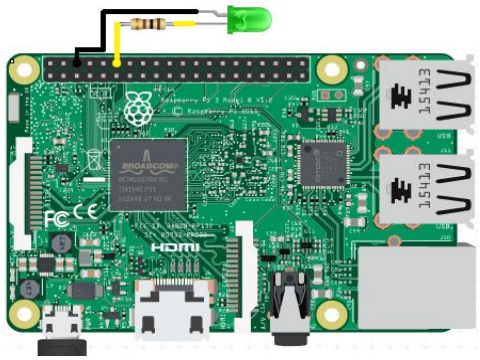
Pri korištenju ovoga sustava vrlo važan parametar je latencija koja se pojavljuje uvođenjem nekoliko dodatnih čvorova između ciljane platforme i ulaznog uređaja. Na slici 6.1 prikazana je topologija u slučaju korištenja TCP/IP protokola za komunikaciju između ciljane platforme i posrednog uređaja arhitektura. Vidljivo je kako će na latenciju utjecati vremena potrošena za obradu događaja na posrednom uređaju, zatim *upload/download* i obrada na usmjerivaču (engl. *router*) te obrada događaja na upravljačkom programu na ciljanoj platformi.



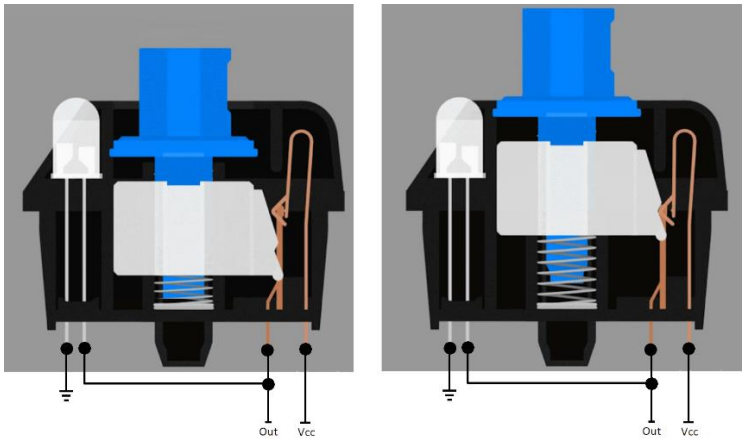
Slika 6.1 Topologija sklopovskih jedinica i jedinica programske podrške

Za testiranje latencije korišteno je testno okruženje sačinjeno od mehaničke tipkovnice s ugrađenim *chery blue swithchevima* slični onima sa slike 6.3, RPI kao ciljana platforma s upravljačkim programom sustava, LE Dioda spojena na RPI GPIO kontakte, kamera s mogućnosti snimanja *slow motion* videozapisa, *windows* računalo s aplikacijom sustava, video preglednik. Na ciljanoj platformi pokrenut je program, odnosno *python* skripta koja će pri pritisku neke od tipki reagirati tako što će jedan od GPIO pinova postaviti u stanje high. Na spomenutom pinu je spojena anoda LE diode, dok je katoda LE diode preko otpornika spojena na ground RPI kontakata kao na slici 6.2. Ovim je postignuto da pritiskom tipkovnice LE Dioda zasvijetli.

Slika 6.2 Prikaz LE diode na RPI



Blue chery switchevi s ugrađenim pozadinskim osvjetljenjem također će zasvijetliti pri pritisku tipke. Oba zasvijetljenja, promatranjem golim okom se događaju u istom trenutku, međutim realnost je da ipak postoji vremenski razmak, odnosno kašnjenje LE Diode spojenu na RPI kontakte u odnosu na onu ugrađenu u tipku tipkovnice.



Slika 6.3 Prikaz chery blue switcheva

Da bi se izmjerilo kašnjenje koristi se kamera s mogućošću snimanja usporenog filma (engl. *slow motion*). Poželjno je imati što veći broj slika u sekundi (engl. *Frame per second* - FPS), kako bi mjerenje bilo preciznije. Potrebno je namjestiti kameru tako da se tokom snimanja u sceni nalaze obje LE diode. Nakon što je snimljeno nekoliko testnih snimaka, nastavak analize vrši se u nekom od alata za pregled i uređivanje video materijala, primjerice VLC media player. Pomoću video preglednika potrebno je pronaći indeks slike u kojem LE dioda tipke prvi puta zasvijetli, te slike u koje LED dioda spojena na RPI prvi puta zasvijetli. Znajući FPS kamere iz specifikacije, ili izračunavanjem iz podataka video materijala prema izrazu 6.1, gdje je *N* ukupni broj slika u videu, a *t* ukupno vrijeme trajanja videa u sekundama.

$$FPS = \frac{N}{t} \tag{6.1}$$

Sada, kad je *FPS* poznat, vrijeme između dva trenutka na dvije različita slike lako se izračuna prema izrazu 6.2, gdje je I_{t_1} index video okvira u trenutku t_1, a I_{t_2} index video okvira u trenutku t_2.

$$t = \frac{|I_{t_1} - I_{t_2}|}{FPS} \tag{6.2}$$

Također, treba uzeti u obzir da korištenje tipkovnice u direktnoj vezi također ima određenu latenciju. Naime, tipkovnica ima definiranu frekvenciju kojom CPU provjerava podatke na USB

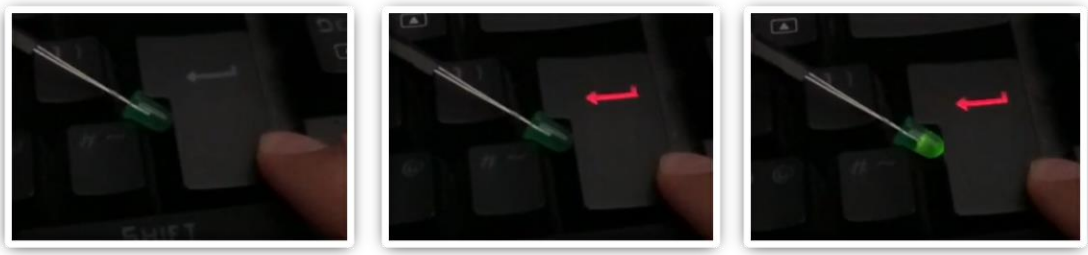
sabirnici (engl. *polling rate*),, što znači da pri pritisku tipke prođe neko vrijeme dok ono bude registrirano od strane CPU-a. Osim *pulling ratea*, važna je i frekvencija mikroprocesora same tipkovnice koji skenira čitavu tipkovnicu (engl. *matrix scan rate*) i ovisno o stanju zapisuje podatke na sabirnicu. Prema tome za određivanje latencije koju uvodi sustav potrebno je ponoviti isto mjerenje pri direktnoj vezi tipkovnice i RPI. Konačni iznos uvedene latencije može se iskazati prema 6.3, gdje je t_1 latencija u sustavu za posredno korištenje ulaznih uređaja, t_2 latencija u direktnom sustavu.

$$t = t_1 - t_2 \tag{6.2}$$

6.1 Rezultati mjerenja

Postupak mjerenja ponovljen je 20 puta za oba slučaja. U tablici 6.2 prikazani su dobiveni rezultati jedne od iteracija opisanog mjerenja pri direktnoj vezi tipkovnice i ciljane platforme, dok su u tablici 6.3 prikazani rezultati mjerenja kada je tipkovnica spojena pomoću sustava za posredno korištenje ulaznih uređaja. Obje tablice prikazuju jedno od mjerenja čiji rezultati po pitanju ΔI_t, odnosno po ukupnoj latenciji odgovaraju srednjoj vrijednosti ya cjelokupno mjerenje. Cjelokupni rezultati mjerenja dani vidljivi su u prilogu **P.1**. Za bolje razumijevanje tablica mjerenja na tablici 6.1 prikazana je legenda.

Indeks okvira	$I_{t_1} - 1$	I_{t_1}	I_{t_2}
Trenutak	t_{1-1}	t_1	t_2
Opis	Okvir neposredno prije pritiska tipke	Okvir u trenutku pritiska tipke. Prvi okvir sa svjetlećom LE diodom tipke i ne svjetlećom LE diodom spojene na ciljanu platformu	Okvir u trenutku detekcije pritiska tipke na ciljanoj platformi. Prvi okvir sa svjetlećom LE diodom tipke i LE diodom spojene na ciljanu platformu



Tablica 6.1 Legenda

FPS	240
I_{t_1}	433
I_{t_2}	439
t	25 ms

Tablica 6.2 Rezultati latencije - direktni spoj

FPS	240
I_{t_1}	316
I_{t_2}	332
t	66.67 ms

Tablica 6.3 Rezultati latencije – spoj preko sustava za posredno korištenje

Kombiniranjem rezultata mjerenja te uvrštavanjem u izraz 6.2 ukupna latencija sustava iznosi $t = 41{,}67\ ms$. Na primjeru ove opreme, uvedena latencija dodaje dodatnih 65% vremena, što rezultira ukupnom latencijom od 66,67 *ms*. Ova brojka je daleko od idealnog uspoređujući s generalnim rezultatima mehaničke tipkovnice, međutim uzimajući u obzir da postoje tipkovnice koje u direktnoj vezi imaju latenciju od 60 *ms*. Također, autori znanstvenog istraživanja iz [11], navode da je frekvencija tipkanja ograničena i iznosi približno 7 HZ, što je približno 130 *ms* između dva pritiska, pa se može reći da ovaj sustav unosi prihvatljivu latenciju.

7. Zaključak

U ovom diplomskog radu napravljen je sustav za omogućavanje korištenja ulaznih uređaja na nekoj ciljanoj platformi bez da su ulazni uređaji i platforma u direktnoj vezi, već su oni indirektno spojeni preko posrednog uređaja. Sustav je sačinjen od dvije aplikacije – upravljačkog program na ciljanom uređaju te aplikacije na posrednom uređaju. Prototip je predstavio mogućnost korištenja nekoliko primjera ulaznih uređaja spojenih na uređaj s *Windows* operacijskim sustavom te su događaji tih ulaznih uređaja uspješno simulirani na drugom uređaju s *Debian* operacijskim sustavom. Osim dvije osnovne aplikacije sustava, u sklopu ovoga rada je pripremljen i demo projekt kroz koji je predstavljen i jedna od mogućih primjena te se kroz demo projekt odradilo testiranje gdje se dokazalo da je izrađeni sustav upotrebljiv. Sustav je ispunio zahtjeve postavljene u zadatku ovog rada. Rezultati testiranja latencije dokazali su da uvedeno vrijeme kašnjenja ne stvara probleme, odnosno sustav kao takav može bit korišten u normalnim okolnostima. Tijekom izrade ovog rada vođeno je računa o arhitekturi programske podrške na obje komponente, te su time postavljeni temelji za lakše unapređenje i ubacivanje novih pogodnosti kao što su novi načini komunikacije ili novi tipovi uređaja. Osim toga u budućim nadogradnjama ovoga sustava plan je izraditi postavljački program upravljačkog programa te objaviti desktop aplikaciju na *Microsoft Store* servisu za posredna računala.

Literatura

[1] VNC documentation, https://www.realvnc.com/en/connect/_downloads/VNC_User_Guide.pdf [28.11.2020.]

[2] SSH documentation, https://www.ssh.com/ssh/protocol/ [28.11.2020.]

[3] Mouse Without Borders documentation, https://www.microsoft.com/en-us/download/details.aspx?id=35460 [23.8.2021.]

[4] Microsoft WPF documentation, https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-5.0 [20.06.2021.]

[5] Mateljan, Vladimir ; Đambić, Goran ; Drenovac, Sergej; „New Languages and Technologies for Delivering Learning and Multimedia Contents – XAML and WPF“, 2008.

[6] RAWinput documentation, https://docs.microsoft.com/en-us/windows/win32/inputdev/about-raw-input [20.06.2020.]

[7] Visual studio documentation, https://docs.microsoft.com/en-us/visualstudio/windows/?view=vs-2019 [8.8.2021]

[8] Argp documentation, https://www.gnu.org/software/libc/manual/html_node/Argp.html [8.8.2021]

[9] Uinput https://01.org/linuxgraphics/gfx-docs/drm/input/uinput.html [8.8.2021]

[10] L298N data sheet https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf [8.8.2021]

[11] Duprez. J.; Stokkermans. M; Drijvers, L; Cohen. M; „Synchronization between keyboard typing and neural oscillations“ 25.8.2020. https://www.biorxiv.org/content/10.1101/2020.08.25.264382v1

Popis i opis upotrjebljenih kratica

VNC	Virtual Network Computing
SSH	Secure Shell
RPI	Raspberry Pi
TCP/IP	Transimsion Control Protocol/Internet Protocol
WPF	Windows Presnetation Foundation
MS	Microsoft
VS	Visaul Studio
CLI	Comand Line Interface
GUI	Graphical User Interface
XAML	Extensible Application Markup Language
IDE	Integrated Development Environment
MVC	Model View Controller
MVVM	Model View ViewModel
MVP	Model View Presenter
FPS	Framse Per Second
API	Application Programming Interface
HID	Human Interface Device
GPIO	General purpose input output
PWM	Puls Width Modulation
CPU	Central Procesor Unit

Sažetak

U sklopu diplomskog rada predložen je sustav za posredno korištenje ulaznih uređaja kako bi se reducirao broj ulaznih uređaja u okruženju korisnika, omogućilo udaljeno upravljanje te omogućilo upravljanje čak i kada ne posjedujemo ulazni. Dizajn prototipa ovakvog sustava opisan je kroz ovaj rad. Obuhvaćene su faze razvoja programske podrške za dvije glavne komponente, aplikacija na posrednom uređaju te upravljački program na ciljanoj platformi. Uz to, opisana je izrada demo projekta kao i postupak testiranja odnosno mjerenja performansi koje su dokazale da ovakav sustav može biti upotrebljiv.

Ključne riječi

Udaljeno upravljanje; RPI; Ulazni uređaji; uinput bibloteka; rawinput biblioteka

SYSTEM FOR INDIRECT USE OF INPUT DEVICES

Abstract

As part of the thesis, a system for indirect use of input devices was proposed in order to reduce the number of input devices in the user environment, enable remote control and enable control even when we do not have an input. The prototype design of such a system is described through this paper. The stages of software development for the two main components, the application on the intermediate device and the driver on the target platform, are included. In addition, the development of a demo project is described, as well as the process of testing or measuring performance, which proved that such a system can be used.

Key words

Remote controll; RPI; Input devices; uninput library, rawinput library

Životopis

Luka Kruljac rođen je 25.3.1997. u Đakovu. Odrastao u Gašincima gdje je i pohađao Osnovnu školu J.A.Ćolnića od 1. do 4. razreda. Osnovnu školu od 5. do 8. razreda pohađa u istoimenoj školi u Satnici Đakovačkoj. Godine 2010. upisuje prirodoslovno-matematičku gimnaziju A.G.Matoš u Đakovu, istu završava 2015. godine kada upisuje program vojnog kadeta, smjer Vojno inženjerstvo. Program kadeta napušta iz osobnih razloga te u listopadu 2015. godine upisuje sveučilišni preddiplomski studij elektrotehnike na Elektrotehničkom fakultet Osijek. Za vrijeme preddiplomskog studija 2016. i 2017. odrađuje prakse u tvrtki Siemens Convergence Creators, a 2018. u tvrtki Inchoo. Nakon završenog preddiplomskog studija komunikacija i informacijskih tehnologija upisuje diplomski studij Automobilskog računarstva i komunikacija na istom fakultetu. Tijekom studija bio je član, a zatim i predsjednik Studentskog Zbora, te član studentskog ogranka IEEE Osijek, gdje također u jednom periodu obnaša neke od dužnosti uprave ogranka. Sudionik je studentskog Work&Travel programa u SAD-u tijekom ljeta 2019., a po povratku s programa, u jesen iste godine počinje raditi kao stipendist u tvrtki RT-RK koju napušta nakon gotovo godinu dana. Zatim, od jeseni 2020. godine sudjeluje na Erasmus Internship programu u tvrtki Infineon Technologies u Austriji gdje i danas radi kao software & component validation engineer.

U Osijeku, 26.6.2021.

Prilozi

Prilog P.1. Cjelokupni rezultati mjerenja latencije

Prilog P.2. Izvorni kod python skripte za upravljanje kotačima

Elektronička verzija rada

1. Na linku https://github.com/lkruljac/diplomskiV2 može se pronaći izvorni kod čitavog projekta
2. Na linku https://github.com/lkruljac/diplomskiV2/raw/master/Diplomski%20rad%20-%20Luka%20Kruljac%20_%20Sep21.docx može se pronaći elektronička verzija ovog dokumenta
3. Na linku https://drive.google.com/drive/folders/19gSh3u_O7lwJ6STntR2eCAQP0J6EXJQw može se pronaći demo prikazi sustava

Prilog P.1. – Cjelokupni rezultati mjerenja

Rezultati mjerenja - direktan spoj					
rbr.	FPS [Hz]	I_{t_1}	I_{t_2}	t [ms]	$\bar{t}$ [ms]
1	240	261	267	25	25
2	240	382	389	29,16	
3	240	433	439	25	
4	240	797	803	25	
5	240	967	973	25	
6	240	1481	1487	25	
7	240	2412	2418	25	
8	240	4104	4110	25	
9	240	5494	5499	20,83	
10	240	7101	7107	25	
11	240	8383	8389	25	
12	240	9521	9527	25	
13	240	10548	10554	25	
14	240	11296	11302	25	
15	240	11381	11387	25	
16	240	12267	12273	25	
17	240	12680	12686	25	
18	240	13025	13031	25	
19	240	13134	13140	25	
20	240	14135	14141	25	

Rezultati mjerenja – spoj preko sustava za posredno korištenje					
rbr.	FPS [Hz]	I_{t_1}	I_{t_2}	t [ms]	$\bar{t}$ [ms]
1	240	316	332	66,67	66,67
2	240	937	953	66,67	
3	240	1526	1542	66,67	
4	240	1998	2015	70,83	
5	240	2258	2274	66,67	
6	240	2928	2944	66,67	
7	240	3489	3506	70,83	
8	240	4354	4370	66,67	
9	240	4896	4912	66,67	
10	240	6146	6162	66,67	
11	240	7120	7135	62,50	
12	240	8756	8772	66,67	
13	240	9893	9909	66,67	
14	240	10994	11010	66,67	
15	240	12621	12636	62,50	
16	240	13164	13180	66,67	
17	240	14370	14386	66,67	
18	240	14456	14472	66,67	
19	240	15425	15441	66,67	
20	240	15688	15704	66,67	

Prilog P.2. – Izvorni kod python skripte za upravljanje kotačima

```
import os, struct, array
from fcntl import ioctl
import RPi.GPIO as GPIO
from time import sleep

global RPWM
global RIN1
global RIN2
global LIN1
global LIN2
global LPWM

def main():
    print('Available devices:')

    for fn in os.listdir('/dev/input'):
        if fn.startswith('js'):
            print('  /dev/input/%s' % (fn))

    axis_states = {}
    button_states = {}

    axis_names = {
        0x00 : 'x',
        0x01 : 'y',
        0x02 : 'z',
        0x03 : 'rx',
        0x04 : 'ry',
        0x05 : 'rz',
        0x06 : 'trottle',
        0x07 : 'rudder',
        0x08 : 'wheel',
        0x09 : 'gas',
        0x0a : 'brake',
        0x10 : 'hat0x',
        0x11 : 'hat0y',
        0x12 : 'hat1x',
        0x13 : 'hat1y',
        0x14 : 'hat2x',
        0x15 : 'hat2y',
        0x16 : 'hat3x',
        0x17 : 'hat3y',
        0x18 : 'pressure',
        0x19 : 'distance',
        0x1a : 'tilt_x',
        0x1b : 'tilt_y',
        0x1c : 'tool_width',
        0x20 : 'volume',
        0x28 : 'misc',
    }

    button_names = {
        0x120 : 'trigger',
        0x121 : 'thumb',
        0x122 : 'thumb2',
        0x123 : 'top',
        0x124 : 'top2',
        0x125 : 'pinkie',
        0x126 : 'base',
```

```
    0x127 : 'base2',
    0x128 : 'base3',
    0x129 : 'base4',
    0x12a : 'base5',
    0x12b : 'base6',
    0x12f : 'dead',
    0x130 : 'a',
    0x131 : 'b',
    0x132 : 'c',
    0x133 : 'x',
    0x134 : 'y',
    0x135 : 'z',
    0x136 : 'tl',
    0x137 : 'tr',
    0x138 : 'tl2',
    0x139 : 'tr2',
    0x13a : 'select',
    0x13b : 'start',
    0x13c : 'mode',
    0x13d : 'thumbl',
    0x13e : 'thumbr',

    0x220 : 'dpad_up',
    0x221 : 'dpad_down',
    0x222 : 'dpad_left',
    0x223 : 'dpad_right',
}

axis_map = []
button_map = []

fn = '/dev/input/js0'
print('Opening %s...' % fn)
jsdev = open(fn, 'rb')

buf = array.array('B', [0] * 64)
ioctl(jsdev, 0x80006a13 + (0x10000 * len(buf)), buf)
js_name = buf.tobytes().rstrip(b'\x00').decode('utf-8')
print('Device name: %s' % js_name)

buf = array.array('B', [0])
ioctl(jsdev, 0x80016a11, buf)
num_axes = buf[0]

buf = array.array('B', [0])
ioctl(jsdev, 0x80016a12, buf)
num_buttons = buf[0]

buf = array.array('B', [0] * 0x40)
ioctl(jsdev, 0x80406a32, buf)
for axis in buf[:num_axes]:
    axis_name = axis_names.get(axis, 'unknown(0x%02x)' % axis)
    axis_map.append(axis_name)
    axis_states[axis_name] = 0.0

buf = array.array('H', [0] * 200)
ioctl(jsdev, 0x80406a34, buf)

for btn in buf[:num_buttons]:
    btn_name = button_names.get(btn, 'unknown(0x%03x)' % btn)
    button_map.append(btn_name)
```

```
    button_states[btn_name] = 0

print('%d axes found: %s' % (num_axes, ', '.join(axis_map)))
print('%d buttons found: %s' % (num_buttons, ', '.join(button_map)))


RPWM = 11
RIN1 = 10
RIN2 = 9
LIN1 = 24
LIN2 = 23
LPWM = 25

GPIO.setmode(GPIO.BCM)
GPIO.setup(RPWM, GPIO.OUT)
GPIO.setup(RIN1, GPIO.OUT)
GPIO.setup(RIN2, GPIO.OUT)

GPIO.setup(LIN1, GPIO.OUT)
GPIO.setup(LIN2, GPIO.OUT)
GPIO.setup(LPWM, GPIO.OUT)

rightPWM = GPIO.PWM(RPWM, 100)
leftPWM = GPIO.PWM(LPWM, 100)


while True:
    evbuf = jsdev.read(8)
    if evbuf:
        time, value, type, number = struct.unpack('IhBB', evbuf)



        if type & 0x01:
            button = button_map[number]
            if button:
                button_states[button] = value
                if value:
                    print("%s pressed" % (button))
                else:
                    print("%s released" % (button))
                GPIO.output(LPWM, 0)
                GPIO.output(RPWM, 0)

        if type & 0x02:
            axis = axis_map[number]
            rspeed = 0
            lspeed = 0
            speed=0
            if axis:
                fvalue = value / 32767.0
                axis_states[axis] = fvalue
                print("%s: %.3f" % (axis, fvalue))
                if axis == 'y':
                    speed = fvalue
                    rspeed = speed
                    lspeed = speed
                if axis == 'x':
                    if fvalue > 0 :
                        rspeed = speed - fvalue
                    elif fvalue < 0:
```

```
                    lspeed = speed + fvalue
                print("left: %.3f, right: %.3f" % (lspeed, rspeed))
                print("driving2")

                if speed > 0:
                    GPIO.output(RIN1, 0)
                    GPIO.output(RIN2, 1)
                    GPIO.output(LIN1, 0)
                    GPIO.output(LIN2, 1)
                else:
                    GPIO.output(RIN1, 1)
                    GPIO.output(RIN2, 0)
                    GPIO.output(LIN1, 1)
                    GPIO.output(LIN2, 0)
                rightPWM.start(abs(rspeed)*100)
                leftPWM.start(abs(lspeed)*100)

if __name__ == "__main__":
    print("starting")
    main()
```

```
public abstract class Connector : BaseViewModel
{
    #region Properites
    private string _Status;
    public string Status
    {
        get { return _Status; }
        set { _Status = value; RaisePropertyChangedEvent("Status"); }
    }
    private bool _IsConnected;

    public bool IsConnected
    {
        get { return _IsConnected; }
        set
        {
            _IsConnected = value;
            RaisePropertyChangedEvent("IsConnected");
            RaisePropertyChangedEvent("ConnectionStatus");
        }
    }
    private string _ConnectionStatus;
    public string ConnectionStatus
    {
        get
        {
            if (IsConnected){ _ConnectionStatus = "Connected"; }
            else { _ConnectionStatus = "Disconnected"; }
            return _ConnectionStatus;
        }
        set
        {
            _ConnectionStatus = value;
            RaisePropertyChangedEvent("ConnectionStatus");
        }
    }
    #endregion
    public abstract void SendMessage(byte[] data);
    public abstract void Connect();
}
```

Prilog