Razvoj programske podrške za Raspberry Pi mobilnu robotsku platformu uz korištenje ROS operativnog sustava

Miličić, Stjepan

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:882777

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij Robotika i Umjetna inteligencija

RAZVOJ PROGRAMSKE PODRŠKE ZA RASPBERRY PI MOBILNU ROBOTSKU PLATFORMU UZ KORIŠTENJE ROS OPERATIVNOG SUSTAVA

Diplomski rad

Stjepan Miličić

Osijek, 2022.

SADRŽAJ

1. UVOD

Robotika svoju primjenu pronalazi u gotovo svim sferama ljudske djelatnosti. Jedan od razloga eksponencijalnog širenja znanstvene discipline je činjenica kako se na tržištu nude mnogobrojna pristupačna i praktična robotska rješenja nastala kombinacijom mikrokontrolera ili manjih računala, robotskih platformi obogaćenih senzorima te okvira (engl. *framework*) kojima je svrha ponuditi razvojne alate i podršku za integraciju programskog rješenja na robota.

Raspberry Pi je projekt nastao kako bi omogućio globalnu informatičku pismenost preko svojih malenih i pristupačnih računala s gotovo svim karakteristikama nužnim za rad i funkcionalnost jednog osobnog računala. Noviji modeli dolaze s boljim hardverskim specifikacijama i brojnijim funkcionalnostima, a GPIO (General-Purpose Input/Output) pinovi omogućuju dodavanje novih komponenti i perifernih uređaja za potrebe projekta. Računalo se može oplemeniti senzorima i programskom podrškom za kontrolu i komunikaciju istih, stoga Raspberry Pi računalo može predstavljati polazišnu točku u izradi projekata u robotici.

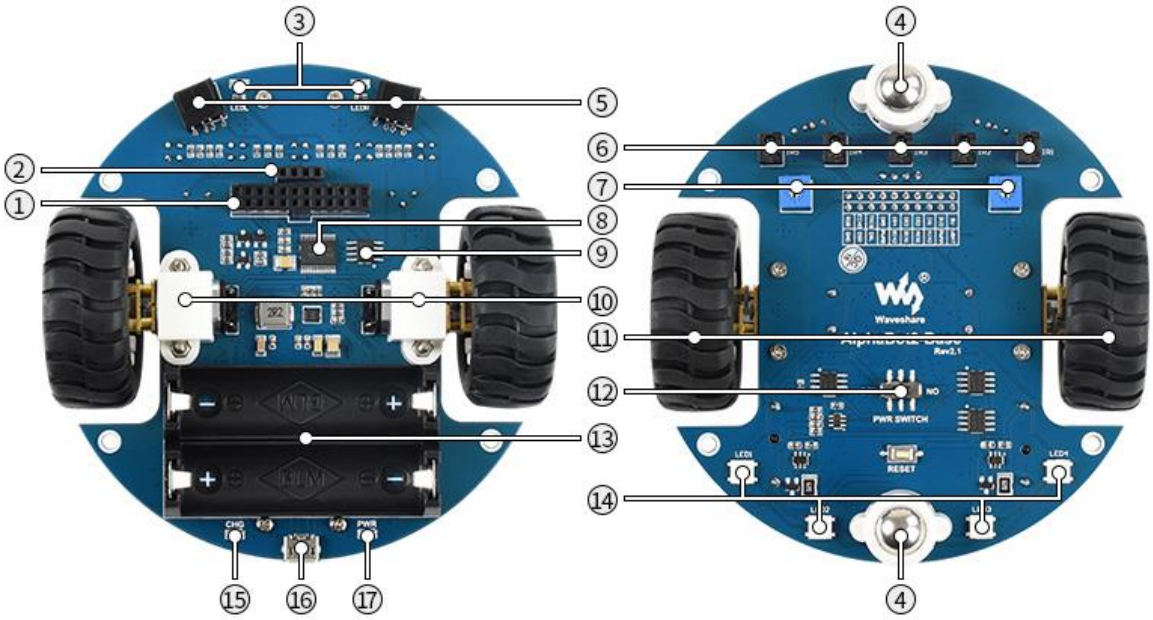
AlphaBot2-Pi gotovo je rješenje mobilne robotske platforme temeljeno na Raspberry Pi 3 računalu. Implementiranjem programskog rješenja za prikladan i efikasan rad motora i senzora, platforma se može osposobiti za kontrolirano i autonomno kretanje te prilagoditi potrebama projekta, odnosno zadatka.

ROS (Robot Operating System), odnosno Operacijski Sustav Robota je međuprogramska oprema (engl. *middleware*) za robotiku otvorenog koda (engl. *open-source*), odnosno zbirka okvira, alata i biblioteka namijenjenih olakšavanju razvoja softvera za robote. Trenutno je jedan od najboljih i najkorištenijih alata u području robotike te predstavlja standard za daljnja istraživanja i razvoj discipline. Projekti izrađeni u ROS-u mogu se kretati od manjih, individualnih sve do globalnih industrijskih projekata i kolaboracija.

2. ALPHABOT-2 PI

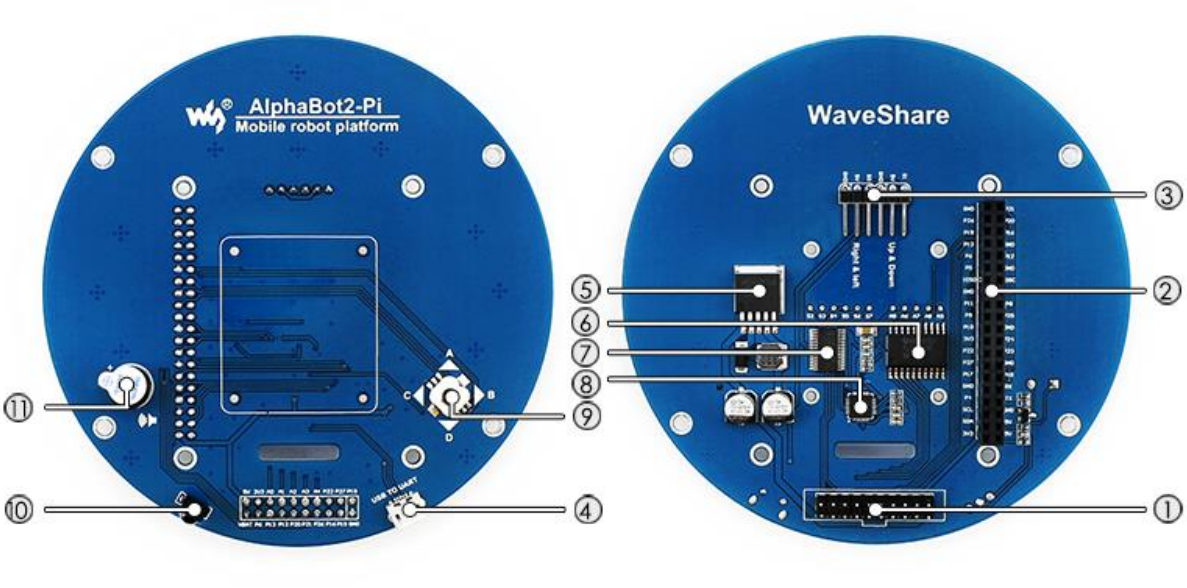
AlphaBot2-Pi mobilna je robotska platforma temeljena na Raspberry Pi 3 Model B računalu koje se nalazi unutar zaštitnog kućišta. Platforma se kreće pomoću ugrađenih motora i kotačića te omogućuje implementaciju autonomnog kretanja pomoću senzora od kojih će se, u svrhu izrade praktičnog dijela zadatka, koristiti ultrazvučni senzor HC-SR04 te infracrveni senzor ST188. Prije razrade programskog rješenja potrebno je upoznati se s karakteristikama računala i senzora.

Slika 2.1 AlphaBot2-Pi



Slika 2.2 AlphaBot2-Pi šasija

1. AlphaBot2 kontrolno sučelje
2. Sučelje ultrazvučnog modula
3. Indikatori prepreka
4. Višesmjerni kotačići
5. ST188: reflektivni infracrveni fotoelektrični senzor za izbjegavanje prepreka
6. ITR20001/T: reflektivni infracrveni fotoelektrični senzor za praćenje linije
7. Potenciometar za podešavanje dometa za izbjegavanje prepreka
8. TB6612FNG dvojni H-bridge upravljač motora
9. LM393 uspoređivač napona
10. N20 motor s mikrozupčanicima
11. Gumeni kotačiči promjera 42 mm i širine 19 mm
12. Prekidač za napajanje
13. Kontejner za baterije
14. WS2812B RGB LED diode
15. Indikator za punjenje baterije
16. 5V USB priključak za napajanje
17. Indikator napajanja



Slika 1.3 AlphaBot2-Pi adapterska ploča za Raspberry Pi računalo

1. AlphaBot2 kontrolno sučelje za povezivanje AlphaBot2 šasije
2. Raspberry Pi sučelje za povezivanje Raspberry Pi 3 Model B računala
3. Servo sučelje
4. USB u UART priključak
5. LM2596: 5V regulator napona
6. TLC1543: 10-bitni pretvarač signala, omogućuje korištenje analognih senzora
7. PCA9685: servo upravljač
8. CP2102: USB u UART pretvarač
9. Upravljačka ručica
10. Infracrveni prijemnik
11. Zujalica

2.1 Raspberry Pi 3 Model B

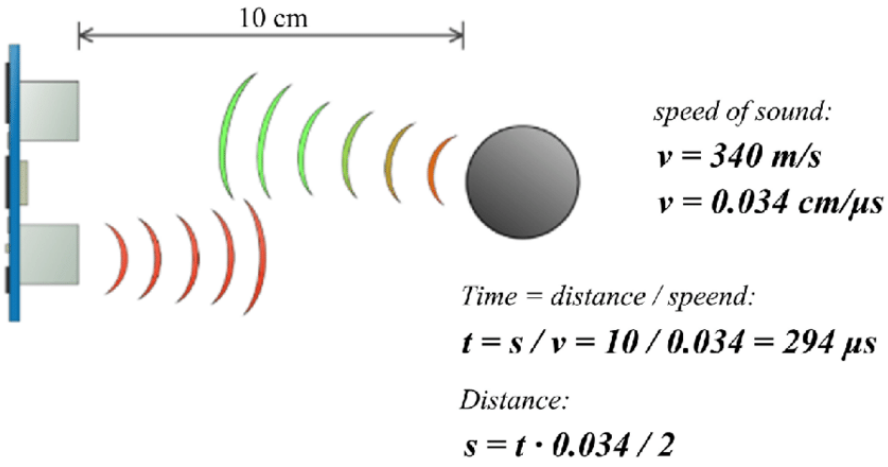
Raspberry Pi serija je malih računala na pločici napravljena od strane Raspberry Pi Foundation-a, humanitarne organizacije iz Ujedinjenog Kraljevstva kojoj je glavni cilj obrazovati i informatički opismeniti ljude na globalnoj razini. Korisnici diljem svijeta pomoću Raspberry Pi računala uče programirati ili izrađuju samostalne projekte u području robotike i automatizacije. Raspberry Pi 3 Model B najstariji je model treće generacije Raspberry Pi računala. Na pločici dimenzija 87x58 mm nalazi se četverojezgreni Broadcom BCM2837 64-bitni procesor te 1 GB radne memorije. Omogućuje Bluetooth i WiFi povezivost, sadrži 4 USB2 priključka, LAN priključak, HDMI priključak te 40 GPIO (General-Purpose Input/Output) pinova za spajanje perifernih uređaja, komponenti ili senzora na računalo. Pokreće se na Linuxu te ima vlastiti standardizirani operacijski sustav Raspberry Pi OS (prethodno se zvao Raspbian), temeljen na Debianu.

Slika 2.2 Raspberry PI 3 Model B računalo

2.2 Ultrazvučni senzor HC-SR04

HC-SR04 ultrazvučni senzor koristi ultrazvučne valove kako bi odredio udaljenost od predmeta. Karakteriziraju ga dobre tehničke mogućnosti, ne zbunjuju ga boja ili transparentnost objekta, može se koristiti u mračnom okruženju te precizno računa udaljenost od objekta zbog svog načina djelovanja. Princip rada zasniva se na dva osnovna dijela ovog modula, to su prekidač (TRIG) i refleksija (ECHO). Udaljenost objekta ultrazvučni senzor računa preko formule gdje se kao parametri uzimaju brzina zvuka (v = 340 m/s) i vrijeme refleksije vala. Impuls se šalje na TRIG pin, senzor odašilja signal te se ECHO pin postavlja u stanje HIGH, pokreće se mjerač vremena. Kada se odašiljani signal reflektira nazad na mikrofon, ECHO pin mijenja svoje stanje u LOW i mjerač vremena se zaustavlja. S obzirom na to kako je udaljenost polovina prijeđenog puta signala, vrijednost dijelimo s 2 (Slika 2.3).

Slika 2.2 HC-SR04 ultrazvučni senzor



Slika 2.3 Princip rada ultrazvučnog senzora

2.3 Infracrveni senzor ST188

ST188 infracrveni senzor radi na principu samorefleksije. Senzor se sastoji od dva dijela: infracrvenog odašiljača i infracrvenog prijemnika. Infracrveni odašiljač izrađen je od LED (Light-emitting-diode) diode koja djeluje kao luminofor, odnosno zaprimljenu energiju emitira kao elektromagnetsko zračenje te p-n spoja kojeg karakterizira visoka efikasnost pri emitiranju infracrvene radijacije. Kada se struja pošalje na p-n spoj, odašiljač generira infracrvenu svjetlost proporcionalnu snazi zaprimljene struje. Prijemnik detektira reflektiranu svjetlost te reagira pomoću signalnog indikatora na modulu. Prijemnik se sastoji od visokoosjetljivog Darlingtonovog fototranzistora, odnosno poluvodičkog uređaja koji detektira infracrvenu svjetlost i pretvara je u električni signal. Senzor može odašiljati svjetlost u rasponu od 4 do 13 milimetara što mu daje prednost u uočavanju i reagiranju na prepreku koja se nalazi neposredno u blizini mobilne robotske platforme.

Slika 2.4 ST188 infracrveni senzor

3. ROS

ROS (Robot Operating System) ili Operacijski Sustav Robota je međusoftver za robotiku, odnosno zbirka okvira namijenjena olakšavanju razvoja softvera za robote. Smatra se jednim od najboljih i najkorištenijih alata u području robotike, implementiran je u mnogobrojnim industrijskim rješenjima te predstavlja normu u učenju i daljnjem razvoju robotike. Prednost u usporedbi s ostalim okvirima za robotiku mu je mogućnost primjene gotovih alata, algoritama i programske podrške na robote, neovisno o njihovoj hardverskoj ili softverskoj različitosti. Osim toga, ROS je globalna zajednica inženjera koji međusobno pridonose razvoju robotike i samog okvira te je oko njega izgrađen snažan ekosustav softverske podrške i održavanja iste od strane razvojnih inženjera. Napisan je u C++-u, Pythonu i Lispu te je otvorenog koda (engl. *open-source*). Iako ne ispunjava sve kriterije pravog operacijskog sustava, ROS pruža sloj apstrakcije i rukovanja hardverom, kontrolu niskorazinskih uređaja (engl. *low-level device control*), izmjenjivanje podataka između procesa i rukovanje paketima. Također, sadrži alate i biblioteke za održavanje, izgradnju, pisanje, čitanje i pokretanje programskog koda na više računala. Usmjeren je na UNIX sustave, poput Linuxa, a primarna platforma je Ubuntu operacijski sustav zbog svoje fleksibilnosti i prilagođenosti korisniku. Osnovna zadaća ROS-a je omogućiti korisniku dizajniranje složenog i efikasnog softvera bez potrebe za detaljnim razumijevanjem principa rada hardverskog sklopovlja. On omogućuje uspješnu komunikaciju cjelokupnog sustava komponenti i senzora za efikasan i uspješan rad robota. Strukturiran je u obliku manjih računalnih programa koji međusobno komuniciraju i zajedno čine sustav kontrole robota.

Ukratko, ROS je kolekcija programskih biblioteka i alata pomoću kojih možemo izgraditi aplikacije za robote. Okvir sadrži gotovo kompletnu podršku za projekte u robotici, neovisno o različitosti i zahtjevima pojedinog robota. Također nudi mnoge alate za 2D i 3D simulaciju koji, u kombinaciji s gotovim ponuđenim rješenjima za razne industrijske robote, čine ROS platformom za učenje robotike i umjetne inteligencije. U usporedbi s ostalim okvirima za robotiku, ROS-ov primarni cilj je omogućiti višestruko korištenje funkcionalnog programskog koda u istraživanju i razvoju robotskih rješenja. Zbog svoje fleksibilnosti, programski kod napisan u ROS-u može se integrirati i u ostale okvire za robotiku.

3.1 Povijest ROS-a

ROS je projekt koji se odvijao kroz nekoliko faza razvoja. Glavni problem koji je tada bio zastupljen u robotici bili su visoki vremenski zahtjevi pri ponovnoj implementaciji programske podrške unutar istog sustava ili organizacije (engl. *reinventing the wheel*), što je ostavljalo znatno manje vremena za provođenje novih istraživanja i daljnji razvoj robotike. Prisutnost takvih problema doveo je do formiranja istraživačkih zajednica orijentiranih prema cilju razvoja robotike i umjetne inteligencije kao znanstvene discipline te okupljanje stručnjaka u svrhu ispunjavanja industrijskih zahtjeva modernog svijeta i optimiziranja okvira i razvojnih alata u robotici.

3.1.1 Stanford faza razvoja

ROS je započeo kao osobni projekt Keenana Wyrobeka i Erica Bergera u pokušaju pronalaska rješenja za navedene probleme: previše vremena odlazilo je na ponovnu implementaciju programske podrške za aktuatore i senzore unutar istog robotskog sustava ili organizacije, premalo vremena odlazilo je na nova istraživanja i razvoj softverskih rješenja. 2006. godine započeli su projekt Stanford Personal Robotics Program s ciljem izgradnje okvira koji će omogućiti komunikaciju između procesa u sustavu te sadržavati osnovne alate za razvoj robotskog softvera. Kako bi uključili ostale članove istraživačke zajednice i nastavili s optimiziranjem okvira i programske podrške, izradili su eksperimentalnog robota Personal Robot - PR1 (Slika 3.1) kojeg su donirali sveučilištima diljem SAD-a kao podlogu za provođenje daljnjih istraživanja.

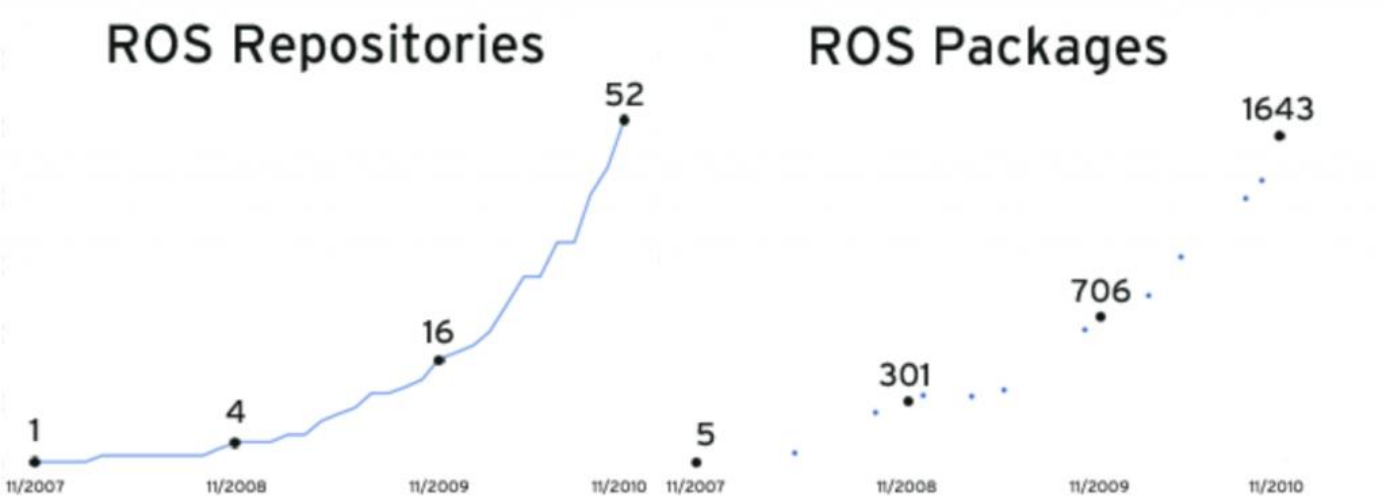
Slika 3.1 Personal Robot (PR1)

3.1.2 Willow Garage faza razvoja

Na Stanfordu su Keenan Wyrobek i Eric Berger dobili 50 000 $ poticaja kojeg su iskoristili u svrhu izgradnje eksperimentalnog robota PR1 i demo verzije projekta. Za provođenje daljnjeg istraživanja, izgradnje univerzalnog sustava i opskrbom istraživača eksperimentalnim robotima, projekt je zahtijevao dodatno financiranje. Počeli su tražiti investitore te su 2008. godine upoznali Scotta Hassana, osnivača tvrtke Willow Garage, istraživačkog centra za robotiku i robotske proizvode. Bio je uvelike zainteresiran za ideju izgradnje univerzalnog robotskog sustava te je odlučio financirati projekt i započeti Personal Robotics Program unutar svoje tvrtke. Nastaje Robotski Operacijski Sustav te eksperimentalni robot Personal Robot 2 - PR2 (Slika 3.2).

Slika 3.2 Personal Robot 2 (PR2)

Razvoj i širenje ROS-a postaje primarni fokus tvrtke te su ostali projekti bili odbačeni. ROS se u sklopu Willow Garage-a razvijao šest godina, dok se tvrtka nije zatvorila 2014. godine. U tom razdoblju, ostvario se značajan napredak projekta. 2009. godine objavljen je ROS Mango Tango (ROS 0.4), prva distribucija ROS-a. Box Turtle (ROS 1.0) objavljen je godinu dana poslije, 2010. godine, te sve sljedeće distribucije dobivaju ime po vrstama kornjača. U međuvremenu, 3D simulacija sustava postaje značajan faktor pri konstrukciji programskog rješenja, stoga razvojni inženjeri uključuju Gazebo, 3D simulator Player/Stage projekta u ROS. Dodavanjem novih funkcionalnosti, repozitorija i paketa, ROS postaje sve rasprostranjeniji po sveučilištima i istraživačkim zajednicama. 2011. godine Willow Garage izdaje novog eksperimentalnog robota pod imenom TurtleBot.



Slika 3.3 Evolucija ROS-a u ranom stadiju

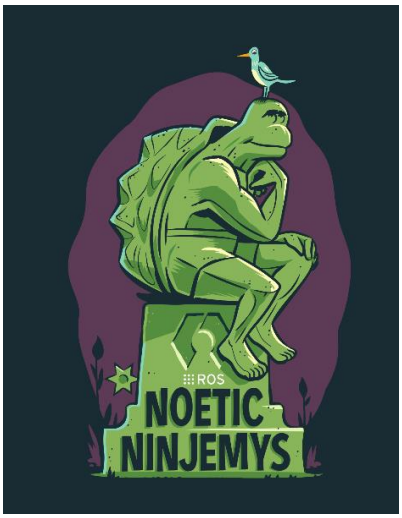
Slika 3.4 TurtleBot

3.1.3 Open Source Robotics Foundation faza razvoja

U sklopu nove tvrtke Open Source Robotics Foundation (OSRF), ROS se nastavio dalje razvijati te su izdane nove distribucije sustava:

- ROS Hydro Medusa (2013.)
- ROS Indigo Igloo (2014.)
- ROS Jade Turtle (2015.)
- ROS Kinetic Kame (2016.)
- ROS Lunar Loggerhead (2017.)
- ROS Melodic Morenia (2018.)
- ROS Noetic Ninjemys (2020.)

ROS Noetic Ninjemys posljednja je distribucija ROS sustava bazirana na Pythonu 3, prethodne distribucije bazirane su na Pythonu 2. S obzirom na to kako je ROS pokazivao određene nedostatke pri integraciji u komercijalne proizvode: jedna točka kvara (engl. *single point of failure*), manjak sigurnosti, nedostatak pravovremene podrške (engl. *real-time support*), morao se optimizirati kako bi se postavio kao industrijski standard. U sklopu OSRF-a razvija se ROS 2.0 s ciljem ispravljanja nedostataka ROS-a 1.0.



Slika 3.5 ROS Noetic Ninjemys, posljednja distribucija ROS-a 1.0

3.2 Koncepti ROS-a

Koncepti ROS-a daju nam bolji uvod u cjelokupnu strukturu ROS sustava. U ovom potpoglavlju upoznat ćemo se s rasporedom direktorija ROS-a, kao i njihovim ulogama, osnovnim elementima upravljanja i načinima uspostavljanja komunikacije između procesa preko tema i poruka kako bi međusobno činili sustav upravljanja robotom. Također ćemo sagledati ROS kao globalnu zajednicu razvojnih inženjera koji platformu koriste za poboljšanje okvira i provođenje daljnjih istraživanja u području robotike i umjetne inteligencije. Robotski operacijski sustav dijelimo na 3 koncepta (*eng. Levels*):

- **Razina sustava datoteka (engl. *Filesystem Level*)**
- **Razina nadzora i upravljanja (engl. *Computation Graph Level*)**
- **Razina zajednice (engl. *Community Level*)**

3.2.1 ROS razina sustava datoteka

ROS razina sustava datoteka odnosi se na raspored direktorija i datoteka u ROS-u, osnovni elementi su:

- **Paketi (engl. *Packages*)**
  Paketi su osnovna jedinica organizacije softvera u ROS-u. Paket može sadržavati ROS izvršne procese (čvorove), biblioteke, skupove podataka, datoteke za konfiguraciju i sve ostalo što se koristi organizirano zajedno.
- **Metapaketi (engl. *Metapackages*)**
  Metapaketi su specijalizirani paketi kojima je glavna uloga predstavljati grupu međuzavisnih paketa.
- **Manifesti paketa (engl. *Package Manifests*)**
  Manifest (*package.xml*) sadrži metapodatke (engl. *metadata*) o paketu, uključujući njegovo ime, verziju, opis, informaciju o licenci, zavisnosti i ostale meta informacije kao što su izvezeni paketi.

- **Repozitoriji (engl. *Repositories*)**

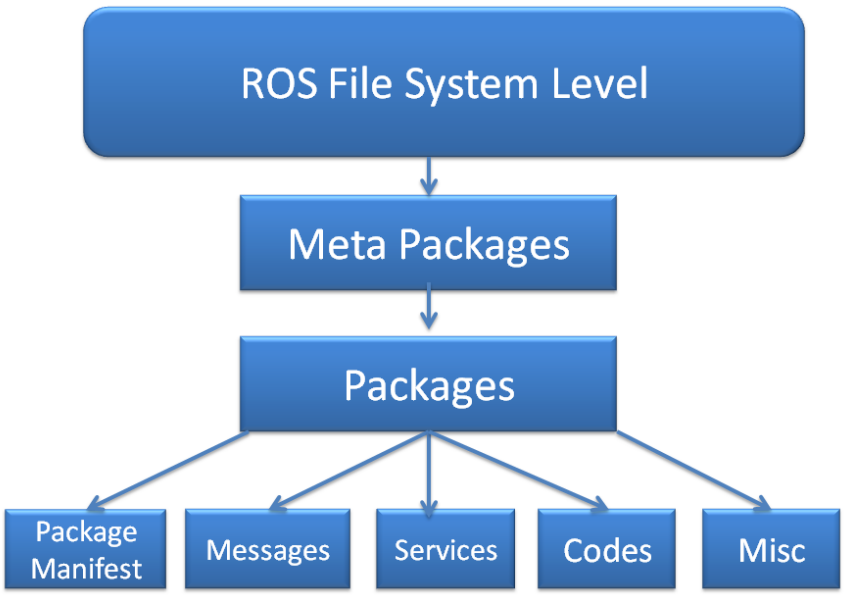
  Kolekcija paketa koji dijele zajednički VCS (Version Control System) sistem. Paketi koji dijele VCS dijele istu verziju i mogu biti objavljeni zajedno.

- **Poruke (engl. *Message types*)**

  Opisi poruka, pohranjeni u *my_package/srv/MyMessageType.msg*, definiraju strukture podataka za poruke koje se izmjenjuju u ROS-u

- **Servis (engl. *Service types*)**

  Opisi servisa, pohranjeni u *my_package/src/MyServiceType.srv*, definiraju strukture podataka za zahtjev (engl. *request*) i odgovor (engl. *response)* za servise u ROS-u.



Slika 3.6 ROS razina sustava datoteka

3.2.2 ROS razina nadzora i upravljanja

ROS razina nadzora i upravljanja je ravnopravna (engl. *peer-to-peer*) mreža procesa koji zajedno obrađuju podatke. Osnovna zadaća nadzora upravljanja je olakšati komunikaciju između čvorova (engl. *nodes*), procesa koji izvršava računanja i koji se mogu izvršiti na jednom ili na više računala. Prednost ovakvog sustava nadzora i upravljanja je što čvorovi mogu kontrolirati jedan aspekt sustava, primjerice pojedini čvor može biti zadužen za prosljeđivanje podataka s jednog od senzora priključenog na robota. Elementi ROS razine nadzora i upravljanja su:

- **Čvorovi (engl. *nodes*)**

  Čvorovi su procesi zaduženi za izračunavanje i distribuciju podataka. ROS je dizajniran modularno, sastoji se od manjih podsustava koji međusobno komuniciraju i izmjenjuju podatke. Robotski sustav upravljanja uobičajeno se sastoji od više čvorova, gdje je svaki zadužen za procesiranje podataka s jednog od senzora ili kontrolu pojedine komponente sustava. Takvu modularnost u dizajnu sustava postižemo pomoću čvorova.
- **Master**

  ROS Master pruža registraciju imena i servisa u sustavu čvorova. On nadzire komunikaciju između čvorova i ostalih komponenti ROS razine za nadzor i upravljanje. Osnovna zadaća Master-a je omogućiti individualnim čvorovima da se međusobno pronađu i uspješno ostvaruju ravnopravnu komunikaciju.
- **Poslužitelj parametara (engl. *Parameter Server*)**

  Poslužitelj parametara omogućuje podacima da se pohranjuju pomoću ključa na centralnoj lokaciji i dio je Master-a.
- **Poruke (engl. *Messages*)**

  Poruka u ROS-u je jednostavna struktura podatka koja se sastoji od polja s tipom podatka te pomoću njih čvorovi međusobno komuniciraju. Standardne strukture podataka (npr. integer, floating point, boolean) su podržane, kao i njihova polja (engl. *arrays*).

- **Teme (engl. *Topics*)**

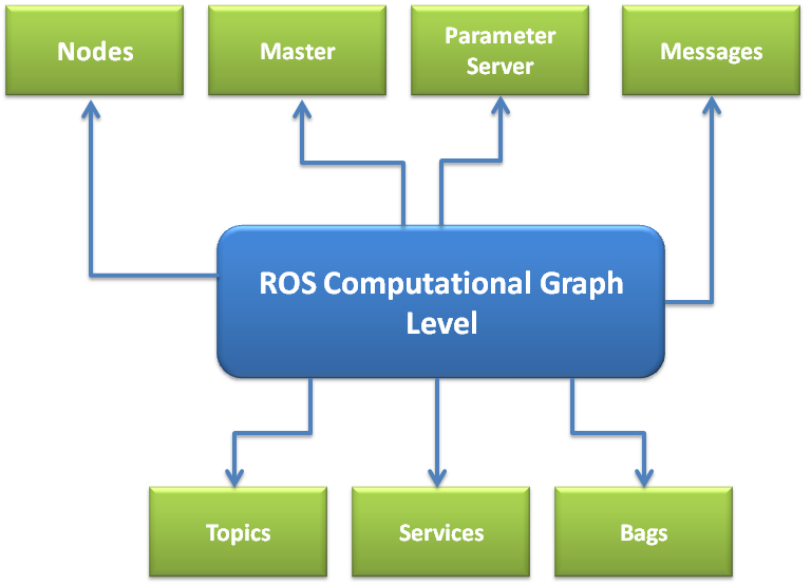
  Poruke u sustavu funkcioniraju na principu objavljivanje/pretplata dinamike. Čvor šalje poruku tako što ju objavi na Temu, odnosno nazivu koji se koristi kako bi se identificirao sadržaj poruke. Čvor koji je zainteresiran za određen skup podataka će se pretplatiti na adekvatan čvor koji iste isporučuje. Svaki čvor može se pretplatiti ili objavljivati na jedan ili više drugih čvorova.

- **Servisi (engl. *Services*)**

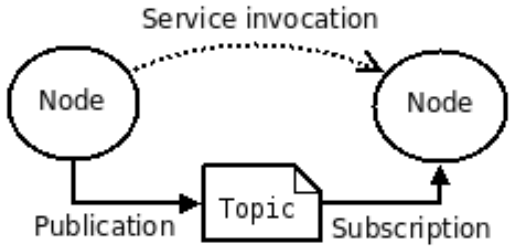
  Servisi su definirani parom struktura poruka, jedna struktura poruka zadužena je za zahtjev (engl. *request*), a druga za odgovor (engl. *reply*). Teme nisu optimalno rješenje za *Request-Reply* razmjenu informacija, stoga Servisi rješavaju problem interakcije u kojem klijent šalje zahtjev na čvor koji se procesira i očekuje adekvatan odgovor.

- **Spremnici (engl. *Bags*)**

  Spremnici su formati zadužen za pohranu i ponovno korištenje ROS poruka. Oni su bitan mehanizam za pohranu podataka kao što su iščitavanja sa senzora. Nužni su za razvoj i testiranje algoritama.



Slika 3.7 ROS razina nadzora i upravljanja



Slika 3.8 Primjer komunikacije između ROS čvorova

3.2.3 ROS razina zajednice

ROS razina zajednice odnosi se na koncepte koji omogućuju izmjenu softverskih rješenja i znanja između različitih zajednica. Prethodno smo se upoznali s problemima koji su se kroz povijest pojavljivali u području robotike i ROS-a. Kroz distribuciju eksperimentalnih robota, stvaranjem istraživačkih skupina i pristupom otvorenog koda, razvojni inženjeri međusobno su stvorili snažan ekosustav razmjene znanja i programskih rješenja u ROS-u. On se razvija za zajednicu i od strane zajednice. Elementi ROS razine zajednice su:

- **Distribucije (engl. *Distributions*)**
  ROS distribucije su kolekcije verzioniranih paketa koji se mogu instalirati. Imaju sličnu ulogu kao Linux distribucije, odnosno olakšavaju instalaciju kolekcije softvera te ih održavaju.
- **Repozitoriji (engl. *Repositories*)**
  ROS se oslanja na zajedničku mrežu repozitorija gdje različite institucije mogu razvijati i objavljivati softverska rješenja za vlastite komponente.
- **ROS wiki**
  ROS wiki glavni je forum za objavljivanje dokumentacije vezane uz ROS, otvoren je za sve korisnike i svatko može doprinijeti ažuriranju i nadopunjavanju dokumentacije.
- **Bug Ticket System**
  Sistem za detekciju i indikaciju problema vezanih uz ROS softver.

- **Mailing Lista (engl. *Mailing Lists*)**

  Primarni komunikacijski kanal za obavještavanje o novim ažuriranjima ROS sustava, također platforma za postavljanje pitanja.
- **ROS Odgovori (engl. *ROS Answers*)**

  Stranica namijenjena odgovaranju na korisničke upite i nedoumice vezane uz ROS softver
- **Blog**

  ROS blog sadrži redovne obavijesti vezane uz sustav

4. PROGRAMSKO RJEŠENJE

Cilj ovog Diplomskog rada je ponuditi programsko rješenje za AlphaBot2-Pi mobilnu robotsku platformu koristeći ROS Noetic Ninjemys okvir te prikazati primjere kontroliranog kretanja robotske platforme preko vanjskog računala, kao i autonomno kretanje platforme uz pomoć senzora. U ovom poglavlju proći će se kroz sve korake integracije rješenja, započevši od instalacije odgovarajuće inačice Ubuntu operacijskog sustava na Raspberry Pi računalo, instalacije ROS Noetic-a, uspostavljanja komunikacije s vanjskim računalom te implementiranja odgovarajućeg programskog rješenja na robota koristeći ROS Noetic Ninjemys okvir.

4.1 Ubuntu 20.04.4 LTS instalacija

Početni korak je instalirati operacijski sustav na Raspberry Pi računalo. U prethodnom poglavlju ustanovljeno je kako Ubuntu sadrži najbolju podršku za ROS okvir i odabire se kao operacijski sustav mobilne robotske platforme. Podržane verzije Ubuntua za Raspberry Pi 3 Model B mogu se pronaći na Ubuntu službenoj stranici[1]. Za instalaciju operacijskog sustava na Raspberry Pi potrebna je microSD kartica i Raspberry Pi Imager alat za Windows (Slika 4.1). Prilikom odabira verzije ROS-a, važno je na Raspberry Pi računalo instalirati inačicu Ubuntua koja istu podržava. S obzirom na specifikacije računala, odabire se za Ubuntu 20.04.4 LTS Server, optimalan za najnoviji ROS Noetic Ninjemys okvir. LTS (Long Term Support) ukazuje na činjenicu da se inačica za vrijeme svog životnog vijeka neprestano ažurira, modificira i održava. Raspberry Pi ne podržava grafičko sučelje na ovoj verziji Ubuntua stoga će se koristiti komandno sučelje za preostale korake pripreme robota za rad.

[1] https://ubuntu.com/download/raspberry-pi

Slika 4.1 Raspberry Pi Imager v1.7.2

Nakon što se otvorio Raspberry Pi Imager i umetnula microSD kartica na računalo, pritiskom na opciju *CHOOSE OS → Other general-purpose OS* mogu se pronaći dostupne inačice Ubuntu operacijskog sustava za Raspberry Pi. Odabire se verzija Ubuntu Server 20.04.4 LTS te se pohranjuje na microSD karticu koja se zatim uključuje u Raspberry Pi računalo nakon instalacije.

4.2 Povezivanje na Raspberry Pi

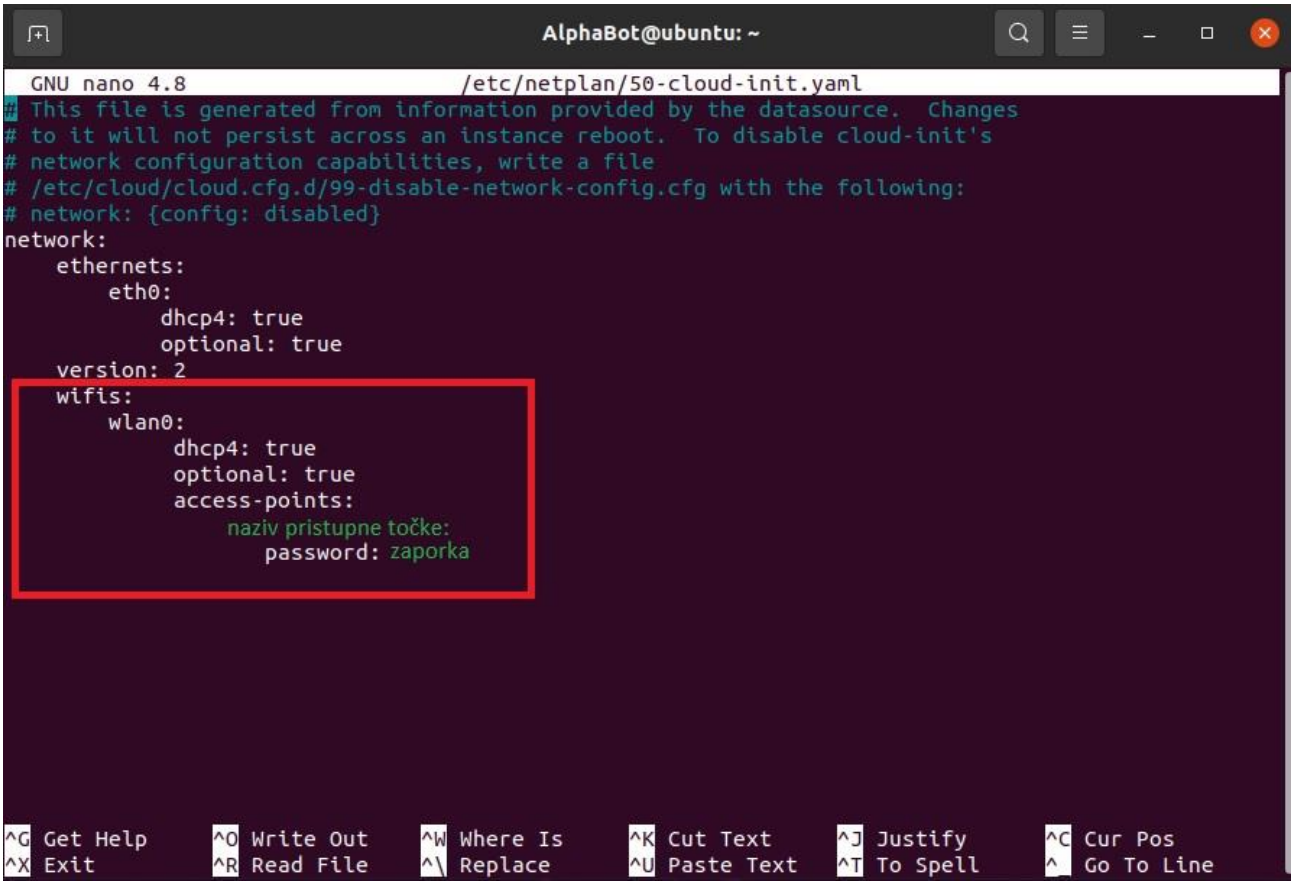
Na Raspberry Pi računalo priključuje se monitor preko HDMI kabla, tipkovnica i umeće microSD kartica s instaliranim Ubuntu 20.04.4 LTS operacijskim sustavom. Pokreće se Raspberry Pi računalo te unose korisnički podaci za pristup operacijskom sustavu. Prvobitna vrijednost za oba podatka je *ubuntu*. Potrebno je promijeniti zaporku nakon prve prijave i ponoviti unos. Mobilnoj robotskoj platformi daje se ime AlphaBot kako bi se ona mogla prepoznati i razlikovati u terminalu s udaljenog računala. Ulazi se u *root* korisnika i mijenja ime željenom korisniku pozivom naredbe:

```
root@ubuntu:~# sudo usermod -l ubuntu AlphaBot
```

Prva vrijednost nakon naredbe odnosi se na staro korisničko ime, a druga na novo. S obzirom na to kako je Ubuntu tek instaliran, potrebno je instalirati pakete i zavisnosti neophodne za održiv i stabilan rad ROS okvira. Kako bi se omogućilo preuzimanje potrebnih paketa, Ubuntu se prvo treba povezati na internet. Otvara se 50-*cloud-init.yaml* datoteka u kojoj se može postaviti WiFi konfiguracija u obliku pristupne točke i zaporke[2] (Slika 4.2). Poziva se sljedeća naredba:

```
AlphaBot@ubuntu:~$ sudo nano /etc/netplan/50-cloud-init.yaml
```

[2] https://itsfoss.com/connect-wifi-terminal-ubuntu/



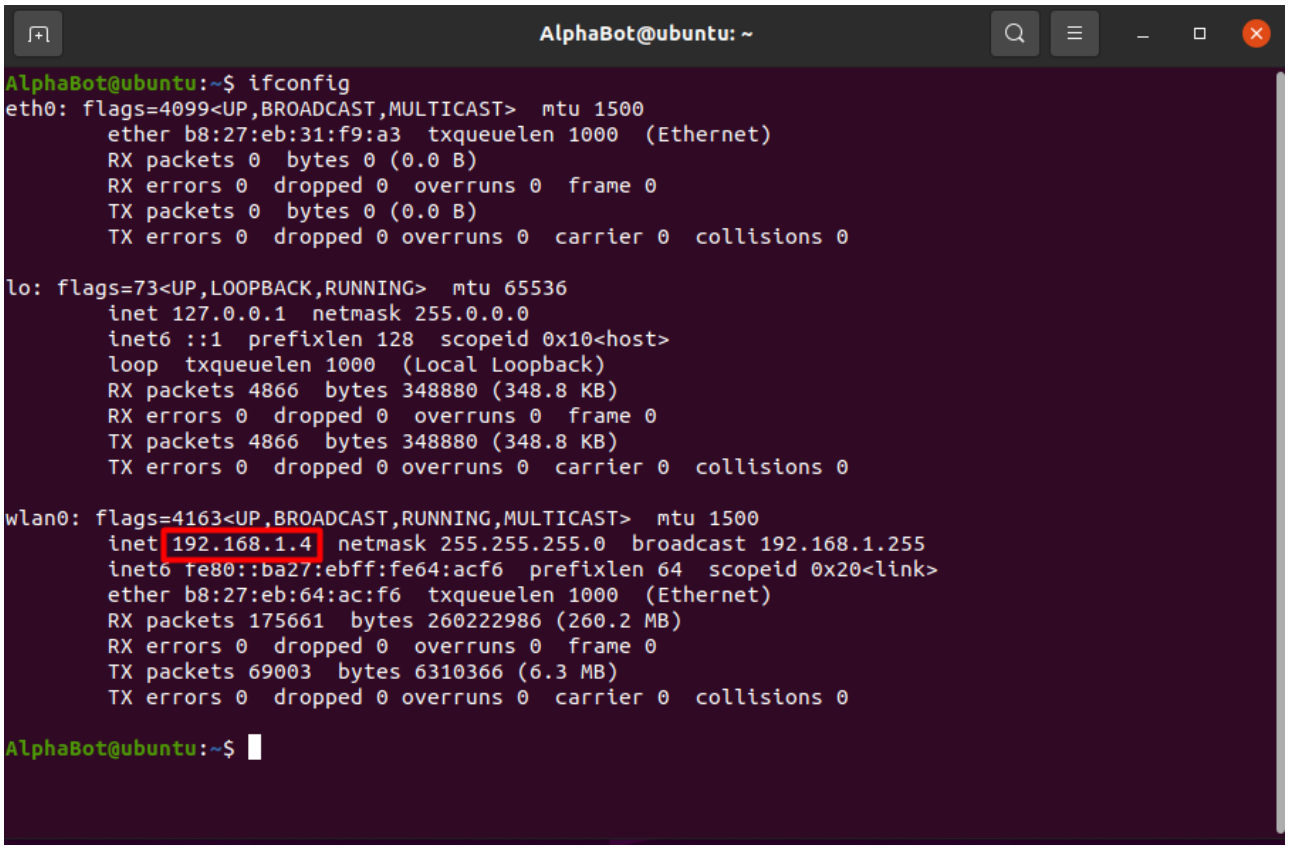
Slika 4.2 Ažurirana 50-cloud-init.yaml datoteka

Nakon što su se postavile vrijednosti za pristupnu točku, izmijenjena mrežna konfiguracija ažurira se pozivom sljedećih naredbi:

```
AlphaBot@ubuntu:~$ sudo netplan generate
AlphaBot@ubuntu:~$ sudo netplan apply
```

Na Raspberry Pi računalo povezuje se pomoću VMware Workstation Player alata (Virtualne Mašine) i Secure Shell (SSH) protokola. VMware Workstation linija je hipervizor proizvoda za radnu površinu (engl. *Desktop*), odnosno alat koji omogućuje pokretanje i upravljanje jedne ili više virtualnih mašina na osobnom računalu. Za efikasnu komunikaciju između osobnog računala i Raspberry Pi-ja, na VMware Workstation instalira se Ubuntu 20.04.4 LTS s grafičkim sučeljem kako bi se mogli koristiti alati za vizualizaciju procesa i elemenata ROS-a, kao i Integrirano razvojno

okruženje (engl. *Integrated Development Environment* (IDE)) za jednostavnije pisanje programskog koda robota. SSH je UNIX orijentirano sučelje i protokol namijenjen za sigurno uspostavljanje komunikacijskog kanala između dvaju računala te omogućuje vanjski pristup komandnom sučelju drugog računala. Sljedeći korak je pozvati naredbu *ifconfig* te pročitati vrijednost IP adrese Raspberry Pi računala. Vrijednost će se prikazati pod *wlan0* konfiguracijom uz *inet.*

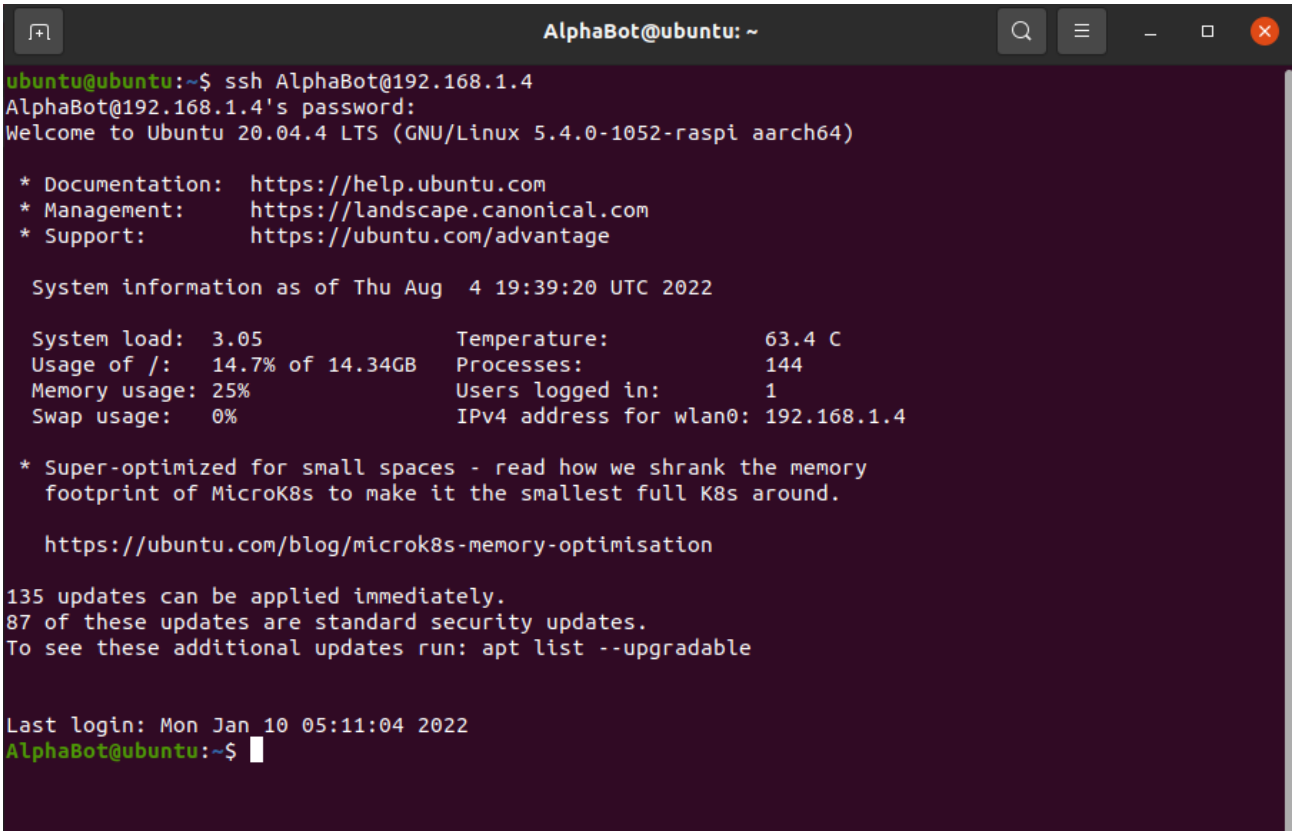


Slika 4.3 IP adresa Raspberry Pi računala

IP adresa Raspberry Pi računala je ***192.168.1.4***. Otvara se VMware Workstation Player na osobnom računalu s instaliranim Ubuntu operacijskim sustavom i u terminal upisuje sljedeća naredba:

```
ubuntu@ubuntu:~$ ssh AlphaBot@192.168.1.4
```

Upisuju se korisničko ime i zaporka AlphaBota za uspješno povezivanje s mobilnom robotskom platformom.



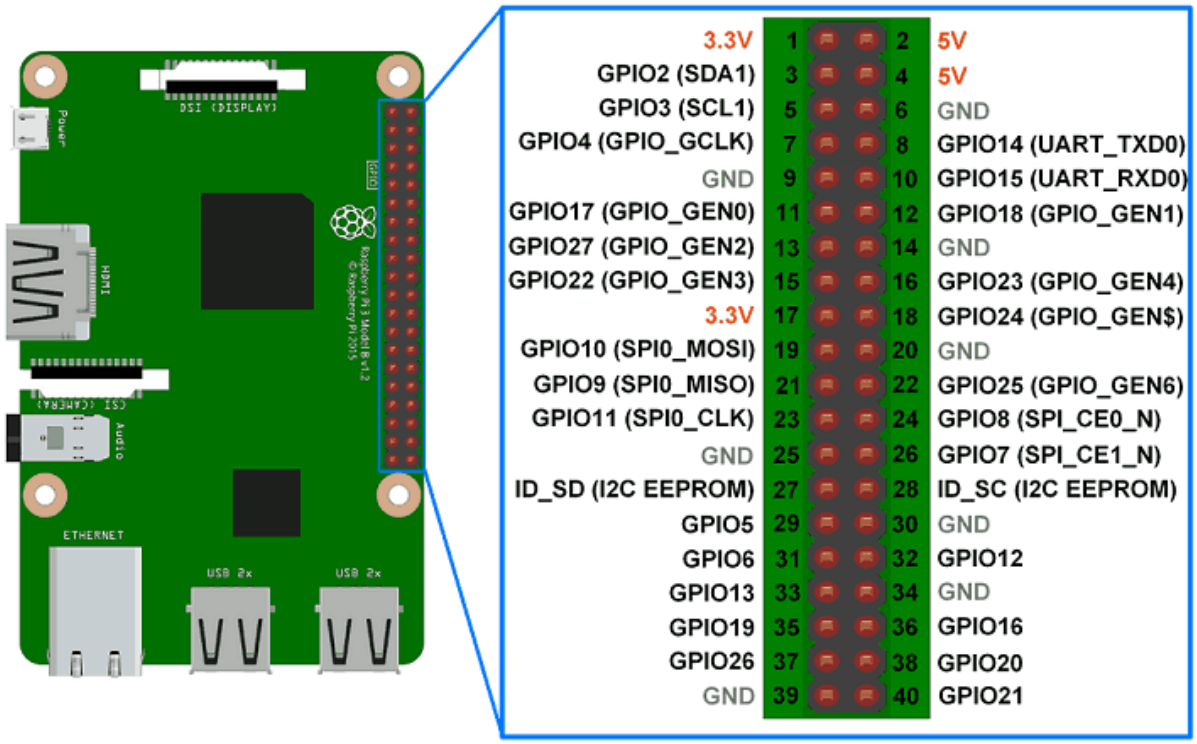
Slika 4.4 Povezivanje na Raspberry Pi u terminalu virtualne mašine

4.3 ROS Noetic Ninjemys instalacija

ROS Noetic je trinaesta i zadnja distribucija ROS 1.0 okvira. Objavljen je 23. svibnja 2020. godine od strane Open Robotics-a te je kao LTS inačica usmjerena na Ubuntu 20.04 operacijski sustav, izdan 23. travnja 2020. godine. ROS Noetic će kao LTS verzija biti podržan do svibnja 2025. godine, kada bilježi EOL (End of Life). Glavna promjena u usporedbi s prethodnim distribucijama jest to što koristi Python 3.

Na Raspberry Pi računalo povezuje se pomoću SSH protoka te će se ROS Noetic instalirati preko virtualne mašine na osobnom računalu. Potrebno je prvo omogućiti računalu preuzimanje paketa sa stranice koja ih sadrži te postaviti ključeve. Potpuni proces instalacije može se pronaći na wiki stranici ROS-a[3]. Prilikom instalacije može se odabrati više opcija, uključujući instalaciju alata za 2D i 3D simulaciju te vizualizaciju procesa. Odabire se ROS-Base zato što Ubuntu inačica na Raspberry Pi računalu nema grafičko sučelje. Potrebno je zatim postaviti odgovarajuće okruženje i instalirati zavisnosti. Proces instalacije ponavlja se i na virtualnoj mašini, ali uz odabir ROS Desktop-Full opcije. Nadalje, potrebno je riješiti komunikaciju računala s vanjskim senzorima. Raspberry Pi sadrži GPIO pinove preko kojih su motor i senzori spojeni. Kako bi se omogućilo računalu primanje i slanje informacija motorima i senzorima, instalira se GPIO (General Purpose Input/Output) biblioteka na Ubuntu operacijski sustav naredbom:

```
AlphaBot@ubuntu:~$ sudo apt-get install rpi.gpio
```



Slika 4.5 Raspored GPIO pinova na Raspberry Pi-ju

[3] http://wiki.ros.org/noetic/Installation/Ubuntu

Za pisanje i ažuriranje samog programskog rješenja na robotu instalirat će se Vim i Git na Raspberry Pi. Vim je uređivač teksta otvorenog koda i temelji se na naredbama unesenim preko tekstualnog korisničkog sučelja. Omogućuje uređivanje i modifikaciju izvornog koda. Koristit će se za uređivanje programskog koda na Raspberry Pi računalu te pravljenje manjih izmjena u kodu ili ispravljanje eventualnih grešaka koje će se pojaviti prilikom razrade programskog rješenja. Za inicijalizaciju Git repozitorija potrebno je generirati SSH ključeve na Raspberry Pi-ju i Ubuntu virtualnoj mašini pozivom naredbe *ssh-keygen* i ispisati ga pomoću naredbe *cat ~/.ssh/id_rsa.pub*[4]. Ako se na Raspberry spojilo preko virtualne mašine na vanjskom računalu i SSH protokola, može se vrlo brzo postaviti ključ na osobni GitHub profil u *SSH and GPG keys* izbornik. S obzirom na to kako Ubuntu na Raspberry Pi-ju koristi komandno sučelje, pisanje većeg i zahtjevnijeg koda direktno na Raspberry može se ispostaviti kao nezahvalan i vremenski zahtjevan zadatak. Inicijalizacijom Git-a ostvaruje se kontrola verzije programskog koda i mogućnost korištenja IDE-ja s grafičkim sučeljem preko vanjskog računala za efikasnije pisanje i ažuriranje programskog rješenja za robota.

4.4 Catkin

Catkin je službeni sustav izrade za ROS te je nasljednik i nadogradnja prvobitnog rosbuild-a. Kako bi se započela razrada samog programskog rješenja, izrađuje se Catkin radna okolina na Raspberry Pi računalu. On kombinira CMake makronaredbe s Python skriptama i omogućuje podršku za razvoj na velikim skupovima neovisnih paketa povezanih u zavisni ekosustav koda. S obzirom na to kako se ROS bazira upravo na takvim paketima, Catkin je razvijen kao službena razvojna okolina za ROS s ciljem pojednostavljivanja izrade programske podrške za robote. Izrađuje se direktorij s imenom *alphabot_ws* unutar kojeg se izrađuje *src* direktorij gdje će se izraditi paketi za Catkin radnu okolinu. Naredba kao parametre prima ime paketa i biblioteka koje će se koristiti. Zatim se izlazi iz *src* direktorija te poziva naredba *catkin_make* za izgradnju koda u radnoj okolini koja će se pozvati kada se dodaju nove zavisnosti ili biblioteke u programsko rješenje i želi izvršiti proces kompiliranja paketa i koda. Catkin radna okolina izrađuje se pozivom sljedećih naredbi u terminalu:

[4] https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys-on-ubuntu-20-04

```
AlphaBot@ubuntu:~$ mkdir alphabot_ws

AlphaBot@ubuntu:~$ cd alphabot_ws

AlphaBot@ubuntu:~/alphabot_ws$ mkdir src

AlphaBot@ubuntu:~/alphabot_ws$ cd src

AlphaBot@ubuntu:~/alphabot_ws/src$ catkin_create_pkg alphabot_controller rospy

AlphaBot@ubuntu:~/alphabot_ws/src$ cd ..

AlphaBot@ubuntu:~/alphabot_ws$ catkin_make
```

Slika 4.6 Catkin radna okolina

4.5 Implementacija motora i kotačića

Unutar *alphabot_controller* direktorija izrađuje se direktorij *scripts* gdje će se implementirati programsko rješenje za sve potrebne funkcionalnosti robota. Naredbom *touch motors_controller.py* izrađuje se Python skripta za rad motora i kotačića te joj se dodjeljuje dozvola za izvršavanje (engl. *execute permission*) pozivom naredbe *chmod +x motors_controller.py*. Pisanje programskog koda moguće je izvršiti preko Vim uređivača koda ili IDE-ja na vanjskom računalu. Izrada programskog rješenja u ovom diplomskom radu bit će izvedena koristeći Visual Studio Code koji je instaliran na virtualnoj mašini na vanjskom računalu. Za početak su potrebne *rospy* i *Rpi.GPIO* biblioteke koje se pozivaju u skripti. Ustanovljeno je kako se pomoću GPIO biblioteke mogu definirati pinovi na Raspberry Pi računalu i koristiti u programskom rješenju za konfiguraciju rada motora, senzora i ostalih hardverskih komponenti. Na internetskoj stranici AlphaBot2-Pi mobilne robotske platforme[5] može se pronaći raspored pinova za motore kako bi ih inicijalizirali. Izrađuje se klasa *AlphaBot* gdje će se definirati metode za kretanje i inicijalizaciju GPIO pinova. Skripta počinje s *#!/usr/bin/env python3*, deklaracijom kojom se definira kako će skripta biti izvršena kao Python skripta. Prvobitno se definira *__init__* metoda, koja se svakim instanciranjem navedene klase izvršava. Unutar metode konfiguriraju se vrijednosti pinova i postavlja im se vrijednosti u GPIO.OUT, implicirajući kako će motori i kotačići iščitavati vrijednosti iz programskog koda te na temelju njih operirati i mijenjati smjer kretanja. Brzina motora i kotačića ostvaruje se pomoću *GPIO.PWM*[6] klase, odnosno pomoću modulacije širine impulsa (engl. *Pulse Width Modulation*). Modulacija širine impulsa je metoda reduciranja snage zaprimljene od električnog signala usitnjavanjem na diskretne dijelove. Ona kao parametre prima vrijednost GPIO pina te frekvenciju rada. PWM vrijednost će se prilikom testiranja mijenjati i optimizirati jer direktno utječe na brzinu kretanja motora i kotačića na robotu.

[5] https://www.waveshare.com/wiki/AlphaBot2-Pi
[6] https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/

```
class AlphaBot(object):

        def __init__(self,ain1=12,ain2=13,ena=6,bin1=20,bin2=21,enb=26):
                self.AIN1 = ain1
                self.AIN2 = ain2
                self.BIN1 = bin1
                self.BIN2 = bin2
                self.ENA = ena
                self.ENB = enb
                self.PA  = 20
                self.PB  = 20

                GPIO.setmode(GPIO.BCM)
                GPIO.setwarnings(False)
                GPIO.setup(self.AIN1,GPIO.OUT)
                GPIO.setup(self.AIN2,GPIO.OUT)
                GPIO.setup(self.BIN1,GPIO.OUT)
                GPIO.setup(self.BIN2,GPIO.OUT)
                GPIO.setup(self.ENA,GPIO.OUT)
                GPIO.setup(self.ENB,GPIO.OUT)

                self.PWMA = GPIO.PWM(self.ENA,500)
                self.PWMB = GPIO.PWM(self.ENB,500)
                self.PWMA.start(self.PA)
                self.PWMB.start(self.PB)
                self.stop()
```

Slika 4.7 *__init__* metoda unutar klase AlphaBot

GPIO.BCM način rada ukazuje na činjenicu kako se brojke pinova iščitavaju prema *Broadcom SOC Channel* numeriranju. Svaki od motora definiran je s dvjema varijablama koje nose vrijednosti GPIO pinova. Lijevi motor definiran je s AIN1 i AIN2, a desni s BIN1 i BIN2 pomoću kojih se određuje smjer kretanja motora u ovisnosti o pinovima na koje se pošalje signal. Ukoliko se želi ostvariti kretanje unaprijed, potrebno je poslati signal na pinove AIN2 i BIN2. Pomoću *GPIO.output* metode postavljaju se vrijednosti pinovima AIN2 i BIN2 u GPIO.HIGH, dok će pinovi AIN1 i BIN1 imati vrijednost GPIO.LOW. Na ENA i ENB pinove motora poziva se modulacija širine impulsa koja će odrediti brzinu kretanja. *ChangeDutyCycle* metoda iz klase *PWM* kao parametar prima vrijednosti između 0 i 100, a ovisno o poslanoj vrijednosti konfigurira se brzina kretanja motora mobilne robotske platforme jer poslani parametar utječe na vrijednost aktivnog stanja signala koji se šalje na motore u ovisnosti o njegovom periodu. Kako bi se ostvarilo kretanje unazad, potrebno je obrnuti logiku metode za kretanje prema naprijed te pinovima AIN1 i BIN1 postaviti vrijednost u GPIO.HIGH, dok će AIN2 i BIN2 preći u GPIO.LOW. Za zaustavljanje mobilne robotske platforme svi pinovi motora

postavljaju se u GPIO.LOW i metodi *ChangeDutyCycle* šalje se vrijednost 0. Kombinacijom navedenih varijabli mogu se implementirati metode za kretanje ulijevo i udesno.

```
    def forward(self):
            self.PWMA.ChangeDutyCycle(self.PA)
            self.PWMB.ChangeDutyCycle(self.PB)
            GPIO.output(self.AIN1,GPIO.LOW)
            GPIO.output(self.AIN2,GPIO.HIGH)
            GPIO.output(self.BIN1,GPIO.LOW)
            GPIO.output(self.BIN2,GPIO.HIGH)

    def stop(self):
            self.PWMA.ChangeDutyCycle(0)
            self.PWMB.ChangeDutyCycle(0)
            GPIO.output(self.AIN1,GPIO.LOW)
            GPIO.output(self.AIN2,GPIO.LOW)
            GPIO.output(self.BIN1,GPIO.LOW)
            GPIO.output(self.BIN2,GPIO.LOW)

    def backward(self):
            self.PWMA.ChangeDutyCycle(self.PA)
            self.PWMB.ChangeDutyCycle(self.PB)
            GPIO.output(self.AIN1,GPIO.HIGH)
            GPIO.output(self.AIN2,GPIO.LOW)
            GPIO.output(self.BIN1,GPIO.HIGH)
            GPIO.output(self.BIN2,GPIO.LOW)

    def left(self):
            self.PWMA.ChangeDutyCycle(17)
            self.PWMB.ChangeDutyCycle(17)
            GPIO.output(self.AIN1,GPIO.HIGH)
            GPIO.output(self.AIN2,GPIO.LOW)
            GPIO.output(self.BIN1,GPIO.LOW)
            GPIO.output(self.BIN2,GPIO.HIGH)


    def right(self):
            self.PWMA.ChangeDutyCycle(17)
            self.PWMB.ChangeDutyCycle(17)
            GPIO.output(self.AIN1,GPIO.LOW)
            GPIO.output(self.AIN2,GPIO.HIGH)
            GPIO.output(self.BIN1,GPIO.HIGH)
            GPIO.output(self.BIN2,GPIO.LOW)
```

Slika 4.8 Metode za smjer kretanja robota unutar klase AlphaBot

Inicijalizira se *motors_controller* čvor pozivom metode *rospy.init_node("motors_controller")* kako bi se u daljnjoj razradi programskog rješenja omogućilo primanje informacija od tipkovnice i senzora te pravovremeno promijenio smjer kretanja i brzina motora mobilne robotske platforme.

4.6 Implementacija kontroliranog upravljanja

Razrada programskog rješenja za kretanje robota započet će implementacijom kontroliranog kretanja preko tipkovnice s vanjskog računala kako bi se testiranjem parametara brzine motora pronašle optimalne vrijednosti prije realizacije autonomnog kretanja. Korisniku se omogućuje unos tipke s tipkovnice u terminal bez potrebe za pritiskom tipke *Enter* pozivom metode *__call__* iz klase *_GetchUnix*[7] koja omogućuje čitanje pritisnute tipke u konzoli.

```
class _GetchUnix:
    def __init__(self):
        import tty, sys

    def __call__(self):
        import sys, tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch
```

Slika 4.9 *_GetchUnix* klasa unutar koje se nalazi __call__ metoda

Izrađuje se klasa *Remote* gdje će se definirati metoda *keyboard_input* koja će kao vrijednost vraćati pritisnutu tipku na tipkovnici. Omogućuje se korisniku da pritiskom na tipke W, A, S, D odredi smjer kretanja robota stoga se pravilno postavlja uvjetovanje temeljem navedenih vrijednosti. Dodaje se i opcija za zaustavljanje mobilne robotske platforme pritiskom na *Space*, kao i zaustavljanje programa pritiskom tipke Q koja će pozvati *KeyboardInterrupt*.

```
class Remote:
    def keyboard_input(self):
        rospy.loginfo_once("\nw - forward\na - left\ns - backward\nd - right\nspace - stop\n\nq - exit\n----------------")
        key_pressed = _GetchUnix().__call__()
        while key_pressed == 'w' or key_pressed == 'a' or key_pressed == 's' or key_pressed == 'd' or key_pressed == ' ':
            return str(key_pressed)
        if key_pressed == 'q':
            raise KeyboardInterrupt
```

Slika 4.10 *Remote* klasa unutar koje se nalazi *keyboard_input* metoda

[7] https://code.activestate.com/recipes/134892-getch-like-unbuffered-character-reading-from-stdin/

Inicijalizira se čvor pozivom metode *rospy.init_node("remote_controller")* u skripti *remote_controller.py* koji će čvoru za upravljanje motora *motors_controller* slati informaciju o pritisnutoj tipki na tipkovnici preko teme *remote/control/input* u obliku String poruke. S obzirom na to kako će ovaj čvor objavljivati poruku na temu, pozivom klase *rospy.Publisher* definira se ime teme na koju će se poruka objavljivati, vrsta poruke i veličina reda čekanja (engl. *queue_size*). Odabire se String tip poruke iz paketa *std_msgs* jer se u ovom slučaju šalje vrijednosti pritisnutih tipki s tipkovnice. Poruka će se objavljivati na temu sve dok je čvor aktivan, odnosno dok se ne pritisne tipka Q koja će ugasiti čvor.

```
if __name__ == '__main__':
    rospy.init_node("remote_controller")
    remote_cntrl = Remote()
    remote_pub = rospy.Publisher("remote/control/input",String,queue_size=10)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        pressed_key = remote_cntrl.keyboard_input()
        rospy.loginfo(pressed_key)
        remote_pub.publish(pressed_key)
        rate.sleep()
```

Slika 4.11 Inicijalizacija čvora i teme na koju objavljujemo pritisnutu tipku na tipkovnici

Kako bi se *motors_controller* čvor mogao pretplatiti na temu *remote/control/input* i primati informaciju o pritisnutoj tipki od *remote_controller* čvora, potrebno je pozvati klasu *rospy.Subscriber* gdje će se kao parametri navesti naziv teme, tip zaprimljene poruke i *callback* metoda koja će na zaprimljenu poruku reagirati. Metoda *remote_movement* će na osnovi zaprimljene poruke mijenjati smjer kretanja mobilne robotske platforme.

```
if __name__ == '__main__':
    rospy.init_node("motors_controller")
    Ab = AlphaBot()
    remote_sub = rospy.Subscriber("remote/control/input", String, callback=remote_movement)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        rate.sleep()
```

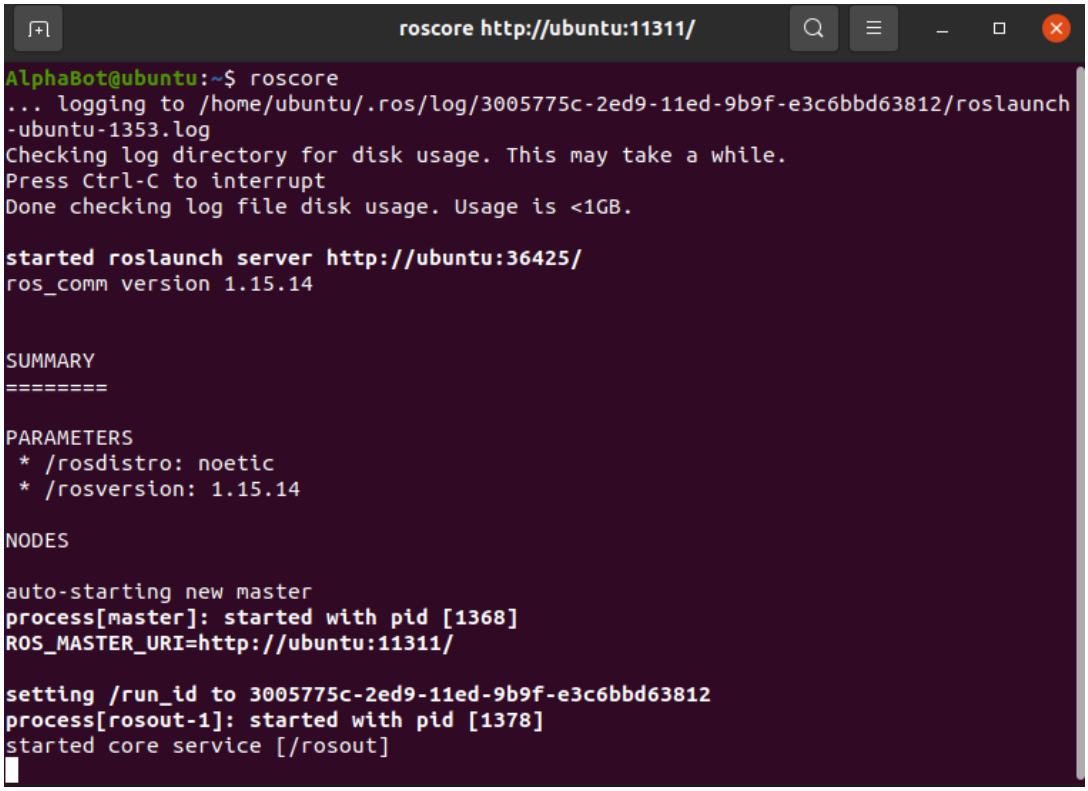
Slika 4.12 Pretplata čvora *motors_controller* na *remote/control/input* temu

```
def remote_movement(pressed_key: String):
        if pressed_key.data == 'w':
                Ab.forward()
                rospy.loginfo("Forward")
        elif pressed_key.data == 'a':
                Ab.left()
                time.sleep(0.8)
                Ab.forward()
                rospy.loginfo("Left")
        elif pressed_key.data == 's':
                Ab.backward()
                rospy.loginfo("Backward")
        elif pressed_key.data == 'd':
                Ab.right()
                time.sleep(0.8)
                Ab.forward()
                rospy.loginfo("Right")
        elif pressed_key.data == ' ':
                Ab.stop()
                rospy.loginfo("Motors are stopped")
```

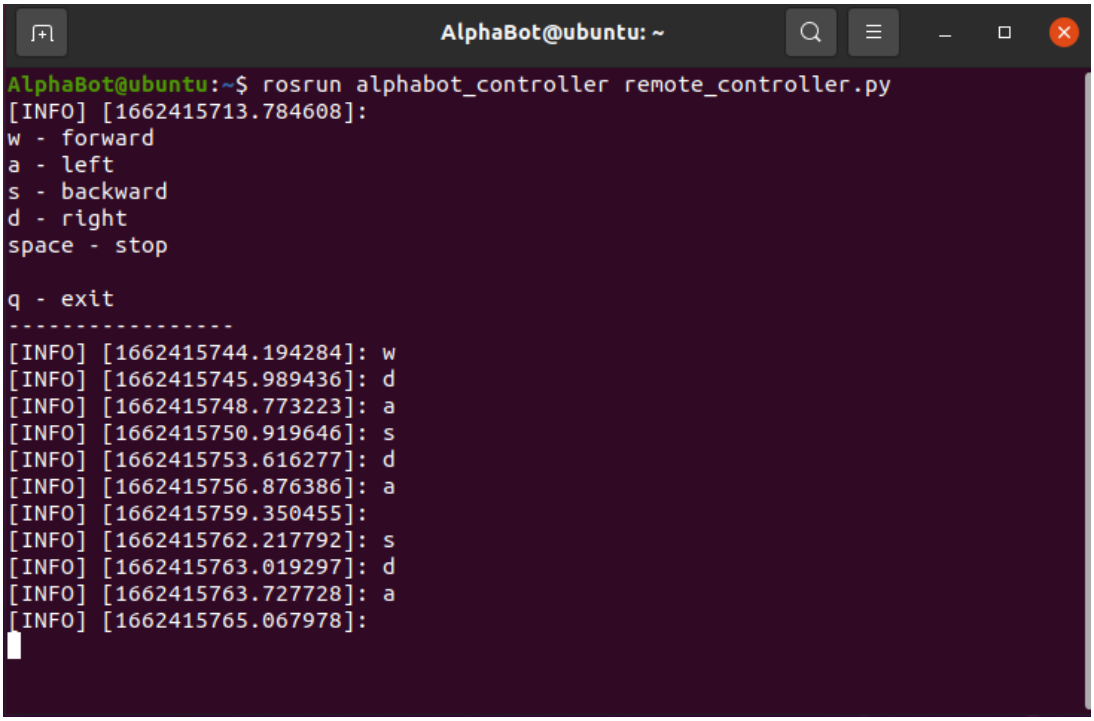
Slika 4.13 *remote_movement* metoda za određivanje smjera kretanja

S obzirom na to kako se u skripti koristi String tip poruke iz *std_msgs* paketa, potrebno je ažurirati *package.xml* Manifest i navesti korištene biblioteke kako bi se programski kod uspješno kompilirao te pozvati naredbu *catkin_make*. Nakon kompiliranja programskog koda, poziva se naredba *roscore* i pokreće Master čvor kako bi se ispitala komunikacija između dvaju čvorova i funkcionalnost kontroliranog upravljanja robota. Čvorovi se pokreću naredbama *rosrun alphabot_controller remote_controller.py* i *rosrun alphabot_controller motors_controller.py* u terminalima. Prilikom razrade programskog rješenja, vizualizacija procesa i aktivnih elemenata sustava na vanjskom računalu može se postići pomoću alata rqt_graph pozivom naredbe *rqt_graph* u terminalu.

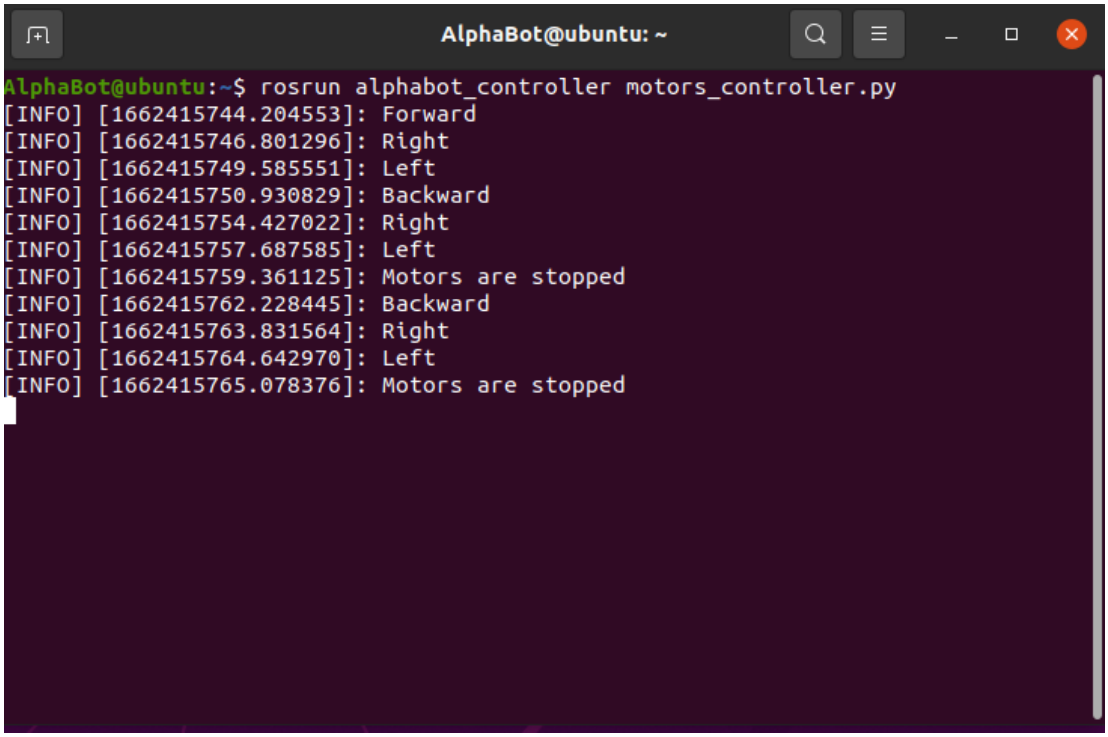
Slika 4.14 *rqt_graph* graf procesa kontroliranog upravljanja robota



Slika 4.15 Master čvor



Slika 4.16 *remote_controller* čvor



Slika 4.17 *motors_controller* čvor

4.7 Implementacija ultrazvučnog senzora (HC-SR04)

S funkcionalnosti HC-SR04 senzora upoznalo se u 2. poglavlju diplomskog rada te je ustanovljeno kako senzor radi na principu samorefleksije. Ideja implementacije rješenja ultrazvučnog senzora jest izraditi metodu koja će računati i vraćati udaljenost od objekta, vrijednost objavljivati na temu na koju će se *motors_controller* čvor pretplatiti te promijeniti smjer kretanja u ovisnosti o zaprimljenoj udaljenosti od objekta. Izrađuje se Python skripta *ultrasonic_sensor_controller.py* u kojoj započinje razrada programskog rješenja za senzor. Definira se klasa *Sonar* gdje se unutar *__init__* metode inicijaliziraju GPIO pinovi i BCM način rada. Unutar klase se također definira metoda *distance* za računanje udaljenosti od objekta. Impuls se šalje na TRIG pin koji odašilja signal i označuje se prva točka vremenskog intervala. Postavlja se uvjetovanje na ECHO pin koji, kada detektira reflektirani signal, mijenja svoje stanje i označuje drugu točku vremenskog intervala. Izvedbom matematičke formule (Slika 2.3) uzima se razlika između zabilježenih vremena i množi s

brzinom zvuka (v = 340 *m/s*). Vrijednost se dijeli s 2 s obzirom na to kako je prava udaljenost polovina prijeđenog puta vala.

```python
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
import RPi.GPIO as GPIO
import time

class Sonar(object):

    def __init__(self, trig=22, echo=27):
        self.TRIG = trig
        self.ECHO = echo
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.TRIG,GPIO.OUT,initial=GPIO.LOW)
        GPIO.setup(self.ECHO,GPIO.IN)

    def distance(self):
        GPIO.output(self.TRIG,GPIO.HIGH)
        time.sleep(0.000015)
        GPIO.output(self.TRIG,GPIO.LOW)
        while not GPIO.input(self.ECHO):
            pass
        t1 = time.time()
        while GPIO.input(self.ECHO):
            pass
        t2 = time.time()
        return (t2-t1)*34000/2
```

Slika 4.18 *Sonar* klasa

Inicijalizira se čvor *ultrasonic_sensor_controller* koji će objavljivati Float64 tip poruke na temu *ultrasonic/distance.* Čvor *motors_controller* će se pretplatiti na temu, dobivati informaciju o udaljenosti od objekta preko ultrazvučnog senzora te, prema uvjetovanju na temelju postavljene vrijednosti kritične udaljenosti, promijeniti smjer kretanja. Informacija će se slati kontinuirano, odnosno sve dok je čvor za senzor aktivan kako bi mobilna robotska platforma u svakom trenutku imala vrijednost udaljenosti od objekta.

```
if __name__ == '__main__':
    rospy.init_node("ultrasonic_sensor_controller")
    rospy.loginfo("Ultrasonic sensor is engaged")
    us_sensor = Sonar()
    us_pub = rospy.Publisher("ultrasonic/distance", Float64, queue_size=10)
    rate = rospy.Rate(2)
    while not rospy.is_shutdown():
        dist = us_sensor.distance()
        rospy.loginfo(dist)
        us_pub.publish(dist)
        rate.sleep()
```

Slika 4.19 Konfiguracija *ultrasonic_sensor_controller* čvora koji objavljuje vrijednost udaljenosti od prepreke na temu */ultrasonic/distance*

Logika za inicijalizaciju čvorova i tema na koju će se objavljivati informacije sa senzora piše se prema slici 4.19. Čvor će objavljivati vrijednosti sa senzora sve dok se on ne ugasi, a metodom *rospy.Rate* određuje se frekvencija rada čvora. *rospy.loginfo* je metoda za ispisivanje sadržaja u konzoli.

Nadalje, potrebno je konfigurirati *motors_controller* čvor, pretplatiti ga na temu i dodati mu *callback* metodu za reagiranje na zaprimljenu poruku od strane senzora. Uvjetovanje će se postaviti na temelju primljene udaljenosti od objekta i promijeniti robotu smjer kretanja udesno ako je došao do kritične udaljenosti od prepreke. Definira se *callback* metoda *react_to_distance* u kojoj se razrađuje logika za komunikaciju motora sa zaprimljenom informacijom udaljenosti od objekta.

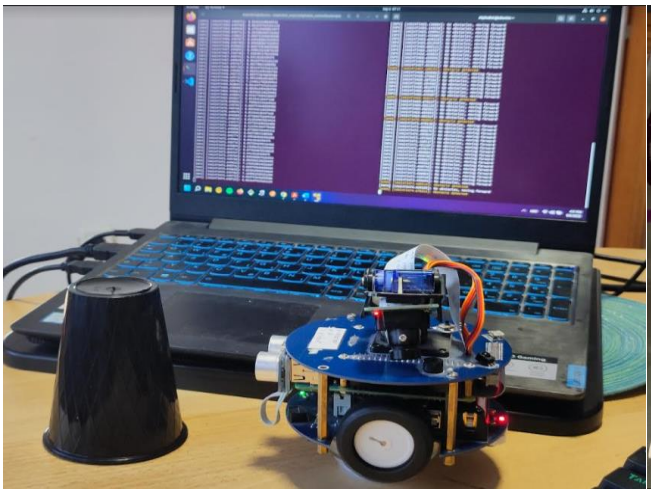
```
if __name__ == '__main__':
    rospy.init_node("motors_controller")
    Ab = AlphaBot()
    #remote_sub = rospy.Subscriber("remote/control/input", String, callback=remote_movement)
    us_sub = rospy.Subscriber("ultrasonice/distance", Float64, callback=react_to_distance)
    while not rospy.is_shutdown():
        rate.sleep()
```

Slika 4.19 Konfiguracija *motors_controller* čvora

```
def react_to_distance(dist: Float64):
    if int(dist.data) <= 12:
        Ab.right()
        rospy.loginfo("Right")
    else:
        Ab.forward()
        rospy.loginfo("Forward")
```

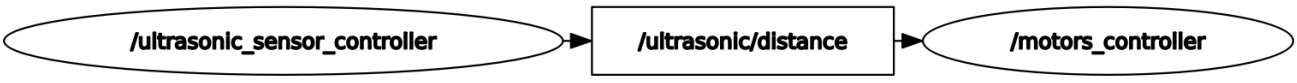
Slika 4.20 *react_to_distance* metoda

U zasebnim terminalima se pozivaju naredbe *roscore, rosrun alphabot_controller ultrasonic_sensor-_controller.py* te *rosrun alphabot_controller motors_controller.py* za testiranje ispravnosti komunikacije između čvorova.




Slika 4.21 i Slika 4.22 Ispitivanje funkcionalnosti ultrazvučnog senzora

Prilikom testiranja *ultrasonic_sensor_controller* čvora mogu se optimizirati vrijednost kritične udaljenosti i frekvencije rada čvora. Pomoću metode *rospy.Rate* mijenja se frekvencija rada, prosljeđuje joj se vrijednost 5, odnosno čvor objavljuje poruku na temu 5 puta u sekundi što je mobilnoj robotskoj platformi dovoljno za pravovremeno reagiranje na prepreku.



Slika 4.23 *rqt_graph* graf procesa

4.8 Implementacija infracrvenog senzora (ST188)

ST188 infracrveni senzor detektira objekt koji se nalazi u neposrednoj blizini mobilne robotske platforme pomoću signalnih indikatora. S obzirom na to kako je na svakoj strani mobilne robotske platforme po jedan ST188 senzor (Slika 2.2, redni broj 5), rješenje za autonomno kretanje robota može se nadograditi kombinacijom ultrazvučnog senzora i senzora slike tako da mobilna robotska platforma dobije informaciju o prisutnosti prepreke ispred, lijevo i desno od sebe za vrijeme kretanja, a zatim pravilno promijeni smjer kretanja obrnuto od strane na kojoj je detektirala prepreku. Izrađuje se skripta *infrared_sensor_controller.py* unutar koje će se napraviti klasa *IRSensors.* Unutar klase inicijaliziraju se GPIO pinovi u *__init__* metodi te definira metoda *detect_obstacle* za vraćanje vrijednosti signalnih indikatora senzora. Kada jedan od senzora detektira prepreku, mijenja stanje indikatora iz 1 u 0 te daje do znanja m*otors_controller* čvoru na kojoj strani je detektirana prepreka kako bi motori pravovremeno promijenili smjer kretanja.

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
import RPi.GPIO as GPIO
import time

class IRSensors(object):

    def __init__(self, dr=16, dl=19):
        self.DR = dr
        self.DL = dl
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.DR,GPIO.IN,GPIO.PUD_UP)
        GPIO.setup(self.DL,GPIO.IN,GPIO.PUD_UP)

    def detect_obstacle(self):
        DR_status = GPIO.input(self.DR)
        DL_status = GPIO.input(self.DL)
        return str(DR_status) + str(DL_status)
```

Slika 4.24 *IRSensors* klasa

Inicijalizira se čvor *infrared_sensors_controller* koji će objavljivati indikatorske vrijednosti obaju infracrvenih senzora o detektiranoj prepreci na temu *infrared/obstacle* na koju će se *motors_controller* pretplatiti i reagirati na temelju zaprimljene informacije. Poslana poruka će biti String tipa, sadržavat će kombinaciju dvaju brojeva (1,0), odnosno stanja senzora, a ovisno o njihovim vrijednostima u

poslanoj poruci, *motors_controller* čvor će moći pravilno promijeniti smjer kretanja. ST188 senzori domet detekcije objekta povećavaju pomoću potenciometara koji se nalaze na donjoj strani šasije mobilne robotske platforme (Slika 2.2, redni broj 7). Kada se detektira prepreka također se pali zelena indikatorska LED-ica.

```
if __name__ == '__main__':
    rospy.init_node("infrared_sensors_controller")
    rospy.loginfo("Infrared sensors are engaged")
    ir_sensor = IRSensors()
    ir_pub = rospy.Publisher("infrared/obstacle",String,queue_size=10)
    rate = rospy.Rate(5)
    while not rospy.is_shutdown():
        obj_detected = ir_sensor.detect_obstacle()
        rospy.loginfo(obj_detected)
        ir_pub.publish(obj_detected)
        rate.sleep()
```

Slika 4.25 Konfiguracija *infrared_sensors_controller* čvora

Unutar *motors_controller* čvora napravit će se jedna *callback* metoda koja će istovremeno primati informacije obaju senzora. Takav oblik konfiguracije pretplatničkog čvora ostvaruje se pomoću *message_filters* biblioteke koja sadrži često korištene algoritme filtriranja poruka. Na primjer, *ApproximateTimeSynchronizer* sinkronizira poruke na temelju njihovih vremenskih oznaka te omogućuje njihovo sparivanje i objavljivanje na teme u gotovo istoj vremenskoj točki tako da odgodi objavljivanje poruke s jednog od senzora dok se ne uskladi s porukom drugog senzora. Čvor se pretplaćuje na teme obaju senzora i poziva metodu *move_robot* tek kada su oba senzorska čvora aktivna.

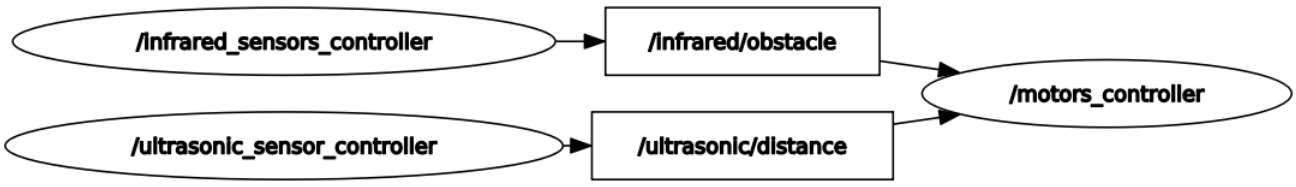
```
if __name__ == '__main__':
    rospy.init_node("motors_controller")
    Ab = AlphaBot()
    #remote_sub = rospy.Subscriber("remote/control/input", String, callback=remote_movement)
    us_sub = message_filters.Subscriber('ultrasonic/distance', Float64)
    ir_sub = message_filters.Subscriber('infrared/obstacle', String)
    ts = message_filters.ApproximateTimeSynchronizer([us_sub, ir_sub], 10, 0.1, allow_headerless=True)
    ts.registerCallback(move_robot)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        rate.sleep()
```

Slika 4.26 Konfiguracija *motors_controller* čvora

Čvor mijenja smjer kretanja u *move_robot* metodi na osnovi primljenih parametara sa senzora. Ako se prepreka nalazi lijevo od mobilne robotske platforme, potrebno je promijeniti smjer kretanja udesno i obrnuto. Ultrazvučni senzor odgovoran je za reagiranje na prepreke ispred mobilne platforme.

```
def move_robot(us_dist, ir_det):
    if int(us_dist.data) <= 12 or ir_det.data == '01':
        Ab.right()
        rospy.loginfo("Going right!")
    elif ir_det.data == '10':
        Ab.left()
        rospy.loginfo("Going left!")
    elif ir_det.data == '00':
        Ab.backward()
        rospy.loginfo("Going backwards")
    else:
        Ab.forward()
        rospy.loginfo("Going forward")
```

Slika 4.27 *move_robot* metoda

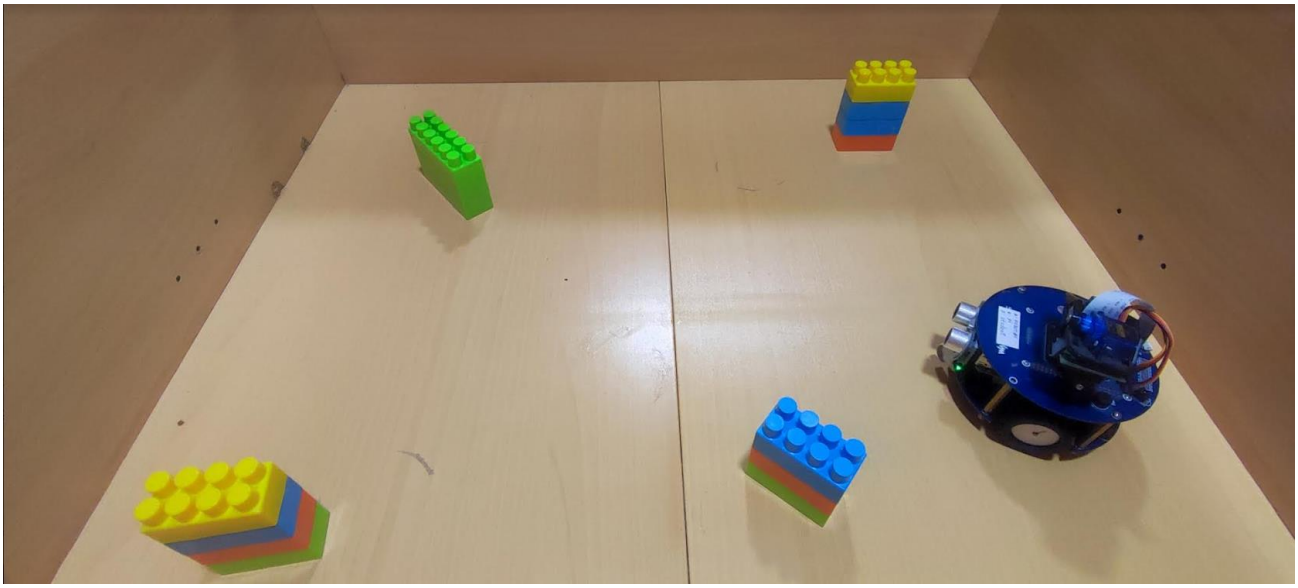


Slika 4.28 *rqt_graph* graf procesa autonomnog kretanja robota

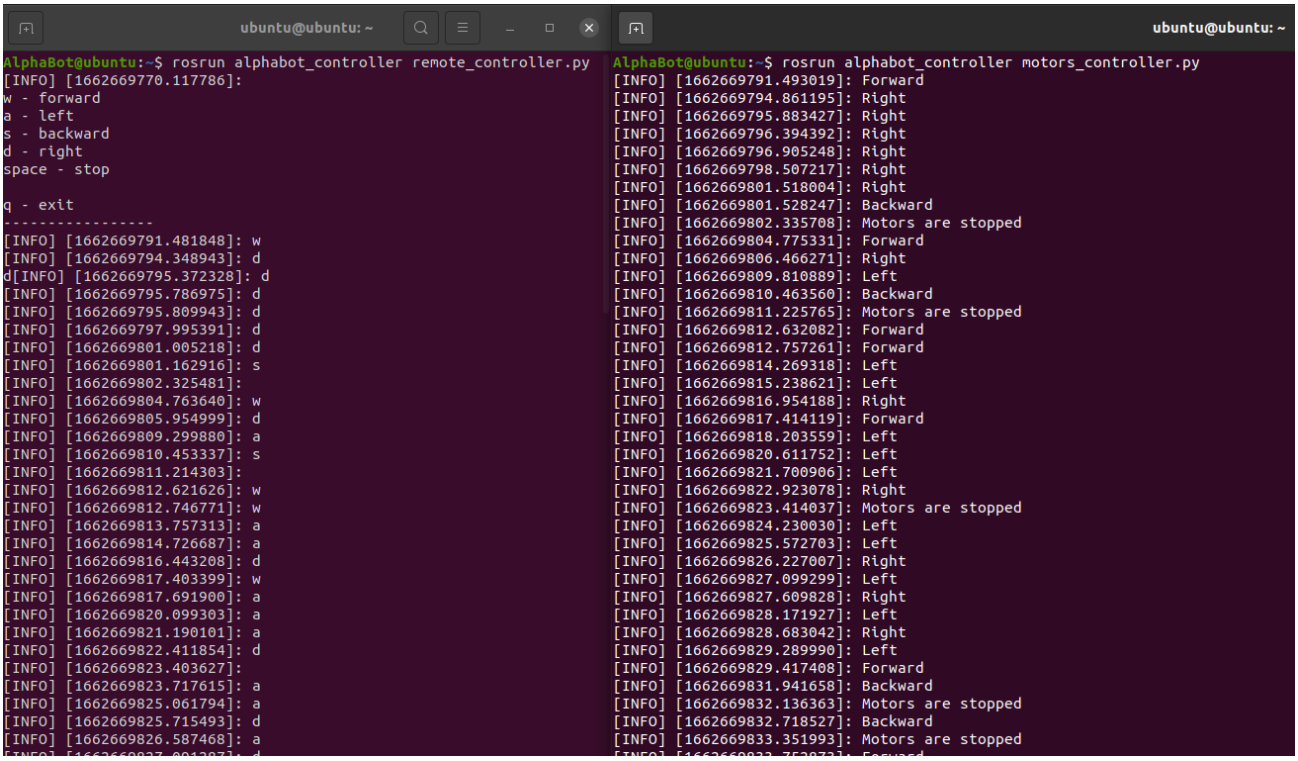
5. TESTIRANJE

Testiranjem mobilne robotske platforme ispitat će se funkcionalnost programske podrške za kontrolirano i autonomno kretanje robota te po potrebi izmijeniti vrijednosti parametara brzine, kritične udaljenosti od prepreke i dometa detekcije infracrvenih senzora. Testiranje će se izvoditi na poligonu s preprekama po kojem će se mobilna robotska platforma kretati.

Kontrolirano kretanje mobilne robotske platforme preko tipkovnice daje uvid o brzini robota na poligonu i omogućuje optimiziranje parametra prije realizacije autonomnog kretanja. Robot se postavlja na poligon i pokreću se Master čvor i čvorovi *remote_controller* te *motors_controller* u terminalima.



Slika 5.1 Poligon za testiranje robota



Slika 5.2 Ispisi u terminalu *remote_controller* i *motors_controller* čvora

Prilikom testiranja ustanovljeno je kako se mobilna robotska platforma dovoljno dobro kreće poligonom kada se metodi *ChangeDutyCycle* proslijedi vrijednost 15 za kretanje unaprijed i unazad, a za kretanje ulijevo i udesno 17 s obzirom na to kako je poželjno da pri detekciji prepreke mobilna robotska platforma nešto brže reagira promjenom smjera ne bi li došlo do kolizije. Komunikacija između *motors_controller* i *remote_controller* čvorova je uspješna i robot pravovremeno reagira na pritisnutu tipku s tipkovnice. Za male vrijednosti brzine mobilna robotska platforma ima poteškoća s kretanjem uzrokovano trenjem površine po kojoj se kreće.

Testiranje autonomnog kretanja mobilne robotske platforme provest će se kroz 10 krugova gdje se u svakom krugu nasumično mijenja raspored prepreka na poligonu. Uzet će se uzorak od 20 prepreka i izračunati uspješnost robota da ih detektira i pravovremeno reagira. Prvo će se ispitati autonomno kretanje robota korištenjem samo jednog od senzora, a zatim kombinacijom obaju senzora. Rezultati će se usporediti u tablicama.

Krug	Efikasnost (%)
1.	55
2.	60
3.	65
4.	45
5.	70
6.	60
7.	50
8.	40
9.	50
10.	55

Tablica 5.1 Efikasnost ultrazvučnog senzora

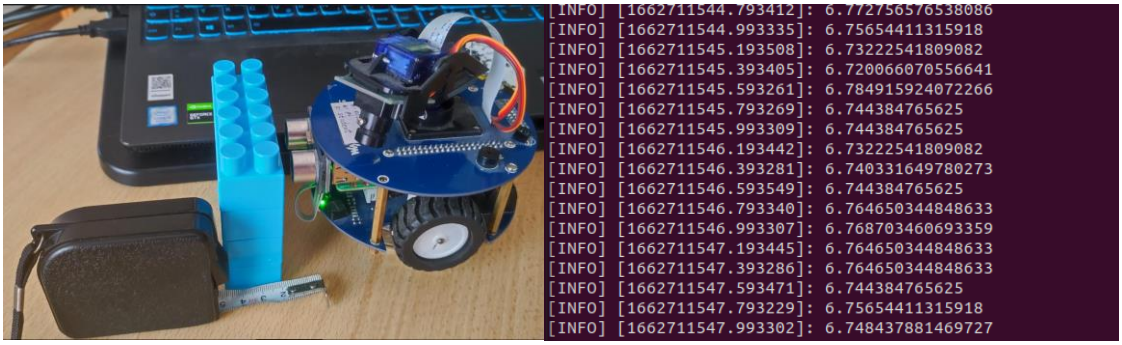
Krug	Efikasnost (%)
1.	75
2.	80
3.	75
4.	80
5.	50
6.	70
7.	85
8.	90
9.	85
10.	95

Tablica 5.2 Efikasnost infracrvenih senzora

Krug	Efikasnost (%)
1.	80
2.	85
3.	90
4.	90
5.	100
6.	100
7.	90
8.	95
9.	90
10.	80

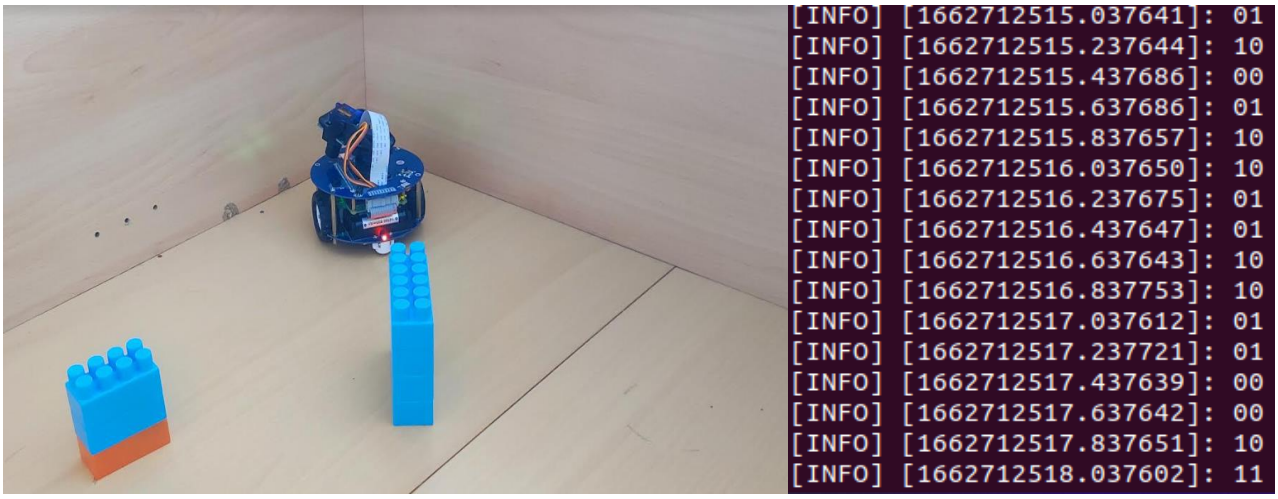
Tablica 5.3 Efikasnost ultrazvučnog senzora i infracrvenih senzora

Testiranje je ukazalo na to kako ultrazvučni senzor ima problema s računanjem udaljenosti rubova poligona, kao i zida kada nije u ravnini robotovog kretanja. Za prepreke koje se pojave sa strane robota, senzor ne uspijeva reagirati. Također, za određene prepreke koje se pojave u neposrednoj blizini robota ili djelomično ispred, senzor ne iščitava točne vrijednosti. Vrijednost kritične udaljenosti ne smije biti premala jer senzor ima poteškoća s računanjem udaljenosti za manje vrijednosti.



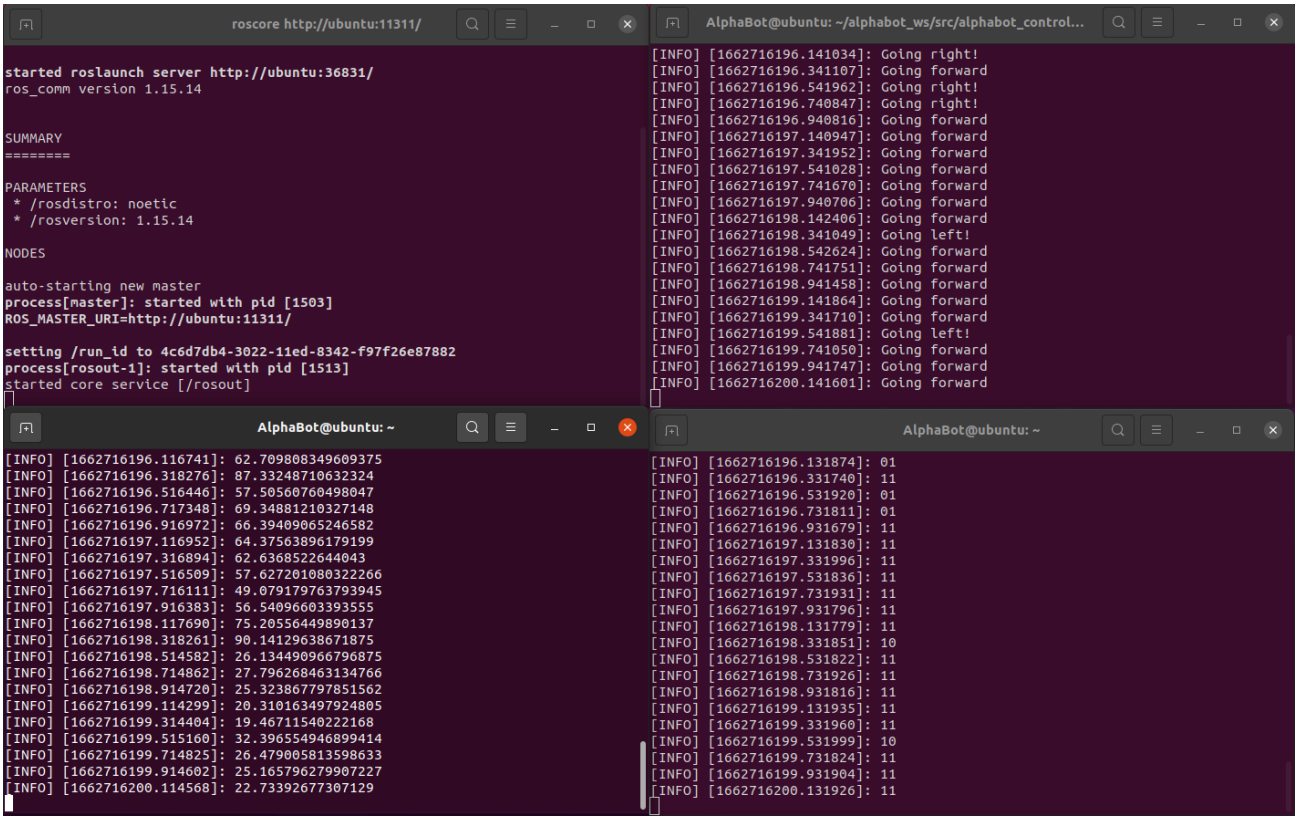
Slika 5.3 Greška prilikom računanja udaljenosti ultrazvučnog senzora

Potencijalno rješenje je povećati vrijednost kritične udaljenosti što daje robotu više vremena za reakciju i mogućnost izračuna točne vrijednosti, predlaže se vrijednost između 10 i 12 cm. ST188 senzori zadovoljavajuće prolaze poligonom, ali isto nailaze na probleme pri detekciji rubova ili snalaženju između dvije prepreke zato što oba senzora u tom trenutku iščitavaju vrijednosti i ne može se donijeti odluka o promjeni smjera.



Slika 5.4 Greška prilikom određivanja smjera kretanja za rubove

Prednost u usporedbi s ultrazvučnim senzorom je što motori primaju informaciju o preprekama s lijeve i desne strane, a povećanjem dometa infracrvenih senzora može se detektirati i prepreka ispred robota. Kada se optimiziraju vrijednosti kritične udaljenosti i dometa infracrvenih senzora tako da se infracrvenim senzorima postave vrijednosti dometa približno jednako vrijednosti kritične udaljenosti, mobilna robotska platforma se najbolje kreće poligonom. Detekcija rubova znatno se poboljšala zajedničkim radom obaju senzora i robotska platforma uspijeva donijeti odluku o kretanju.



Slika 5.5 Ispisi aktivnih čvorova u procesu autonomnog kretanja koristeći oba senzora

6. ZAKLJUČAK

Prilikom izrade diplomskog rada ustanovljeno je kako su na tržištu dostupna brojna rješenja gotovih robotskih platformi različitih funkcionalnosti i karakteristika ovisnih o motorima, senzorima i ostalim hardverskim komponentama koje se nalaze na robotu. AlphaBot-2 Pi mobilna je robotska platforma temeljena na Raspberry Pi računalu obogaćena senzorima za detekciju objekata i autonomno kretanje robota, stoga je razrada praktičnog dijela zadatka išla u smjeru razvoja programske podrške za kontrolirano i autonomno kretanje robotske platforme uz pomoć tipkovnice i senzora. Za realizaciju autonomnog kretanja koristio se ultrazvučni senzor HC-SR04 te infracrveni senzor ST188 koji kombiniranim radom mogu pravovremeno proslijediti motorima i kotačićima informaciju o detekciji i udaljenosti objekta kako bi mobilna platforma promjenom brzine i smjera kretanja pravovremeno reagirala i izbjegla prepreku. Instalacijom Ubuntu operacijskog sustava i konfiguracijom radne okoline na Raspberry Pi računalu može se implementirati programska podrška za rad priključenih senzora i uspostaviti međusobna komunikacija između zasebnih procesa u sustavu kako bi se realizirale funkcionalnosti kontroliranog i autonomnog kretanja robota. ROS je okvir za razvoj programske podrške u robotici i omogućuje uspostavljanje komunikacije između procesa u sustavu robotskog kretanja. Sadrži brojne pakete i alate koji olakšavaju razvoj rješenja za robota. Instalacijom ROS Noetic Ninjemys okvira, najnovije distribucije ROS 1.0 okvira, na Raspberry Pi računalo uspostavljena je radna okolina i ostvarena podrška za potrebnu komunikaciju između hardverskog i softverskog dijela sustava. Praktični dio zadatka započeo je implementiranjem podrške za rad motora i kotačića, a implementacijom rješenja za kontrolirano kretanje robota korisnikovim unosom tipke s tipkovnice testirani su i podešeni parametri brzine i smjera kretanja robota na optimalne vrijednosti prije integracije autonomnog kretanja. Autonomno kretanje robotske platforme zasniva se na međusobnoj komunikaciji motora i senzora gdje senzori objavljuju informaciju na temu na koju je čvor za rukovanje brzinom i smjerom kretanja motora pretplaćen. Čvor za rukovanje motorima i kotačićima će na temelju dobivene informacije od ultrazvučnog i infracrvenog senzora pravovremeno konfigurirati brzinu i smjer kretanja mobilne robotske platforme ako se nalazi u neposrednoj blizini prepreke.

LITERATURA

[1] https://www.waveshare.com/wiki/AlphaBot2-Pi - AlphaBot2-Pi, Lipanj 2022.

[2] https://opensource.com/resources/raspberry-pi - Raspberry Pi, Lipanj 2022.

[3] https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/ - HC-SR04 ultrazvučni senzor, Lipanj 2022.

[4] http://www.npnec.com/en_pdf/ST188-EN.pdf - ST188 infracrveni senzor, Lipanj 2022.

[5] http://wiki.ros.org/ROS/Introduction - Uvod u ROS, Lipanj 2022.

[6] https://www.theconstructsim.com/history-ros/ - Povijest ROS-a, Lipanj 2022.

[7] http://wiki.ros.org/ROS/Concepts - Koncepti ROS-a, Lipanj 2022.

[8] https://subscription.packtpub.com/book/hardware-&-creative/9781788478953/1/ch01lvl1sec13/understanding-the-ros-filesystem-level – ROS datotečni sustav, Lipanj 2022.

[9] https://subscription.packtpub.com/book/hardware-&-creative/9781788478953/1/ch01lvl1sec14/understanding-the-ros-computation-graph-level - ROS razina nadzora i upravljanja, Lipanj 2022.

[10] https://ubuntu.com/blog/what-is-an-ubuntu-lts-release - Ubuntu LTS, Srpanj 2022.

[11] https://varhowto.com/ros-noetic/ - ROS Noetic, Srpanj 2022.

[12] http://wiki.ros.org/noetic/Installation/Ubuntu - ROS Noetic instalacija, Srpanj 2022.

[13] http://wiki.ros.org/catkin - Catkin, Srpanj 2022.

[14] http://wiki.ros.org/msg - ROS tipovi poruka , Srpanj 2022.

[15] http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29 - ROS izrada čvora objavljivača i pretplatnika, Srpanj 2022.

[16] http://wiki.ros.org/message_filters - ROS message_filters, Kolovoz 2022.

SAŽETAK

Cilj ovog diplomskog rada je prikazati postupak izrade programskog rješenja za kontrolirano i autonomno kretanje AlphaBot2-Pi mobilne robotske platforme uz korištenje ROS Noetic Ninjemys okvira. Prije instalacije odgovarajuće verzije ROS-a na Raspberry Pi i same razrade programskog rješenja, potrebno je upoznati se s karakteristikama mobilne robotske platforme, Raspberry Pi računala i senzora kako bi pravilno postavili radno okruženje i omogućili uspješnu komunikaciju procesa u robotskom sustavu. Upoznali smo se s povijesti ROS-a i njegovim konceptima gdje se steklo znanje o elementima sustava, njihovim ulogama i što ROS predstavlja u robotskoj zajednici. U praktičnom dijelu rada izradila se programska podrška za kontrolirano kretanje robota preko tipkovnice, kao i autonomno kretanje realizirano kombinacijom HC-SR04 ultrazvučnog senzora i ST188 infracrvenog senzora te je stečeno znanje o konceptima ROS-a primijenjeno na konkretnom primjeru.

Ključne riječi: AlphaBot-2 Pi, Raspberry Pi, ROS, Ubuntu

ABSTRACT

The goal of this master's paper is to show the process of creating a programming solution for controlled and autonomous movement of the AlphaBot2-Pi mobile robotic platform with the use of the ROS Noetic Ninjemys framework. Before the installation of the appropriate version of the ROS onto the Raspberry Pi and the development of the programming solution itself, it is required to familiarise oneself with the characteristics of the mobile robotic platform, the Raspberry Pi computer, and the sensors, to properly set up the work environment and enable a successful communication between the processes in a robotic system. We have familiarised ourselves with the history of ROS and its concepts, where we have acquired knowledge about the system elements, their roles, and what they mean in a robotic community. In the practical part of the paper, a programming support for controlled movement of the robot with the use of a keyboard was created, as well as autonomous movement implemented through the combined use of an HC-SR04 ultrasound sensor and the ST188 infrared sensor, and the acquired knowledge of the ROS concepts was applied on a concrete example.

ŽIVOTOPIS

Stjepan Miličić rođen je 30.04.1998. u Zagrebu. Nakon završetka Osnovne Škole Mitnica Vukovar, upisuje opći smjer u Gimnaziji Vukovar. 2016. godine upisuje Fakultet Elektrotehnike, Računarstva i Informacijskih Tehnologija Osijek te je trenutno 2. godina diplomskog studija Robotika i umjetna inteligencija.

PRILOZI

motors_controller.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64,String
import RPi.GPIO as GPIO
import time
import message_filters

class AlphaBot(object):
        def __init__(self,ain1=12,ain2=13,ena=6,bin1=20,bin2=21,enb=26):
                self.AIN1 = ain1
                self.AIN2 = ain2
                self.BIN1 = bin1
                self.BIN2 = bin2
                self.ENA = ena
                self.ENB = enb
                self.PA  = 15
                self.PB  = 15

                GPIO.setmode(GPIO.BCM)
                GPIO.setwarnings(False)
                GPIO.setup(self.AIN1,GPIO.OUT)
                GPIO.setup(self.AIN2,GPIO.OUT)
                GPIO.setup(self.BIN1,GPIO.OUT)
                GPIO.setup(self.BIN2,GPIO.OUT)
                GPIO.setup(self.ENA,GPIO.OUT)
                GPIO.setup(self.ENB,GPIO.OUT)
```

```
        self.PWMA = GPIO.PWM(self.ENA,500)
        self.PWMB = GPIO.PWM(self.ENB,500)
        self.PWMA.start(self.PA)
        self.PWMB.start(self.PB)
        self.stop()

    def forward(self):
        self.PWMA.ChangeDutyCycle(self.PA)
        self.PWMB.ChangeDutyCycle(self.PB)
        GPIO.output(self.AIN1,GPIO.LOW)
        GPIO.output(self.AIN2,GPIO.HIGH)
        GPIO.output(self.BIN1,GPIO.LOW)
        GPIO.output(self.BIN2,GPIO.HIGH)

    def stop(self):
        self.PWMA.ChangeDutyCycle(0)
        self.PWMB.ChangeDutyCycle(0)
        GPIO.output(self.AIN1,GPIO.LOW)
        GPIO.output(self.AIN2,GPIO.LOW)
        GPIO.output(self.BIN1,GPIO.LOW)
        GPIO.output(self.BIN2,GPIO.LOW)

    def backward(self):
        self.PWMA.ChangeDutyCycle(self.PA)
        self.PWMB.ChangeDutyCycle(self.PB)
        GPIO.output(self.AIN1,GPIO.HIGH)
        GPIO.output(self.AIN2,GPIO.LOW)
        GPIO.output(self.BIN1,GPIO.HIGH)
        GPIO.output(self.BIN2,GPIO.LOW)
```

```
def left(self):
        self.PWMA.ChangeDutyCycle(17)
        self.PWMB.ChangeDutyCycle(17)
        GPIO.output(self.AIN1,GPIO.HIGH)
        GPIO.output(self.AIN2,GPIO.LOW)
        GPIO.output(self.BIN1,GPIO.LOW)
        GPIO.output(self.BIN2,GPIO.HIGH)

def right(self):
        self.PWMA.ChangeDutyCycle(17)
        self.PWMB.ChangeDutyCycle(17)
        GPIO.output(self.AIN1,GPIO.LOW)
        GPIO.output(self.AIN2,GPIO.HIGH)
        GPIO.output(self.BIN1,GPIO.HIGH)
        GPIO.output(self.BIN2,GPIO.LOW)

def setPWMA(self,value):
        self.PA = value
        self.PWMA.ChangeDutyCycle(self.PA)

def setPWMB(self,value):
        self.PB = value
        self.PWMB.ChangeDutyCycle(self.PB)
```

```
def remote_movement(pressed_key: String):
        if pressed_key.data == 'w':
                Ab.forward()
                rospy.loginfo("Forward")
        elif pressed_key.data == 'a':
                Ab.left()
                time.sleep(0.5)
                Ab.forward()
                rospy.loginfo("Left")
        elif pressed_key.data == 's':
                Ab.backward()
                rospy.loginfo("Backward")
        elif pressed_key.data == 'd':
                Ab.right()
                time.sleep(0.5)
                Ab.forward()
                rospy.loginfo("Right")
        elif pressed_key.data == ' ':
                Ab.stop()
                rospy.loginfo("Motors are stopped")

def react_to_distance(dist: Float64):
    if int(dist.data) <= 12:
        Ab.right()
        rospy.loginfo("Right")
    else:
        Ab.forward()
        rospy.loginfo("Forward")

def move_robot(us_dist, ir_det):
```

```
    if int(us_dist.data) <= 12 or ir_det.data == '01':
        Ab.right()
        rospy.loginfo("Going right!")
    elif ir_det.data == '10':
        Ab.left()
        rospy.loginfo("Going left!")
    elif ir_det.data == '00':
        Ab.backward()
        rospy.loginfo("Going backwards")
    else:
        Ab.forward()
        rospy.loginfo("Going forward")

if __name__ == '__main__':
    rospy.init_node("motors_controller")
    Ab = AlphaBot()
    remote_sub = rospy.Subscriber("remote/control/input", String, callback=remote_movement)
    us_sub = message_filters.Subscriber('ultrasonic/distance', Float64)
    ir_sub = message_filters.Subscriber('infrared/obstacle', String)
    ts = message_filters.ApproximateTimeSynchronizer([us_sub, ir_sub], 10, 0.1, allow_headerless=True)
    ts.registerCallback(move_robot)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        rate.sleep()
```

remote_controller.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String

class _GetchUnix:
    def __init__(self):
        import tty, sys

    def __call__(self):
        import sys, tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch
```

```
class Remote:
    def keyboard_input(self):
        rospy.loginfo_once("\nw - forward\na - left\ns - backward\nd - right\nspace - stop\n\nq - exit\n----------------")
        key_pressed = _GetchUnix().__call__()
        while key_pressed == 'w' or key_pressed == 'a' or key_pressed == 's' or key_pressed == 'd' or key_pressed == ' ':
            return str(key_pressed)
        if key_pressed == 'q':
            raise KeyboardInterrupt


if __name__ == '__main__':
    rospy.init_node("remote_controller")
    remote_cntrl = Remote()
    remote_pub = rospy.Publisher("remote/control/input",String,queue_size=10)
    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        pressed_key = remote_cntrl.keyboard_input()
        rospy.loginfo(pressed_key)
        remote_pub.publish(pressed_key)
        rate.sleep()
```

ultrasonic_sensor_controller.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float64
import RPi.GPIO as GPIO
import time

class Sonar(object):
    def __init__(self, trig=22, echo=27):
        self.TRIG = trig
        self.ECHO = echo
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.TRIG,GPIO.OUT,initial=GPIO.LOW)
        GPIO.setup(self.ECHO,GPIO.IN)

    def distance(self):
        GPIO.output(self.TRIG,GPIO.HIGH)
        time.sleep(0.000015)
        GPIO.output(self.TRIG,GPIO.LOW)
        while not GPIO.input(self.ECHO):
            pass
        t1 = time.time()
        while GPIO.input(self.ECHO):
            pass
        t2 = time.time()
        return (t2-t1)*34000/2
```

```
if __name__ == '__main__':
    rospy.init_node("ultrasonic_sensor_controller")
    rospy.loginfo("Ultrasonic sensor is engaged")
    us_sensor = Sonar()
    us_pub = rospy.Publisher("ultrasonic/distance", Float64, queue_size=10)
    rate = rospy.Rate(5)
    while not rospy.is_shutdown():
        dist = us_sensor.distance()
        rospy.loginfo(dist)
        us_pub.publish(dist)
        rate.sleep()
```

infrared_sensors_controller.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
import RPi.GPIO as GPIO
import time

class IRSensors(object):

    def __init__(self, dr=16, dl=19):
        self.DR = dr
        self.DL = dl
        GPIO.setmode(GPIO.BCM)
        GPIO.setwarnings(False)
        GPIO.setup(self.DR,GPIO.IN,GPIO.PUD_UP)
        GPIO.setup(self.DL,GPIO.IN,GPIO.PUD_UP)

    def detect_obstacle(self):
        DR_status = GPIO.input(self.DR)
        DL_status = GPIO.input(self.DL)
        return str(DR_status) + str(DL_status)
```

```
if __name__ == '__main__':
    rospy.init_node("infrared_sensors_controller")
    rospy.loginfo("Infrared sensors are engaged")
    ir_sensor = IRSensors()
    ir_pub = rospy.Publisher("infrared/obstacle",String,queue_size=10)
    rate = rospy.Rate(5)
    while not rospy.is_shutdown():
        obj_detected = ir_sensor.detect_obstacle()
        rospy.loginfo(obj_detected)
        ir_pub.publish(obj_detected)
        rate.sleep()
```