Učenje robota putem višemodalne interakcije

Vitez, Nikola

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Nikola Vitez

Zagreb, 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

prof. dr. sc. Bojan Jerbić

Student:

Nikola Vitez

Zagreb, 2017.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću na pomoći i sugestijama prilikom izrade diplomskog rada. Također se želim zahvaliti dr. sc. Bojanu Šekoranji na korisnim savjetima i pomoći pri izradi diplomskog rada kao i ostatku tima iz Laboratorija za projektiranje izradbenih i montažnih sustava na Fakultetu strojarstva i brodogradnje u Zagrebu.

Posebno se zahvaljujem svojim roditeljima Zlatku i Mirjani te ostatku obitelji i dugogodišnjoj djevojci Eni na pruženoj podršci i razumijevanju tijekom čitavog studija.

Nikola Vitez

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala te mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur. broj:	

DIPLOMSKI ZADATAK

Student: **Nikola Vitez** Mat. br.: 0035190410

Naslov rada na hrvatskom jeziku: **Učenje robota putem višemodalne interakcije**

Naslov rada na engleskom jeziku: **Teaching Robot by Multimodal Interaction**

Opis zadatka:

U radu je potrebno istražiti mogućnost učenja robota putem višemodalne interakcije čovjeka i robota. Višemodalna interakcija podrazumijeva robotsku percepciju stanja okoline temeljem više vrsta senzora (brzine, ubrzanja, položaja, sile, vizualnih senzora itd.). Prikupljene informacije sa senzora potrebno je analitički obraditi i klasificirati s ciljem prepoznavanja i učenja različitih radnih postupaka. Razvijeni model učenja potrebno je implementirati i provjeriti na raspoloživoj robotskoj opremi u Laboratoriju za projektiranje izradbenih i montažnih sustava.
U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 11. svibnja 2017.

Rok predaje rada: 13. srpnja 2017.

Predviđeni datum obrane: 19., 20. i 21. srpnja 2017.

Zadatak zadao:

Prof. dr. sc. Bojan Jerbić

Predsjednica Povjerenstva:

Prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS OZNAKA

Oznaka	Jedinica	Opis
ϕ	°	Kut valjanja
θ	°	Kut poniranja
ψ	°	Kut skretanja
$\mathbf{R}$	-	Matrica rotacije
$\mathbf{T}$	-	Matrica homogene transformacije
$\mathbf{t}$	-	Vektor translacije
Q	-	Kvaternion
α	°	Kut rotacije
$\boldsymbol{\theta}$	-	Axis-angle zapis rotacije
$\mathbf{e}$	-	Jedinični vektor axis-angle zapisa
$\bar{x}$	-	Srednja vrijednost
x_{RMS}	-	RMS vrijednost
x_{maxabs}	-	Maksimalna apsolutna vrijednost
R	-	Raspon varijacije
σ	-	Standardna devijacija
I_Q	-	Interkvartilni raspon

SAŽETAK

Ovaj diplomski rad istražuje mogućnosti učenja robota višemodalnom interakcijom čovjeka i robota koja podrazumijeva robotsku percepciju stanja okoline temeljem više vrsta senzora.

U radu je najprije opisan korišteni eksperimentalni postav, a zatim je ispitana mogućnost klasifikacije i prepoznavanja radnji pisanja i brisanja ploče. Klasifikacija radnji provedena je na temelju analize mjerenih podataka koji se prikupljaju iz inercijskog mjernog uređaja i vizijskog sustava Polaris Vicra. U drugom dijelu rada je razvijeno učenje robota brisanju na temelju prepoznavanja radnje brisanja razvijenim klasifikacijskim algoritmom te je implementiran primjer rada robotskog sustava na temelju jednostavne klasifikacije naučenih pokreta brisanja.

Ključne riječi: robot, učenje, klasifikacija, inercijski mjerni uređaj, vizijski sustav.

SUMMARY

This master thesis explores the possibilities of robot learning by the multi-modal interaction of man and robot, which implies a robotic perception of the environment on the basis of several different types of sensors.

In this paper, the experimental setup used was first described, and then the possibility of classifying and recognizing actions of writing and wiping off of whiteboard was examined. The classification of the actions was carried out on the basis of the analysis of measured data collected from the inertial measurement unit and the vision system Polaris Vicra. In the second part of the paper, learning a robot to wipe off the whiteboard has been developed based on the recognition of the action of wiping by a developed classification algorithm, futhermore an example of work of robotic system based on a simple classification of the learned movements during action of wiping is implemented.

Key words: robot, learning, classification, inertial measurement unit, vision system.

1. UVOD

Primjena robota kao fizičke nadogradnje računala svakim danom sve se više širi iz klasičnog industrijskog okruženja u naše svakodnevne aktivnosti. Razvijaju se razni roboti od onih za rad u neposrednoj blizini ljudi u industrijskom okruženju, robota za primjenu u medicini, kućanskim poslovima pa sve do robota za pomoć u njezi i brizi o starijim osobama kao i robota za rehabilitaciju. Iz sve veće prisutnosti robota u neposrednoj ljudskoj blizini proizlazi potreba za što jednostavnijom komunikacijom između robota i korisnika odnosno potreba za jednostavnim učenjem robota željenim pokretima ili radnjama, a kako bi se roboti što bolje uklopili u ljudsko okruženje cilj je postići što veću sličnost robota čovjeku kako njegovim vanjskim izgledom tako i oponašanjem ljudskih pokreta. Pojam učenja robota se najčešće povezuje s programiranjem robota kao dugotrajnim i prilično sporim procesom za koji je potrebna velika količina predznanja te je stoga on veoma zastrašujuć za korisnike koji ne posjeduju takva specijalizirana znanja. Razvojem i sve većom dostupnošću jeftinih inercijskih senzora (za mjerenje ubrzanja i brzina) razvijaju se nosivi sustavi za prepoznavanje pojedinog pokreta ili cijelog obrasca kretanja kako bi se olakšala komunikacija između robota i čovjeka te omogućilo prevladavanje straha od složenog postupka učenja robota. Spajanjem podataka s navedenih inercijskih senzora s podacima senzora položaja, sile, vizualnih senzora i sličnih moguće je znatno olakšati proces učenja. Učenje pokazivanjem pokreta i radnji je za ljude prirodno i intuitivno te može omogućiti razvoj robota koji će biti općeprihvaćeni zbog jednostavnog načina učenja kao i gibanja koje je nalik ljudskom.

U ovom radu će biti opisan prikaz i analiza mogućnosti učenja robota korištenjem podataka dobivenih inercijskim mjernim uređajem (IMU) i vizijskim sustavom Polaris Vicra kako bi se dobilo vjerno i pouzdano učenje ljudskih pokreta. U radu je ispitana mogućnost klasifikacije i prepoznavanja radnji pisanja i brisanja ploče. Radnje se klasificiraju na temelju analize seta izmjerenih vrijednosti za učenje pri navedenim pokretima. Na temelju prepoznavanja radnje brisanja implementirano je učenje robota brisanju ploče. Uz to je također implementiran primjer rada robotskog sustava na temelju jednostavne klasifikacije naučenih pokreta brisanja.

1.1. Zadatci

Za potrebe istraživanja mogućnosti učenja robota putem višemodalne interakcije čovjeka i robota primjenom inercijskog mjernog uređaja Sparkfun 9DoF Razor i NDI Polaris Vicra vizijskog sustava bilo je potrebno riješiti niz zadataka :

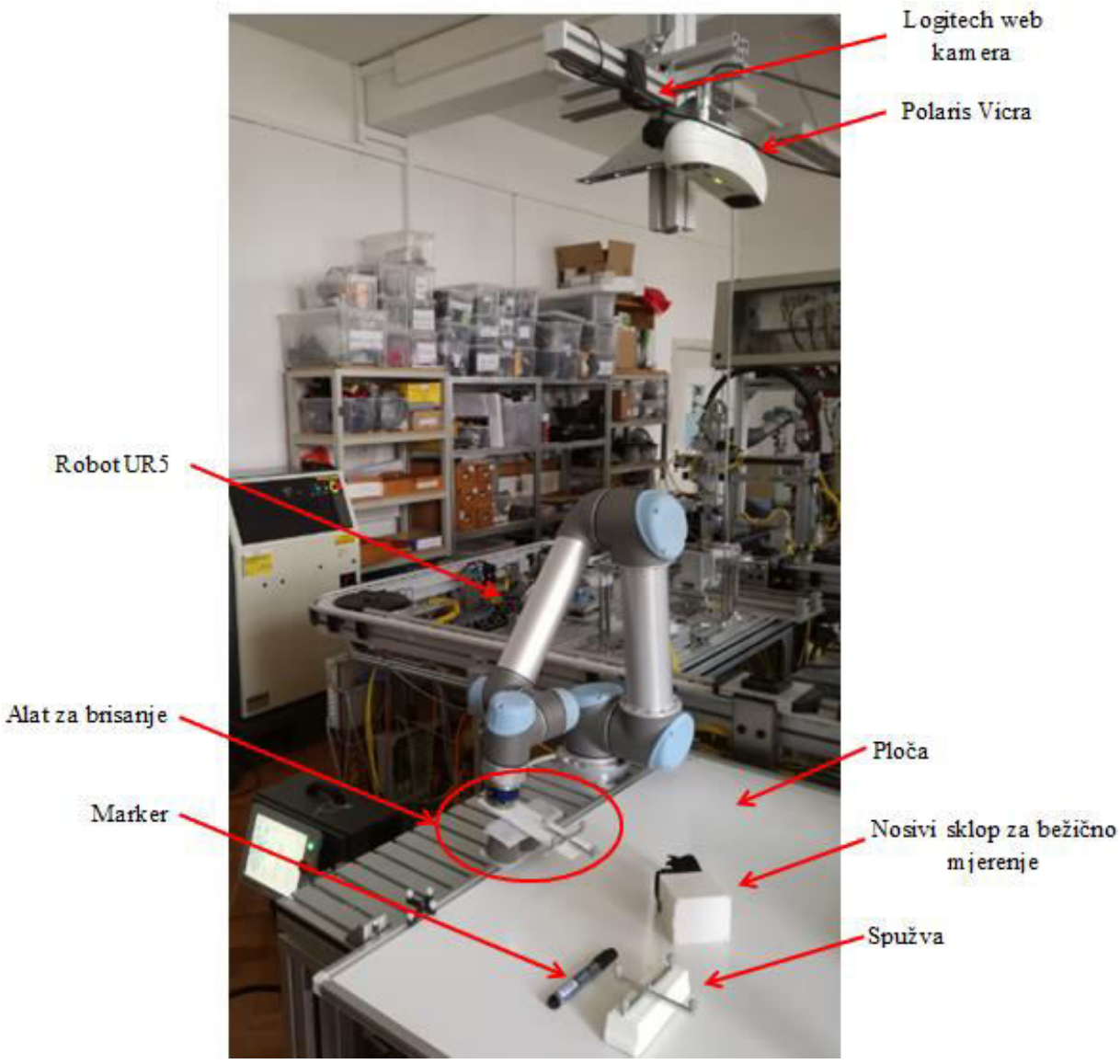
1. Projektirati nosivi eksperimentalni sklop za bežično mjerenje inercijskim senzorom,
2. Kalibrirati i povezati sve elemente sustava,
3. Eksperimentalno prikupiti podatke,
4. Analizirati prikupljene podatke,
5. Razviti algoritam za klasifikaciju radnji,
6. Razviti program za učenje različitih pokreta kod pojedine radnje
7. Osmisliti i implementirati primjer rada na temelju klasifikacije radnji i naučenih pokreta.

1.2. Cilj

Cilj ovog rada je istražiti mogućnosti učenja robota višemodalnom interakcijom koja podrazumijeva percepciju stanja okoline temeljem više vrsta senzora te na taj način stvoriti osnovu za daljnje istraživanje na području intuitivne i jednostavne interakcije robota i čovjeka kao i osnovu za razvoj kognitivnih robota koji su sposobni na temelju nekoliko naučenih radnji riješiti neki novi njima nepoznati problem.

2. EKSPERIMENTALNI POSTAV

Eksperimentalni postav sustava s kojim su provođena mjerenja nalazi se u Laboratoriju za projektiranje izradbenih i montažnih sustava na Fakultetu strojarstva i brodogradnje u Zagrebu. Na slici 1 je prikazan eksperimentalni postav sa svim elementima korištenim tijekom izrade ovog rada.



Slika 1. Elementi eksperimentalnog postava

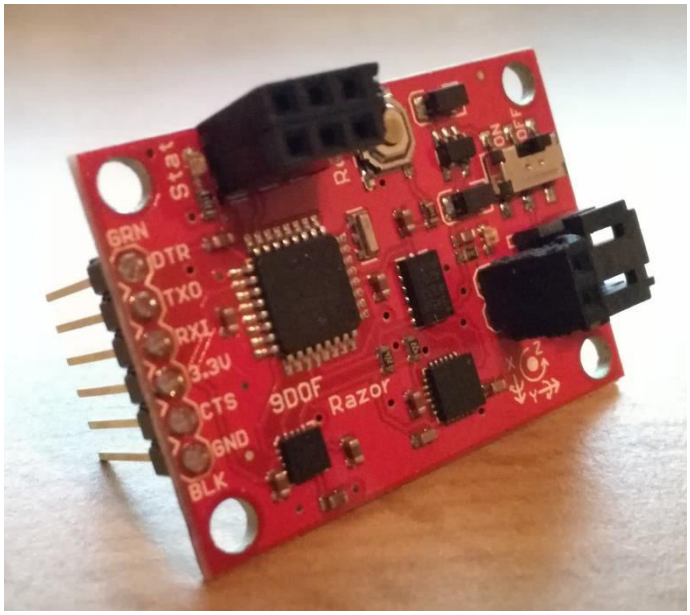
U nastavku su detaljnije opisani najvažniji elementi sustava.

2.1. Nosivi sklop za bežično mjerenje

Za potrebe prepoznavanja obrazaca kretanja ljudske ruke pomoću IMU-a sastavljen je sklop koji omogućuje bežično mjerenje te time i potpunu slobodu kretanja. U svrhu jednostavnog i lakog mjerenja linearnih ubrzanja, kutnih brzina i Eulerovih kuteva koji se javljaju tijekom promatranih radnji bilo je potrebno opremiti IMU s dodatnim dijelovima koji omogućavaju bežični prijenos podataka i napajanje, te onda taj cijeli sklop staviti u kučište kako bi se zaštitila elektronika i omogućila potpuna sloboda kretanja i jednostavno nošenje sklopa tijekom rada.

2.1.1. Inercijski mjerni uređaj

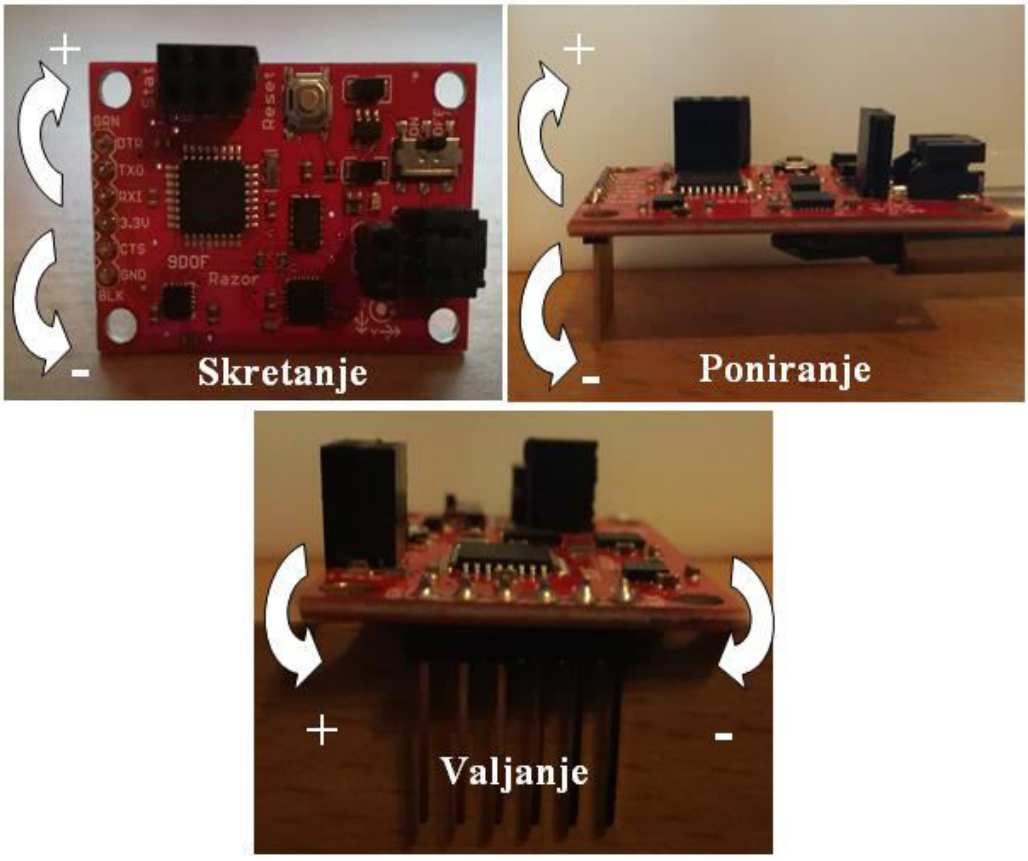
Za prepoznavanje obrazaca kretanja prilikom promatranih radnji korišten je inercijski mjerni uređaj Sparkfun 9DoF Razor, slika 2. IMU se sastoji od troosnog akcelerometra ADXL 345 proizvođača Analog Devices, troosnog žiroskopa ITG-3200 proizvođača InvenSense te troosnog magnetometra HMC5883L proizvođača Honeywell. Izlazne veličine pojedinih senzora obrađuju se pomoću mikrokontrolera ATmega328 proizvođača Atmel, koji je također dio IMU-a, te šalju dalje serijskom vezom.



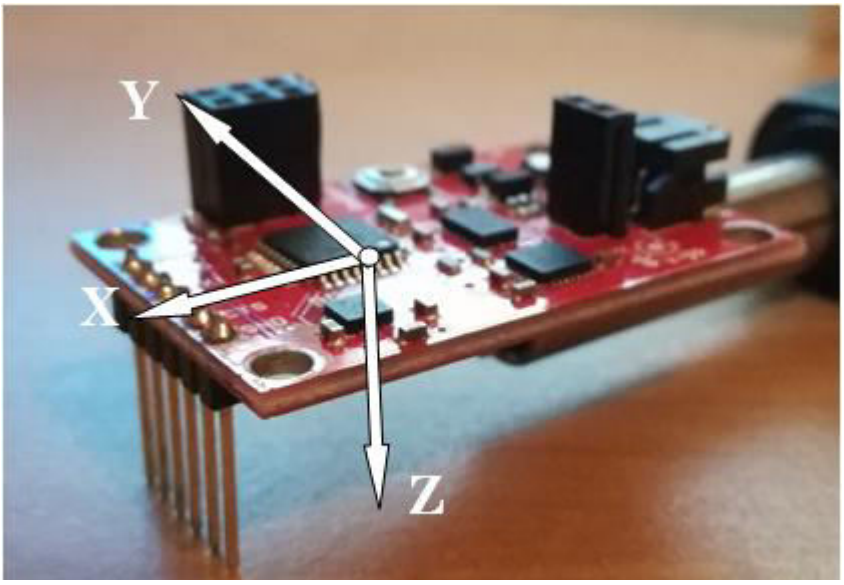
Slika 2. Sparkfun 9Dof Razor IMU

Osim osnovnih veličina koje akcelerometar, žiroskop i magnetometar primarno mjere spajanjem njihovih podataka te njihovom lokalnom obradom dobiva se orijentacija IMU-a u prostoru definirana Eulerovim kutevima (skretanje, poniranje i valjanje) čije je značenje

definirano na slici 3. Eulerovi kutevi definirani su oko koordinatnih osi IMU-a prikazanih na slici 4.



Slika 3. Eulerovi kutevi

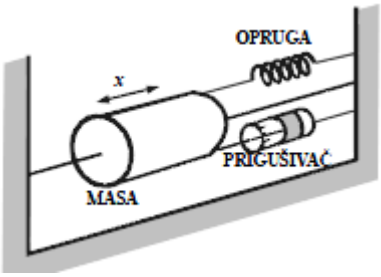


Slika 4. Koordinatne osi IMU-a

Zajedno sa IMU-om proizvođač daje i program koji prima i obrađuje podatke sa senzora te ih šalje na izlaze mikrokontrolera. Program je moguće izmjenjivati pomoću Arduino softvera koji ujedno služi i za kalibraciju senzora, odabir mjernog raspona senzora i formatiranje podataka koji se šalju kao izlaz iz IMU-a.

2.1.1.1. Akcelerometar

Senzor za mjerenje linearnih ubrzanja duž koordinatnih osi naziva se akcelerometar. Akcelerometar može mjeriti ubrzanja duž jedne, dvije ili tri osi. Danas se uglavnom koriste akcelerometri s mikroelektromehaničkim sustavima (MEMS akcelerometri) koji rade na principu mjerenja inercijske sile. Osnovni dijelovi takvog akcelerometra su tijelo mase *m* koje je pričvršćeno za kućište senzora pomoću opruge i prigušivača, slika 5. Kada senzor ubrzava tijelo mase *m* se pomiče u smjeru suprotnom od smjera kretanja te je njegov pomak proporcionalan ubrzanju. Pomak tijela se obično mjeri pomoću kapacitivnog, otporničkog ili piezoelektričnog senzora pomaka. Akcelerometri mjere ubrzanje uzrokovano statičkim (gravitacijska) i dinamičkim (gibanje, vibracije) silama te je zbog toga bitno naglasiti kako akcelerometri zapravo mjere razliku između ubrzanja sile teže i ubrzanja uzrokovanog kretanjem. Akcelerometar koji miruje na horizontalnoj podlozi će zato uvijek mjeriti ubrzanje sile teže koje će biti jednako + ili – 1 g ($1g \approx 9.81\ m/s^2$), ovisno o tome kako su definirani pozitivni smjerovi koordinatnih osi akcelerometra (slika 4) dok će akcelerometar koji je u slobodnom padu davati očitanje od 0 g.



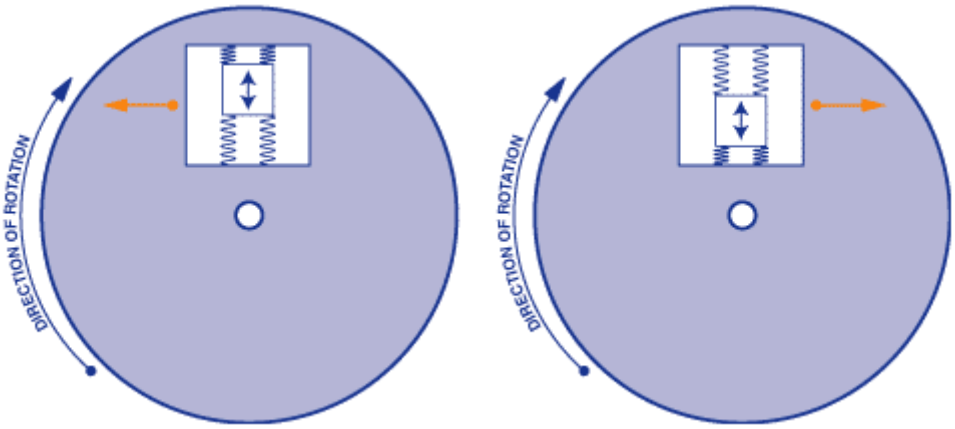
Slika 5. Unutarnja struktura akcelerometra [1]

U korištenom IMU-u se nalazi troosni MEMS akcelerometar ADXL 345 proizvođača Analog Devices (slika 5) mjernog raspona ±16 g koji je za potrebe rada ograničen na ±8 g budući da mjerena ubrzanja nisu prelazila te vrijednosti, a rezolucija senzora se smanjuje s odabirom većeg mjernog raspona.

Slika 6. Akcelerometar ADXL 345 u usporedbi s kovanicom [2]

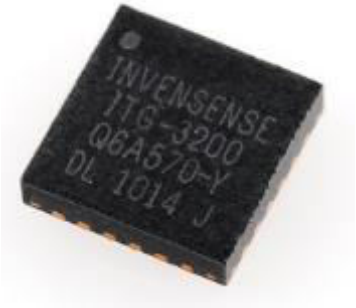
2.1.1.2. Žiroskop

Senzor za mjerenje kutnih brzina oko koordinatnih osi naziva se žiroskop. Žiroskop može mjeriti kutne brzine oko jedne, dvije ili tri osi. Danas se u potrošačkoj elektronici uglavnom koriste MEMS žiroskopi s vibrirajućim tijelom mase m koje se pomiče uslijed pojave Coriolisovog efekta, slika 7. Iz pomaka tijela mase m izračunava se kutna brzina. Žiroskopi se obično koriste za stabilizaciju objekata mjerenjem malih promjena kutnih brzina oko pojedinih osi objekta kao što su zrakoplovi, bespilotne letjelice (dronovi), samobalansirajući roboti i sl.



Slika 7. Princip rada žiroskopa [3]

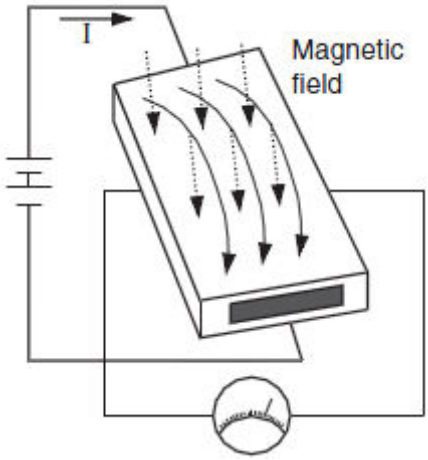
U korištenom IMU-u se nalazi troosni MEMS žiroskop ITG-3200 proizvođača InvenSense prikazan na slici 7. koji ima mjerni raspon od ±2000 °/s. Kutna brzina se mjeri oko koordinatnih osi IMU-a prikazanih na slici 4, a predznak brzine je određen strelicama prikazanim na slici 3.



Slika 8. MEMS žiroskop ITG-3200 [3]

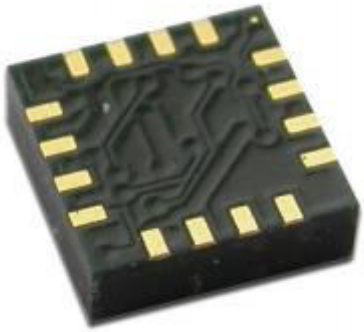
2.1.1.3. Magnetometar

Senzor za mjerenje gustoće i smjera magnetskog toka naziva se magnetometar. MEMS magnetometri obično mjere magnetski tok s obzirom na svoje tri osi. Magnetometri za potrošačku elektroniku se oslanjaju na pojavu Hallovog efekta (slika 9) pri mjerenju, takvi magnetometri ne mjere Hallov napon koji se pojavljuje uslijed skretanja elektrona već mjere promjenu otpora koja se događa zbog skretanja elektrona i nazivaju se magnetootporni magnetometri. Magnetometri se obično koriste u mobilnim telefonima, prijenosnim računalima te u raznim uređajima za navigaciju.



Slika 9. Skretanje elektrona uzrokovano Hallovim efektom [4]

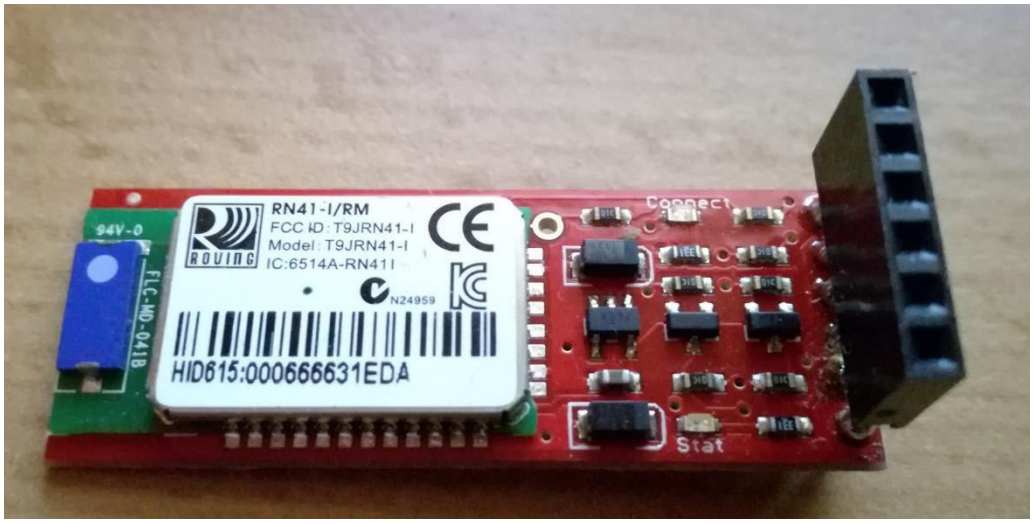
U korištenom IMU-u se nalazi troosni magnetometar HMC5883L proizvođača Honeywell prikazan na slici 10 koji ima mjerni raspon od ±8 gaussa i rezoluciju od 5 miligaussa. Posebno razvijena anizotropna magnetootporna (AMR) tehnologija omogućuje veliku osjetljivost po osima i bolju linearnost u odnosu na obične magnetooporne senzore. Osim gustoće magnetskog toka omogućeno je i mjerenje smjera Zemljinog magnetskog toka.



Slika 10. Honeywell HMC5883L magnetometar [5]

2.1.2. Bežični prijenos podataka

Za bežični prijenos podataka s IMU-a na računalo upotrijebljen je bluetooth modul SparkFun Bluetooth Mate Gold, prikazan na slici 11, koji je predviđen za lako spajanje na IMU pomoću strip konektora. Korišteni bluetooth modul omogućuje serijski prijenos podataka brzinama od 2400 do 115200 bps te predstavlja odličnu alternativu kablovima za serijski prijenos podataka. Bluetooth Mate Gold prenosi podatke na udaljenostima do približno 100 m bez fizičkih prepreka.



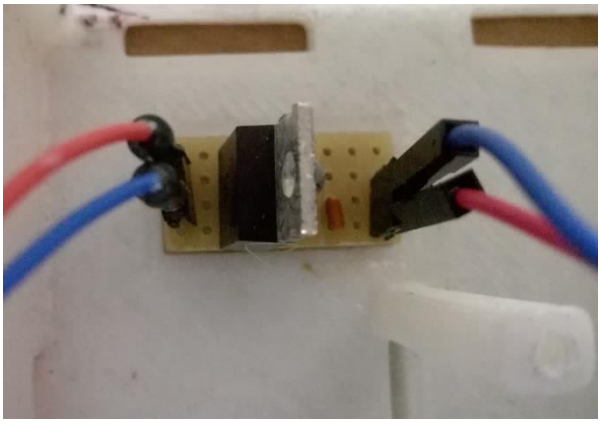
Slika 11. SparkFun Bluetooth Mate Gold modul

2.1.3. Napajanje sklopa

Kako bi se omogućilo napajanje cijeloga sklopa te dugotrajan rad i nesmetano mjerenje odabrana je punjiva litij-polimer baterija Tiger POWER Atomic-Platinum s dvije ćelije (7.4 V) ukupnog kapaciteta 900mAh, prikazana na slici 12.

Slika 12. Baterija Tiger POWER Atomic-Platinum

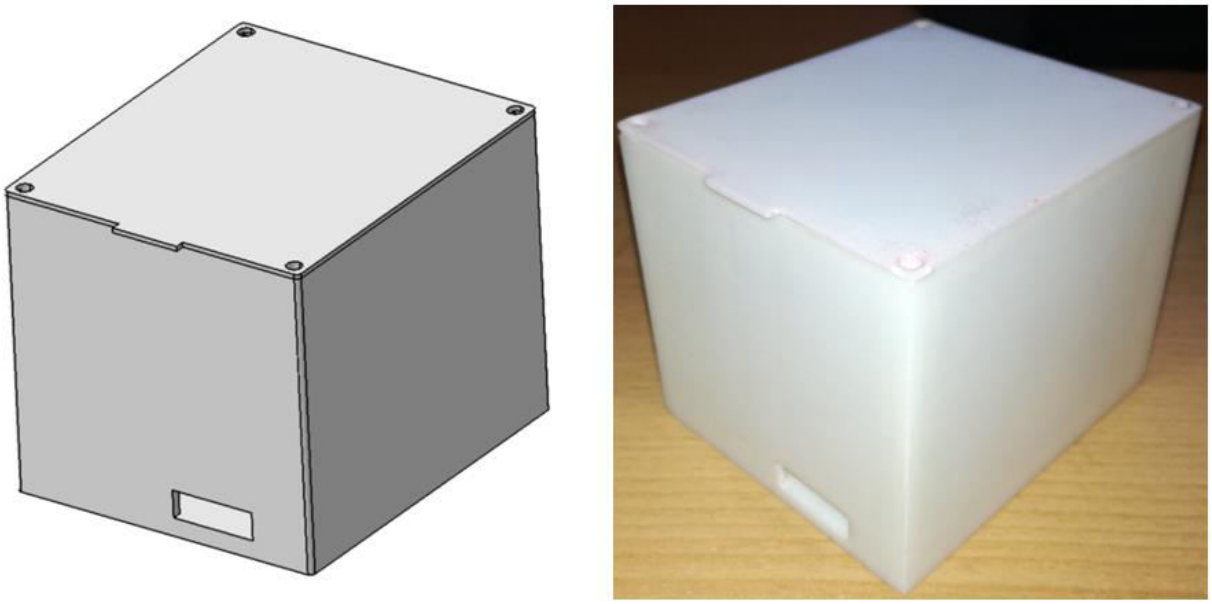
Kako bi se osigurao stabilni napon na ulazu u sklop, sastavljen je modul za stabilizaciju napona (slika 13) sa stabilizatorom napona LN7805 koji na ulazu u sklop osigurava stabilan napon od 5 V što je pogodno za korišteni IMU (ulazni napon 3.5 do 16 V) i bluetooth modul (ulazni napon 3.3 do 6 V).



Slika 13. Modul za stabilizaciju napona

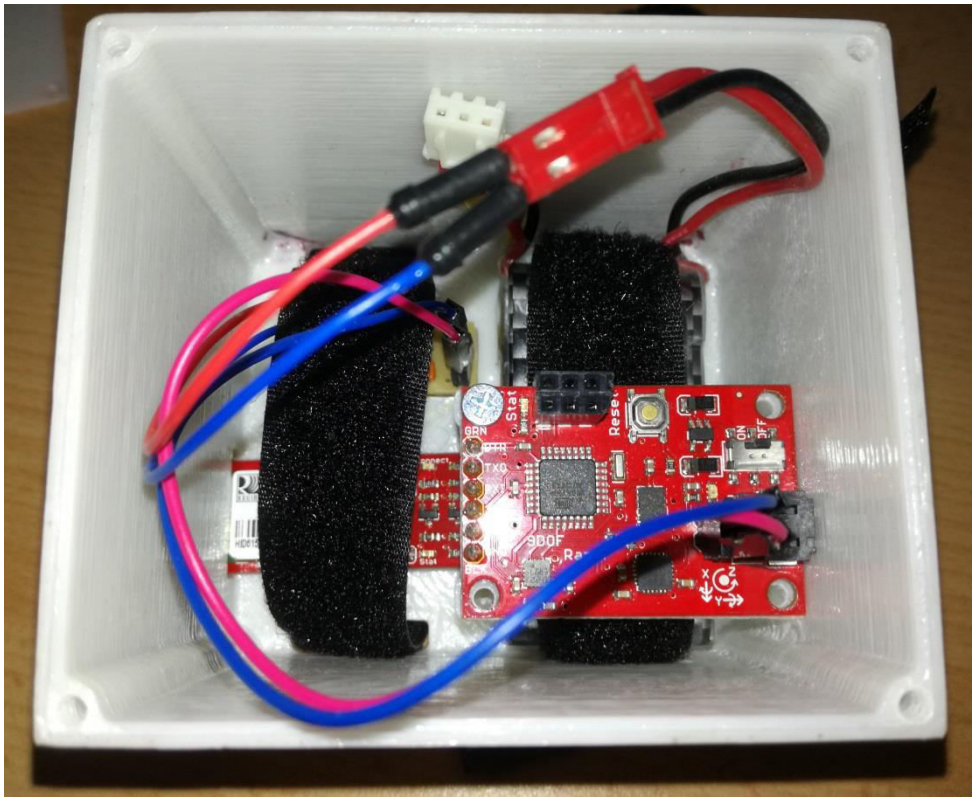
2.1.4. Kućište sklopa

Nakon što su prikupljeni i sastavljeni svi dijelovi sklopa za mjerenje konstruiran je 3D model kućišta (slika 14, lijevo) programskim paketom CATIA V5R20. Kućište je izrađeno u dva dijela od ABS-a (Akrilonitril Butadien Stiren) postupkom taložnog očvršćivanja polimera koji je poznatiji pod engleskim nazivom FDM (Fused Deposition Modeling), izrađeno kućište je također prikazano na slici 14, desno.



Slika 14. 3D model (lijevo) i izrađeno kućište (desno)

Unutrašnjost kućišta sa svim montiranim dijelovima je prikazana na slici 15.



Slika 15. Dijelovi montirani u kućište

2.2. NDI Polaris Vicra

Inercijski mjerni uređaji pružaju veliku fleksibilnost prilikom prepoznavanja pokreta zbog svojih malih dimenzija i bežičnog prijenosa podataka, ali se iz podataka koje daju još uvijek ne može dobiti pouzdano i precizno određivanje položaja u prostoru odnosno mapiranje prostora bez upotrebe dodatnih mjernih uređaja. Da bi robot mogao ponoviti prepoznatu radnju mora znati u koji položaj u prostoru treba doći te je stoga potrebno na neki način određivati taj željeni položaj, to je omogućeno optičkim sustavom Polaris Vicra prikazanim na slici 16.

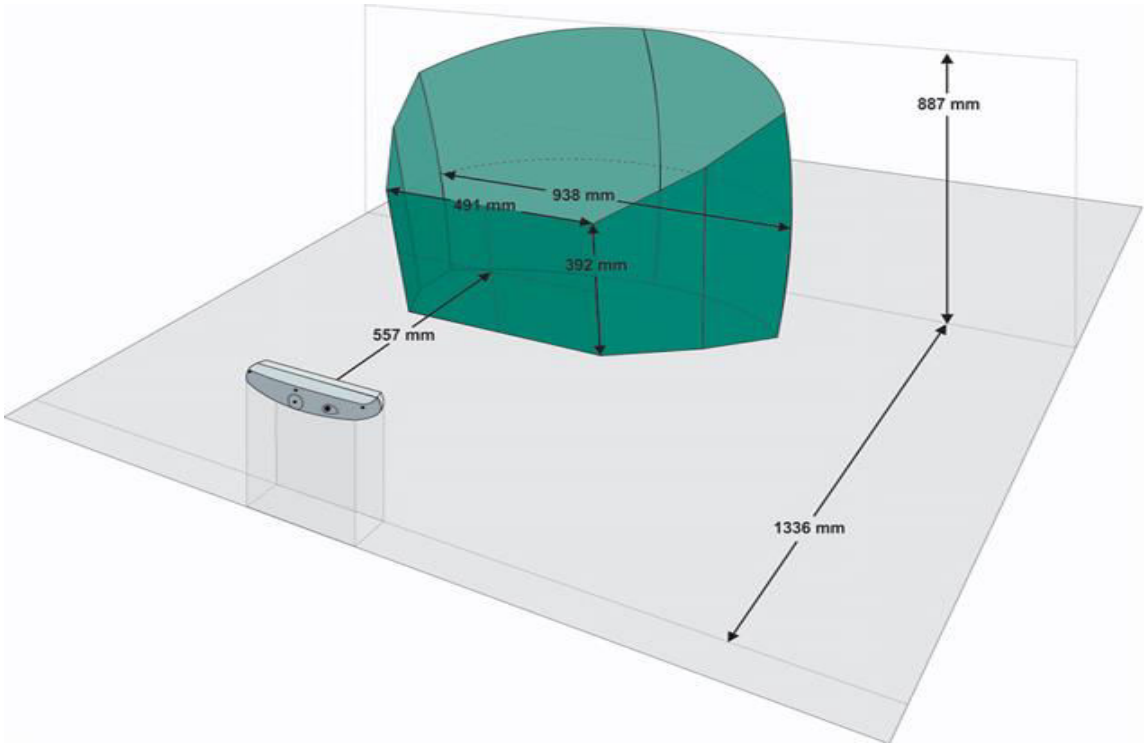
Slika 16. Polaris Vicra [7]

Polaris Vicra je stereovizijski optički sustav za praćenje kojeg proizvodi tvrtka Northern Digital Inc. (NDI). Navedeni sustav omogućuje određivanje položaja i orijentacije posebnih aktivnih ili pasivnih alata koji na sebi imaju tzv. markere pomoću kojih se određuju položaj i orijentacija u radnom području uređaja. Optički sustav se sastoji od izvora infracrvene svjetlosti i dvije kamere čiji senzori su sposobni detektirati infracrvenu svjetlost koje dolazi od markera pozicioniranih na alatima. Markeri mogu biti pasivni (retroreflektivne sfere) ili aktivni koji sami emitiraju infracrvenu svjetlost. Orijentaciju alata je moguće odrediti s minimalno tri markera pa je to ujedno i minimalan broj potrebnih markera na alatu. U radu su korišteni alati s pasivnim retroreflektivnim markerima prikazani na slici 17. Gore je prikazan alat montiran na spužvu (alat M2), dolje lijevo alat montiran na ploču (alat M3), a dolje desno alat montiran na sklop za brisanje na robotu (alat M1). Alat montiran na spužvu služi za pohranu položaja i orijentacije prilikom radnje brisanja koju obavlja korisnik, a alati montirani na robotu i ploči za izračun koordinata u koje robot mora doći te kasnije oponašanje

gibanja robotom. Alat montiran na ploči je originalni alat kojeg prodaje NDI dok su preostala dva konstruirana u prije spomenutom Laboratoriju za projektiranje izradbenih i montažnih sustava na fakultetu za potrebe znanstvenih istraživanja. Korišteni sustav se povezuje s računalom preko usb priključka i svakih 50 ms tj. frekvencijom od 20 Hz, daje nova očitanja pozicije i orijentacije alata u svom mjernom volumenu.

Slika 17. Korišteni alati s pasivnim retroreflektivnim markerima

Glavno ograničenje ovakvog optičkog sustava pri prepoznavanju ljudskih pokreta koje je trebalo uzeti u obzir prilikom primjene su ograničene dimenzije mjernog volumena prikazane na slici 18, te slučajna prekrivanja markera u nepogodnim konfiguracijama robota ili prilikom izvođenja promatranih pokreta.



Slika 18. Mjerni volumen sustava Polaris Vicra [8]

2.3. Logitech C210 web kamera

Prilikom implementacije prikaza mogućnosti rada na temelju naučenih postupaka korištena je Logitech C210 web kamera za pronalaženje dijelova ploče koje je potrebno pobrisati. Upotreba navedene kamere bila je potrebna zato što iz optičkog sustava Polaris Vicra nije moguće dobiti dovoljno kvalitetne slike za pronalaženje dijelova ploče koje je potrebno pobrisati već on daje vrlo tamne slike sivih tonova iz kojih nije moguće jasno razaznati pošarane dijelove ploče. Korištena web kamera se povezuje s računalom pomoću usb kabela te ima VGA rezoluciju od 640x480 piksela. Logitech C210 kamera prikazana je na slici 19.

Slika 19. Logitech C210 web kamera [9]

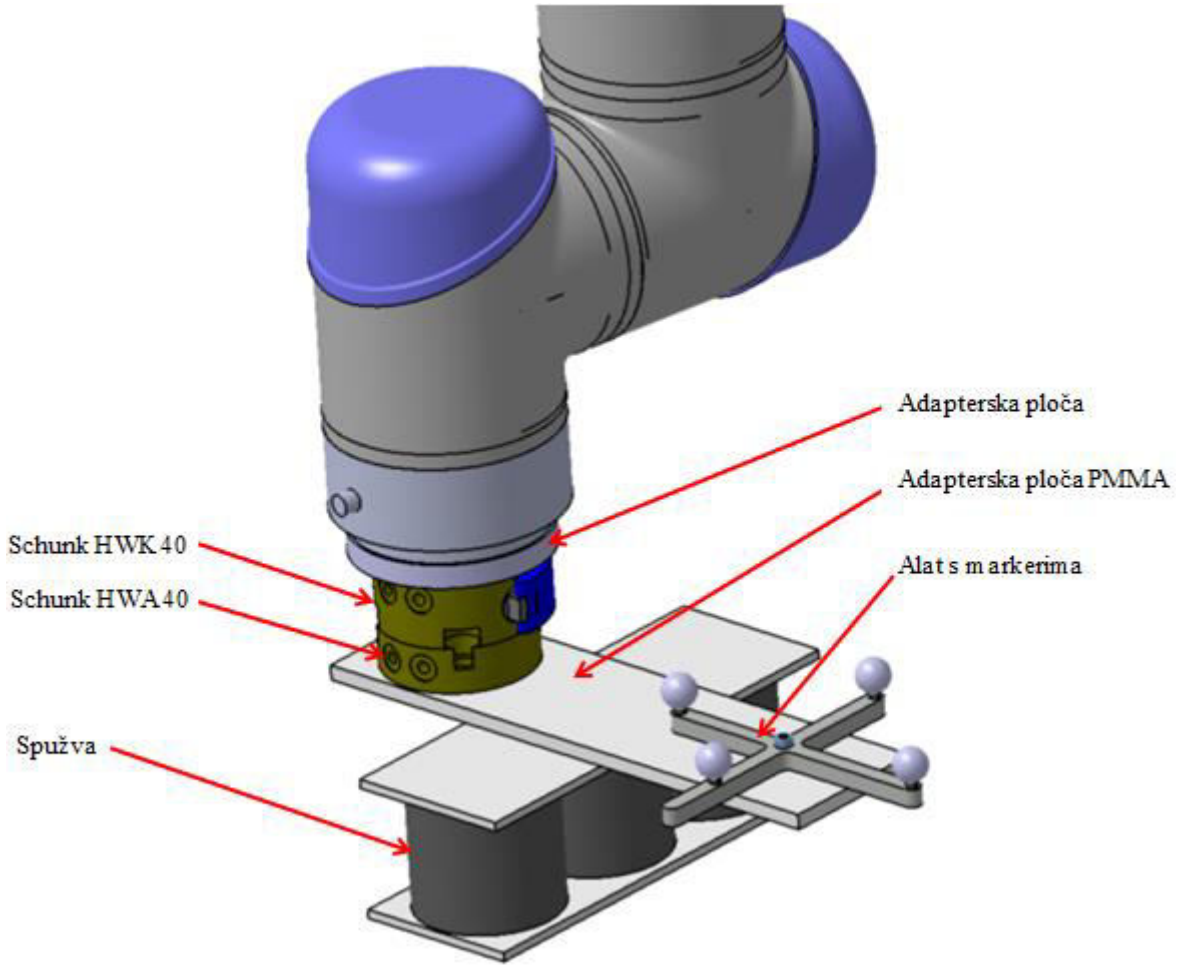
2.4. Robot UR5

Robot koji je korišten za eksperimentalna ispitivanja je UR5 tvrtke Universal Robots s pripadajućom upravljačkom jedinicom i privjeskom za učenje s ekranom osjetljivim na dodir, slika 20. Robot ima nosivost od pet kilograma i doseg od 850 mm te spada u skupinu kolaborativnih robota što znači da može raditi u neposrednoj blizini ljudi bez upotrebe dodatnih sigurnosnih zaštita. Dodatne karakteristike koje UR5 robot čine jednostavnim za upotrebu su mogućnost vođenja vrha alata robota rukom u *freedrive* modu rada (uhvati se vrh alat i dovede u željenu poziciju), te jednostavno i intuitivno korisničko sučelje privjeska za učenje koje omogućuje rad s robotom bez potrebe za nekim specifičnim predznanjem dok to kod drugih proizvođača robota nije slučaj. Osim preko privjeska za učenje programiranje robota je moguće izradom skripte ili putem C-API sučelja, a u ovom radu korišteno je programiranje pisanjem skripte na računalu koja se zatim šalje u kontroler robota ethernet vezom.

Slika 20. UR5 robot s upravljačkom jedinicom i privjeskom za učenje [10]

2.4.1. Alat za brisanje

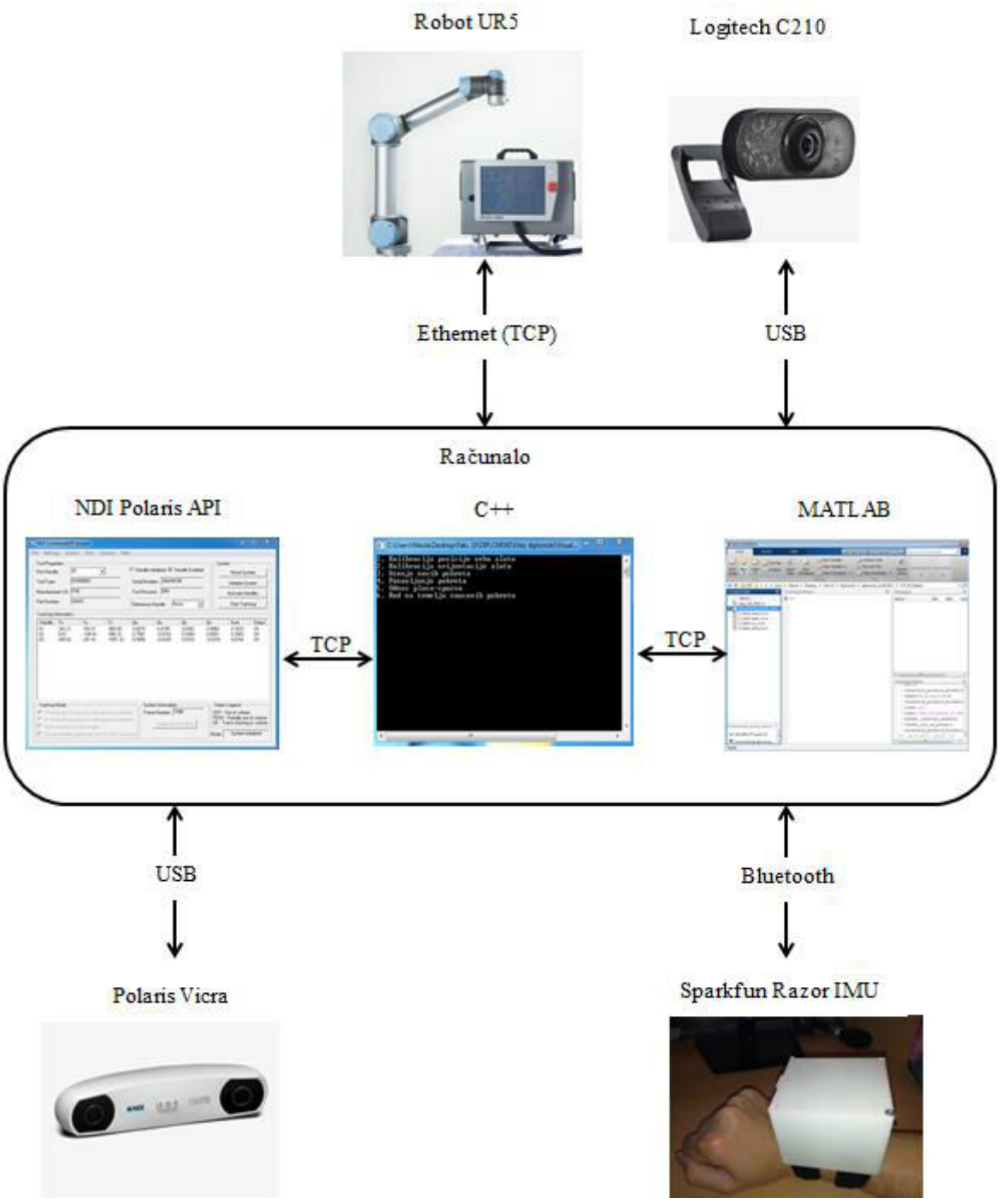
Zamišljeno je da robot na temelju klasifikacije radnji i prepoznavanja pokreta oponaša ljudske pokrete prilikom brisanja ploče. Kako bi se robotu omogućilo brisanje ploče bilo je potrebno projektirati alat za brisanje. Korišteni alat za brisanje prikazan je na slici 21. Alat se sastoji od izmjenjivača alata Schunk HWK 40 i pripadajućeg adaptera izmjenjivača alata Schunk HWA 40 koji su spojeni na prirubnicu robota adapterskom pločom. Na adapter izmjenjivača alata montirana je adapterska ploča izrađena od PMMA (poli(metil-metakrilat)) poznatijeg kao pleksiglas. Na tu adaptersku ploču je zalijepljena spužva i montiran alat s markerima za prepoznavanje alata za brisanje pomoću optičkog sustava Polaris Vicra. Izduženom adapterskom pločom od pleksiglasa na kraju koje je montiran alat s markerima smanjena je mogućnost prekrivanja markera dolaskom robota u nepogodne konfiguracije te je time omogućen jednostavniji rad sa sustavom.



Slika 21. Alat za brisanje montiran na robota

3. UPRAVLJAČKA PODRŠKA

Korišteni sustav sastoji se od velikog broja komponenata koje je potrebno povezati i čiji je rad potrebno uskladiti. Iz sheme sustava prikazane na slici 22 vidi se da se svi ostali elementi (robot, web kamera, IMU i Polaris) povezuju na računalo u kojem se odvija obrada i usklađivanje podataka svakog pojedinog sustava s ciljem klasifikacije i oponašanja pokreta.



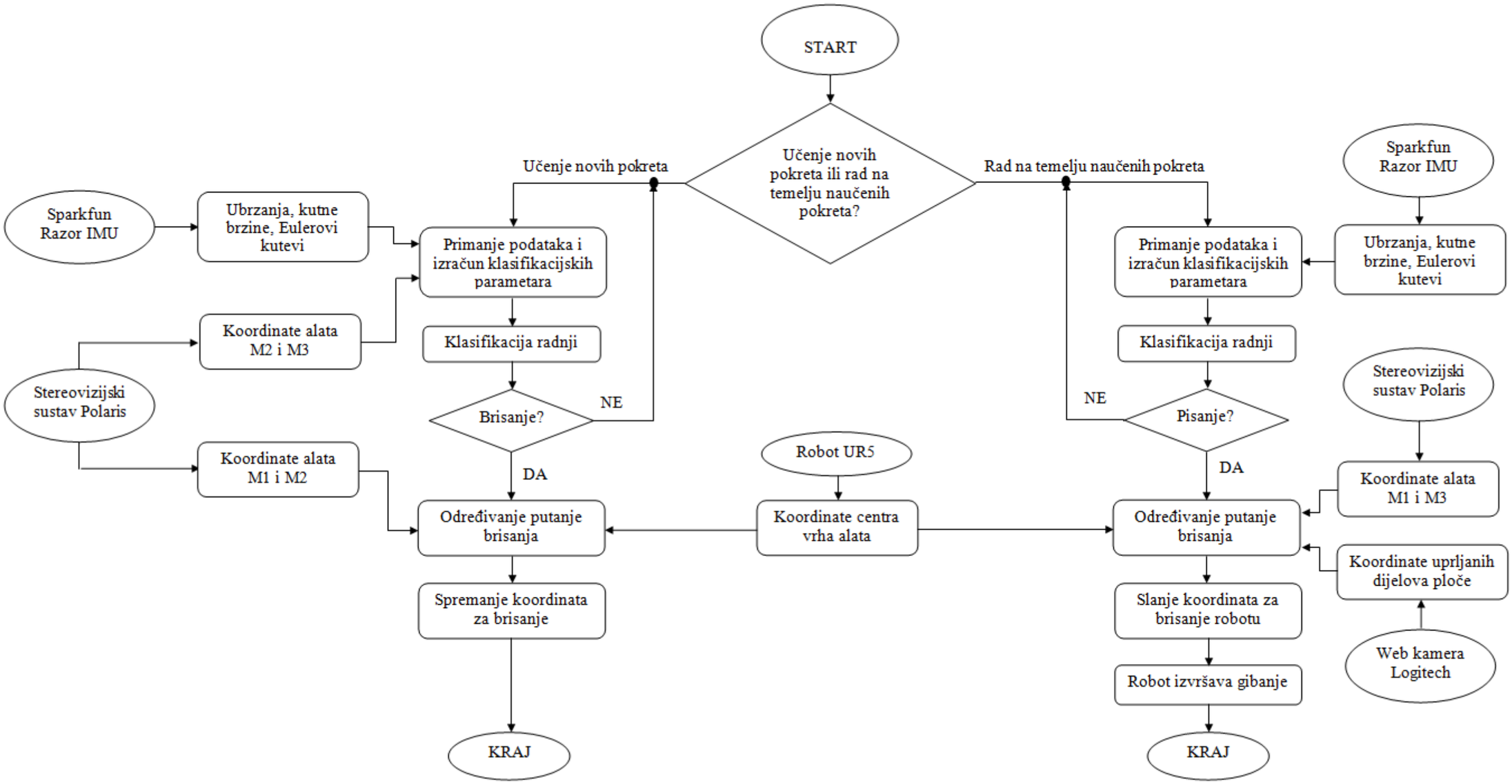
Slika 22. Shema sustava

3.1. Konceptualni opis rješenja upravljačke podrške za učenje robota

Cjelokupni sustav prikazan shemom na slici 22 služi za obavljanje niza zadataka kojima se omogućuje učenje robota radnji brisanja (pokretima potrebnim za brisanje) i brisanje ploče robotom na temelju naučenih pokreta. Dijagramom toka na slici 23 prikazan je zamišljeni način rada sustava. Prije nego što sustav može krenuti s normalnim radom prikazanim slikom 23 potrebno je sve njegove elemente kalibrirati i podesiti kako je opisano u poglavlju 4. Nakon kalibracije sustav radi na način da se prilikom njegovog pokretanja odabire željeni način rada (*Učenje novih pokreta* ili *Rad na temelju naučenih pokreta*). Način rada nazvan *Učenje novih pokreta* omogućava učenje robota pokretima brisanja ploče izvođenjem željenih pokreta brisanja. U slučaju odabira načina rada nazvanog *Učenje novih pokreta* računalo prima podatke iz IMU-a o ubrzanjima, kutnim brzinama i Eulerovim kutevima oko osi IMU-a koji se dobivaju prilikom izvođenja određenih pokreta, te podatke o položaju spužve za brisanje (alat M2) u odnosu na ploču (alat M3) iz stereovizijskog sustava Polaris na temelju kojih se određuje da li su spužva i ploča u dodiru. Na temelju primljenih podataka izračunavaju se klasifikacijski parametri te se provjerava da li se dogodila radnja brisanja. U slučaju da se nije dogodila radnja brisanja sustav se vraća na početak i ponovno započinje snimanje podataka, a ako se dogodila radnja brisanja određuje se putanja kojom robot treba proći da bi ponovio radnju brisanja. Putanja se određuje pomoću podataka dobivenih klasifikacijom radnji, položaja vrha alata robota i položaja spužve kojom korisnik briše ploču (alat M2) unutar koordinatnog sustava baze robota određenog u odnosu na alat M1. Nakon što se odredi putanja spremaju se njezine koordinate kako bi robot mogao obrisati ploču demonstriranim mu pokretima. U slučaju odabira načina rada nazvanog *Rad na temelju naučenih pokreta* ponovno se započinje primanje podatka iz IMU-a na temelju kojih se izračunavaju klasifikacijski parametri. Kako se u ovom slučaju provjerava da li se dogodilo pisanje po ploči odnos ploče i spužve nije bitan. Kada se na temelju klasifikacijskih parametara prepozna da je došlo do pisanja započinje određivanje putanje kojom robot mora proći da bi obrisao uprljani dio ploče. Putanja se u ovom slučaju određuje na temelju položaja uprljanog dijela ploče koji se pronalazi pomoću web kamere te potrebnog pomaka prethodno naučene putanje kako bi se obrisalo upravo uprljano područje ploče. Pomak prethodno naučene putanje izračunava se na temelju položaja vrha alata robota i položaja ploče (alat M3) unutar baznog koordinatnog sustava robota određenog u odnosu na alat M1. Nakon što se

odredi potrebna putanja brisanja koordinate za brisanje se šalju robotu koji izvršava gibanje i briše ploču.

U nastavku ovog poglavlja je opisan način povezivanja svakog elementa s računalom i akvizicija podataka korištenim softverima. Kako bi se svaki od uređaja povezao sa željenim softverom bilo je potrebno podesiti značajke bitne za komunikaciju između korištenih softvera i uređaja. Budući da se TCP protokol koristio prilikom povezivanja robota i računala, ali i za komunikaciju između korištenih softvera na računalu najprije je dan njegov kratki opis, a zatim opis podešavanja komunikacije između svakog pojedinog uređaja i korištenog softvera odnosno između korištenih softvera te načini akvizicije podataka.



Slika 23. Način rada sustava

3.2. TCP/IP protokol

TCP (eng. *Transmission Control Protocol*) je jedan od osnovnih protokola unutar IP grupe protokola. Korištenjem TCP protokola uspostavlja se veza između dva uređaja koji su povezani na istu računalnu mrežu, te se putem te ostvarene veze zatim prenose podatci. Stoga ovaj protokol spada u grupu tzv. spojnih protokola, za razliku od bespojnih protokola kakav je primjerice UDP (eng. *User Datagram Protocol*). TCP garantira pouzdanu isporuku podataka u kontroliranom redoslijedu od pošiljatelja prema primatelju. Osim toga, TCP pruža i mogućnost višestrukih istovremenih povezivanja više klijenata prema jednoj aplikaciji na serveru. [11]

Kako bi se uspostavila veza između klijenta i poslužitelja (servera) potrebno je znati IP adresu i port koji koristi server na koji se klijent želi spojiti. IP adresa se koristi za definiranje uređaja na kojem se poslužitelj koristi dok je port 16 bitni broj (0 - 65535) kojim je određena aplikacija koja koristi poslužitelj. Portovi od 0 do 1023 su rezervirani te se njih ne smije koristiti prilikom definiranja nove veze TCP protokolom, navedeni portovi se koriste za neke dobro poznate aplikacije kao što je pretraživanje interneta (HTTP poslužitelj na portu 80) ili za rad na udaljenim računalima (Telnet poslužitelj na portu 23). Prilikom uspostave novog poslužitelja na mreži s kojom nismo upoznati treba provjeriti koji portovi su slobodni za korištenje kako ne bi dolazilo do pogrešaka u radu. Nakon što je definirana ispravna IP adresa i odgovarajući port započinje proces uspostavljanja veze definiran TCP protokolom koji se naziva *three-way handshake* čiji detalji nisu bitni za ovaj rad. Nakon uspješnog obavljanja procesa uspostavljanja veze klijent i poslužitelj mogu međusobno razmjenjivati podatke u oba smjera (oboje mogu slati i primati podatke).

3.3. MATLAB

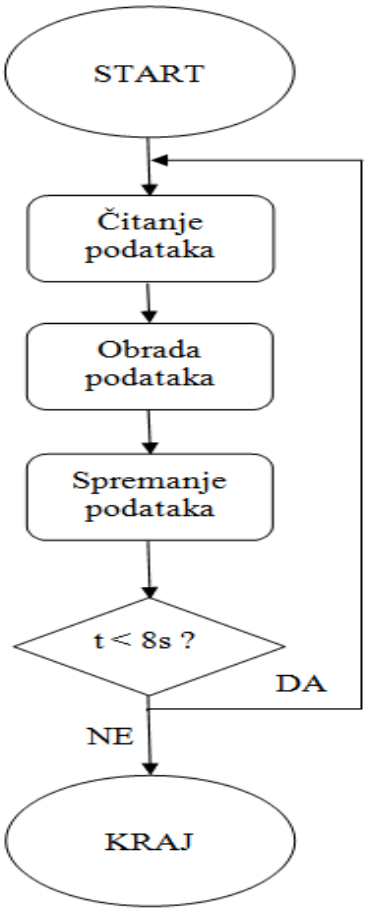
MATLAB je programski jezik visoke razine koji se uz brojne programske pakete koji ga proširuju primjenjuje za čitav niz različitih djelatnosti: programiranje, obrada signala i slike, grafičko prikazivanje rezultata, statističke obrade, analizu u vremenskoj i frekvencijskoj domeni i još mnogo toga. Pri izradi ovog rada MATLAB je korišten za niz zadataka:

- povezivanje s IMU-om i prijem te obradu podataka koje on daje,
- prijem i obradu slike s web kamere,

- skupnu obradu podataka dobivenih IMU-om, optičkim sustavom Polaris Vicra i web kamerom te
- komunikaciju s C++ upravljačkim programom.

3.3.1. Prijem i obrada podatka iz IMU-a

Prvo je napravljen programski kod u MATLAB-u koji služi za očitavanje podataka koje šalje bluetooth modul nosivog mjernog sklopa te obradu i spremanje tih podataka. Kako bi se podaci s bluetooth modula mogli očitati potrebno je prvo učitati bluetooth objekt *s11.mat* kao varijablu u MATLAB. Bluetooth objekt je zapravo datoteka koja sadrži podatke potrebne za povezivanje s bluetooth modulom i definira značajke komunikacije (naziv bluetooth modula, na koji virtualni serijski priključak računala se spaja, brzina prijenosa podataka, veličina ulaznog i izlaznog *buffera* i sl.). Nakon učitavanja bluetooth objekta potrebno je povezati računalo i modul naredbom *fopen*. Kada su računalo i bluetooth modul povezani pokreće se program *Citanje_podataka.m* koji učitava podatke o izmjerenim ubrzanjima, kutnim brzinama i Eulerovim kutevima koji se javljaju tijekom izvođenja promatranih pokreta. Podaci o ubrzanjima i kutnim brzinama se zatim dodatno obrađuju kako bi se dobili iznosi ubrzanja izraženi u g (1 g = 9.81 m/s^2) te iznosi kutnih brzina izraženi u °/s. Dodatno se računa rezultantno ubrzanje IMU-a jer akcelerometar mjeri samo ubrzanja po pojedinim osima, kao i iznosi pravog ubrzanja budući da akcelerometar cijelo vrijeme daje očitanja koja sadrže i komponentu ubrzanja gravitacijske sile. Nakon obrade podaci se spremaju u vektore i služe za kasniju analizu promatranih pokreta. Senzori mjere frekvencijom od 50 Hz što znači da se podaci o ubrzanjima, kutnim brzinama i Eulerovim kutevima zapisuju svakih 20 ms. Za potrebe eksperimentalnih mjerenja program završava nakon 8 s budući da se to pokazalo kao vrijeme dovoljno za obavljanje promatranih radnji. Za pokretanje novog mjerenja potrebno je ponovno pokrenuti program *Citanje_podataka.m*. Na slici 24 prikazan je dijagram toka programa *Citanje_podataka.m*.

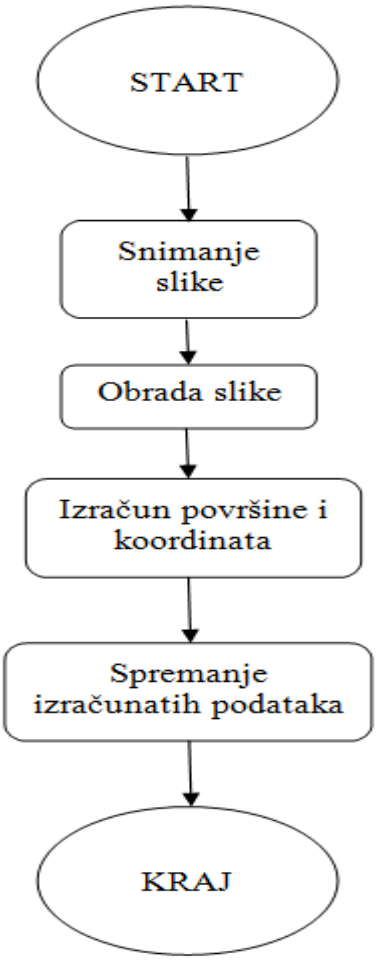


Slika 24. Dijagram toka programa za prikupljanje podataka iz IMU-a

3.3.2. Prijem i obrada slike s web kamere

Web kamera se s računalom povezuje preko usb kabela te je za dobivanje slike u MATLAB-u potrebno definirati objekt koji opisuje kameru. Objektom se definira tip adaptera kamere (*Windows Video* u slučaju Logitech C210 kamere) te format (MJPG) i veličina slike (640x480 piksela). Definirani objekt (*cam.mat*) je potrebno učitati kao varijablu u MATLAB-u te zatim povezati kameru s MATLAB-om naredbom *fopen*. Program *Obrada_slike.m* služi za pronalaženje uprljanih dijelova ploče koje je potrebno pobrisati kao i za povezivanje obrazaca kretanja tijekom radnje brisanja s različitim veličinama i oblicima površina koje je potrebno pobrisati. Kamerom se dobiva slika koju je potrebno zakrenuti i izrezati kako bi se izdvojilo područje ploče koje se želi promatrati, nakon izdvajanja željenog područja unutar

njega se traže uprljani dijelovi ploče pronalaskom tamnih piksela pošto je korištena bijela ploča i crni marker za pisanje. Nakon što su pronađeni svi tamni pikseli stvara se pravokutnik koji obuhvaća područje koje je potrebno obrisati te se računaju njegove dimenzije i koordinate njegovih vrhova na slici. Dijagram toka programa *Obrada_slike.m* prikazan je na slici 25.



Slika 25. Dijagram toka programa za snimanje i obradu slike s web kamere

3.3.3. Komunikacija s C++ upravljačkim programom

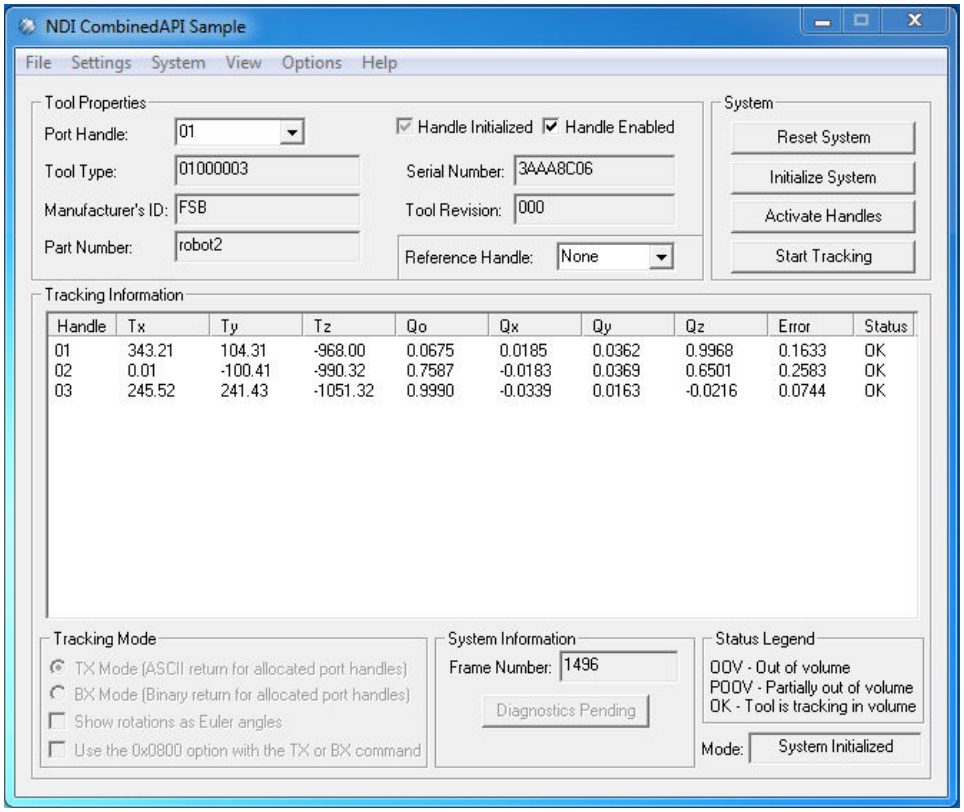
Učenje robota promatranim pokretima zahtijeva sinkronizaciju podataka iz IMU-a i optičkog sustava Polaris, a kako bi sinkronizacija bila moguća potrebno je ostvariti komunikaciju između MATLABA, u kojem se odvija prijem i obrada podataka dobivenih IMU-om, i C++ programa u kojem se odvija obrada podataka iz optičkog sustava Polaris. Komunikacija je ostvarena TCP protokolom pri čemu je C++ program definiran kao

poslužitelj, a MATLAB kao klijent te je korištena lokalna IP adresa poslužitelja 127.0.0.1. (budući da se oba korištena programa nalaze na istom računalu) i port 30006. Korišteni programski kodovi za povezivanje nalaze se u prilogu.

3.4. NDI Polaris API

NDI Polaris API (eng. *Application Program Interface*) je program koji služi za primanje podataka o položaju i orijentaciji definiranih alata u mjernom volumenu optičkog sustava Polaris. Program se skida sa stranica proizvođača te je otvorenog koda koji je moguće nadograđivati po želji, a napisan je u programskom jeziku C++. Unutar koda je potrebno definirati broj alata koji se koriste, a za koje je ujedno potrebno i učitati datoteke kojima su oni definirani (.rom datoteke). Alati se definiraju unutar softvera NDI 6D Architect kako je opisano u poglavlju 4.3.1.. Dodatno je iskorišten kod koji definira Polaris API kao TCP klijent razvijen u [12]. Tim kodom je pritiskom na *Start Tracking* omogućeno spajanje na poslužitelj koji je definiran unutar C++ upravljačkog programa i slanje podataka o položaju i orijentaciji svih alata u obliku tekstualne poruke. [12]

Korisničko sučelje NDI Polaris API programa prikazano je na slici 26.



Slika 26. NDI Polaris API korisničko sučelje

3.5. C++ program

C++ program je izrađen pomoću softvera Visual Studio 2015. Dio C++ programa je preuzet iz [12]. Preuzeti dio programa omogućuje prijem podataka o položaju i orijentaciji alata definiranih u optičkom sustavu Polaris iz NDI Polaris API programa. Preuzeti dio programa također omogućuje spajanje računala kao klijenta na poslužitelj u upravljačkoj jedinici robota i primanje podataka o trenutnoj poziciji i orijentaciji vrha alata robota u koordinatnom sustavu baze robota frekvencijom od 125 Hz. Također je omogućeno slanje naredbi ili cijele skripte robotu kao i spajanje upravljačke jedinice robota kao klijenta na poslužitelj na računalu pri čemu se iz računala robotu šalju tekstualne poruke koje se zatim obrađuju programom u upravljačkoj jedinici robota i upravljaju gibanjem robota. Za detalje o povezivanju računala i robota kao i detalje o načinu akvizicije podataka iz upravljačke jedinice robota i NDI Polaris API programa pogledati u [12]. U prilogu se nalaze izmijenjeni i dodani dijelovi C++ programu iz [12].

4. KALIBRACIJA I PODEŠAVANJE SUSTAVA

Za ispravan rad sustava potrebno je sve njegove elemente kalibrirati kako bi se uklonile razne greške i netočnosti te uskladio rad pojedinih elemenata sustava. Za kalibraciju sustava kao i izračun položaja i orijentacije u koje robot treba doći te izračun odgovarajućih akceleracija prilikom obavljanja različitih pokreta potreban je složen matematički aparat čiji pregled je dan u prvom dijelu ovog poglavlja, a u kasnijim dijelovima poglavlja je opisan proces kalibracije pojedinih dijelova sustava.

4.1. Matematička podloga

4.1.1. Eulerovi kutevi

Prilikom promatranja gibanja nekog tijela u prostoru potrebno je definirati odgovarajuće koordinatne sustave za opis tog gibanja. Obično se koriste dva koordinatna sustava, lokalni koordinatni sustav promatranog tijela (koji se giba zajedno s promatranim tijelom) i globalni koordinatni sustav u kojem se tijelo giba i koji je fiksan. Orijentacija lokalnog koordinatnog sustava unutar globalnog koordinatnog sustava se često opisuje pomoću tri uzastopne rotacije tijela oko osi njegovog lokalnog koordinatnog sustava. Spomenute rotacije se nazivaju Eulerovim kutevima. Redoslijed rotacija je vrlo bitan jer različiti redoslijedi rotacija daju različite orijentacije jednog koordinatnog sustava u odnosu na drugi. Postoji 12 različitih redoslijeda rotacija koji se primjenjuju, a u ovome radu je korišten x-y-z redoslijed rotacija pri čemu se kut rotacije oko x-osi (ϕ) naziva valjanje, oko y-osi (θ) poniranje, a oko z-osi (ψ) skretanje. Primjer osi i kuteva zakreta oko njih prikazan je na slikama 3 i 4.

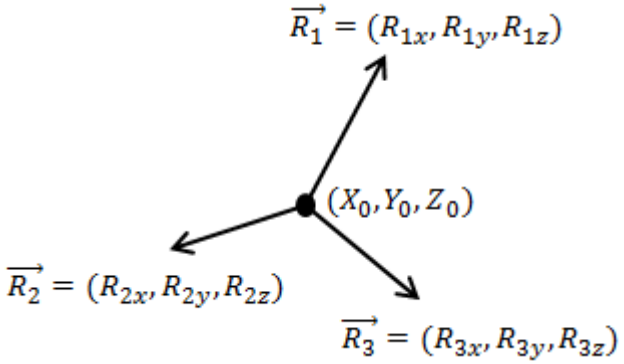
4.1.2. Matrica rotacije

Matricom rotacije **R** dimenzija 3 x 3 definira se orijentacija jednog koordinatnog sustava u odnosu na drugi u prostoru**.** Opći oblik matrice rotacije je:

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix}. \quad (1)$$

Pri čemu svaki stupac matrice predstavlja jedan trodimenzionalni vektor kojim je definirana pojedina koordinatna os zakrenutog koordinatnog sustav kako je prikazano na slici 27.

$$\boldsymbol{R} = \left[\overrightarrow{R_1}\ \overrightarrow{R_2}\ \overrightarrow{R_3}\right] \quad (2)$$



Slika 27. Koordinatni sustav opisan s tri vektora

4.1.3. Transformacija Eulerovih kuteva u matricu rotacije

Množenjem triju matrica rotacije od kojih svaka predstavlja zakret oko pojedine osi prema korištenom x-y-z redoslijedu rotacija dobiva se ukupna matrica rotacije **R** koja se često naziva i DCM matrica (eng. *Direction Cosine Matrix*). Matrice rotacije oko pojedine osi i ukupna matrica rotacije **R** dobivena njihovim množenjem prikazane su u nastavku.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{4}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

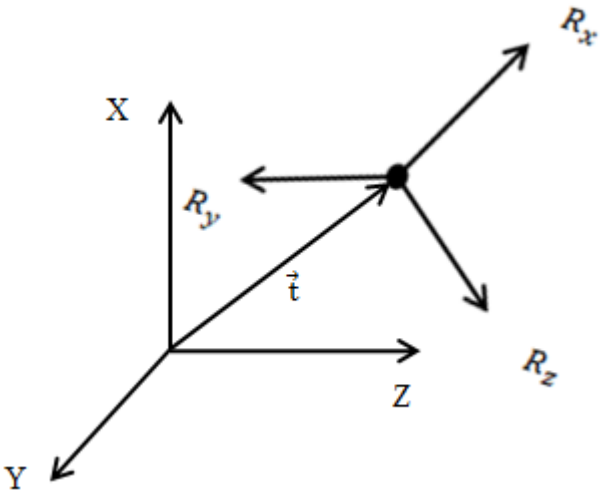
$$R = \begin{bmatrix} \cos\theta\cos\psi & \cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi - \cos\phi\sin\theta\cos\psi \\ -\cos\theta\sin\psi & \cos\phi\cos\psi - \sin\phi\sin\theta\sin\psi & \sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ \sin\theta & -\sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{6}$$

4.1.4. Matrica homogene transformacije

Matrica homogene transformacije je matrica dimenzija 4 x 4 koja osim rotacije definira i pomak (translaciju) jednog koordinatnog sustava u odnosu na drugi. Ona se sastoji od matrice rotacije **R** i vektora translacije **t** kako je prikazano sljedećom jednadžbom:

$$\mathbf{T} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{7}$$

Na slici 28 prikazana je definicija koordinatnog sustava u prostoru matricom homogene transformacije.



Slika 28. Definicija koordinatnog sustava matricom homogene transformacije

4.1.5. Transformacija između koordinatnih sustava

Kada želimo izraziti položaj i orijentaciju jednog koordinatnog sustava (O_1) u drugome (O_2) pri čemu su oba promatrana sustava definirana u nekom baznom koordinatnom sustavu (O) tada se služimo jednadžbama navedenim u nastavku:

$$ {}^{0}_{O_1}\mathrm{T} \cdot {}^{O_1}_{O_2}\mathrm{T} = {}^{0}_{O_2}\mathrm{T} \quad \text{i} \tag{8} $$

$$ {}^{O_1}_{O_2}\mathrm{T} = {}^{0}_{O_1}T^{-1} \cdot {}^{0}_{O_2}\mathrm{T} \quad . \tag{9} $$

4.1.6. Kvaternionski zapis

Optički sustav Polaris šalje podatke o položaju i orijentaciji definiranih alata unutar svog lokalnog koordinatnog sustava u obliku kvaterniona. Budući da je u ovom radu unutar C++ programa korištena knjižnica za linearnu algebru (Eigen) koja omogućuje pozivom funkcije transformaciju kvaterniona u matricu rotacije neće se detaljno opisivati matematički postupak navedene transformacije već je u nastavku dan samo kratki opis kvaterniona kao pojma.

Kvaternion je četverodimenzionalna veličina koja služi za prikazivanje rotacije u trodimenzionalnom prostoru. Upravo je četverodimenzionalnost najveća mana kvaterniona jer je ljudima teško zamisliti četiri dimenzije, a zapis kojim bi se najbolje mogao predstaviti kvaternion bio bi:

$$Q = \cos\left(\frac{\alpha}{2}\right) + i\cdot\left[x\cdot\sin\left(\frac{\alpha}{2}\right)\right] + j\cdot\left[y\cdot\sin\left(\frac{\alpha}{2}\right)\right] + k\cdot\left[z\cdot\sin\left(\frac{\alpha}{2}\right)\right], \quad (10)$$

gdje je:

- α - kut rotacije,
- x, y, z – komponente jediničnog vektora koji predstavlja os rotacije,
- i, j, k – kompleksne veličine koje predstavljaju rotaciju od 180° oko x, y i z osi. [13]

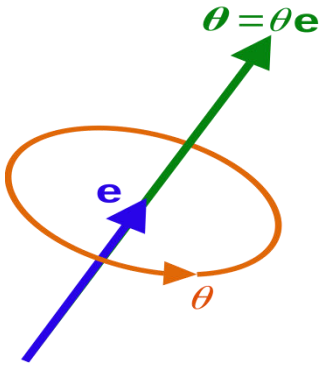
Unutar programskog koda koji se nalazi u prilogu je korišten vektorski format zapisa kvaterniona koji ima sljedeći oblik:

$$Q = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \\ x\cdot\sin\left(\frac{\alpha}{2}\right) \\ y\cdot\sin\left(\frac{\alpha}{2}\right) \\ z\cdot\sin\left(\frac{\alpha}{2}\right) \end{bmatrix}. \quad (11)$$

4.1.7. Axis – angle zapis

Robot UR5 iz svoje upravljačke jedinice šalje podatke o orijentaciji vrha alata u svom baznom koordinatnom sustavu u obliku *axis–angle* zapisa. *Axis-angle* zapis služi za prikaz rotacije u trodimenzionalnom prostoru pomoću tri veličine: jediničnog vektora **e** koji definira smjer osi oko koje se vrši rotacija i kuta θ koji definira veličinu zakreta oko te osi. Pošto je vektor **e** jediničan i ima hvatište u ishodištu koordinatnog sustava dovoljne su samo dvije veličine da se on definira te iz toga proizlazi da je kod *axis-angle* zapisa rotacija definirana s tri veličine. *Axis-angle* zapis prikazan je u obliku vektora:

$$\boldsymbol{\theta} = \theta\boldsymbol{e} = [R_x\ R_y\ R_z]\,. \quad (12)$$



Slika 29. Ilustaracija *axis-angle* zapisa [13]

Za potrebe transformacije iz *axis-angle* zapisa u matricu rotacije potrebno je izračunati kut rotacije θ i komponente jediničnog vektora e_x, e_y i e_z prema jednadžbama:

$$\theta = \sqrt{R_x^2 + R_y^2 + R_z^2}\ , \tag{13}$$

$$e_x = \frac{R_x}{\theta}, \tag{14}$$

$$e_y = \frac{R_y}{\theta}, \tag{15}$$

$$e_z = \frac{R_z}{\theta}. \tag{16}$$

Nakon izračuna prethodnih veličina koristi se funkcija iz Eigen knjižnice za linearnu algebru koja iz izračunatih podataka daje matricu rotacije **R**.

Transformacija iz matrice rotacije **R** u *axis-angle* zapis vrši se upotrebom sljedećih jednadžbi:

$$\theta = \arccos\left(\frac{1}{2} \cdot [R_{11} + R_{22} + R_{33} - 1]\right), \tag{17}$$

$$e_x = \frac{R_{32} - R_{23}}{2 \sin\theta}\ , \tag{18}$$

$$e_y = \frac{R_{13} - R_{31}}{2 \sin\theta}\ , \tag{19}$$

$$e_z = \frac{R_{21} - R_{12}}{2 \sin\theta}\ . \tag{20}$$

4.2. Podešavanje i kalibracija IMU-a

Program koji upravlja radom IMU-a preuzima se sa stranice proizvođača, no prilikom podešavanja i kalibracije senzora potrebno je u program unijeti odgovarajuće vrijednosti kako bi IMU ispravno radio. Mikrokontroler koji upravlja radom IMU-a programira se pomoću Arduino softvera koji ujedno služi i za prijenos programa u mikrokontroler. Za potrebe prijenosa programa u mikrokontroler potrebno je IMU spojiti s računalom. Budući da IMU koristi serijsku komunikaciju na računalu je potrebno imati serijski priključak, a budući da većina modernih računalu nema taj priključak potrebno je koristiti drugi način povezivanja. Ovdje je komunikacija ostvarena pomoću modula SparkFun FTDI Basic Breakout koji omogućuje serijsku komunikaciju preko usb priključka na računalu. Prije početka kalibracije senzora u programu koji upravlja radom IMU-a potrebno je odabrati željeni mjerni raspon akcelerometra (odabran raspon od ±8 g kako je prije spomenuto), te podesiti način

komunikacije s računalom (komunikacija pomoću bluetooth modula). Detaljne upute za podešavanje nalaze se u komentarima unutar samog koda programa te je za više informacija potrebno pogledati kod dostupan na stranicama proizvođača [14]. Nakon početnih podešavanja sastavljen je nosivi mjerni sklop kako bi IMU bio u okolini u kojoj će se kasnije koristiti, a time se povećala učinkovitost kalibracije.

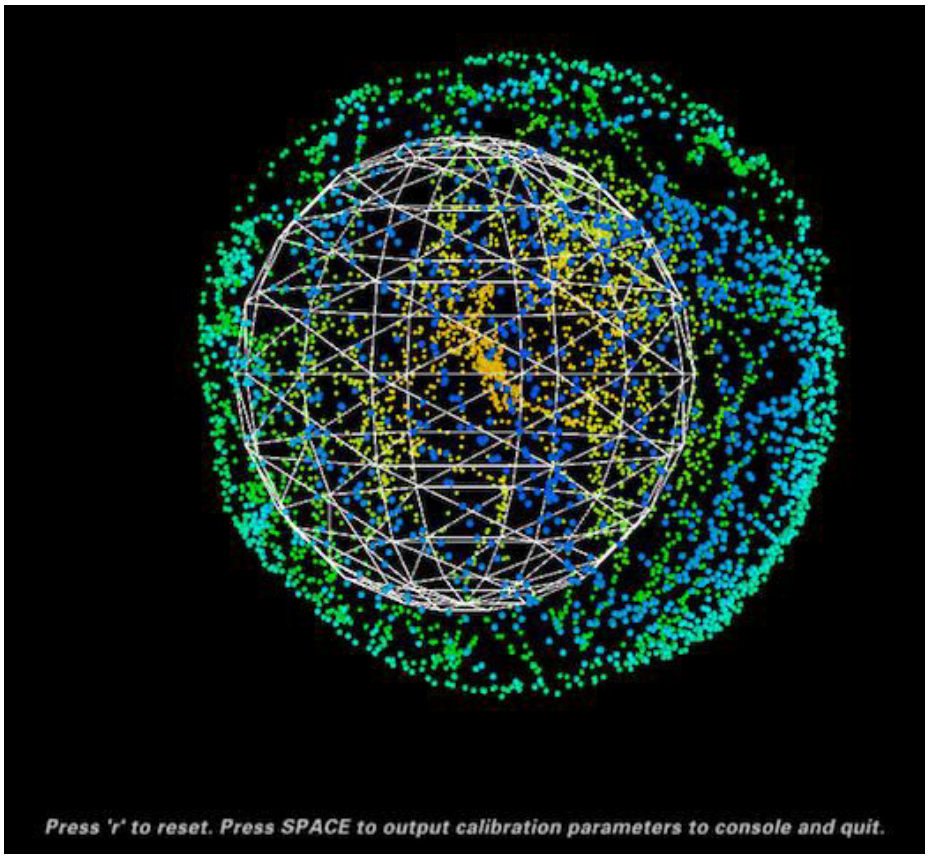
Svaki od korištenih senzora ima neke specifične greške prilikom mjerenja koje se u potpunosti ili djelomično uklanjaju kalibracijom. Uobičajene greške koje utječu na rad sva tri korištena senzora su:

- odmak od prave vrijednost (npr. senzor daje vrijednost nula, a trebao bi 0.5),
- različita pojačanja po osima senzora (npr. očitanje različitih vrijednosti ubrzanja po pojedinim osima, a zapravo je djelovalo jednako ubrzanje),
- postupna promjena očitanja uslijed utjecaja temperature (npr. senzor miruje, a očitanje se mijenja),
- nelinearnost,
- neokomitost osi te
- šum mjerenja.

Kod žiroskopa se još javlja i utjecaj linearnog ubrzanja na vrijednosti koje daje senzor, a magnetometri su osjetljivi na smetnje u magnetskom polju uzrokovane feromagnetskim materijalima koji se kreću zajedno sa senzorom ili su stacionarni u okolini senzora, magnetometri su također osjetljivi na promjenjive iznose električne struje koja protječe vodičima u njihovoj blizini. Za potrebe ovoga rada provedena je kalibracija koja uklanja greške odmaka od pravih vrijednosti te različita pojačanja po osima senzora. Dodatno je za magnetometar provedena kalibracija u svrhu uklanjanja grešaka koje uzrokuju feromagnetski materijali koji se kreću zajedno sa senzorom.

Kalibracija senzora se provodi po naputku sa stranica proizvođača [14] te se vrijednosti za kalibraciju koje je potrebno upisati u osnovni kod programa ispisuju na računalu upotrebom serijskog monitora u Arduino softveru. Kalibracija akcelerometra se provodi na način da se pojedina os akcelerometra usmjeri prema tlu odnosno da se preklapaju smjer osi i smjer djelovanja gravitacijske sile, dok je kalibracija žiroskopa nešto jednostavnija te je potrebno IMU ostaviti da miruje 10 s te očitati kalibracijske vrijednosti. Kalibracija magnetometra se provodi na sličan način kao i ona akcelerometra osim što je sada potrebno da se osi magnetometra poklope s osi magnetskog polja Zemlje koja je na sjevernoj polutci usmjerena

prema sjeveru i dolje. Kalibracija magnetometra u svrhu uklanjanja grešaka koje uzrokuju feromagnetski materijali koji se kreću zajedno sa senzorom provodi se na način da je senzor potrebno okretati u što više mogućih različitih položaja te se onda na temelju toga izračunavaju kalibracijske vrijednosti. Dodatna kalibracija magnetometra provodi se pomoću softvera Processing koji služi za vizualizaciju postignutih položaja (slika 30) kako bi se omogućila što kvalitetnija kalibracija.



Slika 30. Vizualizacija postignutih položaja u Processingu

Nakon očitavanja svih kalibracijskih vrijednosti, potrebno je te vrijednosti unijeti u kod programa koji upravlja radom IMU-a te prenijeti taj izmijenjeni program u mikrokontroler IMU-a kako je prethodno opisano.

4.3. Podešavanje i kalibracija sustava Polaris Vicra i robota

Kako bi se omogućilo spremanje i oponašanje putanje kojom prolazi spužva (na kojoj je alat s markerima) tijekom brisanja, dovođenje u odnos pozicije i orijentacije spužve i ploče te rad na temelju naučenih obrazaca kretanja potrebno je kalibrirati sustav na način da se

koordinatama definiranih alata iz sustava Polaris pomoću transformacija točno odrede koordinate u koje treba doći vrh alata robota unutar baznog koordinatnog sustava robota. Zbog toga je potrebno provesti niz radnji navedenih u nastavku:

- definirati alate s retroreflektivnim markerima,
- kalibrirati poziciju vrha alata robota,
- kalibrirati orijentaciju alata robota.

4.3.1. Definiranje alata s markerima

Definiranje alata koje sustav Polaris prepoznaje provodi se pomoću softvera NDI 6D Architect. Postupak definiranja alata je napravljen u vrlo jednostavnim koracima. Prvi koraci uključuju odabir korištenog sustava (Polaris) i vrste markera (pasivni ili aktivni) te dodjeljivanje imena novom alatu i odabir načina određivanja položaja alata (odabran je način koji omogućuje određivanje položaja sve dok su vidljiva barem tri od ukupno četiri markera na alatu). Sljedeći korak je postavljanje alata s markerima unutar mjernog volumena sustava i automatsko snimanje koordinata markera. Nakon snimanja koordinata markera definira se lokalni koordinatni sustav alata odabirom ishodišta u jednom od markera te dodatno definiranjem jedne od ravnina kartezijskog koordinatnog sustava odabirom još dva markera od kojih jedan definira smjer željene koordinatne osi, a drugi željenu ravninu pomoću koje se onda definira cijeli koordinatni sustav. Nakon definiranja lokalnog koordinatnog sustava potrebno je odrediti normale svakog retroreflektivnog markera čime se završava postupak definiranja alata i ukoliko alat prođe testiranje koje se provodi u zadnjem koraku on je spreman za upotrebu učitavanjem njegove .rom datoteke prilikom korištenja programa NDI Polaris API za određivanje položaja alata u mjernom volumenu sustava Polaris.

4.3.2. Kalibracija pozicije vrha alata robota

Izvršavanje željenog gibanja vrha alata robota (u ovom slučaju sredine površine za brisanje na spužvi) omogućava se definiranjem koordinatnog sustava alata robota za korišteni sklop alata. Koordinatni sustav alata robota definira se u odnosu na preddefinirani koordinatni sustav koji se nalazi u središtu prirubnice u osi zadnjeg (šestog) zgloba robota. Pozicija novog koordinatnog sustava definira se translacijama po osima preddefiniranog koordinatnog sustava. Kalibracija pozicije vrha alata robota provodi se dovođenjem željenog vrha alata

robota u istu poziciju u prostoru s barem tri različite konfiguracije robota, dok su u ovom slučaju korištene četiri različite konfiguracije, te izračunavanjem potrebnih translacija na temelju spremljenih pozicija i orijentacija za svaku od konfiguracija. Ovakav način kalibracije je omogućen upotrebom privjeska za učenje nakon čega je potrebno iznose translacija prepisati u datoteku *TCP_Position.txt* ili upotrebom preuzetog dijela C++ programa kod kojeg se vrijednosti translacija automatski spremaju u navedenu datoteku, a čiji se kod i detaljni opis može pronaći u [12]. Vrijednosti translacija se spremaju u datoteku kako prilikom novog pokretanja robota i C++ programa ne bi bilo potrebno ispočetka provoditi postupak kalibracije već se kalibracijske vrijednosti samo učitaju iz spremljene datoteke na početku programa.

4.3.3. Kalibracija orijentacije vrha alata robota

Orijentacija vrha alata robota definira se matricom rotacije u odnosu na preddefinirani koordinatni sustav u središtu prirubnice robota spomenut u 4.3.2.. Kako bi robot i optički sustav Polaris jednako vidjeli orijentaciju vrha alata robota definira se novi koordinatni sustav u prostoru koji je definiran u dva različita bazna koordinatna sustava (robota i Polarisa). Prije provođenja ove kalibracije potrebno je provesti kalibraciju pozicije vrha alata robota opisanu u 4.3.2.. Definicija novog koordinatnog sustava provodi se dovođenjem vrha alata robota u tri različite točke u prostoru čije se koordinate spremaju u dva navedena bazna koordinatna sustava te se zatim izračunava matrica rotacije koja povezuje ta dva bazna koordinatna sustava na način opisan u [12]. Transformacijom matrice rotacije u *axis-angle* zapis prema formulama 17 do 20 dobivaju se sve vrijednosti potrebne za definiranje orijentacije alata robota u odnosu na preddefinirani koordinatni sustav prirubnice robota. Izračunate vrijednosti se spremaju u datoteku *TCP_Orientation.txt* kako prilikom novog pokretanja robota i C++ programa ne bi bilo potrebno ispočetka provoditi postupak kalibracije. Ovakav način kalibracije je omogućen upotrebom preuzetog dijela C++ programa iz [12].

Postupkom kalibracije pozicije i orijentacije vrha alata robota zapravo se definira matrica homogene transformacije koordinatnog sustava vrha alata robota u odnosu na koordinatni sustav prirubnice robota.

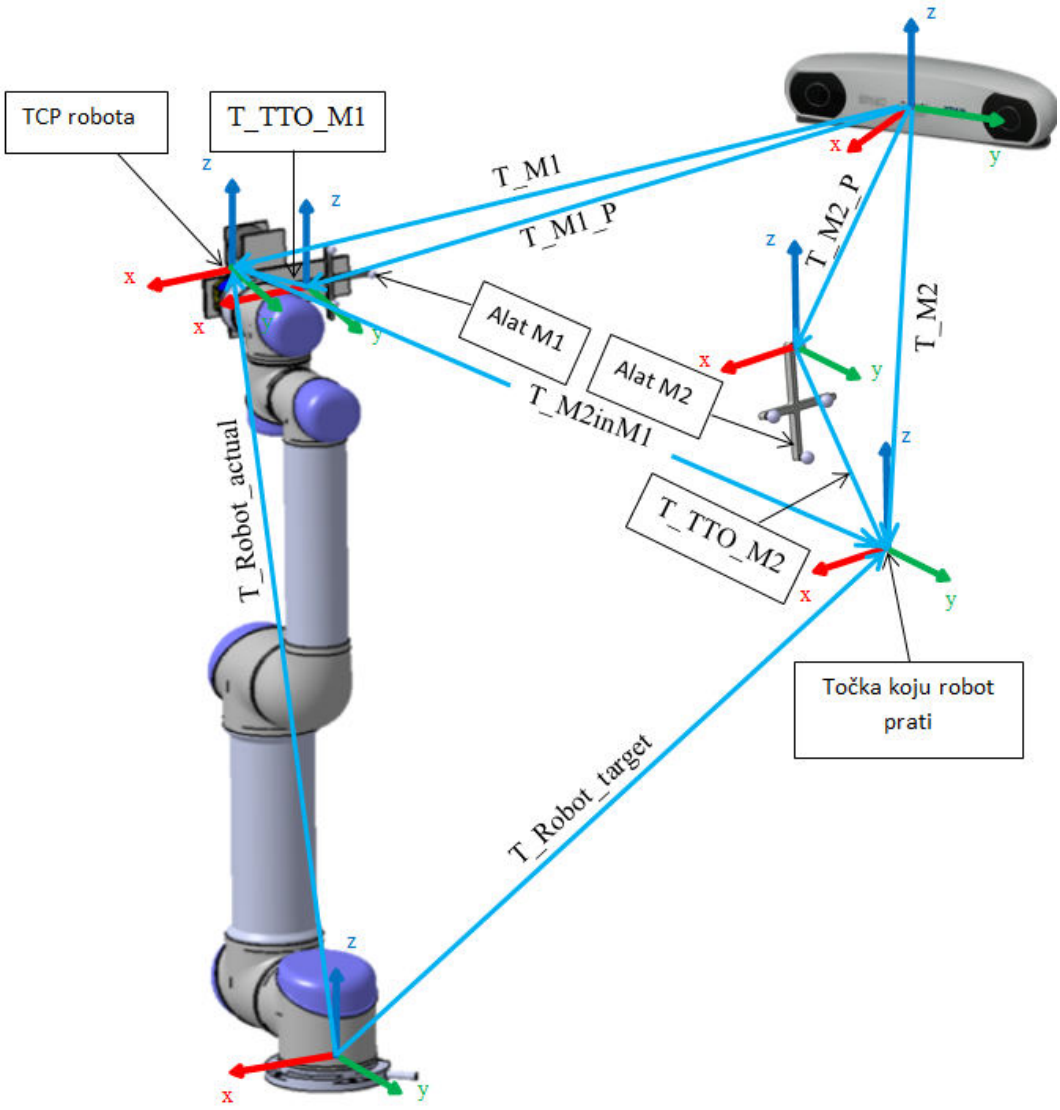
4.3.4. Izračun koordinata praćenja

Provedbom kalibracija opisanih u 4.3.1. do 4.3.3. omogućuje se izračun položaja u prostoru u koji robot treba doći da bi oponašao demonstrirane pokrete. Postupak izračuna

položaja u prostoru omogućen je preuzetim dijelom koda iz [12], a na slici 31 su prikazane sve matrice transformacije potrebne za izračun položaja u prostoru. Detalje izračuna pogledati u [12].

Uloga matrica:

- T_M1_P – pozicija i orijentacija alata M1 u baznom koordinatnom sustavu Polarisa,
- T_TTO_M1 – pomak iz ishodišta alata M1 u vrh alata robota (eng. *Tool Center Point - TCP*),
- T_M1 – pozicija i orijentacija vrha alata robota u baznom koordinatnom sustavu Polarisa,
- T_M2_P – pozicija i orijentacija alata M2 u baznom koordinatnom sustavu Polarisa,
- T_TTO_M2 – pomak iz ishodišta alata M2 u točku za praćenje,
- TM2 – pozicija i orijentacija točke za praćenje u baznom koordinatnom sustavu Polarisa,
- T_Robot_actual – pozicija i orijentacija vrha alata robota (TCP) u baznom koordinatnom sustavu robota,
- T_M2inM1 – pozicija i orijentacija točke za praćenje u koordinatnom sustavu alata M1,
- T_Robot_target – pozicija i orijentacija točke za praćenje u baznom koordinatnom sustavu robota. [12]

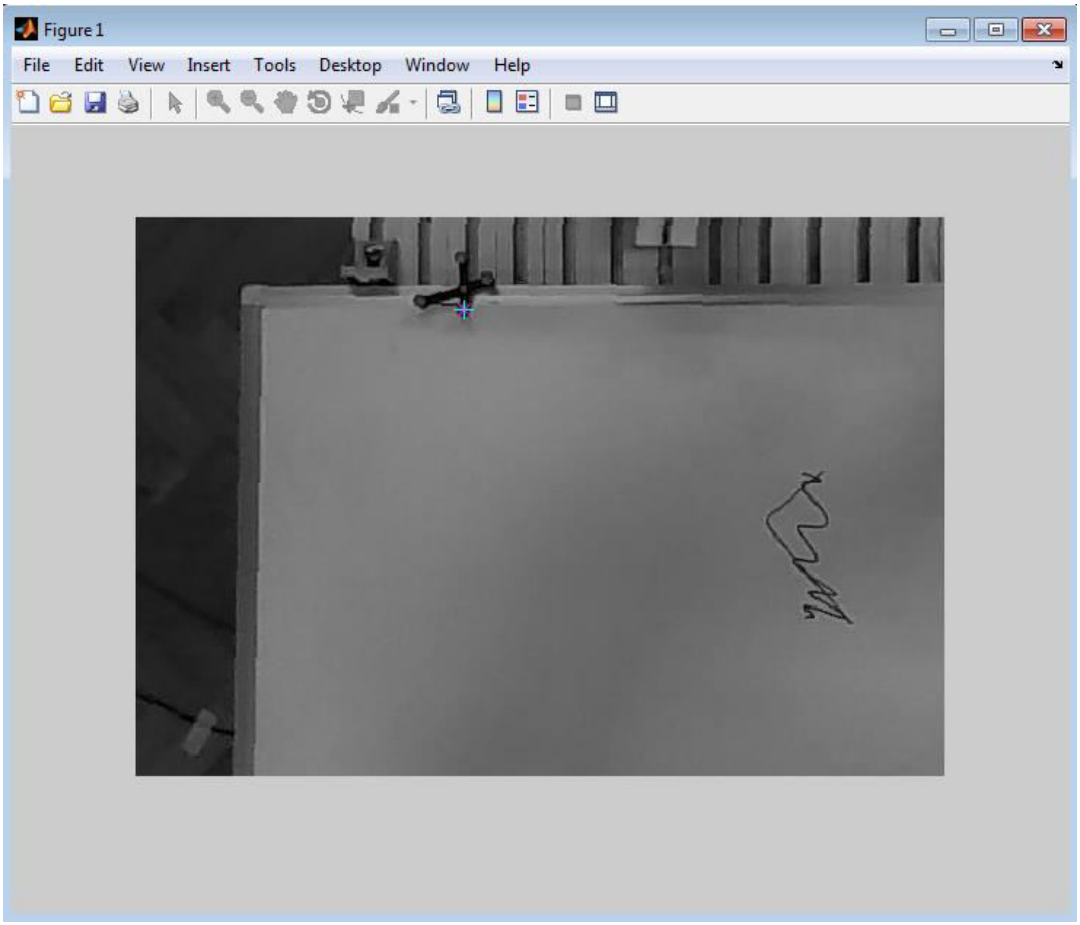


Slika 31. Matrice transformacije potrebne za izračun položaja u prostoru

4.4. Podešavanje i kalibracija web kamere

Web kamera služi za pronalazak uprljanih dijelova ploče koje je potrebno obrisati kako je prije navedeno i opisano. Kada su određeni dijelovi koje je potrebno obrisati potrebno je na neki način odrediti gdje se pronađeni dijelovi nalaze u radnom prostoru robota kako bi se robotu mogle zadati ispravne koordinate za brisanje. Zato je potrebno provesti kalibraciju kamere i povezati sliku kamere s koordinatnim sustavima optičkog sustava Polaris i robota. Kalibracija kamere se provodi na način da se nakon pozicioniranja kamere unutar njezinog vidnog polja stavi predmet poznatih dimenzija te se zatim snimi slika iz koje se onda pomoću

funkcije *getpts* ručnim odabirom klikom miša na rubne piksele u MATLAB-u odrede koordinate rubova poznatog predmeta. Nakon određivanja koordinata rubova jednostavno se dimenzija predmeta podijeli s brojem piksela između rubova te se na taj način dobije udaljenost koju predstavlja jedan piksel na slici. Kako korištena kamera ima dimenzije slike od 640x480 piksela a promatrano je prilično veliko područje, kalibracijom je određeno da jedan piksel na slici predstavlja udaljenost od ≈ 1.4 mm. Lokalni 2D koordinatni sustav slike povezuje se s koordinatnim sustavom alata M3 (definiran unutar optičkog sustava Polaris) koji je montiran na promatranu ploču na način da se ponovno pomoću funkcije *getpts* odredi položaj ishodišta koordinatnog sustava alat M3 unutar slike, kako je prikazano na slici 32, te se ta točka definira kao ishodište koordinatnog sustava slike. Na opisani način dobiveno je poklapanje koordinatnog sustava slike s lokalnim koordinatnim sustavom alata M3 čija se pozicija i orijentacija dobiva iz optičkog sustava Polaris. Opisani postupak kalibracije je dao zadovoljavajuću točnost u primjeni pa se nije provodila složenija kalibracija kamere koja bi u obzir uzela i distorziju slike.



Slika 32. Određivanje ishodišta koordinatnog sustava slike

5. UČENJE POKRETA

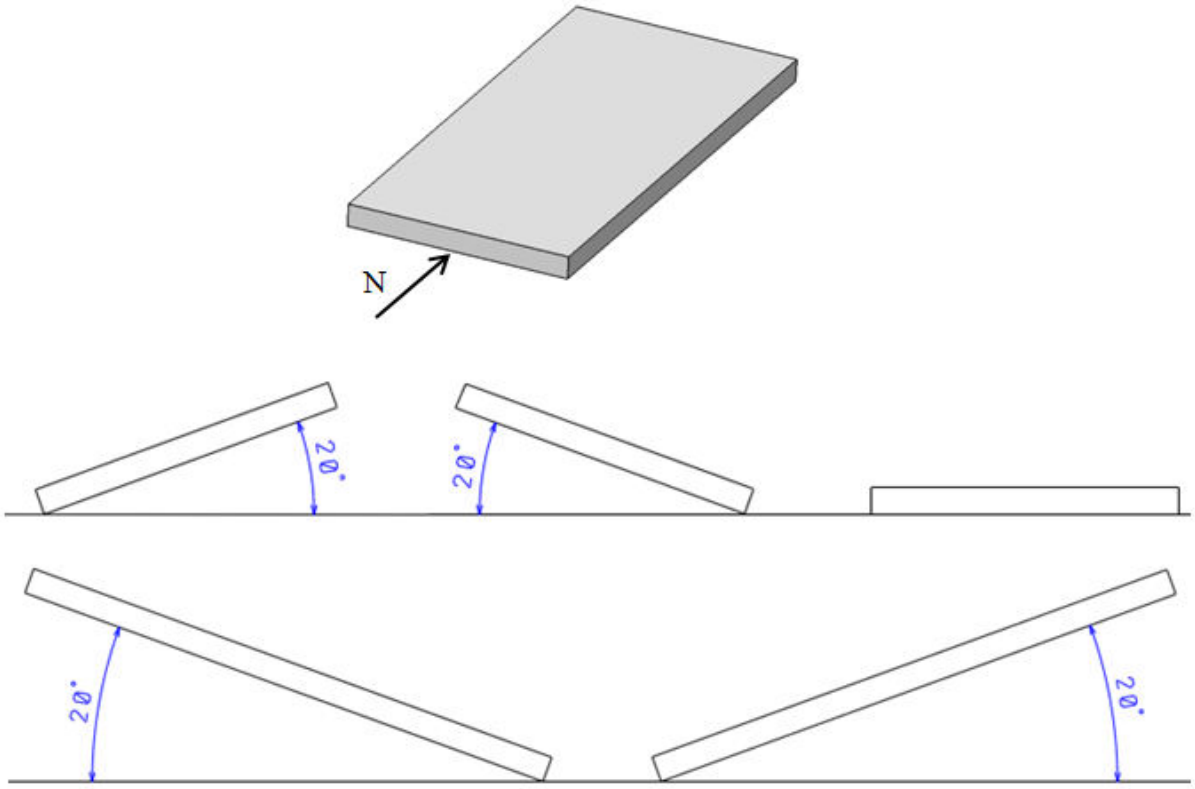
U prvom dijelu ovoga poglavlja će biti prikazan proces razvoja klasifikacijskog algoritma za prepoznavanje radnji pisanja i brisanja pomoću IMU-a te rezultati koje daje takav klasifikacijski algoritam. U drugom dijelu poglavlja je opisana nadogradnja algoritma za prepoznavanje brisanja upotrebom podataka koje daje optički sustav Polaris te je napravljena usporedba rezultata dvaju klasifikacijskih algoritama. Prvo je razvijen klasifikacijski algoritam pomoću IMU-a, jer IMU omogućuje jednostavnu upotrebu (nosivi sklop se samo učvrsti na ruku) i nema ograničenja optičkog sustava kao što su ograničeni radni prostor, potreba za dodatnim alatima (markerima) i utjecaj okolišnih uvjeta na rad (npr. utjecaj osvjetljenja). Nadalje IMU omogućuje direktno očitanje ubrzanja potrebnih za izvođenje pokreta nalik ljudskima, dok je optički sustav upotrijebljen kao nadogradnja klasifikacijskog algoritma za učenje robota specifičnom zadatku brisanja ploče.

5.1. Učenje i prepoznavanje pokreta pomoću IMU-a

Razvoj algoritma za prepoznavanje pokreta pomoću IMU-a zahtijevao je prikupljanje podataka za učenje promatranih radnji, zatim odabir klasifikacijskih parametara i razvoj algoritma za klasifikaciju na temelju podataka za učenje. Navedeni koraci su opisani u nastavku, a na kraju je provedeno i ispitivanje razvijenog algoritma te su prikazani rezultati ispitivanja.

5.1.1. Prikupljanje podataka za analizu

Kako bi se obrasci kretanja ljudske ruke prilikom određenih radnji mogli prepoznati potrebno je najprije provesti niz mjerenja koja će poslužiti kao primjeri za učenje iz kojih će se odrediti specifični parametri za prepoznavanje pojedinih radnji. Radnje koje se žele prepoznati pomoću klasifikacijskih parametara su pisanje po ploči i brisanje ploče. Za svaku od navedenih radnji provedeno je 25 mjerenja koja služe kao setovi za određivanje parametara. Mjerenja su raspoređena na način da je provedeno po 5 mjerenja za 5 različitih orijentacija ploče u prostoru. Mjerenja su provedena pri različitim nagibima ploče kako je prikazano na slici 33. Gornji dio slike prikazuje ploču u izometriji i smjer pogleda nacrta, srednji dio slike prikazuje tri primijenjene orijentacije ploče u nacrtu, a donji dio slike prikazuje posljednje dvije primijenjene orijentacije ploče koje su prikazane u bokocrtu.



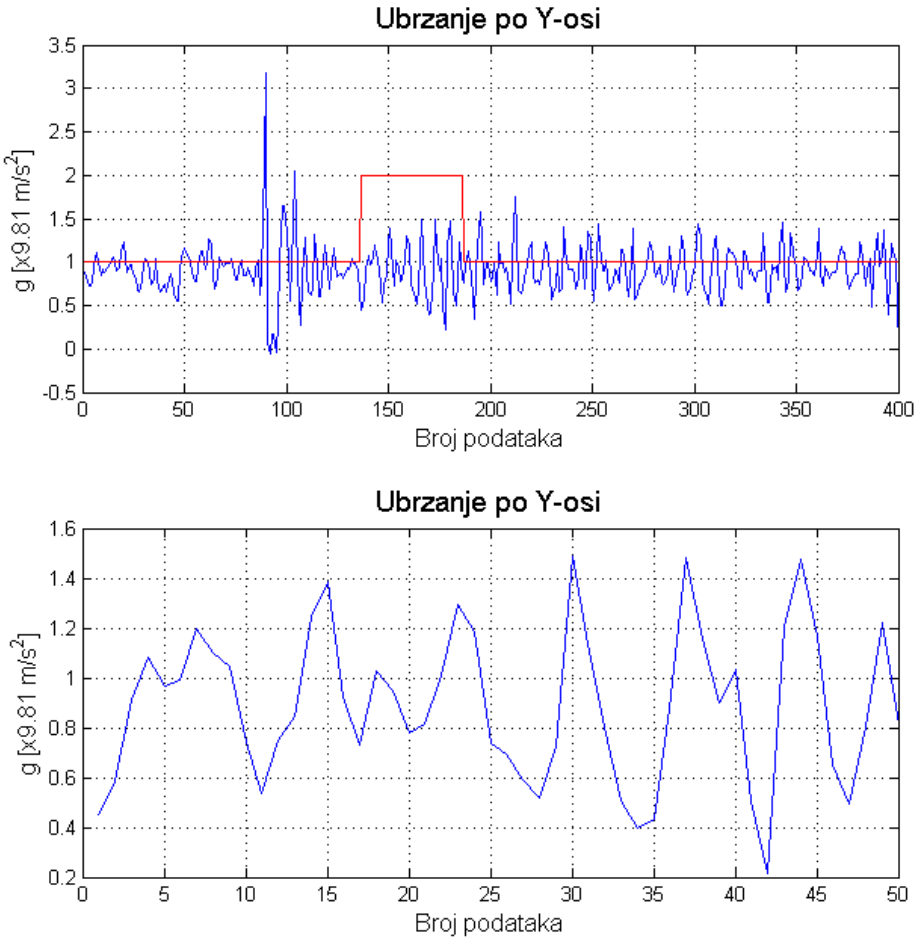
Slika 33. Primijenjene orijentacije ploče prilikom mjerenja

5.1.2. Definiranje skupa za učenje

Nakon što su napravljena mjerenja za obje radnje u različitim orijentacijama bilo je potrebno izdvojiti korisne podatke za klasifikaciju pojedine radnje odnosno uzeti samo one podatke koji su dobiveni za vrijeme obavljanja radnje. Mjereni podaci sastoje se od linearnih ubrzanja po osima IMU-a, kutnih brzina oko osi IMU-a, Eulerovih kuteva i rezultantnog ubrzanja IMU-a. Podaci izdvojeni iz početnih mjerenja koji služe za klasifikaciju nazivaju se skupom za učenje. Skup za učenje je definiran kao podaci koji su prikupljeni u prvoj sekundi obavljanja pojedine radnje (prvih 50 podataka pri određenoj radnji budući da je frekvencija rada IMU-a 50 Hz). Interval od jedne sekunde je odabran kako bi se moglo prepoznati kratkotrajno i dugotrajno izvođenje promatranih radnji. Kraćim intervalom se ne bi pouzdano mogla odrediti radnja iz statističke obrade mjerenih veličina, dok bi duži interval onemogućio prepoznavanje kratkotrajnog izvođenja radnji čije bi se karakteristike izgubile zbog količine promatranih podataka.

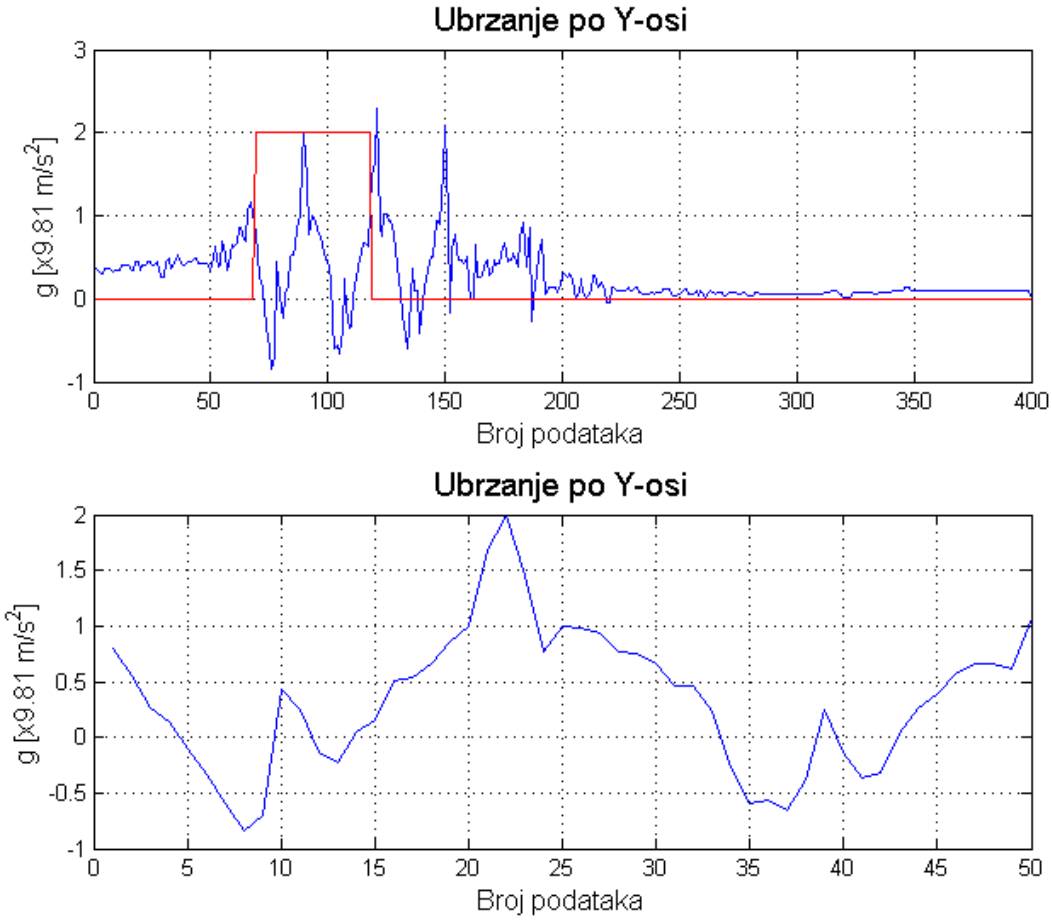
Na slici 34 prikazan je karakterističan odziv ubrzanja po Y-osi IMU-a, koji sadrži i komponentu gravitacijskog ubrzanja, za cijelo vrijeme mjerenja tijekom kojeg je obavljena radnja pisanja te izdvojeni dio koji je odabran kao jedan od primjera u skupu za učenje. Na

gornjem grafu je crveno označeno područje podataka koje je uzeto kao primjer za skup za učenje, a na donjem grafu su prikazani samo podaci koji su dodijeljeni skupu za učenje.



Slika 34. Ubrzanje po Y-osi IMU-a tijekom pisanja

Na slici 35 prikazan je karakterističan odziv ubrzanja po Y-osi IMU-a za cijelo vrijeme mjerenja tijekom kojeg je obavljena radnja brisanja te izdvojeni dio koji je odabran kao jedan od primjera u skupu za učenje. Na gornjem grafu je crveno označeno područje podataka koje je uzeto kao primjer za skup za učenje, a na donjem grafu su prikazani samo podaci koji su dodijeljeni skupu za učenje.



Slika 35. Ubrzanje po Y-osi IMU-a tijekom brisanja

5.1.3. Odabir parametara za klasifikaciju

Nakon što su definirani setovi za učenje za obje promatrane radnje napravljena je statistička obrada setova za učenje. Statistička obrada setova za učenje uključuje izračun srednjih vrijednosti, RMS (*root mean square*) vrijednosti, maksimalnih apsolutnih vrijednosti, raspona varijacije, standardne devijacije i interkvartilnog raspona za sve mjerene veličine i sva provedena mjerenja.

Srednja vrijednost se računa prema jednadžbi:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} . \tag{21}$$

RMS vrijednost se računa prema jednadžbi:

$$x_{RMS} = \sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \ldots + x_n^2\right)} . \tag{22}$$

Maksimalna apsolutna vrijednost se računa prema jednadžbi:

$$x_{maxabs} = |x_{max}| . \quad (23)$$

Raspon varijacije se računa prema jednadžbi:

$$R = x_{max} - x_{min} . \quad (24)$$

Standardna devijacija se računa prema jednadžbi:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}} . \quad (25)$$

Interkvartilni raspon ili raspon varijacije središnjih pedeset posto podataka se računa prema jednadžbi:

$$I_Q = Q_3 - Q_1 . \quad (26)$$

Nakon što su izračunati svi statistički parametri napravljena je njihova analiza kako bi se utvrdilo da li je moguće klasificirati promatrane radnje na temelju različitih iznosa statističkih parametara. Navedena analiza je napravljena u MATLAB-u pomoću programa *Analiza1.m* i *Analiza2.m*. *Analiza1.m* služi za provjeru preklapanja iznosa pojedinih parametara određene radnje s iznosima parametara druge radnje, dok *Analiza2.m* izračunava koji od parametara koji se ne preklapaju s parametrima druge radnje najviše odstupa od parametara druge radnje računanjem omjera dviju najbližih vrijednosti parametara radnji. *Analiza2.m* ispisuje izračunate omjere u Excel tablicu iz koje se onda odabiru najbolji parametri za klasifikaciju. Parametar je to bolji za klasifikaciju što mu je dodijeljen veći iznos omjera susjednih vrijednosti parametara, a parametrima prema kojima nije moguće napraviti klasifikaciju su dodijeljene nule. Tablica za odabir parametara za klasifikaciju dana je na slici 36. U prikazanoj tablici se vidi da je najbolji parametar za razlikovanje pisanja od brisanja RMS iznos kutne brzine oko Z-osi IMU-a jer on ima najveći dodijeljeni iznos.

Brisanje vs pisanje	Srednja vrijednost	IQR	Standard deviation	RMS	Max (abs)	Raspon
axg [g]	0	0	0	0	0	0
ayg [g]	1,183331365	0	0	1,00439084	0	0
azg [g]	0	0	0	0	0	0
ωx [°/s]	0	0	0	0	0	0
ωy [°/s]	0	0	0	0	0	0
ωz [°/s]	0	0	1,112941799	1,27016413	0	0
Skretanje [°]	0	0	0	0	0	0
Poniranje [°]	0	0	0	0	0	0
Valjanje [°]	0	0	0	0	0	0
aR [g]	0	0	0	0	0	0

Slika 36. Tablica najboljih klasifikacijskih parametra

Na temelju podataka iz generirane tablice korišteni su sljedeći parametri za klasifikaciju:

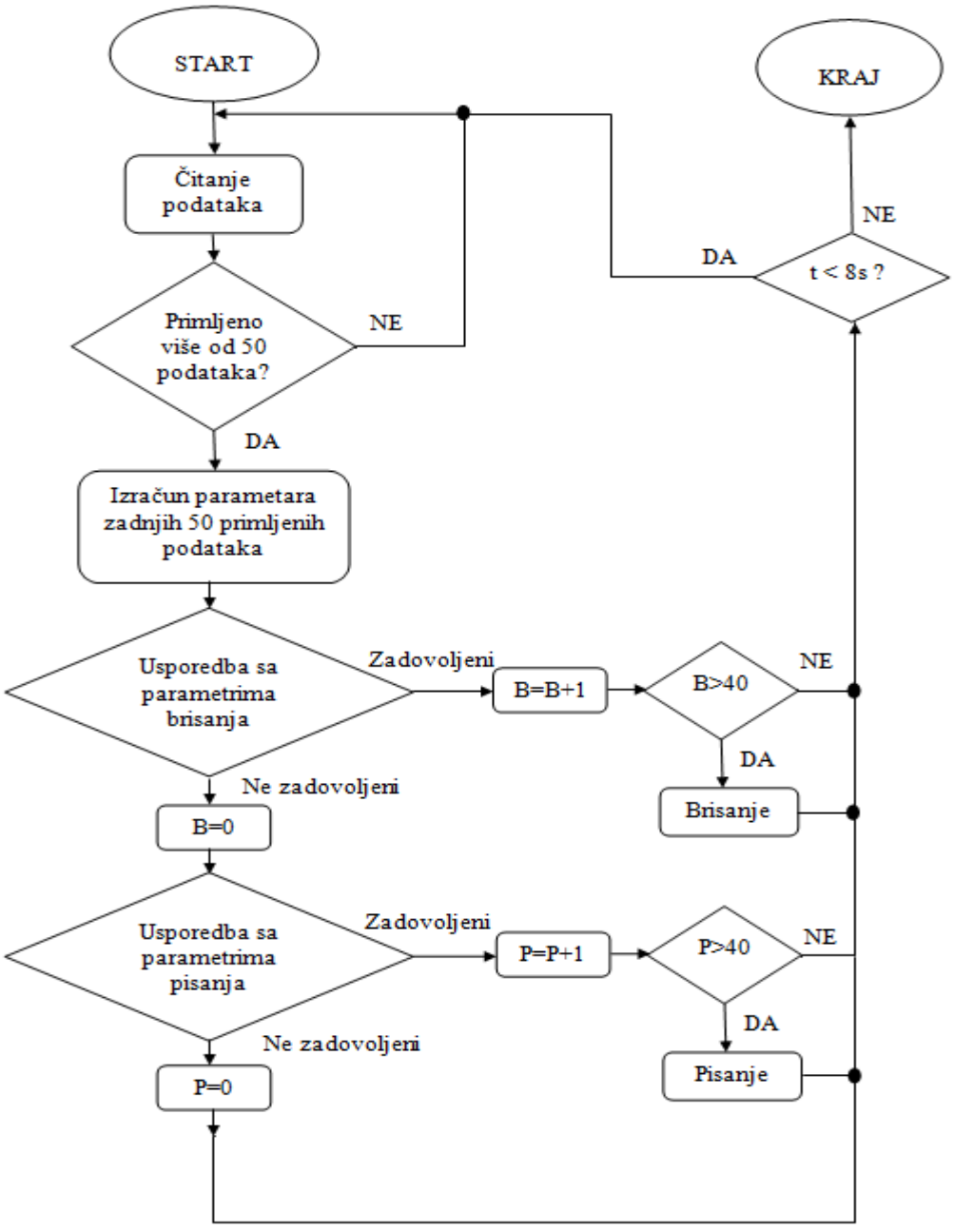
- pisanja:
    - srednja vrijednost ubrzanja po Y-osi,
    - standardna devijacija kutne brzine oko Z-osi,
    - RMS iznos kutne brzine oko Z-osi,
- brisanja:
    - srednja vrijednost ubrzanja po Y-osi,
    - standardna devijacija kutne brzine oko Z-osi,
    - RMS iznos kutne brzine oko Z-osi,
    - srednja vrijednost ubrzanja po X-osi,
    - raspon varijacije ubrzanja po Y-osi.

Zadnja dva parametra za klasifikaciju brisanja se ne pojavljuju u tablicama kao pogodni za klasifikaciju tj. razlikovanje od radnje pisanja, ali se oni koriste kako bi se osigurala veća točnost otkrivanja radnje brisanja (kako bi se što bolje odredio kraj i početak radnje) te se smanjila mogućnost pogrešnog prepoznavanja pokreta brisanja uslijed nekog drugog sličnog pokreta. Kao dodatni parametri su odabrane vrijednosti povezane s ubrzanjima po X i Y-osi IMU-a jer su zbog položaja IMU-a na ruci, najviše se mijenja ubrzanje po Y-osi dok je ubrzanje po X-osi skoro nula, one karakteristične za pokret brisanja ploče.

5.1.4. Algoritam za klasifikaciju

Odabrani parametri za klasifikaciju su implementirani u algoritam za klasifikaciju koji je napravljen u MATLAB-u. Algoritam za klasifikaciju se naziva *Online_provjera_pokreta.m*.

Algoritam radi na način da cijelo vrijeme za posljednjih 50 primljenih podataka računa iznose odabranih parametara za klasifikaciju te ih uspoređuje s vrijednostima parametara seta za učenje. Kako bi se spriječila klasifikacija nekih sličnih pokreta u jednu od klasa naučenih pokreta u algoritmu je dodatno napravljen uvjet koji zahtijeva da primljeni podaci 40 puta zaredom zadovolje klasifikacijske parametre kako bi bili svrstani u jednu od naučenih radnji. Osim klasifikacije algoritam također sprema primljene podatke kako bi se oni mogli iskoristiti za kasniju analizu. Na slici 37 prikazan je dijagram toka algoritma za klasifikaciju.



Slika 37. Dijagram toka algoritma za klasifikaciju

5.1.5. Rezultati prepoznavanja obrazaca kretanja

Ispitivanje rada algoritma za klasifikaciju provedeno je izvođenjem 20 mjerenja za svaku od naučenih radnji. Radnje koje je trebalo prepoznati izvođene su u nasumičnim položajima u prostoru s istim predmetima rada kao i kod prikupljanja podataka seta za učenje.

Postotci prepoznavanja naučenih radnji su izračunati kao (broj prepoznavanja/ukupni broj mjerenja) x100 te su oba slučaja vrlo visoki i iznose:

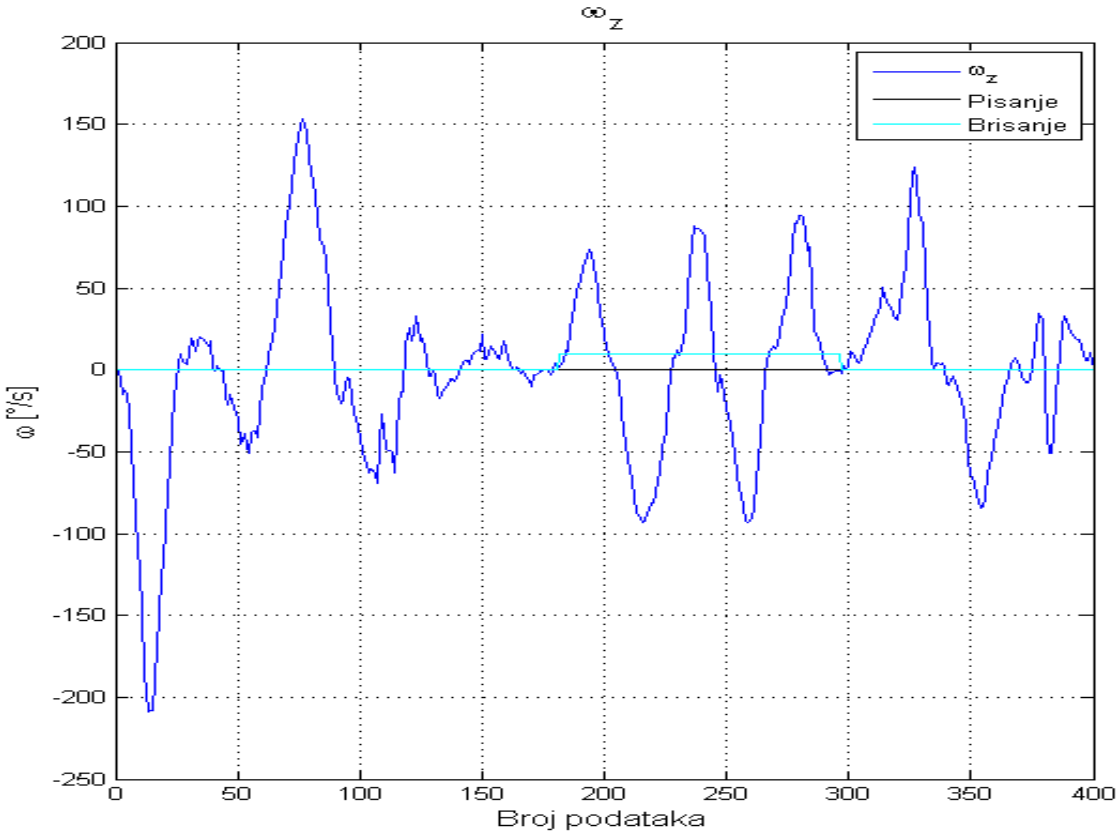
- 80 % za brisanje i
- 85 % za pisanje.

Postotci lažnih prepoznavanja koji govore o tome da li je prepoznata jedino ispravna radnja ili je uz ispravnu radnju pogrešno prepoznata i neka druga radnja kao jedna od promatranih (npr. nasumično mahanje rukom po prostoru) ili je pak pogrešno prepoznata samo neka druga radnja kao jedna od promatranih su izračunati kao (broj pogrešnih prepoznavanja/ukupni broj mjerenja) x100 te iznose:

- 40 % za brisanje i
- 0 % za pisanje.

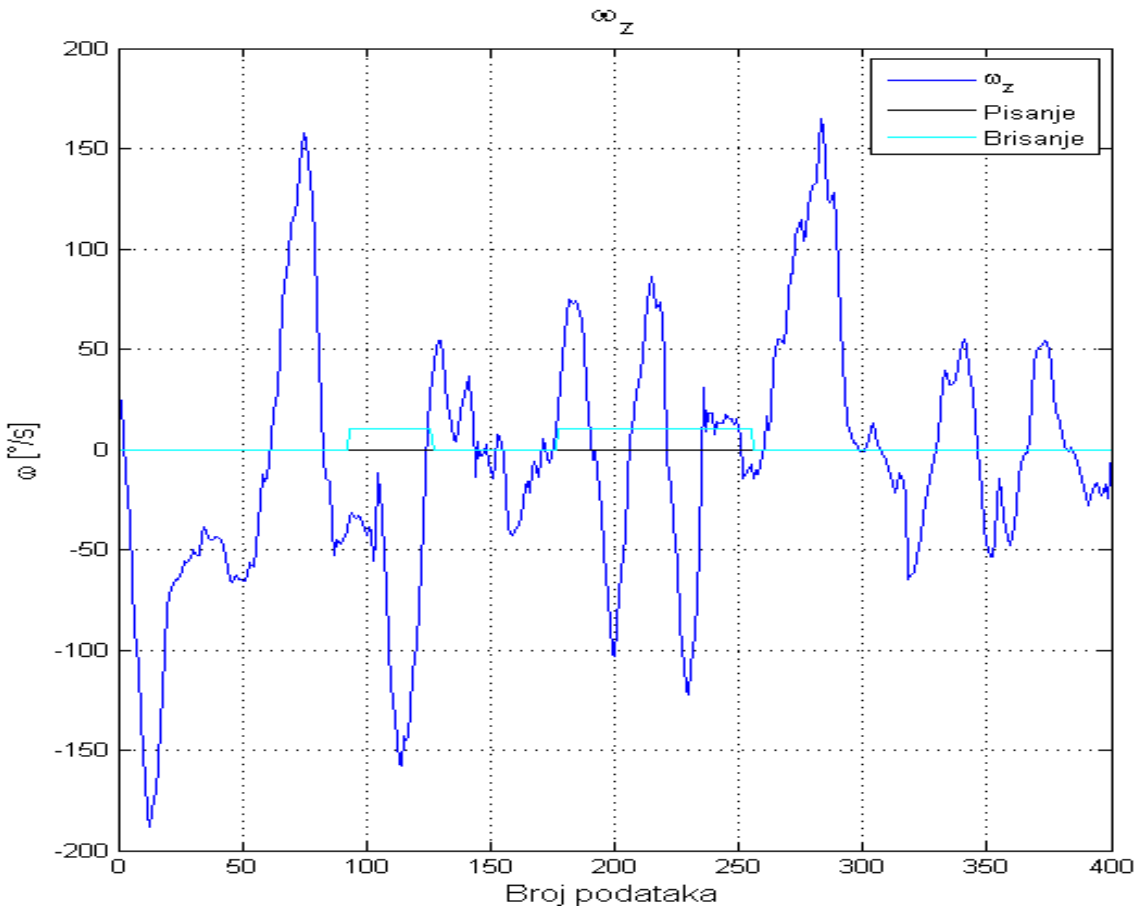
Iz navedenih postotaka lažnih prepoznavanja se vidi da kod pisanja uopće nema lažnih prepoznavanja, dok kod brisanja postoji problem lažnih prepoznavanja jer se pokreti prilikom uobičajenog kretanja prostorom prepoznaju kao pokreti brisanja, a to se događa jer korištene statističke vrijednosti promatranih parametara ne isključuju mogućnost da se takve vrijednosti pojave prilikom nekih drugih radnji (pokreta) već se u ovome slučaju u potpunosti isključuju mogućnosti otkrivanja pisanja kao radnje brisanja za prikupljeni set podataka za učenje u promatranim položajima u prostoru i obrnuto.

Na slici 38 je prikazan karakteristični odziv tijekom provjere prepoznavanja brisanja algoritmom za klasifikaciju, te uspješno prepoznavanje radnje brisanja. Plava linija na grafu pokazuje raspon podataka za koje je detektirano brisanje. Na grafu je prikazana kutna brzina oko Z-osi prilikom provedenog mjerenja.



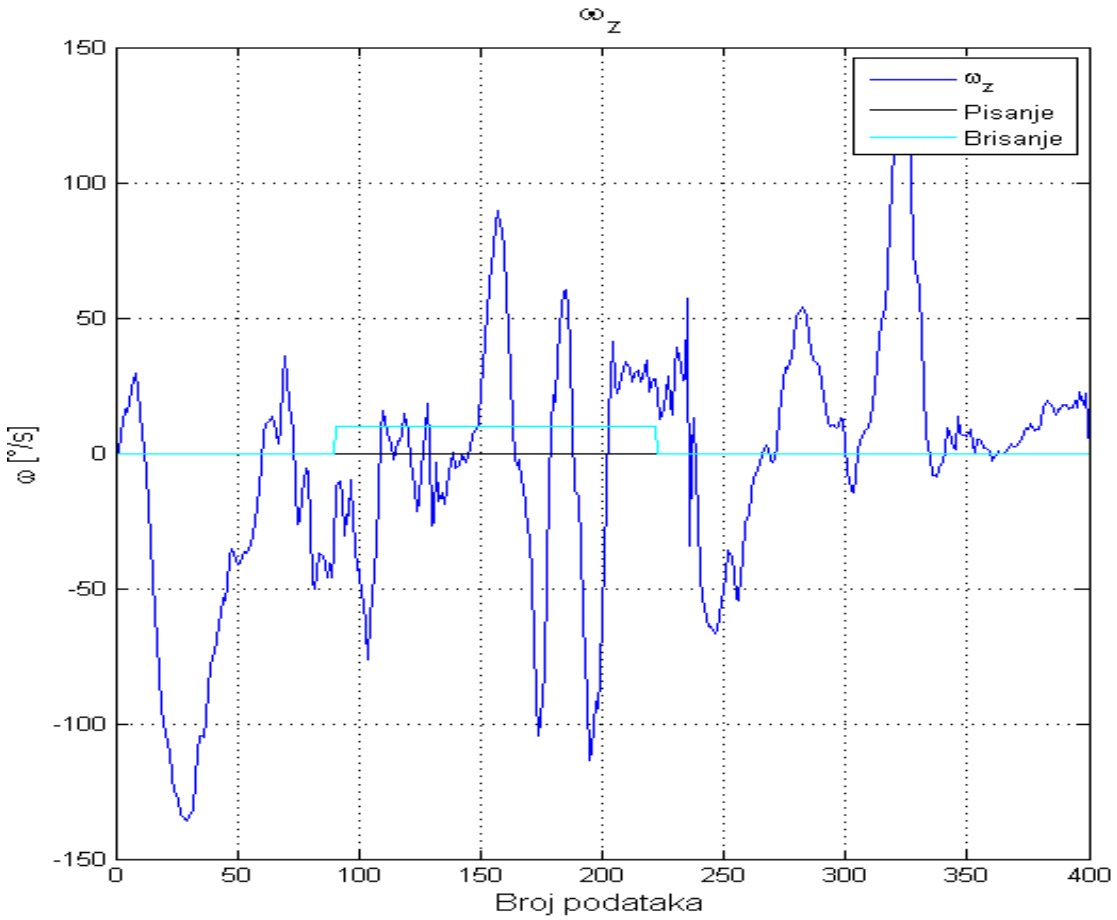
Slika 38. Karakteristični graf kutne brzine oko Z-osi prilikom brisanja

Na slici 39 je prikazan odziv tijekom provjere prepoznavanja brisanja algoritmom za klasifikaciju, graf na ovoj slici prikazuje slučaj kada se osim ispravnog prepoznavanja radnje javlja i pogrešno prepoznavanje radnje uslijed nasumičnog kretanja prostorom. Plavom linijom na grafu su prikazani skupovi podataka za koje je prepoznata radnja brisanja, lijevi skup podataka predstavlja lažno prepoznavanje, a desni skup ispravno prepoznavanje brisanja. Do pogrešnog prepoznavanja dolazi uslijed upada korištenih statističkih parametara unutar graničnih vrijednosti kojima je definirana radnja brisanja, a sve iz razloga ograničenosti seta za učenje i upotrebe jednostavnog algoritma. Na grafu je prikazana kutna brzina oko Z-osi prilikom provedenog mjerenja.



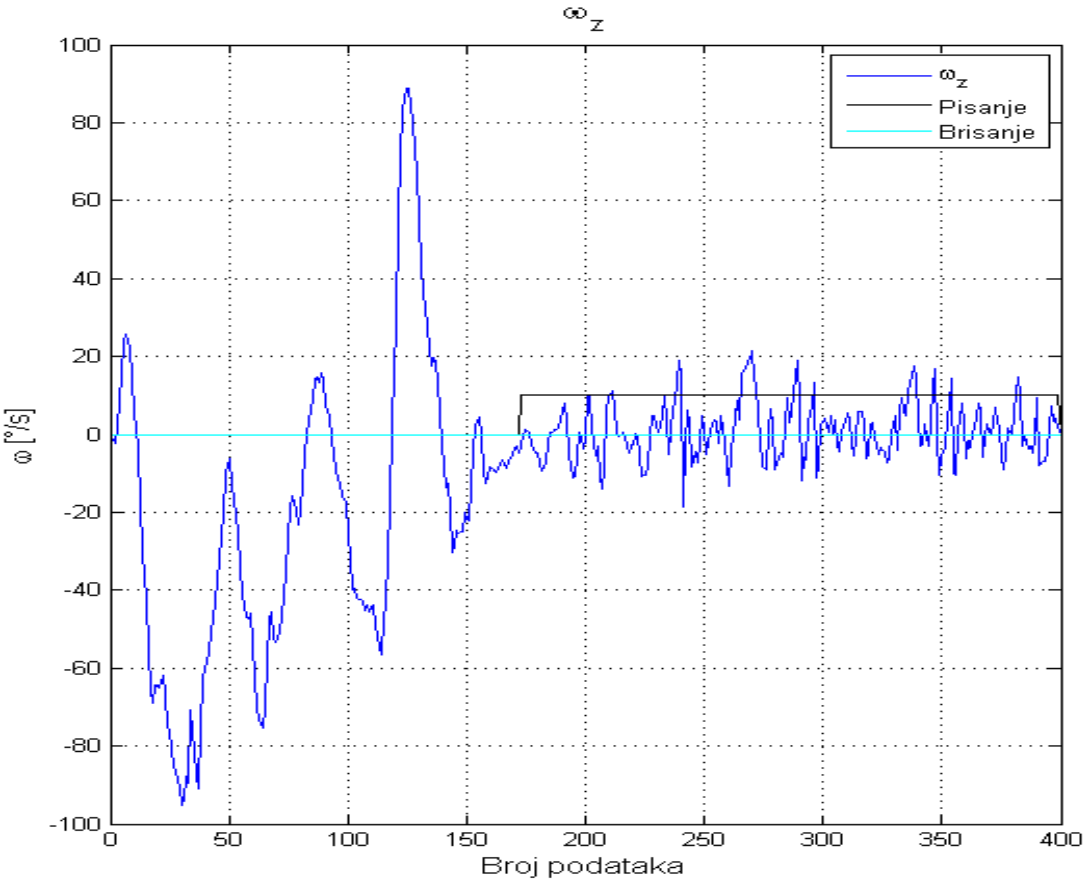
Slika 39. Lažno (lijevo) i ispravno (desno) prepoznavanje brisanja

Osim problema lažnog prepoznavanja te neprepoznavanja radnji javlja se i problem točnog određivanja početka i kraja radnje brisanja što je ključno ukoliko se robota želi naučiti brisanju uz ponavljanje ljudskih pokreta. Na slici 40 prikazan je odziv tijekom kojeg je prepoznata radnja brisanja, ali njezin početak i kraj nisu ispravno definirani. Skup podataka za koje je prepoznata radnja brisanja ponovno je označen plavom linijom na grafu, a graf prikazuje kutnu brzinu oko Z-osi prilikom provedenog mjerenja. Identičan problem javlja se i prilikom prepoznavanja radnje brisanja, a on je posljedica upotrebe graničnih vrijednosti statističkih parametara, kada bi se granice proširile višak pokreta bi se u velikom broju slučajeva detektirao kao tražena radnja, dok bi sužavanjem granica postotak prepoznavanja traženih radnji bio znatno manji. Navedeni problem proizlazi iz toga što ljudi nikada ne obavljaju istu radnja dva puta doslovno identičnim pokretima pa se zbog toga koriste statističke veličine izračunate na temelju više mjerenja koje onda onemogućuju točnu detekciju kraja i početka obavljanja određene radnje.



Slika 40. Netočno određen početak i kraj brisanja

Na slici 41 je prikazan karakteristični odziv tijekom provjere prepoznavanja pisanja algoritmom za klasifikaciju, te uspješno prepoznavanje radnje pisanja. Crna linija na grafu pokazuje raspon podataka za koje je detektirano pisanje. Na grafu je prikazana kutna brzina oko Z-osi prilikom provedenog mjerenja. Prilikom radnje pisanja također se javljaju problemi s određivanjem točnog početka i kraja pisanja kako je prije spomenuto. Učenje robota pisanju bi predstavljalo znatno složeniji problem (potreba za definiranjem svakog slova ili linije posebno, potreba za popustljivim markerom za pisanje (s oprugom) ili regulacija sile kojom vrh alata pritišće ploču dok je to kod brisanja riješeno svojstvima spužve) pa je u ovome radu naglasak na mogućnosti učenja robota brisanju na temelju prepoznavanja radnje brisanja.



Slika 41. Karakteristični graf kutne brzine oko Z-osi prilikom pisanja

5.2. Učenje i prepoznavanje pokreta višemodalnom interakcijom

Problemi točnog određivanja početka i kraja tražene radnje kao i problem prepoznavanja pogrešnih radnji kao željenih onemogućili bi ispravno učenje robota željenim radnjama te bi u nekim slučajevima mogli dovesti i do oštećenja robota ili okoline uslijed neželjenih pokreta robota. Navedeni problemi za potrebe učenja robota brisanju riješeni su upotrebom optičkog sustava Polaris koji je ujedno služio i za određivanje putanje kretanja tijekom radnje brisanja. Algoritam za prepoznavanje radnje pisanja nije unaprijeđen upotrebom optičkog sustava pošto je zaključeno da bi učenje robota pisanju predstavljalo složeni problem kako je spomenuto u 5.1.5..

5.2.1. Odabir dodatnog parametra za klasifikaciju

Optički sustav Polaris kao izlaz daje poziciju i orijentaciju definiranih alata s markerima unutar svog mjernog volumena pa je u svrhu učenja putanje kojom robot treba proći prilikom brisanja ploča smještena unutar radnog volumena optičkog sustava te je na spužvu montiran alat M2 s markerima za određivanje njezine putanje. Problem otkrivanja pogrešnih radnji kao

radnje brisanja je zato riješen dodavanjem alata M3 s markerima na ploču te je određen odnosu između alata M2 i alata M3 koji garantira da je spužva u dodiru s pločom. Odnos alata M2 i M3 koji garantira dodir spužve s pločom je definiran kao udaljenost njihovih ishodišta po Z-osi lokalnog koordinatnog sustava alata M3 u rasponu od 24 do 32 mm. Raspon udaljenosti je određen pomoću razvijenog programa u C++ programskom jeziku i dio je C++ upravljačkog programa te se njegov kod nalazi u prilogu, a pokretanjem C++ programa pristupa mu se unošenjem broja *5*. Udaljenost ishodišta alata M2 i M3 se izračunava upotrebom jednadžbe 9, te očitavanjem translacije po Z-osi iz matrice homogene transformacije T_{M2inM3} kako je prikazano u nastavku:

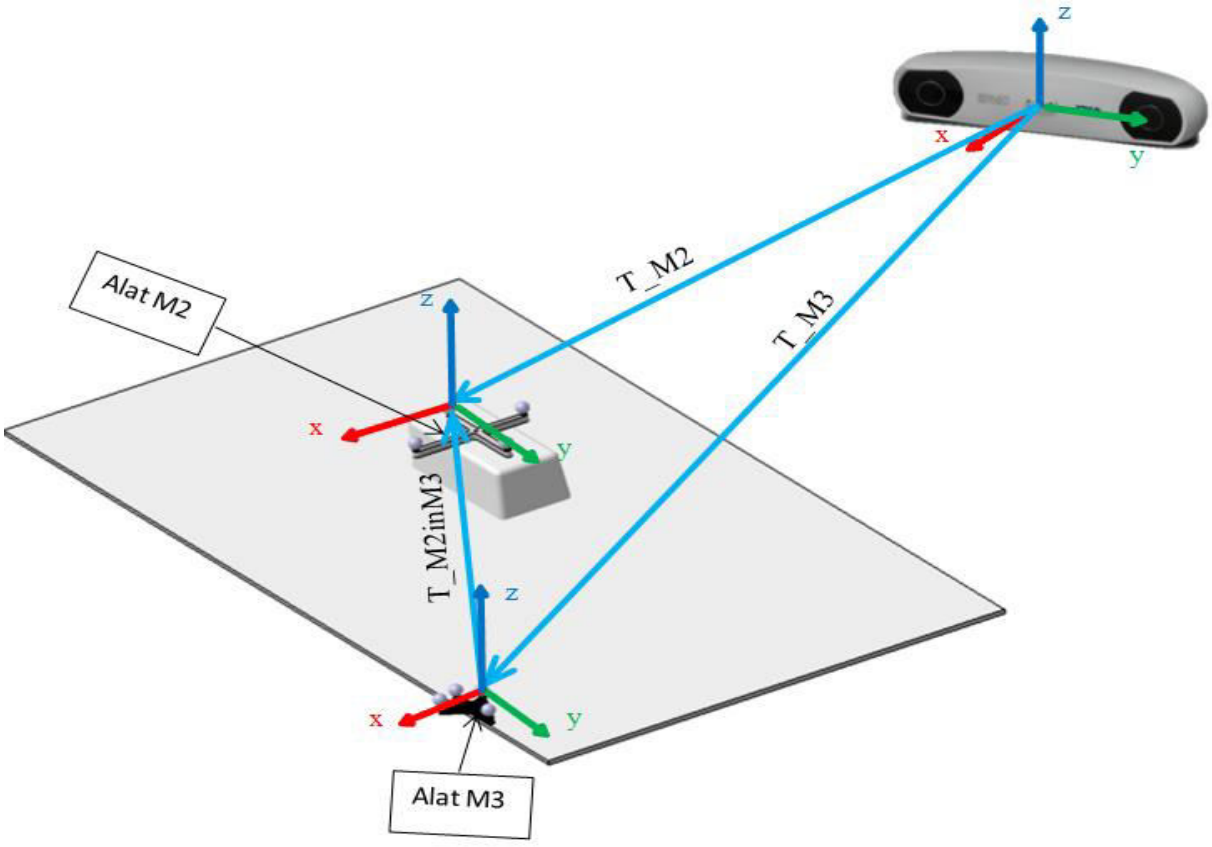
$$T_{M2inM3}=T_{M3}^{-1}\cdot T_{M2}\ , \tag{27}$$

gdje je:

T_{M2inM3} - pozicija i orijentacija ishodišta alata M2 u koordinatnom sustavu alata M3,

T_{M2} - pozicija i orijentacija ishodišta alata M2 u baznom koordinatnom sustavu Polarisa,

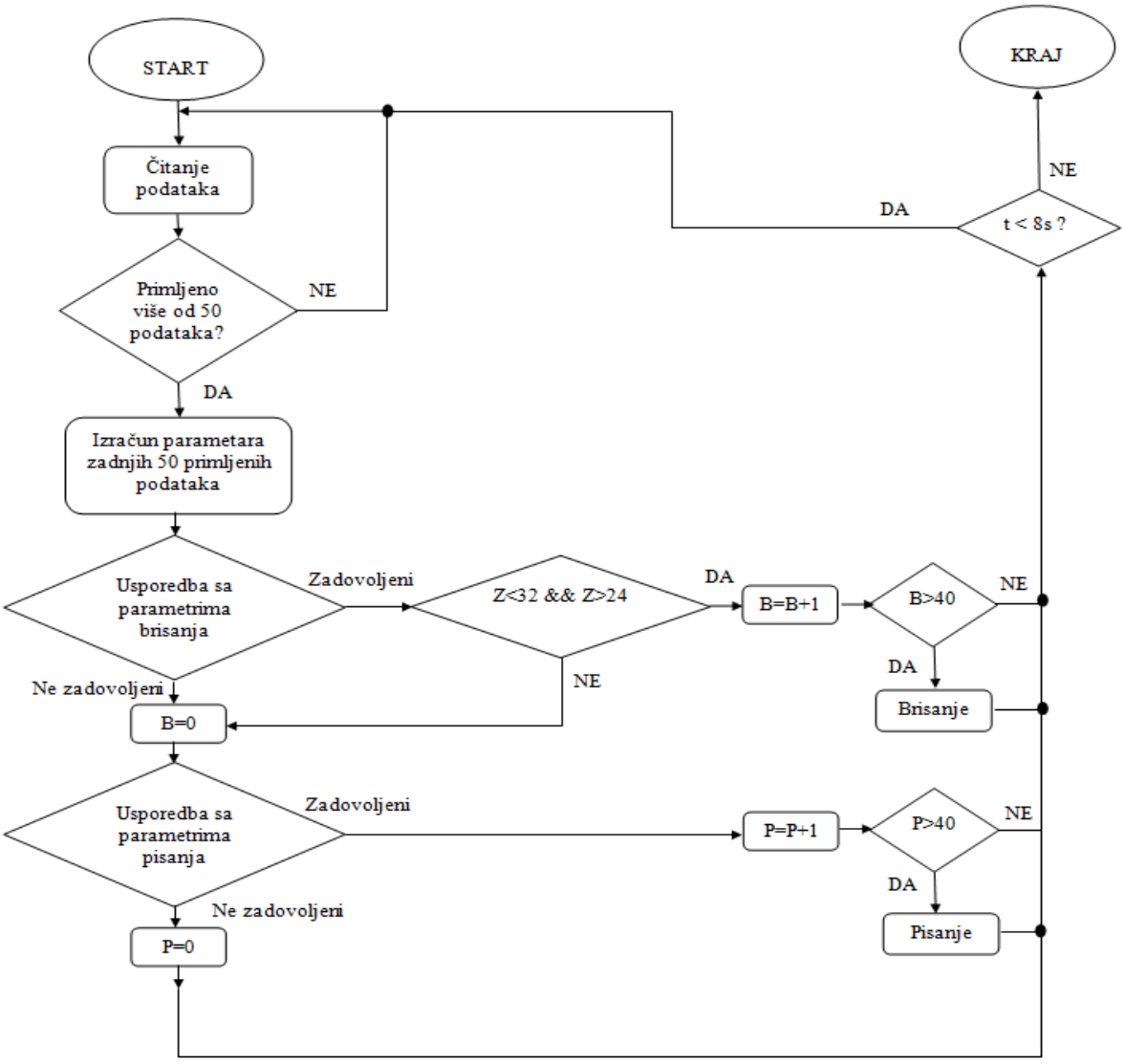
T_{M3} - pozicija i orijentacija ishodišta alata M3 u baznom koordinatnom sustavu Polarisa.



Slika 42. Matrice transformacije za određivanje odnosa alata M2 i M3

5.2.2. Algoritam za klasifikaciju

Algoritmu za klasifikaciju iz 5.1.4. dodan je uvjet da je udaljenost ishodišta alata M2 i M3 u rasponu između 24 i 32 mm te je kod navedenog algoritma prikazan u prilogu pod imenom *Provjera_pokreta_fusion.m*. Pošto ovaj algoritam zahtijeva i primanje podataka koji se dobivaju na temelju očitanja iz optičkog sustava Polaris njegov rad se usklađuje s radom C++ upravljačkog programa upisivanjem broja *3* nakon pokretanja C++ programa. C++ program se prvi pokreće, a zatim se pokreće MATLAB skripta *Provjera_fusion.m* unutar koje se poziva i izvodi algoritam za klasifikaciju *Provjera_pokreta_fusion.m*. Opis rada C++ dijela programa i usklađivanje podataka opisano je u 5.2.4.. Dijagram toka novog algoritma prikazan je na slici 43.



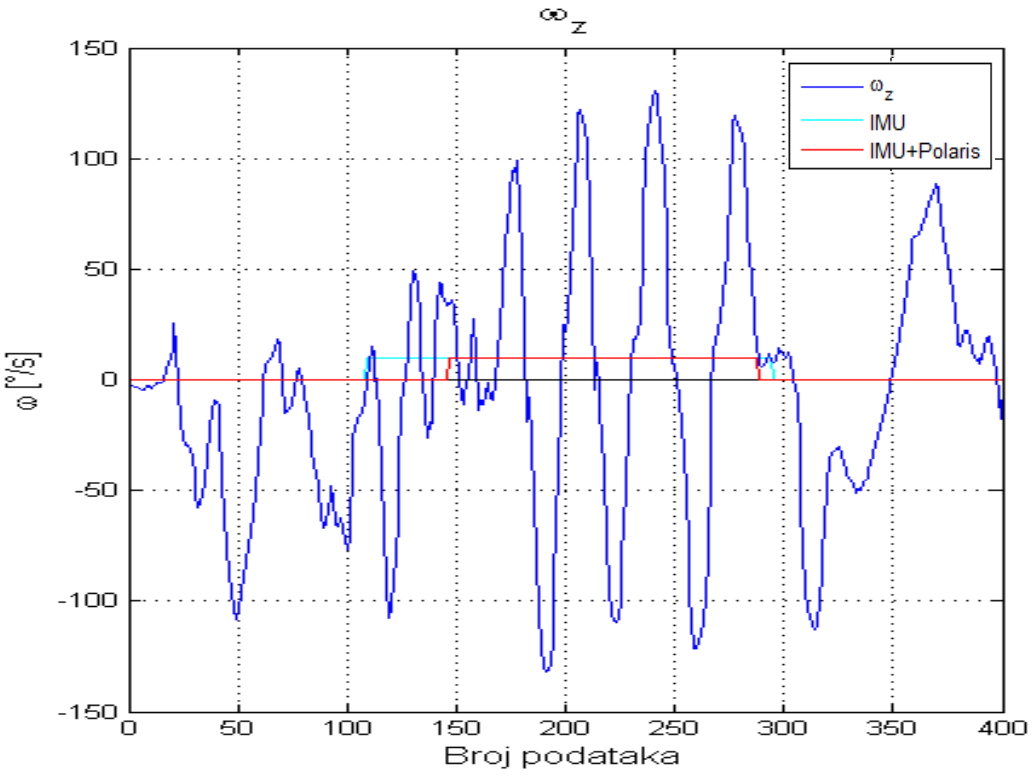
Slika 43. Dijagram toka novog algoritma za klasifikaciju

5.2.3. Rezultati prepoznavanja radnje brisanja

Ispitivanje rada novog algoritma za klasifikaciju provedeno je izvođenjem 20 mjerenja za radnju brisanja. Radnja je izvođena u nasumičnim položajima u prostoru s istim predmetima rada kao i kod testiranja rada klasifikacije samo na temelju podataka iz IMU-a.

Postotak prepoznavanja radnje brisanja je izračunat kao (broj prepoznavanja/ukupni broj mjerenja) x100 te iznosi 80 % što je jednako kao i kod prepoznavanja radnje samo sa IMU-om, dok su problematična lažna prepoznavanja radnje brisanja u potpunosti uklonjena. Pošto su lažna prepoznavanja uklonjena jasna je prednost upotrebe dodatnih podataka iz optičkog sustava.

Na slici 44 prikazan je karakteristični odziv tijekom provjere prepoznavanja brisanja novim algoritmom za klasifikaciju, te uspješno prepoznavanje radnje brisanja. Plava linija na grafu prikazuje skup podataka za koje je detektirana radnja brisanja algoritmom za klasifikaciju koji koristi samo podatke iz IMU-a, a crvena linija prikazuje skup podataka za koje je detektirana radnja brisanja algoritmom za klasifikaciju koji upotrebljava podatke iz IMU-a i optičkog sustava. Problem točnog određivanja početka i kraja radnje brisanja nije riješen prikazanim algoritmom u 5.2.2., već će njegovo rješenje biti opisano u 5.2.4..



Slika 44. Usporedba prepoznavanja razvijenim klasifikacijskim algoritmima

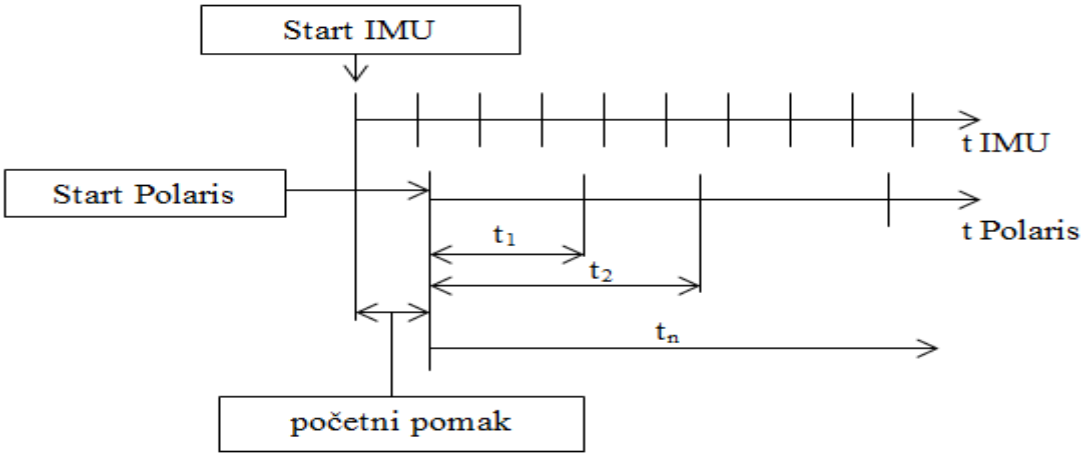
5.2.4. Učenje novih pokreta

Nakon što je riješen problem lažnih prepoznavanja radnje brisanja implementirano je učenje robota radnji brisanja na način da robot ponovi izvedenu i prepoznatu radnju brisanja oponašajući putanju i ubrzanja ljudskih pokreta. Za ispravno izvođenje radnje brisanja ključno je točno odrediti početak i kraj radnje brisanja kao i ispravno uskladiti podatke dobivene iz IMU-a s onima dobivenim iz Polarisa kako bi se oni poklapali u vremenu.

Učenje robota pokretima koje treba ponoviti provodi se pomoću MATLAB skripte *Snimanje_fusion.m* i dijela C++ upravljačkog programa pod brojem *3* nazvanog *Ucenje novih pokreta*. Učenje pokreta započinje upisivanjem broja *3* unutar prozora C++ upravljačkog programa čime se pokreće dio programa nazvan *Ucenje novih pokreta*, on tada ulazi u petlju te čeka znak iz MATLAB-a da započne sa snimanjem putanje. Nakon pokretanja skripte *Snimanje_fusion.m* programi se povežu i međusobno komuniciraju putem TCP protokola, kada su povezani započinje snimanje podataka iz IMU-a unutar skripte *Snimanje_fusion.m* te se ujedno šalje znak za start snimanja putanje C++ programu. Nakon 8 sekundi zaustavlja se snimanje podataka iz IMU-a te se potom zaustavlja i snimanje točaka putanje. Nakon snimanja obavlja se dodatna obrada podataka kako bi se odredila točna putanja i ubrzanja koje robot treba upotrijebiti. Interval od 8 sekundi se pokazao kao dovoljan za učenje pokreta brisanja.

Problem usklađivanja podataka u vremenu nastaje zbog toga što se prikupljanje podataka pokreće iz dva različita programa s vremenskim razmakom te zbog različite frekvencije prikupljanja podataka (50 Hz IMU i ≈20 Hz Polaris). Podaci o putanji iz Polarisa se ne prikupljaju u fiksnom vremenskom intervalu već su različita vremena između primanja dvije točke putanje te je zbog toga dodatno otežano usklađivanje podataka u vremenu. Usklađivanje podataka u vremenu je riješeno na način da se prilikom početka prikupljanja podataka u svakom od programa zabilježi globalno vrijeme računala s preciznošću od milisekunde te se na taj način odredi početni pomak u vremenu. Vrijeme između primanja podataka iz IMU-a je fiksno i ono iznosi 20 ms dok se ukupno proteklo vrijeme od početka pokretanja C++ programa za spremanje točaka putanje mjeri korištenjem funkcije *GetTickCount()* i sprema uz svaku točku putanje, na taj način je omogućeno naknadno spajanje odgovarajućih točaka putanje s podacima iz IMU-a što je bitno kako bi se robot po putanji kretao s odgovarajućim ubrzanjima. Vremenski pomak i vremena spremanja podataka su ilustrativno prikazani na slici 45. Naknadno spajanje podataka provodi se dodavanjem početnog vremenskog pomaka

svakoj od točaka putanje te se na taj način vrijeme primanja točke putanje poklapa s vremenom primanja podataka iz IMU-a. Vrijeme primanja točke putanje često nije jednako vremenu u kojem su primljeni podaci iz IMU-a već točka putanje „upada“ između dva podatka o ubrzanju iz IMU-a, tada se računa srednja vrijednost dva ubrzanja između kojih je točka putanje snimljena te se ona pridružuje točki putanje.



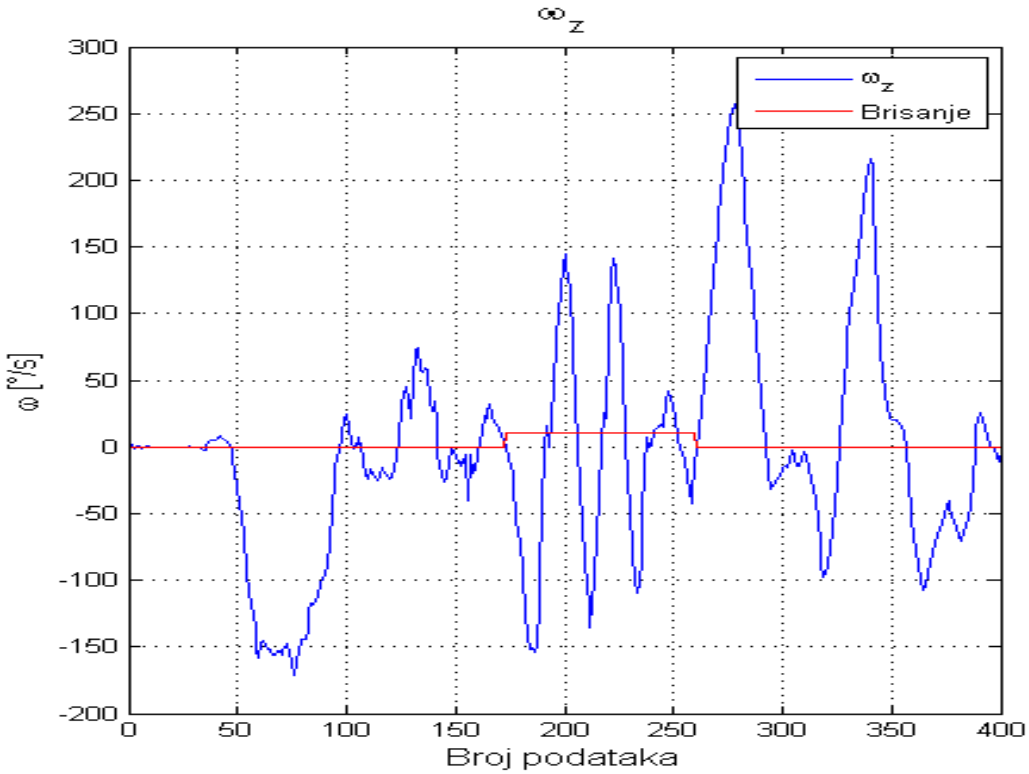
Slika 45. Vremenski pomaci prilikom prikupljanja podataka

Određivanje točnog početka i kraja radnje brisanja provodi se provjerom početka i kraja intervala određenog klasifikacijskim algoritmom opisanim u 5.2.2.. Postoje četiri moguća slučaja netočnog prepoznavanja početka odnosno kraja radnje brisanja:

1. prerano otkrivanje na početku radnje (kao točke putanje odabrane i točke prije početka radnje brisanja),
2. prekasno otkrivanje na početku radnje (nisu odabrane početne točke putanje prilikom radnje brisanja),
3. prerani prekid na kraju radnje (nisu odabrane krajnje točke putanje prilikom radnje brisanja),
4. prekasni prekid na kraju radnje (kao točke putanje odabrane i točke nakon završetka radnje brisanja).

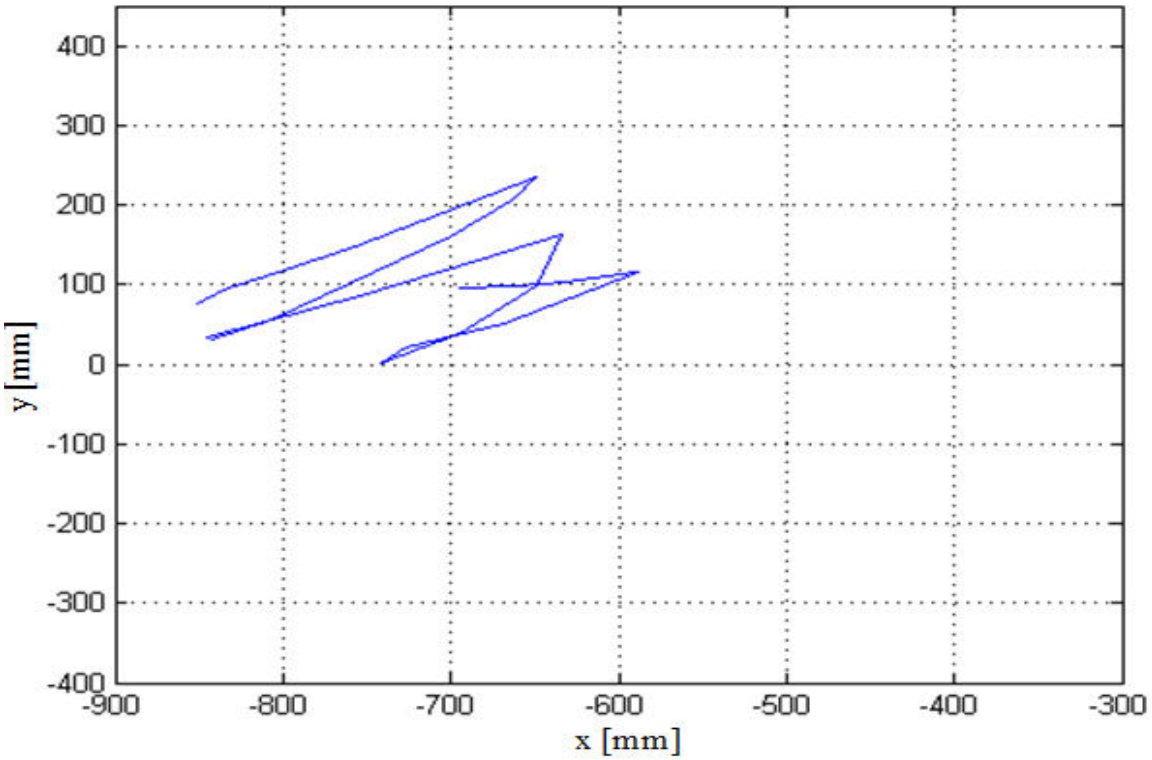
Svaki od navedena četiri moguća scenarija se provjerava pozivom MATLAB skripte *Odredi_putanju_fusion.m* unutar skripte *Snimanje_fusion.m*. Kako je klasifikacijskim algoritmom uklonjena mogućnost da ploča i spužva nisu u dodiru, ukoliko se dogodio prvi slučaj to znači da su kao početne točke putanje uzete točke koje su vrlo blizu (unutar 2 mm) jedna drugoj te spužva u tom vremenu zapravo miruje na ploči što znači da nema brisanja,

zbog toga se provjerava svaka sljedeća točka putanje sve dok razlika udaljenosti između dviju točaka putanje nije veća od 2 mm te se tada definira nova početna točka putanje. Drugi slučaj znači da je udaljenost između odabrane početne točke putanje i prethodne snimljene točke veća od 2 mm pa se zbog toga provjeravaju udaljenosti između prethodnih točaka sve dok udaljenost ne padne ispod 2 mm, te se tada definira nova početna točka putanje. Treći slučaj je vrlo sličan drugome samo što se u ovom slučaju provjerava udaljenost između odabrane zadnje točke putanje i sljedeće snimljene točke te se to ponavlja sve dok udaljenost ne padne ispod 2 mm kada se definira nova završna točka putanje. Četvrti slučaj je pak sličan prvome i znači da spužva miruje na ploči odnosno da je udaljenost između posljednje točke putanje i prethodne točke manja od 2 mm, zato se provjeravaju udaljenosti između prethodnih točaka sve dok udaljenost ne prijeđe 2 mm te se tada definira nova završna točka putanje brisanja. Kao dodatni uvjet u sva četiri slučaja provjerava se i odnos alata M2 i M3 kako bi se osiguralo uzimanje samo novih točaka putanje za koje je spužva u dodiru s pločom. Na opisani način je omogućeno točno određivanje početka i kraja radnje brisanja. Na slici 46 prikazan je primjer grafa kutne brzine oko Z-osi IMU-a te je crvenom linijom označen raspon podataka za koje je definirana radnja brisanja prethodno opisanim postupkom.



Slika 46. Primjer grafa kutne brzine oko Z-osi IMU-a tijekom učenja novih pokreta

Na slici 47 prikazan je primjer naučene putanje u koordinatama baze robota za prepoznatu radnju brisanja skupom podataka prikazanim na slici 46.



Slika 47. Primjer naučene putanje u koordinatama baze robota

5.2.5. Ponavljanje pokreta

Nakon određivanja putanje i ubrzanja za oponašanje radnje brisanja razvijen je dio C++ upravljačkog programa naziva *Ponavljanje pokreta* koji služi za ponavljanje snimljenih pokreta prilikom prethodno prepoznate radnje brisanja. Pokretanjem programa uspostavlja se komunikacija s robotom te se robotu šalje program koji se zatim izvršava u upravljačkoj jedinici robota. Pokrenuti program na robotu omogućuje primanje numeričkih vrijednosti pozicije, orijentacije i ubrzanja za oponašanje radnje brisanja. Robot ne prolazi glatko definiranom putanjom jer se robotu šalju nove numeričke vrijednosti u fiksnom vremenskom intervalu od 200 ms. Vremenski interval između slanja novih koordinata i ubrzanja je tako dugačak jer robot zbog ograničenja uslijed ugrađenih sigurnosnih algoritama ne može prolaziti kroz točke putanje bez zaustavljanja u njima što ima za posljedicu da robot u slučaju prevelike udaljenosti između točaka, a uslijed nedostatka početne brzine, za slučaj primjene kraćeg vremenskog intervala ne prođe svim točkama snimljene putanje jer mu se pošalju

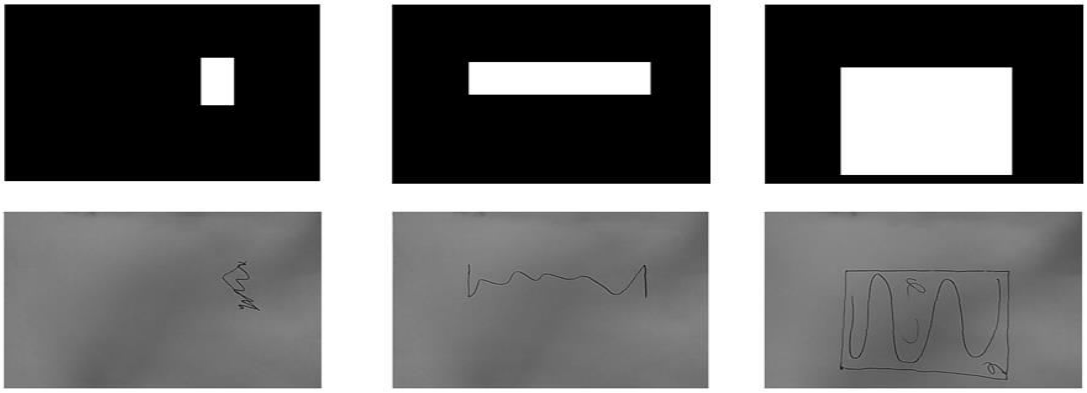
koordinate za odlazak u drugu točku, zatim u treću točku i tako dalje dok on još uvijek nije došao u prvu točku, te to ima za posljedicu preskakanje svih točaka koje su poslane prije nego što je on stigao u prvotno zadanu točku. Fiksni interval također uzrokuje dodatno zastajkivanje robota u slučaju da on stigne u točku prije isteka 200 ms gdje onda čeka naredbu za gibanje u novu točku. Unatoč navedenim problemima jasno su vidljive promjene ubrzanja prilikom gibanja robota koje su se događale i tijekom učenja pokreta brisanja. Kodovi svih navedenih programa u poglavlju 5 nalaze se u prilogu.

6. PRIMJER RADA NA TEMELJU NAUČENIH POKRETA

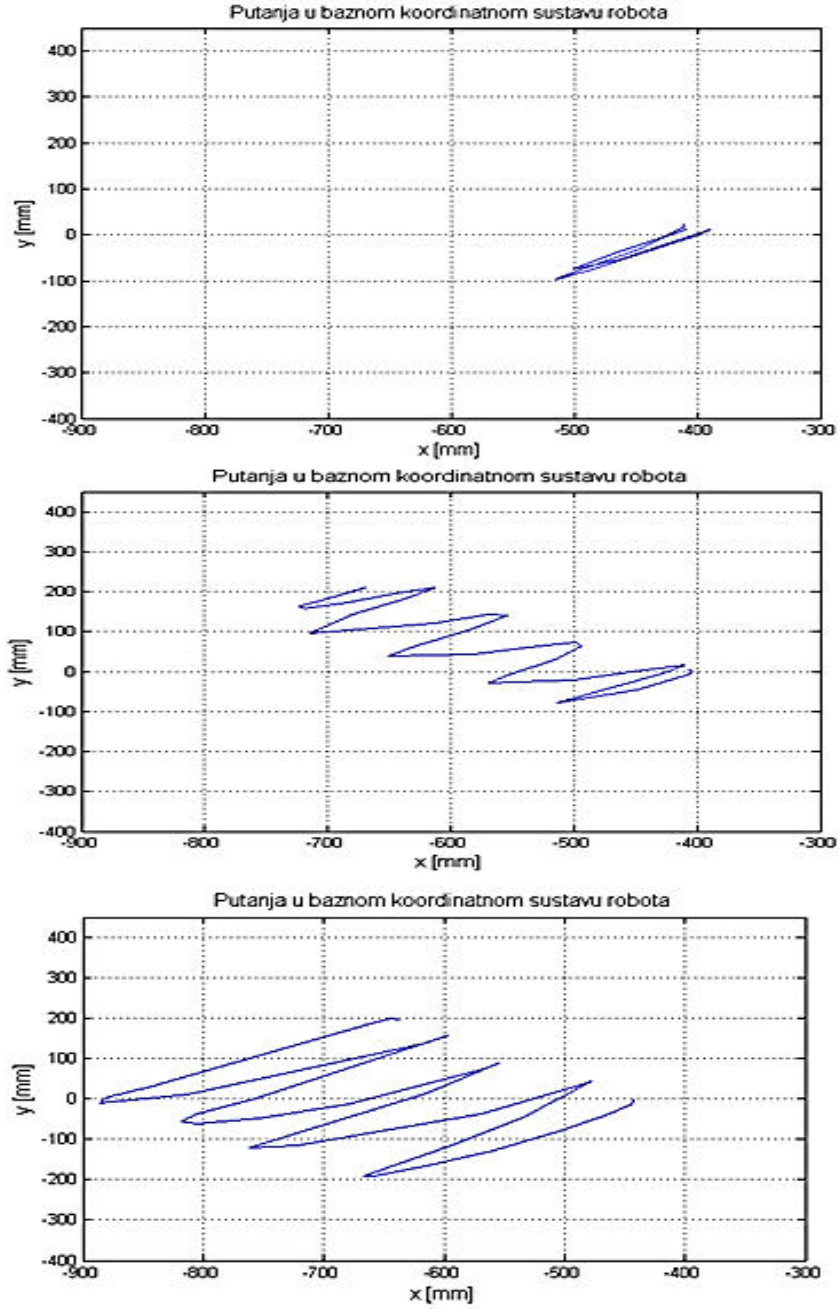
U ovom poglavlju će biti opisan primjer povezivanja naučenih radnji brisanja s različitim površinama koje je potrebno obrisati te jednostavan algoritam za rad na temelju naučenih pokreta brisanja i površina.

6.1. Učenje pokreta brisanja i površina

Zamišljen je primjer rada na temelju tri različite površine koje je potrebno pobrisati. Učenje površina koje je potrebno pobrisati obavlja se pomoću programa *Obrada slike.m* opisanog u 3.3.2., dok se učenje pokreta za brisanje tih površina obavlja pomoću MATLAB skripte *Snimanje_fusion.m* i dijela C++ upravljačkog programa nazvanog *Ucenje novih pokreta* kako je opisano u 5.2.4.. Proces učenja površina i pokreta za brisanje podrazumijeva prvo učenje površine, a zatim učenje pokreta brisanja za naučenu površinu. Podaci za svaku naučenu površinu (koordinate u koordinatnom sustavu slike i površina) se spremaju kako bi se mogli koristiti za kasniji rad. Koordinate površine u koordinatnom sustavu slike se spremaju u *.txt* datoteke imena *Povrsina1* do *Povrsina3,* koje se koriste prilikom kasnijeg određivanja nove putanje za brisanje, dok se iznosi površina određenog područja brisanja koriste za usporedbu s otkrivenim površinama prilikom rada. Učenje pokreta brisanja za svaku od tri površine se provodi nakon što su podaci za pojedinu površinu obrađeni i spremljeni, te se po završetku učenja obrasca kretanja putanje robota potrebne za brisanje pojedine površine spremaju u .txt datoteke imena *Putanja2pov1* do *Putanja2pov3* i služe za kasnije određivanje nove putanje brisanja robota. Tri naučene površine za brisanje prikazane su na slici 48. Bijeli dijelovi na slikama predstavljaju površine koje je potrebno pobrisati. Na slici 49 prikazane su naučene putanje u koordinatama baze robota za svaku od površina.



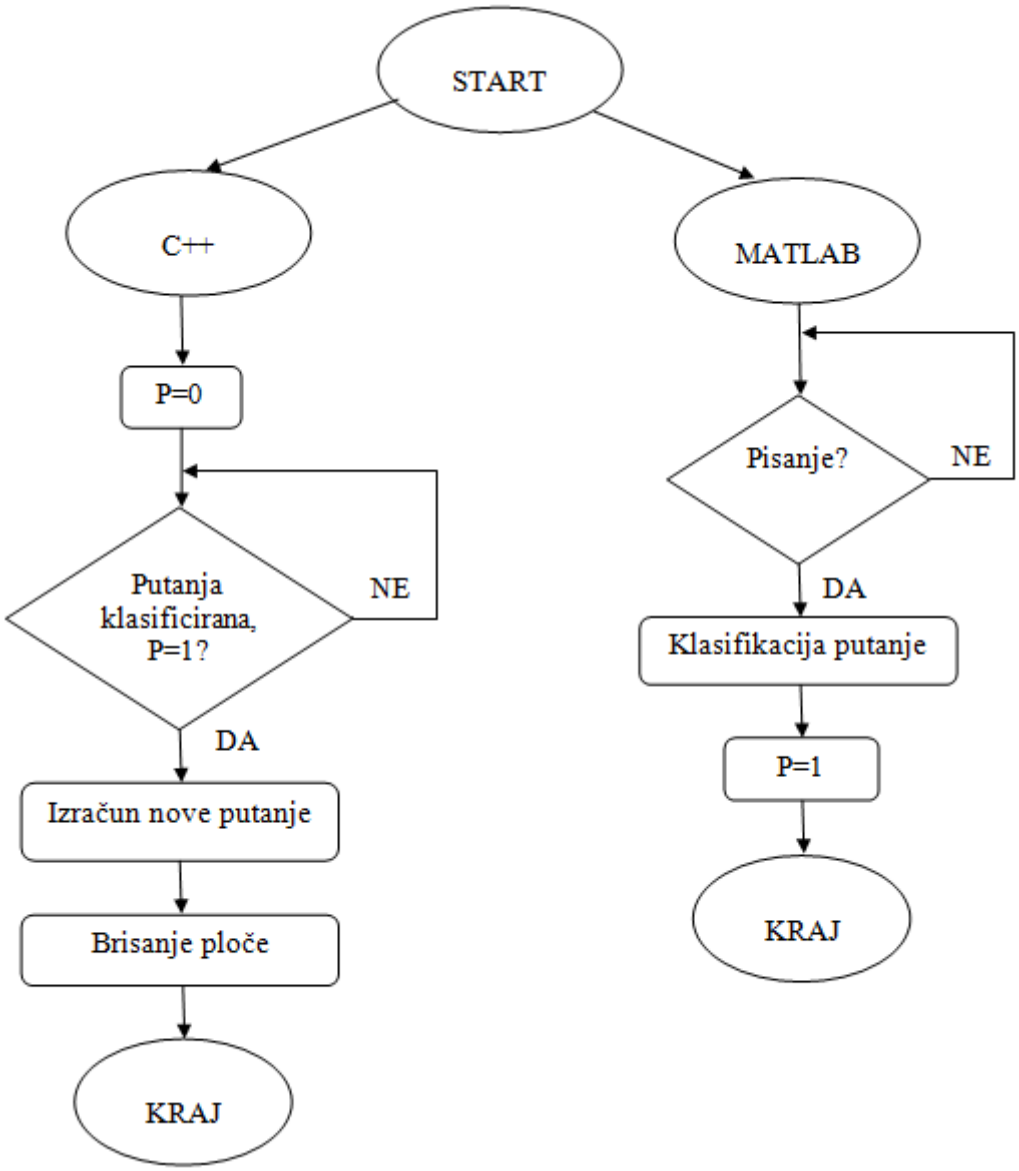
Slika 48. Naučene površine za brisanje



Slika 49. Naučene putanje brisanja u baznom koordinatnom sustavu robota

6.2. Opis rada

Rad na temelju naučenih pokreta omogućen je MATLAB skriptom *Detekcija_pisanja.m* i dijelom C++ upravljačkog program pod nazivom *Rad na temelju naucenih pokreta*. Prvo se pokreće C++ dio programa upisivanjem broja *6* unutar prozora C++ upravljačkog programa koji ulazi u petlju i čeka znak za izračunavanje nove putanje koji se šalje iz skripte *Detekcija_pisanja.m*. Nakon toga se pokreće skripta koja ulazi u petlju u kojoj se cijelo vrijeme provjerava da li je došlo do pisanja po ploči na temelju klasifikacijskog algoritma opisanog u 5.1.4., ako je otkriveno da je došlo do pisanja korisnik je obaviješten o tome pomoću skočnog prozora koji zaustavlja izvođenje programa dok korisnik ne pritisne tipku *Enter* dva puta čime se osigurava izračun nove putanje tek kada je korisnik u potpunosti završio pisanje i izašao iz radnog područja robota i upotrijebljenih optičkih sustava. Nakon dvostrukog pritiska tipke Enter snima se površina ploče i traže se uprljani dijelovi automatskim pozivom skripte *Klasifikacija_povrsine.m* koja ujedno služi i za određivanje u koju skupinu od triju naučenih pronađeno područje brisanja spada na temelju usporedbe površine pronađenog područja s prethodno naučenim površinama područja za brisanje. Zatim slijedi izračun nove putanje brisanja na temelju pomaka novog područja za brisanje u odnosu na naučeno područje brisanja unutar baznog koordinatnog sustava robota . Za izračun nove putanje brisanja koriste se koordinate nove površine unutar koordinatnog sustava slike spremljene u datoteku *Povrsina_new.txt*, a izračun nove površine se provodi u C++ dijelu programa koji se pokreće signalom iz MATLABA nakon određivanja kojoj od naučenih skupina nova površina pripada. Po izračunu nove putanje brisanja pokreće se C++ dio programa naziva *Ponavljanje pokreta* te robot briše pronađeni uprljani dio ploče. Na slici 50 prikazan je dijagram toka rada na temelju naučenih pokreta i površina pri čemu je korištena varijabla P globalna.



Slika 50. Dijagram toka rada na temelju naučenih pokreta i površina

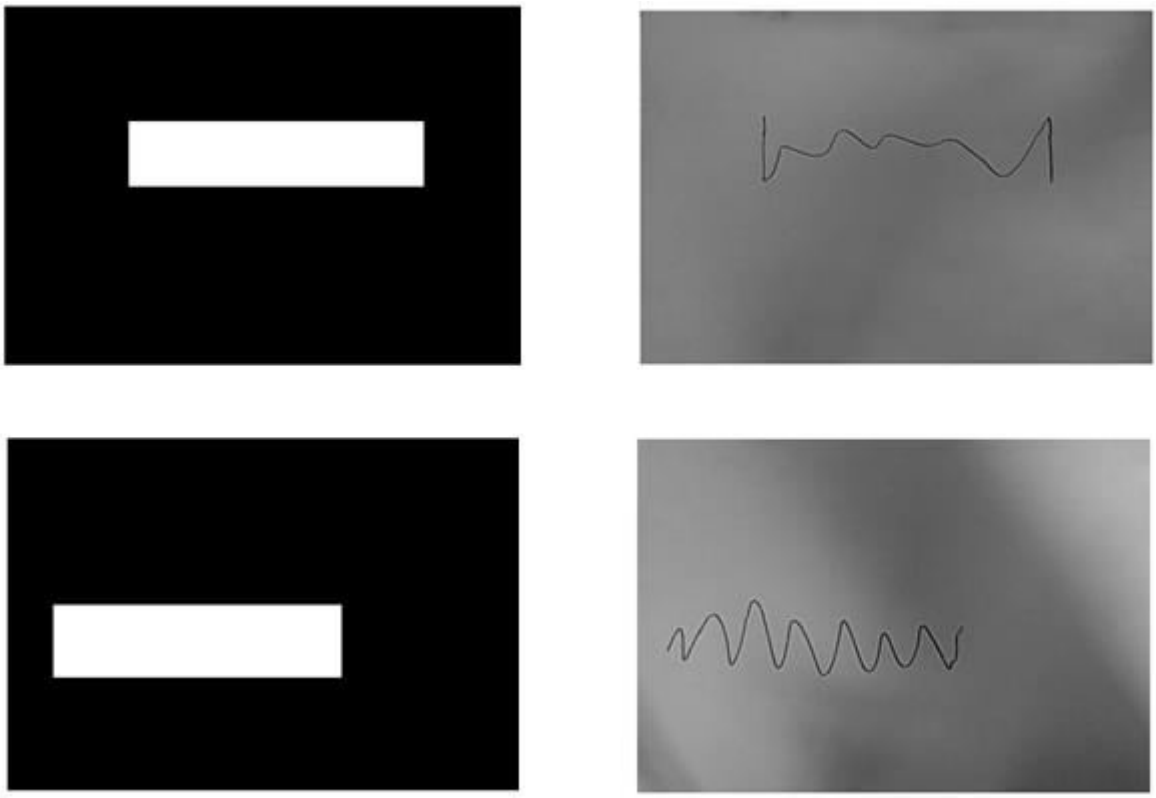
U nastavku je prikazan izračun nove putanje brisanja i dano je objašnjenje korištenih matrica:

- T_T_new – matrica translacije koordinata nove površine unutar koordinatnog sustava alata M3,
- T_M3 - pozicija i orijentacija ishodišta alata M3 u baznom koordinatnom sustavu Polarisa,
- T_MP_new - pozicija i orijentacija koordinata nove površine u baznom koordinatnom sustavu Polarisa,
- T_MP_new = T_M3 · T_T_new, (28)

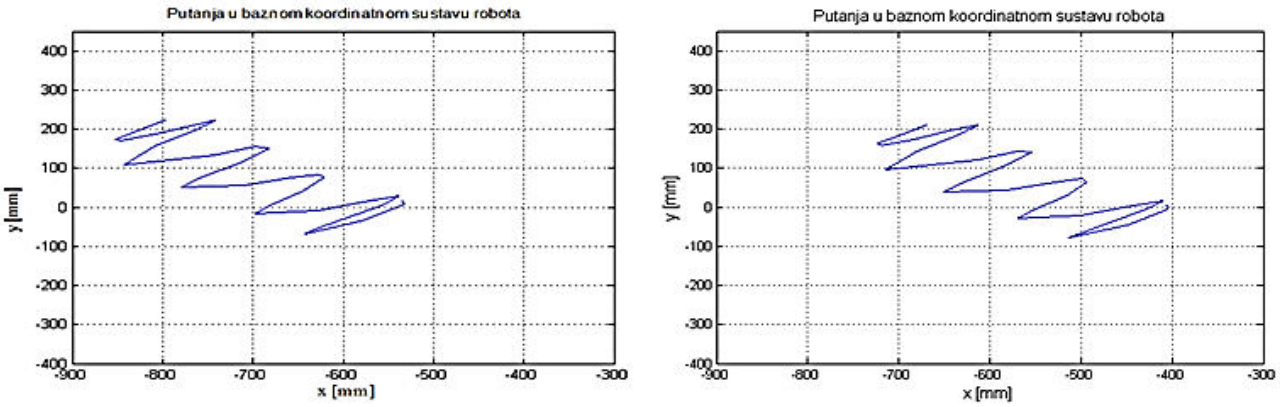
- T_MP_newinM1 - pozicija i orijentacija koordinata nove površine u koordinatnom sustavu alata M1,
- T_M1 - pozicija i orijentacija ishodišta alata M1 u baznom koordinatnom sustavu Polarisa,
- $T\_MP\_newinM1 = T\_M1^{-1} \cdot T\_MP\_new,$ (29)
- T_PinR_new - pozicija i orijentacija nove površine za brisanje u baznom koordinatnom sustavu robota,
- T_Robot_actual – pozicija i orijentacija vrha alata robota (TCP) u baznom koordinatnom sustavu robota,
- $T\_PinR\_new = T\_Robot\_actual \cdot T\_MP\_newinM1,$ (30)
- T_T_old – matrica translacije koordinata naučene površine unutar koordinatnog sustava alata M3,
- T_MP_old - pozicija i orijentacija koordinata naučene površine u baznom koordinatnom sustavu Polarisa,
- $T\_MP\_old = T\_M3 \cdot T\_T\_old,$ (31)
- T_MP_oldinM1 - pozicija i orijentacija koordinata naučene površine u koordinatnom sustavu alata M1,
- $T\_MP\_oldinM1 = T\_M1^{-1} \cdot T\_MP\_old,$ (32)
- T_PinR_old - pozicija i orijentacija naučene površine za brisanje u baznom koordinatnom sustavu robota,
- $T\_PinR\_old = T\_Robot\_actual \cdot T\_MP\_oldinM1,$ (33)
- X_T , Y_T – koordinate pomaka nove površine unutar baznog koordinatnog sustava robota,
- X_PinR_old, Y_PinR_old – koordinate naučene površine unutar baznog koordinatnog sustava robota,
- X_PinR_new, Y_PinR_new – koordinate nove površine unutar baznog koordinatnog sustava robota,
- $X\_T = X\_PinR\_new - X\_PinR\_old,$ (34)
- $Y\_T = Y\_PinR\_new - Y\_PinR\_old,$ (35)
- X_new, Y_new – koordinate nove putanje unutar baznog koordinatnog sustava robota,

- X_old, Y_old – koordinate stare putanje unutar baznog koordinatnog sustava robota,
- X_new = X_old + (X_T), (36)
- Y_new = Y_old + (Y_T). (37)

Na slici 51 u gornjem retku su prikazane naučene površine, a u donjem retku je prikazan primjer nove površine. Na slici 52 su prikazane putanje brisanja, naučena desno i nova lijevo za površine prikazane slikom 51.



Slika 51. Naučena površina (gore) i nova površina brisanja (dolje)



Slika 52. Nova putanja (lijevo) i naučena putanja brisanja (desno)

7. ZAKLJUČAK

Kako bi se ispitale mogućnosti prepoznavanja i klasifikacije različitih radnji u svrhu analize mogućnosti učenja robota korištenjem podataka dobivenih inercijskim mjernim uređajem (IMU) i vizijskim sustavom Polaris Vicra implementirana je klasifikacija radnji pisanja i brisanja ploče. Da bi se IMU mogao koristiti kao nosivi mjerni uređaj za njega je konstruirano posebno kućište te mu je dodana baterija i modul za komunikaciju putem bluetootha. Radnje se klasificiraju na temelju analize seta izmjerenih vrijednosti za učenje pri navedenim pokretima. Prvo je razvijen klasifikacijski algoritam samo na temelju podataka iz IMU-a koji omogućuje znatno jednostavniju primjenu, a zatim je taj algoritam nadograđen podacima iz vizijskog sustava koji je zahtjevniji za upotrebu zbog ograničenog radnog prostora, potrebe za dodatnim alatima, osjetljivosti na okolišne uvjete i slično. Razvijeni algoritam je dao vrlo dobre rezultate prepoznavanja radnji od 85 % za pisanje i 80 % za brisanje uz probleme s prepoznavanjem točnog početka i kraja radnji koji proizlaze iz klasifikacije statističkim parametrima koji onemogućuju točno određivanje početka i kraja radnje jer se ista radnja nikada ne obavlja identičnim pokretima. Također se pojavio i problem lažnog prepoznavanja radnje brisanja prilikom klasifikacije podacima iz IMU-a koji je riješen upotrebom podataka iz vizijskog sustava. Navedeni problem točnog prepoznavanja početka i kraja radnje je riješen za radnju brisanja upotrebom skupnih podataka iz IMU-a i vizijskog sustava pošto je u ovom radu bio naglasak na mogućnosti učenja robota brisanju na temelju prepoznavanja radnje brisanja, dok je za provedenu implementaciju primjera rada na temelju naučenih pokreta brisanja i klasifikacije pisanja bilo dovoljno samo prepoznati pojavu radnje pisanja, ali ne i točan početak ili kraj te radnje. Učenje robota radnji brisanja dalo je zadovoljavajuće rezultate pri čemu je robot oponašao naučenu radnju prolaskom po istoj putanji i približno jednakim ubrzanjima. Upotrebom robota nekog drugog proizvođača koji ima drugačije algoritme sigurnosti postoji mogućnost još vjernijeg oponašanja gibanja točnijim ubrzanjima.

Jednostavan primjer rada robota na temelju naučenih pokreta brisanja pokazuje veliki potencijal učenja robota višemodalnom interakcijom za razvoj kognitivnih robota koji su sposobni na temelju nekoliko jednostavnih naučenih radnji riješiti neki novi njima nepoznati problem. Ovim radom je stvoren alat za jednostavno učenje robota radnji brisanja koji može biti prilagođen i za neke druge radnje provedbom dodatnih mjerenja i analize pokreta

prilikom radnji koje bi mogle biti korisne kod učenja robota zadacima koje obavljaju u industrijskom okruženju kao što je lakiranje, poliranje i slično. Unaprijeđenjem klasifikacijskog algoritma i upotrebom više inercijskih mjernih uređaja (za mapiranje pokreta u prostoru) koji bi se nosili na različitim dijelovima ruke (dlan, podlaktica, nadlaktica) mogle bi se istražiti mogućnosti učenja robota pokretima bez upotrebe vizijskog sustava čiji je glavni nedostatak ograničen mjerni volumen.

LITERATURA

[1] Mladen Popović: SENZORI I MERENJA, Beograd, 2004.

[2] https://www.sparkfun.com/products/9045, pristupljeno: 10.04.2017.

[3] https://learn.sparkfun.com/tutorials/gyroscope?_ga=1.239211909.78352075.14894874, pristupljeno: 25.04.2017.

[4] https://www.sparkfun.com/products/9793, pristupljeno: 25.04.2017.

[5] http://cdn.sparkfun.com/datasheets/Sensors/Magneto/HMC5883L-FDS.pdf, pristupljeno: 25.04.2017.

[6] https://www.sparkfun.com/products/retired/10736, pristupljeno: 25.04.2017.

[7] https://plus.google.com/+Ndigitalinc, pristupljeno: 29.06.2017.

[8] https://www.ndigital.com/medical/products/polaris-family/features/vicra-measurement-volume/, pristupljeno: 29.06.2017.

[9] http://support.logitech.com/en_ch/product/webcam-c210, pristupljeno: 29.06.2017.

[10] http://www.zacobria.com/universal-robots-zacobria-industrial-robot-ur5-ur10-specifications.html, pristupljeno: 30.06.2017.

[11] https://hr.wikipedia.org/wiki/TCP, pristupljeno: 30.06.2017.

[12] Kirić, N..: Diplomski rad, Zagreb, 2015.

[13] https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation, pristupljeno: 01.07.2017.

[14] https://www.sparkfun.com/products/retired/10736, pristupljeno: 25.04.2017.

[15] Vitez, N..: Završni rad, Zagreb, 2016.

PRILOZI

I. Programski kod MATLAB

II. Programski kod C++

III. CD-R disc

I. Programski kod MATLAB

I.1. Citanje_podataka.m

```
clearvars -except s11 cam
%tic
t=1;
i=1;
pisanje=0;
% load('s11');
raspon=1200;
a=1:1:raspon;
flushinput(s11);
pause(0.5);
s11.bytesavailable;
yaw=zeros(1,raspon);
pitch=zeros(1,raspon);
roll=zeros(1,raspon);

acc_x=zeros(1,raspon);
acc_y=zeros(1,raspon);
acc_z=zeros(1,raspon);
gyro_x=zeros(1,raspon);
gyro_y=zeros(1,raspon);
gyro_z=zeros(1,raspon);

accx=zeros(1,raspon/3);
accy=zeros(1,raspon/3);
accz=zeros(1,raspon/3);
gyrox=zeros(1,raspon/3);
gyroy=zeros(1,raspon/3);
gyroz=zeros(1,raspon/3);
YAW=zeros(1,raspon/3);
PITCH=zeros(1,raspon/3);
ROLL=zeros(1,raspon/3);
arx=zeros(1,raspon/3);
ary=zeros(1,raspon/3);
arz=zeros(1,raspon/3);
mag=zeros(1,raspon/3);
mag_r=zeros(1,raspon/3);
cr=zeros(1,raspon/3);
cp=zeros(1,raspon/3);
cy=zeros(1,raspon/3);
sy=zeros(1,raspon/3);
sp=zeros(1,raspon/3);
sr=zeros(1,raspon/3);
cln=1;
cl=1;
vrijemee=0;
vrijemeuk=0;
stanka=0;
in_1=fscanf(s11);
 pause(0.025)

 vrijeme_start=clock;
while(i<raspon)
```

```
in=fscanf(s11);
data=textscan(in,'%s%f%f%f','delimiter',',');
[type,jen,dva,tri]=deal(data{:});

if strcmp('#A-C',type(1,1))==1

    acc_x(i)=jen/61.35;
    acc_x(i+1)=jen/61.35;
    acc_x(i+2)=acc_x(i+1);
    if i>3
       razlika=abs(abs(acc_x(i))-abs(acc_x(i-1)));
       if razlika<0.02
           acc_x(i)=acc_x(i-1);
           acc_x(i+1)=acc_x(i-1);
           acc_x(i+2)=acc_x(i-1);
       end
    end

    acc_y(i)=dva/62.775;
    acc_y(i+1)=dva/62.775;
    acc_y(i+2)=acc_y(i+1);
    if i>3
       razlika=abs(abs(acc_y(i))-abs(acc_y(i-1)));
       if razlika<0.02
           acc_y(i)=acc_y(i-1);
           acc_y(i+1)=acc_y(i-1);
           acc_y(i+2)=acc_y(i-1);
       end
    end

    acc_z(i)=tri/62.635;
    acc_z(i+1)=tri/62.635;
    acc_z(i+2)=acc_z(i+1);
    if i>3
       razlika=abs(abs(acc_z(i))-abs(acc_z(i-1)));
       if razlika<0.02
           acc_z(i)=acc_z(i-1);
           acc_z(i+1)=acc_z(i-1);
           acc_z(i+2)=acc_z(i-1);
       end
    end

elseif strcmp('#M-C',type(1,1))==1

      continue
elseif strcmp('#G-C',type(1,1))==1

    if i>3
      gyro_x(i-1)=jen/14.735;
      gyro_x(i+1)=jen/14.735;
      gyro_x(i)=jen/14.735;
      dif_g=abs(abs(gyro_x(i))-abs(gyro_x(i-2)));
      if dif_g<0.3
          gyro_x(i-1)=gyro_x(i-2);
          gyro_x(i)=gyro_x(i-2);
          gyro_x(i+1)=gyro_x(i-2);
      end
```

```
        gyro_y(i-1)=dva/14.735;
        gyro_y(i+1)=dva/14.735;
        gyro_y(i)=jen/14.735;
        dif_g=abs(abs(gyro_y(i))-abs(gyro_y(i-2)));
        if dif_g<0.03
            gyro_y(i-1)=gyro_y(i-2);
            gyro_y(i)=gyro_y(i-2);
            gyro_y(i+1)=gyro_y(i-2);
        end

        gyro_z(i-1)=tri/14.735;
        gyro_z(i+1)=tri/14.735;
        gyro_z(i)=tri/14.735;
        dif_g=abs(abs(gyro_z(i))-abs(gyro_z(i-2)));
        if dif_g<0.03
            gyro_z(i-1)=gyro_z(i-2);
            gyro_z(i)=gyro_z(i-2);
            gyro_z(i+1)=gyro_z(i-2);
        end
    end

elseif strcmp('#YPR',type(1,1))==1

    if i>4
     yaw(i-2)=jen;
     yaw(i-1)=jen;
     yaw(i)=jen;
      dif=abs(abs(yaw(i))-abs(yaw(i-3)));
      if dif<0.3
          yaw(i)=yaw(i-3);
          yaw(i-1)=yaw(i-3);
          yaw(i-2)=yaw(i-3);
      end
    end
    if i>4
        pitch(i-2)=dva;
        pitch(i-1)=dva;
        pitch(i)=dva;
      dif=abs(abs(pitch(i))-abs(pitch(i-3)));
      if dif<0.3
          pitch(i)=pitch(i-3);
          pitch(i-1)=pitch(i-3);
          pitch(i-2)=pitch(i-3);
      end
    end

    if i>4
        roll(i-2)=tri;
        roll(i-1)=tri;
        roll(i)=tri;
      dif=abs(abs(roll(i))-abs(roll(i-3)));
      if dif<0.3
          roll(i)=roll(i-3);
          roll(i-1)=roll(i-3);
          roll(i-2)=roll(i-3);
      end
    end
    else
    disp('Citanje nije uspjelo')
```

```
    end

    %%% ciscenje_podataka
    cl=1;
    for j=4:3:i

    accx(cl)=acc_x(j);
    accy(cl)=acc_y(j);
    accz(cl)=acc_z(j);
    gyrox(cl)=gyro_x(j);
    gyroy(cl)=gyro_y(j);
    gyroz(cl)=gyro_z(j);
    YAW(cl)=yaw(j);
    PITCH(cl)=pitch(j);
    ROLL(cl)=roll(j);

     mag(cl)=sqrt(accx(cl)^2+accy(cl)^2+accz(cl)^2);
     cl=cl+1;
    end

     %%%%%%%%
    s11.bytesavailable;
    i=i+1;

    if mod(i,3)==0
        stanka=0.020-vrijemee;
         vrijemeuk=vrijemeuk+vrijemee+stanka;
        pause(stanka)
        vrijemee=0;
    end
end
%%%%%  POCETAK RACUNANJA PRAVOG UBRZANJA

cl=1;
   for cl=1:400
    cy=cos(YAW(cl)*pi/180);
    cp=cos(PITCH(cl)*pi/180);
    cr=cos(ROLL(cl)*pi/180);
    sy=sin(YAW(cl)*pi/180);
    sp=sin(PITCH(cl)*pi/180);
    sr=sin(ROLL(cl)*pi/180);

    R=[cp*cy (cy*sr*sp)-(cr*sy) (sr*sy)+(cr*cy*sp); cp*sy
(cr*cy)+(sr*sp*sy) (cr*sp*sy)-(cy*sr); -sp cp*sr cr*cp];
    gimu=R'*[0;0;1];

    aimu=[accx(cl);accy(cl);accz(cl)];
    ap=gimu-aimu;
    mag_r(cl)=sqrt((ap(1))^2+(ap(2))^2+(ap(3))^2);
   end
%%%%%%%% KRAJ RACUNANJA PRAVOG UBRZANJA

save('test')
```

I.2. Obrada_slike.m

```
clearvars -except cam s11

slika=getsnapshot(cam);

siva=rgb2gray(slika);

%% KALIBRACIJA
B=imrotate(siva,179);
rect=[94 114 532 367];
C=imcrop(B,rect); %% kalibrirana slika
figure(3)
imshow(C)
% 1pixsel=1.4mm
%%
dim=size(C);
C(1:10,:)=255;
C(:,end-50:end)=255;

for i=1:dim(1)

    for j=1:dim(2)


        if C(i,j)<50
            bin(i,j)=1;

        else
            bin(i,j)=0;
        end
    end
end

%%%% Trazenje prvog i zadnjeg piksela pa stvaranje velike regije
prvi=0;
for i=1:dim(1)-1

    for j=1:dim(2)-1


        if bin(i,j)==1 && prvi==0

            poc_r1=i;
            poc_s1=j;
            prvi=1;
         elseif bin(i,j)==1
            zad_r1=i;
            zad_s1=j;
        else
            continue
        end


    end
end
```

```
prvi=0;
for i=1:dim(2)-1

    for j=1:dim(1)-1


        if bin(j,i)==1 && prvi==0

            poc_r2=j;
            poc_s2=i;
            prvi=1;
         elseif bin(j,i)==1
            zad_r2=j;
            zad_s2=i;
        else
            continue
        end

    end
end

redak=[poc_r1 poc_r2 zad_r1 zad_r2];
stupac=[poc_s1 poc_s2 zad_s1 zad_s2];

poc_r=min(redak);
zad_r=max(redak);
poc_s=min(stupac);
zad_s=max(stupac);

povrsina=zeros(dim(1),dim(2));
povrsina(poc_r:zad_r,poc_s:zad_s)=1;
imshow(povrsina)
figure(4)
imshow(C)

dim1=abs((poc_r-zad_r)*1.4); %%smjer x osi
dim2=abs((poc_s-zad_s)*1.4); %%smjer y osi
POV_new=dim1*dim2;

x_koo_01=(4-poc_r)*1.4;
x_koo_02=(4-zad_r)*1.4;
y_koo_01=(134-poc_s)*1.4;
y_koo_02=(134-zad_s)*1.4;
```

I.3. **Analiza1.m**

```
clear all
load('learning_set_new')


sr_vr_pisanje=zeros(14,25);
sr_vr_brisanje2=zeros(14,40);
max_pisanje=zeros(14,25);
max_brisanje2=zeros(14,40);
iqr_pisanje=zeros(14,25);
iqr_brisanje2=zeros(14,40);
std_pisanje=zeros(14,25);
std_brisanje2=zeros(14,40);
rms_pisanje=zeros(14,25);
rms_brisanje2=zeros(14,40);
sr_vr_pisanje_abs=zeros(14,25);
sr_vr_brisanje2_abs=zeros(14,40);
max_pisanje_abs=zeros(14,25);
max_brisanje2_abs=zeros(14,40);
range_pisanje=zeros(14,25);
range_brisanje2=zeros(14,40);

for i=1:25

    for j=1:14

       sr_vr_pisanje_abs(j,i)=mean(abs(pisanje_cl{j,i}));

        sr_vr_pisanje(j,i)=mean((pisanje_cl{j,i}));

        iqr_pisanje(j,i)=iqr(pisanje_cl{j,i});

        std_pisanje(j,i)=std(pisanje_cl{j,i});

       rms_pisanje(j,i)=rms(pisanje_cl{j,i});

        sorted_pisanje_abs=sort(abs(pisanje_cl{j,i}));

        max_pisanje_abs(j,i)=sorted_pisanje_abs(1,50);

        sort_pisanje=sort((pisanje_cl{j,i}));

        max_pisanje(j,i)=sort_pisanje(1,50);

        range_pisanje(j,i)=sort_pisanje(1,50)-sort_pisanje(1,1);

    end
end


%% brisanje

for i=1:25
```

```
    for j=1:14

       sr_vr_brisanje2_abs(j,i)=mean(abs(brisanje2_cl{j,i}));

        sr_vr_brisanje2(j,i)=mean((brisanje2_cl{j,i}));

        iqr_brisanje2(j,i)=iqr(brisanje2_cl{j,i});

        std_brisanje2(j,i)=std(brisanje2_cl{j,i});

       rms_brisanje2(j,i)=rms(brisanje2_cl{j,i});

        sorted_brisanje2_abs=sort(abs(brisanje2_cl{j,i}));

        max_brisanje2_abs(j,i)=sorted_brisanje2_abs(1,50);

        sort_brisanje2=sort((brisanje2_cl{j,i}));

        max_brisanje2(j,i)=sort_brisanje2(1,50);

        range_brisanje2(j,i)=sort_brisanje2(1,50)-sort_brisanje2(1,1);

    end
end


for i=1:14



      sort_pisanje=sort(sr_vr_pisanje_abs(i,:));
      min_max_stat{1}(i,1)=sort_pisanje(1,25);
      min_max_stat{1}(i,3)=sort_pisanje(1,1);

      sort_brisanje=sort(sr_vr_brisanje2_abs(i,:));
      min_max_stat{1}(i,2)=sort_brisanje(1,25);
      min_max_stat{1}(i,4)=sort_brisanje(1,1);

 end

  %%%mean
 for i=1:14


      sort_pisanje=sort(sr_vr_pisanje(i,:));
      min_max_stat{2}(i,1)=sort_pisanje(1,25);
      min_max_stat{2}(i,3)=sort_pisanje(1,1);

      sort_brisanje=sort(sr_vr_brisanje2(i,:));
      min_max_stat{2}(i,2)=sort_brisanje(1,25);
      min_max_stat{2}(i,4)=sort_brisanje(1,1);

 end

 %%% IQR
  for i=1:14
```

```
    sort_pisanje=sort(iqr_pisanje(i,:));
    min_max_stat{3}(i,1)=sort_pisanje(1,25);
    min_max_stat{3}(i,3)=sort_pisanje(1,1);

    sort_brisanje=sort(iqr_brisanje2(i,:));
    min_max_stat{3}(i,2)=sort_brisanje(1,25);
    min_max_stat{3}(i,4)=sort_brisanje(1,1);

end


%%% STD
  for i=1:14


    sort_pisanje=sort(std_pisanje(i,:));
    min_max_stat{4}(i,1)=sort_pisanje(1,25);
    min_max_stat{4}(i,3)=sort_pisanje(1,1);

    sort_brisanje=sort(std_brisanje2(i,:));
    min_max_stat{4}(i,2)=sort_brisanje(1,25);
    min_max_stat{4}(i,4)=sort_brisanje(1,1);


  end


  %%%  RMS
 for i=1:14


    sort_pisanje=sort(rms_pisanje(i,:));
    min_max_stat{5}(i,1)=sort_pisanje(1,25);
    min_max_stat{5}(i,3)=sort_pisanje(1,1);

    sort_brisanje=sort(rms_brisanje2(i,:));
    min_max_stat{5}(i,2)=sort_brisanje(1,25);
    min_max_stat{5}(i,4)=sort_brisanje(1,1);


 end

 %%%% MAX(abs)


    for i=1:14


    sort_pisanje=sort(max_pisanje_abs(i,:));
    min_max_stat{6}(i,1)=sort_pisanje(1,25);
    min_max_stat{6}(i,3)=sort_pisanje(1,1);

    sort_brisanje=sort(max_brisanje2_abs(i,:));
    min_max_stat{6}(i,2)=sort_brisanje(1,25);
```

```
      min_max_stat{6}(i,4)=sort_brisanje(1,1);

      end

      %%%% MAX

  for i=1:14


      sort_pisanje=sort(max_pisanje(i,:));
      min_max_stat{7}(i,1)=sort_pisanje(1,25);
      min_max_stat{7}(i,3)=sort_pisanje(1,1);

      sort_brisanje=sort(max_brisanje2(i,:));
      min_max_stat{7}(i,2)=sort_brisanje(1,25);
      min_max_stat{7}(i,4)=sort_brisanje(1,1);

  end

  %%%%RANGE

   for i=1:14


      sort_pisanje=sort(range_pisanje(i,:));
      min_max_stat{8}(i,1)=sort_pisanje(1,25);
      min_max_stat{8}(i,3)=sort_pisanje(1,1);

      sort_brisanje=sort(range_brisanje2(i,:));
      min_max_stat{8}(i,2)=sort_brisanje(1,25);
      min_max_stat{8}(i,4)=sort_brisanje(1,1);


  end
```

I.4. Analiza2.m

```
clear all
load('min_max_stat_dipl')


for  j=1:8      %%% znacajke (mean(abs)....range)

    for i=1:14  %%% vrijednosti ocitanja(acc_x...mag_r)

        if min_max_stat{j}(i,4)>min_max_stat{j}(i,1)

            sorted=sort(min_max_stat{j}(i,:));
            gain{j}(i)=sorted(1,3)/sorted(1,2);
            if sorted(1,3)<0 && sorted(1,2)<0
                gain{j}(i)=sorted(1,2)/sorted(1,3);
            elseif sorted(1,3)>0 && sorted(1,2)<0
                r=abs(sorted(1,3))+abs(sorted(1,2));
                gain{j}(i)=r/sorted(1,3);
            end
```

```
        elseif  min_max_stat{j}(i,3)>min_max_stat{j}(i,2)

            sorted=sort(min_max_stat{j}(i,:));
           gain{j}(i)=sorted(1,3)/sorted(1,2);

           if sorted(1,3)<0 && sorted(1,2)<0
                gain{j}(i)=sorted(1,2)/sorted(1,3);
            elseif sorted(1,3)>0 && sorted(1,2)<0
                r=abs(sorted(1,3))+abs(sorted(1,2));
                gain{j}(i)=r/sorted(1,3);
            end

        else
            gain{j}(i)=0;
        end

    end
end



xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{1}','B4:B17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{2}','C4:C17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{3}','D4:D17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{4}','E4:E17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{5}','F4:F17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{6}','G4:G17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{7}','H4:H17');
xlswrite('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx',gai
n{8}','I4:I17');

winopen('Tablica_za_odredivanje_klasifikacijskih_parametara_dipl.xlsx')
```

I.5. Online_provjera_pokreta.m

```
clearvars -except s11
pisanje=0;
t=1;
i=1;
raspon=1200;
a=1:1:raspon;
flushinput(s11);
pause(0.5);
s11.bytesavailable;
yaw=zeros(1,raspon);
pitch=zeros(1,raspon);
roll=zeros(1,raspon);
acc_x=zeros(1,raspon);
acc_y=zeros(1,raspon);
acc_z=zeros(1,raspon);
gyro_x=zeros(1,raspon);
```

```
gyro_y=zeros(1,raspon);
gyro_z=zeros(1,raspon);
ploting_h=zeros(1,raspon/3);
ploting_hn=zeros(1,raspon/3);
ploting_s=zeros(1,raspon/3);
ploting_sn=zeros(1,raspon/3);
ploting_as=zeros(1,raspon/3);
ploting_ah=zeros(1,raspon/3);
ploting_ar=zeros(1,raspon/3);
ploting_ap=zeros(1,raspon/3);
ploting_ab=zeros(1,raspon/3);
sr_vr=1;
max_vr=0;
ham_p=0;
scr_p=0;
unk=0;
zbroj_h=0;
zbroj_s=0;
zbroj_r=0;
zbroj_p=0;
zbroj_b=0;
max_roll=0;

accx=zeros(1,raspon/3);
accy=zeros(1,raspon/3);
accz=zeros(1,raspon/3);
gyrox=zeros(1,raspon/3);
gyroy=zeros(1,raspon/3);
gyroz=zeros(1,raspon/3);
YAW=zeros(1,raspon/3);
PITCH=zeros(1,raspon/3);
ROLL=zeros(1,raspon/3);
arx=zeros(1,raspon/3);
ary=zeros(1,raspon/3);
arz=zeros(1,raspon/3);
mag=zeros(1,raspon/3);
mag_r=zeros(1,raspon/3);
cr=zeros(1,raspon/3);
cp=zeros(1,raspon/3);
cy=zeros(1,raspon/3);
sy=zeros(1,raspon/3);
sp=zeros(1,raspon/3);
sr=zeros(1,raspon/3);
cln=1;
cl=1;
vrijemee=0;
vrijemeuk=0;
stanka=0;
in_1=fscanf(s11);
 pause(0.025)

vrijeme_start=clock;
while(i<raspon)

     tic
    in=fscanf(s11);
    data=textscan(in,'%s%f%f%f','delimiter',',');
    [type,jen,dva,tri]=deal(data{:});

    if strcmp('#A-C',type(1,1))==1
```

```
        acc_x(i)=jen/61.35;
        acc_x(i+1)=jen/61.35;
        acc_x(i+2)=acc_x(i+1);
        if i>3
           razlika=abs(abs(acc_x(i))-abs(acc_x(i-1)));
           if razlika<0.02
               acc_x(i)=acc_x(i-1);
               acc_x(i+1)=acc_x(i-1);
               acc_x(i+2)=acc_x(i-1);
           end
        end

        acc_y(i)=dva/62.775;
        acc_y(i+1)=dva/62.775;
        acc_y(i+2)=acc_y(i+1);
        if i>3
           razlika=abs(abs(acc_y(i))-abs(acc_y(i-1)));
           if razlika<0.02
               acc_y(i)=acc_y(i-1);
               acc_y(i+1)=acc_y(i-1);
               acc_y(i+2)=acc_y(i-1);
           end
        end

        acc_z(i)=tri/62.635;
        acc_z(i+1)=tri/62.635;
        acc_z(i+2)=acc_z(i+1);
        if i>3
           razlika=abs(abs(acc_z(i))-abs(acc_z(i-1)));
           if razlika<0.02
               acc_z(i)=acc_z(i-1);
               acc_z(i+1)=acc_z(i-1);
               acc_z(i+2)=acc_z(i-1);
           end
        end

    elseif strcmp('#M-C',type(1,1))==1

          continue
    elseif strcmp('#G-C',type(1,1))==1

        if i>3
          gyro_x(i-1)=jen/14.735;
          gyro_x(i+1)=jen/14.735;
          gyro_x(i)=jen/14.735;
          dif_g=abs(abs(gyro_x(i))-abs(gyro_x(i-2)));
          if dif_g<0.3
              gyro_x(i-1)=gyro_x(i-2);
              gyro_x(i)=gyro_x(i-2);
              gyro_x(i+1)=gyro_x(i-2);
          end

          gyro_y(i-1)=dva/14.735;
          gyro_y(i+1)=dva/14.735;
          gyro_y(i)=jen/14.735;
          dif_g=abs(abs(gyro_y(i))-abs(gyro_y(i-2)));
          if dif_g<0.03
              gyro_y(i-1)=gyro_y(i-2);
              gyro_y(i)=gyro_y(i-2);
```

```
            gyro_y(i+1)=gyro_y(i-2);
        end

        gyro_z(i-1)=tri/14.735;
        gyro_z(i+1)=tri/14.735;
        gyro_z(i)=tri/14.735;
        dif_g=abs(abs(gyro_z(i))-abs(gyro_z(i-2)));
        if dif_g<0.03
            gyro_z(i-1)=gyro_z(i-2);
            gyro_z(i)=gyro_z(i-2);
            gyro_z(i+1)=gyro_z(i-2);
        end

    end

elseif strcmp('#YPR',type(1,1))==1

    if i>4
     yaw(i-2)=jen;
     yaw(i-1)=jen;
     yaw(i)=jen;
      dif=abs(abs(yaw(i))-abs(yaw(i-3)));
      if dif<0.3
          yaw(i)=yaw(i-3);
          yaw(i-1)=yaw(i-3);
          yaw(i-2)=yaw(i-3);
      end
    end
    if i>4
        pitch(i-2)=dva;
        pitch(i-1)=dva;
        pitch(i)=dva;
      dif=abs(abs(pitch(i))-abs(pitch(i-3)));
      if dif<0.3
          pitch(i)=pitch(i-3);
          pitch(i-1)=pitch(i-3);
          pitch(i-2)=pitch(i-3);
      end
    end
    if i>4
        roll(i-2)=tri;
        roll(i-1)=tri;
        roll(i)=tri;
      dif=abs(abs(roll(i))-abs(roll(i-3)));
      if dif<0.3
          roll(i)=roll(i-3);
          roll(i-1)=roll(i-3);
          roll(i-2)=roll(i-3);
      end
    end
   else
   disp('Citanje nije uspjelo')
end


%%% PROVJERA KLASE


%%%prvo_ciscenje_podataka
cl=1;
```

```
    for j=4:3:i

    accx(cl)=acc_x(j);
    accy(cl)=acc_y(j);
    accz(cl)=acc_z(j);
    gyrox(cl)=gyro_x(j);
    gyroy(cl)=gyro_y(j);
    gyroz(cl)=gyro_z(j);
    YAW(cl)=yaw(j);
    PITCH(cl)=pitch(j);
    ROLL(cl)=roll(j);



    mag(cl)=sqrt(accx(cl)^2+accy(cl)^2+accz(cl)^2);

     cl=cl+1;
    end

    if i>300 && mod(i,3)==0
        n=i/3-50;
        max_m_yaw=max(YAW(n:i/3-1));
        iqr_gy=iqr(gyroy(n:i/3-1));
        rms_ar=rms(mag(n:i/3-1));
          max_gz=max(gyroz(n:i/3-1));
          rms_yaw=rms(YAW(n:i/3-1));
          sr_vr_ax=mean(accx(n:i/3-1));
          sr_vr_yaw=mean(YAW(n:i/3-1));
          sr_vr_ay=mean(accy(n:i/3-1));
          std_gz=std(gyroz(n:i/3-1));
          rms_accy=rms(accy(n:i/3-1));
          rms_gz=rms(gyroz(n:i/3-1));
          sorted_s=sort(accy(n:i/3-1));
         ay_dif=sorted_s(1,50)-sorted_s(1,1);



 %%


            if  sr_vr_ay>0.3 && sr_vr_ay<0.72 && std_gz>14 && std_gz<115 &&
rms_gz>16 && rms_gz<115 && sr_vr_ax>0.1 && sr_vr_ax<0.7 && ay_dif>0.8 &&
ay_dif<3.5
                zbroj_b=zbroj_b+1;
                zbroj_s=0;
                zbroj_r=0;
                zbroj_p=0;
                zbroj_h=0;
                if zbroj_b>40

                      ploting_ab(n-40:i/3-1)=1;
                end
            elseif   sr_vr_ay>0.75 && sr_vr_ay<1.05 && std_gz>4 &&
std_gz<14 && rms_gz>4 && rms_gz<15
                  zbroj_p=zbroj_p+1;
                zbroj_s=0;
                zbroj_r=0;
                zbroj_b=0;
                zbroj_h=0;
```

```
                if zbroj_p>40
                      pisanje=1;
                      ploting_ap(n-40:i/3-1)=1;
                end
            else
                  unk=unk+1;
                  zbroj_s=0;
                  zbroj_h=0;
                  zbroj_r=0;
                  zbroj_p=0;
                  zbroj_b=0;
                  ploting_ab(i/3-40:i/3-1)=0;
                  ploting_ap(i/3-40:i/3-1)=0;
            end

    end
    %%%%%%%%
    s11.bytesavailable;
    i=i+1;

    vrijeme=toc;
    vrijemee=vrijemee+vrijeme;
    if mod(i,3)==0
        stanka=0.020-vrijemee;
         vrijemeuk=vrijemeuk+vrijemee+stanka;
        pause(stanka)
        vrijemee=0;
    end


end


%%%%%  POCETAK RACUNANJA PRAVOG UBRZANJA

cl=1;
   for cl=1:400
    cy=cos(YAW(cl)*pi/180);
    cp=cos(PITCH(cl)*pi/180);
    cr=cos(ROLL(cl)*pi/180);
    sy=sin(YAW(cl)*pi/180);
    sp=sin(PITCH(cl)*pi/180);
    sr=sin(ROLL(cl)*pi/180);

    R=[cp*cy (cy*sr*sp)-(cr*sy) (sr*sy)+(cr*cy*sp); cp*sy
(cr*cy)+(sr*sp*sy) (cr*sp*sy)-(cy*sr); -sp cp*sr cr*cp];
    gimu=R'*[0;0;1];

    aimu=[accx(cl);accy(cl);accz(cl)];
    ap=gimu-aimu;
    mag_r(cl)=sqrt((ap(1))^2+(ap(2))^2+(ap(3))^2);
   end
%%%%%%%% KRAJ RACUNANJA PRAVOG UBRZANJA

%% uređivanje intervala pokreta
ploting_ab1=ploting_ab;
ploting_ap1=ploting_ap;

i=0;
```

```
for i=1:400

    if i>1 && ploting_ab(i-1)==0 && ploting_ab(i)==1
        ploting_ab1(i:i+39)=0;
    end

     if i>1 && ploting_ap(i-1)==0 && ploting_ap(i)==1
        ploting_ap1(i:i+39)=0;
    end

end

ploting_ab=ploting_ab1;
ploting_ap=ploting_ap1;

%%
figure(1)
subplot(1,2,1)
plot(mag)
grid on
hold on
subplot(1,2,1)
plot(ploting_ap1+1,'k')
hold on
grid on
subplot(1,2,1)
plot(ploting_ab1+1,'c')
hold on
grid on
title('Rezultantno ubrzanje','FontSize',14)
xlabel('Broj podataka','FontSize',12)
ylabel('g [x9.81 m/s^2]','FontSize',12)
legend('a_R','Pisanje','Brisanje')
subplot(1,2,2)
plot(gyroz)
grid on
hold on
subplot(1,2,2)
plot(ploting_ap1*10,'k')
hold on
grid on
subplot(1,2,2)
plot(ploting_ab1*10,'c')
hold on
grid on
title('\omega_z','FontSize',14)
xlabel('Broj podataka','FontSize',12)
ylabel('\omega [°/s]','FontSize',12)
legend('\omega_z','Pisanje','Brisanje')

save('test')
return
```

I.6. **Provjera_pokreta_fusion.m**

```
clearvars -except s11 cam
t=1;
i=1;
raspon=1200;
a=1:1:raspon;
flushinput(s11);
pause(0.5);
s11.bytesavailable;
yaw=zeros(1,raspon);
pitch=zeros(1,raspon);
roll=zeros(1,raspon);

acc_x=zeros(1,raspon);
acc_y=zeros(1,raspon);
acc_z=zeros(1,raspon);
gyro_x=zeros(1,raspon);
gyro_y=zeros(1,raspon);
gyro_z=zeros(1,raspon);

ploting_h=zeros(1,raspon/3);
ploting_hn=zeros(1,raspon/3);
ploting_s=zeros(1,raspon/3);
ploting_sn=zeros(1,raspon/3);
ploting_as=zeros(1,raspon/3);
ploting_ah=zeros(1,raspon/3);
ploting_ar=zeros(1,raspon/3);
ploting_ap=zeros(1,raspon/3);
ploting_ab=zeros(1,raspon/3);
sr_vr=1;
max_vr=0;
ham_p=0;
scr_p=0;
unk=0;
zbroj_h=0;
zbroj_s=0;
zbroj_r=0;
zbroj_p=0;
zbroj_b=0;
max_roll=0;

accx=zeros(1,raspon/3);
accy=zeros(1,raspon/3);
accz=zeros(1,raspon/3);
gyrox=zeros(1,raspon/3);
gyroy=zeros(1,raspon/3);
gyroz=zeros(1,raspon/3);
YAW=zeros(1,raspon/3);
PITCH=zeros(1,raspon/3);
ROLL=zeros(1,raspon/3);
arx=zeros(1,raspon/3);
ary=zeros(1,raspon/3);
arz=zeros(1,raspon/3);
mag=zeros(1,raspon/3);
mag_r=zeros(1,raspon/3);
cr=zeros(1,raspon/3);
cp=zeros(1,raspon/3);
cy=zeros(1,raspon/3);
```

```
sy=zeros(1,raspon/3);
sp=zeros(1,raspon/3);
sr=zeros(1,raspon/3);
cl=1;
vrijemee=0;
vrijemeuk=0;
stanka=0;
in_1=fscanf(s11);
 pause(0.025)
 X_client_send_rw('1');

vrijeme_start=clock;
while(i<raspon)

     tic
    in=fscanf(s11);
    data=textscan(in,'%s%f%f%f','delimiter',',');
    [type,jen,dva,tri]=deal(data{:});

    if strcmp('#A-C',type(1,1))==1

        acc_x(i)=jen/61.35;
        acc_x(i+1)=jen/61.35;
        acc_x(i+2)=acc_x(i+1);
        if i>3
           razlika=abs(abs(acc_x(i))-abs(acc_x(i-1)));
           if razlika<0.02
               acc_x(i)=acc_x(i-1);
               acc_x(i+1)=acc_x(i-1);
               acc_x(i+2)=acc_x(i-1);
           end
        end

        acc_y(i)=dva/62.775;
        acc_y(i+1)=dva/62.775;
        acc_y(i+2)=acc_y(i+1);
        if i>3
           razlika=abs(abs(acc_y(i))-abs(acc_y(i-1)));
           if razlika<0.02
               acc_y(i)=acc_y(i-1);
               acc_y(i+1)=acc_y(i-1);
               acc_y(i+2)=acc_y(i-1);
           end
        end

        acc_z(i)=tri/62.635;
        acc_z(i+1)=tri/62.635;
        acc_z(i+2)=acc_z(i+1);
        if i>3
           razlika=abs(abs(acc_z(i))-abs(acc_z(i-1)));
           if razlika<0.02
               acc_z(i)=acc_z(i-1);
               acc_z(i+1)=acc_z(i-1);
               acc_z(i+2)=acc_z(i-1);
           end
        end

    elseif strcmp('#M-C',type(1,1))==1

          continue
```

```
    elseif strcmp('#G-C',type(1,1))==1

        if i>3
          gyro_x(i-1)=jen/14.735;
          gyro_x(i+1)=jen/14.735;
          gyro_x(i)=jen/14.735;
          dif_g=abs(abs(gyro_x(i))-abs(gyro_x(i-2)));
          if dif_g<0.3
              gyro_x(i-1)=gyro_x(i-2);
              gyro_x(i)=gyro_x(i-2);
              gyro_x(i+1)=gyro_x(i-2);
          end

          gyro_y(i-1)=dva/14.735;
          gyro_y(i+1)=dva/14.735;
          gyro_y(i)=jen/14.735;
          dif_g=abs(abs(gyro_y(i))-abs(gyro_y(i-2)));
          if dif_g<0.03
              gyro_y(i-1)=gyro_y(i-2);
              gyro_y(i)=gyro_y(i-2);
              gyro_y(i+1)=gyro_y(i-2);
          end

          gyro_z(i-1)=tri/14.735;
          gyro_z(i+1)=tri/14.735;
          gyro_z(i)=tri/14.735;
          dif_g=abs(abs(gyro_z(i))-abs(gyro_z(i-2)));
          if dif_g<0.03
              gyro_z(i-1)=gyro_z(i-2);
              gyro_z(i)=gyro_z(i-2);
              gyro_z(i+1)=gyro_z(i-2);
          end
        end

     elseif strcmp('#YPR',type(1,1))==1

         if i>4
          yaw(i-2)=jen;
          yaw(i-1)=jen;
          yaw(i)=jen;
           dif=abs(abs(yaw(i))-abs(yaw(i-3)));
           if dif<0.3
               yaw(i)=yaw(i-3);
               yaw(i-1)=yaw(i-3);
               yaw(i-2)=yaw(i-3);
           end
         end

         if i>4
             pitch(i-2)=dva;
             pitch(i-1)=dva;
             pitch(i)=dva;
           dif=abs(abs(pitch(i))-abs(pitch(i-3)));
           if dif<0.3
               pitch(i)=pitch(i-3);
               pitch(i-1)=pitch(i-3);
               pitch(i-2)=pitch(i-3);
           end
         end
         if i>4
```

```
            roll(i-2)=tri;
            roll(i-1)=tri;
            roll(i)=tri;
          dif=abs(abs(roll(i))-abs(roll(i-3)));
          if dif<0.3
              roll(i)=roll(i-3);
              roll(i-1)=roll(i-3);
              roll(i-2)=roll(i-3);
          end
       end
      else
      disp('Citanje nije uspjelo')
  end


  %%% PROVJERA KLASE

  %%%ciscenje_podataka
  cl=1;
  for j=4:3:i

  accx(cl)=acc_x(j);
  accy(cl)=acc_y(j);
  accz(cl)=acc_z(j);
  gyrox(cl)=gyro_x(j);
  gyroy(cl)=gyro_y(j);
  gyroz(cl)=gyro_z(j);
  YAW(cl)=yaw(j);
  PITCH(cl)=pitch(j);
  ROLL(cl)=roll(j);

  mag(cl)=sqrt(accx(cl)^2+accy(cl)^2+accz(cl)^2);
   cl=cl+1;
  end

  %%%%%%%%
  s11.bytesavailable;
  i=i+1;
  vrijeme=toc;
  vrijemee=vrijemee+vrijeme;
  if mod(i,3)==0
      stanka=0.020-vrijemee;
       vrijemeuk=vrijemeuk+vrijemee+stanka;
      pause(stanka)
      vrijemee=0;
  end


end

X_client_send_rw('0');


pause(1)

%%%%% PREPOZNAVANJE POKRETA
br_t_put=0;
fin=fopen('Putanja.txt');
```

```
dataput=textscan(fin,'%s %f %f %f %f %f %f %f %f %f %s %f %f
%f','delimiter',',');
br_t_put=size(dataput{1,2},1);
fclose(fin);


%% izlučivanje vremena točaka putanje
i=0;
for i=1:br_t_put
    vrijeme_sve{i,1}=textscan(dataput{1,11}{i,1},'%f %s');
end




i=0;
j=0;
Odredi_pomak();
for i=1:raspon

    if i>300 && mod(i,3)==0

        tIMU=i*20/3;
         prvi=0;
         for j=2:br_t_put
             if (vrijeme_sve{j,1}{1,1}+t_pomak)>tIMU && prvi==0
                 zdist=dataput{1,12}(j-1);
                 if zdist<0
                     disp('z kriv')
                 end
                 prvi=1;
             end
         end
         j=0;

         n=i/3-50;

        max_m_yaw=max(YAW(n:i/3-1));
        iqr_gy=iqr(gyroy(n:i/3-1));
        rms_ar=rms(mag(n:i/3-1));
          max_gz=max(gyroz(n:i/3-1));
          rms_yaw=rms(YAW(n:i/3-1));
          sr_vr_ax=mean(accx(n:i/3-1));
          sr_vr_yaw=mean(YAW(n:i/3-1));
          sr_vr_ay=mean(accy(n:i/3-1));
          std_gz=std(gyroz(n:i/3-1));
          rms_accy=rms(accy(n:i/3-1));
          rms_gz=rms(gyroz(n:i/3-1));
          sorted_s=sort(accy(n:i/3-1));
         ay_dif=sorted_s(1,50)-sorted_s(1,1);

            if  sr_vr_ay>0.3 && sr_vr_ay<0.72 && std_gz>14 && std_gz<115 &&
rms_gz>16 && rms_gz<115 && sr_vr_ax>0.1 && sr_vr_ax<0.7 && ay_dif>0.8 &&
ay_dif<3.5 && zdist>24 && zdist<32
                zbroj_b=zbroj_b+1;
                zbroj_s=0;
                zbroj_r=0;
                zbroj_p=0;
                zbroj_h=0;
```

```
                if zbroj_b>40

                      ploting_ab(n-40:i/3-1)=1;
                end

             elseif   sr_vr_ay>0.75 && sr_vr_ay<1.05 && std_gz>4 &&
std_gz<14 && rms_gz>4 && rms_gz<15
                zbroj_p=zbroj_p+1;
                zbroj_s=0;
                zbroj_r=0;
                zbroj_b=0;
                zbroj_h=0;

                if zbroj_p>40

                      ploting_ap(n-40:i/3-1)=1;
                end
            else
                  unk=unk+1;
                  zbroj_s=0;
                  zbroj_h=0;
                  zbroj_r=0;
                  zbroj_p=0;
                  zbroj_b=0;
                  ploting_ab(i/3-40:i/3-1)=0;
                  ploting_ap(i/3-40:i/3-1)=0;
            end


    end
end
%%%%% KRAJ PREPOZNAVANJA POKRETA

%%%%%  POCETAK RACUNANJA PRAVOG UBRZANJA

cl=1;
   for cl=1:400
    cy=cos(YAW(cl)*pi/180);
    cp=cos(PITCH(cl)*pi/180);
    cr=cos(ROLL(cl)*pi/180);
    sy=sin(YAW(cl)*pi/180);
    sp=sin(PITCH(cl)*pi/180);
    sr=sin(ROLL(cl)*pi/180);

    R=[cp*cy (cy*sr*sp)-(cr*sy) (sr*sy)+(cr*cy*sp); cp*sy
(cr*cy)+(sr*sp*sy) (cr*sp*sy)-(cy*sr); -sp cp*sr cr*cp];
    gimu=R'*[0;0;1];

    aimu=[accx(cl);accy(cl);accz(cl)];
    ap=gimu-aimu;
    mag_r(cl)=sqrt((ap(1))^2+(ap(2))^2+(ap(3))^2);
   end
%%%%%%%% KRAJ RACUNANJA PRAVOG UBRZANJA

%% uređivanje intervala pokreta
ploting_ab1=ploting_ab;
ploting_ap1=ploting_ap;
```

```
i=0;
for i=1:400

    if i>1 && ploting_ab(i-1)==0 && ploting_ab(i)==1
        ploting_ab1(i:i+39)=0;
    end

     if i>1 && ploting_ap(i-1)==0 && ploting_ap(i)==1
        ploting_ap1(i:i+39)=0;
    end

end

ploting_ab=ploting_ab1;
ploting_ap=ploting_ap1;


return
```

I.7. Provjera_fusion.m

```
lobal message_c
message_c='';

%% uspostavljanje veze
ip='127.0.0.1';port=30006;
X_client_open_rw(ip,port,100000);
fprintf('Veza uspješno uspostavljena\n')

%% šalji znak za snimanje
X_client_send_rw('1');
pause(0.1);
%% čekaj odgovor (da c++ pošalje znak da je krenuo u snimanje) te tada
odradi snimanje
while true

        X_client_rcv_rw();
        if size(message_c,2)==6
             if strcmp(message_c(1:5),'start')
                   Provjera_pokreta_fusion();
                   pause(0.1);
                   break
             end
        end
end
```

I.8. Odredi_pomak.m

```
fin=fopen('Pocetak.txt');
pocetak_cpp=textscan(fin,'%f %f','delimiter',',');
fclose(fin);
t_s_cpp=pocetak_cpp{1};
t_ms_cpp=pocetak_cpp{2};

t_s_mat=floor(vrijeme_start(6));

if t_s_mat==t_s_cpp
```

```
    t_ms_mat=(vrijeme_start(6)-t_s_mat)*1000;
    t_pomak=t_ms_cpp-t_ms_mat;

elseif t_s_cpp>t_s_mat
    t_ms_mat=(vrijeme_start(6)-t_s_mat)*1000;
    t_ms_cpp=t_ms_cpp+1000;
    t_pomak=t_ms_cpp-t_ms_mat;
end
```

I.9. Snimanje_fusion.m

```
global message_c
message_c='';

%% uspostavljanje veze
ip='127.0.0.1';port=30006;
X_client_open_rw(ip,port,100000);
fprintf('Veza uspješno uspostavljena\n')

%% šalji znak za snimanje
X_client_send_rw('1');
pause(0.1);
%% čekaj odgovor (da c++ pošalje znak da je krenuo u snimanje) te tada
odradi snimanje
while true

        X_client_rcv_rw();
        if size(message_c,2)==6
             if strcmp(message_c(1:5),'start')
                   Provjera_pokreta_fusion();
                   pause(0.1);
                   break
             end
        end
end

%% Odredi interval pokreta te usrednji ubrzanja za točke putanje
Odredi_interval_pokreta();
if start>0
Odredi_putanju_fusion();
end
save('test_fusion')
```

I.10 Odredi_interval_pokreta.m

```
i=0;
start=0;
kraj=0;
startp=0;
krajp=0;
startacc=0;
krajacc=0;
br_intervala=0;
ploting_ab(401)=ploting_ab(400);
for i=1:400

   if ploting_ab(i)==0 && ploting_ab(i+1)==1
       br_intervala=br_intervala+1;
```

```
        start(br_intervala)=i+1
    elseif ploting_ab(i)==1 && ploting_ab(i+1)==0
       kraj(br_intervala)=i

    elseif i==400 && ploting_ab(i)==1
        kraj(br_intervala)=i
    end

end
```

I.11. **Odredi_putanju_fusion.m**

```
br_t_put=0;
i=0;
j=0;
interval=0;
% učitaj snimljene točke putanje
fin=fopen('Putanja.txt');
dataput=textscan(fin,'%s %f %f %f %f %f %f %f %f %f %s %f %f
%f','delimiter',',');
br_t_put=size(dataput{1,2},1);
fclose(fin);
fout=fopen('Putanja2.txt','wt');

%% određivanje početnog i krajnjeg vremena nove putanje

for interval=1:size(start,2)
t_np_start(interval)=(start(interval)*20)-t_pomak;
t_np_kraj(interval)=(kraj(interval)*20)-t_pomak;
end

%% izlučivanje vremena točaka putanje
i=0;
for i=1:br_t_put
    vrijeme_sve{i,1}=textscan(dataput{1,11}{i,1},'%f %s');
end

%% Određivanje početnih i krajnjih točaka putanje generirane
klasifikacijskim algoritmom
interval=0;
j=0;
i=0;
z=0;
for interval=1:size(start,2)
   prvi=1;
   for i=1:br_t_put-1
      if vrijeme_sve{i,1}{1,1}> t_np_start(interval) && prvi==1
          poc_IMU(interval)=i;
          poc_np(interval)=i;
          prvi=0;
      elseif vrijeme_sve{i,1}{1,1}< t_np_kraj(interval) &&
vrijeme_sve{i+1,1}{1,1}> t_np_kraj(interval)
          kraj_IMU(interval)=i;
          kraj_np(interval)=i;
      end
   end
end

%% Određivanje točnog početka i kraja brisanja
```

```
for interval=1:size(start,2)
     E1(1)=0;
     E1(2)=0;
     E2(1)=10;
     E2(2)=10;
     E3(1)=10;
      E3(2)=10;
     E4(1)=0;
     E4(2)=0;
     E1(3)=sqrt((dataput{1,2}(poc_IMU(interval)+1)-
dataput{1,2}(poc_IMU(interval)))^2+(dataput{1,3}(poc_IMU(interval)+1)-
dataput{1,3}(poc_IMU(interval)))^2+(dataput{1,4}(poc_IMU(interval)+1)-
dataput{1,4}(poc_IMU(interval)))^2);
     E2(3)=sqrt((dataput{1,2}(poc_IMU(interval))-
dataput{1,2}(poc_IMU(interval)-1))^2+(dataput{1,3}(poc_IMU(interval))-
dataput{1,3}(poc_IMU(interval)-1))^2+(dataput{1,4}(poc_IMU(interval))-
dataput{1,4}(poc_IMU(interval)-1))^2);

     E3(3)=sqrt((dataput{1,2}(kraj_IMU(interval)+1)-
dataput{1,2}(kraj_IMU(interval)))^2+(dataput{1,3}(kraj_IMU(interval)+1)-
dataput{1,3}(kraj_IMU(interval)))^2+(dataput{1,4}(kraj_IMU(interval)+1)-
dataput{1,4}(kraj_IMU(interval)))^2);
     E4(3)=sqrt((dataput{1,2}(kraj_IMU(interval))-
dataput{1,2}(kraj_IMU(interval)-1))^2+(dataput{1,3}(kraj_IMU(interval))-
dataput{1,3}(kraj_IMU(interval)-1))^2+(dataput{1,4}(kraj_IMU(interval))-
dataput{1,4}(kraj_IMU(interval)-1))^2);
     tt1=poc_IMU(interval);
     tt2=kraj_IMU(interval);
     j=3;
    if E1(j)<0.002
        while (E1(j)<0.002 || E1(j-1)<0.002 || E1(j-2)<0.002) &&
dataput{1,12}(tt1+1)>24 && dataput{1,12}(tt1+1)<32
            j=j+1;
            tt1=tt1+1;
            vrijemepon=vrijeme_sve{tt1,1}{1,1};
            if vrijemepon~=vrijeme_sve{tt1+1,1}{1,1}
             E1(j)=sqrt((dataput{1,2}(tt1+1)-
dataput{1,2}(tt1))^2+(dataput{1,3}(tt1+1)-
dataput{1,3}(tt1))^2+(dataput{1,4}(tt1+1)-dataput{1,4}(tt1))^2);
            else
                E1(j)=E1(j-1);
            end
        end
        poc_np(interval)=tt1-2;

    elseif E2(j)>0.002

        if dataput{1,12}(tt1)<24 || dataput{1,12}(tt1)>32

            while dataput{1,12}(tt1)<24 || dataput{1,12}(tt1)>32
            j=j+1;
            tt1=tt1+1;
            E2(j)=sqrt((dataput{1,2}(tt1+1)-
dataput{1,2}(tt1))^2+(dataput{1,3}(tt1+1)-
dataput{1,3}(tt1))^2+(dataput{1,4}(tt1+1)-dataput{1,4}(tt1))^2);
            poc_np(interval)=tt1;
            end
        end
```

```
        while (E2(j)>0.002 || E2(j-1)>0.002  || E2(j-2)>0.002) &&
dataput{1,12}(tt1-1)>24 && dataput{1,12}(tt1-1)<32
            j=j+1;
            tt1=tt1-1;
            vrijemepon=vrijeme_sve{tt1,1}{1,1};
            if vrijemepon~=vrijeme_sve{tt1-1,1}{1,1}
             E2(j)=sqrt((dataput{1,2}(tt1)-dataput{1,2}(tt1-
1))^2+(dataput{1,3}(tt1)-dataput{1,3}(tt1-1))^2+(dataput{1,4}(tt1)-
dataput{1,4}(tt1-1))^2);
            else
                E2(j)=E2(j-1);
            end
        end
        poc_np(interval)=tt1+2;
        if  dataput{1,12}(tt1-1)<24 || dataput{1,12}(tt1-1)>32
            poc_np(interval)=tt1;
        end

    end
    j=3;
    if E3(j)>0.002

        if dataput{1,12}(tt2)<24 || dataput{1,12}(tt2)>32

            while dataput{1,12}(tt2)<24 || dataput{1,12}(tt2)>32
            j=j+1;
            tt2=tt2-1;
            E3(j)=sqrt((dataput{1,2}(tt2)-dataput{1,2}(tt2-
1))^2+(dataput{1,3}(tt2)-dataput{1,3}(tt2-1))^2+(dataput{1,4}(tt2)-
dataput{1,4}(tt2-1))^2);
            kraj_np(interval)=tt2;
            end
        end

        while (E3(j)>0.002 || E3(j-1)>0.002 || E3(j-2)>0.002) &&
dataput{1,12}(tt2-1)>24 && dataput{1,12}(tt2-1)<32
           j=j+1;
            tt2=tt2+1;
             vrijemepon=vrijeme_sve{tt2,1}{1,1};
            if vrijemepon~=vrijeme_sve{tt2+1,1}{1,1}
             E3(j)=sqrt((dataput{1,2}(tt2+1)-
dataput{1,2}(tt2))^2+(dataput{1,3}(tt2+1)-
dataput{1,3}(tt2))^2+(dataput{1,4}(tt2+1)-dataput{1,4}(tt2))^2);
            else
                E3(j)=E3(j-1);
            end
        end
        kraj_np(interval)=tt2-2;


    elseif E4(j)<0.002
        while (E4(j)<0.002 || E4(j-1)<0.002 || E4(j-2)<0.002) &&
dataput{1,12}(tt2)>24 && dataput{1,12}(tt2)<32
            j=j+1;
            tt2=tt2-1;
            vrijemepon=vrijeme_sve{tt2,1}{1,1};
            if vrijemepon~=vrijeme_sve{tt2-1,1}{1,1}
             E4(j)=sqrt((dataput{1,2}(tt2)-dataput{1,2}(tt2-
1))^2+(dataput{1,3}(tt2)-dataput{1,3}(tt2-1))^2+(dataput{1,4}(tt2)-
dataput{1,4}(tt2-1))^2);
```

```
            else
                E4(j)=E4(j-1);
            end
        end
        kraj_np(interval)=tt2+2;
    end

end

%% određivanje nove putanje i računanje ubrzanja
interval=0;
j=0;
i=0;
z=0;
for interval=1:size(poc_np,2)
   for i=poc_np(interval):kraj_np(interval)
          j=j+1;
          t_acc(j)=vrijeme_sve{i,1}{1,1};
          for z=1:9
          newput{1,z}(j,1)=dataput{1,z}(i);
          end
   end
end

i=0;
for i=1:size(t_acc,2)

    t_put_in_imu(i)=(t_acc(i)+t_pomak)/20;
    if floor(t_put_in_imu(i))==t_put_in_imu(i)
        newput{1,8}(i,1)=mag_r(t_put_in_imu(i))*9.80665;
    else

newput{1,8}(i,1)=(mag_r(floor(t_put_in_imu(i)))+mag_r(ceil(t_put_in_imu(i))
))/2*9.80665;
    end
    newput{1,10}{i,1}='0)';
end

 zt=size(t_acc,2);

 %% dodavanje zadnje točke za prekid gibanja robota
j=0;
for j=1:10
    if j==1
        newput{1,1}{zt+1,1}='(0';

    else
     newput{1,j}(zt+1,1)=newput{1,j}(zt,1);
    end
end

%% ispis nove putanje u datoteku
i=0;
for i=1:zt+1
    fprintf(fout,'%s, %f, %f, %f, %f, %f, %f, %f, %f,
%s\n\n',newput{1,1}{i,1}(1:2),newput{1,2}(i),newput{1,3}(i),newput{1,4}(i),
newput{1,5}(i),newput{1,6}(i),newput{1,7}(i),newput{1,8}(i),newput{1,9}(i),
newput{1,10}{i,1}(1:2));
end
fclose(fout);
```

I.12. Detekcija_pisanja.m

```
pisanje=0;
while pisanje~=1
    Online_provjera_pokreta();
end
myicon = imread('ur5.png');
h = msgbox({'Pisanje otkriveno!' 'Pritisni Enter dva puta da robot pobriše
ploču :)'},'Brisanje ploče','custom',myicon);
pause
Klasifikacija_povrsine();
```

I.13. Klasifikacija_povrsine.m

```
Obrada_slike();
%%zapis u txt datoteku
fout=fopen('Povrsina_new.txt','wt');
fprintf(fout,'%f\n %f\n %f\n %f',x_koo_01,y_koo_01,x_koo_02,y_koo_02);
fclose(fout);
pause(0.5)
%%
if POV_new>25000 && POV_new<40000 && dim1<dim2
    %%koristi gibanje 2
    X_client_send_rw('P2');
    pause(0.1);
elseif POV_new>1000 && POV_new<15000 && dim2<dim1
    %%koristi gibanje 1
    X_client_send_rw('P1');
elseif POV_new>80000 && POV_new<115000
    X_client_send_rw('P3');
else
    disp ('losa povrsina')
end
```

I.14. X_client_open_rw.m [15]

```
% CLIENT connect to a server and read a message
%
% Usage - message = client(host, port, number_of_retries)
function client_open_rw(host1, port1, number_of_retries1)

    import java.net.Socket
    %import java.net.*
    import java.io.*

    global input_socket1
    global out1
    retry1        = 0;
    input_socket1 = [];
    message1      = [];

    while true

        retry1 = retry1 + 1;
        if ((number_of_retries1 > 0) && (retry1 > number_of_retries1))
            fprintf(1, 'Too many retries\n');
```

```
            break;
        end

        try
            fprintf(1, 'Retry %d connecting to %s:%d\n', ...
                    retry1, host1, port1);

            % throws if unable to connect
            input_socket1 = Socket(host1, port1);
            out1 = PrintWriter(input_socket1.getOutputStream,true);
            fprintf(1, '%s\n',host1)
            break
            pause(10000000)

            break

        catch
            if ~isempty(input_socket1)
                input_socket1.close;
            end

            % pause before retrying
            pause(0.05);
        end
    end
end
```

I.15. X_client_rcv_rw.m [15]

```
% CLIENT connect to a server and read a message
%
% Usage - message = client(host, port, number_of_retries)
function X_client_rcv_rw

    import java.net.Socket
    import java.io.*

    global input_socket1
    global message_c message_p
    global in
    in=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];


            % get a buffered data input stream from the socket
            input_stream1   = input_socket1.getInputStream;
            d_input_stream1 = DataInputStream(input_stream1);

            % read data from the socket - wait a short time first
            bytes_available = input_stream1.available;
            j=0;
            k=0;
            jj=0;
            rb=0; %redni broj poruke
            reset=0; %resetiranje na novi ulazni broj
            pom=0;

            message_p = zeros(1, bytes_available, 'uint8'); %init poruke
            message_c = zeros(1, bytes_available, 'uint8'); %init poruke
```

```
            for ii = 1:bytes_available

                message_c(ii) = d_input_stream1.readByte;  %ucitaj bajt po
bajt u petlji
                if message_c(ii)=='@' %KRAJ PODATKA - DELIMITER
                   j=0;
                   pom=jj;
                   jj=0;
                   rb=rb+1;
                    message_p = char(message_p);
                    message_p(1:pom);
                    x=str2double(message_p(1:pom));
                    in(rb)=x;
                end

                %pisanje poruke u message_p koji treba zapisati svaki broj
                %odvojeno
                if j>0
                    jj=jj+1;
                    message_p(jj) = message_c(ii);
                end
                if message_c(ii)=='P' %POCETAK PORUKE
                   j=1;
                end

                if message_c(ii)=='@' %KRAJ PODATKA - DELIMITER
                   j=1;
                end

            end
            in;
            message_c = char(message_c);
            fprintf(message_c, '\n');

end
```

II. Programski kod C++

II.1. **matlab_server.cpp** – za komunikaciju s MATLABOM

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <windows.h>
// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30006"
#define DEFAULT_BUFFER_LENGTH     512

using namespace std;
extern char* sendmat;
int startaj, odabir_put;
 int prekid;
 extern int poc;

void matlab_server(void *P) {
	WSADATA wsaData;

	// Initialize Winsock
	int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (iResult != 0) {
		printf("matlab_server: WSAStartup failed: %d\n", iResult);
		return;
	}

	struct addrinfo     *result = NULL,
		hints;

	ZeroMemory(&hints, sizeof(hints));
	hints.ai_family = AF_INET;        // Internet address family is unspecified so
that either an IPv6 or IPv4 address can be returned
	hints.ai_socktype = SOCK_STREAM;  // Requests the socket type to be a stream
socket for the TCP protocol
	hints.ai_protocol = IPPROTO_TCP;
	hints.ai_flags = AI_PASSIVE;

	// Resolve the local address and port to be used by the server
	iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
	if (iResult != 0) {
		printf("matlab_server: getaddrinfo failed: %d\n", iResult);
		WSACleanup();
		return;
	}

	SOCKET ListenSocket = INVALID_SOCKET;
```

```
	// Create a SOCKET for the server to listen for client connections
	ListenSocket = socket(result->ai_family, result->ai_socktype, result-
>ai_protocol);


	if (ListenSocket == INVALID_SOCKET) {
		printf("matlab_server: Error at socket(): %d\n", WSAGetLastError());
		freeaddrinfo(result);
		WSACleanup();
		return;
	}

	// Setup the TCP listening socket
	iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);

	if (iResult == SOCKET_ERROR) {
		printf("matlab_server: bind failed: %d", WSAGetLastError());
		freeaddrinfo(result);
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	freeaddrinfo(result);

	// To listen on a socket
	if (listen(ListenSocket, SOMAXCONN) == SOCKET_ERROR) {
		printf("matlab_server: listen failed: %d\n", WSAGetLastError());
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	SOCKET ClientSocket;

	ClientSocket = INVALID_SOCKET;
	// Accept a client socket
	ClientSocket = accept(ListenSocket, NULL, NULL);

	if (ClientSocket == INVALID_SOCKET) {
		printf("matlab_server: accept failed: %d\n", WSAGetLastError());
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}
	char recvbuf[DEFAULT_BUFFER_LENGTH];
	int iSendResult;
	prekid = 0;
	startaj = 0;
	do {    ///primanje dok klijent ne zatvori vezu
		iResult = recv(ClientSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);

		if (iResult > 0) {
			char msgmat[DEFAULT_BUFFER_LENGTH];
			memset(&msgmat, 0, sizeof(msgmat));
			strncpy_s(msgmat, recvbuf, iResult);

			if (msgmat[0] == '1') {

				startaj = 1;
```

```
                        iSendResult = send(ClientSocket, sendmat,
(int)strlen(sendmat), 0);

                        printf("data sent\n");
                    }
                    if (poc == 1) {
                        poc = 0;
                        iSendResult = send(ClientSocket, sendmat,
(int)strlen(sendmat), 0);
                    }
                    if (msgmat[0] == '0') {
                        prekid = 1;
                        iSendResult = send(ClientSocket, sendmat,
(int)strlen(sendmat), 0);
                        printf("komunikacija prekinuta\n");
                    }
                    if (msgmat[0] == 'P'&& msgmat[1] =='1' ) {

                        odabir_put= 1;

                        printf("\nPutanja 1\n");
                    }

                    if (msgmat[0] == 'P'&& msgmat[1] == '2') {

                        odabir_put = 2;

                        printf("\nPutanja 2\n");
                    }

                    if (msgmat[0] == 'P'&& msgmat[1] == '3') {

                        odabir_put = 3;

                        printf("\nPutanja 2\n");
                    }

                    if (iSendResult == SOCKET_ERROR) {
                        printf("matlab_server: send failed: %d\n",
WSAGetLastError());
                        closesocket(ClientSocket);
                        WSACleanup();
                        return;
                    }
                }
                else if (iResult == 0) {
                    printf("matlab_server: Connection closed\n");
                }
                else {
                    printf("matlab_server: recv failed: %d\n", WSAGetLastError());
                    closesocket(ClientSocket);
                    WSACleanup();
                    return;
                }
        } while (iResult > 0);

        // Free the resouces
        closesocket(ListenSocket);
        WSACleanup();

        getchar();
        return; }
```

II.2. **Odnos ploca-spuzva** – dodano unutar math.cpp skripte u programu iz [12]

```
bool snimanje(FALSE);
bool ponovno(FALSE);
bool koma(FALSE);
bool klasa(FALSE);
double X_old, Y_old, Z_old, Rx_old, Ry_old, Rz_old,X_new,Y_new;
Matrix4d T_Robot_CS, T_Polaris_M1_CS;
Matrix4d T_M1_P;
extern double M3_x, M3_y, M3_z, M3_qw, M3_qx, M3_qy, M3_qz;
Matrix3d R_M3;
Matrix4d T_TTO_M1, T_TTO_M2, T_T_old, T_T_new, T_T_P;
Matrix4d T_M22;
Matrix4d T_M3_P, T_M3, T_MP_old, T_MP_new;
Matrix4d T_M2inM3;
Matrix4d T_MP_oldinM1, T_MP_newinM1;
Matrix4d T_PinR_old, T_PinR_new;
double aa, vv, rr;
extern bool M3_OOV;
stringstream sendbufss;
string sendbufs, inbufs;
char *sendbuf;
char inbuff;
stringstream putanjabufss;
string putanjabufs;
string delimiterr = ",";
char *putanjabuf;
fstream Putanja, Povrsina_new,Povrsina_old, Putanja_old,Putanja_new;
extern int startaj, odabir_put;
string line;
char *sendmat;
stringstream sendmatss;
string sendmats, numberr;
double brojuzoraka;
extern int prekid;
int pocetak, poc,brstr,iii;
int start_p, inicijalizacija, ii, j;
double  vrijeme;
extern DWORD T_p_uk;
DWORD dw1, dw2;
double zdistance, yaww, rolll, pitchh,X_T,Y_T;
double M3_old[16];
double T_old[4], T_new[4];
double Target_old[9];
size_t poss;

Odnos:
	do {
		tracking = TRUE;
		inicijalizacija = 0;
		T_TTO_M2 << 1, 0, 0, 50 + x_offset, 0, 1, 0, 0 + y_offset, 0, 0, 1, 0 +
z_offset, 0, 0, 0, 1;

		R_rotM2 << cos(pitch)*cos(yaw), cos(roll)*sin(yaw) +
sin(roll)*sin(pitch)*cos(yaw), sin(roll)*sin(yaw) - cos(roll)*sin(pitch)*cos(yaw), 0,
-cos(pitch)*sin(yaw), cos(roll)*cos(yaw) - sin(roll)*sin(pitch)*sin(yaw),
sin(roll)*cos(yaw) + cos(roll)*sin(pitch)*sin(yaw), 0, sin(pitch), -
sin(roll)*cos(pitch), cos(roll)*cos(pitch), 0, 0, 0, 0, 1;
		M2_q.w() = M2_qw;
		M2_q.x() = M2_qx;
		M2_q.y() = M2_qy;
```

```
                M2_q.z() = M2_qz;
                R_M2 = M2_q.normalized().toRotationMatrix(); //Quaternion to DCM
                T_M2_P << R_M2(0, 0), R_M2(0, 1), R_M2(0, 2), M2_x, R_M2(1, 0), R_M2(1,
1), R_M2(1, 2), M2_y, R_M2(2, 0), R_M2(2, 1), R_M2(2, 2), M2_z, 0, 0, 0, 1;
                T_M22 = T_M2_P*T_TTO_M2;

                M3_q.w() = M3_qw;
                M3_q.x() = M3_qx;
                M3_q.y() = M3_qy;
                M3_q.z() = M3_qz;
                R_M3 = M3_q.normalized().toRotationMatrix(); //Quaternion to DCM
                T_M3_P << R_M3(0, 0), R_M3(0, 1), R_M3(0, 2), M3_x, R_M3(1, 0), R_M3(1,
1), R_M3(1, 2), M3_y, R_M3(2, 0), R_M3(2, 1), R_M3(2, 2), M3_z, 0, 0, 0, 1;
                T_M3 = T_M3_P;

                T_M2inM3 = T_M3.inverse()*T_M22;

                if (M2_OOV == TRUE) { zdistance = 0;
                }
                else { zdistance = T_M2inM3(2, 3); }
                cout << "\n Z=" << zdistance << " mm \n";
                Sleep(200);

        } while (tracking==TRUE);

        goto start;
```

II.3. **Ucenje novih pokreta** – dodano unutar math.cpp skripte u programu iz [12]

```
Ucenje:
        vrijeme = 0;
        start_p = 0;
        inicijalizacija = 0;
        //cekaj na zahtjev za snimanje
        sendmatss.str("");
        sendmatss << "start@";
        sendmats = sendmatss.str();
        sendmat = (char*)sendmats.c_str();
        Putanja.open("Putanja.txt", fstream::out | fstream::trunc);
        if (Putanja.is_open()) {
                Putanja.close();
        }


        do {

        } while (startaj <1);

        startaj = 0;
        do {

        } while (startaj <1);
        startaj = 0;
        inicijalizacija = 0;
                //do petlja za snimanje
        do {
                start_p = 1;

                T_TTO_M2 << 1, 0, 0, 50 + x_offset, 0, 1, 0, 50 + y_offset, 0, 0, 1, 0 +
z_offset, 0, 0, 0, 1;
```

```
            R_rotM2 << cos(pitch)*cos(yaw), cos(roll)*sin(yaw) +
sin(roll)*sin(pitch)*cos(yaw), sin(roll)*sin(yaw) - cos(roll)*sin(pitch)*cos(yaw), 0,
-cos(pitch)*sin(yaw), cos(roll)*cos(yaw) - sin(roll)*sin(pitch)*sin(yaw),
sin(roll)*cos(yaw) + cos(roll)*sin(pitch)*sin(yaw), 0, sin(pitch), -
sin(roll)*cos(pitch), cos(roll)*cos(pitch), 0, 0, 0, 0, 1;

            M1_q.w() = M1_qw;
            M1_q.x() = M1_qx;
            M1_q.y() = M1_qy;
            M1_q.z() = M1_qz;
            R_M1 = M1_q.normalized().toRotationMatrix(); //Quaternion to DCM
            T_M1_P << R_M1(0, 0), R_M1(0, 1), R_M1(0, 2), M1_x, R_M1(1, 0), R_M1(1,
1), R_M1(1, 2), M1_y, R_M1(2, 0), R_M1(2, 1), R_M1(2, 2), M1_z, 0, 0, 0, 1;
            T_M1 = T_M1_P*T_TTO_M1;

            M2_q.w() = M2_qw;
            M2_q.x() = M2_qx;
            M2_q.y() = M2_qy;
            M2_q.z() = M2_qz;
            R_M2 = M2_q.normalized().toRotationMatrix(); //Quaternion to DCM
            T_M2_P << R_M2(0, 0), R_M2(0, 1), R_M2(0, 2), M2_x, R_M2(1, 0), R_M2(1,
1), R_M2(1, 2), M2_y, R_M2(2, 0), R_M2(2, 1), R_M2(2, 2), M2_z, 0, 0, 0, 1;
            T_M2 = T_M2_P*T_TTO_M2*R_rotM2;

            E = sqrt(pow(T_M2(0, 3) - T_M1(0, 3), 2) + pow(T_M2(1, 3) - T_M1(1, 3),
2) + pow(T_M2(2, 3) - T_M1(2, 3), 2));
            if (E > MinDist) { MinMove = FALSE; }
            else { MinMove = TRUE; }

            T_M2inM1 = T_M1.inverse()*T_M2;

            RxRyRz_in << Rx_in, Ry_in, Rz_in;
            theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
            R_Robot_in = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to
DCM

            T_Robot_actual << R_Robot_in(0, 0), R_Robot_in(0, 1), R_Robot_in(0, 2),
X_in, R_Robot_in(1, 0), R_Robot_in(1, 1), R_Robot_in(1, 2), Y_in, R_Robot_in(2, 0),
R_Robot_in(2, 1), R_Robot_in(2, 2), Z_in, 0, 0, 0, 1;

            T_Robot_target = T_Robot_actual*T_M2inM1;

            X_out = T_Robot_target(0, 3) / 1000;
            Y_out = T_Robot_target(1, 3) / 1000;
            Z_out = T_Robot_target(2, 3) / 1000;

            R_Robot_out = T_Robot_target.block<3, 3>(0, 0);
            theta_out = acos((R_Robot_out(0, 0) + R_Robot_out(1, 1) + R_Robot_out(2,
2) - 1) / 2);
            Rx_out = (R_Robot_out(2, 1) - R_Robot_out(1, 2)) / (2 *
sin(theta_out))*theta_out;
            Ry_out = (R_Robot_out(0, 2) - R_Robot_out(2, 0)) / (2 *
sin(theta_out))*theta_out;
            Rz_out = (R_Robot_out(1, 0) - R_Robot_out(0, 1)) / (2 *
sin(theta_out))*theta_out;

            /// DODATAK ZA SENSOR FUSION

            T_M22 = T_M2_P*T_TTO_M2;
            M3_q.w() = M3_qw;
            M3_q.x() = M3_qx;
            M3_q.y() = M3_qy;
```

```
			M3_q.z() = M3_qz;
			R_M3 = M3_q.normalized().toRotationMatrix(); //Quaternion to DCM
			T_M3_P << R_M3(0, 0), R_M3(0, 1), R_M3(0, 2), M3_x, R_M3(1, 0), R_M3(1,
1), R_M3(1, 2), M3_y, R_M3(2, 0), R_M3(2, 1), R_M3(2, 2), M3_z, 0, 0, 0, 1;
			T_M3 = T_M3_P;

			T_M2inM3 = T_M3.inverse()*T_M22;

			if (M2_OOV == TRUE) {
				zdistance = 0;
			}
			else { zdistance = T_M2inM3(2, 3); }

			/// KRAJ DODATKA


			if (M1_OOV == FALSE && M2_OOV == FALSE && MinMove == FALSE) {
				if (step_motion == FALSE) {


					Putanja.open("Putanja.txt", fstream::out | fstream::app);
					if (Putanja.is_open()) {
						Putanja << "(1," << X_out << "," << Y_out << "," <<
Z_out << "," << Rx_out << "," << Ry_out << "," << Rz_out << "," << a << "," << v <<
"," << r << "," << T_p_uk <<"),"<<zdistance << "," << pitchh << "," << rolll <<"\n" <<
endl;
						Putanja.close();
						Sleep(50);
					}
					else cout << "Unable to open file Putanja.txt\n\n";

					X_out1 = X_out;
					Y_out1 = Y_out;
					Z_out1 = Z_out;
					Rx_out1 = Rx_out;
					Ry_out1 = Ry_out;
					Rz_out1 = Rz_out;
				}

			}
			else if (M1_OOV == TRUE || M2_OOV == TRUE || MinMove == TRUE) {
				if (step_motion == FALSE) {

					Putanja.open("Putanja.txt", fstream::out | fstream::app);
					if (Putanja.is_open()) {
						Putanja << "(0," << X_out1 << "," << Y_out1 << ","
<< Z_out1 << "," << Rx_out1 << "," << Ry_out1 << "," << Rz_out1 << "," << a << "," <<
v << "," << r << "," << T_p_uk << ")," << zdistance << "," << pitchh << "," << rolll
<< "\n" << endl;
						Putanja.close();
						Sleep(50);
					}
					else cout << "Unable to open file Putanja.txt\n\n";

				}
				else {
					sendstr = "";
				}

			}
```

```
        } while (prekid<1);
        prekid = 0;
        start_p = 0;
        sendmatss.str("");
        sendmatss << "zatvoreno@";
        sendmats = sendmatss.str();
        sendmat = (char*)sendmats.c_str();
        Sleep(200);
        cout << "Time is " << (vrijeme) << " milliSeconds" << endl;


        goto start;
```

II.4. **Ponavljanje pokreta** – dodano unutar math.cpp skripte u programu iz [12]

```
Ponavljanje:
        ponovno = TRUE;



        program.str("");
        program << "def vitez_diplomski() :\n";
        program << "  set_standard_analog_input_domain(0, 1)\n";
        program << "  set_standard_analog_input_domain(1, 1)\n";
        program << "  set_tool_analog_input_domain(0, 1)\n";
        program << "  set_tool_analog_input_domain(1, 1)\n";
        program << "  set_analog_outputdomain(0, 0)\n";
        program << "  set_analog_outputdomain(1, 0)\n";
        program << "  set_tool_voltage(0)\n";
        program << "  set_standard_digital_input_action(0, \"default\")\n";
        program << "  set_standard_digital_input_action(1, \"default\")\n";
        program << "  set_standard_digital_input_action(2, \"default\")\n";
        program << "  set_standard_digital_input_action(3, \"default\")\n";
        program << "  set_standard_digital_input_action(4, \"default\")\n";
        program << "  set_standard_digital_input_action(5, \"default\")\n";
        program << "  set_standard_digital_input_action(6, \"default\")\n";
        program << "  set_standard_digital_input_action(7, \"default\")\n";
        program << "  set_tool_digital_input_action(0, \"default\")\n";
        program << "  set_tool_digital_input_action(1, \"default\")\n";
        program << "  set_tcp(p[" << X_TCP << ", " << Y_TCP << ", " << Z_TCP << ", " <<
Rx_TCP << ", " << Ry_TCP << ", " << Rz_TCP << "])\n";
        program << "  set_payload(0.25)\n";
        program << "  $ 1 \"BeforeStart\"\n";
        program << "  $ 2 \"Receive_Data:=[0,1,0,0,0,0,0,0,0,0,0,0]\"\n";
        program << "  global Receive_Data = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\n";
        program << "  $ 3 \"a:=0.5\"\n";
        program << "  global a = 0.5\n";
        program << "  $ 4 \"v:=0.4\"\n";
        program << "  global v = 0.4\n";
        program << "  $ 5 \"r:=0\"\n";
        program << "  global r = 0\n";
        program << "  $ 6 \"P1:=get_actual_tcp_pose()\"\n";
        program << "  global P1 = get_actual_tcp_pose()\n";
        program << "  $ 7 \"socket_open('192.168.1.1', 30000)\"\n";
        program << "  socket_open(\"192.168.1.1\", 30000)\n";
        program << "  $ 8 \"Wait: 2.0\"\n";
        program << "  sleep(2.0)\n";
        program << "  $ 27 \"Thread_2\"\n";
        program << "  thread Thread_2() :\n";
        program << "  while (True) :\n";
        program << "         socket_send_byte(1)\n";
```

```
	program << "		sleep(0.100)\n";
	program << "  end\n";
	program << "  end\n";
	program << "		threadId_Thread_2 = run Thread_2()\n";
	program << "		$ 16 \"Thread_1\"\n";
	program << "		thread Thread_1() :\n";
	program << "		while (Receive_Data[1]>0) :\n";
	program << "			Receive_Data = socket_read_ascii_float(10)\n";
	program << "			global Pointer = 0\n";
	program << "			sleep(0.01)\n";
	program << "			while (Pointer + 4<Receive_Data[0]) :\n";
	program << "				P1[Pointer] = Receive_Data[Pointer + 2]\n";
	program << "				global Pointer = Pointer + 1\n";
	program << "			end\n";
	program << "				global a = Receive_Data[8]\n";
	program << "				global v = Receive_Data[9]\n";
	program << "				global r = Receive_Data[10]\n";
	program << "		end\n";
	program << "		end\n";
	program << "				threadId_Thread_1 = run Thread_1()\n";
	program << "				while (True) :\n";
	program << "					$ 9 \"Robot Program\"\n";
	program << "					$ 12 \"Loop Receive_Data[1]>0\"\n";
	program << "					thread Thread_while_13() :\n";
	program << "					while (True) :\n";
	program << "						$ 13 \"movel(P1, a=a, v=v,
r=r)\"\n";
	program << "						movel(P1, a = a, v = v, r =
r)\n";
	program << "						$ 15 \"Wait: 0.01\"\n";
	program << "						sleep(0.01)\n";
	program << "					end\n";
	program << "					end\n";
	program << "						if (Receive_Data[1]>0) :\n";
	program << "							global thread_handler_13
= run Thread_while_13()\n";
	program << "							while (Receive_Data[1]>0)
:\n";

	program << "								sync()\n";
	program << "							end\n";

	program << "							while (Receive_Data[1]<1)
:\n";

	program << "								join
thread_handler_13 \n";

	program << "								sleep(0.5)\n";
	program << "								halt\n";
	program << "							end\n";
	program << "								kill
thread_handler_13\n";
	program << "				end\n";
	program << "		end\n";
	program << "end\n";

	sendstr = program.str();
	Sleep(sleep1 + 100);

	sending = FALSE;
		Putanja.open("Putanja2.txt", fstream::in);
```

```
		if (Putanja.is_open()) {

			pocetak = 1;
			Sleep(2000);
			dw1 = GetTickCount();
			while (getline(Putanja, line))
			{
				istringstream iss(line);
				if (!iss) { break; }
				sendbufs = line;
				sendbuf = (char*)sendbufs.c_str();
				cout << sendbufs << "\n";

				if (pocetak == 1) {
					Sleep(100);
					pocetak = 0;
				}
				else Sleep(100);

			}
			Putanja.close();
			ponovno = FALSE;
		}

	goto start;
```

II.5. **Rad na temelju naucenih pokreta** – dodano unutar math.cpp skripte u programu iz [12]

```
Klasifikacija:

	odabir_put = 0;

	do {
		Sleep(1000);
	} while (odabir_put == 0);
	M1_q.w() = M1_qw;
	M1_q.x() = M1_qx;
	M1_q.y() = M1_qy;
	M1_q.z() = M1_qz;
	R_M1 = M1_q.normalized().toRotationMatrix(); //Quaternion to DCM
	T_M1_P << R_M1(0, 0), R_M1(0, 1), R_M1(0, 2), M1_x, R_M1(1, 0), R_M1(1, 1),
R_M1(1, 2), M1_y, R_M1(2, 0), R_M1(2, 1), R_M1(2, 2), M1_z, 0, 0, 0, 1;
	T_M1 = T_M1_P*T_TTO_M1;

	M3_q.w() = M3_qw;
	M3_q.x() = M3_qx;
	M3_q.y() = M3_qy;
	M3_q.z() = M3_qz;
	R_M3 = M3_q.normalized().toRotationMatrix(); //Quaternion to DCM
	T_M3_P << R_M3(0, 0), R_M3(0, 1), R_M3(0, 2), M3_x, R_M3(1, 0), R_M3(1, 1),
R_M3(1, 2), M3_y, R_M3(2, 0), R_M3(2, 1), R_M3(2, 2), M3_z, 0, 0, 0, 1;
	T_M3 = T_M3_P;

	RxRyRz_in << Rx_in, Ry_in, Rz_in;
	theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
	R_Robot_in = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to DCM
```

```
	T_Robot_actual << R_Robot_in(0, 0), R_Robot_in(0, 1), R_Robot_in(0, 2), X_in,
R_Robot_in(1, 0), R_Robot_in(1, 1), R_Robot_in(1, 2), Y_in, R_Robot_in(2, 0),
R_Robot_in(2, 1), R_Robot_in(2, 2), Z_in, 0, 0, 0, 1;

	// ucitaj koordinate nove površine
	Povrsina_new.open("Povrsina_new.txt", fstream::in);
	if (Povrsina_new.is_open()) {
		ii = 0;
		while (getline(Povrsina_new, line))
		{
			istringstream iss(line);
			if (!iss) { break; }

			T_new[ii] = stod(line);
			ii = ii++;
		}
		Povrsina_new.close();
	}

	cout << T_new[1]<<"Y1\n";

	T_T_new << 1, 0, 0, T_new[0], 0, 1, 0, T_new[1], 0, 0, 1, 0, 0, 0, 0, 1;

	T_MP_new = T_M3*T_T_new;

	T_MP_newinM1 = T_M1.inverse()*T_MP_new;

	T_PinR_new = T_Robot_actual*T_MP_newinM1;

	cout << T_PinR_new(0, 3) << "Xnew\n";

	if (odabir_put == 1) {
		cout << "Putanja 1\n";
		Putanja.open("Putanja2.txt", fstream::out | fstream::trunc);
		if (Putanja.is_open()) {
			Putanja.close();
		}
		else cout << "Unable to open file Putanja2.txt\n\n";


		// učitaj koordinate naučene površine
		Povrsina_old.open("Povrsina1.txt", fstream::in);
		if (Povrsina_old.is_open()) {
			ii = 0;

			while (getline(Povrsina_old, line))
			{
				istringstream iss(line);
				if (!iss) { break; }

				cout << line << "\n";
				inbufs = line;
				inbufs.append(",");
				cout << inbufs;
				poss = 0;
				j = 0;
				while (j < 4) {
					poss = inbufs.find(delimiterr);
					numberr = inbufs.substr(0, poss);
					T_old[j] = stod(numberr);
					inbufs.erase(0, poss + delimiterr.length());
					cout << j;
```

```
                        j++;
                    }
                    cout << "\n proso jednom \n";
                    break;
                }
                cout << "\n stigo\n";
                Povrsina_old.close();
            }
            else cout << "Unable to open file Pvrsina1.txt\n\n";

            T_T_old << 1, 0, 0, T_old[0], 0, 1, 0, T_old[1], 0, 0, 1, 0, 0, 0, 0, 1;

            T_MP_old = T_M3*T_T_old;

            T_MP_oldinM1 = T_M1.inverse()*T_MP_old;

            T_PinR_old = T_Robot_actual*T_MP_oldinM1;

            X_T = (T_PinR_new(0, 3) - (T_PinR_old(0, 3))) / 1000;
            Y_T = (T_PinR_new(1, 3) - (T_PinR_old(1, 3))) / 1000;
            cout << "X_T=" << X_T << "\n Y_T=" << Y_T << "\n";

            ///ucitavanje stare putanje robota i izracun nove
            iii = 0;
            Putanja_old.open("Putanja2pov1.txt", fstream::in);
            if (Putanja_old.is_open()) {


                while (getline(Putanja_old, line))
                {
                    if (iii % 2 == 0) {

                        istringstream iss(line);
                        if (!iss) { break; }
                        inbufs = line;
                        brstr = inbufs.length();
                        cout << brstr << " brstr\n";
                        inbufs.erase(brstr - 1, 1);
                        inbufs.erase(0, 3);

                        poss = 0;
                        j = 0;
                        while (j < 9) {
                            poss = inbufs.find(delimiterr);
                            numberr = inbufs.substr(0, poss);
                            Target_old[j] = stod(numberr);
                            inbufs.erase(0, poss + delimiterr.length());
                            j++;
                        }
                        X_old = Target_old[0];
                        Y_old = Target_old[1];
                        Z_old = Target_old[2];
                        Rx_old = Target_old[3];
                        Ry_old = Target_old[4];
                        Rz_old = Target_old[5];
                        aa = Target_old[6];
                        vv = Target_old[7];
                        rr = Target_old[8];
                        X_new = X_old + (X_T);
                        Y_new = Y_old + (Y_T);
                        cout << "X_old=" << X_old << "Y_old" << Y_old;
```

```
                                    Putanja.open("Putanja2.txt", fstream::out |
fstream::app);

                                    if (Putanja.is_open()) {
                                          Putanja << "(1," << X_new << "," << Y_new <<
"," << Z_old << "," << Rx_old << "," << Ry_old << "," << Rz_old << "," << aa << "," <<
vv << "," << rr << ")\n" << endl;
                                          Putanja.close();
                                          Sleep(10);
                                    }
                                    else cout << "Unable to open file Putanja2.txt\n\n";
                              }
                              iii++;
                        }
                  }
                  Putanja_old.close();
                  // Dodatak zadnje točke za stop programa u robotu
                  Putanja.open("Putanja2.txt", fstream::out | fstream::app);

                  if (Putanja.is_open()) {
                        Putanja << "(0," << X_new << "," << Y_new << "," << Z_old << ","
<< Rx_old << "," << Ry_old << "," << Rz_old << "," << aa << "," << vv << "," << rr <<
")\n" << endl;
                        Putanja.close();
                        Sleep(10);
                  }
                  else cout << "Unable to open file Putanja2.txt\n\n";


      }
if (odabir_put == 2) {
                  cout << "Putanja 2\n";
                  Putanja.open("Putanja2.txt", fstream::out | fstream::trunc);
                  if (Putanja.is_open()) {
                        Putanja.close();
                  }
                  else cout << "Unable to open file Putanja2.txt\n\n";


                  // učitaj koordinate naučene površine
                  Povrsina_old.open("Povrsina2.txt", fstream::in);
                  if (Povrsina_old.is_open()) {
                        ii = 0;

                        while (getline(Povrsina_old, line))
                        {
                              istringstream iss(line);
                              if (!iss) { break; }

                              cout << line<<"\n";
                              inbufs = line;
                              inbufs.append(",");
                              cout << inbufs;
                              poss = 0;
                              j = 0;
                              while (j < 4) {
                                    poss = inbufs.find(delimiterr);
                                    numberr = inbufs.substr(0, poss);
                                    T_old[j] = stod(numberr);
                                    inbufs.erase(0, poss + delimiterr.length());
                                    cout << j;
                                    j++;
```

```
                }
                break;
        }
        Povrsina_old.close();
}
else cout << "Unable to open file Pvrsina2.txt\n\n";

T_T_old << 1, 0, 0, T_old[0], 0, 1, 0, T_old[1], 0, 0, 1, 0, 0, 0, 0, 1;

T_MP_old = T_M3*T_T_old;

T_MP_oldinM1 = T_M1.inverse()*T_MP_old;

T_PinR_old = T_Robot_actual*T_MP_oldinM1;

X_T = (T_PinR_new(0, 3) - (T_PinR_old(0, 3)))/1000;
Y_T = (T_PinR_new(1, 3) - (T_PinR_old(1, 3)))/1000;
cout << "X_T=" << X_T << "\n Y_T=" << Y_T << "\n";

///ucitavanje stare putanje robota i izracun nove
iii = 0;
Putanja_old.open("Putanja2pov2.txt", fstream::in);
if (Putanja_old.is_open()) {


        while (getline(Putanja_old, line))
        {
                if (iii%2==0) {

                        istringstream iss(line);
                        if (!iss) { break; }
                        inbufs = line;
                        brstr = inbufs.length();
                        cout << brstr << " brstr\n";
                        //ucitavanje starog targeta
                        inbufs.erase(brstr - 1, 1);
                        inbufs.erase(0, 3);

                        poss = 0;
                        j = 0;
                        while (j < 9) {
                                poss = inbufs.find(delimiterr);
                                numberr = inbufs.substr(0, poss);
                                Target_old[j] = stod(numberr);
                                inbufs.erase(0, poss + delimiterr.length());
                                j++;
                        }

                        X_old = Target_old[0];
                        Y_old = Target_old[1];
                        Z_old = Target_old[2];
                        Rx_old = Target_old[3];
                        Ry_old = Target_old[4];
                        Rz_old = Target_old[5];
                        aa = Target_old[6];
                        vv = Target_old[7];
                        rr = Target_old[8];
                        X_new = X_old + (X_T);
                        Y_new = Y_old + (Y_T);
                        cout << "X_old=" << X_old << "Y_old" << Y_old;
```

```
                                    Putanja.open("Putanja2.txt", fstream::out |
fstream::app);

                                    if (Putanja.is_open()) {
                                          Putanja << "(1," << X_new << "," << Y_new <<
"," << Z_old << "," << Rx_old << "," << Ry_old << "," << Rz_old << "," << aa << "," <<
vv << "," << rr << ")\n" << endl;
                                          Putanja.close();
                                          Sleep(10);
                                    }
                                    else cout << "Unable to open file Putanja2.txt\n\n";
                              }
                              iii++;
                        }
                  }
                  Putanja_old.close();

                  // Dodatak zadnje točke za stop programa u robotu
                  Putanja.open("Putanja2.txt", fstream::out | fstream::app);

                  if (Putanja.is_open()) {
                        Putanja << "(0," << X_new << "," << Y_new << "," << Z_old << ","
<< Rx_old << "," << Ry_old << "," << Rz_old << "," << aa << "," << vv << "," << rr <<
")\n" << endl;
                        Putanja.close();
                        Sleep(10);
                  }
                  else cout << "Unable to open file Putanja2.txt\n\n";


      }
if (odabir_put == 3) {
                  cout << "Putanja 3\n";
                  Putanja.open("Putanja2.txt", fstream::out | fstream::trunc);
                  if (Putanja.is_open()) {
                        Putanja.close();
                  }
                  else cout << "Unable to open file Putanja2.txt\n\n";


                  // učitaj koordinate naučene površine
                  Povrsina_old.open("Povrsina3.txt", fstream::in);
                  if (Povrsina_old.is_open()) {
                        ii = 0;

                        while (getline(Povrsina_old, line))
                        {
                              istringstream iss(line);
                              if (!iss) { break; }

                              cout << line << "\n";
                              inbufs = line;
                              inbufs.append(",");
                              cout << inbufs;
                              poss = 0;
                              j = 0;
                              while (j < 4) {
                                    poss = inbufs.find(delimiterr);
                                    numberr = inbufs.substr(0, poss);
                                    T_old[j] = stod(numberr);
                                    inbufs.erase(0, poss + delimiterr.length());
                                    cout << j;
```

```
                                j++;
                        }
                        break;
                }
                        Povrsina_old.close();
        }
        else cout << "Unable to open file Pvrsina3.txt\n\n";

        T_T_old << 1, 0, 0, T_old[0], 0, 1, 0, T_old[1], 0, 0, 1, 0, 0, 0, 0, 1;

        T_MP_old = T_M3*T_T_old;

        T_MP_oldinM1 = T_M1.inverse()*T_MP_old;

        T_PinR_old = T_Robot_actual*T_MP_oldinM1;

        X_T = (T_PinR_new(0, 3) - (T_PinR_old(0, 3))) / 1000;
        Y_T = (T_PinR_new(1, 3) - (T_PinR_old(1, 3))) / 1000;
        cout << "X_T=" << X_T << "\n Y_T=" << Y_T << "\n";

        ///ucitavanje stare putanje robota i izracun nove

        iii = 0;

        Putanja_old.open("Putanja2pov3.txt", fstream::in);
        if (Putanja_old.is_open()) {

                while (getline(Putanja_old, line))
                {
                        if (iii % 2 == 0) {
                                istringstream iss(line);
                                if (!iss) { break; }
                                inbufs = line;
                                brstr = inbufs.length();
                                cout << brstr << " brstr\n";
                                inbufs.erase(brstr - 1, 1);
                                inbufs.erase(0, 3);
                                poss = 0;
                                j = 0;
                                while (j < 9) {
                                        poss = inbufs.find(delimiterr);
                                        numberr = inbufs.substr(0, poss);
                                        Target_old[j] = stod(numberr);
                                        inbufs.erase(0, poss + delimiterr.length());
                                        j++;
                                }
                                X_old = Target_old[0];
                                Y_old = Target_old[1];
                                Z_old = Target_old[2];
                                Rx_old = Target_old[3];
                                Ry_old = Target_old[4];
                                Rz_old = Target_old[5];
                                aa = Target_old[6];
                                vv = Target_old[7];
                                rr = Target_old[8];
                                X_new = X_old + (X_T);
                                Y_new = Y_old + (Y_T);
                                cout << "X_old=" << X_old << "Y_old" << Y_old;

                                Putanja.open("Putanja2.txt", fstream::out |
fstream::app);
```

```
                                    if (Putanja.is_open()) {
                                          Putanja << "(1," << X_new << "," << Y_new <<
"," << Z_old << "," << Rx_old << "," << Ry_old << "," << Rz_old << "," << aa << "," <<
vv << "," << rr << ")\n" << endl;
                                          Putanja.close();
                                          Sleep(10);
                                    }
                                    else cout << "Unable to open file Putanja2.txt\n\n";
                              }
                              iii++;
                        }
                  }
                  Putanja_old.close();

                  // Dodatak zadnje točke za stop programa u robotu
                  Putanja.open("Putanja2.txt", fstream::out | fstream::app);

                  if (Putanja.is_open()) {
                        Putanja << "(0," << X_new << "," << Y_new << "," << Z_old << ","
<< Rx_old << "," << Ry_old << "," << Rz_old << "," << aa << "," << vv << "," << rr <<
")\n" << endl;
                        Putanja.close();
                        Sleep(10);
                        }
                  else cout << "Unable to open file Putanja2.txt\n\n";


      }

      goto start;

      return;
}
```