UNIVERZA V MARIBORU
FAKULTETA ZA ELEKTROTEHNIKO,
RAČUNALNIŠTVO IN INFORMATIKO

Mihael Flac

# VODENJE KVADROKOPTERJA S HAPTIČNIM VMESNIKOM NOVINT FALCON

Diplomsko delo

Maribor, september 2014


UNIVERZA V MARIBORU

FAKULTETA ZA ELEKTROTEHNIKO,

RAČUNALNIŠTVO IN INFORMATIKO

# VODENJE KVADROKOPTERJA S HAPTIČNIM VMESNIKOM NOVINT FALCON

**Diplomsko delo**


| | |
|---|---|
| Študent: | Mihael Flac |
| Študijski program: | univerzitetni študijski program Elektrotehnika |
| Smer: | Avtomatika in robotika |
| Mentor: | izr. prof. dr. Aleš Hace |
| Somentor: | / |


Maribor, september 2014

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

Številka: E1060324
Datum in kraj: 01. 04. 2014, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 46/2012)
izdajam

**SKLEP O DIPLOMSKEM DELU**

1. **Mihaelu Flacu,** študentu univerzitetnega študijskega programa ELEKTROTEHNIKA, smer Avtomatika in robotika, se dovoljuje izdelati diplomsko delo pri predmetu Mikrokrmilniki.

2. **MENTOR:** **izr. prof. dr. Aleš Hace**

3. **Naslov diplomskega dela:**
   **VODENJE KVADROKOPTERJA S HAPTIČNIM VMESNIKOM NOVINT FALCON**

4. **Naslov diplomskega dela v angleškem jeziku:**
   **QUADCOPTER CONTROL BY HAPTIC INTERFACE NOVINT FALCON**

5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela" in ga oddati v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije do 30. 09. 2014 v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:
red. prof. dr. Borut Žalik

UNIVERZA V MARIBORU FAKULTETA ZA ELEKTROTEHNIKO RAČUNALNIŠTVO IN INFORMATIKO 8

Obvestiti:
- kandidata,
- mentorja,
- odložiti v arhiv.

# ZAHVALA

*Zahvaljujem se mentorju izr. prof. dr. Alešu Hacetu za omogočenje izvedbe diplomskega dela ter za nasvete in pomoč med samim potekom izvedbe.*

*Zahvaljujem se tudi sodelavcu v laboratoriju dipl. ing. Roku Pučku za pomoč pri praktičnem delu diplomskega dela.*

*Posebna zahvala velja mojim staršem za omogočanje študija in spodbujanje v celotnem času šolanja.*

*Zahvaljujem se tudi prijatelju in sostanovalcu Mihaelu Sklepiću za podporo v času študija.*

# VODENJE KVADROKOPTERJA S HAPTIČNIM VMESNIKOM NOVINT FALCON

**Ključne besede:** kvadrokopter, haptični vmesnik, regulacija, UDP, WiFi, AT-ukazi.

**UDK:** 681.51: 533.664(043.2)

**Povzetek**

*V delu je opisan kvadrokopter AR.Drone 1.0 proizvajalca Parrot, ki ga vodimo s haptičnim vmesnikom Novint Falcon. Opisana sta tudi haptični vmesnik in potek komuniciranja s kvadrokopterjem prek brezžične komunikacije in uporabe UDP-protokola. Podrobneje so opisani AT-ukazi, uporabljeni za vodenje kvadrokopterja. Predstavljeno je delovanje operacijskega sistema Linux, na katerem je narejeno diplomsko delo. Opisan je program za vodenje kvadrokopterja s haptičnim vmesnikom, podani pa so tudi rezultati njegovega vodenja in testiranja.*

# QUADCOPTER CONTROL BY HAPTIC INTERFACE NOVINT FALCON

**Key words:** quadcopter, haptic interface, control, UDP, WiFi, AT-commands

**UDK:** 681.51: 533.664(043.2)

**Abstract**

*In the work described quadcopter AR.Drone 1.0 companies Parrot, which is controlled with haptic interface Novint Falcon. Described is also haptic interface and the course of communicating with quadcopter via wireless communications and the use of UDP-protocol. Detailed are described AT-commands used for controlling quadcopter. Presented is the operating system Linux, on which is made graduation work. Described is a program for controlling quadcopter with haptic interface and presents the results of controlling and testing quadcopter.*

# KAZALO VSEBINE

# KAZALO SLIK

## UPORABLJENI SIMBOLI

$\Omega_n$ – kotna hitrost n-tega rotorja

$\Delta_A$ – sprememba kotne hitrosti (povečanje)

$\Delta_B$ – sprememba kotne hitrosti (zmanjšanje)

$\Phi$ – zasuk ali kotaljenje, kot okoli osi x

$\vartheta$ – naklon, kot okoli osi y

$\Psi$ – odklon, kot okoli osi z

## UPORABLJENE KRATICE

**UDP** – (angl. User Datagram Protocol)

**TCP** – (angl. Transmission Control Protocol)

**FTP** – (angl. File Transfer Protocol)

**IP** – (angl. Internet Protocol)

**WiFi** – (angl. Wireless-Fidelity)

**USB** – (angl. Universal Serial Bus)

**RAM** – (angl. Random Accrss Memory)

**2D** – dvodimenzionalni prikaz

**3D** – tridimenzionalni prikaz

**Hz** – merska enota za frekvenco (herc)

**W** – merska enota za moč (watt)

**m** – merska enota za dolžino (meter)

**N** – merska enota za silo (njuton)

**roll** – zasuk ali kotaljenje (rotacija okoli osi x)

**pitch** – naklon (rotacija okoli osi y)

**yaw** – odklon (rotacija okoli osi z)

**CW** – (angl. clockwise) – v smeri urinega kazalca

**CCW** – (angl. counterclockwise) – v nasprotni smeri urinega kazalca

**SDK** – (angl. Software Development Kit) – programski paket

**Firmware** – programska oprema

**LED** – (angl. light-emitting diode)

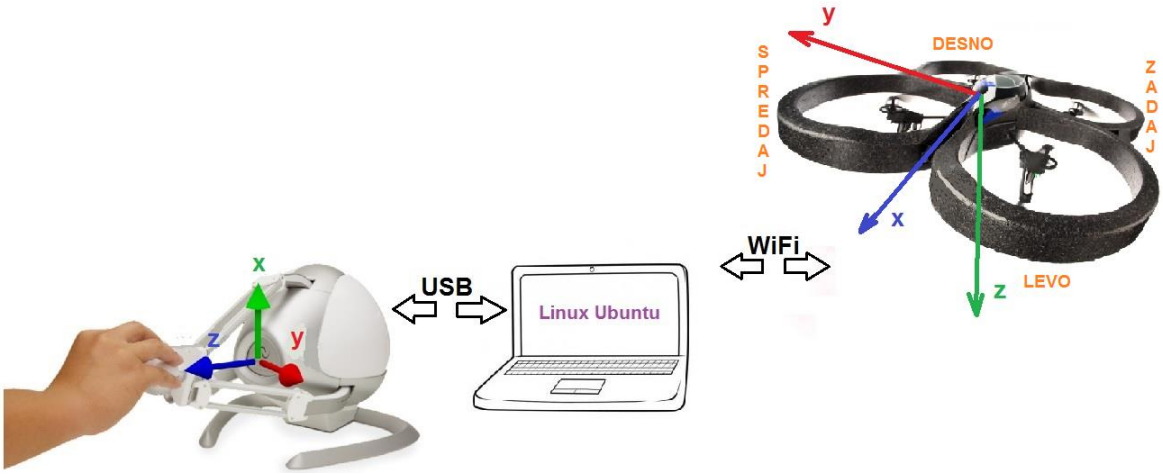
# 1 UVOD

Kvadrokopterji postajajo vedno bolj popularni. Najpogosteje se uporabljajo za snemanje, pregledovanje ali nadzor iz zraka (polja, električna in druga infrastruktura, jezovi, mostovi, jaški za dvigala in ventilacijski kanali ter drugi težko dostopni objekti), pri tem pa jih človek upravlja na daljavo. Prav zato potrebuje optični stik s kvadrokopterjem, kar omejuje področje delovanja. V realnem neurejenem 3D-okolju je pogosto tudi veliko ovir, zaradi katerih je gibanje omejeno, posledično pa je tudi veliko možnosti za nenadzorovane trke in poškodbe. Haptični vmesnik omogoča intuitivno upravljanje robotskih sistemov. Prek haptičnega komunikacijskega kanala lahko operaterju sporočimo različne informacije o stanju kvadrokopterja iz njegovega lokalnega okolja in na tak način olajšamo daljinsko vodenje kvadrokopterja.

Cilj diplomskega dela je omogočiti lažje vodenje kvadrokopterja v realnem neurejenem okolju. Ker zraven kvadrokopterja ne dobimo daljinca in je za vodenje kvadrokopterja narejena aplikacija za Android in iPhone, je vodenje kvadrokopterja oteženo, saj človek nima občutka in ima majhno natančnost. Zaradi tega lahko pride do poškodb kvadrokopterja. Ker haptični vmesnik omogoča intuitivno upravljanje robotskega sistema in ima možnost povratne sile, je idealna rešitev za vodenje kvadrokopterja.

V diplomskem delu bomo spoznali delovanje kvadrokopterja AR.Drone 1.0, njegovo strojno, krmilno in programsko opremo kvadrokopterja ter potek komuniciranja med njim in računalnikom. Spoznali bomo haptično napravo Novint Falcon, strojno programsko opremo te naprave in AT-ukaze, ki jih pošiljamo prek UDP-ja (angl. User Datagram Protocol) na kvadrokopter. V nadaljevanju bomo opisali operacijski sistem Linux Ubuntu in njegovo namestitev, na čemer temelji diplomsko delo. Opisali bomo naš program za vodenje kvadrokopterja s haptičnim vmesnikom in kreiranje vtičnice za pošiljanje paketov prek UDP-ja. V nadaljevanju bomo opisali namestitev gonilnika haptične naprave.

Razložili bomo tudi bistvene dele kode napisanega programa in opisali upravljanje AR.Drona. Ogledali si bomo in prikazali rezultate testiranja in vodenja kvadrokopterja. Na sliki 1.1 je prikazan celotni sistem vodenja kvadrokopterja s haptično napravo Novint Falcon.




Slika 1.1: Vodenje kvadrokopterja s haptičnim vmesnikom Novint Falcon

Za namen diplomskega dela smo na računalnik namestili operacijski sistem Linux Ubuntu 13.10. Zanj smo se odločili, ker je programska oprema kvadrokopterja (AR.Drone SDK) narejena za Linux. Ker Novint nima gonilnika za Linux, smo za haptično napravo uporabili gonilnik Libnifalcon.

Koordinatni sistem haptičnega vmesnika smo izbrali za globalni koordinatni sistem. Koordinatni sistem kvadrokopterja pa je lokalni koordinatni sistem. Kvadrokopter vodimo v notranjih koordinatah. Translacije kvadrokopterja v vse tri smeri (x, y, z) dobimo z enačbami (1.1).

$$
\begin{aligned}
{}^{R}v_{Q}^{x} &= \mathrm{Sx} \cdot (\mathrm{p}_{H}^{z} - \mathrm{o}_{H}^{z}) \\
{}^{R}v_{Q}^{y} &= \mathrm{Sy} \cdot (\mathrm{p}_{H}^{y} - \mathrm{o}_{H}^{y}) \\
{}^{R}v_{Q}^{z} &= \mathrm{Sz} \cdot (\mathrm{p}_{H}^{x} - \mathrm{o}_{H}^{x})
\end{aligned}
\tag{1.1}
$$

Pri tem so:

${}^{R}v_{Q}^{x}$ – hitrost v smeri osi x v lokalnem (referenčnem) koordinatnem sistemu kvadrokopterja

${}^{R}v_{Q}^{y}$ – hitrost v smeri osi y v lokalnem (referenčnem) koordinatnem sistemu kvadrokopterja

${}^{R}v_{Q}^{z}$ – hitrost v smeri osi z v lokalnem (referenčnem) koordinatnem sistemu kvadrokopterja

Sx, Sy, Sz – skalirni fakorji

$\mathrm{p}_{H}^{x}$ – položaj prijemala haptične naprave v smeri osi x

$\mathrm{p}_{H}^{y}$ – položaj prijemala haptične naprave v smeri osi y

$\mathrm{p}_{H}^{z}$ – položaj prijemala haptične naprave v smeri osi z

$\mathrm{o}_{H}^{x}, \mathrm{o}_{H}^{y}, \mathrm{o}_{H}^{z}$ – izhodišče koordinatnega sistema haptične naprave (0, 0, 0)

Kvadrokopter lahko rotiramo okoli osi z lokalnega koordinatnega sistema kvadrokopterja s tipkami, ki so na sliki 5.2, označene s števili 3 in 4. Če pritisnemo tipko 3, bomo kvadrokopter rotirali okoli osi z lokalnega koordinatnega sistema kvadrokopterja v levo stran ($\omega_{z}^{-}$), če pa pritisnemo tipko 4, bomo kvadrokopter rotirali okoli osi z lokalnega koordinatnega sistema kvadrokopterja v desno stran ($\omega_{z}^{+}$).

# 2 KVADROKOPTER PARROT AR.DRONE 1.0

V tem poglavju bomo na splošno opisali delovanje kvadrokopterja, podrobneje pa strojno opremo krmilja, senzorje gibanja in mobilno platformo kvadrokopterja. Opisali bomo vsebino kompleta z vsemi sestavnimi deli in njihove specifikacije. V nadaljevanju bomo predstavili programsko opremo kvadrokopterja, s katero se povežemo na kvadrokopter in z njim komuniciramo. Opisali bomo tudi vrsto povezave in način komunikacije s kvadrokopterjem. Na koncu poglavja bomo podrobno opisali uporabljene AT-ukaze.

Pri diplomskem delu smo uporabili AR.Drone 1.0, ki je prikazan na slikah 2.1 in 2.2. Na sliki 2.1 je kvadrokopter z zaščito, ki je namenjena za vodenje v zaprtem okolju, da pri trkih ne pride do velikih poškodb. Na sliki 2.2 je kvadrokopter z okvirjem, ki je namenjen za vodenje v naravi in s katerim lahko dosežemo tudi večje hitrosti in agresivnejše vodenje. Zaščita je narejena iz ekspandiranega polipropilena, ki je zelo lahek in je z njim lahko narediti kompleksne oblike. Namenjen je letenju, nadzorujemo pa ga tako, da se nanj povežemo z mobilno napravo prek brezžične dostopne točke, ki se nahaja na samem kvadrokopterju. Na voljo so aplikacije za vodenje naprave za operacijska sistema iOS in Android.

Slika 2.1: Kvadrokopter AR.Drone 1.0 z zaščito

Slika 2.2: Kvadrokopter AR.Drone 1.0 brez zaščite

## 2.1 Opis in delovanje

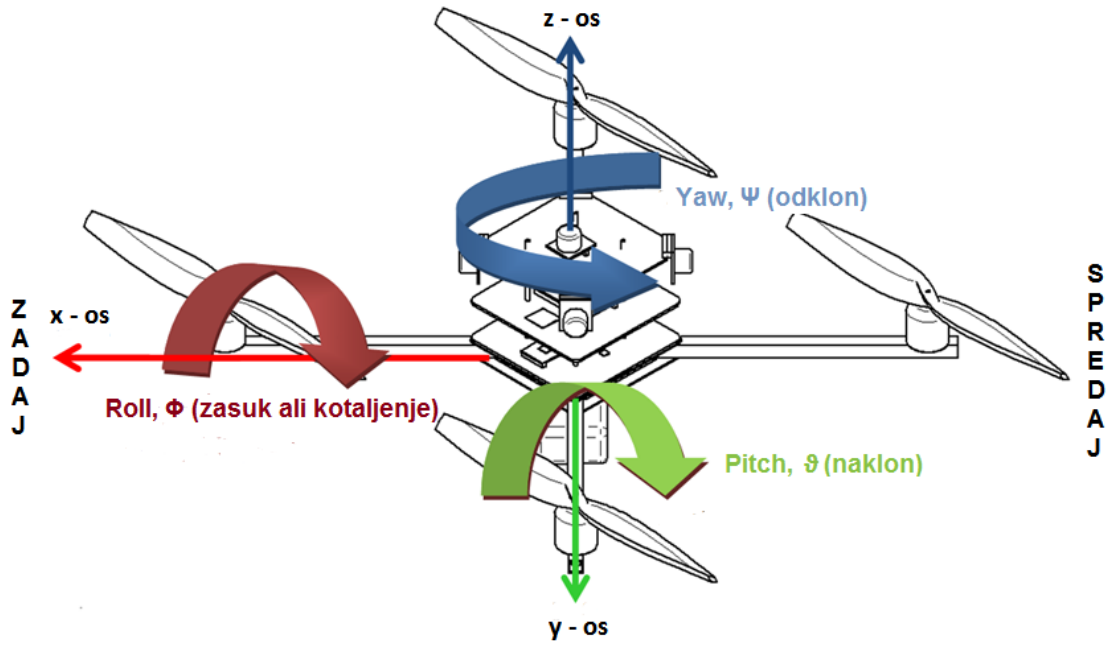
Struktura kvadrokopterja je narejena s štirimi palicami, ki so narejene iz karbonskih vlaken in razporejene tako, da je med dvema palicama kot $90^{o}$. Vse štiri palice so na sredini spojene in tvorijo centralni križ. Centralni križ je podlaga za strojno opremo in motorje, ki se nahajajo na kvadrokopterju. Motorji se nahajajo na koncu vsake palice in na njih so pritrjeni propelerji. Na sredini centralnega križa je strojna oprema s senzorji in baterijo. Na sliki 2.3 je prikazan centralni križ.

Slika 2.3: Centralni križ kvadrokopterja Ar.Drone 1.0 [**1**]

Par motorjev, ki je na enaki palici, se vrti v smeri urinega kazalca, drugi par pa v nasprotni smeri urinega kazalca, kot je prikazano na sliki 2.4.

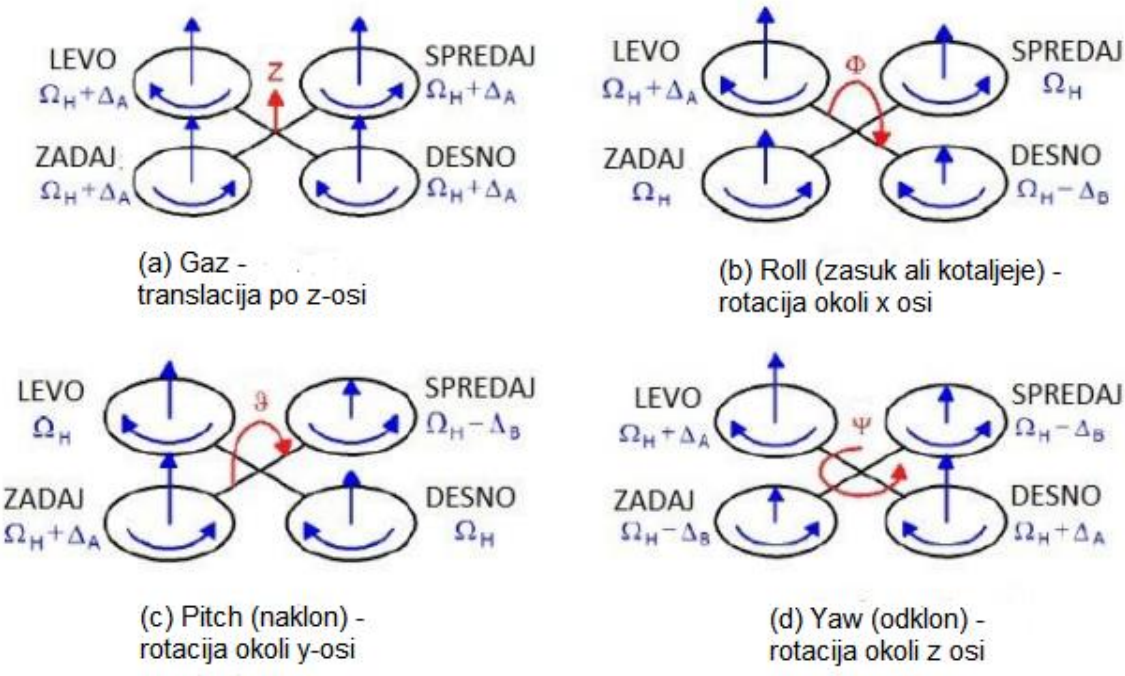

Slika 2.4: Smeri vrtenja motorjev kvadrokopterja [**2**]

Manevre izvedemo s spreminjanjem kotov nagiba kvadrokopterja, ki jih prikazuje slika 2.5.




Slika 2.5: Koti kvadrokopterja: roll (zasuk), pitch (naklon), yaw (odklon) [**3**]

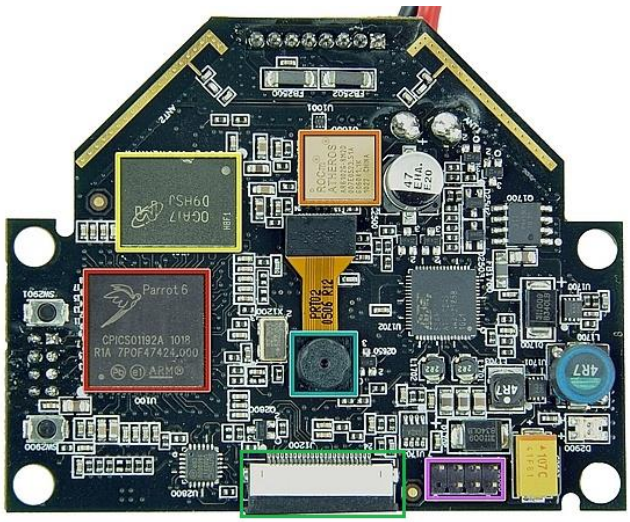
Z regulacijo kotne hitrosti motorjev lahko izvajamo različne manevre s kvadrokopterjem. Osnovne manevre bomo prikazali s pomočjo slike 2.6. Če vsakemu motorju enako povečamo kotno hitrost, bo prišlo do translacije po osi z. Če enemu motorju, levemu ali desnemu, povečamo kotno hitrost, drugemu pa jo zmanjšamo, pri čemer se motorja spredaj in zadaj vrtita z enako kotno hitrostjo, bo prišlo do rotacije okoli osi x. Če enemu motorju, spredaj ali zadaj, povečamo kotno hitrost, drugemu pa jo zmanjšamo, pri čemer se motorja levo in desno vrtita z enako kotno hitrostjo, bo prišlo do rotacije okoli osi y. Če enemu paru motorjev (spredaj in zadaj ali levo in desno) zmanjšamo in drugemu povečamo kotno hitrost, bo prišlo do rotacije okoli osi z.




Slika 2.6: Osnovni manevri kvadrokopterja [**4**]

## 2.2 Strojna oprema krmilja

Med strojno opremo krmilja kvadrokopterja spadata: matična plošča, navigacijska plošča, ki sta med sabo povezani prek serijskega vodila, in motorji. Na matični plošči sta mikrokontroler iz družine ARM9 s frekvenco delovanja 468 MHz, ki ga vidimo na sliki 2.7, označenega z rdečo barvo, in 128 MB pomnilnika RAM (Micron OGA17 D9HSJ LPDDR), označenega z rumeno barvo, ki skrbita za procesiranje informacij, ki jih procesor prejema in pošilja po brezžičnem omrežju WiFi. Na matični plošči je tudi WiFi-modul, ROCm Atheros AR6102G-BM2D b/g, ki je na sliki 2.7 označen z oranžno barvo. Na matični plošči je nameščena programska oprema (angl. firmware) na osnovi operacijskega sistema Linux. Z vijolično barvo je označen USB-konektor, ki omogoča povezavo z računalnikom in služi za nadgradnjo firmwara. Na matični plošči sta tudi dve kameri. Spodnja kamera zajema slike s hitrostjo 60 slik na sekundo, vidni kot pa znaša $64^{o}$. Prikazana je na sliki 2.7 in je označena z modro barvo. Sprednja kamera zajema slike v ločljivost 640 x 480 pik. Njen vidni kot je $93^{o}$, prikazana pa je na sliki 2.8. Najvišja hitrost, ki jo kvadrokopter lahko doseže, je 5 metrov na sekundo oziroma 18 km/h in jo meri z vertikalno (spodnjo) kamero. Z zeleno barvo je označen konektor za sprednjo kamero.




Slika 2.7: Matična plošča kvadrokopterja AR.Drone 1.0 [**5**]

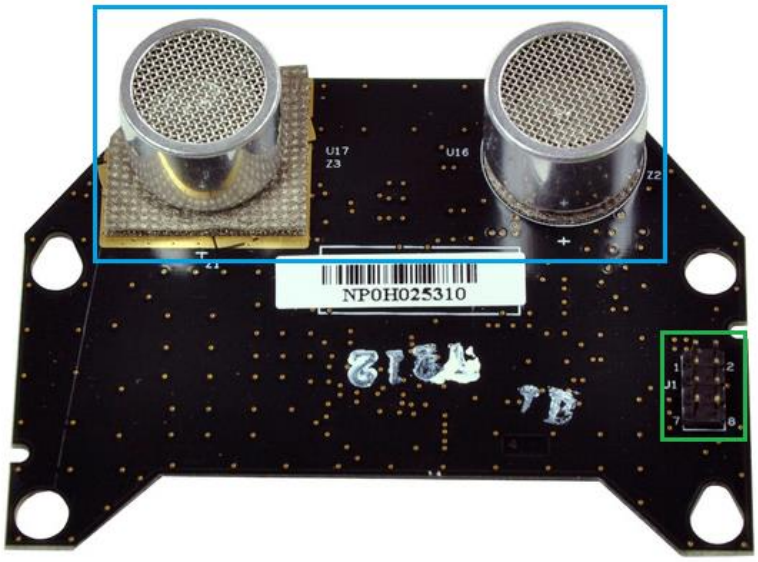

Slika 2.8: Sprednja kamera kvadrokopterja AR.Drone 1.0 [**6**]

Električno energijo zagotavlja 1000 mAh litij-ionska baterija z napetostjo 11.1 V, kar zadostuje za 12–15 minut letenja. Baterija se polni približno 90 minut. Polnilec za polnjenje omogoča, da se vsaka izmed treh celic polni enakomerno, s čimer se obdrži zmogljivost in podaljša življenjska doba baterije.

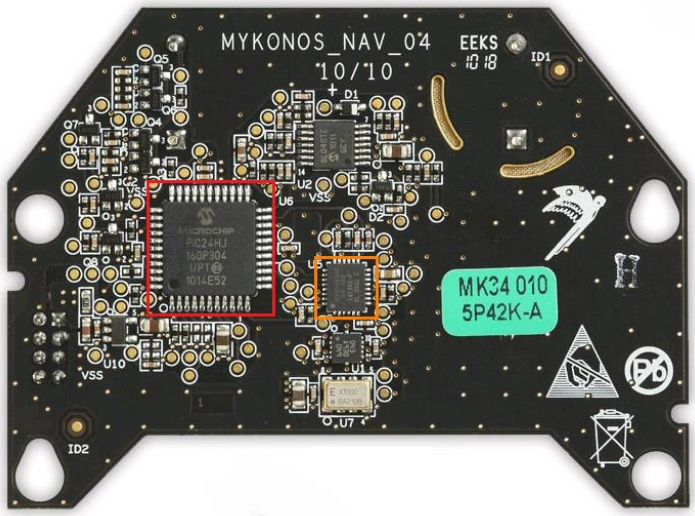
Baterija vsebuje zaščitni modul, ki ščiti baterijo pred prevelikim izpraznjenjem, prenapolnjenjem ali kratkim stikom. Baterija je prikazana na sliki 2.9.


Slika 2.9: Baterija kvadrokopterja AR.Drone 1.0 s 1000 mAh [**7**]

Kvadrokopter potrebuje za optimalno letenje informacije o trenutni orientaciji v prostoru. Te informacije pridobiva navigacijska plošča, na kateri so žiroskop (okoli dveh osi – x, y), pospeškometer (okoli vseh treh osi), kompas in ultrazvočni senzor višine z dometom 6 metrov, ki je prikazan na sliki 2.10 in označen z modro barvo. Na sliki 2.10 je z zeleno barvo označen konektor (serijska komunikacija) za povezovanje matične plošče z navigacijsko ploščo. Na sliki 2.11 je prikazana zgornja stran navigacijske plošče, kjer je z rdečo barvo označen 16-bitni mikrokrmilnik PIC24HJ16GP303 podjetja Microchip. Z oranžno barvo je označen dvoosni (x, y) žiroskop InvenSense IDG 500.

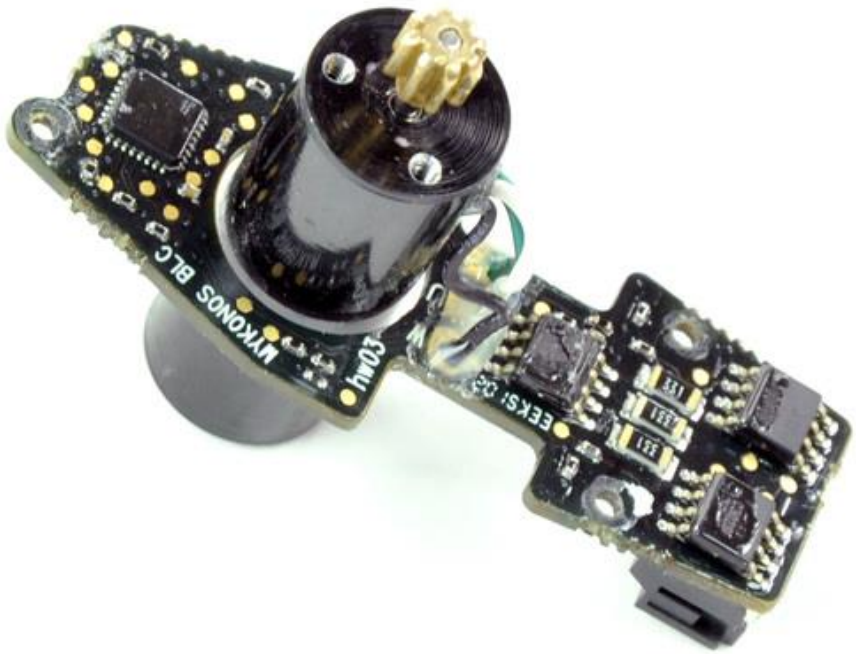



Slika 2.10: Spodnja stran navigacijske plošče kvadrokopterja AR.Drone 1.0 [**8**]




Slika 2.11: Zgornja stran navigacijske plošče kvadrokopterja AR.Drone 1.0 [**9**]

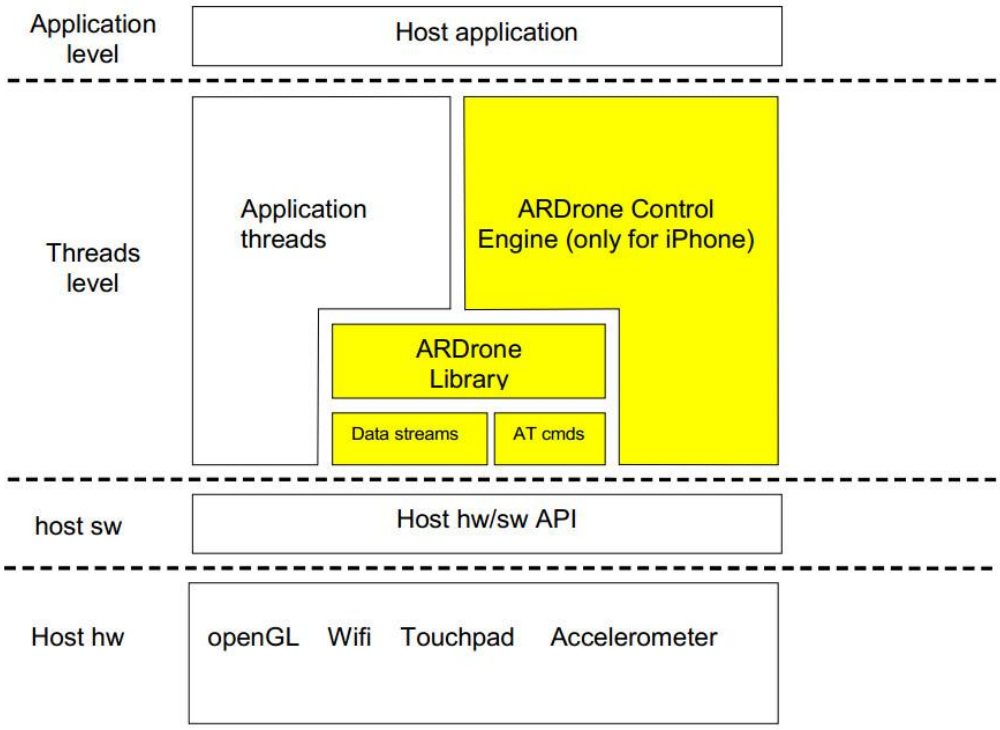
Letenje omogočajo štirje propelerji, ki so pritrjeni na brezkrtačne, trifazne motorje z močjo 15 W. Motorji se vrtijo z 28,000 vrtljajev na minuto med lebdenjem. Pri maksimalnem pospešku dosežejo 41,400 vrtljajev na minuto. Prestavno razmerje med motorji in propelerji je 8.625 : 1. Na sliki 2.12 vidimo, da so motorji priključeni na regulatorje. Hitrost motorjev regulirata 8-bitni mikrokrmilnik in 10-bitni analogno-digitalni pretvornik.



Slika 2.12: Motor z regulatorjem, mikrokrmilnikom in AD-pretvornikom kvadrokopterja AR.Drone 1.0 [**10**]

## 2.3 Programska oprema

Podjetje Parrot skrbi tudi za razvoj programske opreme za kvadrokopter AR.Drone in podporo razvijalcem na tem področju. S spletne strani proizvajalca je mogoče prenesti uradno verzijo programskega paketa AR.Drone SDK (angl. Software Development Kit), ki nam omogoča, da se na kvadrokopter povežemo tudi s prenosnim računalnikom, in nam nudi ogrodje za ustvarjanje lastnih aplikacij. Shemo zgradbe programske opreme vidimo na sliki 2.13. Prva plast predstavlja uporabnikovo aplikacijo, druga pa programe, ki so del programskega paketa SDK. Tretja plast ponazarja programsko opremo, ki je nameščena na kvadrokopterju. Spodnja plast predstavlja strojno opremo na AR.Dronu in tisto, s katero upravljamo kvadrokopter.




Slika 2.13: Shema programskega paketa kvadrokopterja AR.Drone 1.0 [**4**]

AR.Drone 2.0 knjižnica (ARDrone Library) je trenutno predvidena kot odprtokodna knjižnica, ki ponuja visokonivojski dostop do kvadrokopterja. Vsebina knjižnice:

• **SOFT**: specifična koda kvadrokopterja vsebuje:
  – COMMON: zaglavlje (.h) datoteke opisujejo komunikacijske strukture, ki jih uporablja kvadrokopter;
  – Lib/ardrone_tool: paket orodij, ki omogočajo enostavno upravljanje kvadrokopterja;
  – Lib/utils: vsebuje nabor pripomočkov za pisanje aplikacij;

• **VLIB**: vsebuje Ar.Drone knjižnico za obdelavo videa in funkcije za sprejem in dekodiranje videa;

• **FFMPEG**: vsebuje celoten posnetek FFMPEG-knjižnice, z narejenimi skripti za AR.Drone 2.0 aplikacijo;

• **ITTIAM**: vsebuje video dekodirane knjižnice za iOS in Android aplikacije;

• **VPSDK**: vsebuje nabor knjižnic za splošne namene in vključuje:
  – VPSTAGES: vsebuje videoobdelavo kosov, s katerimi lahko naredimo posnetek;
  – VPOS: vsebuje več arhitekturnih plasti (Linux, Windows, Parrot) za dodelitev pomnilnika, upravljanje itd.;
  – VPCOM: vsebuje več arhitekturnih plasti za komunikacijske funkcije (WiFi, Bluetooth);
  – VPAPI: vsebuje pripomočke za upravljanje z videom.

## 2.4 Komunikacija

Na kvadrokopter se lahko povežemo in ga upravljamo z vsako napravo, ki omogoča povezovanje v brezžična WiFi-omrežja in je združljiva s TCP/IP-komunikacijskim skladom. Pri povezovanju mobilne naprave na kvadrokopter AR.Drone se zgodijo naslednji koraki:

**1.** kvadrokopter ustvari WiFi-omrežje;

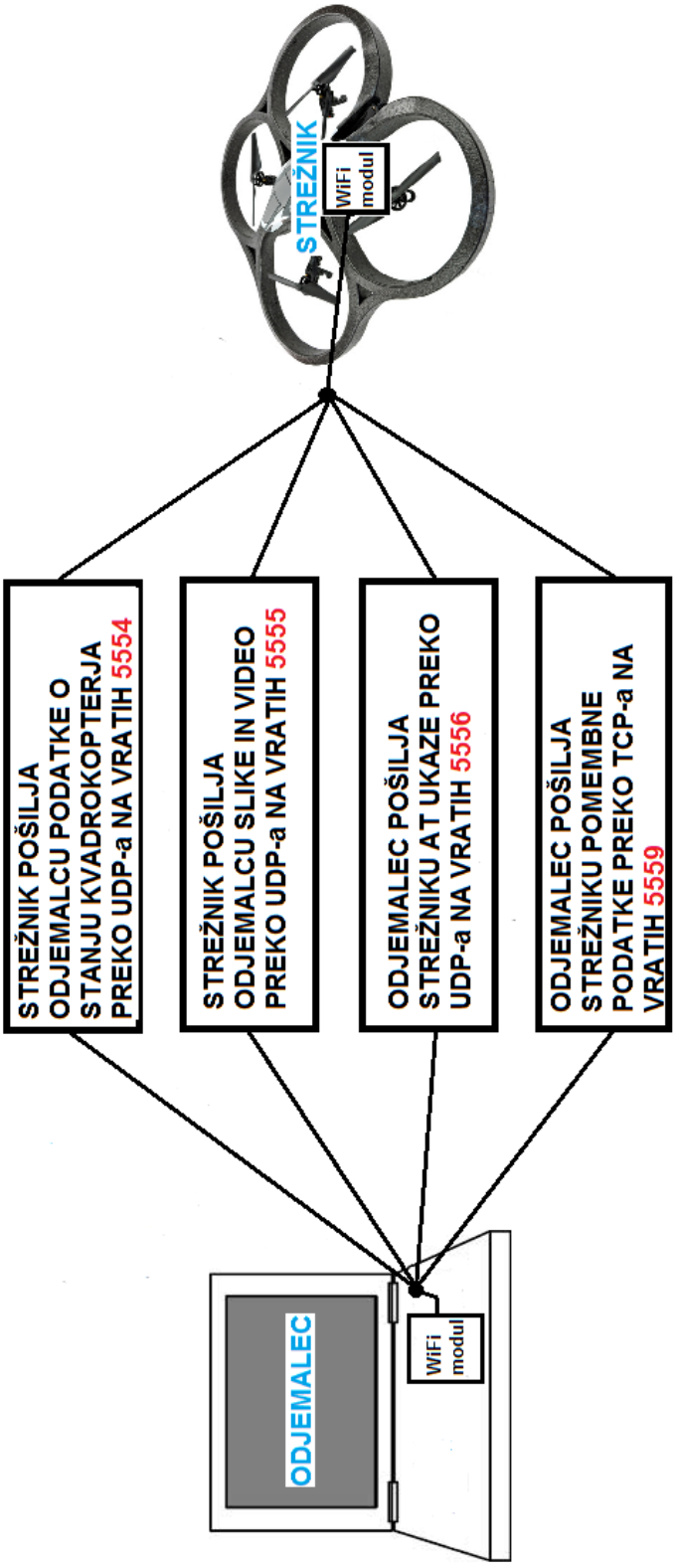
**2.** mobilna naprava se poveže v ustvarjeno omrežje. IP-naslov dobi od DHCP-strežnika, ki teče na kvadrokopterju;

**3.** mobilna naprava v vlogi klienta lahko začne pošiljati ukaze kvadrokopterju, ki je v vlogi strežnika, prek vnaprej določenih TCP- in UDP-vtičnic.

Komunikacija klienta s kvadrokopterjem poteka na več načinov. Krmiljenje kvadrokopterja je realizirano tako, da klient pošilja strežniku AT-ukaze prek protokola UDP na vratih 5556.

Informacije o kvadrokopterju strežnik pošilja klientu prek protokola UDP na vratih 5554. Te informacije obsegajo stanje, v katerem je kvadrokopter, višino, na kateri je, položaj, orientacijo, hitrost, s katero se premika, vrtljaje motorjev in stanje baterije. Kvadrokopter pošilja klientu tudi sliko s kamer prek protokola UDP na vratih 5555. Slika je lahko zakodirana na dva načina, UVLC in P264. Zadnji način komunikacije poteka prek protokola TCP na vratih 5559 in je namenjen pošiljanju pomembnejših podatkov na strežnik. Prek te povezave lahko spreminjamo konfiguracijske nastavitve kvadrokopterja.

Na sliki 2.14 je prikazana komunikacija med računalnikom in kvadrokopterjem, poteka pa prek WiFi-povezave.




Slika 2.14: Komunikacija med računalnikom in kvadrokopterjem AR.Drone 1.0

## 2.5 Krmilni AT-ukazi

AT-ukazi so besedni nizi, ki jih pošiljamo na kvadrokopter prek UDP-protokola, od klienta (uporabnika) na strežnik (kvadrokopter), prek porta (vrat) 5556. S pošiljanjem določenih AT-ukazov kvadrokopterju povemo, kaj naj naredi. V eni vrstici je lahko poslanih tudi več ukazov, maksimalna dolžina informacij v paketu, ko pošiljamo prek UDP-protokola, pa je 1024 znakov, sicer je vrstica zavrnjena s strani kvadrokopterja. Ti besedni nizi so 8-bitni ASCII-znaki, ki so shranjeni v ArDroneLib in ArDroneTool knjižnicah.

Glede na teste je najboljše, če AT-ukaze pošiljamo vsakih 30 ms. Če ne pošljemo dveh zaporednih ukazov v dveh sekundah, potem se prekine komunikacija med računalnikom in kvadrokopterjem.

Glava AT-ukaza je sestavljena iz treh znakov **AT***, ki so kodirani kot $42_{(16)}$, $54_{(16)}$ in $2A_{(16)}$. Za glavo ukaza sledijo ime ukaza, enačaj in argumenti, katerih število je odvisno od vrste ukaza. Argumenti so med sabo ločeni z vejico. Na koncu ukaza se vpiše znak za naslednjo vrstico (<LF>).

### Ukaz AT*REF

Sintaksa: **AT*REF=%d,%d<LF>**

*Argument 1*: sekvenčna številka
*Argument 2*: 32-bitno celo število

Ta ukaz smo uporabili za vzletanje, pristajanje in zasilni izklop. V prvem argumentu je sekvenčno število zaporedno število in se povečuje z vsakim poslanim ukazom. Vsaki

naslednji poslani ukaz mora imeti za eno večjo sekvenčno število od predhodnega sekvenčnega števila oziroma ukaza. Namenjeno je za zaščito, da se ukazi pošiljajo zaporedoma od najmanjšega sekvenčnega števila do največjega, da ne pride do napak in da se nekateri ukazi ne izvedejo večkrat. Pri tem ukazu smo uporabljali bita 8 in 9.

Spreminjali smo bit 9 na 0 in 1. Pri postavljanju bita 9 na 1 se kvadrokopter dvigne in se s pomočjo ultrazvočnega senzorja ustali na višini 1 m od tal, potem se avtomatsko izvede preklop v način lebdenja. Pri preklopu bita 9 na 0 se izvede avtomatsko pristajanje s pomočjo ultrazvočnega senzorja. Pri postavljanju bita 8 na 1 smo vključili prisilni izklop, kar pomeni izključitev vseh motorjev. Biti 18, 20, 22, 24 in 28 morajo imeti vrednost 1, ostali pa 0.

*Primer ukaza:*

AT*REF=23,512<LF>

Iz ukaza vidimo, da ima argument 1 sekvenčno številko 23 in argument 2 številko 512, kar pomeni, da je bit 9 postavljen na 1 in da se bo kvadrokopter dvignil na višino 1 m od tal.

## Ukaz AT*PCMD

Sintaksa: **AT*PCMD=%d,%d,%d,%d,%d,%d<LF>**

*Argument 1:* sekvenčna številka
*Argument 2:* zastavica za lebdenje
*Argument 3:* podaja odstotek maksimalnega naklona okoli osi x, zasuk ali kotaljenje (roll)
*Argument 4:* podaja odstotek maksimalnega naklona okoli osi y, naklon (pitch)
*Argument 5:* podaja odstotek maksimalne hitrosti translacijskega gibanja v smeri osi z (gaz)

*Argument 6:* podaja odstotek maksimalne kotne hitrosti rotacije okoli osi z, odklon (yaw)

Ta ukaz smo uporabili za izvajanje premikov kvadrokopterja. Argument 2 je celo število in je zastavica za lebdenje. Ko je zastavica postavljena na 0, je vključeno lebdenje, ko pa je postavljena na 1, se kvadrokopter odziva na vrednosti v argumentih 3, 4, 5 in 6.

Argumenti 3, 4, 5 in 6 so števila s plavajočo vejico (IEEE754) v območju med –1 in 1.

Roll (zasuk ali kotaljenje) vrednosti argumentov določajo odstotek maksimalnega naklona, ki je nastavljen v parametrih kvadrokopterja za levi in desni nagib. Smer nagiba je določena s predznakom števila. Negativna vrednost bo nagib kvadrokopterja v levo smer, pozitivna vrednost pa nagib kvadrokopterja v desno smer.

Pitch (naklon) vrednosti argumentov določajo odstotek maksimalnega naklona, ki je nastavljen v parametrih kvadrokopterja za nagib naprej in nazaj. Smer nagiba je določena s predznakom števila. Negativna vrednost bo nagib kvadrokopterja naprej, pozitivna vrednost pa nagib kvadrokopterja nazaj.

Gaz vrednosti argumentov določajo odstotek največje vertikalne hitrosti, ki je nastavljena v parametrih kvadrokopterja za vertikalni premik kvadrokopterja navzgor ali navzdol. Smer nagiba je določena s predznakom števila. Če je negativna vrednost, se bo kvadrokopter spuščal, če je pa pozitivna vrednost, se bo dvigoval.

Yaw (odklon) vrednosti argumentov, določajo odstotek največje kotne hitrosti, ki je nastavljena v parametrih kvadrokopterja za vrtenje kvadrokopterja okoli osi z v levo in desno smer. Smer vrtenja je določena s predznakom števila. Če je negativna vrednost, se bo kvadrokopter vrtel v levo stran okoli osi z, če je pa pozitivna vrednost, pa se bo kvadrokopter vrtel v desno stran okoli osi z.

*Vrednosti, ki se največ uporabljajo:*

–1082130432: najmanjša pretvorjena vrednost -1, kvadrokopter bo letel v levo smer z maksimalno močjo;
–1086324736: pretvorjena vrednost -0,75, kvadrokopter bo letel v levo smer s 3/4 moči;
–1090519040: pretvorjena vrednost -0,5, kvadrokopter bo letel v levo smer s polovično močjo;
–1098907648: pretvorjena vrednost -0,25, kvadrokopter bo letel v levo smer z 1/4 moči;
– 0: kvadrokopter bo letel vodoravno;
– 1048576000: pretvorjena vrednost 0,25, kvadrokopter bo letel v desno smer z 1/4 moči;
– 1056964608: pretvorjena vrednost 0,5, kvadrokopter bo letel v desno smer s polovično močjo;
– 1061158912: pretvorjena vrednost 0,75, kvadrokopter bo letel v desno smer s 3/4 moči;
– 1065353216: največja pretvorjena vrednost 1, kvadrokopter bo letel v desno smer z maksimalno močjo.

*Primer ukaza:*

AT*PCMD=5,1,0,0,1061158912,-1082130432<LF>

Iz ukaza vidimo, da ima argument 1 sekvenčno številko 5, argument 2 ima zastavico postavljeno na 1, kar pomeni, da se kvadrokopter odziva na vrednosti v argumentih 3, 4, 5 in 6. Argumenta 3 in 4 sta postavljena na 0, kar pomeni, da se kvadrokopter ne bo premikal v smeri naprej–nazaj in levo–desno. Argument 5 pomeni, da se bo kvadrokopter dvigoval s ¾ moči. Argument 6 pomeni, da se bo kvadrokopter vrtel okoli osi z v levo stran z maksimalno močjo.

## Ukaz AT*FTRIM

Sintaksa: **AT*FTRIM=%d,<LF>**

*Argument 1*: sekvenčna številka

Ta ukaz smo uporabili za vodoravno usmeritev, za postavljanje žiroskopa v vseh oseh na 0°. Ukaz je treba na kvadrokopter poslati, preden se dvigne, torej ko je kvadrokopter na tleh. Če tega ukaza ne pošljemo, preden se kvadrokopter dvigne, lahko pride do problema s stabilizacijo kvadrokopterja in do poškodbe kvadrokopterja. Ko kvadrokopter sprejme ta ukaz, samodejno prilagaja in nadzoruje kote roll (zasuk ali kotaljenje) in pitch (naklon).

*Primer ukaza:*

AT*FTRIM=17<LF>

Ta ukaz ima samo 1 argument, kar pomeni, da se bo ta ukaz izvedel sedemnajsti po vrsti.

## Ukaz AT*LED

Sintaksa: **AT*LED=%d,%d,%d,%d<LF>**

*Argument 1*: sekvenčna številka (celo število)
*Argument 2*: številka animacije (celo število med 0 in 20)
*Argument 3*: frekvenca animacije v Hz (število s plavajočo vejico)
*Argument 4*: končno trajanje v sekundah (celo število)

Kvadrokopter ima na vsakem motorju eno LED-diodo. Vse skupaj ima štiri diode. Ta ukaz smo uporabili za spreminjanje utripanja LED-diod po vnaprej določenem zaporedju.

Kot argument 2 si izberemo, katera animacija se nam bo prižgala, ali bodo vse diode zelene, ali bodo sprednje zelene, zadnje rdeče itd.

V argumentu 3 določimo, ali bo dioda utripala in s kako frekvenco.

V argumentu 4 določimo, koliko časa bo trajala animacija.

*Primer ukaza:*

AT*LED=21,3,1084227584,2

Iz ukaza vidimo, da ima argument 1 sekvenčno številko 23, argument 2 pa ima številko 3, kar pomeni, da bodo vse štiri LED-diode utripale oranžno. V argument 3 je vpisana številka 1084227584, kar pomeni, da bodo vse LED-diode utripale (prižgane bodo 0,2 s in izklopljene 0,2 s). V argument 4 je vpisano število 2, kar pomeni, da bo animacija trajala 2 sekundi.

# 3 HAPTIČNI VMESNIK NOVINT FALCON

Haptični vmesnik Novint Falcon je produkt podjetja Novint Technologies. Podjetje je ustanovljeno v Delawaru in ima sedež v Albuquerque (Novi Mexico, ZDA). Podjetje Novint Technologies načrtuje in proizvaja 3D-naprave in njihovo programsko opremo. Razvili so Novint Falcona, 3D-haptično napravo, ki omogoča uporabnikom, da uporabljajo občutek dotika na področju računalništva (predvsem v virtualnih aplikacijah). Falcon ima dve glavni področji uporabe: videoigre in profesionalna uporaba. V videoigrah lahko s Falconom čutimo predmete in dogodke v igri, kar daje igralcu poglobljeno doživetje in dodatne izkušnje. V profesionalni uporabi se Falcon uporablja za dodajanje občutka dotika v različnih profesionalnih aplikacijah in projektih. Prikazan je na sliki 3.1.


Slika 3.1: Haptični vmesnik Novint Falcon

Zgrajen je iz treh rok, ki jih lahko premikamo v notranjost ali zunanjost stožčaste oblike. Prijemalo se nahaja na vrhu treh rok in ima obliko krogle. Prijemalo lahko tudi odstranimo s Falcona. Na prijemalu so štiri tipke, kot je prikazano na sliki 3.2. Na sprednji strani stožca Falcona je Novint Falcon logo, ki lahko sveti v različnih barvah in tako označuje stanje, v katerem se nahaja haptična naprava.

Slika 3.2: Prijemalo haptične naprave Novint Falcon v obliki krogle [**11**]

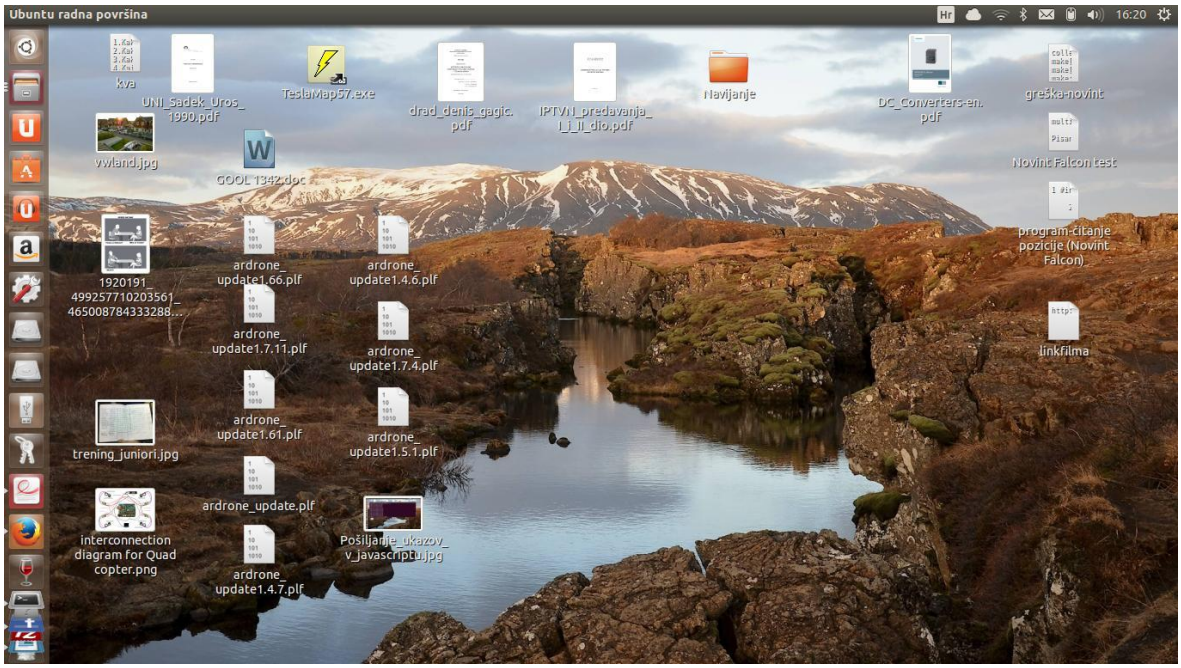
Telo Falcona vsebuje tri motorje z optičnimi senzorji. Prenos med motorji in ročicami je izveden z vrvicami (angl. "capstan drive"). Optični senzorji zaznajo gibanje ročice v ločljivosti delcev milimetrov. Motorji so posodobljeni 1000-krat na sekundo, kar daje realističen občutek dotika. Teža in dinamika objektov sta simulirani na način, da lahko čutimo vztrajnostni moment. Notranjost Falcona je prikazana na sliki 3.3.

Slika 3.3: Notranjost haptične naprave Novint Falcon [**12**]

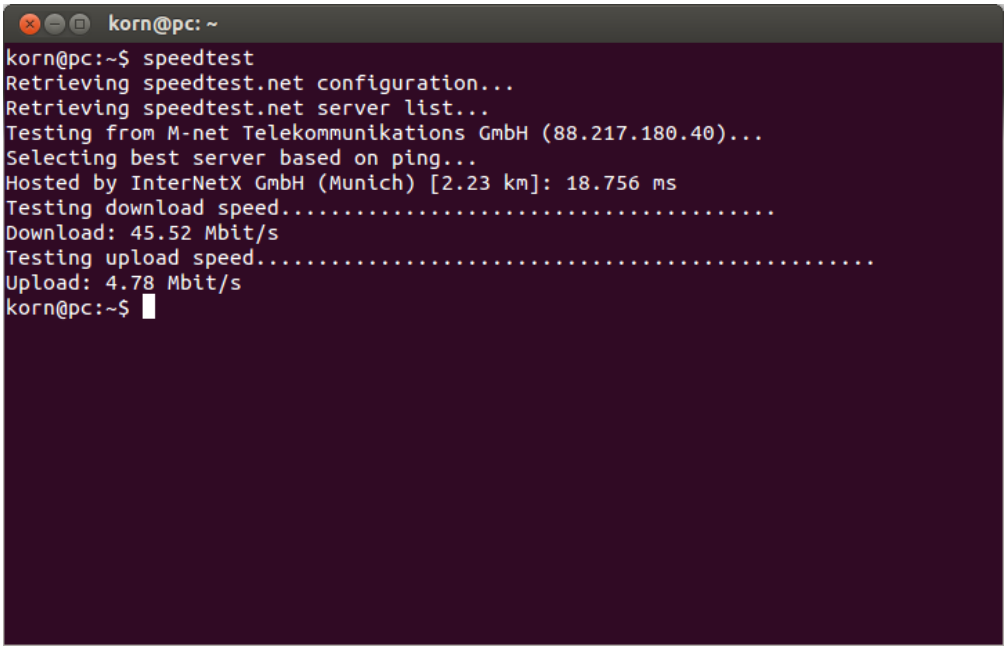
Ko uporabnik premakne prijemalo, krmilnik s programsko opremo, ki se nahaja znotraj Falcona, spremlja premike. Izmerjeni premiki sestavljajo neko trajektorijo gibanja. Trajektorija je posredovana do aplikacije na PC-ju. Na trajektorijo se aplikacija odzove bodisi s premikanjem po virtualnem prostoru bodisi s kreiranjem (izračunavanjem) neke virtualne sile, ki je nastala kot posledica trajektorije. Na podlagi virtualne sile iz aplikacije Falcon ustvari realno silo, ki jo uporabnik čuti.

# 4 OPERACIJSKI SISTEM LINUX UBUNTU

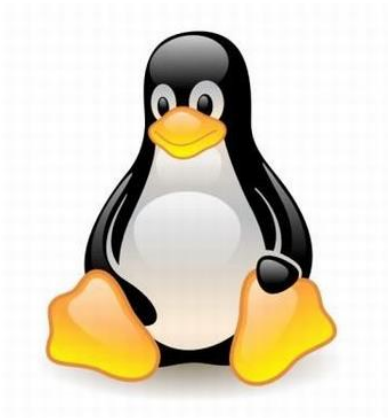
Pri delu uporabljamo operacijski sistem Linux Ubuntu 13.10, katerega namizje je prikazano na sliki 4.1. OS Linux je v primerjavi z OS Windows enostavnejši za uporabo, je brezplačen in odprtokoden (njegova izvorna koda je na voljo vsem uporabnikom). Dostopen je v več jezikih. Razlika med OS Linux in OS Windows je v tem, da ima OS Linux boljšo programsko podporo. Vsi ukazi se vpisujejo v Terminal, s katerimi se izvaja inštaliranje gonilnikov, programov, zaganjanje programov, nadgradnja programov itd. Program Terminal lahko odpremo s kombinacijo tipk: Ctrl+Alt+T. Program pišemo v tekstni urejevalnik Gedit in potem v Terminalu z namenjenimi ukazi prevajamo kodo v različne programske jezike. Program tudi zaženemo v Terminalu. V nadaljevanju bomo opisali postopek inštaliranja OS Linux Ubuntu 13.10. Okno programa Terminal je prikazano na sliki 4.2. Zaščitni znak operacijskega sistema Linux je pingvin TUX, ki je prikazan na sliki 4.3.




Slika 4.1: Namizje OS Linux Ubuntu 13.10




Slika 4.2: Okno programa Terminal



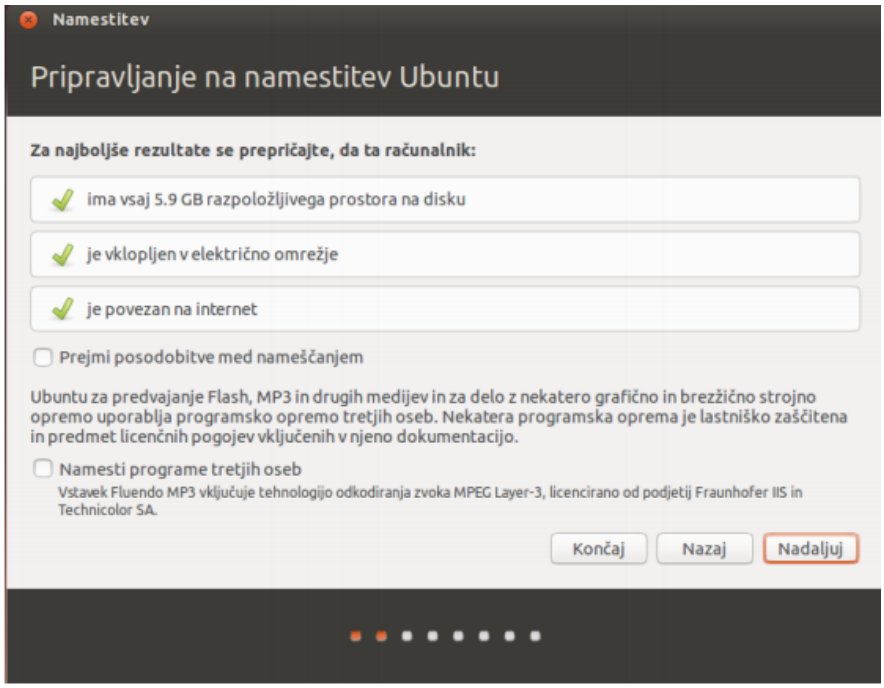
Slika 4.3: Zaščitni znak operacijskega sistema Linux [**13**]

## 4.1 Namestitev operacijskega sistema Linux Ubuntu na računalnik

Linuxa smo prevzeli s strani *www.ubuntu.com/download* [**14**]. Izbrali smo 64-bitno različico in kliknili "**Začni prejem**". Ko smo Linuxa prenesli na računalnik, je datoteka imela naziv "**ubuntu-13.10-desktop-amd64.iso**". Ker smo Ubuntuja želeli inštalirati z USB-ključka, smo kliknili na "**Zapišite svoj DVD ali ustvarite pogon USB**" in izbrali možnost "**Ključ USB**". Morali smo izbrati operacijski sistem, ki ga uporabljamo za ustvarjanje pogona USB, in klikniti "**Pokaži mi, kako**".

Za namestitev Ubuntuja smo potrebovali vsaj 5 GB razpoložljivega prostora na trdem disku, vendar je priporočenih 15 GB ali več. S tem smo zagotovili, da smo imeli pozneje dovolj prostora za nameščanje dodatnih programov kot tudi shranjevanje svojih dokumentov, glasbe in slik. Za začetek smo vstavili USB-ključek v računalnik in ponovno zagnali računalnik. Ob prvem zagonu se je pojavil zaslon, ki nas je vprašal, ali želimo Ubuntu najprej preizkusiti ali namestiti. Izbrali smo jezik, v katerem želimo videti namestilnik, in kliknili "**Namesti Ubuntu**". Začelo se je njegovo nameščanje.

Zaslon »Pripravljanje na namestitev Ubuntu« sporoča, ali imamo dovolj prostora na disku in ali smo povezani z električnim omrežjem (ker nameščamo Ubuntu na prenosnik, ki se napaja iz baterije). Pripravljanje na namestitev Ubuntuja je prikazano na sliki 4.4. Izbrali smo želene možnosti in kliknili "**Nadaljuj**".




Slika 4.4: Pripravljanje na namestitev Ubuntuja

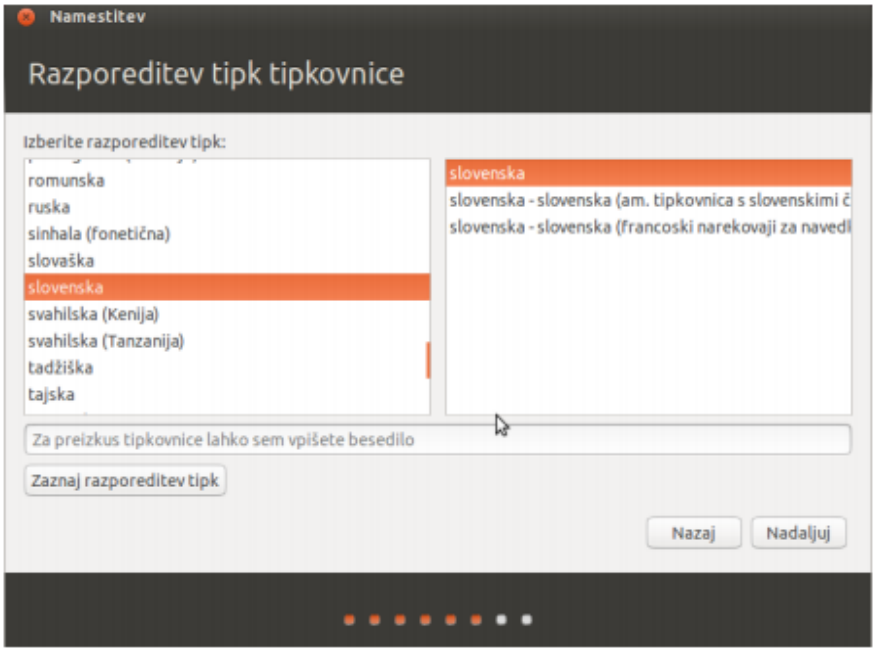
Ker nismo bili povezani s spletom, nam je namestilnik ponudil izbiro brezžičnega omrežja (če je na voljo). Izbrali smo želeno omrežje in kliknili "**Poveži se**" za nadaljevanje. Nastavitev brezžične povezave je prikazana na sliki 4.5.


Slika 4.5: Nastavitev brezžične povezave za namestitev Ubuntuja

Namestilnik Ubuntu je samodejno zaznal morebitni obstoječi operacijski sistem na računalniku in predstavil možnosti, primerne za naš sistem. Možnosti so: namesti poleg drugih operacijskih sistemov, nadgradite Ubuntu, izbriši disk in namesti Ubuntu itd. Ker smo želeli na našem računalniku imeti samo OS Linux, smo izbrali možnost "**Izbriši disk in namesti Ubuntu**". To je izbrisalo morebitne obstoječe operacijske sisteme, ki so bili nameščeni na tem disku, in namesto njih namestilo Ubuntu. Po izbrani vrsti namestitve smo kliknili "**Namesti zdaj**". Za zmanjšanje časa, zahtevanega za namestitev, je Ubuntu nadaljeval z opravilom namestitve v ozadju, mi pa smo medtem nastavljali pomembne uporabniške podrobnosti, kot so: uporabniško ime, geslo, nastavitve tipkovnice in privzeti časovni pas. Naslednji zaslon, ki je viden na sliki 4.6, prikazuje zemljevid sveta. Z miško smo kliknili na področje na zemljevidu, kjer se nahajamo. To Ubuntuju omogoča nastavitev naše sistemske ure in drugih zmožnosti, ki so odvisne od našega mesta. Ko smo končali z izbiro mesta, smo kliknili "**Naprej**".

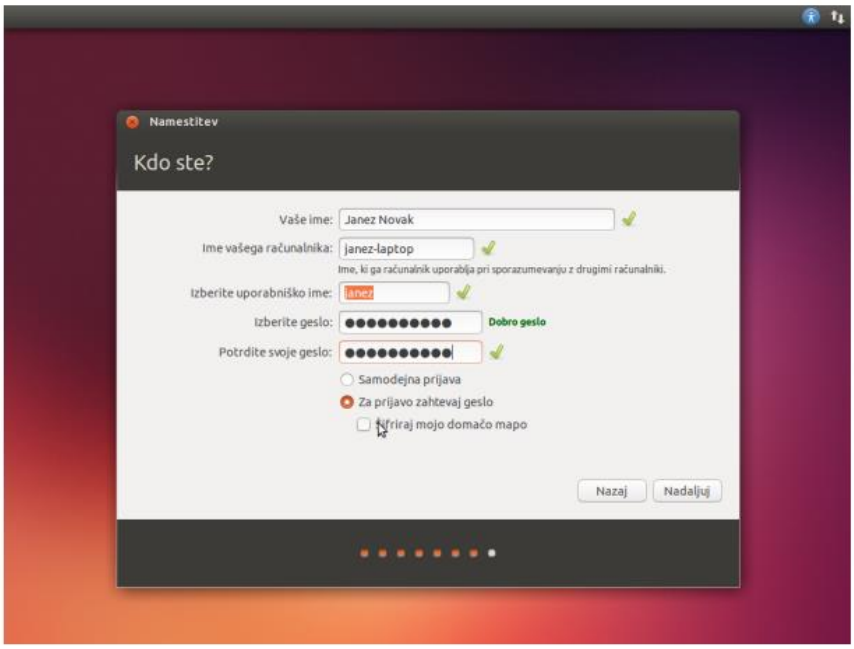

Slika 4.6: Namestitev časovnega pasa Ubuntuja

V nadaljevanju smo morali izbrati vrsto tipkovnice, ki jo uporabljamo. Ker nismo bili prepričani, katero tipkovnico izbrati, smo kliknili "**Zaznaj razporeditev tipk**". Ubuntu je določil pravo tipkovnico tako, da nas je pozival k pritisku zaporedja tipk. Nato smo kliknili "**Nadaljuj**".



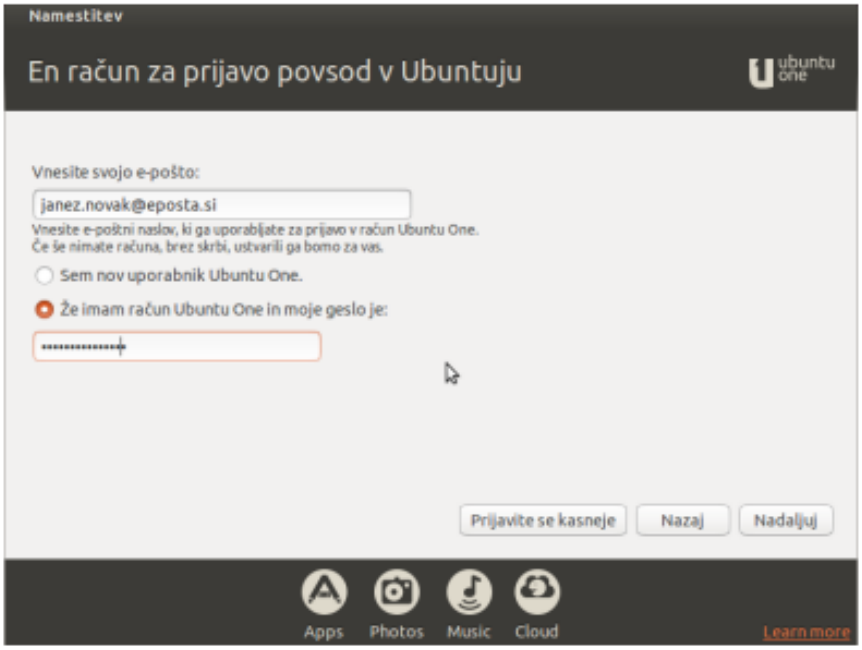

Slika 4.7: Namestitev razporeditve tipk tipkovnice Ubuntuja

Ubuntu potrebuje nekaj podatkov, da lahko nastavi glavni uporabniški račun na računalniku. Pri tem koraku smo morali Ubuntuju povedati: svoje ime, želeno ime računalnika, želeno uporabniško ime, želeno geslo in kako se želimo prijaviti v Ubuntu. Namestitev uporabniškega računa je prikazana na sliki 4.8.




Slika 4.8: Namestitev uporabniškega računa Ubuntu-ja

Med namestitvijo smo ustvarili novi račun Ubuntu One. Izpolnili smo potrebne podrobnosti in kliknili "**Nadaljuj**". Prijava v račun Ubuntu One je prikazana na sliki 4.9.

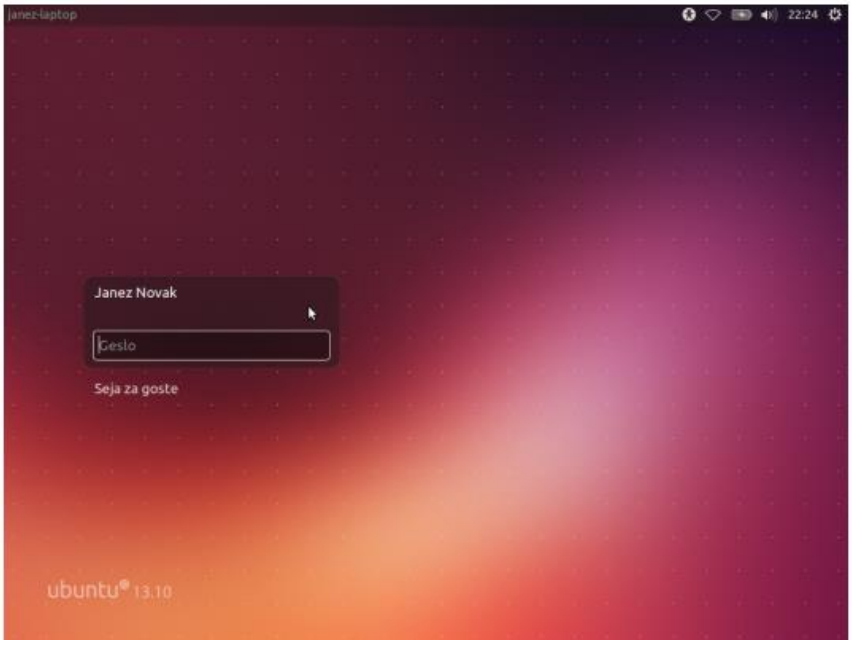



Slika 4.9: Prijava v račun Ubuntu One

Ubuntu je končal z nameščanjem na trdi disk. Medtem ko je namestitev potekala, nam je predstavitev ponudila uvod v nekatere privzete programe, ki so vključeni v Ubuntu. Predstavitev vsebuje tudi predstavitev možnosti podpore za Ubuntu, kar lahko vidimo na zaslonu, ki je prikazan na sliki 4.10.


Slika 4.10: Možnosti podpore za Ubuntu

Po približno dvajsetih minutah se je namestitev končala. Kliknili smo "**Ponovno zaženi**" za ponovni zagon računalnika in zagon Ubuntuja. Ko se je namestitev končala, se je računalnik znova zagnal in pozdravil nas je Ubuntujev prijavni zaslon, ki je prikazan na sliki 4.11. Na prijavnem zaslonu je bilo prikazano naše uporabniško ime in morali smo vnesti geslo za dostop. Ko smo končali, smo pritisnili tipko "**Enter**" in prišli v namizje Ubuntu.




Slika 4.11: Prijavni zaslon Ubuntuja

# 5 IMPLEMENTACIJA

V tem poglavju bomo opisali naš program za vodenje kvadrokopterja s haptičnim vmesnikom. Opisali bomo kreiranje vtičnice za pošiljanje paketov prek UDP-ja. V nadaljevanju bomo opisali namestitev gonilnika haptične naprave. Razložili bomo tudi bistvene dele kode napisanega programa in opisali upravljanje Ar.Drona. Prikazali bomo diagram poteka našega programa.

Program je sestavljen iz treh delov:

- UDP-vtičnice za komunikacijo med računalnikom in kvadrokopterjem;
- dela, ki skrbi za upravljanje haptične naprave Falcon;
- dela, ki skrbi za upravljanje AR.DRONE kvadrokopterja.

## 5.1 Kreiranje komunikacijske (UDP) vtičnice

Program smo napisali v urejevalniku besedila Gedit. Naredili smo novo datoteko z imenom "**DroneControl.cpp**" in začeli pisati v programskem jeziku C++.

Fizični nivo komunikacije deluje na brezžičnem omrežju WiFi. Uporabljen je TCP/IP-komunikacijski sklad z uporabljeno UDP-vtičnico. Programa za strežnik ni bilo treba narediti, naredili smo samo program za odjemalca, ker se kvadrokopter obnaša kot strežnik. V program smo vključili knjižnice, brez katerih ni možno pošiljanje paketov prek UDP-ja:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<sys/socket.h>
```

Potem smo definirali IP-naslov, velikost paketa in vrata za pošiljanje paketov:

```
#define SERVER "192.168.1.1"  //definiran IP naslov za posiljanje paketov preko UDP-a
#define BUFLEN 512  //definirana veličina paketov za slanje preko UDP-a, 512 Kbit-ov
#define PORT 5556   //definirana vrata za slanje paketov preko UDP
```

V nadaljevanju smo ustvarili in konfigurirali vtičnico za pošiljanje paketov:

```
//ustvarimo svtičnico za pošiljanje paketov preko UDP-a
    if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
        {
            perror("socket");
            exit(1);
        }

//konfiguriramo vtičnico za pošiljanje paketov preko UDP-a
    memset((char *)&si_other, 0, sizeof(si_other));
    si_other.sin_family = AF_INET;
    si_other.sin_port = htons(PORT);

    if (inet_aton(SERVER, &si_other.sin_addr) == 0)
    {
        fprintf(stderr, "inet_aton() failed\n");
        exit(1);
    }
```

Na koncu smo podatke, ki se nahajajo v "**message**", poslali prek UDP-ja na kvadrokopter:

```
// ukaz pošljemo preko UDP-a na kvadrokopter
    if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
        {
            perror("sendto()");
            exit(1);
        }
```

## 5.2 Upravljanje haptične naprave

Ker podjetje Novint nima gonilnikov za operacijski sistem Linux, smo uporabili gonilnik **Libnifalcon-master**, ki je narejen za operacijske sisteme Linux in iOS. Zagotavlja komunikacijo in model kinematike za krmilnik. Gonilnik je dostopen na spletni strani https://github.com/qdot/libnifalcon. Pred namestitvijo gonilnika za haptični vmesnik smo morali na računalnik namestiti dodatne programe: **Libusb 1.0, Boost in Cmake**.

Libusb je C knjižnica, ki omogoča aplikacijam enostaven dostop do USB-naprav na različnih operacijskih sistemih. Boost vsebuje dodatne C++ knjižnice, potrebne za namestitev gonilnika haptične naprave. Cmake je družina orodij, namenjena za izgradnjo in testiranje programov. Ustvarja Makefile datoteke in delovne prostore, ki jih je mogoče uporabiti v okolju prevajalnika po izbiri. Programe smo na računalnik namestili s pomočjo programa Terminal.

1) Program Libusb 1.0 smo namestili z vpisom ukaza: "**sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev**".

2) Program Boost smo namestili z vpisom ukaza: "**sudo apt-get install libboost1.42-all-dev**".

3) Program Cmake smo namestili z vpisom ukaza: "**sudo apt-get install cmake**".

4) Da smo lahko začeli s prevajanjem gonilnika, smo se morali prestaviti v tisto mapo, v katero smo shranili gonilnik. To smo naredili s pomočjo ukaza "**cd**".

5) V mapi libnifalcon-master smo naredili novo mapo pod imenom "**build**", tako da smo vpisali ukaz: "**sudo mkdir build**".

6) Odprli smo novo mapo z imenom "**build**", tako da smo vpisali ukaz: "**cd build**".

7) Za ustvarjanje Makefile datoteke smo vpisali ukaz: "**cmake -G "Unix Makefiles"**.

8) Za prevajanje datoteke smo vpisali ukaz: "**sudo make**".

9) Da smo naredili novo datoteko po prevajanju, smo vpisali: "**sudo make install**".

10) S tem smo končali prevajanje gonilnika za haptični vmesnik. Knjižnica vsebuje nekaj primerov, s katerimi smo poskusili, ali driver pravilno deluje. Z ukazom: "**cd bin**" smo prišli v mapo "**bin**", v kateri se nahajajo primeri. Nekaj primerov:

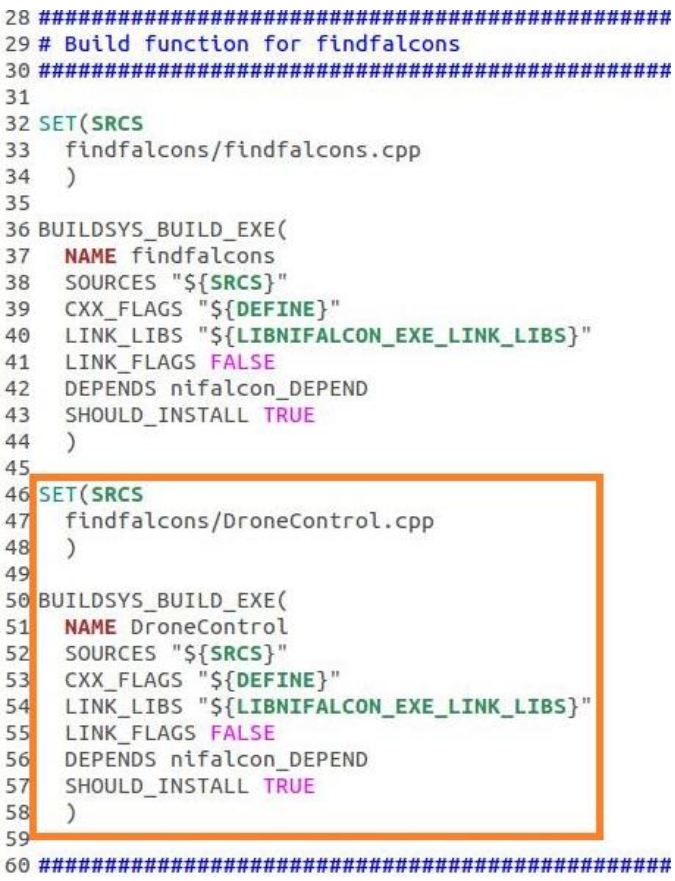
- "**sudo ./findfalcons**" – ta primer bi moral ugotoviti, ali je na računalnik priklopljen Falcon;
- "**sudo ./falcon_led --device_index 0 --led_red**" – ta primer bi moral prižgati rdečo LED na Falconu.

Po preizkušnji je gonilnik delal pravilno.

V nadaljevanju smo v naš program "**DroneControl.cpp**" vključili potrebne knjižnice za delovanje Falcona in definirali objekte, potrebne za komunikacijo s Falconom:

```
#include "falcon/core/FalconDevice.h"
#include "falcon/firmware/FalconFirmwareNovintSDK.h"
#include "falcon/util/FalconFirmwareBinaryNvent.h"
#include "falcon/kinematic/FalconKinematicStamper.h"


//definiramo objekte potrebne za komunikacijo s novint-om
	boost::shared_ptr<FalconFirmware> f;
	FalconDevice dev;
	 FalconKinematicStamper fks(1);
```

Ker smo v našem programu uporabili knjižnice od Falcona (Libnifalcon), smo morali program vključiti v prevajanje skupaj s knjižnicami od Libnifalcona. To smo naredili tako, da smo spremenili "**CMakeLists.txt**", ki se nahaja v "**.../libnifalcon-master/examples**". V datoteko smo dodali vrstice, ki so na sliki 5.1 obkrožene z oranžno barvo. S tem smo vključili tudi prevajanje našega programa in naredili "**DroneControl**" datoteko, ki jo lahko zaženemo. Nahaja se v "**.../libnifalcon-master/build/bin**". Zaženemo jo z ukazom: "./**DroneControl**".




Slika 5.1: Spremenjena datoteka CmakeLists.txt

Potem smo preverili, ali je Falcon priključen na računalnik, in na Falcona naložili programsko opremo:

```
//prerimo, ali je naprava prikljucena oz. zaznana
  //poskusimo odpreti komunikacijo s napravo
  if (!dev.open(0))
  {
        std::cout << "Ne morem odpreti Falcon-a - Napaka: " << std::endl;
        return -1;
  }
  std::cout << "Odpiranje Falcon-a" << std::endl;

  //nastavimo firmware, ki ga bomo uporabljali za komunikacijo s novint-om
  dev.setFalconFirmware<FalconFirmwareNovintSDK>();

  f = dev.getFalconFirmware();
```

```
	//poskusamo naloziti firmware
	if (!dev.isFirmwareLoaded())
	{
		std::cout << "Nalaganje firmware-a" << std::endl;
		for (int i = 0; i < 10; ++i)
		{
			if (!dev.getFalconFirmware()->loadFirmware(true,
	NOVINT_FALCON_NVENT_FIRMWARE_SIZE,
	const_cast<uint8_t*>(NOVINT_FALCON_NVENT_FIRMWARE)))
			{
				std::cout << "Ne morem naložiti firmware-a" << std::endl;

				dev.close();
				if (!dev.open(0))
				{
					std::cout << "Ne morem odpreti Falcon-a - Napaka: 2" << std::endl;
					return -2;
				}
			}
			else
			{
				std::cout << "Firmware naložen" << std::endl;
				break;
			}
		}
		if (!dev.isFirmwareLoaded())
		{
			std::cout << "Firmware se ni naložil pravilno. Poskusi zagnati droneControl ponovno !" <<
std::endl;
			return -3;
		}
	}
```

Haptična naprava nima absolutnega dajalnika položaja, in ko jo prvič povežemo prek USB-ja na računalnik, nima informacij o trenutnem položaju. Naredili smo pogoj, s katerim smo uspešno kalibrirali napravo. Pogoj zahteva, da uporabnik, preden začne napravo uporabljati, tega premakne štirikrat po osi z (dvakrat naprej in dvakrat nazaj).

```
//haptična naprava ima svoje središce na ploţaju (0, 0, 0.125)
	boost::array<double, 3> center, center2;
	center[0] = center[1] = 0;
	 center[2] = .125;

//haptična naprava kalibrira svoj položaj
	int spremembe = 4;
	int lastPos = -1;

	std::cout << "Kalibracija: prosim premikajte prijemalo naprej in nazaj" << std::endl;

	while (spremembe){
		if (dev.runIOLoop()) ++i;
```

```
        else continue;

        //metoda getEncoderValues vrne trenutne vrednosti položaja nog na novintu,
        //s metodo getPosition dobimo položaj prijemala v (x,y,z) koordinatnem sistemu
        boost::array<int, 3> encoded = f->getEncoderValues();
        fks.getPosition(encoded, pos2);

        if (pos2[2] - center[2] < 0){
            if (lastPos > 0){
                --spremembe;
                lastPos = -1;
            }
        }
        else if (lastPos < 0){
            --spremembe;
            lastPos = 1;
        }

    }
```

Pri branju položaja prijemala s haptičnega vmesnika smo dobili za vse osi (x, y, z) vrednosti med -1600 in 1600. Ker za premike kvadrokopterja potrebujemo vrednosti med -1 in 1, smo vrednosti morali skalirati. Cela števila med -1600 in 1600 smo skalirali in pretvorili v števila s plavajočo vejico v vrednosti med -1 in 1 ter jih zaokrožili na eno decimalno mesto. Potem smo s funkcijo **toArg** števila s plavajočo vejico pretvorili v celoštevilčna števila (za podrobnosti glej funkcijo **toArg**).

```
//realno stevilo zapisano z IEEE 754 standardom pretvorimo v celo stevilo tako, da spremenimo podatkovni tip (iz float v int, pri tem moramo paziti da ne spremenimo "vsebino" podatka)
        int toArg(float val){
            return *((int*)((void*)&val));
            }

    //dobimo položaj osi s haptične naprave in ga pretvorimo v realen 3D koordinatni sistem
        boost::array<double, 3> pos;
        boost::array<int, 3> encoded = f->getEncoderValues();
        fks.getPosition(encoded, pos);
        fks.getPosition(encoded, pos2);

    //vrednosti položaja prijemala skaliramo
        pos[0] = (pos[0] - center2[0]) * 16.6666;
        pos[1] = (pos[1] - center2[1]) * 16.6666;
        pos[2] = (pos[2] - center2[2]) * 20;
```

Dobljene vrednosti položaja prijemala smo vnesli v ukaz AT*PCMD, in sicer v argumente 3, 4, 5 in 6. Vrednosti z osi x smo vnesli v argument 3, s katerim podajamo

odstotek zasuka okoli osi x, nagib v levo in desno stran (roll). Vrednosti z osi y smo vnesli v argument 4, s katerim podajamo odstotek naklona okoli osi y, nagib naprej in nazaj (pitch). Vrednosti z osi z smo vnesli v argument 5, s katerim podajamo odstotek maksimalne hitrosti translacijskega gibanja v smeri osi z (gaz). V argument 6, ki podaja odstotek maksimalne kotne hitrosti rotacije okoli osi z, odklon (yaw), smo za začetek vpisali 0.

```
//vnesemo vrednosti položaja prijemala v ukaz AT*PCMD
    sprintf(message, "AT*PCMD=%d,%d,%d,%d,%d,%d\r", st++, 1, toArg(pos[0]), toArg(pos[2]),
    toArg(pos[1]), 0);
```

Na koncu smo s pomočjo nastavljanja sil na haptičnem vmesniku nastavili sile tako, da vse delujejo v isto smer z enako močjo, da se prijemalo vedno vrne v izhodišče koordinatnega sistema:

```
float napaka = 1000;
    int last = time(0);

    //dokler se prijemalo premika iscemo novo središče haptične naprave
    do{
        //če naprava ni pripravljena za dobivanje in pošiljanje podatkov čakamo, sicer povečamo stevec
        if (dev.runIOLoop()) ++i;
        else continue;

        boost::array<int, 3> encoded = f->getEncoderValues();
        //branje trenutnega položaja s haptične naprave
        fks.getPosition(encoded, pos2);

        //izračunamo vektor od centra do trenutnega položaja haptične naprave
        //prijemalo elimo premakniti v center, tako da uporabimo silo, ki kaze v enaki smeri, kot izračunan
        vektor
        force[0] = (pos2[0] - center[0]) * 75;
        force[1] = (pos2[1] - center[1]) * 75 - .6;
        force[2] = (pos2[2] - center[2]) * 75;

        //izračunamo razdaljo med trenutnim položajem in središčem haptične naprave
        float currNapaka = abs(pos2[0] - center[0]) + abs(pos2[1] - center[1]) + abs(pos2[2] - center[2]);

        //metoda getForces izračuna silo posameznih nog, ki je potrebna, da prestavimo prijemalo
        //za prej izračunano silo (proti središču)
        //metoda vrne true, v kolikor je uspela izračunat sile
        forceFinal[0] = forceFinal[1] = forceFinal[2] = 0;
        //ker smo uporabili vektor od centra proti trenutnem položaju, potrebno je dobljenim silam zamenjati
        predznak
        if (fks.getForces(pos2, force, forceFinal)){
            forceFinal[0] = -forceFinal[0];
            forceFinal[1] = -forceFinal[1];
            forceFinal[2] = -forceFinal[2];
```

```
            f->setForces(forceFinal);
        }

        //če se je prijemalo premaknilo za dovolj veliko vrednost (vec od .1) upoštevamo, da je prišlo do
        premika, v nasprotnem primeru upoštevamo, kot da miruje
        if (napaka - currNapaka > .1f || napaka - currNapaka < -.1f){
            napaka = currNapaka;
            last = time(0);
        }

        sched_yield();
    }

    //kalibracija se izvaja, dokler napaka ni dovolj majhna, ali dokler prijamalo ne miruje za vsaj 3s
    while (napaka > .5 || (time(0) - last) < 3);
```

## 5.3 Upravljanje kvadrokopterja

Na koncu smo v naš program "**DroneControl.cpp**" napisali kodo za upravljanje AR.Drona.

Po končani kalibraciji haptičnega vmesnika pošljemo prvi ukaz (AT*FTRIM=1) na kvadrokopter.

```
sprintf(message, "AT*FTRIM=%d\r", st++);

std::cout << message << std::endl;

//ukaz posljemo preko protokola UDP na kvadrokopter
    if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
        {
            perror("sendto()");
            exit(1);
    }
```

Ukaz smo poslali samo enkrat in smo ga uporabili za vodoravno usmeritev oziroma za postavljanje žiroskopa v vseh treh oseh (x, y, z) na 0°. Ukaz je treba poslati, ko je kvadrokopter na tleh. Sledi pošiljanje ukaza (AT*REF=2,0), v katerem je bit 9 postavljen na 0, kar pomeni, da se bo kvadrokopter nahajal na tleh.

```
sprintf(message, "%s%d,%d\r", "AT*REF=", st++, 0);

std::cout << message << std::endl;
```

```
//ukaz posljemo preko protokola UDP na kvadrokopter
      if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
            {
                  perror("sendto()");
                  exit(1);
      }
```

Ukaze pošiljamo tako dolgo, dokler ne pritisnemo tipke, ki se nahaja na prijemalu in jo lahko vidimo na sliki 5.2, označeno s številko 1. Ukaze pošiljamo, da ne bi izgubili WiFi-povezave med kvadrokopterjem in računalnikom. Ko pritisnemo tipko, ki je označena na sliki 5.2, z oznako 1 pošljemo ukaz (AT*REF=19,512) s postavljenim bitom 9 na 1, kar pomeni, da se kvadrokopter dvigne in se s pomočjo ultrazvočnega senzorja ustali na višini 1 m od tal.

```
sprintf(message, "AT*REF=%d,%d\r", st++, (val | buttonState) << 9);

std::cout << message << std::endl;


//ukaz posljemo preko protokola UDP na kvadrokopter
  if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
            {
                  perror("sendto()");
                  exit(1);


   }
```

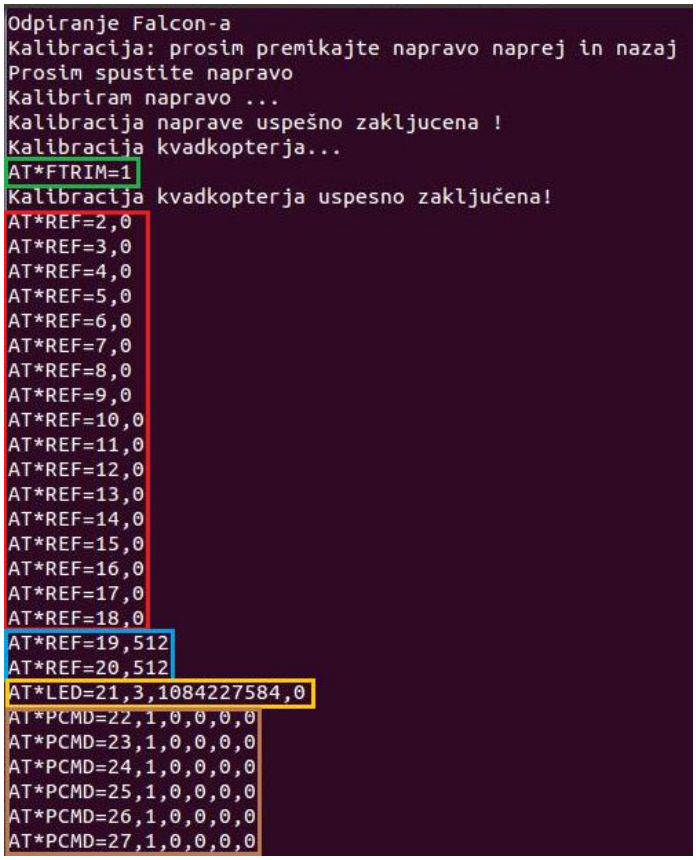
Istočasmo pošljemo tudi ukaz (AT*LED=21,3,1084227584,0), ki pomeni, da v času vodenja kvadrokopterja vse štiri LED-diode začnejo utripati z oranžno barvo in frekvenco 0,2 Hz.

```
sprintf(message, "AT*LED=%d,%d,%d,%d\r", st++, 3, toArg(5), 0);

std::cout << message << std::endl;

//ukaz posljemo preko protokola UDP na kvadrokopter
      if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
            {
                  perror("sendto()");
                  exit(1);
   }
```


Slika 5.2: Prijemalo haptične naprave Novint Falcon [**15**]

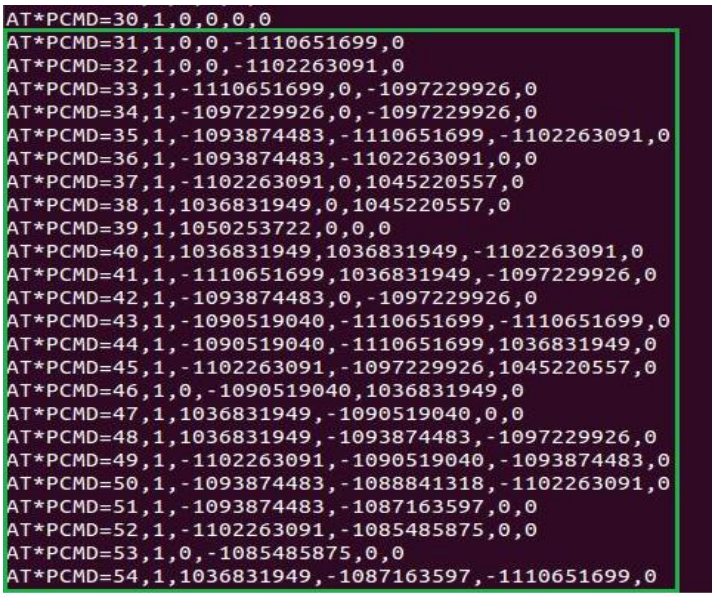
Po ukazih za dvig kvadrokopterja in nastavljanje barve svetleči diodi pošiljamo ukaz (AT*PCMD=22,1,0,0,0,0) za premikanje kvadrokopterja. Trenutno se kvadrokopter ne premika, ker se prijemalo haptičnega vmesnika nahaja v izhodišču ter so argumenti 3, 4, 5 in 6 ukaza AT*PCMD postavljeni na 0, kot vidimo na sliki 5.3.

```
sprintf(message, "AT*PCMD=%d,%d,%d,%d,%d,%d\r", st++, 1, toArg(pos[0]), toArg(pos[2]),
toArg(pos[1]), toArg(rotation));


std::cout << message << std::endl;

//ukaz posljemo preko protokola UDP na kvadrokopter
      if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
            {
                  perror("sendto()");
                  exit(1);
   }
```




Slika 5.3: Pošiljanje AT-ukazov prek UDP-ja na kvadrokopter (1)

Ko premikamo prijemalo, se začnejo vrednosti v argumentih ukaza AT*PCMD spreminjati, kot vidimo na sliki 5.4.

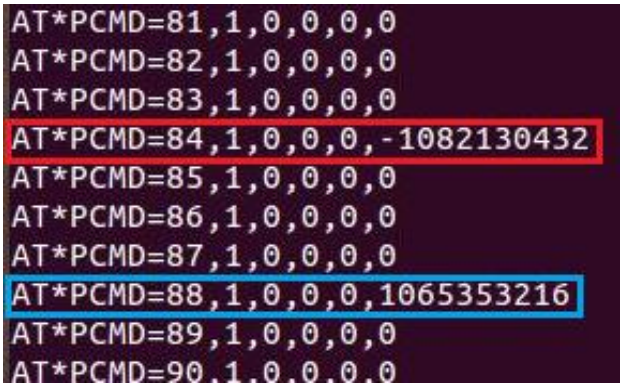



Slika 5.4: Pošiljanje AT-ukazov prek UDP-ja na kvadrokopter (2)

Ko pritisnemo tipko, ki je označena na sliki 5.2 z oznako 3, pošljemo ukaz (AT*PCMD=84,1,0,0,0,-1082130432), kot vidimo na sliki 5.5, kar pomeni, da smo v argument 6 vnesli število -1,0. Argument 6 podaja odstotek maksimalne kotne hitrosti rotacije okoli osi z, odklon (yaw). Ker ima število negativen predznak, se bo kvadrokopter z maksimalno hitrostjo vrtel okoli osi z v levo stran.

Ko pritisnemo tipko, ki je označena na sliki 5.2, z oznako 4, pošljemo ukaz (AT*PCMD=88,1,0,0,0,1065353216), kot vidimo na sliki 5.5, kar pomeni, da smo v argument 6 vnesli število 1,0. Argument 6 podaja odstotek maksimalne kotne hitrosti rotacije okoli osi z, odklon (yaw). Ker ima število pozitiven predznak, se bo kvadrokopter z maksimalno hitrostjo vrtel okoli osi z v desno stran.

```
//nastavimo rotacije z gumbom za levo ali desno stran
        if (*(f->getGripInfo()) & 1)
                rotation = 1;
        else if (*(f->getGripInfo()) & 8)
                rotation = -1;
        else
                rotation = 0;
```




Slika 5.5: Pošiljanje AT-ukazov prek UDP-ja na kvadrokopter (3)

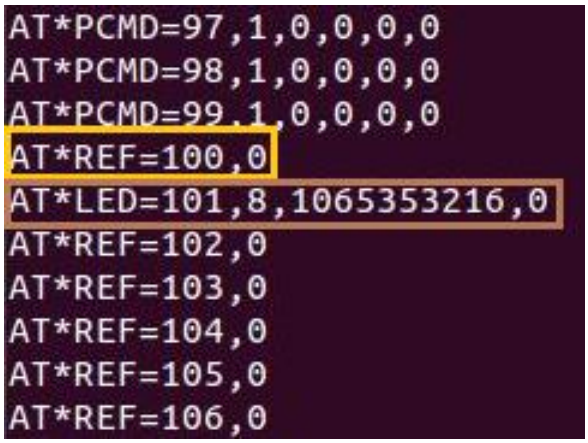
Ko pritisnemo tipko, ki je označena na sliki 5.2, z oznako 2, pošljemo ukaz (AT*REF=93,256). Vidimo, da ima argument 2 vrednost 256, kar pomeni, da smo Bit 8 postavili na 1. S tem smo vključili prisilni izklop, kar pomeni izključitev vseh motorjev.

```
//ob pritisku na okrogel gumb pošlemo signal za spremembo stanja emergency
            if ((*(f->getGripInfo()) & 4)){
                  //signal za spremembo stanja poslemo samo 1x na pritisk gumba
                  if (!emergencyShift){
                        emergencyShift = 1;

                        sprintf(message, "AT*REF=%d,%d\r", st++, 1 << 8);

                        std::cout << message << std::endl;

//ukaz posljemo preko protokola UTP na letalo
      if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
                        {
                              perror("sendto()");
                              exit(1);
                        }
                  }

            }
            else
                              emergencyShift = 0;
```

Ko ponovno pritisnemo tipko, ki je označena na sliki 5.2, z oznako 1, pošljemo ukaz (AT*REF=100,0). Vidimo, da ima argument 2 vrednost 0, kar pomeni, da smo bit 9 ponovno postavili na 0. S tem smo spustili kvadrokopter na tla. Istočasno pošljemo tudi

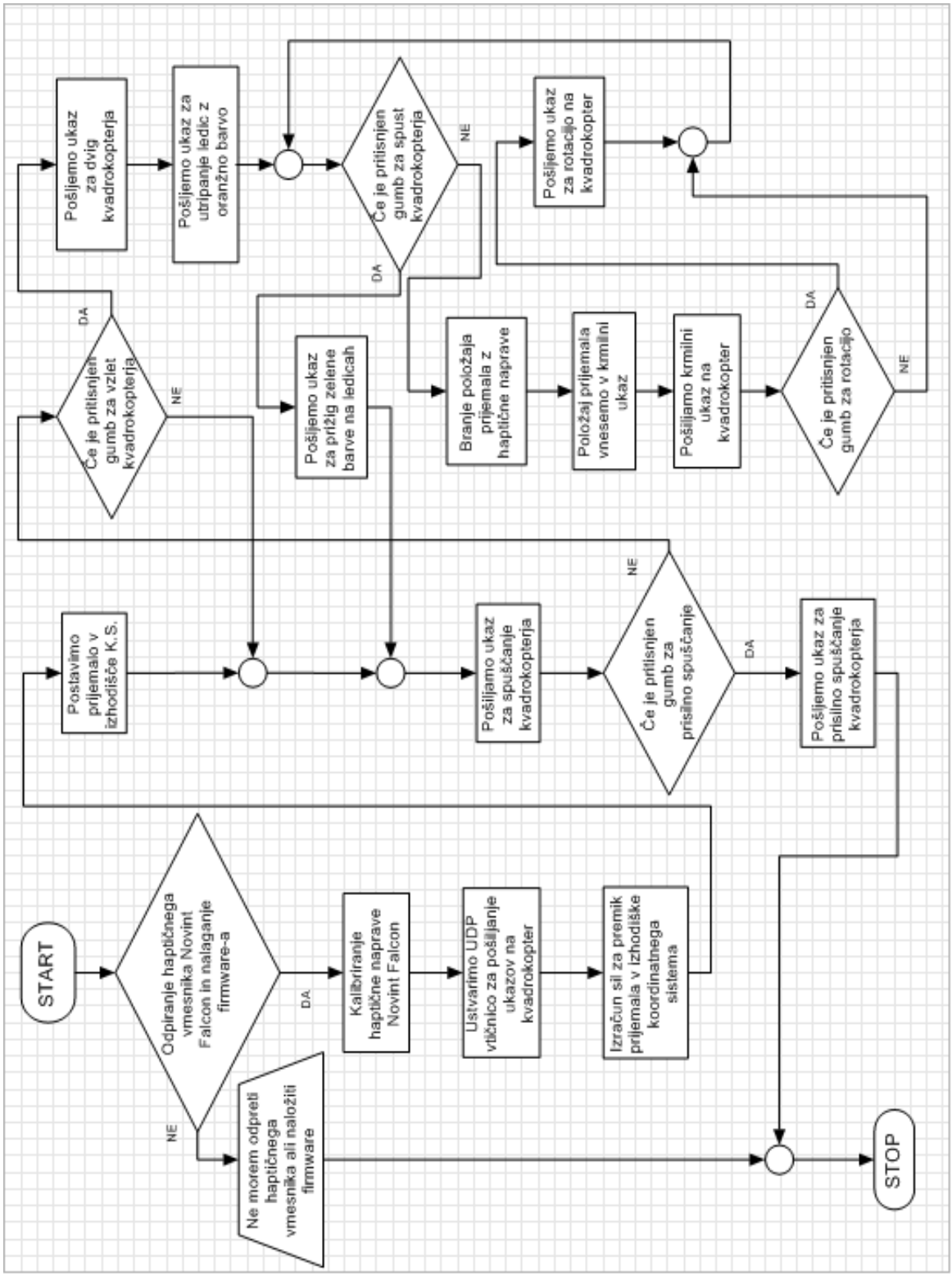
ukaz (AT*LED=101,8,1065353216,0), kot vidimo na sliki 5.6. S tem smo vključili vsem svetlečim diodam zeleno barvo.

```
sprintf(message, "%s%d,%d\r", "AT*REF=", st++, 0);

std::cout << message << std::endl;


//ukaz posljemo preko protokola UDP na kvadrokopter
      if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
            {
                  perror("sendto()");
                  exit(1);
       }


sprintf(message, "AT*LED=%d,%d,%d,%d\r", st++, 8, toArg(1), 0);

std::cout << message << std::endl;


//ukaz posljemo preko protokola UDP na kvadrokopter
      if (sendto(s, message, strlen(message), 0, (struct sockaddr *) &si_other, slen) == -1)
            {
                  perror("sendto()");
                  exit(1);
     }
```




Slika 5.6: Pošiljanje AT-ukazov prek UDP-ja na kvadrokopter (4)

Delovanje programa je možno lažje predstaviti s pomočjo diagrama poteka. Lažje in hitreje tudi vidimo korake delovanja programa. Diagram je prikazan na sliki 5.7.




Slika 5.7: Diagram poteka

# 6 REZULTATI

## 6.1 Testiranje kvadrokopterja s programsko opremo AR.Drone SDK

Za testiranje kvadrokopterja smo uporabili AR.Drone SDK 2.0.1 (programsko opremo), ki je dostopna na https://projects.ardrone.org/boards/1/topics/show/5346. Prevzeli smo paket in začeli z inštalacijo v programu Terminal. Ime paketa je "**ARDrone_SDK_2_0_1**".

1) Da smo lahko začeli z nameščanjem programske opreme, smo se morali prestaviti v tisto mapo, v katero smo shranili programsko opremo. To smo naredili s pomočjo ukaza "**cd**". Z ukazom "**ls**" lahko vidimo, katere datoteke se nahajajo v trenutni mapi.

2) Vpisali smo ukaz "**uname -a**".

3) Inštalirali smo dodatne knjižnice, ki jih potrebujemo za inštaliranje programske opreme. To smo naredili z ukazi:

"**sudo apt-get install libgtk2.0-dev**",
"**sudo apt-get install libsdl1.2-dev**",
"**sudo apt-get install libiw-dev**",
"**sudo apt-get install libxml2-dev**",
"**sudo apt-get install libudev-dev**",
"**sudo apt-get install libncurses5-dev libncursesw5-dev**".

4) Za prevajanje programa smo vpisali ukaz "**sudo make -j8**".

5) Vpisali smo ukaz "**cd /Build/Release/**", da pridemo v mapo "**Release**".

6) V mapi "**Release**" se nahaja program pod imenom "**ardrone_navigation**"
, ki smo ga uporabili za testiranje kvadrokopterja.

7) Prek WiFi-ja smo se povezali na kvadrokopter.

8) Program smo zagnali z ukazom: "**sudo ./ardrone_navigation**".

Na začetku, ko smo prevajali program, je prevajalnik našel napake. Napake smo popravili tako, da smo v nekaterih datotekah zamenjali vrstice:

1) Vrstico "**USE_LINUX=no**" smo spremenili v "**USE_LINUX=yes**" v datoteki "**custom.makefile**", ki se nahaja v "**ARDronelib/Soft/Build/**".
2) V Terminalu smo z ukazom "**cd /ARDroneLib/Soft/Build/**" prišli v mapo "**Build**" in vpisali ukaz "**sudo make**".

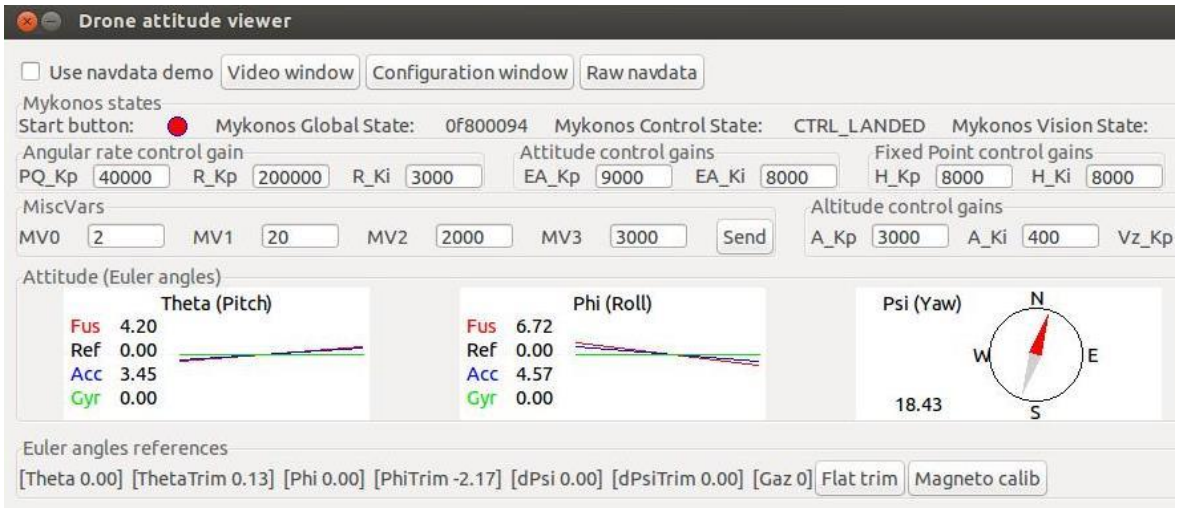
3) V datoteki "**Makefile**", ki se nahaja v "**ARDrone_SDK_2_0_1/Examples/Linux/video_demo/Build/**" smo vrstico "**Add to GENERIC_LIBS in line 47 -lm -lgobject-2.0 -lgdk-x11-2.0** " zamenjali z vrstico "**GENERIC_LIBS=-lpc_ardrone -lrt -lgtk-x11-2.0 -lcairo -lgobject-2.0 -lgdk-x11-2.0 –lm**".

4) V datoteki "**Makefile**", ki se nahaja v "**ARDronelib/Examples/Linux/Navigation/**", smo dodali k vrstici "**GENERIC_LIBS**" ukaz "**-lm**".

5) Z ukazom: "**cd ..**" smo prišli v mapo "**Linux**" in vpisali ukaz "**sudo make**".

Testirali smo ultrazvočni senzor višine, 3-osni žiroskop, 3-osni pospeškometer, motorje in kamere.

Žiroskop smo testirali na način, da smo kvadrokopter nagibali okoli vseh treh osi in s tem spreminjali kote nagiba, na računalniku pa smo opazovali rezultate. Rezultati meritev so bili natančni. Na slikah 6.1 in 6.2 je prikazano testiranje 3-osnega žiroskopa.



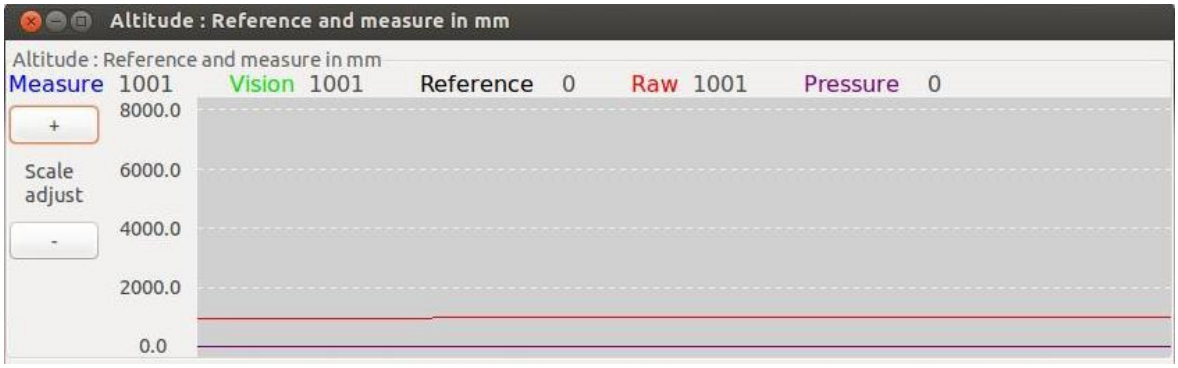

Slika 6.1: Testiranje 3-osnega žiroskopa kvadrokopterja AR.Drone (1)


Slika 6.2: Testiranje 3-osnega žiroskopa kvadrokopterja AR.Drone (2)

Potem smo kvadrokopter hitro premikali po vseh treh oseh in na računalniku opazovali pospeške. Na sliki 6.3 je prikazano testiranje 3-osnega pospeškometra. Pospeškometer je deloval pravilno.

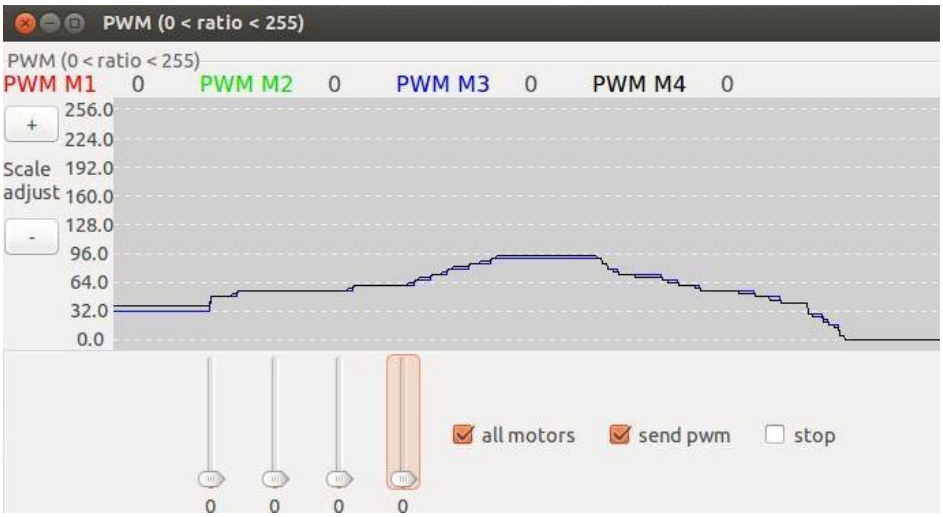

Slika 6.3: Testiranje 3-osnega pospeškometra kvadrokopterja AR.Drone

Ultrazvočni senzor višine smo testirali tako, da smo kvadrokopterja postavili na 1 m višine. Izkazalo se je, da senzor deluje pravilno. Testiranje ultrazvočnega senzorja višine je prikazano na sliki 6.4.



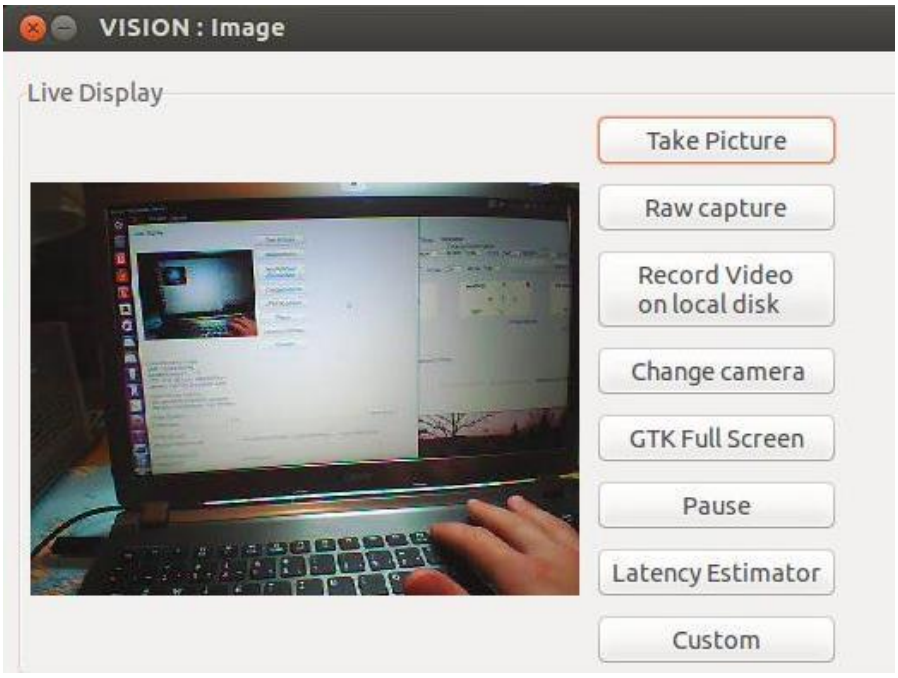

Slika 6.4: Testiranje ultrazvočnega senzorja višine kvadrokopterja AR.Drone

Motorje smo testirali na način, da smo na vsak motor pošiljali enak PWM-signal (pulzno-širinsko moduliran) med vrednostmi 0 in 255 ter smo na računalniku opazovali, ali se vsi štirje motorji vrtijo z enako kotno hitrostjo.

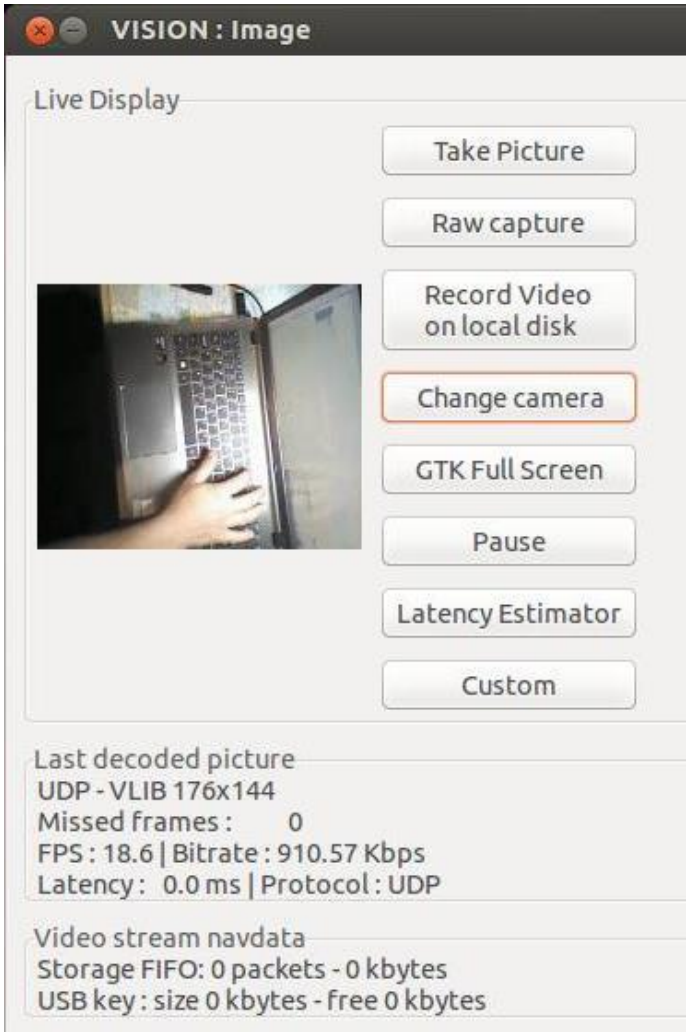



Slika 6.5: Testiranje motorjev kvadrokopterja AR.Drone

Na slikah 6.6 in 6.7 so prikazani rezultati testiranja kamer.




Slika 6.6: Testiranje prednje kamere kvadrokopterja AR.Drone




Slika 6.7: Testiranje vertikalne (spodnje) kamere kvadrokopterja AR.Drone

## 6.2 Vodenje kvadrokopterja s haptično napravo

Kot rezultat vodenja kvadrokopterja s haptično napravo smo posneli video posnetek, ki se nahaja na spletni strani https://www.youtube.com/watch?v=_1bwsfBexCo in v prilogi. V prilogo smo dodali slike vodenja kvadrokopterja in program napisan v programskem jeziku C++.

Na slikah 6.8 in 6.9 je prikazano vodenje kvadrokopterja Ar.Drone s haptičnim vmesnikom Novint Falcon.

Slika 6.8: Vodenje kvadrokopterja AR.Drone s haptičnim vmesnikom Novint Falcon (1)

Slika 6.9: Vodenje kvadrokopterja AR.Drone s haptičnim vmesnikom Novint Falcon (2)

# 7 SKLEP

V diplomskem delu smo imeli cilj izdelati vodenje kvadrokopterja AR.Drone 1.0 s haptičnim vmesnikom Novint Falcon. Na začetku smo imeli manjše težave s programsko opremo kvadrokopterja za operacijski sistem Windows in smo se odločili, da bomo diplomsko delo opravili na operacijskem sistemu Linux Ubuntu. Da smo namestili vse gonilnike za Linux Ubuntu, smo izgubili veliko časa, a smo se s tem naučili uporabljati različne programe in ukaze, brez katerih Linux sploh ni koristen. Tako smo se naučili uporabljati program Terminal, v katerem smo inštalirali vse programe, potrebne za opravljanje diplomskega dela, in gonilnike, ki jih Linux ni uspel sam naložiti. Naučili smo se tudi prevajati programe z različnimi prevajalniki.

Ko smo napisali program za vodenje kvadrokopterja s haptično napravo v programskem jeziku C++, smo želeli poskusiti delovanje programa. Ko smo dvignili kvadrokopter, se ta ni ustalil na višini 1 m od tal. Začeli smo s testiranjem pospeškometra, žiroskopa, senzorja višine itd. in ugotovili, da vse deluje pravilno. Težava je bila v firmwaru in tako smo prek FTP-ja (File Transfer Protocol), ki se uporablja za premikanje datotek iz enega gostitelja na drugega, prek omrežja, ki temelji na TCP-ju (Transmission Control Protocol), naložili novi firmware. Zamenjali smo tudi propelerje in gredi in tako je kvadrokopter po ponovnem vklopu deloval pravilno. S tem smo podrobneje spoznali strojno in programsko opremo kvadrokopterja.

Diplomsko delo ni bilo preprosto in enostavno, a je bila zanimivo in izobraževalno. Vložili smo veliko dela in truda, a smo na koncu dosegli zadani cilj, ki je bil vodenje kvadrokopterja s haptičnim vmesnikom, s čimer smo potrdili tehnično izvedljivost zadane naloge.

## 7.1 Nadaljnje delo

Ker s haptičnim vmesnikom lahko čutimo sile, bi lahko kvadrokopterja opremili z ultrazvočnimi senzorji. Z njimi bi merili razdaljo med kvadrokopterjem in ovirami ter bi ustvarili virtualno silo, ki bi jo prenesli na haptični vmesnik kot fizično povratno silo. S tem bi omogočili lažje vodenje v realnem neurejenem okolju.

Spremljamo lahko videoprenos, a ne istočasno z vodenjem kvadrokopterja. Lahko vodimo kvadrokopter ali spremljamo video s kvadrokopterja. Zanimivo bi bilo istočasno voditi kvadrokopter s haptičnim vmesnikom in spremljati video. S tem bi ga lahko vozili tudi izven našega vidnega polja.

# 8 VIRI, LITERATURA


[1] Centralni križ kvadrokopterja AR.Drone 1.0. http://www.ebay.com/itm/Parrot-AR-Drone-1-0-APP-Carbon-Fiber-Framework-Frame-Shelf-Shaft-Central-Cross-X-/251324261749.

[2] Smeri vrtenja motorjev kvadrokopterja. http://www.hobbywow.com/en-preorder-walkera-hoten-x-6-axis-gyro-wifi-rc-fpv-quadcopter-bnf-p234364.htm.

[3] Koti kvadrokopterja. https://engineering.purdue.edu/ece477/Archive/2011/Spring/S11-Grp09/nb/ostaton.html.

[4] SDK priročnik za kvadrokopter AR.Drone 1.0. https://projects.ardrone.org/wiki/ardrone-api#SDK-201.

[5] Matična plošča kvadrokopterja AR.Drone 1.0. http://diydrones.com/profiles/blogs/parrot-ardrone-teardown.

[6] Prednja kamera kvadrokopterja AR.Drone 1.0. https://www.ifixit.com/Teardown/Parrot+AR.Drone+Teardown/3984.

[7] Baterija kvadrokopterja AR.Drone 1.0 z 1000 mAh. http://www.batteryforum.org/t313-parrot-ardrone-helicopter-battery-pf070009aa.

[8] Zgornja stran navigacijske plošče kvadrokopterja AR.Drone 1.0. https://www.ifixit.com/Teardown/Parrot+AR.Drone+Teardown/3984.

[9] Spodnja stran navigacijske plošče kvadrokopterja AR.Drone 1.0. https://www.ifixit.com/Teardown/Parrot+AR.Drone+Teardown/3984.

[10] Motor z regulatorjem, mikrokrmilnikom in AD pretvornikom kvadrokopterja AR.Drone 1.0. https://www.ifixit.com/Teardown/Parrot+AR.Drone+Teardown/3984.

[11] Notranjost haptične naprave Novint Falcon. http://www.lunar.com/failure-is-always-an-option/.

[12] Prijemalo haptične naprave Novint Falcon v obliki krogle. http://arstechnica.com/gadgets/2007/10/meet-the-novint-falcon-the-craziest-force-feedback-controller-weve-ever-played-with/.

[13] Zaščitni znak operacijskega sistema Linux.
http://en.wikipedia.org/wiki/Tux.

[14] prilagojeno po viru Prijemalo haptične naprave Novint Falcon.
http://www.bit-tech.net/gaming/pc/2007/12/07/novint_falcon_limited_edition/1.

[15] TCP/IP (internetni sklad protokolov).
http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/06_skladi/tcpip2.html.

[16] TCP.
http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/40_transportni/tcp.html.

[17] UDP.
http://www.s-sers.mb.edus.si/gradiva/w3/omrezja/40_transportni/udp.html.

[18] Uporabniški priročnik kvadrokopterja AR.Drone 1.0.
http://ardrone2.parrot.com/media/uploads/support_ardrone_1/ar.drone_user-guide_uk.pdf.

[19] Opis kvadrokopterja AR.Drone 1.0.
http://www.ifixit.com/Teardown/Parrot+AR.Drone+Teardown/3984.

[20] Opis strojne opreme kvadrokopterja AR.Drone 1.0.
http://edn.com/design/analog/4429404/3/Drone-design--An-electronics-designer-s-point-of-view-Part-one.

[21] Uporabniški priročnik za Novint Falcon. http://76.12.0.65/download/User%20Manual.pdf.

[22] Opis haptičnega vmesnika Novint Falcon. http://en.wikipedia.org/wiki/Novint_Technologies.

[23] Tehnične lasnosti haptičnega vmesnika Novint Falcon.
http://www.novint.com/index.php/novintxio/41.

[24] Kako začeti z Ubuntujem.
https://www.ubuntu.si/wordpress/wp-content/uploads/ubuntu-manual-sl-final.pdf.

# 9 PRILOGA

V prilogo smo vključili opis TCP/IP-komunikacijskega sklada, program, napisan v C++, videoposnetek in slike vodenja kvadrokopterja s haptično napravo Novint Falcon.

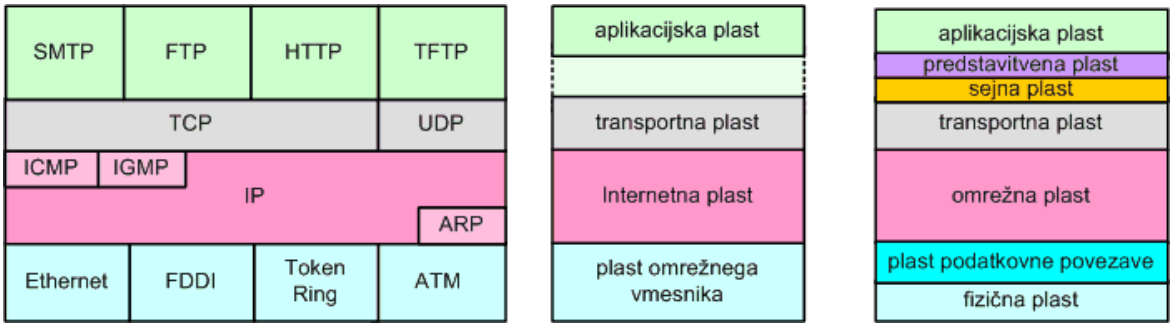
## 9.1 Priloga A: TCP/IP-komunikacijski sklad

**TCP/IP** (angl. "Transmission Control Protocol" / "Internet Protocol") ali **internetni sklad protokolov** (angl. "Internet protocol suite") je množica protokolov, ki izvaja protokolski sklad, prek katerega teče internet. Največ omrežnega prometa poteka prek protokola TCP. Sporočila prek protokola TCP se zaradi vzpostavljene povezave med odjemalcem in servisom prenašajo zanesljivo v obe smeri, so brez napak, podvojevanja in v pravem vrstnem redu.

TCP/IP ali internetni sklad protokolov se popolnoma ne ujema z referenčnim modelom OSI. Namesto sedem plasti uporablja le štiri:

- Aplikacijsko plast;
- Transportno plast;
- Internetno plast;
- Plast omrežnega vmesnika.

Vsaka izmed teh plasti približno ustreza eni ali več plastem OSI-referenčnega modela. Plasti TCP/IP-protokolnega sklada in plasti ISO-modela so prikazane na sliki 9.1.




Slika 9.1: Plasti TCP/IP-protokolnega sklada in referenčnega ISO-modela [**16**]

### Plast omrežnega vmesnika

Plast omrežnega vmesnika ustreza fizični plasti in plasti podatkovne povezave referenčnega modela OSI, komunicira neposredno z omrežjem. Je vmesnik med omrežno arhitekturo in internetnim slojem. Sam protokolni sklad TCP/IP ne standardizira teh slojev in prav to je njegova velika prednost. Na sloju omrežnega vmesnika je veliko protokolov lokalnih omrežij (npr. Ethernet, Token ring, ATM in FDDI), ki jih je TCP/IP zmožen povezati v prostrano omrežje s svojimi protokoli na višjih slojih.

### Internetna plast

Na internetni plasti protokoli enkapsulirajo okvirje protokolov lokalnih omrežij v internetne datagrame in zaženejo vse potrebne usmerjevalne algoritme. Štirje protokoli so najpomembnejši na internetni plasti: Internet Protocol (IP), Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP) in Internet Group Management Protocol (IGMP).

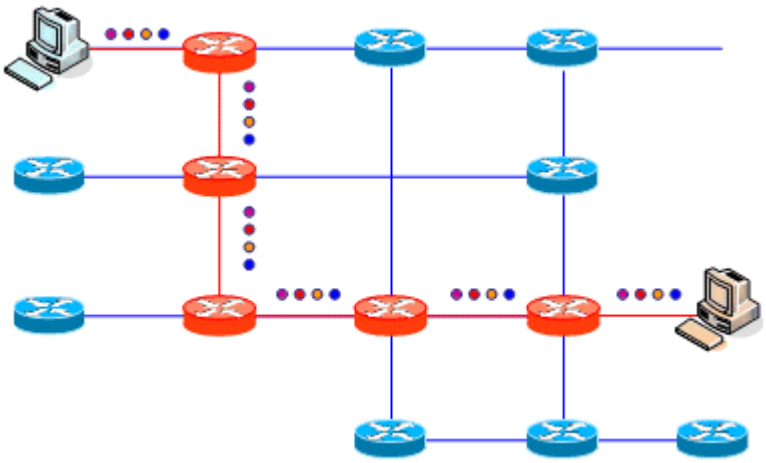
IP lahko povezuje različna omrežja in je predvsem odgovoren za naslavljanje in usmerjanje paketkov med računalniki in omrežji. ARP služi za povezavo med strojnimi in logičnimi (IP) naslovi strojne opreme računalnikov in drugih sistemov v istem fizičnem omrežju. ICMP pošilja sporočila in sporoča napake glede pošiljanja paketa. IGMP

uporabljajo računalniki IP z namenom, da sporočajo imena skupine za oddajanje več sistemom (ang. multicast), to je računalnikom in usmerjevalnikom.

### Transportna plast

Transportna plast nudi dodatno povezavo pod sejnim slojem. Transportna plast zagotavlja zanesljivost prenosa. Paketi so dostavljeni brez napak, v pravilnem zaporedju, brez izgub ali podvajanja in po ugodni ceni. Na pošiljateljevem računalniku ta plast razvrsti sporočila, dolga sporočila deli na več paketkov in zbere manjše paketke v en paket. Ta proces zagotovi, da so paketi uspešno preneseni prek omrežja. Na sprejemnem računalniku transportna plast odpre paket, ponovno sestavi originalno sporočilo in pošlje potrdilo, da je bilo sporočilo sprejeto. Če prispe podvojen paket, ga ta plast zazna in ga zavrže. Dva transportna protokola sta Transmission Control Protocol (**TCP**) in User Datagram Protocol (**UDP**). Izbira transportnega protokola je odvisna od privzete metode pošiljanja podatkov.

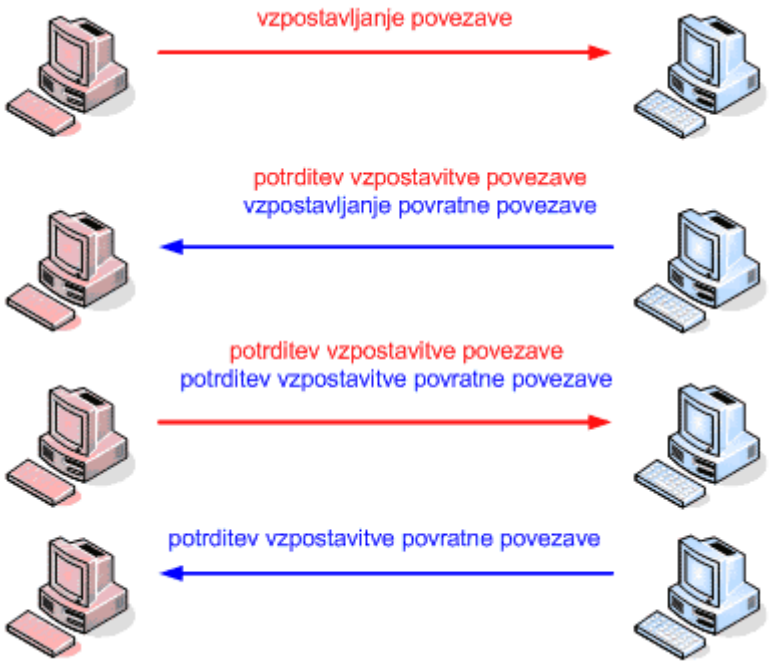
**TCP (**angl. **Transmission Control Protocol)** je protokol, ki se uporablja za zanesljiv pretok podatkov med gostitelji in omrežji. TCP-podatke razdeli na koščke, jim nato doda informacije, ki so potrebne, da podatki najdejo pot do destinacije, in nato koščke ponovno sestavi na koncu povezave. Koščki se imenujejo datagrami. TCP doda k podatkom aplikacijskega sloja v datagram glavo, ki vsebuje informacije, ki so potrebne zato, da podatki pridejo do mesta, na katerega so bili poslani. Najpomembnejše informacije v glavi so: številka vrat, začetno število datagrama in kontrolna vsota. Potek prenosa podatkov preko TCP-ja je prikazan na sliki 9.2.



Slika 9.2: Potek prenosa podatkov prek TCP-ja [**17**]

Ker je TCP povezavni protokol, se najprej vzpostavi povezava med odjemalcem in pošiljateljem. Pri povezavi so določeni odjemalčev naslov IP in vrata ter pošiljateljev naslov IP in vrata, na katerih posluša storitev strežnika. Naslov IP, povezan z določenimi vrati, tvori vtičnico (ang. socket) in par odjemalčeve in pošiljateljeve vtičnice tvori povezavo TCP, ki je edinstveno določena. Glava (angl. header) paketa TCP vsebuje izvorni naslov IP in vrata, ciljni naslov IP in vrata, zaporedno številka paketa, številka potrditve in kontrolne zastavice.

Vsak sistem, ki sprejme podatke, pošlje svoj odgovor. Ko enkrat strežnik dobi odjemalčevo potrditev, je povezava vzpostavljena in sistemi so pripravljeni, da izmenjujejo sporočila, ki vsebujejo aplikacijske podatke. Zato je povezava TCP v bistvu sestavljena iz dveh ločenih enosmernih povezav, ki tečeta v nasprotni smeri, zato je TCP **polno dvosmeren protokol.** Na sliki 9.3 je prikazan potek vzpostavljanja polno dvosmerne povezave.

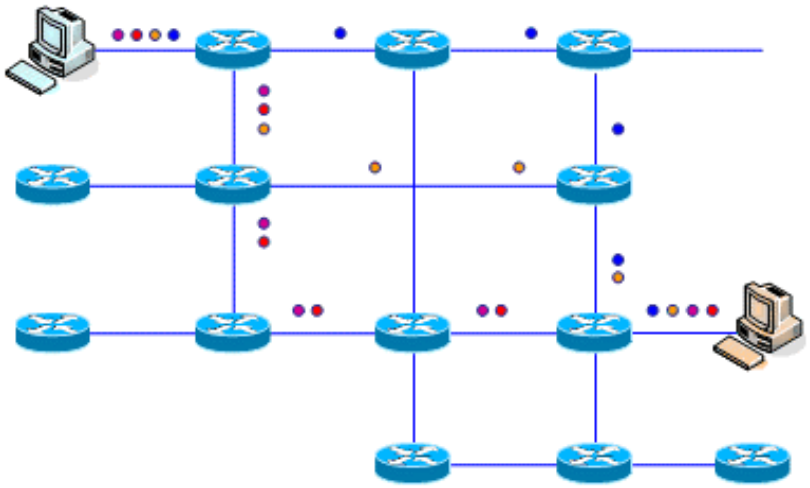



Slika 9.3: Vzpostavljanje polno dvosmerne povezave TCP-a [**17**]

**UDP (**angl. **User Datagram Protocol)** v primerjavi s TCP-jem, ki deluje na istem sloju, ni zanesljiv in tudi ne vzpostavlja povezave. UDP ima enako nalogo kot TCP, toda nima nikakršnih postopkov, ki bi zagotovili, da je paket prišel do prejemnika. Podatke poskuša poslati in preveriti, ali jih je oddaljeni računalnik sprejel. UDP prav tako ne številči paketov in zato ni v stanju prejetih paketov razvrstiti nazaj v pravilno zaporedje. Način prenosa UDP-ja je zlasti primeren za pošiljanje manjših datotek in sicer takšnih, pri katerih ni potrebne garancije za sprejetje. Potek UDP-ja pri komunikaciji uporablja tudi vrata (angl. port) kot TCP, vendar ni potrebno usklajevanje številke vrat in tako ne prihaja do nepotrebnih neprijetnosti. Tudi UDP se tako kot TCP prenaša znotraj datagrama IP.

Protokol je nepovezovalen. To pomeni, da ni potrebna predhodna vzpostavitev povezave z oddaljenim računalnikom (kot je na primer s protokolom TCP), ampak se podatki takoj pošljejo drugemu oddaljenemu računalniku. Prav tako ni zagotovljeno, da že

poslani paketi prispejo do prejemnika, in tudi ne, da bodo prišli v enakem zaporedju, kot so bili poslani. Lahko se tudi zgodi, da se paketi podvojijo. Potek prenosa paketov prek UDP-ja je prikazan na sliki 9.4.



Slika 9.4: Potek prenosa paketov prek UDP-ja [**18**]

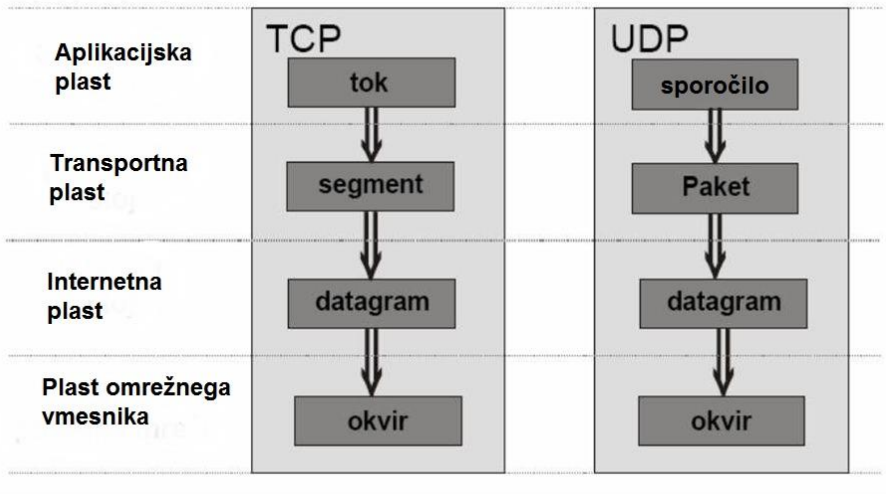
Ker UDP ne zagotavlja dostave podatkov in ne skrbi za kontrolo prenosa, je precej enostavnejši kot TCP. Ne samo, da je glava UDP-ja manjša v primerjavi z glavo TCP-ja, prav tako nima ločenih kontrolnih sporočil, kot recimo za vzpostavitev in prekinitev povezave. UDP-povezava sestoji samo iz dveh sporočil, iz zahteve in iz odgovora. Zato morajo biti podatki dovolj majhni, da gredo v eno sporočilo.

Glava paketa se sestoji iz štirih delov. Polji izvorna vrata in naslovna vrata v glavi UDP-ja in TCP-ja opravljata isto funkcijo, določata protokol na aplikacijskem sloju. Številki vrat, ciljna in izvorna, sta 16-bitni. Polje za dolžino določa, koliko podatkov se prenaša v sporočilu paketa UDP. Zadnje polje je tudi 16-bitno in v njem se nahaja kontrolna vsota. Standard protokola UDP ne predpisuje obvezne uporabe kontrolne vsote. Če se ne uporabi, se polje kontrolne vsote zapolni z ničlami.

### Aplikacijska plast

Na vrhu modela TCP/IP je aplikacijska plast. To je plast, kjer imajo aplikacije dostop do omrežja. Obstaja veliko standardnih orodij TCP/IP in storitev, kot so splet, e-pošta, FTP, Telnet, SNMP, DNS.

Vsaka plast ima svojo strukturo podatkov in terminologijo, ki opisuje to strukturo. Na aplikacijski plasti TCP-protokol za podatke uporablja naziv "**tok**" (angl. stream), za UDP-protokol pa naziv "**sporočilo**" (angl. message). TCP na transportni plasti imenuje podatke "**segment**", UDP pa "**paket**". Na internetni plasti so vsi podatki imenovani "**datagram**", a na plasti omrežnega vmesnika "**okvir** " kot prikazuje slika 9.5.




Slika 9.5: Struktura podatkov pri TCP- in UDP-protokolih [**17**]

## 9.2 Priloga B: Program v programskem jeziku C++

```
#include "falcon/core/FalconDevice.h"
#include "falcon/firmware/FalconFirmwareNovintSDK.h"
#include "falcon/util/FalconFirmwareBinaryNvent.h"
#include "falcon/kinematic/FalconKinematicStamper.h"

/*
FalconFirmwareBinaryNvent - razred vsebuje firmware potreben za delovanje haptične naprave

FalconKinematicStamper - razred vsebuje metode za pretvarjanje vrednosti  nog Falcona v 3D prostor


      getForces(const boost::array<double, 3> (&position), const boost::array<double, 3> (&cart_force),
boost::array<int, 3> (&enc_force)) - metoda vrne sile za posamezne noge haptične naprave (enc_force), ki so
potrebne, da prijemalo premakneme iz določenega položaja (position) za določen vektor (cart_force)

      getPosition(boost::array<int, 3> (&angles), boost::array<double, 3> (&position)) - metoda vrne položaj
prijemala (position) v 3D prostoru glede na znane vrednosti nog (angles)
*/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <csignal>
#include "stdint.h"

#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<sys/socket.h>

#include<time.h>
#include <sys/time.h>


#define SERVER "192.168.1.1"  //definiramo IP adreso za pošiljanje paketov
#define BUFLEN 512  //definiramo veličino paketov za slanje, 512 Kbit-ov
#define PORT 5556   //definiramo port za slanje paketov preko protokola UDP


//na začetku nastavimo neko minimalno silo na haptični napravi
#define MIN_FORCE 5

using namespace libnifalcon;

//realno stevilo zapisano z IEEE 754 standardom pretvorimo v celo stevilo tako, da namesto njegove
vrednosti ohranimo zaporednje bitov

//primer:

      //0.5 zapisemo po IEEE 754 standardu kot 00111111000000000000000000000000(2), kar nato
pretvorimo v desetiski sistem 1056964608(10)
```

```
int toArg(float val){
	return *((int*)((void*)&val));
}


int main(void)
{
	struct sockaddr_in si_other;
	int s, i, slen=sizeof(si_other);

	//ustvarimo buffer za pošiljanje paketov preko UDP protokola
	char message[BUFLEN];

	unsigned int st=1;

	float rotation = 0;
	unsigned char planeStatus = 0;
	unsigned char buttonState = 0;

	boost::array<double, 3> force;
	boost::array<int, 3> forceFinal;

	//Novint Falcon ima svoje središče na položaju (0, 0, 0.125)
	boost::array<double, 3> center, center2;
	center[0] = center[1] = 0;
	center[2] = .125;

	//podatke bomo na kvadrokopter pošljali največ 10x na sekudno, zato ustvarimo strukturo, ki drži
	//podatke o času zadnjega pošiljanja
	timeval zadnje;
	gettimeofday(&zadnje, NULL);

	char emergencyShift = 0;

	//definiramo objekte potrebne za komunikacijo s Novint Falcon-om
	boost::shared_ptr<FalconFirmware> f;
	FalconDevice dev;
	FalconKinematicStamper fks(1);

	//prerimo, ali je haptična naprava priključena,oz. zaznana
	//poskusimo odpreti komunikacijo s  haptično napravo
	if(!dev.open(0))
	{
		std::cout << "Ne morem odpreti Falcon-a - Napaka: " << std::endl;
		return -1;
	}
	std::cout << "Odpiranje Falcon-a" << std::endl;

	//nastavimo firmware, ki ga bomo uporabljali za komunikacijo s haptično napravo
	dev.setFalconFirmware<FalconFirmwareNovintSDK>();

	f = dev.getFalconFirmware();
```

```
	//poskusamo naložiti firmware
	if(!dev.isFirmwareLoaded())
	{
		std::cout << "Nalaganje firmware-a" << std::endl;
		for(int i = 0; i < 10; ++i)
		{
			if(!dev.getFalconFirmware()->loadFirmware(true,
NOVINT_FALCON_NVENT_FIRMWARE_SIZE,
const_cast<uint8_t*>(NOVINT_FALCON_NVENT_FIRMWARE)))
			{
				std::cout << "Ne morem naložiti firmware-a" << std::endl;

				dev.close();
				if(!dev.open(0))
				{
					std::cout << "Ne morem odpreti Falcon-a - Napaka: 2" << std::endl;
					return -2;
				}
			}
			else
			{
				std::cout <<"Firmware naložen" << std::endl;
				break;
			}
		}
		if(!dev.isFirmwareLoaded())
		{
			std::cout << "Firmware se ni naložil pravilno. Poskusi zagnati droneControl ponovno !" <<
std::endl;
			return -3;
		}
	}

	//ustvarimo vtičnico za pošiljanje paketov po protokolu UDP
	if ( (s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
	{
		perror("socket");
		exit(1);
	}

	//nastavimo vtičnico za pošiljanje paketov
	memset((char *) &si_other, 0, sizeof(si_other));
	si_other.sin_family = AF_INET;
	si_other.sin_port = htons(PORT);

	if (inet_aton(SERVER , &si_other.sin_addr) == 0)
	{
		fprintf(stderr, "inet_aton() failed\n");
		exit(1);
	}

	//nastavimo začetne sile za haptično napravo
	forceFinal[0] = MIN_FORCE;
	forceFinal[1] = MIN_FORCE;
	forceFinal[2] = MIN_FORCE;
```

```
f->setForces(forceFinal);

boost::array<double,3> pos2;

//prižgemo LED na haptični napravi
dev.getFalconFirmware()->setLEDStatus(FalconFirmware::RED_LED |
FalconFirmware::GREEN_LED | FalconFirmware::BLUE_LED);
dev.runIOLoop();

//haptična naprava kalibrira svoj položaj
int spremembe = 4;
int lastPos = -1;

std::cout<<"Kalibracija: prosim premikajte napravo naprej in nazaj"<<std::endl;

while(spremembe){
	if(dev.runIOLoop()) ++i;
	else continue;

	//metoda getEncoderValues vrne trenutne vrednosti nog na haptični napravi
	//s metodo getPosition pa dobimo položaj prijemala v (x, y, z) koordinatnem sistemu
	boost::array<int,3> encoded = f->getEncoderValues();
	fks.getPosition(encoded, pos2);

	if(pos2[2] - center[2] < 0){
		if(lastPos > 0){
			--spremembe;
			lastPos = -1;
		}
	}
	else if(lastPos < 0){
		--spremembe;
		lastPos = 1;
	}
	//sprostimo procesor
	sched_yield();
}

//kalibriramo središče haptične naprave
std::cout<<"Prosim spustite napravo"<<std::endl;

int currTime = time(0);

//čakamo 3s
while(time(0) - currTime < 3){
	dev.runIOLoop();

	sched_yield();
}

std::cout<<"Kalibriram napravo ..."<<std::endl;
float napaka = 1000;
```

```
int last = time(0);

//dokler se prijemalo premika išsemo novo središce haptične naprave
do{
	//če haptična naprava ni pripravljena za dobivanje in pošiljanje podatkov čakamo, sicer povečamo števec
	if(dev.runIOLoop()) ++i;
	else continue;

	boost::array<int,3> encoded = f->getEncoderValues();
	fks.getPosition(encoded, pos2);


	//izračunamo vektor od centra do trenutnega položaja prijemala

	//prijemalo želimo premakniti v center, tako da uporabimo silo, ki kaže v enaki smeri, kot izračunan vektor
	force[0] = (pos2[0] - center[0]) * 75;
	force[1] = (pos2[1] - center[1]) * 75 - .6;
	force[2] = (pos2[2] - center[2]) * 75;

	//izračunamo razdaljo med trenutno pozicijo in središčem
	float currNapaka = abs(pos2[0] - center[0]) + abs(pos2[1] - center[1]) + abs(pos2[2] - center[2]);

	//metoda getForces izračuna silo posameznih nog, ki je potrebna, da prestavimo prijemalo za prej izračunano silo (proti središču)
	//metoda vrne true, v kolikor je uspela izračunati sile
	forceFinal[0] = forceFinal[1] = forceFinal[2] = 0;

	//ker smo uporabili vektor od centra proti trenutnem položaju,  je potrebno dobljene sile negirati
	if(fks.getForces(pos2, force,-(forceFinal))){
		forceFinal[0] = forceFinal[0] ;
		forceFinal[1] = forceFinal[1] ;
		forceFinal[2] = forceFinal[2] ;
		f->setForces(forceFinal);
	}

	//če se je prijemalo premaknilo za dovolj veliko vrednost (vec od .1) , upoštevamo, da je prišlo do premika, v nasprotnem primeru upoštevamo, kot da miruje
	if(napaka - currNapaka > .1f || napaka - currNapaka < -.1f){
		napaka = currNapaka;
		last = time(0);
	}

	sched_yield();
}

//kalibracija se izvaja, dokler napaka ni dovolj majhna, ali dokler prijamalo ne miruje za vsaj 3s
while(napaka > .5 || (time(0) - last) < 3);

std::cout<<"Kalibracija naprave uspešno zakljucena !"<<std::endl;

//nastavimo nove centre
center2[0] = pos2[0];
center2[1] = pos2[1];
```

```
center2[2] = pos2[2];

std::cout<<"Kalibracija kvadkopterja..."<<std::endl;

memset(message,'\0', BUFLEN);
sprintf(message, "AT*FTRIM=%d\r",st++);

std::cout<<message<<std::endl;

//ukaz pošljemo preko protokola UDP na kvadrokopter
if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other, slen)==-1)
{
	perror("sendto()");
	exit(1);
}
std::cout<<"Kalibracija kvadkopterja uspesno zaključena!"<<std::endl;

while(1)
{
	//če naprava ni pripravljena za dobivanje in pošiljanje podatkov čakamo, sicer povečamo števec
	if(!dev.runIOLoop()){
		sched_yield();
		continue;
	}

	//ob pritisku na okrogel gumb pošlemo signal za spremembo stanja emergency
	if((*(f->getGripInfo()) & 4)){
		//signal za spremembo stanja pošlemo samo 1x na pritisk gumba
		if(!emergencyShift){
			emergencyShift = 1;

			memset(message,'\0', BUFLEN);
			sprintf(message,"AT*REF=%d,%d\r", st++, 1 << 8);

			std::cout<<message<<std::endl;

			//ukaz pošljemo preko protokola UDP na kvadrokopter
			if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other, slen)==-1)
			{
				perror("sendto()");
				exit(1);
			}
		}

	}
	else
		emergencyShift = 0;

	//dobimo položaj osi s naprave in ga pretvorimo v realen 3D koordinatni sistem
	boost::array<double,3> pos;
	boost::array<int,3> encoded = f->getEncoderValues();
	fks.getPosition(encoded, pos);
	fks.getPosition(encoded, pos2);
```

```
//vrednosti položaja prijemala skaliramo
	pos[0] = (pos[0] - center2[0]) * 16.6666;
	pos[1] = (pos[1] - center2[1]) * 16.6666;
	pos[2] = (pos[2] - center2[2]) * 20;

//postavimo pogoje, da vrednosti položaja prijemala ne grejo pod 1,0 in nad 1,0
	if(pos[0] > 1.0)
		pos[0] = 1.0;
	else if(pos[0] < -1.0)
		pos[0] = -1.0;

	if(pos[1] > 1.0)
		pos[1] = 1.0;
	else if(pos[1] < -1.0)
		pos[1] = -1.0;

	if(pos[2] > 1.0)
		pos[2] = 1.0;
	else if(pos[2] < -1.0)
		pos[2] = -1.0;

	force[0] = (pos2[0] - center2[0]) * 75;
	force[1] = (pos2[1] - center2[1]) * 75 - .6;
	force[2] = (pos2[2] - center2[2]) * 75;

	forceFinal[0] = forceFinal[1] = forceFinal[2] = 0;
	if(fks.getForces(pos2, force,-(forceFinal))){
		forceFinal[0] = forceFinal[0] ;
		forceFinal[1] = forceFinal[1] ;
		forceFinal[2] = forceFinal[2] ;
		f->setForces(forceFinal);
	}

	timeval curr;
	gettimeofday(&curr, NULL);

	//podatke posilajmo največ 10x na sekundo
	//struktura timeval hrani v lastnosti tv_secčcas v sekundah, v tv_usec pa v mikrosekundah
	if(((curr.tv_sec * 1000 + curr.tv_usec / 1000) - (zadnje.tv_sec * 1000 + zadnje.tv_usec / 1000)) >=
	100.0f){
		zadnje = curr;

		//če je kvadrokopter pristal
		if(!planeStatus){
			//preverimo, ali je pritisnjen gumb za vzlet, če je posljemo ukaz za vzlet, drugače
			pošljemo ukaz za pristanek, da se izognemo izgubi signala s računalnikom
			unsigned int val = ((*(f->getGripInfo()) & 2) > 0);
			memset(message,'\0', BUFLEN);
			sprintf(message, "AT*REF=%d,%d\r", st++, (val | buttonState) << 9);

			std::cout<<message<<std::endl;
```

```
        //ukaz pošljemo preko protokola UDP na kvadrokopter
        if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other, slen)==-1)
        {
                perror("sendto()");
                exit(1);
        }



        //v kolikor je bil pritisnjen gumb za vzlet, si to zabeležimo, če pa je bil pritisnjen v
        prejšnji itteraciji, v trenutni pa ne, pa si zabeležimo, da je kvadrokopter vzletel in je sedaj
        v stanju letenja
        if(val)
                buttonState = 1;
        else if(buttonState){
                buttonState = 0;
                planeStatus = 1;
        //ukaz za prižrig LED diod na kvadrokopterju (utripanje z oranžno barvo)
                memset(message,'\0', BUFLEN);
                sprintf(message,"AT*LED=%d,%d,%d,%d\r",st++,3,toArg(5),0);

                std::cout<<message<<std::endl;

                // ukaz pošljemo preko protokola UDP na kvadrokopter
                if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other, slen)==-1)
                {
                        perror("sendto()");
                        exit(1);
                }

        }
}
//če je kvadrokopter v stanju letenja
else
{
        //preverimo, ali je pritisnjen gumb za spust, v kolikor je pošljemo kvadrokopterju ukaz za
        spust
        if(*(f->getGripInfo()) & 2){
                memset(message,'\0', BUFLEN);
                sprintf(message, "%s%d,%d\r", "AT*REF=", st++, 0);

                std::cout<<message<<std::endl;


                //ukaz pošljemo preko protokola UDP na kvadrokopter
                if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other, slen)==-1)
                {
                        perror("sendto()");
                        exit(1);
                }


                buttonState = 1;
```

```
        //ukaz za prižrig LED diod na kvadrokopterju (zelena barva)
        memset(message,'\0', BUFLEN);
        sprintf(message,"AT*LED=%d,%d,%d,%d\r",st++,8,toArg(1),0);

        std::cout<<message<<std::endl;

        //ukaz pošljemo preko protokola UDP na kvadrokopter
        if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other, slen)==-1)
        {
                perror("sendto()");
                exit(1);
        }

}



//v kolikor gumb za spust ni pritisnjen...
else{
//preverimo, ali je bil pritisnjen v prejšnji itteraciji (je bil pravkar spuščen), potem je
kvadrokopter pristal
        if(buttonState){
                buttonState = 0;
                planeStatus = 0;
        }
        //v kolikor gumb za pristajanje ni bil pravkar spuščen...
        else{
                //dobimo podatke s gumbov za levo ali desno rotacijo
                if(*(f->getGripInfo()) & 1)
                        rotation = 1;
                else if(*(f->getGripInfo()) & 8)
                        rotation = -1;
                else
                        rotation = 0;

                //zaokroževanje na 1 decimalno mesto
                for(int i = 0; i < 3; ++i){
                        int tmp = pos[i] * 10;
                        pos[i] = pos[i] - (pos[i] - (tmp / 10.0f));
                }



                //pošljemo dobljene podatke preko protokola UDP na kvadrokopter
                memset(message,'\0', BUFLEN);
                sprintf(message, "AT*PCMD=%d,%d,%d,%d,%d,%d\r",st++, 1,
                toArg(pos[0]), toArg(pos[2]), toArg(pos[1]), toArg(rotation));

                std::cout<<message<<std::endl;

                //ukaz pošljemo preko protokola UDP na kvadrokopter
                if (sendto(s, message,strlen(message) , 0 , (struct sockaddr *) &si_other,
                slen)==-1)
                {
```

```
                                perror("sendto()");
                                exit(1);
                        }



                                }
                        }
                }
        }
        sched_yield();

    }

    close(s);

    return 0;
}
```

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

# IZJAVA O AVTORSTVU

Spodaj podpisani/-a Mihael Flac

z vpisno številko E1060324

sem avtor/-ica diplomskega dela z naslovom: Vodenje kvadrokopterja s haptičnim vmesnikom Novint Falcon

*(naslov diplomskega dela)*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek) izr. prof. dr. Aleš Hace
  in somentorstvom (naziv, ime in priimek)
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela.
- soglašam z javno objavo elektronske oblike diplomskega dela v DKUM.

V Mariboru, dne 10. 09. 2014

Podpis avtorja/-ice:
Flac M.

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

# IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

*Podpisani mentor :*

izr. prof. dr. Aleš Hace

*(ime in priimek mentorja)*

*in somentor (eden ali več, če obstajata):*

_____

*(ime in priimek somentorja)*

*Izjavljam (-va), da je študent*

*Ime in priimek:* Mihael Flac

*Vpisna številka:* E1060324

*Na programu:* Elektrotehnika

*izdelal zaključno delo z naslovom:*

Vodenje kvadrokopterja s haptičnim vmesnikom Novint Falcon

*(naslov zaključnega dela v slovenskem in angleškem jeziku)*

Quadcopter control by haptic interface Novint Falcon

v skladu z odobreno temo zaključnega dela, Navodilih o pripravi zaključnih del in mojimi (najinimi oziroma našimi) navodili.

Preveril (-a, -i) in pregledal (-a, -i) sem (sva, smo) poročilo o plagiatorstvu.

Datum in kraj: 10. 09. 2014, Maribor

Podpis mentorja:

Datum in kraj:

Podpis somentorja (če obstaja):

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Smetanova ulica 17
2000 Maribor, Slovenija

FERI

**IZJAVA O ISTOVETNOSTI TISKANE IN ELEKTRONSKE VERZIJE ZAKLJUČNEGA DELA IN OBJAVI OSEBNIH PODATKOV DIPLOMANTOV**

Ime in priimek avtorja-ice: Mihael Flac

Vpisna številka: E1060324

Študijski program: Elektrotehnika

Naslov zaključnega dela: Vodenje kvadrokopterja s haptičnim vmesnikom Novint Falcon

Mentor: izr. prof. dr. Aleš Hace

Somentor:

Podpisani-a Mihael Flac izjavljam, da sem za potrebe arhiviranja oddal elektronsko verzijo zaključnega dela v Digitalno knjižnico Univerze v Mariboru. Zaključno delo sem izdelal-a sam-a ob pomoči mentorja. V skladu s 1. odstavkom 21. člena Zakona o avtorskih in sorodnih pravicah dovoljujem, da se zgoraj navedeno zaključno delo objavi na portalu Digitalne knjižnice Univerze v Mariboru.

Tiskana verzija zaključnega dela je istovetna z elektronsko verzijo elektronski verziji, ki sem jo oddal za objavo v Digitalno knjižnico Univerze v Mariboru.

---

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, varstva industrijske lastnine ali tajnosti podatkov naročnika: _____ ne sme biti javno dostopno do _____(datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela).

---

Podpisani izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto in kraj rojstva, datum zaključka študija, naslov zaključnega dela), na spletnih straneh in v publikacijah UM.

Datum in kraj: 10. 09. 2014, Maribor Podpis avtorja-ice: Flac M.

Podpis mentorja: _____
*(samo v primeru, če delo ne sme biti javno dostopno)*

Podpis odgovorne osebe naročnika in žig:
*(samo v primeru, če delo ne sme biti javno dostopno)* _____