Primjena ROS operativnog sustava na Android platformi

Ščuric, Vladimir

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

Sveučilišni prijediplomski studij Računarstvo

PRIMJENA ROS OPERATIVNOG SUSTAVA NA ANDROID PLATFORMI

Završni rad

Vladimir Ščuric

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P: Obrazac za ocjenu završnog rada na sveučilišnom prijediplomskom studiju

Ocjena završnog rada na sveučilišnom prijediplomskom studiju

Ime i prezime pristupnika:	Vladimir Ščuric
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. pristupnika, god.	2325, 26.09.2007.
JMBAG:	0165033793
Mentor:	prof. dr. sc. Damir Blažević
Sumentor:	
Sumentor iz tvrtke:	Domagoj Vuković
Naslov završnog rada:	Primjena ROS operativnog sustava na Android platformi
Znanstvena grana završnog rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada:	

Datum prijedloga ocjene završnog rada od strane mentora:	17.09.2024.
Prijedlog ocjene završnog rada od strane mentora:	Izvrstan (5)
Datum potvrde ocjene završnog rada od strane Odbora:	25.09.2024.
Ocjena završnog rada nakon obrane:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije završnog rada čime je pristupnik završio sveučilišni prijediplomski studij:	30.09.2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Vladimir Ščuric
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina upisa:	2325, 26.09.2007.
Turnitin podudaranje [%]:	9

Ovom izjavom izjavljujem da je rad pod nazivom**: Primjena ROS operativnog sustava na Android platformi**

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

i sumentora

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

Potpis pristupnika:

SADRŽAJ

1. UVOD

Operativni sustav je program, tj. skup programa koji upravljaju svim uređajima u računalu/sustavu. Glavni razlog nastanaka je lakša i brža odrada zahtjeva koji su pruženi sustavu od strane korisnika.

Daljnjim razvojem tehnologije i pojavom robota, razvila se i veća potražnja za automatiziranim robotskim sustavima. Kako je za svaki sustav posebno kreiran operativni sustav razvila se ideja o stvaranju jedinstvene programske platforme pod imenom Robotski operativni sustav. Isti omogućuje komunikaciju između korisnika i robota te međusobnu komunikaciju robota. Prednost navedenog operativnog sustava leži u tome što je isti dostupan svima u obliku otvorenog koda.

U prvom poglavlju je odrađena analiza robotskog sustava u kojoj se može pročitati više o povijesnom razvoju robota, razvoju Robotskog operativnog sustava te njegovoj arhitekturi, alatima kao i načinu primjene ROS-a.

Drugo poglavlje daje detaljan opis Robotskog operativnog sustava na Android platformi pojašnjenjem Rosjava okruženja i njegove pripreme za izradu Android aplikacije kao i samu primjenu.

Treće poglavlje se dotiče povijesti *Rosjava* okruženja i njegovih mogućnosti te je napravljena priprema okruženja za razvoj aplikacije.

Dok u četvrtom poglavlju slijedi prikaz primjene Robotskog operativnog sustava na primjeru implementacije upravljanja modulom kamere i obrada podataka dobivenih s akcelerometra. Naposlijetku slijedi zaključak sa smjernicama za daljnji rad i popis korištene literature.

1.1 Zadatak završnog rada

Zadatak ovog završnog rada je napraviti pregled dostupnih tehnologija, inačica i mogućnosti primjene Robotskog operativnog sustava na *Android* platformi. U prvom dijelu rada

predstaviti će se povijest robota, Robotski operativni sustav i njegove inačice, dok će u drugom dijelu rada biti predstavljena rješenja za *Android* platformu te prikaz namjene.

2. ANALIZA ROBOTSKOG SUSTAVA

2.1 Pregled područja

Robotski operativni sustav razvijen je 2007. godine pod imenom *Switchyard* i iste godine formirana je prva stabilna verzija pod nazivom *ROS Box Turtle.* Distribucije ROS-a su u pravilu setovi ROS paketa srodne *Linux* distribucijama. Pušteno je ukupno devet verzija s podrškom za različite inačice Ubuntu operativnog sustava. Posljednja inačica pojavila 23. svibnja 2020. godine.

Izuzev navedenog, godine 2014. je započeo razvoj druge verzije robotskog operativnog sustava (ROS 2) na čijem razvoju se još uvijek aktivno radi. Za razliku od prve verzije gdje je nova inačica izlazila svakih godinu dana, u drugoj verziji nove inačice su izlazile svakih 6 mjeseci no i to se promijenilo te i one sada izlaze jednom godišnje. Zadnja stabilna inačica *Jazzy Jalisco* je postala dostupna 23. svibnja 2024. godine.

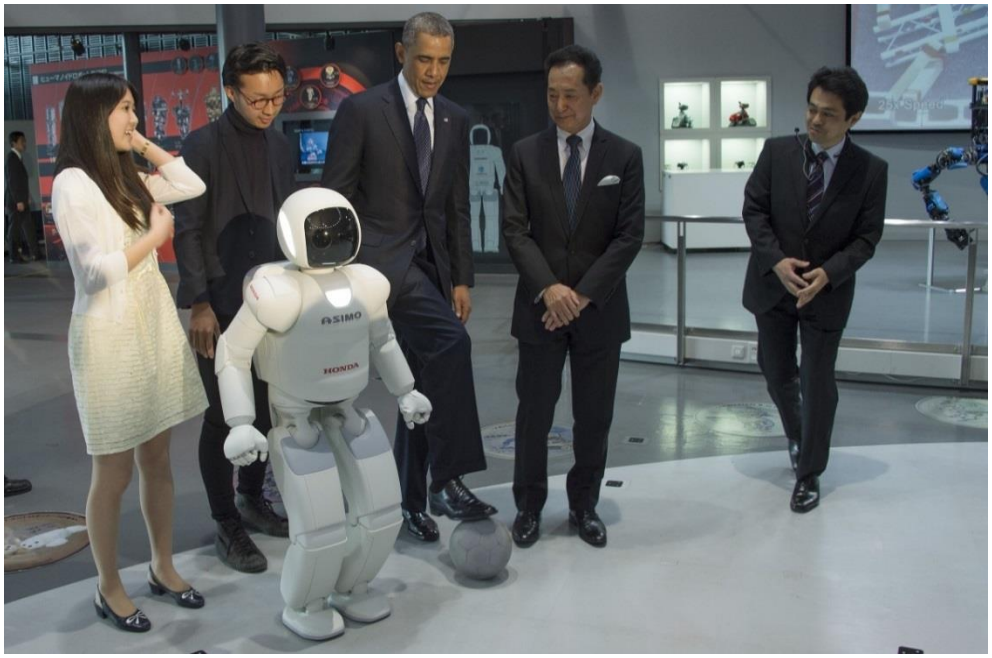
Iako je vodeći sustav otvorenog koda, nekoliko drugih alata pružaju slične funkcionalnosti poput VPL-a (*Virtual Programming Language*), koji je razvila *Open Source Robotics Foundation*, a naglasak stavlja na grafički pristup programiranju. Uz njih postoji i MRDS (*Microsoft Robotics Developer Studio*) koje je razvio *Microsoft*, a koji također nudi podršku za grafički pristup programiranju.

Osim sustava otvorena koda, tvtke poput *NVIDIA* i *Google* razvila su vlastita komercijalna rješenja. Primjerice *NVIDIA*-in *Isaac SDK* kombinira robotske mogućnosti s onima umjetne inteligencije. Dok *Google*-ov API pruža alate za integraciju aplikacija strojnog učenja i robotike. Također vrijedi spomenuti kako *Robot Operating System Industrial* proširuje mogućnosti Robotskog operativnog sustava u industrijskoj domeni, a u razvoj su uključene tvrtke poput *Siemensa* i *Boscha*.

2.2 Razvoj robota kroz povijest

Robot je mehaničko ili umjetno virtualno sredstvo, najčešće elektromehanički stroj kojim upravlja računalni program ili elektronički sklop izrađen da izvrši jednu ili više radnji istodobno s velikom brzinom i preciznošću. Roboti mogu biti autonomni ili polu-autonomni te obuhvaćaju humanoide poput Hondinog *Advanced Step in Innovative Mobility* (ASIMO) (Slika 2.1.),

industrijske robote, robote za medicinske operacije, kolektivno programirane *swarm* robote, pa čak i mikroskopske nano robote [1].



Slika 2.1. *ASIMO (Izvor: https://d.ibtimes.co.uk/en/full/1375474/president-barack-obama-plays-football-hondas-humanoid-robot-asimo-state-visit-japan.jpg)*

Prva vozila na daljinsko upravljanje pojavljuju se u kasnom 19-om stoljeću u obliku nekoliko vrsta torpeda na daljinsko upravljanje. 1897 godine Britanskom izumitelju Ernestu Wilsonu dodijeljen je patent za torpedo na daljinsko upravljanje preko radio valova, a 1898 godine Nikola Tesla javno je demonstrirao torpedo na bežično upravljanje [2].

Izraz „robot“ prvi puta se pojavljuje u noveli iz 1920. godine češkog spisatelja Karela Čapeka. Jedan od prvih humanoidnih robota predstavljen je 1928. godine na godišnjoj izložbi *Model Engineers Society* u Londonu. Izloženi robot sastojao se od aluminijskog tijela s 12 elektromagneta i jednim motorom pokretanim s 12-voltnim napajanjem. Robot je mogao pomicati ruke i glavu, te se mogao kontrolirati preko daljinskog upravljača ili glasovno [1].

Prvi digitalno upravljani i programabilan robot pod nazivom *Unimate* izumio je George Devol 1954 godine, što je postavilo temelje za modernu industriju robotike. Prvi *Unimate* korišten je u tvornici u Trentonu, New Jersey za podizanje i slaganje vrelih komada metala iz lijevnog stroja [3].

Komercijalni i industrijski roboti su danas u širokoj upotrebi obavljajući poslove jeftinije ili s većom preciznošću i pouzdanošću od ljudi. Roboti se također koriste za poslove koji se smatraju „prljavima“, preopasnima ili prejednoličnim za ljude [1].

2.3 Razvoj ROS-a i pregled njegovih inačica

Robotski operativni sustav razvijen je 2007. godine pod imenom *Switchyard*. Isti je razvijen od strane *Stanford Artificial Intelligence Laboratory* kao podrška *Stanford AI Robot Stair projektu*. U razdoblju od 2008. do 2013. godine, za razvoj ROS-a je primarno zadužena tvrtka *Willow Garage*. Navedena tvrtka je u svojoj biti institut, tj. inkubator za razvoj robotike [4]. Tvrtka je osnovana 2006. godine od strane Scott Hassana kako bi se ubrzao razvoj robota opće namjene te unaprjeđenje razvoja softwarea otvorenog koda [5]. Scott Hassan iza sebe ima golemo iskustvo pošto je bio ključni razvojni inženjer u tvrtkama *Google, Alexa Internet* te *Stanford Digital Library* [6].

Godine 2007. formirana je prva stabilna verzija ROS distribucije pod nazivom *ROS Box Turtle.* Distribucije ROS-a su u pravilu set ROS paketa koje su srodne *Linux* distribucijama. Svrha razvoja ROS distribucija je olakšavanje rad razvojnim inženjerima te uvođenje novih funkcija. Svaka distribucija se sastoji od tri dijela, čiji je prvi osnovni dio (engl. *base*), sastoji se od generičke robotske biblioteke, upravljačkih programa za pojedine uređaje i razvojnih alata. Ostala dva dijela sadrže stabilne i eksperimentalne biblioteke koje su zadužene za upravljanje s Osobnim Robotom.

Sve distribucije ROS-a u javnost se puštaju u svibnju svake godine u obliku otvorenog koda te je nedostatak neusklađenost nove verzije s ranijim inačicama. Nakon prve stabilne verzije pušteno je ukupno devet verzija s podrškom za različite inačice Ubuntu operativnog sustava. Posljednja inačica pojavila 23. svibnja 2020. godine, pod imenom *ROS Noetic Ninjemis* [7]. Navedena distribucija primarno je napravljena za Ubuntu 20.04 Međutim, podržani su, do određene mjere, i ostali operativni sustavi poput *MAC OS, Android te Windows* [6]. Popis izdanih distribucija vidljiv je u tablici 2.1.

Distribucija	Datum izlaska
ROS Noetic Ninjemys	23. svibnja 2020
ROS Melodic Morenia	23. svibnja 2018

ROS Lunar Loggerhead	23. svibnja 2017
ROS Kinetic Kame	23. svibnja 2016
ROS Jade Turtle	23. svibnja 2015
ROS Indigo Igloo	22. srpnja 2014
ROS Hydro Medusa	4. rujna 2013
ROS Groovy Galapagos	31. prosinca 2012
ROS Fuerte Turtle	23. travnja 2012
ROS Electric Emys	30. kolovoza 2011
ROS Diamonback	3. ožujka 2011
ROS C Turtle	4. kolovoza 2010
ROS Box Turtle	2. ožuka 2010

Tablica 2.1 *Pregled ROS distribucija [7]*

Uz ROS 1, paralelno se počela razvijati sljedeća inačica, odnosno ROS 2 te pregled distribucija vidljiv je u tablici 2.2.

Kao kod većine velikih operativnih sustava, ROS 1 i dalje postoji zbog svoje duge povijesti u kojoj se akumulirala velika baza korisnika sustava te bi nagli prekid prekinuo brojne tekuće projekte.

Distribucija	**Datum izlaska**
Jazzy Jalisco	23. svibnja 2024
Iron Irwini	23. svibnja 2023
Humble Hawksbill	23. svibnja 2022
Galactic Geochelone	23. svibnja 2021
Foxy Fitzroy	5. srpnja 2020
Eloquent Elusor	22. studenog 2019
Dashing Diademata	31. svibnja 2019
Crystal Clemmys	14. prosinca 2018
Bouncy Bolson	2. srpnja 2018
Ardent Apalone	8. prosinca 2017
beta3	13. rujna 2017
beta2	5. srpnja 2017

beta1	19. prosinca 2016
alpha1 – alpha8	31. kolovoza 2015

Tablica 2.2 *Pregled ROS 2 distribucija [8]*

Obje verzije razvija i održava *Open Robotics*, a dolaze s različitim rješenjima i razvojnim ciljevima. ROS 1 pruža fleksibilne alate radi lakšeg razvoja složenih robotskih aplikacija. ROS 2 je stvoren kako bi uklonio ograničenja svog prethodnika uz dodatne mogućnosti poput rada u stvarnom vremenu, bolje sigurnosti i kompatibilnosti s više platformi.

2.4 Arhitektura Robotskog operativnog sustava

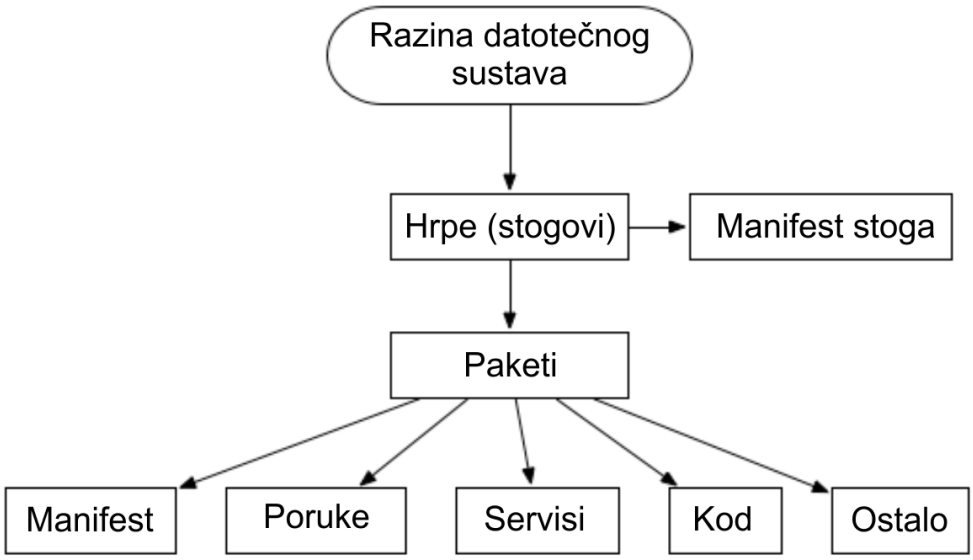
Arhitektura Robotskog operativnog sustava (Slika 2.2.) izrađena je u tri razine:

- Razina datotečnog sustava
- Razine računalne grafike
- Razine inženjerske okoline.

Prva razina prikazuje standarde i pravila koji formiraju ROS kao strukturu osnovnog direktorija koji su potrebni kako bi se sustav mogao koristiti. Druga razina prikazuje protokole i koncepte koje ROS koristi za postavljanje programa, upravljanje procesima te komunikaciju s više računala. Treća razina omogućuje komunikaciju između razvojnog inženjera i ROS-a čime se postiže efikasniji i kvalitetniji razvoj Robotskog operativnog sustava.

Slika 2.2. *Struktura Robotskog operativnog sustava*

Datotečni sustav podijeljen je u datoteke s podacima koji sadrže svrhu i namjenu istih. Paketi su najniža razina Robotskog operativnog sustava te sadrže dovoljno informacija i podataka za kreiranje programa koji koristi robot. Unutar paketa su sadržani manifesti, tipovi poruka, servisi i programski kodovi (Slika 2.3.).



Slika 2.3. *Datotečni sustav ROS-a (Razina datotečnog sustava (eng. Filesystem level); Hrpe (eng. Stacks); Manifest stoga (eng. Stack Manifest); Paketi (eng. Packages); Manifest (eng. Manifest), Poruke (eng. Messages); Servisi (eng. Services); Kod (eng. Code); Ostalo (eng. Others))*

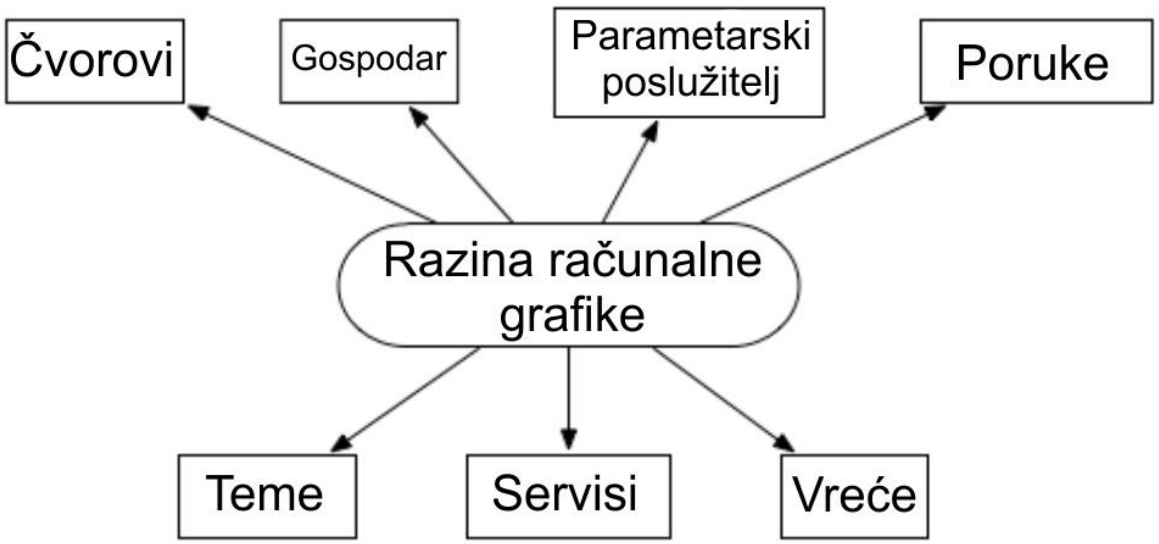
Paketi su datoteke ROS-a te svaki paket unutar sebe mora sadržavati datoteku *manifest.xml*. Navedena datoteka služi za prikaz informacija navedenog paketa. Ako se manifest nalazi u direktoriju tada taj direktorij predstavlja paket. Unutar manifesta nalaze se dvije oznake: ovisnost i izvoz. Ovisnost prikazuje koji paketi moraju biti prethodno instalirani. Izvoz pokazuje sustavu koje će zaglavlje i zastati koristiti prilikom kompiliranja paketa.

Stack u kontekstu ROS-a ima drugačije značanje i funkciju u odnosu na klasični programski kontekst. U klasičnom programskom kontekstu *Stack* predstavlja spremanje varijabli i funkcija sa svim njihovim parametrima dok se u ovom slučaju koriste za grupiranje paketa s određenom funkcionalnošću. Unutar samog ROS-a postoji iznimno velik broj hrpa za različite svrhe, a kao jedan od primjer se može navesti hrpa za navigaciju.

Hrpe se stvaraju korištenjem naredbe *roscreate-stack*. Hrpe sadrže tri datoteke *CMakeList.txt, MakeFile* i *stack.xml*. Direktorij se može nazvati hrpom ukoliko se u njemu nalazi *stack.xml* [9].

Poruke se objavljuju putem ROS čvorova u obliku podataka. Robotski operativni sustav sadrži predefinirane poruke, ali prema potrebi može se izraditi pojedinačna poruka koja se zatim sprema u *msg/* direktorij. Poruka se sastoji od dva dijela, polja i konstanti. Polja definiraju tip podatka, a konstante imena polja. Specifična vrsta poruke je zaglavlje koje se koristi za unos dodatnih podataka poput vremena stvaranja poruke, podataka iz okvira i sl. Pomoću zaglavlja može se pratiti broj poruka između čvorova kao i podatke o vlasniku poruke.

Strukturu zahtjeva i odziv Robotskog operativnog sustava definiraju servisi. Servisi omogućuju komunikaciju između čvorova povezivanjem na poruke. Kod pozivanja servisa koristi se ime paketa uz ime servisa [10].



Slika 2.4. *Računalna grafika ROS-a (Čvorovi (eng. Nodes); Gospodar (eng. Nodes); Parametarski poslužitelj (eng. Parameter server); Poruke (eng. Messages); Razina računalne grafike (eng. Computation Graph Level); Teme (eng. Topics); Servisi (eng. Services); Vreće (eng. Bags))*

Računalna grafika prikazuje način na koji ROS koristi mrežu u kojoj su svi procesi povezani (Slika 2.4.). Čvorovi komuniciraju s drugim procesima pomoću tema, servisa ili parametarskog poslužitelja. Svaki čvor sadrži jedinstveno ime koje se koristi za komunikaciju između čvorova.

Teme su širokopojasni prozori koji služe za prijenos podataka. Mogu se prenositi između čvorova bez međusobnog kontakta te može sadržavati različiti broj pretplatnika. Za prijenos se koriste TCP/IP te UDP protokoli [11].

Poruke služe za prijenos informacija između čvorova. Čvorovi primaju poruke koje u objavljene u temama koje isti prate. Struktura poruke se sastoji od standardnih tipova podataka i podataka specifične namjene.

Vreće su datoteke stvorene od strane Robotskog operativnog sustava, a u njima se čuvaju podaci o porukama, temama i servisima. Navedenim podacima može se pristupiti u bilo kojem trenutku te provjeriti tijek događanja što omogućuje lakše otklanjanje pogrešaka.

Gospodar služi za dodjeljivanje imena te registraciju servisa prema svim čvorovima u sustavu. Osim toga, prati izdavače i pretplatnike u svakoj temi. Također, bitnu ulogu u tome što

omogućuje pojedinačnim čvorovima međusobno povezivanje. Kada se dva čvora povežu komunikacija se nastala prema metodi ravnopravnih članova [12].

Parametarski poslužitelj je dijeljena i varijabilna datoteka, a pristup istoj je moguć putem mreže. Navedeni poslužitelj koriste čvorovi radi spremanja i dohvaćanja parametara u realnom vremenu.

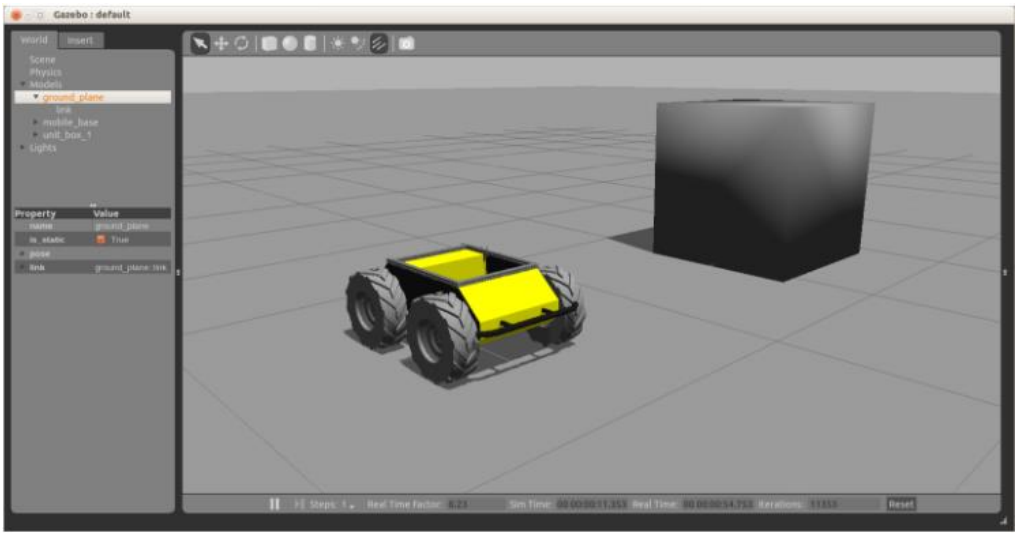
Smisao inženjerske okoline leži u tome da inženjerima koji rade na sličnim projektima, omogući komunikaciju te razmjenu znanja i informacija. Inženjerska okolina sadrži manje cjeline, a svaka od njih je zadužena za pojedino područje. Prva pod cjelina je Distribucija, a ista omogućuje arhiviranje različitih inačica Robotskog operativnog sustava. Arhivirane inačice mogu se pronaći u obliku hrpa [13]. Spremište je druga pod cjelina i temeljena je na konceptu globalnog umreženog spremišta podataka. Na taj način svaka ustanova može pristupiti spremištu unutar kojeg mogu razvijati programe i naknadno ih objavljivati. Treća pod cjelina bi bila službena *Wiki* stranica Robotskog operativnog sustava. Nakon izvršene registracije moguće je pristupiti mnoštvu podataka te dokumenata, dodavati vlastite materijale, pisati ili dopunjavati uputstva za korištenje te ažurirati postojeće materijale.

Osnovni način za komunikaciju između korisnika je popis za slanje poruka putem elektroničke pošte. Popis predstavlja posljednju pod cjelinu, a služi kao svojevrstan oblik Internet foruma.

2.5 Alati dostupni unutar Robotskog operativnog sustava

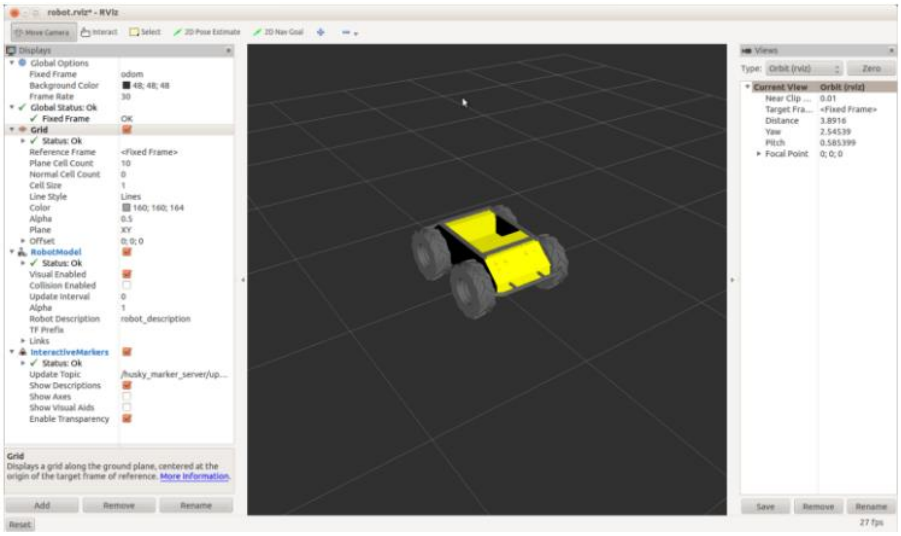
Robotski operativni sustav sadrži određene vrste alata koje se koriste za vizualizaciju robota tj. robotskih sustava.

Pokretanjem virtualnog robota dobivaju se dva prozora gdje svaki predstavlja jedan od alata dostupnih u ROS-u.



Slika 2.5. *Prozor Gazebo alata (Izvor: https://www.clearpathrobotics.com/assets/guides/noetic/ros/_images/Huskysim.png)*

Slika 2.5 prikazuje prozor alata *Gazebo*. *Gazebo* alat predstavlja realnu sliku robotskog sustava s njegovim parametrima. U parametre robotskog sustava ubrajaju se vrijednosti poput veličine i težine robota, koeficijent klizanja, inercije i sl. U *Gazebo*-u se također mogu dodavati objekti u simulaciju, poput prepreke (kocke) prikazane na slici 2.5., ili čak cijele mape stvarnih mjesta.

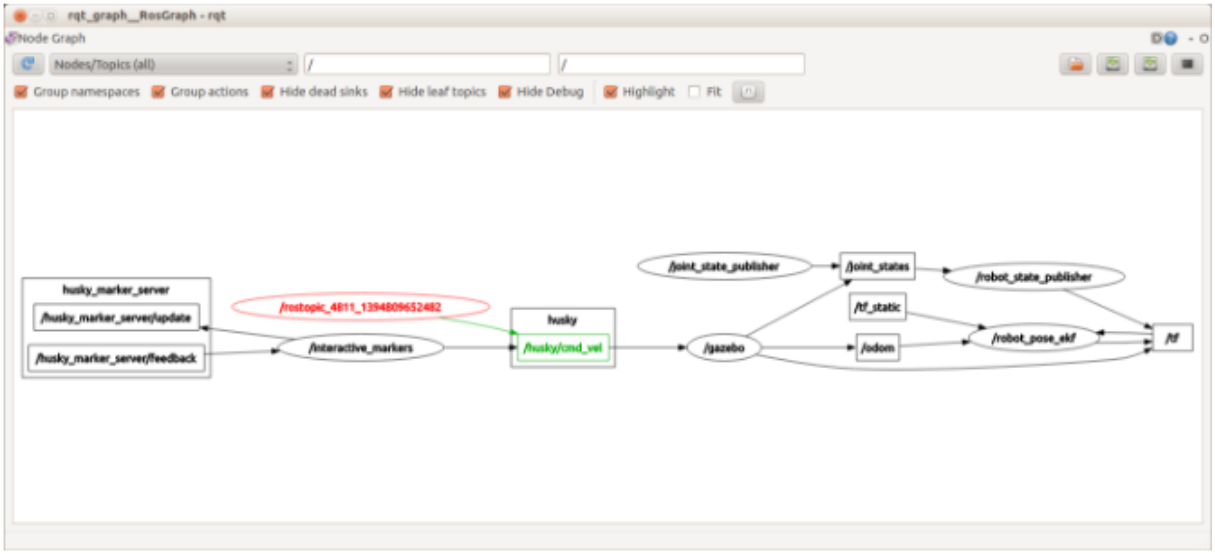


Slika 2.6. *Prozor RZiv alata (Izvor: https://www.clearpathrobotics.com/assets/guides/noetic/ros/_images/Huskyviz.png)*

Slika 2.6. prikazuje prozor alata *RViz*. Ovaj alat omogućuje pregled podataka koju su pruženi od strane senzora robota, te davanje naredbi robotu.

Izvršavanje zadanih naredbi robotu alatom *RViz* prikazano je u prozoru alata *Gazebo* uz vidljiv utjecaj zadanih parametara robotskog sustava poput klizanja kotača i sl.

Prema slici 2.7 moguće je vidjeti strukturu rasporeda tema po sustavu korištenjem naredbe *rqt_graph*. Ova naredba daje prikaz načina povezanosti čvorova i pokrenutih tema u trenutačnom ROS Master-u.



Slika 2.7. *Struktura rasporeda tema*

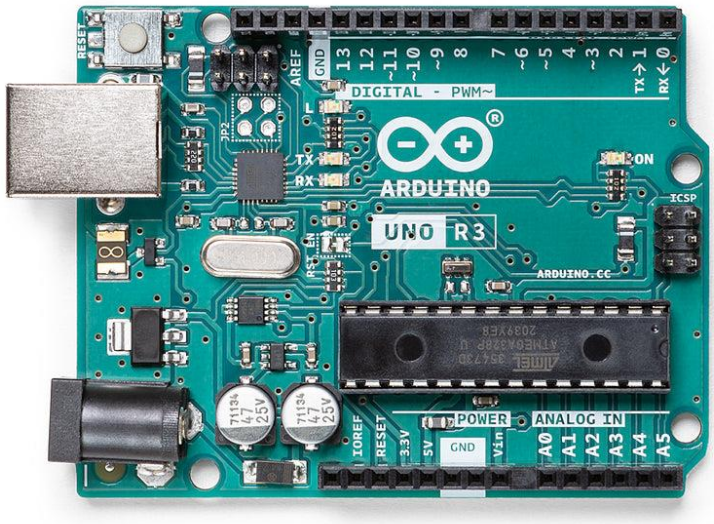
Naglašeni čvorovi i strelice prikazuju temu koja se objavljuje simuliranom robotu. Robot potom ažurira *Gazebo* virtualnu okolinu, čime se osigurava kretanje zglobova (kotača) i fizika robota.

2.6 Korištenje Robotskog operativnog sustava u postojećim rješenjima

Robotski operativni sustav razvijen je na osnovu ideje o stvaranja jedinstvene robotske platforme za programiranje robotskih sustava [14]. Iz tog razloga moguće je upotrebu ROS-a podijeliti u dvije osnovne skupine primjena. Te dvije skupine obuhvaćaju primjenu u fizičkim robotskim sustavima i primjenu unutar drugih programskih platformi. Korištenje ROS-a u fizičkim robotskim sustavima odnosi se na znanstveno-istraživačku i obrazovnu uporabu kao i na uporabu u procesima industrijske proizvodnje.

Osim za potrebe industrije ROS se često koristi i kod izrade specijaliziranih robotskih sustava čiji se program izrađen od strane ROS-a prenosi na druge manje mobilne platforme. Primjer takvog načina korištenja je prenošenje programskog koda na razvojnu ploču *Arduino* [14].

Arduino (Slika 2.8.) se sastoji od niza senzora koji omogućuju interakciju s vanjskom okolinom i na koje može utjecati korištenjem pogonskih mehanizama [15].



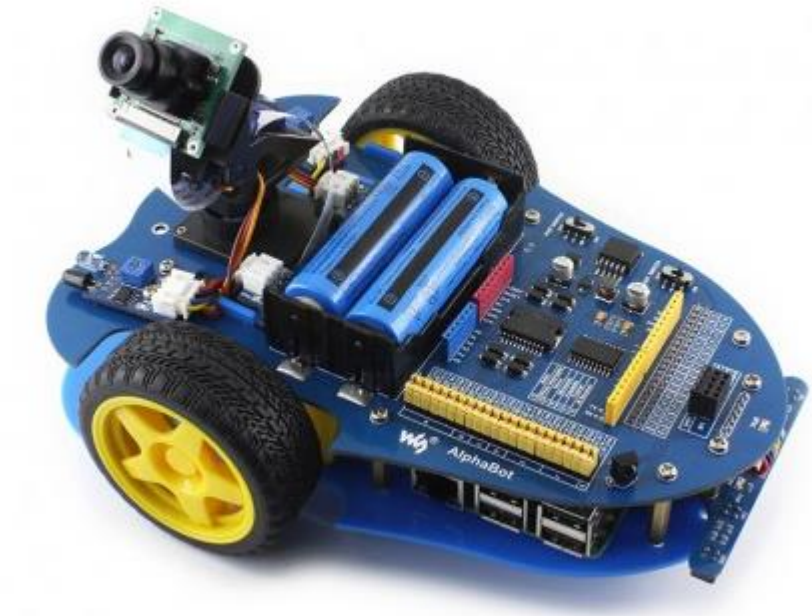
Slika 2.8. *Arduino Uno Rev3 razvojna ploča (Izvor: https://store.arduino.cc/cdn/shop/products/A000066_03.front_804x603.jpg?v=1629815860)*

Jedna od primjena ROS-a je razvoj programskih rješenja kod Lego robota. Lego Mindstorm (Slika 2.9.) serija opreme sadrži programsku podršku i računalno sklopovlje za izradu prilagodljivih, programabilnih robota. Ona uključuje računalo koje kontrolira sustav, set modularnih senzora i motora te Lego dijelove za izradu mehaničkih sustava. Ovakav sustav podržava niz programskih jezika, među kojima je i ROS [16].

Slika 2.9. *Lego Mindstorms EV3 (Izvor: https://www.lego.com/cdn/cs/set/assets/bltc8007894596189e6/Mindstorms-Downloads-EV3_Firmware-Sidekick-Standard.jpg?fit=crop&format=jpg&quality=80&width=800&height=426&dpr=1)*

Također vrijedi spomenuti i *WaveShare AlphaBot Pi* (Slika 2.10.). Riječ je o mobilnoj i modularnoj razvojnoj robotskoj platformi. Platforma je izrazito modularna pošto se komponente vrlo lako priključuju. Neke od komponenti služe za praćenje linija, zaobilaženje prepreka, mjerenje brzine i slično. Platforma dolazi s modulom koji sadržava kameru i bežičnu mrežu pa time omogućava daljinsku kontrolu i praćenje putem mreže [17].

Kako je sama platforma kompatibilna s *Raspberry Pi* razvojnom pločom, na istu je moguće instalirati ROS te putem njega upravljati i samim robotom.



Slika 2.10. *AplhaBot Pi (Izvor: https://www.waveshare.com/media/catalog/product/cache/1/image/560x560/9df78eab33525d08d6e5fb8d27136e95/a/l/alphabot-20.jpg)*

Osim upotrebe u industrijskim robotskim sustavima ROS se može koristiti i unutar različitih programskih rješenja gdje služi kao dodatak preko kojeg se razvijaju programi i aplikacije za rad s robotima.

Jedan od operativnih sustava za koje se ROS može koristiti je i *Android*. *Android* je operativni sustav koji je kreirao Google kao alternativu koja se koristi u uređajima poput telefona i tableta (Slika 2.11.). Sam operativni sustav je temeljen na Linux jezgri [18]. U svrhu lakšeg razvoja programskih rješenja za *Android*, u suradnji s *Willow Garage*, *Google* je razvio posebnu verziju ROS-a pod nazivom *rosjava*.

Slika 2.10 *Upravljanje robota putem Android OS-a*

3. ROBOTSKI OPERATIVNI SUSTAV I ANDROID

3.1 Rosjava okruženje

Prva čista *Java* implementacija Robotskog operativnog sustava je *Rosjava* okruženje. Razvijena je u kooperaciji tvrtke *Google* i *Willow Garage*. *Rosjava* omogućuje integraciju *Android* operativnog sustava s robotima koji koriste Robotski operativni sustav, što znači da je moguće korištenje senzora uređaja s *Android OS* za upravljanje robotom.

Kako je riječ o implementaciji otvorenog koda, kompletan kod se može pronaći na *Github*-u: https://github.com/rosjava. Sam projekt više nije u aktivnom razvoju te je i arhiviran iako je i dalje dostupan za korištenje, a prvi puta je predstavljen na *Cloud Robotics tech talk* u sklopu *Google I/O 2011* [19].

Cilj razvoja okruženja je pružiti razvojnim inženjerima način za izradu *rosjava* paketa, integracija istog s ostalim ROS alatima poput *roslaunch.* Olakšavanje izrade projekata osobama koje nisu stručne u izradi ROS programa. *Rosjava* okruženje koristi mješavinu alata koji su poznati razvojnim inženjerima *Java* rješenja, a to su *Java, Maven, Catkin* [20].

Gradle je sustav za izradu automatizacijskih i robotskih rješenja otvorenog koda koji se nadograđuje na koncepte *Apache Ant* i *Apache Maven.* Dizajniran je za izradu višeprojektnih rješenja [21].

Maven, kao i *Gradle*, služi za izradu automatizacijskih i robotskih rješenja koji se također koristi u *Java* projektima. Baziran je na dva aspekta prilikom izrade softwarea, prvi opisuje kako je projekt izrađen dok drugi opisuje zavisnosti. *Maven* se može koristiti prilikom izrade projekata pisanih u *C#, Ruby, Scala* i drugim jezicima [22].

3.2 Roboti u „oblaku“

Robotika u oblaku je polje koje pokušava potaknuti razvoj robotike koristeći usluge u oblaku poput računarstva u oblaku, pohrane u oblaku i dr. Kada je robot povezan na oblak, isti može značajno profitirati snažnijim računalnim mogućnostima, većom pohranom te komunikacijskim resursima putem modernih podatkovnih centara u oblaku. Podatkovni centri

mogu procesuirati i dijeliti informacije s različitih uređaja. Zbog takvog načina centralizacije sustava moguće je smanjiti cijenu proizvodnje i održavanja. Ljudi također mogu na takav način jednostavnije dodijeliti određeni zadatak robotu. Oblak nudi barem šest bitnih komponenti:

- Globalnu biblioteku slika, mapa, podataka o raznim objektima (često s mehaničkim i geometrijskim svojstvima), baze podataka
- Masivne i paralelne izračune na zahtjev te na takav način planirati pokrete robote, njegove zadatke, kolaboraciju više robota i sl.
- Međurobotsko dijeljenje rezultata određenih radnji, trajektorije, dinamičke kontrole te međusobne podrške
- Međuljudsko dijeljenje otvorenog koda, podataka, dizajna koda kao i dizajna hardwarea te eksperimenata
- Upravljanje i asistiranje na zahtjev radi evaluacije, učenja te otklanjanja pogrešaka
- Interakcija između čovjeka i robota putem proširene stvarnosti

Kako je istaknuto u njemačkom industrijskom planu, industrija je na pragu svoje četvrte revolucije. Pomoću interneta, stvarni i virtualni svijet sve više se približavaju jedan drugom i tako tvore *Internet Stvari* (engl. *Internet of Things*). Industrijska proizvodnja će u budućnosti biti sve više karakterizirana kao jaka individualizacija proizvoda pod uvjetom fleksibilne proizvodnje. U proizvodnim procesima, robot u oblaku može naučiti kako vršiti niz različitih zadaća. Grupa robota može dijeliti informacije i na takav način surađivati. Također, potrošač bi bio u mogućnosti naručiti proizvod u potpunosti prilagođen njegovim potrebama i zahtjevima. Druga potencijalna paradigma je robot-dostavljač kakav se može pronaći u *Amazon Prime Air* ponudi (Slika 3.1.).

Slika 3.11. *Amazon Prime Air (Izvor: https://i.ytimg.com/vi/PwbtzSTGNXc/maxresdefault.jpg)*

Također, jedno od područja primjene bi bio i medicinski oblak koji se sastoji od različitih usluga poput arhive bolesti, elektronskih medicinskih zapisa, analitičke usluge i dr. Robot se može povezati na oblak i na takav način pružiti medicinske usluge pacijentima kao i pružanje dodatne podrške liječnicima (npr. prilikom operacije). Također, može pružiti kolaboracijske usluge dijeljenjem informacija koje su potrebne za daljnje liječenje. Pomoćni roboti se mogu brinuti o zdravlju starijih osoba. Sustav skuplja podatke o zdravstvenom stanju osobe, razmjenjuje informacije s bazama podataka i liječnicima kako bi se mogla pružiti pravovremena reakcija.

Velika prednost robotike u oblaku je i samoupravljački auto. U razvoju takvog robota svakako prednjači *Google* (Slika 3.2.). Navedeni robot koristi mrežu kako bi pristupio enormno velikoj bazi mapa, satelitskih snimaka te informacija o terenu i izgledu okruženja (*Google Streetview*). S takvim informacijama, robot kombinira podatke iz svojih senzora (GPS, kamera i 3D senzori) te prati svoju poziciju s tolerancijom od samo nekoliko centimetara. Uz to prati prati i uzorke prometovanja ostalih vozila kako bi se izbjegao sudar. Svaki auto može naučiti o okolini, cestama, upravljanju, uvjetima na podlozi te skupljene informacije proslijediti u oblak. U oblaku će se podaci dalje procesuirati kako bi se poboljšala učinkovitost samih vozila.

Slika 3.12. *Google self-driving car (Izvor: https://i.ytimg.com/vi/cdgQpa1pUUE/maxresdefault.jpg)*

Iako roboti mogu značajno profitirati putem usluga u oblaku, oblak ipak nije rješenje za cjelokupnu robotiku. Upravljanje robotom koji se najviše oslanja na vlastite senzore nema potrebe za takvom vrstom usluge. Također, aplikacije koje koriste oblak mogu biti sporije zbog ograničenja same mreže, što automatski može donijeti dodatne poteškoće ukoliko je bitna brza reakcija u stvarnom vremenu.

Jedan od rizika usluga u oblaku je i sigurnost. Zbog računalnih resursa često su na meti virtualnih računala koji ciljaju i ranjavaju sustav. Ukoliko se dogodi prodor u sustav, postoji rizik od preuzimanja raznih informacija i kontrole robota s treće strane što za sobom povlači temu etičnosti [23].

3.3 Priprema okruženja za izradu Android aplikacije

Kako je već ranije navedeno *rosjava* je projekt otvorenog koda u *Java*-i koji integrira Robotski operativni sustav s okruženjem za razvoj aplikacija koje se koriste na *Android OS*. Sve upute koje će biti niže prikazane su za *Linux* operativni sustav, iako se isto može instalirati na bilo kojoj 64-bitnoj distribuciji koja koristi *GNOME, KDE* ili *Unity DE* kao i *GNU C Library* (*glibc*) verziju 2.31 ili kasniju.

Instalacija se izvršava u dva dijela – prvo je potrebno instalirati Android Studio i Android SDK, a zatim ROS i ROSJava.

Za instalaciju nam su nam dostupna dva paketa opreme *openjdk* te *Oracle Java Development Kit*. *Android Studio* će odmah u početku dati upozorenje ako se koristi *openjdk*, iz razloga što je *Oracle JDK* bolje rješenje ukoliko računalo ima manje radne memorije.

Kako bi se instalirao *openjdk* potrebno je unijeti:

```
$ sudo apt-get install openjdk-6-jdk
```

Nakon čega je potrebno instalirati *ia-32 libs* neovisno o tome koristi li se 64-bitna ili 32-bitna verzija operativnog sustava na računalu.

```
$ sudo apt-get install ia32-libs
```

Tada slijedi instalacija za *Android Studio*:

```
$ echo export PATH=\${PATH}:/opt/android-studio/sdk/tools:/opt/android-studio
/sdk/platform-tools:/opt/android-studio/bin >> ~/.bashrc

$ echo export ANDROID_HOME=/opt/android-studio/sdk >> ~/.bashrc
```

Na koncu pokrećemo *Android SDK* upravitelj:

```
$ android
```

Nakon što smo pripremili *Android SDK* potrebno je instalirati ROS i ROS Java Debian. Postoje već prethodno kompilirani debiani s velikim brojem paketa za *Ubuntu* što uvelike olakšava instalaciju. Biti će nam potrebni *catkin*, *ros* (za *setup.bash* varijable) te nekoliko paketa poruka kako bi bili sigurni da će se sve uspješno kompilirati. Postoji još nekoliko drugih paketa koji će nam možda biti potrebni, ali se mogu instalirati naknadno i prema potrebi.

```
> sudo apt-get install ros-hydro-catkin ros-hydro-ros ros-hydro-common-msgs
```

Slijedi nam instalacija za *Wstool*:

```
> sudo apt-get install python-wstool
```

Nakon čega je potrebno postaviti lančane *catskin* radne površine za moduliranje projekata i na taj način smanjiti vrijeme kompiliranja.

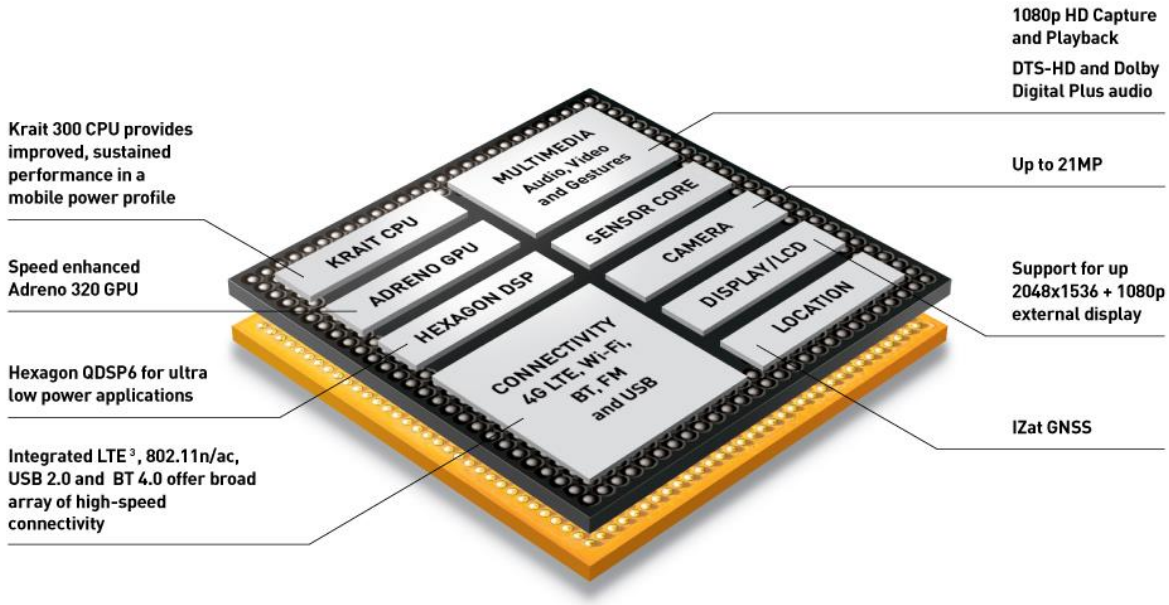
```
> mkdir -p ~/rosjava
> wstool init -j4 ~/rosjava/src https://raw.github.com/rosjava/rosjava/
hydro/rosjava.rosinstall
> source /opt/ros/hydro/setup.bash
> cd ~/rosjava
> catkin_make
```

Kada smo uspješno završili s prethodnim korakom, potrebno je pripremiti *Android* radnu površinu.

```
> mkdir -p ~/android
> wstool init -j4 ~/android/src https://raw.github.com/rosjava/rosjava/
hydro/android_core.rosinstall
# or if you want the android_apps/android remocons repos as well:
# wstool init -j4 ~/android/src https://raw.github.com/rosjava/rosjava/
hydro/android_apps.rosinstall
> source ~/rosjava/devel/setup.bash
> cd ~/android
> catkin_make
```

Sada imamo u potpunosti funkcionalan *Android Studio* i *SDK* koji su u cijelosti integrirani s *rosjava* razvojnim okruženjem [24].

Međutim, ništa ne bi bilo moguće izvesti bez odgovarajućeg računalnog sklopovlja koji pokreće *Android OS*. *Open Source Robotics Foundation* u potpunosti održava i nadgleda sve radnje na Robotskom operativnom sustavu. *Open Source Robotics Foundation* je tako 2014. godine u planu imala dodavanje *ARM* podrške u Robotski operativni sustav, a sve je započelo s *Qualcomm Snapdragon 600* procesorom (Slika 3.3.) u četvrtom kvartalu 2014. godine [25].



Slika 3.13. *Qualcomm Snapdragon 600 arhitektura (Izvor: http://files.linuxgizmos.com/qualcomm-snapdragon-600-diag.jpg)*

4. PRIMJENA ROBOTSKOG OPERATIVNOG SUSTAVA I ANDROID OS-A

U ovom poglavlju prikazuje se kako se Android aplikacije mogu integrirati s robotskim operativnim sustavom (ROS) za snimanje i objavljivanje podataka senzora i kamere.

Ove implementacije pokazuju na koji način Android aplikacije mogu komunicirati s Robotskim operativnim sustavom za obradu i prijenos podataka. Korištenjem kamere i akcelerometra aplikacije objavljuju podatke na ROS mrežu.

Proces postavljanja uključuje kloniranje potrebnih repozitorija, konfiguriranje razvojnog okruženja i instaliranje potrebnih ROS paketa.

4.1 Primjer implementacije kamere

Proces kloniranja repozitorija, kao što je prikazano na slici 4.1, ključni je prvi korak u postavljanju razvojnog okruženja. Ovaj korak osigurava da su potrebne projektne datoteke i ovisnosti ispravno preuzete iz repozitorija. Kloniranjem repozitorija dobiva se kompletna struktura projekta, uključujući izvorni kod, konfiguracijske datoteke i relevantne biblioteke, koje su bitne za izgradnju i implementaciju aplikacije.

```
git clone -b hydro-standalone https://github.com/ros-java/android_apps.git
```

Slika 4.14. *Primjer kloniranja repozitorija*

Nakon što se repozitorij klonira, potrebno ga je uvesti u razvojno okruženje odabirom odgovarajuće opcije za uvoz postojećeg projekta. Zatim treba locirati klonirano spremište i uvesti ga u radni prostor. Tijekom procesa uvoza treba odabrati opciju korištenja vanjskog sustava za izgradnju, kao što je *Gradle*. Konačno, treba potvrditi konfiguraciju alata za izgradnju kako bi se dovršilo postavljanje projekta.

Kako bi se osiguralo ispravno funkcioniranje Android aplikacije, potrebno je instalirati potrebne ROS pakete. To se može učiniti izvršavanjem naredbi sa slike 4.2.

```
sudo apt-get install ros-hydro-image-view ros-hydro-image-transport-plugins

sudo apt-get install ros-hydro-common-msgs
```

Slika 4.2. *Primjer instalacije ROS paketa*

Ovi paketi su bitni za rukovanje pregledom i prijenosom slika, kao i za pružanje uobičajenih tipova poruka koje se koriste u ROS komunikaciji.

Primjena ROS-a na Android platformi može se vidjeti na primjeru koda u kojem je vidljiva implementacija koja omogućuje Android aplikaciji pregled i objavu podataka o kameri na ROS-u uz mogućnost prebacivanja između više kamera na uređaju. Za početak su nam potrebne sljedeće biblioteke:

```
import android.hardware.Camera;
import android.os.Build;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Toast;
import org.ros.address.InetAddressFactory;
import org.ros.android.RosActivity;
import org.ros.android.view.camera.RosCameraPreviewView;
import org.ros.node.NodeConfiguration;
import org.ros.node.NodeMainExecutor;
import android.util.Log;
import java.io.IOException;
```

Ovaj konstruktor poziva nadređeni RosActivity konstruktor, prosljeđujući nazive "CameraTutorial" i kao naziv čvora i kao naziv aplikacije. Klasa *RosActivity* dio je *ROS Android* biblioteke koja omogućuje komunikaciju s glavnim čvorom.

```
public class MainActivity extends RosActivity {

    private int cameraId;
    private RosCameraPreviewView rosCameraPreviewView;

    public MainActivity() {
        super("CameraTutorial", "CameraTutorial");
    }
```

Metoda getCamera() vraća instancu kamere na temelju trenutnog ID-a. Prvo otvara kameru koristeći i dohvaća njezine parametre. Ovisno o verziji Androida, metoda osigurava da kamera

koristi odgovarajući način fokusa. Za uređaje sa sustavom Ice Cream Sandwich ili novijim, provjerava jesu li podržani kontinuirani načini fokusiranja *FOCUS_MODE_CONTINUOUS_VIDEO* ili *FOCUS_MODE_CONTINUOUS_PICTURE*.

```
private Camera getCamera() {
    Camera cam = Camera.open(cameraId);
    Camera.Parameters camParams = cam.getParameters();
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.ICE_CREAM_SANDWICH) {
        if (camParams.getSupportedFocusModes().contains(
                    Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO)) {

camParams.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_VIDEO);
        } else {

camParams.setFocusMode(Camera.Parameters.FOCUS_MODE_CONTINUOUS_PICTURE);
        }
    }
    cam.setParameters(camParams);
    return cam;
}
```

Metoda *onCreate()* se poziva kada se aktivnost prvi put kreira. Inicijalizira sučelje u načinu rada preko cijelog zaslona i bez naslovne trake. Metoda koristi setContentView() za postavljanje rasporeda aktivnosti. Povezuje komponentu *RosCameraPreviewView* s elementom korisničkog sučelja s ID-om ros_*camera*_preview_view, koji je odgovoran za prikaz kamere na zaslonu.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
    setContentView(R.layout.main);
    rosCameraPreviewView = (RosCameraPreviewView)
findViewById(R.id.ros_camera_preview_view);
}
```

Metoda *onTouchEvent()* upravlja korisničkim unosom putem zaslona. Točnije, osluškuje događaj "ACTION_UP" koji se događa kada korisnik podigne prst sa zaslona. Nakon ovog događaja, metoda provjerava broj dostupnih kamera na uređaju.

Ako uređaj ima više kamera, prolazi kroz njih povećanjem vrijednosti varijable *cameraId* i resetiranjem pregleda kamere otpuštanjem trenutne kamere i odabirom sljedeće. Zatim prikazuje poruku koja obavještava korisnika da se kamera prebacuje. Ako nema dostupnih alternativnih kamera, prikazuje se poruka da prebacivanje nije moguće.

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if (event.getAction() == MotionEvent.ACTION_UP) {
        int numberOfCameras = Camera.getNumberOfCameras();
        final Toast toast;
        if (numberOfCameras > 1) {
            cameraId = (cameraId + 1) % numberOfCameras;
            rosCameraPreviewView.releaseCamera();
            rosCameraPreviewView.setCamera(getCamera());
            toast = Toast.makeText(this, "Switching cameras.",
Toast.LENGTH_SHORT);
        } else {
            toast = Toast.makeText(this, "No alternative cameras to switch
to.", Toast.LENGTH_SHORT);
        }
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                toast.show();
            }
        });
    }
    return true;
}
```

Metoda *init()* odgovorna je za inicijalizaciju čvora i pripremu kamere za objavljivanje u ROS. U početku se definira početna kamera putem varijable *cameraId*. Zatim se konfigurira *rosCameraPreviewView* da koristi kameru koju je dohvatio *getCamera()*.

Za uspostavljanje komunikacije s ROS masterom, metoda otvara vezu i dohvaća lokalnu mrežnu adresu uređaja. Ova adresa lokalne mreže koristi se za konfiguriranje ROS čvora, dopuštajući mu objavljivanje i pretplatu na teme nakon čega se pokreće i sam čvor.

```
@Override
protected void init(NodeMainExecutor nodeMainExecutor) {
    cameraId = 0;

    rosCameraPreviewView.setCamera(getCamera());
    ◘     try {
        java.net.Socket socket = new
java.net.Socket(getMasterUri().getHost(), getMasterUri().getPort());
        java.net.InetAddress local_network_address =
socket.getLocalAddress();
        socket.close();
        NodeConfiguration nodeConfiguration =
        NodeConfiguration.newPublic(local_network_address.getHostAddress(),
getMasterUri());
        nodeMainExecutor.execute(rosCameraPreviewView, nodeConfiguration);
    } catch (IOException e) {
        // Socket problem
        Log.e("Camera Tutorial", "socket error trying to get networking
information from the master uri");
    }
```

```
}
```

U slučaju bilo kakve greške prilikom povezivanja javit će se greška: *socket error trying to get networking information from the master uri* [26].

4.2 Primjer integracije akcelerometra

Osim primjera upravljanja modulom kamere jedna od mogućnosti je integracija senzora temeljenih na Androidu, a za to su potrebne biblioteke:

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import org.ros.address.InetAddressFactory;
import org.ros.android.RosActivity;
import org.ros.node.NodeConfiguration;
import org.ros.node.NodeMainExecutor;
import org.ros.node.topic.Publisher;
import sensor_msgs.Imu;
```

Konstruktor *SensorActivity()* inicijalizira aktivnost pozivanjem konstruktora *RosActivity* sa "SensorPublisher" kao nazivom aplikacije i čvora. Konstruktor postavlja aktivnost kao ROS čvor koji će rukovati podacima senzora s uređaja i objaviti ih u ROS temi. Primarna funkcija ove aktivnosti je čitanje podataka senzora (u ovom slučaju s akcelerometra) i objavljivanje dobivenih podataka u obliku ROS poruka.

```
public class SensorActivity extends RosActivity implements
SensorEventListener {

    private SensorManager sensorManager;
    private Publisher<Imu> publisher;

    public SensorActivity() {
        super("SensorPublisher", "SensorPublisher");
    }
```

Pozivanjem metode *onCreate()* postavlja se korisničko sučelje. Također inicijalizira *SensorManager*, koji je odgovoran za upravljanje senzorima na uređaju. Metoda zatim dohvaća senzor akcelerometra uređaja i registrira aktivnost. Nakon što senzor akcelerometra pošalje podatke aktivnosti pokrenut će se metoda *onSensorChanged()* kada novi podaci budu dostupni.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sensor);

        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        Sensor accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        sensorManager.registerListener(this, accelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
}
```

Kad god registrirani senzor otkrije promjenu u svojim podacima poziva se metoda *onSensorChanged()*. Metoda odmah poziva *publishSensorData()* za objavljivanje podataka akcelerometra kao ROS poruke.

```
@Override
public void onSensorChanged(SensorEvent event) {
        publishSensorData(event);
}
```

Metoda *onAccuracyChanged()* potrebna je sučelju *SensorEventListener*, ali se ne koristi u ovoj aktivnosti.

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // No action required
}
```

Za pretvaranje podataka akcelerometra u ROS poruke i objavljivanje u ROS temama poziva se metoda *publishSensorData()*.Podaci akcelerometra pohranjuju se u polja *event.values*, koje sadrže vrijednosti *x, y* i *z*. Nakon što se podaci dodijele poruci, metoda objavljuje poruku u ROS temi koristeći *publisher.publish()* metodu.

```
private void publishSensorData(SensorEvent event) {
        Imu imuMessage = publisher.newMessage();
        imuMessage.getLinearAcceleration().setX(event.values[0]);
        imuMessage.getLinearAcceleration().setY(event.values[1]);
        imuMessage.getLinearAcceleration().setZ(event.values[2]);
        publisher.publish(imuMessage);
}
```

Metoda *init()* postavlja ROS čvor koji objavljuje podatke senzora. Prvo instancira objekt *NodeConfiguration* koristeći mrežnu adresu bez povratne petlje uređaja, što osigurava da ROS čvor može komunicirati preko mreže.

Metoda zatim postavlja URI za ROS čvor i nakon što se postavi konfiguracija, stvara se izdavač za temu */android_imu* s vrstom poruke *sensor_msgs.Imu*. Ovaj izdavač se objavljivanjem poruka kreiranih podacima koji su dobiveni od akcelerometra.

```
    @Override
    protected void init(NodeMainExecutor nodeMainExecutor) {
        NodeConfiguration nodeConfiguration =
NodeConfiguration.newPublic(InetAddressFactory.newNonLoopback().getHostAddres
s());
        nodeConfiguration.setMasterUri(getMasterUri());
        publisher = nodeMainExecutor.newNodeMain("/android_imu",
sensor_msgs.Imu._TYPE);
    }
}
```

5. ZAKLJUČAK

Razvoj robotike, od ranih daljinski upravljanih torpeda do modernih humanoidnih i industrijskih robota, označio je tehnološku evoluciju. Centralni dio ovog napretka je Robotski operativni sustav, koji je uvelike unaprijedio razvoj i funkcionalnost robotskih sustava. Razvijajući se kroz različite verzije pruža skup alata koji podržava razvoj softvera i hardvera.

Utjecaj ROS-a vidljiv je u fizičkim i softverskim robotskim sustavima. Podržava širok raspon aplikacija, od obrazovne i istraživačke robotike do industrijske automatizacije, te se neprimjetno integrira s platformama kao što su *Lego Mindstorms, Arduino* i *AlphaBot*. Alati kao što su *Gazebo* i *RViz* unutar ROS-a nude mogućnosti simulacije. Prilagodljivost sustava i stalna podrška osiguravaju da on ostane pokretač napretka i šireći mogućnosti unutar polja.

Integracija Robotskog operativnog sustava s Androidom kroz *Rosjava* okruženje predstavlja ključni razvoj u robotici. Korištenjem *Rosjave*, olakšava se stvaranje robotskih aplikacija koje koriste mogućnosti *Androida*.

Također, pojava robotike u oblaku uvela je mogućnosti za poboljšanje funkcionalnosti robota putem usluga u oblaku. Rješenja temeljena na oblaku predstavljaju značajne prednosti kao što su dijeljenje podataka i obrada u stvarnom vremenu. Razvoj samovozećih automobila i medicinskih robota primjer je potencijala robotike u oblaku.

Buduća će se istraživanja vrlo vjerojatno usmjeriti na daljnje poboljšanje razvojnih alata, kako komercijalnih tako i otvorenog koda, kompatibilnosti s više platformi, a osim toga i povećanje kvalitete autonomije robota kao i interakcije između čovjeka i robota.

Kako se robotika nastavlja razvijati, integracija Androida i ROS-a, uz tehnologije u oblaku, vjerojatno će potaknuti daljnje inovacije i aplikacije, oblikujući budućnost automatizacije i inteligentnih sustava.

LITERATURA

[1] https://en.wikipedia.org/wiki/Robot#History, pristup ostvaren 27.11.2023.

[2] H.R. Everett, Unmanned Systems of World Wars I and II, 2015, MIT Press, str. 87

[3] Shimon Y. Nof, Handbook of Industrial Robotics (2nd ed.), John Wiley & Sons, 1999, str. 5

[4] https://en.wikipedia.org/wiki/Robot_Operating_System, pristup ostvaren 27.11.2023.

[5] http://www.willowgarage.com/pages/about-us/history, pristup ostvaren 27.06.2016.

[6] http://www.willowgarage.com/pages/about-us/leadership-team, pristup ostvaren 27.06.2016.

[7] http://wiki.ros.org/Distributions, pristup ostvaren 27.06.2016.

[8] https://docs.ros.org/en/rolling/Releases.html pristup ostvaren 2.7.2024

[9] Aaron Martinez, Enrique Fernandez, Learning ROS for Robotics Programming, Packt Publishing, 2013, str. 26

[10] Aaron Martinez, Enrique Fernandez, Learning ROS for Robotics Programming, Packt Publishing, 2013, str. 57

[11] https://www.geeksforgeeks.org/differences-between-tcp-and-udp/ pristup ostvaren 2.7.2024

[12] Aaron Martinez, Enrique Fernandez, Learning ROS for Robotics Programming, Packt Publishing, 2013, str. 63

[13] Aaron Martinez, Enrique Fernandez, Learning ROS for Robotics Programming, Packt Publishing, 2013, str. 69

[14] https://docs.arduino.cc/ pristup ostvaren 2.7.2024

[15] Aaron Martinez, Enrique Fernandez, Learning ROS for Robotics Programming, Packt Publishing, 2013, str. 70

[16] http://hackeducation.com/2015/04/10/mindstorms/, pristup ostvaren 27.06.2016.

[17] https://www.waveshare.com/wiki/AlphaBot, pristup ostvaren 6.7.2024,

[18] Karim Yaghmour, Embedded Android: Porting, Extending, and Customizing, 2013, str. 1

[19] https://github.com/rosjava/rosjava_core, pristup ostvaren 27.11.2023.

[20] http://wiki.ros.org/rosjava/Overview, pristup ostvaren 27.11.2023.

[21] Benjamin Muschko, Gradle in Action, 2014

[22] Sonartype, Maven: The Definitive Guide, 2008

[23] https://en.wikipedia.org/wiki/Cloud_robotics, pristup ostvaren 27.11.2023.

[24] http://wiki.epfl.ch/roscontrol/androidstudio-and-rosjava, pristup ostvaren 27.11.2023.

[25] https://www.youtube.com/watch?v=0f0y7Ki9QC4, pristup ostvaren 27.11.2023.

[26] https://github.com/rosjava/android_core/blob/kinetic/android_tutorial_camera/src/org/ros/android/android_tutorial_camera/MainActivity.java, pristup ostvaren 27.11.2023

SAŽETAK

Krajnji cilj završnog rada bio je proučiti i razraditi primjenu Robotskog operativnog sustava te njegovu primjenu na *Android* platformi. Robotski operativni sustav se sastoji od od razine datotečnog sustava, računalne grafike te inženjerske okoline. Tijekom 2014. godine na prijedlog *Open Source Robotics Foundationa* tvrtka *Qualcomm* je u arhitekturu svojih procesora dodala podršku za Robotski operativni sustav. Tvrtka *Google* u suradnji s *Willow Garageom* je kreirala čistu *Java* implementaciju Robotskog operativnog sustava za *Android* platformu pod imenom *rosjava*. Sam projekt je još uvijek u početnim fazama razvoja. *Rosjava* omogućuje integraciju *Android* operativnog sustava s robotima koji koriste Robotski operativni sustav, što znači da je moguće korištenje senzora uređaja s *Android OS* za upravljanje robotom. Također, jedna od bitnih primjena Robotskog operativnog sustava, a pogotovo u kombinaciji s *Android* platformom biti će robotika u oblaku. Bitno je i napomenuti integraciju ROS-a s *AlphaBot Pi* platformom koja može poboljšati sposobnosti robota pružajući mogućnosti za lakše upravljanje kompliciranim zadacima kao što je upravljanje ROS čovorovima i strojno učenje.

Ključne riječi: AlphaBot Pi, Android, Camera, Google, Java, Robot, ROS, Willow Garage

ABSTRACT

The ultimate goal of the final paper was to study and elaborate the application of the Robot Operating System and its application on the *Android* platform. The ROS consists of the file system level, computer graphics and the engineering environment. In 2014, at the suggestion of the *Open Source Robotics Foundation*, the *Qualcomm* company added support for the Robot Operating System to the architecture of its processors. *Google*, in cooperation with Willow Garage, created a pure *Java* implementation of the *Robot Operating System* for the *Android* platform under the name *rosjava*. The project itself is still in the initial stages of development. *Rosjava* enables the integration of the *Android* operating system with robots using the *Robot Operating System*, which means that it is possible to use the sensors of *Android OS* devices to control the robot. Also, one of the important applications of the *Robot Operating System*, especially in combination with the *Android* platform, will be robotics in the cloud. It is also important to note the integration of ROS with the *AlphaBot Pi* platform, which can improve the capabilities of the robot by providing opportunities for easier management of complicated tasks such as creating of ROS nodes and deep learning.

Keywords: AlphaBot Pi, Android, Camera, Google, Java, Robot, ROS, Willow Garage

ŽIVOTOPIS

Vladimir Ščuric je rođen 03. srpnja 1988. godine u Osijeku. U braku i otac malene djevojčice koja je rođena u lipnju 2023. godine. Osnovnu školu završio je u Višnjevcu 2003. godine, te iste godine upisao je Strojarsku tehničku školu u Osijeku – smjer računalni tehničar za strojarstvo, gdje je ostvario zapažene rezultate iz CAD/CAM sustava na državnoj razini. Godine 2007. je maturirao sa odličnim uspjehom i upisao Elektrotehnički fakultet u Osijeku – smjer preddiplomski studij računarstva, koji trenutno pohađa, te mu je, u ovom trenutku, ostao završni rad.

U studenom 2008. godine, putem studentskog servisa, zaposlio se u tvrtci *Hrvatski Telekom* d.d., kao agent u službi za korisnike – odjel tehničke podrške, koji je zadržao do kraja lipnja 2011. godine. Početkom srpnja 2011. godine premješten je na odjel specijalističke podrške privatnim korisnicima. Od svibnja 2013. godine radio je u službi za korisnike *Tele2 d.o.o.* te od 1. listopada 2015. godine nastavlja svoj rad u sklopu podrške na društvenim mrežama. Zatim 2018. godine je prešao u službu za korisnike tvrtke *Barrage d.o.o.* u kojoj se zadržao do veljače 2020. godine kada je prešao u tvrtku *Base58 d.o.o.* u kojoj je radio sve do lipnja 2024. godine. Započeo je kao specijalist blockchain operacija u sklopu koje je započeo put razvojnog inženjera. Te trenutno radi kao backend razvojni inženjer u tvrtci *Devot Solutions d.o.o.* u sklopu koje se služi raznim tehnologijama i rješenjima (Java, Spring boot, Postgres, Redis, Docker itd.).

Izuzev obitelji, slobodno vrijeme mu je ispunjeno tehnologijom te je upoznat i aktivno koristi razne distribucije Linux operativnog sustava (temeljene na Debianu), a posjeduje i duboko znanje vezano uz MS Windows operativne sustave, utvrđivanje i otklanjanje problema, umrežavanje i slično.

Vladimir Ščuric