Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Matevž Poberžnik

Daljinsko vodenje robota s posnemanjem operaterjevih gibov

diplomsko delo

na interdisciplinarnem univerzitetnem študiju

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00046/2013
Datum: 02.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **MATEVŽ POBERŽNIK**

Naslov: **DALJINSKO VODENJE ROBOTA S POSNEMANJEM OPERATERJEVIH GIBOV**

REMOTE ROBOT CONTROL THROUGH MIMICING OPERATOR'S MOVEMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Eden od pristopov k programiranju robotov temelji na operaterjevi demonstraciji gibov ter robotovem posnemanju teh gibov. Naloga je izdelati vmesnik za zajem podatkov o operaterjevem gibu ter prilagoditi demonstrirano gibanje mehanskim zmožnostim robota. Ta projekt naj bo izveden za gibanje rok robota Nao. Uporabite naslednje senzorje: pospeškometer, žiroskop in magnetometer. Ovrednotite uspešnost razvitega sistema.

Mentor:

akad. prof. dr. Ivan Bratko

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 1

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matevž Poberžnik, z vpisno številko **63080174**, sem avtor diplomskega dela z naslovom:

Daljinsko vodenje robota s posnemanjem operaterjevih gibov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivan Bratko,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 25. septembra 2013 Podpis avtorja:

Zahvaljujem se dr. Juretu Žabkarju za pomoč ter vsem ostalim, ki ste kakorkoli prispevali svoj delež. Hvala.

Kazalo

Povzetek

V diplomski nalogi sem razvil vmesnik za upravljanje robotske roke z uporabo senzorjev na človeški roki. Pri tem naj bi robot posnemal človekov gib. S tem je upravljanje robotske roke intuitivno, hitro in enostavno. Osredotočil sem se na upravljanje robotske roke humanoidnega robota Nao z uporabo nizko-cenovnih senzorjev - žiroskopov, pospeškometrov in magnetometrov. Pri realizaciji sem uporabil dve skupini teh treh senzorjev - ena za nadlaket ter druga za podlaket.

Elektronski vmesnik za zajem podatkov gibanja roke je sestavljen iz mikrokrmilnika, senzorjev in komunikacijskega vmesnika bluetooth. Podrobneje sem opisal senzorje, na kakšen način delujejo ter kako mikrokrmilnik komunicira s senzorji ter z računalnikom. Razvil sem algoritem za izračun prostorske rotacije vmesnika iz podatkov s senzorjev na vmesniku. Za združevanje podatkov sem uporabil komplementarni filter. Senzorje je za dobro delovanje algoritma potrebno tudi kalibrirati - opisal sem težave s senzorji in postopke kalibracije, s katerimi v veliki meri odpravimo napake. Preučil sem razlike in podrobnosti človeške roke in roke robota Nao ter opisal postopek, kako z dobljenimi podatki (rotacije nadlakti in podlakti) upravljamo roko robota Nao. Na treh preprostih nalogah sem preizkusil zahtevnost in natančnost vodenja robotske roke.

Ključne besede

uporavljanje robotske roke, humanoidni robot Nao, komplementarni filter, žiroskop, pospeškometer, magnetometer, rotacijska matrika

Abstract

In this thesis I describe the development of a human robot interface, specifically, an interface to control the arm of humanoid robot Nao. My controller consists of low-cost sensors - gyroscopes, accelerometers and magnetometers, which are strapped to a human arm. Using this hardware, my goal was to implement software filters to establish an accurate mapping between a human and a robot arm.

The hardware part of the interface that takes care of data acquisition consists of a microcontroller, the sensors and a Bluetooth communication interface. I describe all hardware parts in details and explain the communication between the microcontroller and sensors and PC. Furthermore, I designed an algorithm for computation of spatial rotation from the obtained sensory data. I used complementary filter to join the data. I describe sensor calibration procedures, the problems I encountered during calibration and propose the solutions, which diminish the errors as much as possible. I study the differences between a human and Nao's arm and propose a method for Nao arm control from gathered data. I test my system on three simple tasks with respect to the complexity and accuracy of controlling Nao's arm.

Keywords

robotic arm control, humanoid robot Nao, complementary filter, gyroscope, accelerometer, magnetometer, rotational matrix

Poglavje 1

Uvod

Z razvojem robotov se razvijajo tudi nove tehnike interakcije z roboti. Roboti postajajo vedno bolj tehnično dovršeni in s tem tudi vedno bolj zapleteni za upravljanje. Z uporabo klasičnih orodij za interakcijo, kot so tipkovnica, miška in igralna palica, ki se v praksi še vedno v veliki meri uporabljajo, je upravljanje robota z velikim številom prostostnih stopenj zelo zapleteno in neprijazno za uporabnika. Učinkovito upravljanje robota je koristno na več področjih, predvsem takrat, kadar človek ni zmožen opraviti določene naloge ali pa je ogroženo njegovo življenje. Robote lahko uporabimo v industriji, v medicini (za izvajanje zapletenih kirurških posegov), pri iskanju ponesrečencev (na primer iz gorečih hiš, z območja povečanega radioaktivnega sevanja ipd.), v vojski (pri detonaciji eksplozivnih snovi in vojskovanju), pri misijah v vesolju ipd.

V diplomski nalogi sem razvil sistem za upravljanje robotske roke z uporabo senzorjev na človeški roki. Upravitelj z gibanjem roke upravlja robotsko roko, ki sledi njegovim gibom, kolikor to omogoča konstrukcija robota. V diplomski nalogi sem se osredotočil na gibanje roke humanoidnega robota Nao. Za upravljanje roke je dovolj, da za nadlaket in podlaket ugotovimo rotacijo (oz. prostorsko usmeritev) v 3D prostoru. To lahko izračunamo z uporabo žiroskopa, pospeškometra in magnetometra. Podobni sistemi, ki temeljijo na računanju rotacije s podobnim naborom senzorjev se večinoma uporabljajo v

brezpilotnih letalih za detekcijo naklona, odklona in zasuka. V enostavnejši obliki se uporabljajo tudi pri dvo-kolesnih samouravnoteževalnih robotih in dvo-kolesnih samouravnoteževalnih prevoznih sredstvih Segway, kjer je potreben samo izračun naklona, kar precej poenostavi problem.

Pri brezpilotnih letalih se večinoma uporablja Kalmanov filter, ki za delovanje potrebuje uporabo kvaternionov, za popravljanje azimuta pa se ponavadi uporabijo podatki GPS, ki služijo tudi za določanje položaja [5]. V svoji rešitvi sem uporabil komplementarni filter in rotacijske matrike, ki so bolj enostavne in razumljive za uporabo. Komplementarni filter uporabljajo pri dvo-kolesnih samouravnoteževalnih robotih, saj je zelo enostaven za uporabo pri dvo-dimenzionalnem problemu [4]. V diplomski nalogi sem dvo-dimenzionalni komplementarni filter razširil na uporabo v tro-dimenzionalnih problemih računanja prostorske rotacije.

Za zajem podatkov sem uporabil mikrokrmilnik Arduino. Mikrokrmilnik komunicira s senzorji in njihove podatke posreduje preko povezave bluetooth na računalnik. Na računalniku se podatki procesirajo z uporabo prej omenjenega komplementarnega filtra. Podatki s senzorjev so šumni in pristranski, zato jih moramo kalibrirati. Največ težav povzroča magnetometer, saj nanj delujejo vsi feromagnetni materiali v okolici senzorja. Kalibracija magnetometra je zapletena, poleg tega pa nam ravno nekalibrirani podatki magnetometra povzročajo največje napake.

Humanoidni robot Nao ima roko na videz podobno človeški, razlikujeta pa se v razporeditvi prostostnih stopenj v sklepih ter v gibljivosti sklepov. Za upravljanje robota moramo narediti preslikavo med rotacijami človeške roke ter rotacijami roke robota Nao. Ker ima robot Nao določene fizične omejitve v gibljivosti sklepov, to ni vedo mogoče.

Za preverjanje učinkovitosti vmesnika za upravljanje robota sem določil preproste naloge - premik šahovske figure na določeno polje, postavitev stolpa šahovskih trdnjav in pisanje. Naloge preverjajo različne spretnosti in omejitve pri uporabi vmesnika.

Poglavje 2

Razvoj vmesnika za zajem podatkov gibanja roke

Razvil sem elektronski vmesnik za zajem podatkov gibanja, ki temelji na žiroskopih, pospeškometrih in magnetometrih. Uporabil sem odprto-kodno razvojno ploščico Arduino, magnetometra ter mehanska pospeškometra in žiroskopa. V nadaljevanju bom opisal sestavo vezja ter razložil njegovo delovanje in komunikacijo med komponenatami sistema.

2.1 Arduino

Projekt temelji na odprto-kodni razvojni ploščici Arduino. Arduino je osnovan na Atmel-ovem 8-bitnem megaAVR mikrokrminiliku. V primerjavi z alternativnimi mikrokrmilniki je Arduino zaradi svoje enostavnosti namenjen širši množici uporabnikov. Obstaja več različnih modelov, ki se med seboj razlikujejo v karakteristikah. Za potrebe projekta sem uporabil Arduino Pro Mini.

Arduino Pro Mini je namenjen naprednejšim uporabnikom. Na razvojni ploščici v primerjavi z drugimi modeli nima podpore za USB - s tem se mu zmanjšata velikost in cena. Za programiranje tako potrebujemo poseben vmesnik. Osnovne karakteristike so na voljo v tabeli A.1 (glej dodatek A).

2.1.1 Razvojno okolje Arduino

Arduino nam nudi posebno integrirano razvojno okolje, v katerem lahko pišemo programe imenovane skice (angl. sketch), jih nato preverimo in prevedemo ter naložimo na mikrokrmilnik. Programiranje poteka v posebnem programskem jeziku Arduino, ki temelji na odprto-kodnem ogrodju za programiranje mikrokrmilnikov Wiring. Wiring je knjižnica za C/C++, ki poenostavi programiranje, še posebej osnovne vhodno/izhodne operacije. V programskem okolju Arduino so vključene tudi nekatere osnovne knjižnice, ki nam olajšajo delo.

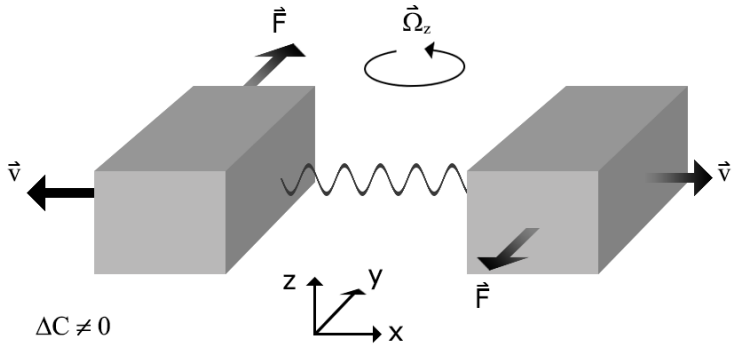
2.2 Senzorji

V projektu sem uporabil žiroskop, pospeškometer in magnetometer (vsi trije so tri osni senzorji). Žiroskop in pospeškometer sta tehnologije MEMS (Micro-Electro-Mechanical Systems), za razliko od magnetometra, ki ne vsebuje mehanskih delov. Vsi trije senzorji komunicirajo z mikrokrminikom preko protokola I^2C.

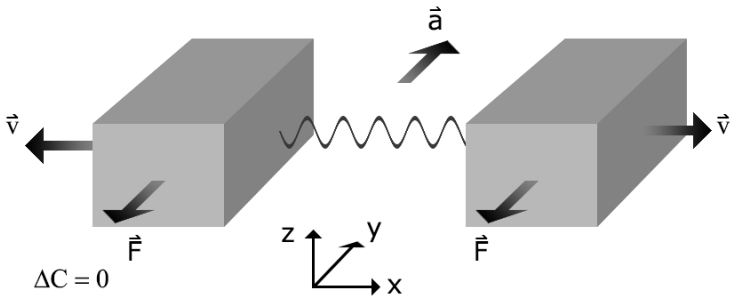
2.2.1 Žiroskop

Uporabil sem tri osni žiroskop L3G4200D (proizvajalec STMicroelectronics) [11, 6], ki je narejen s tehnologijo MEMS (Micro-Electro-Mechanical Systems). Tri osni žiroskop meri rotacijo okrog treh glavnih osi. Podatki, ki jih dobimo s senzorja, so skalirani glede na resolucijo in so v enotah °/s.

MEMS žiroskopi za merjenje kotne hitrosti uporabljajo Coriolisov efekt. V osnovi delujejo tako, da dve masi nihata (vsaka v svojo smer). Ko zarotiramo senzor, na ti masi deluje Coriolisova sila, ki je za vsako maso v nasprotni smeri, kar se odraža v razliki kapacitivnosti (prikazano na sliki 2.1). Kadar pa na senzor deluje linearni pospešek, deluje sila na obe masi v enako smer - v tem primeru ni razlike v kapacitivnosti in tako žiroskop ne bo zaznal rotacije, kar je seveda pravilno (pikazano na sliki 2.2). Če bi uporabili samo



Slika 2.1: Delovanje Coriolisove sile na nihajoči se masi.



Slika 2.2: Delovanje pospeška na nihajoči se masi.

eno maso, ne bi mogli ločiti sile pospeška od Coriolisove sile.

V tabeli A.2 so prikazane osnovne karakteristike uporabljenega žiroskopa. Žiroskop ima tri različne resolucije, izhodni podatki pa se ne preslikajo na celotni 16-bitni interval - v tabeli A.3 so prikazane občutljivosti glede na resolucijo.

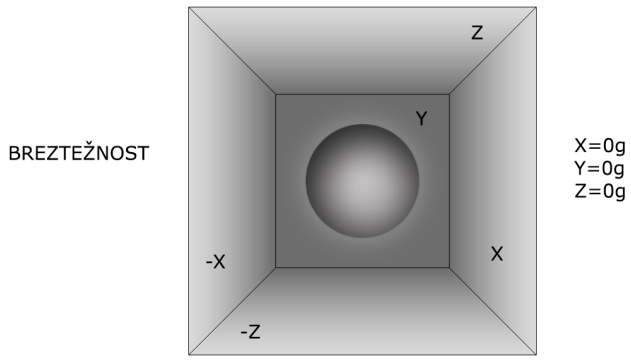
Pomembna značilnost MEMS žiroskopov pri razvoju našega algoritma je, da je rezultat zelo občutljiv in z malo šuma. Kot bomo videli v nadaljevanju, je težava žiroskopov v tem, da se napaka z integriranjem akumulira. Zaradi tega moramo uporabiti tudi pospeškometer in magnetometer, ki nam popravljata napako med delovanjem.

2.2.2 Pospeškometer

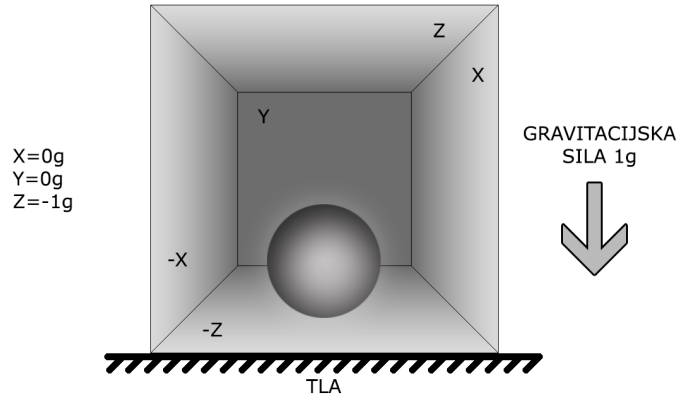
Pospeškometer je prav tako tri osni in sicer ADXL345 (proizvajalec ANALOG DEVICES) [2, 10]. Pospeškometer meri sile, ki delujejo na senzor. Delovanje pospeškometra si lahko predstavljamo s kroglo z določeno maso v kocki (slika 2.3), kot izhod pa uporabimo sile, s katerimi deluje krogla na ploskve kocke. Tako bi v prostoru brez gravitacije krogla v kocki lebdela in s tem ne bi povzročila sile na nobeno izmed ploskev v kocki - prikazano na sliki 2.3. Pri pospeševanju (slika 2.6) deluje sila krogle v kocki na ploskev, ki je v nasprotni smeri kot deluje pospešek. Tako zaznamo silo v nasprotni smeri kot deluje pospešek, kar lahko enostavno z uporabo nasprotne vrednosti spremenimo v pospešek.

Zemeljska gravitacija povleče kroglo v kocki proti središču Zemlje in tako nastane sila na spodnjo površino kocke v velikosti 1 g (slika 2.4). V tem razdelku uporabljamo 1 g kot enoto za pospešek, pa tudi kot relativno enoto za silo. In sicer je to sila, ki jo povzroči zemeljski gravitacijski pospešek na enoto mase. Npr. pri masi 1 kg je ta sila 9,81 $\frac{kg}{s^2}$. Če kocka ni poravnana s površino Zemlje, se sila 1 g odraža na več ploskev hkrati v sestavljeni velikosti 1g (slika 2.5). V primeru prostega pada zaznamo breztežnost, v primeru mirovanja pa kot sem že omenil silo gravitacije v velikosti 1 g.

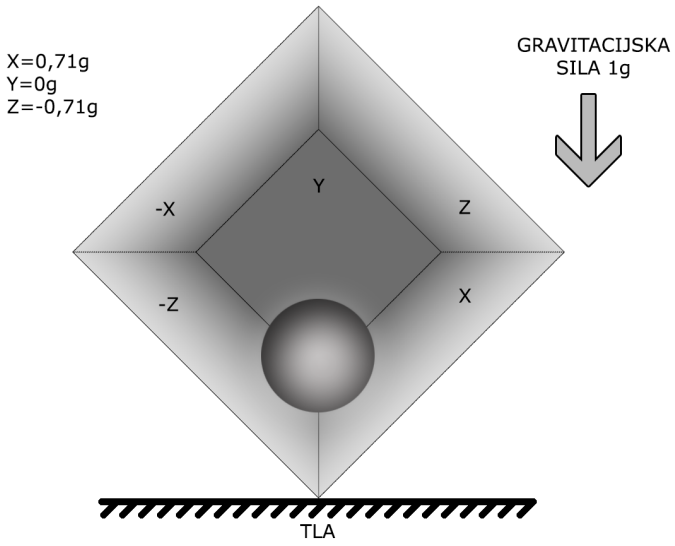
Kadar ob prisotnosti gravitacije delujemo na naš model še z drugo zuna-



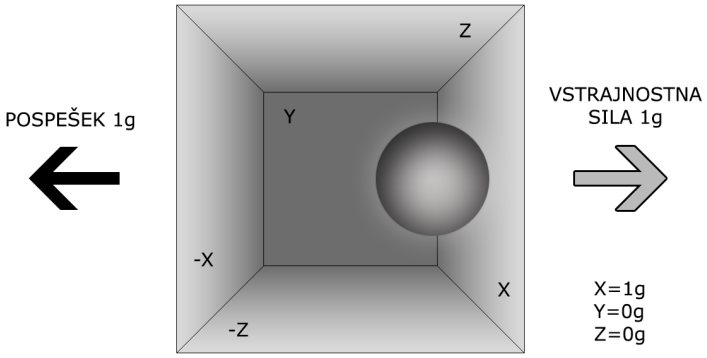
Slika 2.3: Model za predstavo delovanja pospeškometra - breztežnost.



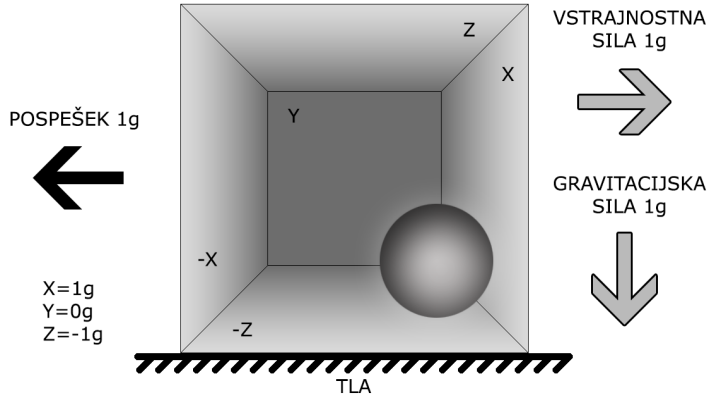
Slika 2.4: Delovanje gravitacije na naš model pospeškometra.



Slika 2.5: Delovanje gravitacije na model v poševni legi pod kotom 45°.



Slika 2.6: Delovanje pospeška v breztežnostnem prostoru.



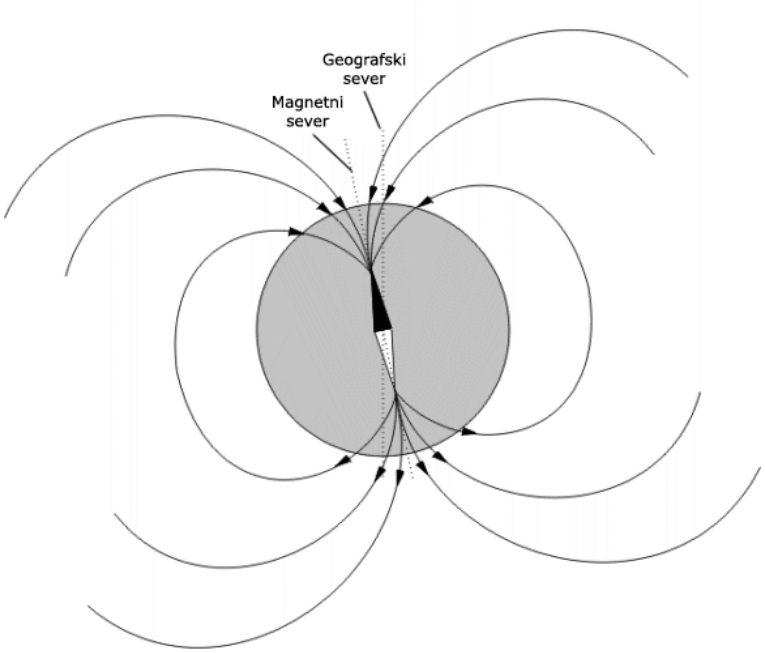
Slika 2.7: Delovanje gravitacije in pospeška na model.

njo silo (pospeškom), se sila gravitacije in naša sila seštejeta (slika 2.7). Če želimo dobiti kot rezultat samo silo pospeška, s katerim mi delujemo na pospeškometer, moramo od rezultata odšteti silo gravitacije. To ni enostavno, saj med pospešenim gibanjem samo s pospeškometrom ne moremo ugotoviti rotacije v prostoru in s tem tudi ne smeri, v kateri deluje gravitacija. Nam pa zmožnost zaznave smeri gravitacije v mirovanju oz. enakomernem gibanju omogoča, da zelo natančno ugotovimo nagib pospeškometra, kar bomo izkoristili pri našem algoritmu. Kot bomo videli v poglavju 3, to velja tudi za šibke pospeške.

Nekatere pomembne karakteristike uporabljenega pospeškometra ADXL345 se nahajajo v tabeli A.5, uporabljene nastavitve pa v tabeli A.6.

2.2.3 Magnetometer

Magnetometer HMC5883L (proizvajalec Honeywell) [7] zaznava magnetno polje v prostoru in meri magnetne silnice v vseh treh oseh. Enota za magnetno polje je gauss (Ga). Za zaznavanje magnetnega polja uporablja magnetoresistivne senzorje. Magnetometer ima tri takšne senzorje, za vsako dimenzijo svojega. Na te senzorje vpliva magnetno polje, ki se odraža v količni toka, ki steče skozi senzor. Za razliko od žiroskopa in pospeškometra,



Slika 2.8: Odklon in naklon magnetnega polja Zemlje.

magnetometer nima mehanskih delov.

Magnetometer je zelo natančen in omogoča izračun azimuta 1° do 2° natančno, vendar ima veliko težavo, saj na magnetno polje delujejo vsi feromagnetni materiali v okolici senzorja in motijo pravilno delovanje. V določeni meri lahko te motnje odpravimo s kalibracijo (podrobnejši opis motenj in postopek kalibracije sta opisana v poglavju 3).

Pri uporabi magnetometra moramo biti pozorni tudi na odklon in naklon magnetnega polja, ki sta odvisna od lokacije na Zemlji (prikazano na sliki 2.8). Odklon magnetnega polja se kaže v tem, da sta magnetni in geografski sever Zemlje različna. Naklon pa se kaže v tem, da je magnetno polje na polih skoraj navpično, ob ekvatorju pa skoraj vodoravno. Glede na lokacijo se spreminja tudi jakost magnetnega polja. Podatki o jakosti, naklonu in odklonu magnetnega polja glede na lokacijo na Zemlji so dostopni na *http://magnetic-declination.com*.

Podrobne karakteristike magnetometra in uporabljene nastavitve se nahajajo v tabli A.7 in A.8.

2.2.4 Gumb

Pri projektu sem uporabil tudi dodatni gumb. Gre za manjši potisni gumb, ki ustvari stik, kadar je stisnjen in se po spustu vrne v prvotno stanje (brez stika). Gumb sem uporabil na prstih in bo skrbel za upravljanje pesti robota (stisk in spust).

2.3 Komunikacija

V tem poglavju si bomo ogledali komunikacijo med mikrokrmilnikom in senzorji (I^2C) ter komunikacijo med mikrokrmilnikom in računalnikom (serijska povezava).

2.3.1 Komunikacija mikrokrmilnika s senzorji

Na mikrokrmilnik je preko protokola I^2C (Inter Integrated Circuit) [3] povezanih šest senzorjev - dva pospeškometra, dva žiroskopa in dva magnetometra. Ker imata enaka senzorja isti naslov v protokolu I^2C, je bilo potrebno uporabiti tudi analogni multiplekser - CD74HC4052E (proizvajalec Texas Instruments) [12].

I^2C vmesnik je sinhron prenos podatkov preko dveh signalov (žic). Podatki se prenašajo serijsko po signalu SDA, SCL pa služi kot ura. Oba signala sta tipa odprti kolektor. I^2C deluje na principu gospodar - suženj. Arduino je v našem primeru gospodar, senzorji pa so sužnji. Zaradi tipa odprti kolektor so lahko vsi senzorji priklopljeni na oba signala, vsak suženj pa mora imeti svoj naslov. Ker imata v mojem primeru po dva senzorja enak naslov, se bi ob naslavljanju odzvala oba senzorja, kar bi privedlo do napak. Zaradi tega je potrebno preklapljati signal med prvimi in drugimi senzorji s pomočjo analognega multiplekserja.

Prenos podatkov

Osnovni primer signalov je prikazan na sliki 2.9. Gospodar pred in po prenosu tvori START bit in STOP bit. START je negativna fronta signala SDA, medtem ko je STOP pozitivna fronta SDA (oboje, kadar je SCL v visokem stanju). START in STOP sta edina, ki se spremenita, kadar je SCL v visokem stanju - med prenosom se SDA lahko spremeni samo takrat, ko je SCL v nizkem stanju.

Po START bitu gospodar pošlje (ponavadi) 7-bitni naslov sužnja (lahko je tudi 10-biten), ki mu sledi bit, ki sporoči sužnju, če želi gospodar pošiljati (0) ali brati (1). Vsi prenosi so dolžine enega bajta, ki jim sledi potrditveni (ACK) bit - tudi kadar gospodar pošlje naslov, mora počakati na ACK bit od sužnja. Pri pošiljanju se najprej prenese najpomembnejši bit (MSB). Nato sledi izmenjava podatkov (enega ali več bajtov, ki jim vedno sledi ACK bit) ter na koncu STOP bit.

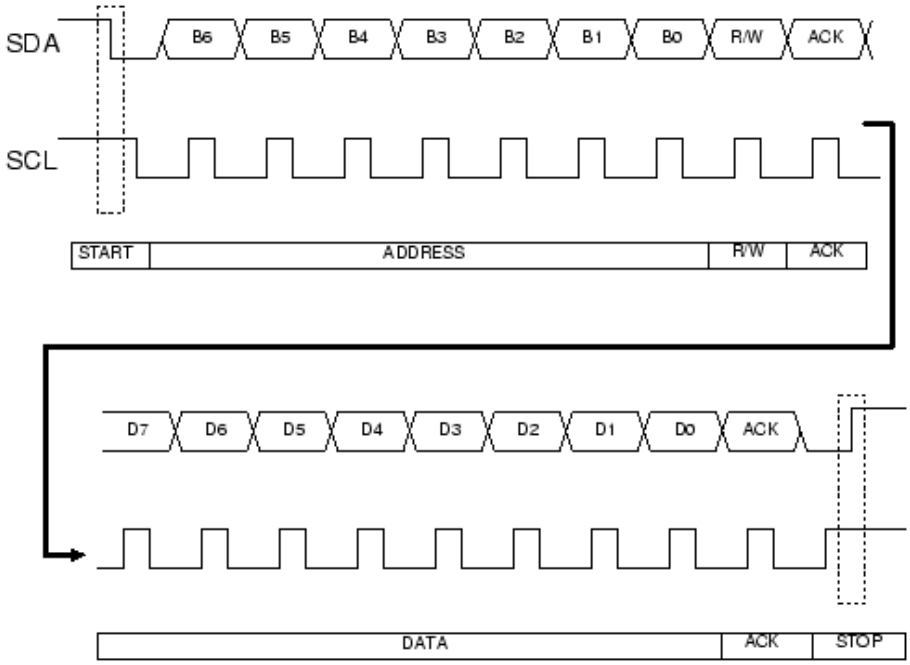
Potrditev po koncu vsakega bajta se izvede tako, da gospodar med generiranjem impulza na SCL sprosti SDA linijo. Sedaj mora tisti, ki je sprejel podatke postaviti SDA linijo v nizko stanje (dokler je SCL v visokem stanju). Če ni potrditve, je potrebno prenos ponoviti.

Komunikacija se vedno izvaja med gospodarjem in enem od sužnjev - ne more se izvajati med dvema sužnjema. Vmesnik I^2C tudi omogoča, da ga upravljata dva gospodarja.

2.3.2 Komunikacija računalnika z mikrokrmilnikom

Za povezavo med Arduinom in računalnikom sem uporabil brezžično povezavo bluetooth. Za povezavo sem uporabil modul JY-MCU. Arduino z modulom komunicira preko serijske povezave. Računalnik mora najprej vzpostaviti povezavo z modulom (računalnik je gostitelj) in tako kreira virtualna vrata COM. Nato modul sam poskrbi za pošiljanje podatkov, ki jih dobi z mikrokrmilnika preko serijske povezave.

Pred uporabo je potrebno modulu nastaviti hitrost in nastavitve prenosa



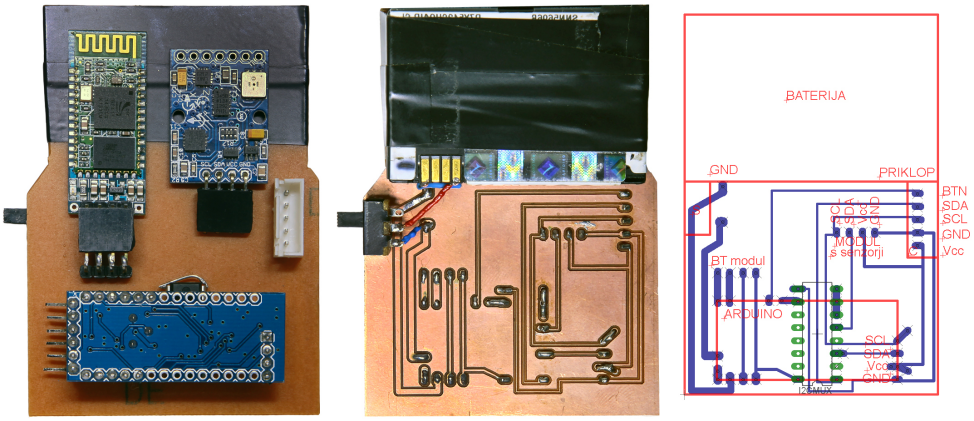
Slika 2.9: Osnovni primer signalov na SDA in SCL žici.

podatkov na serijski povezavi. Lahko mu nastavimo tudi ime naprave in geslo, ki je potrebno za združevanje brezžičnih naprav pri protokolu bluetooth.

Kodiranje podatkov

Z omenjemin modulom smo dosegli, da lahko uporabimo brezžično povezavo, čeprav tako mikrokrmilnik kot računalnik sprejmeta oz. pošljeta podatke v obliki serijske povezave. Tako je potrebno podatke za serijsko povezavo zakodirati v neko primerno obliko.

Ob vsakem prenosu je potrebno poslati po 2 bajta za vsako izmed osi na vsakem senzorju. Tako je potrebno poslati 36 bajtov. Potrebno je vedeti, kdaj se prenos začne, da lahko podatke pravilno interpretiramo - za ta namen sem uporabil zgornji bit (prvi bajt ima ta bit 1, vsi ostali pa 0). Zaradi tega je potrebno poslati več podatkov (natančneje 40 bajtov), vendar nam to tudi omogoča, da ugotovimo, če se je pri prenosu izgubila kakšna infor-



Slika 2.10: Prikaz modula z zgornje in spodnje strani ter načrt vezja.

macija in zavržemo paket. Podatki s pospeškometra so 13-bitni in jih lahko zakodiramo v 2 bajta (pri pravilu, da je zgornji bit v naprej določen), pri žiroskopu in magnetometru pa je potrebno za vsak senzor (dva žiroskopa in dva pospeškometra) dodati še en bajt, kjer se nahajajo zgornji biti podatkov, ki jih je bilo potrebno zaradi protokola odstraniti (tako dobimo 4 dodatne bajte). Tak način kodiranja je robusten in učinkovit.

2.4 Vezje

Vezje je sestavljeno iz dveh modulov, ki sta med seboj povezana. Glavni modul se nahaja na nadlakti. Na sliki 2.10 je prikazan glavni modul z zgornje in s spodnje strani ter na skrajni desni načrt vezja. Na glavnem modulu je mikrokrmilnik Arduino, baterija za napajanje, I^2C selektor, stikalo za vklop, vmesnik bluetooth za komunikacijo z računalnikom, vmesnik s senzorji (žiroskop, pospeškometer in magnetometer) ter priklop za drugi modul. Drugi modul se nahaja na podlakti in ga sestavljata vmesnik s senzorji in gumb za upravljanje stiska zapestja robota. Na sliki 4.3 v poglavju 4.1 je prikazana uporaba vezja na roki upravitelja.

Poglavje 3

Razvoj algoritma za obdelavo podatkov

Razvil sem algoritem, ki z uporabo vmesnika za zajem podatkov gibanja roke s poglavja 2 v realnem času izračunava položaj roke v 3D prostoru. Algoritem temelji na združevanju podatkov iz različnih senzorjev (pospeškometra, žiroskopa in magnetometra). Za izračun položaja roke potrebujemo dve skupini senzorjev (ena za nadlaket ter druga za podlaket) - obe skupini vsebujeta vse tri uporabljene senzorje. Vsaka skupina nam daje podatke za izračun rotacije določenega dela roke, z združitvijo pa lahko dobimo položaj celotne roke. V tem poglavju je opisan postopek, kako združiti podatke z različnih senzorjev ene instance v končni rezultat, ki nam prikazuje rotacijo v prostoru. Sprva so opisani enostavni pristopi, ki jim sledi uporaba komplementarnega filtra. Na koncu poglavja pa so opisani postopki kalibracije senzorjev, kar je zelo pomembno za natančno in robustno delovanje algoritma.

3.1 Karakteristike uporabljenih senzorjev

3.1.1 Uporaba pospeškometra

Marsikdo bi sprva pomislil, da je pospeškometer dovolj za detektiranje položaja roke. Z integriranjem pospeška lahko dobimo hitrost, z integriranjem hitro-

sti opravljeno pot ter tako premik roke in njen nov položaj. To v praksi ni mogoče, saj so podatki zelo šumni ter tako prihaja do prevelikih napak pri dvojnem integriranju. Imeli bi tudi težave pri računanju kotne hitrosti ter pri eliminiranju gravitacijske sile.

Pospeškometer ima tudi dobro lastnost, ki jo bomo uporabili pri našem algoritmu. Pospeškometer pravzaprav meri sile, ki delujejo na senzor. Kadar senzor miruje, nanj deluje sila gravitacijskega pospeška, ki jo detektira senzor. Tako lahko zelo natančno zaznamo naklon senzorja v prostoru (zaznamo lahko spremembo naklona, ki je manjša od 1°). Dobra lastnost takšne uporabe je tudi ta, da lahko kadarkoli zaznamo absoluten naklon v prostoru, ne samo relativne premike, za katere je potrebna začetna inicializacija ter integriranje in težijo h komulativni napaki.

Uporaba samo pospeškometra ne bi bila zadostna za naš algoritem. Prva težava nastane, ker med premikanjem roke senzor zazna tudi pospešek roke in ne samo gravitacijske sile. Tako bi dobili med gibanjem roke napačne rezultate, pravilni rezultat bi bil zgolj, kadar je roka v mirovanju oz. v gibanju z zelo šibkim pospeškom. Druga težave je v detektiranju rotacije okrog sile gravitacije - pospeškometer je sposoben detektirati le naklon, ne pa tudi azimuta.

3.1.2 Uporaba žiroskopa

Žiroskop meri kotno hitrost. Je zelo natančna naprava in nam z integriranjem ter poznavanjem začetne rotacije objekta omogoča, da izračunamo trenutno rotacijo objekta.

Vendar ima tudi žiroskop slabe lastnosti. Pri vsakem integriranju nastane mala napaka, ki se akomulira. Tako izračunana rotacija počasi drsi iz pravilne rotacije. Poleg tega žiroskop ni sposoben globalno zaznati rotacije in jo mora izračunati s prejšnje rotacije - tako moramo na začetku narediti inicializacijo rotacije objekta.

Za odpravljanje komulativne napake pri integriranju in za začetno inicializacijo bomo uporabili podatke s pospeškometra in magnetometra, ki jih

bomo združili s podatki žiroskopa.

3.2 Komplementarni filter in razvoj algoritma

Komplementarni filter je algoritem, ki združi podobne podatke iz različnih senzorjev (ponavadi dveh), da bi dosegel robusten približek določene veličine. Pri tem gre ponavadi za dva, po karakteristiki različna signala. Eden vsebuje nizko-frekvenčni šum (ki ga filtriramo z visoko-prepustnim filtrom), drugi pa vsebuje visoko-frekvenčni šum (ki ga filtriramo z nizko-prepustnim filtrom). Izkaže se, da imajo podatki z žiroskopa in pospeškometra ravno takšne karakteristike in ju lahko zelo učinkovito združimo s komplementarnim filtrom.

Pri tem uporabimo podatke s pospeškometra kot približek smeri gravitacije. S tem smo naredili predpostavko, da bodo med premikanjem roke delovali šibki pospeški - na primer, v [14] uporabijo mejo za šibke pospeške do 5 g. V praksi morajo biti premiki roke ekstremni, da presežejo mejo 5 g. Poleg tega bi za večje napake algoritma morali takšen pospešek na senzorje vršiti dlje časa, ob prenehanju pa bi se napaka hitro zmanjšala. Z manjšim popravkom, kot ga opisujejo v [14], bi algoritem lahko uporabili tudi pri višjih pospeških. Enostavno bi določili mejo (recimo 5 g), nad katero se ignorirajo podatki s pospeškometra - s tem pospeškometer pri višjih pospeških ne bi imel vpliva na delovanje algoritma, bi pa zelo hitro po zmanjšanju pospeška izničil napako, ki se je pojavila v času višjih pospeškov. Podobno bi lahko storili pri zelo nizkih pospeških (kadar gre za prosti pad je s pomočjo pospeškometra nemogoče določiti smer gravitacije).

S pomočjo pospeškometra lahko tako zelo dobro odpravljamo napake za naklon, ki nastanejo pri integriranju podatkov z žiroskopa. Ker pospeškometer detektira smer gravitacije, ne more zaznati rotacije okrog gravitacijske osi (azimuta). Tako moramo za popravek napak žiroskopa pri rotaciji okrog gravitacijske osi uporabiti še magnetometer, ki detektira magnetno polje Zemlje.

Osnovna 1D oblika komplementarnega filtra, ki združi podatke z žiroskopa

ter drugega senzorja, ki so lahko podatki s pospeškometra ali z magnetometra (odvisno od osi, okrog katere želimo izračunati rotacijo):

$$y[n] = (1 - \alpha) * (y[n-1] + gyro * dt) + \alpha * (sensor), \tag{3.1}$$

pri čemer je

α : časovni parameter glajenja (ponavadi $\alpha < 0.1$),

y[0] : začetna inicializacija,

y[i], za $i > 0$: izračun i-te rotacije,

gyro : podatek z žiroskopa,

dt : časovni interval med dvema meritvama,

sensor : podatek s pospeškometra ali magnetometra.

V enačbi 3.1 je spremenljivka *sensor* kót, izračunan iz podatkov s senzorja. Kadar računamo azimut (oz. rotacijo okrog osi z), uporabimo podatke z magnetometra. Magnetno polje, ki ga zazna magnetometer, moramo projicirati na ravnino, katere normala je gravitacija, ter v tej ravnini izračunati azimut (azimut je kót med smerjo magnetnega polja in smerjo, v katero je obrnjen senzor). Podobno računamo tudi naklon, kjer dobimo podatke o smeri gravitacije s pospeškometra. V tem primeru računamo kót med koordinatno osjo -z in projekcijo gravitacijskega vektorja na ravnino z normalo v smeri y-osi oz. na ravnino z normalo v smeri x-osi (odvisno od tega, okrog katere osi želimo računati rotacijo).

Enačba 3.1 je splošna in omejena na eno dimenzijo (za rotacije okrog ene osi). Da bi enostavno uporabili tri enačbe 3.1, vsako za rotacijo okrog ene osi, ne bi bilo učinkovito. Podatki z žiroskopa rotirajo objekt okrog njegovega lokalnega koordinatnega sistema, medtem ko je potrebno popravke s pospeškometra in žiroskopa uporabiti za rotacijo okrog osi globalnega koordinatnega sistema. V naslednjem poglavju je opisan moj algoritem za uporabo splošne enačbe 3.1 v 3D prostoru z uporabo podatkov s treh uporabljenih senzorjev - žirskopa, pospeškometra in magnetometra.

3.3 Algoritem

V tem poglavju je opisan algoritem, ki sem ga razvil za uporabo komplementarnega filtra v 3D prostoru z uporabo treh senzorjev (žiroskopa, pospeškometra in magnetometra) pri računanju prostorske rotacije objekta ob predpostavki, da delujejo na objekt šibki pospeški.

Opis v algoritmu uporabljenih spremenljivk in oznak:

M[i] : rotacijska matrika objekta v i-tem koraku,

g[i], a[i], m[i] : vektorji podatkov z žiroskopa, pospeškometra in magnetometra v i-tem koraku,

indeks x v izrazu $g[i]_x$: označuje komponento x vektorja $g[i]$,

strešica v izrazu $\hat{g}[i]$: označuje normaliziran vektor $g[i]$,

funkcija $fromEuler("XYZ", x, y, z)$: kreira rotacijsko matriko, kjer je prvi parameter zaporedje rotacijskih matrik, ki rotirajo okrog ene osi (podrobnosti v dodatku B),

funkcija $angle(a, b)$: vrne kot θ med vektorjema a in b ($cos(\theta) = \frac{a \cdot b}{|a||b|}$),

α : časovna konstanta (pri realizaciji algoritma sem uporabil vrednost 0.04).

Algoritmu sledi natančna razlaga in opis vsake operacije.

Algoritem

1: $M[0] \leftarrow I$
2: **for** $i = 1$ to end **do**
3: $M[i] \leftarrow M[i-1] \times fromEuler("XYZ", (1-\alpha)g[i]_x, (1-\alpha)g[i]_y, (1-\alpha)g[i]_z)$
4: $a_g[i] \leftarrow M[i] \times a[i]$
5: $dx \leftarrow angle(a_g[i], (0, a_g[i]_y, a_g[i]_z))$
6: $dy \leftarrow angle(a_g[i], (a_g[i]_x, 0, a_g[i]_z))$
7: **if** $a_g[i]_x > 0$ **then**
8: $dx = -dx$
9: **end if**

10: **if** $a_g[i]_y < 0$ **then**
11: $dy = -dy$
12: **end if**
13: $M[i] = fromEuler(\text{"}XYZ\text{"}, \alpha * dy, \alpha * dy, 0) \times M[i]$
14: $s = M[i] \times (\hat{a}[i] \times \hat{m}[i])$
15: $dz = angle(s, (1, 0, 0))$
16: **if** $s_y > 0$ **then**
17: $dz = -dz$
18: **end if**
19: $M[i] = fromEuler(\text{"}XYZ\text{"}, 0, 0, \alpha * dz) \times M[i]$
20: **end for**

Razlaga algoritma

1: Inicializacija začetne matrike z identiteto velikosti 3×3.

2: Začetek zanke, ki se sprehodi skozi vse podatke.

3: Izračun nove rotacije glede na podatke z žiroskopa - uporabimo množenje z desne, kar nam rotira objekt okrog lokalnega koordinatnega sistema.

4: Preslikava podatkov pospeškometra z njegovega lokalnega koordinatnega prostora v globalni koordinatni prostor.

5, 6: Izračun popravkov kotov (dx in dy) glede na projekcijo gravitacije na ravnino z normalo v smeri x-osi oz. y-osi in trenutne rotacije.

7 - 12: Ker funkcija *angle* s prejšnjih dveh vrstic vrne absolutno vrednost kota, moramo narediti popravek glede na smer rotacije.

13: Uporaba izračunanih popravkov pospeškometra - uporabimo množenje z leve, kar nam rotira objekt okrog globalnega koordinatnega sistema.

14: Vektorski produkt ($\hat{a}[i] \times \hat{m}[i]$) vrne vektor, ki je v ravnini z normalo v smeri gravitacije (v ravnini Zemljine površine) in je usmerjen proti vzhodu glede na smeri neba. Vektor s je preslikava vektorja s prejšnje trditve v globalni koordinatni sistem.

15: dz je kot med vektorjem s s prejšnjega koraka in koordinatno osjo x, ki je v modelu privzeto usmerjena proti vzhodu (y pa proti severu).

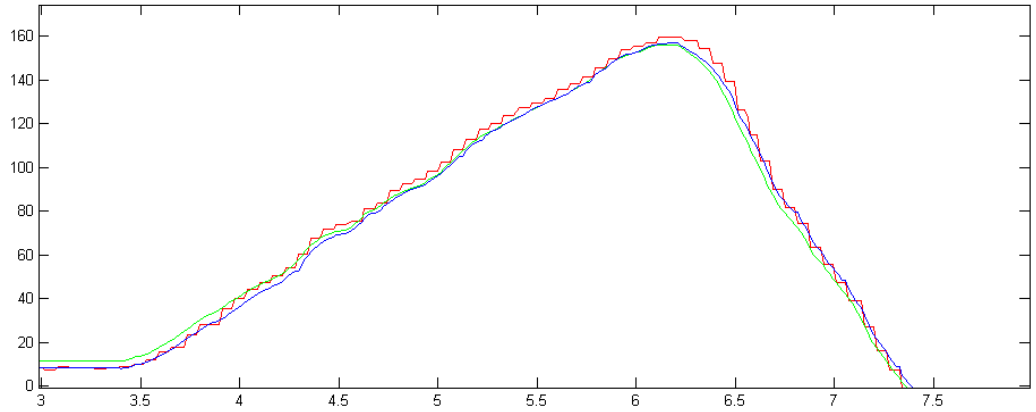
19: Uporaba izračunanega popravka s pospeškometra - uporabimo množenje z leve, kar nam rotira objekt okrog globalnega koordinatnega sistema.

3.4 Analiza algoritma

Algoritem z razdelka 3.3 sem analiziral s primerjavo dveh preprostih algoritmov, ki sem ju že omenil v razdelku 3.1. Prvi preprosti algoritem za izračun naklona uporabi podatke s pospeškometra. Te podatke interpretira kot gravitacijski pospešek in izračuna naklon gravitacije glede na senzor - s tem dobimo predznačen naklon senzorja. Drugi preprosti algoritem uporabi podatke z žiroskopa in jih integrira.

Na sliki 3.2 je prikazan graf z nakloni, kot jih izračuna določen algoritem. Rdeče je obarvan izhod algoritma z uporabo podatkov samo s pospeškometra, zeleno izhod algoritma z uporabo podatkov samo z žiroskopa ter z modro izhod algoritma z razdelka 3.3. Na x-osi je prikazan čas v sekundah, na y-osi pa naklon v stopinjah. Za zajem podatkov sem uporabil v naprej določene gibe (vsakemu sledi vrnitev v začetno lego) in sicer najprej naklon v pozitivno smer za 90° (v času 0 do 3), nato naklon v isto smer za 180° (v času 3 do 7), nato naklon v nasprotno smer za 90° (v času 8 do 11) ter nato naklon v to smer za 180° (v času 12 do 16) in na koncu v času 16 do 20 intenzivno tresenje senzorja brez spremembe naklona.

Z grafa so razvidne lastnosti algoritmov. Podatki s pospeškometra so šumni in na delovanje algoritma vpliva vsak pospešek na senzor - zaradi tega algoritem v času 16 do 20 zazna naklon, čeprav ga v resnici ni. Algoritem, ki samo integrira podatke z žiroskopa, napako akumulira in rezultat drsi s pravilne lege. Na grafu je to prikazano z zeleno barvo. Za to napako je značilno, da vpliva na vse kasnejše izhode algoritma, kar se odraža v odmaknjenosti zelenega grafa. V času 16 do 20 žiroskop za razliko od pospeškometra ne zazna rotacij, kar je pravilno. Algoritem z razdelka 3.3 (prikazan z modro barvo) z združevanjem podatkov omeji vpliv šuma in pospeška ter odpravlja akumulacijsko napako žiroskopa.

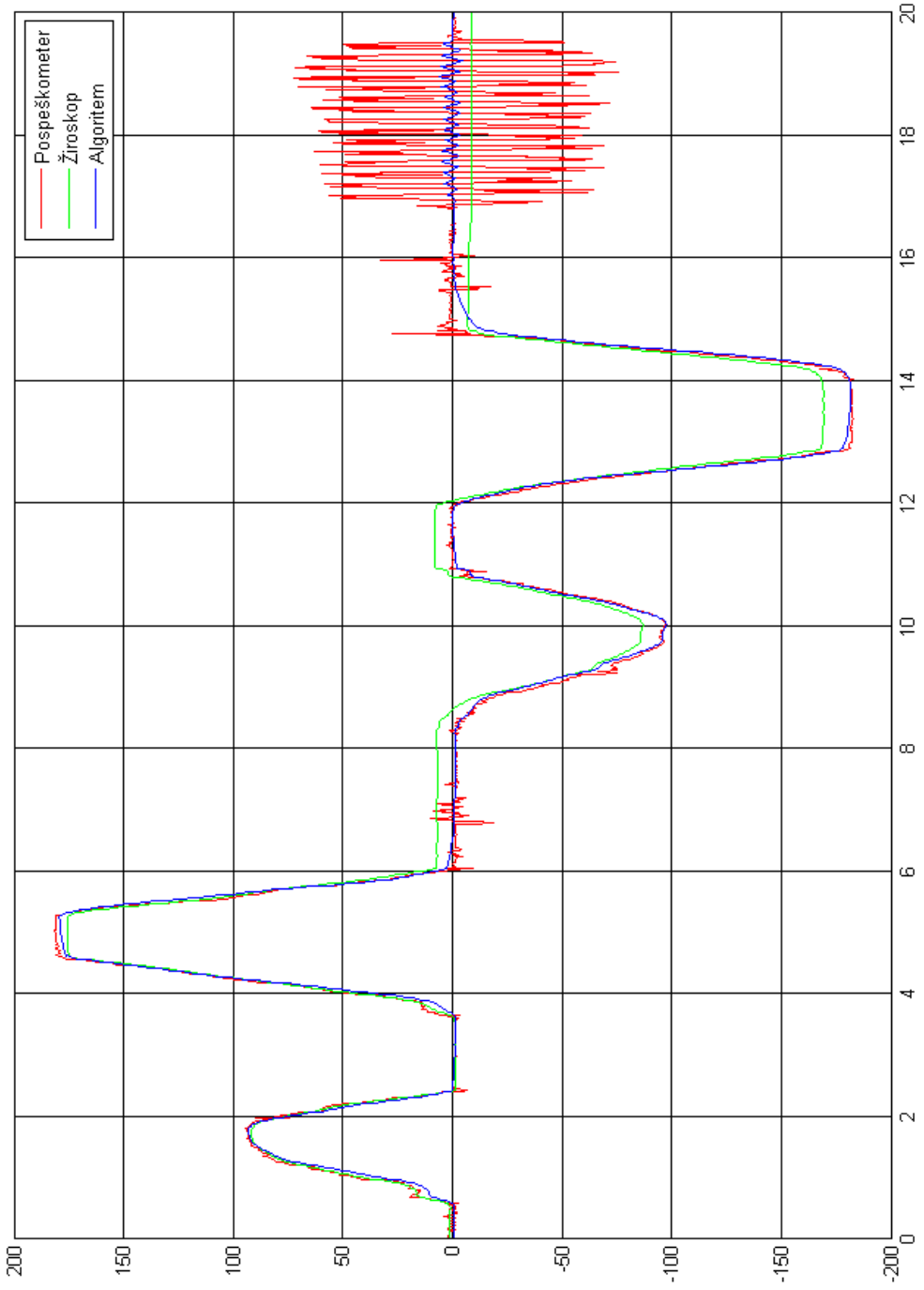


Slika 3.1: Primerjava algoritmov z uporabo podatkov samo z magnetometra (rdeč graf), samo z žiroskopa (zelen graf) ter uporabo algoritma z razdelka 3.3 (moder graf).

Podobno velja pri računanju azimuta, vendar v tem primeru akumulacijsko napako žiroskopa odpravlja magnetometer. Problem magnetometra za razliko od pospeškometra je v tem, da so njegove meritve počasnejše in tako dobimo stopničast graf, prikazan na sliki 3.1 z rdečo barvo. Na tem grafu je od časa 3.5 do 6 prikazano rotiranje v smeri urinega kazalca ter nato rotiranje v obratni smeri.

3.5 Kalibracija senzorjev

Za pravilno in dobro delovanje algoritma z razdelka 3.3 morajo biti vhodni podatki kar se da natančni. Žiroskop in pospeškometer sta mehanska in se zaradi tega po določenem času njuni mehanski deli rahlo spremenijo in je potrebno narediti kalibracijo. Magnetometer pa ima še več težav, saj na magnetno polje, ki ga zaznava, vplivajo vsi feromagnetni materiali v njegovi okolici. Tako ga je potrebno kalibrirati za vsak prostor, kjer ga uporabljamo, posebej. Poleg tega se napake pri nekalibriranem magnetometru najbolj vidno odražajo pri delovanju algoritma. Kalibriranje magnetometra je najbolj



Slika 3.2: Primerjava algoritmov z uporabo podatkov samo s pospeškometra (rdeč graf), samo z žiroskopa (zelen graf) ter uporabo algoritma z razdelka 3.3 (moder graf). Graf je prikazan navpično.

zapleten in tudi najbolj pomemben postopek v primerjavi s pospeškometrom in žiroskopom.

3.5.1 Kalibracija pospeškometra

S kalibracijo pospeškometra želimo doseči, da bodo podatki z vseh treh osi pravilno translirani in skalirani. Torej, ko deluje gravitacija v mirovanju senzorja samo na eno os v določeni smeri, mora senzor v tej smeri zaznati silo 1 g, za drugi dve osi pa senzor ne sme zaznati sile pospeška - kot je prikazano na sliki 2.4.

Za uspešno kalibracijo potrebujemo ravno površino, na katero v navpični smeri deluje gravitacija. Na to površino položimo senzor in ga obračamo - za vsako izdem šestih ploskev moramo shraniti izhod s senzorja.
Naj bodo:
x_i sila, ki jo zazna senzor v smeri x-osi za i-ti položaj,
x_s faktor skaliranja x-osi,
x_t vrednost transliranja.
Položaji:
1 - kjer je senzor obrnjen na x-ploskev (ploskve so poimenovane na sliki 2.3),
2 - kjer je senzor obrnjen na $-x$-ploskev,
3 - 6 - ostali položaji.
Tedaj velja:

$$x_s * (x_1 - x_2) = 2g \tag{3.2a}$$

$$x_1 + x_2 + 2 * x_t = 0g \tag{3.2b}$$

$$x_3 + x_t = 0g \tag{3.2c}$$

$$x_4 + x_t = 0g \tag{3.2d}$$

$$x_5 + x_t = 0g \tag{3.2e}$$

$$x_6 + x_t = 0g \tag{3.2f}$$

Iz enačbe 3.2a dobimo rešitev za x_s, iz drugih enačb pa dobimo predoločeni sistem, iz katerega izračunamo najboljši približek za x_t. S podobnim

Slika 3.3: Eno-osni instrument.

Slika 3.4: Tri-osni instrument.

postopkom nato izračunamo še faktor skaliranja in vrednost transliranja za osi y in z.

3.5.2 Kalibracija žiroskopa

Za najbolj zanesljivo kalibracijo žiroskopa bi potrebovali natančen rotirajoči se instrument. Instrument mora imeti nadzor nad hitrostjo in smerjo rotiranja in mora zelo natančno prikazat njegovo kotno hitrost. V praksi obstajajo manjši instrumenti z rotiranjem ene osi (prikazano na sliki 3.3), ter večji simulatorji gibanja z možnostjo rotiranja vseh treh osi (prikazano na sliki 3.4) [13].

Za grobo kalibracijo lahko uporabimo tudi algoritmični pristop [8]. S tem pristopom lahko odpravimo fiksno pristransko napako - napako, ki je vidna ob mirovanju senzorja (kljub mirovanju lahko žiroskop zazna rotiranje). To pristranskost lahko enostavno odpravimo tako, da naredimo nekaj meritev ob mirovanju senzorja in dobljeno povprečno vrednost meritev odštejemo od nadaljnjih meritev (translacija podatkov). Za faktor skaliranja podatkov lahko dobimo grobo oceno brez dodatnih instrumentov tako, da ročno čimbolj enakomerno rotiramo senzor za vnaprej določen kot in podatke integriramo. Postopek je dobro večkrat ponoviti.

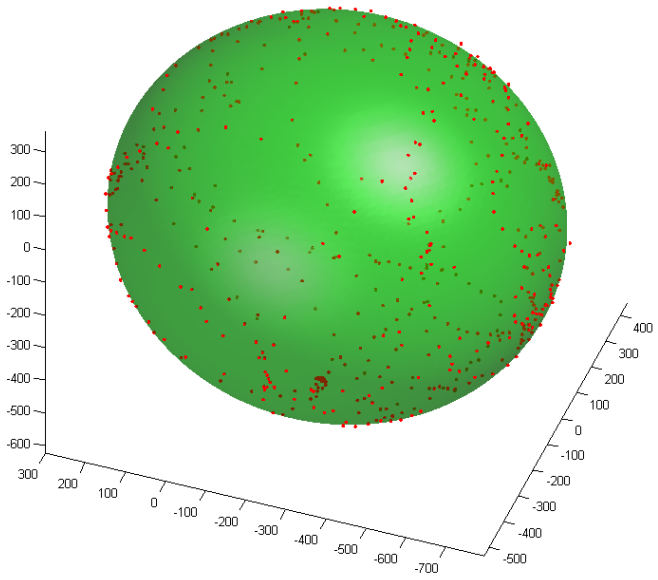
3.5.3 Kalibracija magnetometra

Kalibracija magnetometra je zelo pomembna, saj nanj delujejo vsi feromagnetni elementi v okolici senzorja. Ti elementi pačijo magnetno polje, ki ga magnetometer zaznava, in s tem prihaja do napak v meritvah. Imamo dve vrsti motenj in sicer motnje elementov, ki se rotirajo in premikajo zraven senzorja (stalna pristranskost, angl. hard-iron), ter motnje drugih elementov, ki se ne premikajo (trenutna pristranskost, angl. soft-iron) - na primer jeklena konstrukcija hiše, v kateri premikamo senzor. Če ne bi bilo nobenih motenj magnetnega polja, bi v vsaki legi magnetometra zaznali enako veliko magnetno silo. Torej, če bi grafično prikazali vse meritve v 3D prostoru (vsaka meritev naj predstavlja eno točko), bi kot rezultat dobili točke na krogli s središčem v točki (0, 0, 0) in radijem v velikosti sile magnetnega polja.

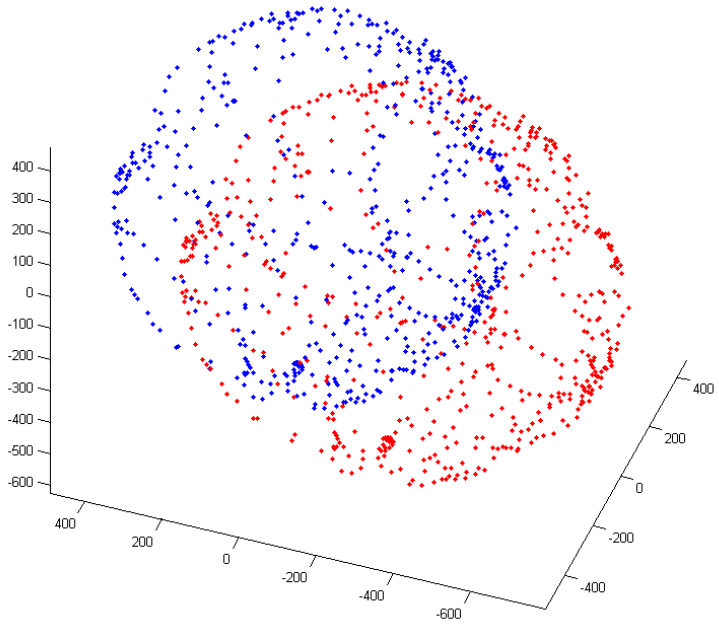
Stalna pristranskost konstantno vplivajo na magnetno polje, ki ga zaznava senzor. Ta vrsta motenj se odraža v tem, da nam središče navidezne krogle translira s središča koordinatnega sistema. Trenutna pristranskost pa nam kroglo popači v elipsoid, ki je lahko tudi rotiran v prostoru. V praksi se stalna in trenutna pristranskost seštejeta ter dobimo rotiran in transliran elipsoid. Poleg tega je pri podatkih prisoten tudi šum in možnost vpliva drugih nestatičnih feromagnetnih elementov - zaradi tega točke ne ležijo neposredno na plašču elipsoida in moramo skozi točke aproksimirati elipsoid (v algoritmu bomo uporabili metodo najmanjših kvadratov). Primer dobljenih podatkov z uporabljenega senzorja so na sliki 3.5.

S kalibracijo želimo dobiti preslikavo, ki bo dobljeni aproksimirani elipsoid preslikala v kroglo s središčem v točki (0, 0, 0) in radijem velikosti magnetnega polja. Na sliki 3.6 je prikazan rezultat preslikave, kjer se rdeče točke iz okolice plašča elipsoida preslikajo v modre točke (ki so v okolici plašča krogle s središčem v (0, 0, 0) in radijem v velikosti magnetnega polja).

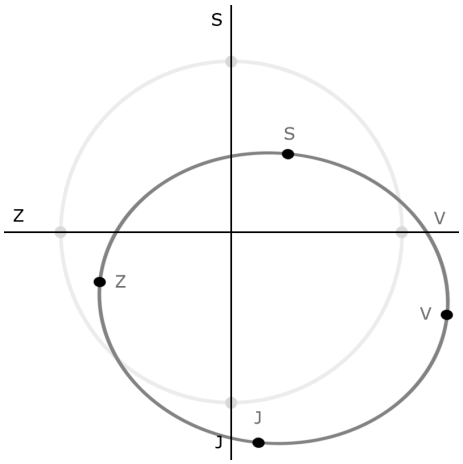
Slika 3.7 prikazuje en izmed primerov delovanje magnetometra v ravnini (za lažji prikaz in razumevanje), če kalibracija ne bi bila izvedena. S slike je na primer razvidno, da bi magnetometer v tem primeru ob rotaciji proti vzhodu detektiral rotacijo v smeri severo-vzhoda. V ekstremnem primeru (ob velikem



Slika 3.5: Podatki z magnetometra - elipsoid.



Slika 3.6: Rdeča: podatki z magnetometra. Modra: preslikani podatki.



Slika 3.7: Primer napačnega izračuna azimuta pri nekalibriranim magnetometru.

vplivu feromagnetnih elementov iz ene smeri) bi se lahko zgodilo tudi to, da bi se elipsa v celoti nahajala zgolj v enem izmed kvadrantov koordinatnega sistema (na primer v 1. kvadrantu) - v tem primeru bi magnetometer za vsako rotacijo zaznal rotacijo samo v smeri med severom in vzhodom ter bil tako popolnoma neuporaben oz. celo škodljiv pri uporabi v algoritmu.

Algoritem za kalibracijo magnetometra

Naj bo funkcija $[center,\ r,\ evecs,\ pars] = ellipsoid\_fit([x\ y\ z])$ z vira [9].
Vhod: $[x\, y\, z]$, kjer so x, y in $z \in R^{n\times 1}$ (vektorji podatkov z magnetometra).
Izhodi funkcije $ellipsoid\_fit()$:
$center \in R^3$: središče elipsoide,
$r \in R^3$: radiji elipsoide,
$evecs \in R^{3\times 3}$: ortogonalna matrika smeri radijev v stolpcih,
$pars$: parametri za algebraično obliko zapisa elipse.
Izhod algoritma: preslikava M in središče $center$.

1: $[center,\ r,\ evecs,\ pars] = ellipsoid\_fit([x\ y\ z])$;
2: $D = [x - center(1), y - center(2), z - center(3)]'$;

3: $M_s = inv(\begin{bmatrix} r(1) & 0 & 0 \\ 0 & r(2) & 0 \\ 0 & 0 & r(3) \end{bmatrix}) \times min(r);$

4: $M = evecs \times M_s \times evecs' \times D;$

Preslikava M elipsoid translira v koordinatno izhodišče, ga nato zarotira, da so njegove osi poravnane s koordinatnimi osmi, nato skalira elipsoid, da nastane krogla ter na koncu še nastalo kroglo zarotira nazaj v položaj, kot je bil zarotiran elipsoid. Ko imamo izračunan *center* in preslikavo M, za vsak podatek $d \in R^3$ iz magnetometra izračunamo kalibriran podatek:

1: $d_c = M \times (d - center)$

Poglavje 4

Upravljanje robotske roke

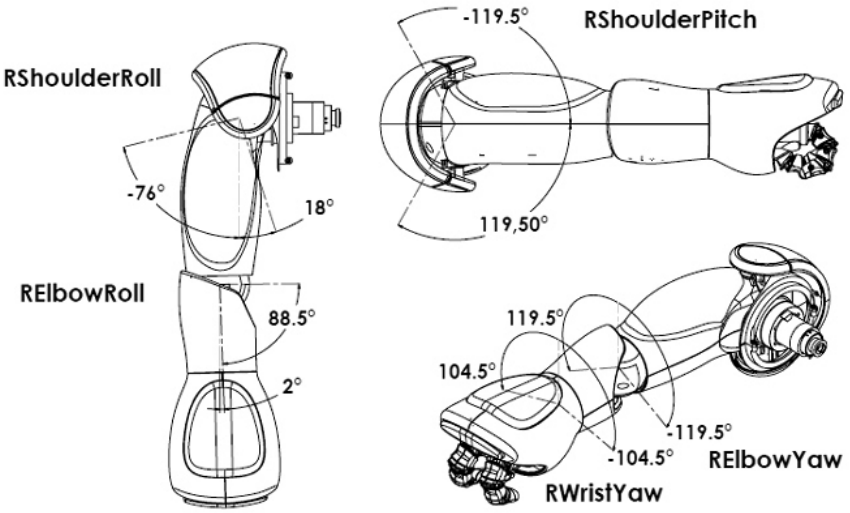
Z vmesnikom za zajem podatkov s poglavja 2 in uporabo algoritma na teh podatkih s poglavja 3 dobimo rotacijsko matriko za nadlaket in podlaket. V tem poglavju je opisano, kako lahko s temi podatki upravljamo roko robota Nao ter kje morajo biti locirani senzorji na upraviteljevi roki. Pred tem pa so opisane karakteristike roke robota Nao in primerjava s človeško roko.

4.1 Roka robota Nao v primerjavi s človeško roko

Robot Nao je programabilen humanoidni robot, prikazan na sliki 4.1. Vgrajen ima procesor Intel ATOM 1,6 gHz, ki poganja Linux. Robot izvaja premike z vgrajenimi motorji v sklepih, skupno ima 25 prostostnih stopenj. Vgrajene ima tudi različne senzorje, kot so kamera, mikrofon, sonar itd. Robot ima vgrajeno tudi povezavo WiFi, preko katere ga lahko upravljamo in dobivamo informacije s senzorjev [1].

Roka robota Nao ima tri različne sklepe - ramo, komolec in zapestje. Rama in komolec imata po 2 prostostni stopnji, zapestje pa eno - torej skupno 5 prostostnih stopenj za vsako roko, prikazanih z njihovimi omejitvami na sliki 4.2. Robot ima tudi možnost enostavnega upravljanja prstov (stisk in sprostitev pesti).

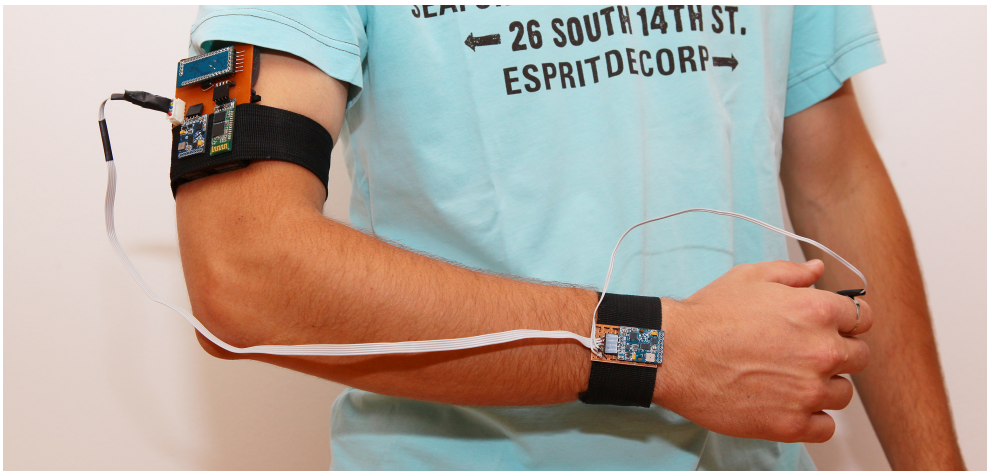
Slika 4.1: Robot Nao.



Slika 4.2: Roka robota Nao z imeni in omejitvami gibljivih delov.

V primerjavi s človeško roko je nekaj bistvenih razlik. Prva bistvena razlika je v razponu, za koliko stopinj se lahko gibajo gibljivi deli - ljudje imamo za določene sklepe večje razpone. Druga razlika je v številu in razporeditvi prostostnih stopenj po sklepih. Ljudje imamo 3 prostostne stopnje v ramenu in zapestju ter eno v komolcu - torej skupno 7 prostostnih stopenj. Robot nima tako gibljivega zapestja - ne zmore upogniti zapestja gor in dol ter levi in desno, tako bom te gibe v rešitvi ignoriral. Razlika je tudi v razporeditvi prostostnih stopenj med sklepi. Robot Nao ima dve prostostni stopnji v komolcu, človek pa samo eno. V bistvu druga prostostna stopnja (*ElbowYaw*) nadomešča eno prostostno stopnjo v ramenu, kjer ima robot samo dve, človek pa tri. Torej to, kar robot doseže z rotiranjem v *ElbowYaw*, človek doseže z rotacijo v ramenu - s tem so razvijalci robota Nao poenostavili sklep v ramenu, ki je pri ljudeh bistveno bolj zapleten in bi bila sicer realizacija robota bolj zapletena.

Ne glede na zgornje omejitve, lahko robot Nao doseže skoraj vse položaje roke (razen nekaj ekstremnih položajev). Ker moramo upravljati 5 prostostnih stopenj lahko s pravilno postavitvijo senzorjev upravljamo robotsko



Slika 4.3: Prikaz uporabe vmesnika za zajem podatkov na roki upravitelja.

roko z dvema sistemoma 3-osnih senzorjev. Kot bomo videli, roka robota pri tem ne bo popolnoma imitirala človeške roke (rotacija nadlakti robota ne bo enaka rotaciji človeške nadlakti, kar pa za upravljanje ni pomembno), bo pa imela popolnoma enako usmeritev in rotacijo zapestja, kar je bistveno pri upravljanju. Za lažjo predstavo o delovanju algoritma je na sliki 4.3 prikaz uporabe vmesnika za zajem podatkov na roki upravitelja.

Razlike v primerjavi s človeško roko so tudi pri prstih, ki so pri ljudeh neprimerljivo bolj zapleteni. Robot Nao nam omogoča samo stisk in sprostitev pesti. Tako lahko zapestje upravljamo z navadnim potisnim gumbom - kadar je gumb stisnjen, robot Nao stisne zapestje, sicer pa ga sprosti.

4.2 Uporaba podatkov

Za lažje razumevanje in računanje rotacij v sklepih moramo najprej definirati usmerjenost koordinatnih osi robota in njegov osnovni položaj ter usmerjenost koordinatnih osi senzorjev glede na ta koordinatni sistem. Predpostavimo, da robot stoji pokončno s pogledom naravnost. Uporabimo desno-sučni koordinatni sistem, ki ima y-os v smeri pogleda robota, x-os na desno ter z-os navpično navzgor. Osnovni položaj roke robota naj bo takrat, ko so koti

v njegovih sklepih nastavljeni na 0 oz. čim bližje 0 za kote, kjer robot ne more doseči kota 0. V tem primeru ima robot roko iztegnjeno naravnost predse (usmerjeno v smeri y-osi). Kadar so senzorji pripeti na roki so usmerjeni tako, da je x-os njihovega lokalnega koordinatnega sistema usmerjena v smeri roke, y-os na levo ter z-os navpično navzgor. Zaradi tega moramo rotacijski matriki zarotirati za $-\pi/2$ okrog z-osi, da dobimo poravnane koordinatne osi. Algoritem iz poglavja 3 je narejen tako, da je osnovni položaj dosežen takrat, ko je globalno gledano x-os lokalnega koordinatnega sistema senzorjev usmerjena proti magnetnemu severu Zemlje. Zaradi rotacije za poravnavo koordinatnih osi robota in senzorjev dobimo tako osnovni položaj senzorjev in robota v primeru, kadar sta v njunem lokalnem koordinatnem sistemu y-os robota in x-os senzorjev usmerjena proti magnetnemu severu - torej kadar ima človek predse iztegnjeno roko usmerjeno proti severu in prav tako tudi robot. V nadaljevanju tega poglavja je opisan tudi postopek kalibracije začetnega položaja, ki spremeni osnovni položaj usmerjenosti na katero koli drugo usmeritev. Zaradi enostavnosti bomo v tem poglavju vedno uporabili osnovni položaj z usmerjenostjo proti severu.

Pri izračunu kotov bomo uporabili funkcijo $toEuler(\text{"}XYZ\text{"},\ M)$, ki za rotacijsko matriko M vrne trojico Eulerjevih kotov $(x,\ y,\ z)$, dobljenih iz rotacij okrog osi v vrstnem redu, ki ga določa prvi parameter (podrobnosti v dodatku B). Velja:

$$toEuler(\text{"}XYZ\text{"},\ fromEuler(\text{"}XYZ\text{"},\ x,\ y,\ z)) = (x,\ y,\ z) \qquad (4.1)$$

4.2.1 Izračun pregibov za nadlaket

Kot smo videli v razdelku 4.1, se ramenski obroč pri ljudeh in robotu Nao razlikujeta v prostostnih stopnjah. Robot Nao ima v ramenskem sklepu dve prostostni stopnji, kar nam omogoča, da lahko dosežemo enako usmerjenost nadlakti kot pri človeku, ne moremo pa nadlakti popolnoma imitirati, saj ima človeški ramenski sklep tri prostostne stopnje.

Robot Nao ima v ramenskem sklepu dva pregiba in sicer *ShoulderPitch* in *ShoulderRoll*. *ShoulderPitch* vedno rotira roko okrog x-osi koordinatnega

sistema robota (kot smo to definirali na začetku poglavja 4.2). *ShoulderRoll* pa je odvisen od položaja nadlakti in sicer rotira okrog z-osi lokalnega koordinatnega sistema nadlakti (lokalni koordinatni sistem nadlakti naj se v osnovnem položaju roke ujema s koordinatnim sistemom robota).

Ker *ShoulderPitch* rotira okrog osi referenčnega koordinatnega sistema, moramo to rotacijo narediti na koncu in ker robot Nao ni zmožen rotacije okrog y-osi, moramo to rotacijo narediti prvo. Ker je roka robota v osnovnem položaju (kadar so koti enaki 0) usmerjena v smeri y-osi, ta rotacija ne bo spremenila usmerjenosti roke robota, ampak samo njeno rotacijo - tako bosta usmerjenost nadlakti robota in usmerjenost človeške nadlakti enaki, rotacija nadlakti pa niti ni pomembna. Tako dobimo enačbo:

$$(-ShoulderPitch,\ \_,\ ShoulderRoll) \leftarrow toEuler(\text{"}XZY\text{"},\ M_N) \quad (4.2)$$

ShoulderPitch mora biti predznačen, saj ima Nao na desni roki obrat v pozitivni smeri *ShoulderPitch* definiran kot obrat v smeri urinega kazalca.

4.2.2 Izračun pregibov za podlaket

Izračun rotacije nadlakti je nekoliko bolj zapleten. S pozicioniranjem senzorjev na človeško roko tik pred zapestjem dosežemo to, da dobimo z njegove rotacijske matrike tako usmerjenost podlakti kot tudi rotacijo zapestja. Naj se ta rotacijska matrika imenuje M_P. S to rotacijsko matriko upravljamo nagibe v *ElbowRoll*, *ElbowYaw* in *WristYaw*. Kadar premikamo iztegnjeno roko in je ne krčimo v komolcu in rotiramo v zapestju, sta rotacijski matriki M_P in M_N skoraj enaki (manjša razlika nastane zaradi napake senzorjev in pozicioniranja na roki). Ko pa apliciramo podatke na robota Nao, se spreminjata samo kota *ShoulderPitch* in *ShoulderRoll*, medtem ko so nagibi v *ElbowRoll*, *ElbowYaw* in *WristYaw* enaki 0. Zaradi tega moramo najprej matriko M_P zarotirati v obratni smeri, kot smo zarotirali ramenski sklep robota. Popravljena rotacija (*ShoulderPitch* ni predznačen, saj smo že kot rezultat vzeli

nasprotno vrednost):

$$M_P \leftarrow fromEuler(\text{"}YZX\text{"},\ ShoulderPitch,\ 0,\ -ShoulderRoll) \times M_P \tag{4.3}$$

Rotacija okrog y-osi je 0, saj te rotacije nismo aplicirali na robota in je robot tudi ni zmožen izvesti.

Referenčni koordinatni sistem podlakti naj bo koordinatni sistem nadlakti, lokalni koordinatni sistem podlakti pa naj bo v stanju osnovnega položaja roke enak koordinatnemu sistemu nadlakti. Prav tako naj bo osnovni položaj podlatki usmerjen v smeri y-osi referenčnega koordinatnega sistema. V tem kontekstu *ElbowYaw* rotira okrog y-osi referenčnega koordinatnega sistema (ta gib nadomešča rotacijo v ramenu), *ElbowRoll* rotira okrog z-osi lokalnega koordinatnega sistema podlakti (ta pregib je enak človeškemu pregibu komolca) ter *WristYaw* rotira okrog y-osi prav tako lokalnega koordinatnega sistema podlakti (enako rotaciji zapestja pri človeku).

Pri algoritmu sem se omejil na uporabo Tait-Bryan različico Eulerjevih kotov (in ne klasičnih Eulerjevih kotov), torej samo na uporabo funkcije $toEuler(\text{"}XZY\text{"},\ M)$ z različnim vrstnim redom rotacij okrog vseh treh osi. Če bi uporabljali klasične Eulerjeve kote, bi lahko naredili rotacijo "YZY", tako pa moramo naredi manjši trik. Rotacija *ElbowYaw* je rotacija okrog referenčnega koordinatnega sistema, torej jo bomo naredili na koncu in sicer okrog y-osi. Rotacija *WristYaw* na robotu spremeni samo rotacijo zapestja in ne vpliva na komolec in njuna giba pri robotu. Tako je ta rotacija neodvisna in jo lahko naredimo na začetku. Ker pa je to rotacija okrog y-osi lokalnega koordinatnega sistema, ki je v osnovnem položaju enaka y-osi referenčnega koordinatnega sistema, moramo pred izračunom naredi rotacijo in y-os lokalnega koordinatnega sistema podlakti usmeriti v x-os referenčnega koordinatnega sistema (pravzaprav zaradi karakteristike desne roke robota v smer osi $-x$). Tako moramo na začetku narediti rotacijo za $-\pi/2$ okrog z-osi, ki jo bomo po izračunu prišteli rezultatu rotacije okrog z-osi. Sedaj moramo uporabiti funkcijo, ki sprva naredi rotacijo okrog x-osi, nato z-osi ter na koncu y-osi (vedno okrog osi referenčnega koordinatnega sistema),

ter na koncu rezultatu rotacije okrog z-osi prišteti še $\pi/2$. Priprava (4.4a), izračun (4.4b) in popravek (4.4c):

$$M_P \leftarrow M_P \times fromEuler(\text{"}XYZ\text{"},\ 0,\ 0,\ -\pi/2) \tag{4.4a}$$

$$(-WristYaw,\ ElbowYaw,\ ElbowRoll) \leftarrow toEuler(\text{"}YZX\text{"},\ M_P) \tag{4.4b}$$

$$ElbowRoll \leftarrow ElbowRoll + \pi/2 \tag{4.4c}$$

4.2.3 Upravljanje robota Nao

Za upravljanje robota Nao sem uporabil ogrodje *NAOqi*, ki je na voljo tudi za programski jezik *Python*. *NAOqi* omogoča, da se povežemo z robotom ter ga tako upravljamo. Za samo upravljanje sklepov sem uporabil $ALMotionProxy :: setAngles()$. Funkcija $setAngles()$ sprejme parametra *names* in *angles* preko katerih določimo, kateri sklepi in v kakšen položaj naj se zarotirajo. Za to funkcijo je značilno, da ob klicanju ne blokira možnosti nadaljnjega klicanja te funkcije v času, ko se še izvaja ukaz na robotu. S tem nam funkcija omogoča, da jo kličemo pogosto ter nam zagotavlja zvezno hitrost in gladko gibanje - izvaja interpolacijo gibanja.

4.2.4 Kalibracija začetnega položaja

S kalibracijo začetnega položaja želimo doseči dve stvari in sicer spremeniti usmerjenost začetnega (osnovnega) položaja senzorjev ter odpraviti napako, ki nastane zaradi montaže senzorjev na roko (različne konstrukcije rok pri različnih ljudeh). S tem bomo veliko manj omejeni pri uporabi algoritma, ki bi sicer zahteval, da smo pri upravljanju robota vedno obrnjeni proti severu. Namesto kalibracije bi lahko uporabili tudi kompas na trupu upravljalca ki bi zaznaval usmerjenost trupa - v tem primeru bi se lahko upravitelj robota obračal med samim upravljanjem, kar pa brez uporabe dodatnega magnetometra ne bo mogoče.

Kalibracija poteka tako, da držimo roko iztegnjeno in usmerjeno predse ter med tem naredimo več meritev (pri dejanski kalibraciji sem uporabil 200 meritev). Tako dobimo več rotacijskih matrik za nadlaket in podlaket. Naj

bodo $\bar{x_n}$, $\bar{y_n}$ in $\bar{z_n}$ ter $\bar{x_p}$, $\bar{y_p}$ in $\bar{z_p}$ povprečne vrednosti kotov, dobljenih s funkcijo $toEuler("ZXY",\ M_i)$ iz vseh dvesto rotacijskih matrik za nadlaket in podlaket posebej.

Ker s to kalibracijo določamo usmerjenost, katere popravek je globalna rotacija ter odpravljamo anomalije zaradi montaže senzorjev, katerih popravek je lokalna rotacija, moramo to pri uporabi na podatkih ločiti. Za glavno usmerjenost (ki bo po kalibraciji določala osnovni položaj človeške roke) sem izbral usmerjenost podlakti (torej $\bar{z_p}$), saj se senzorji bolje prilegajo roki na podlakti. Pri sami implementaciji moramo upoštevati tudi to, da je potrebno narediti preslikavo med koordinatnim sistemom senzorjev in globalnim koordinatnim sistemom - narediti moramo rotacijo za $-\pi/2$ okrog z-osi z množenjem z desne (rotacija okrog lokalnega koordinatnega sistema). Zaradi tega moramo pri računanju popravka usmerjenosti prišteti $\pi/2$, da na koncu dosežemo željen osnovni položaj. Sprememba osnovnega položaja:

$$M_N \leftarrow fromEuler("XYZ",\ 0,\ 0,\ -\bar{z_p} + \pi/2) \times M_N \tag{4.5a}$$

$$M_P \leftarrow fromEuler("XYZ",\ 0,\ 0,\ -\bar{z_p} + \pi/2) \times M_P \tag{4.5b}$$

Nato sledi popravek anomalij senzorjev in preslikava v globalni koordinatni sistem. Pri rotaciji okrog z-osi moramo za podlaket upoštevati samo $-\pi/2$, pri nadlakti pa še razliko med spremembo osnovnega položaja ($\bar{z_p}$) in anomalije ($\bar{z_n}$):

$$M_N \leftarrow M_N \times fromEuler("ZYX",\ -\bar{x_n},\ -\bar{y_n},\ -(\bar{z_n} - \bar{z_p}) - \pi/2) \tag{4.6a}$$

$$M_P \leftarrow M_P \times fromEuler("ZYX",\ -\bar{x_p},\ -\bar{y_p},\ -\pi/2) \tag{4.6b}$$

Popravke z enačb 4.5 in 4.6 je potrebno narediti na začetku algoritma, pred računanjem pregibov za podlaket in nadlaket.

4.2.5 Algoritem

Spodaj opisani algoritem združuje vse korake, ki so opisani v razdelkih od 4.2.1 do 4.2.4 z referencami na posamezne enačbe v komentarjih.

Vhod: Rotacijski matriki M_N in M_P.
Izhod: Upravljanje robota Nao.

1: $M_N \leftarrow fromEuler(\text{"}XYZ\text{"},\ 0,\ 0,\ -\bar{z_p} + \pi/2) \times M_N$ (4.5)
2: $M_P \leftarrow fromEuler(\text{"}XYZ\text{"},\ 0,\ 0,\ -\bar{z_p} + \pi/2) \times M_P$
3: $M_N \leftarrow M_N \times fromEuler(\text{"}ZYX\text{"},\ -\bar{x_n},\ -\bar{y_n},\ -(\bar{z_n} - \bar{z_p}) - \pi/2)$ (4.6)
4: $M_P \leftarrow M_P \times fromEuler(\text{"}ZYX\text{"},\ -\bar{x_p},\ -\bar{y_p},\ -\pi/2)$
5: $(-ShoulderPitch,\ \_,\ ShoulderRoll) \leftarrow toEuler(\text{"}XZY\text{"},\ M_N)$ (4.2)
6: $M_P \leftarrow fromEuler(\text{"}YZX\text{"},\ ShoulderPitch,\ 0,\ -ShoulderRoll) \times M_P$ (4.3)
7: $M_P \leftarrow M_P \times fromEuler(\text{"}XYZ\text{"},\ 0,\ 0,\ -\pi/2)$ (4.4)
8: $(-WristYaw,\ ElbowYaw,\ ElbowRoll) \leftarrow toEuler(\text{"}YZX\text{"},\ M_P)$
9: $ElbowRoll \leftarrow ElbowRoll + \pi/2$
10: $setAngles([\text{'}RShoulderRoll\text{'},\ \text{'}RShoulderPitch\text{'},\ \text{'}RElbowRoll\text{'},\ \text{'}RElbowYaw\text{'},\ \text{'}RWristYaw\text{'}],\ [ShoulderRoll,\ ShoulderPitch,\ ElbowRoll,\ ElbowYaw,\ WristYaw])$

Opisani algoritem kličemo vsakič, ko dobimo novi matriki M_N in M_P. Matriki M_N in M_P kreiramo z algoritmom iz razdelka 3.3, ki dobi za vhod podatke z vmesnika za zajem podatkov.

4.3 Preverjanje učinkovitosti upravljanja robota

Za preverjanje učinkovitosti algoritma sem določil za večino ljudi preproste naloge: premik šahovske figure na drugo šahovsko polje, postavitev stolpa z uporabo šahovskih trdnjav in pisanje. S temi nalogami sem želel v praksi preizkusiti zahtevnost in natančnost vodenja roke ter njene omejitve. S pomočjo nalog sem prišel do zaključka, da vmesnik v manevrskem prostoru robota deluje dobro, zahteva pa začetno privajanje in trening upravitelja. Robot Nao ima za upravljanje roke precej omejen manevrski prostor - sku-

pna dolžina nadlakti in podlakti je 29 cm, z določenimi omejitvami pa se manevrski prostor še dodatno zmanjša.

Posnetek nekaterih preizkusov nalog je na priloženem elektronskem mediju v mapi *Multimedija*, v skrajšani verziji pa je dostopen tudi na povezavi http://youtu.be/JP2hjCmg3tY.

Prva naloga - premik šahovske figure

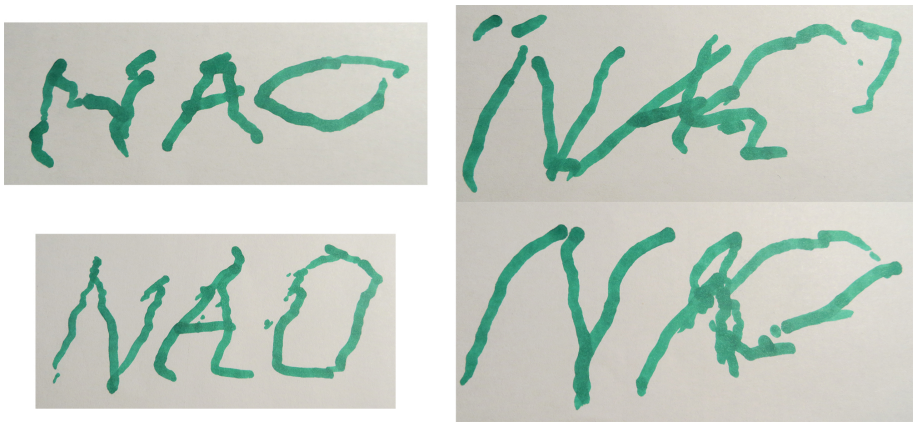
Premik šahovske figure zahteva prijem figure, premik na določeno lokacijo in izpust figure. Robot Nao s prijemom, premikom in izpustom figure v njegovem manevrskem prostoru ni imel nobenih težav in je zlahka postavil figuro na ciljno polje. Največji doseg za premik šahovske figure po polju samo s premikom roke robot doseže takrat, ko je polje samo nekaj centimetrov pod njegovim ramenom. Nekaj več možnosti za upravljanje bi lahko dobili s premikanjem trupa v bokih, vendar bi za to potrebovali dodatne senzorje na upravitelju.

Druga naloga - postavitev stolpa z uporabo šahovskih trdnjav

Postavitev stolpa z uporabo šahovskih trdnjav je dober testni primer, pri katerem lahko vidimo možnost natančnega upravljanja roke in njeno stabilnost. Pri tem z upravljanjem robota postavljamo šahovsko trdnjavo eno na drugo. Robot je bil v treh od petih preizkusov sposoben postaviti stolp s štirimi trdnjavami, ne da bi se stolp pri tem porušil. S postavitvijo treh trdnjav ni bilo težav, pri četrti trdnjavi pa se je pojavil problem zaradi omejitve manevrskega prostora. Pri postavitvi figure mora biti podlaket čim bolj vodoravna, (kadar je vodoravna, je trdnjava pokončna, sicer pa je poševna in nastanejo težave ob spustu), s tem pa se izgubi nekaj centimetrov potrebne višine.

Tretja naloga - preizkus v pisanju

Za preizkus pisanja sem si zadal napisati besedo NAO na list z uporabo flomastra. Pisanje je predstavljalo težavno nalogo, ki je potrebovala največ



Slika 4.4: Na levi primera bolj uspešnega ter na desni primera manj uspešnega poskusa pisanja z robotom Nao.

treninga. Po nekaj ponesrečenih poizkusov mi je uspelo napisati zadovoljiv napis. Na sliki 4.4 sta na levi prikazana dva bolj uspešna ter na desni dva manj uspešna poskusa. Po krajšem treningu vmesnik tako omogoča pisanje, ki je berljivo, vendar zahteva veliko potrpežljivosti in natančnosti pri upravljanju. Se je pa tudi pri pisanju pojavila omejitev manevrskega prostora in bi ob pisanju daljše besede bilo potrebno sproti prilagajati list, na katerega robot piše.

Poglavje 5

Sklepne ugotovitve

Razviti vmesnik za upravljanje roke robota Nao se je skozi zadane naloge v praksi izkazal za zelo dobrega. Omogoča natančno in stabilno upravljanje roke robota s premikanjem človeške roke na intuitivni način. Za zahtevnejša opravila, kot je na primer pisanje, je potrebno nekaj uvajanja in treninga upravitelja ter zahteva njegovo potrpežljivost in spretnost.

Pri razvoju mi je veliko težav povzročala elektronika, saj prvi prototipi niso bili dovolj zanesljivi za učinkovito delovanje in je prihajalo do pogostih prekinitev, kar je tudi oteževalo razvoj algoritmov za obdelavo podatkov. Problem zanesljivosti elektronskega vezja sem rešil z novim modulom s senzorji in izdelavo rezkane bakrene ploščice. Nekaj težav je bilo tudi pri uporabi rotacijskih matrik in uporabi funkcij, povezanih z njimi. Uporaba rotacijskih matrik zahteva dobro prostorsko in vizualizacijsko sposobnost.

Možnosti za izboljšave je še veliko, vendar zahtevajo tudi veliko dodatnega dela. Za še boljše delovanje algoritma bi bilo potrebno s senzorjev dobiti višjo frekvenco podatkov. Uporabljena ploščica s senzorji žal ne dopušča višjih hitrosti (čeprav so senzorji tega sposobni) in bi za višje hitrosti morali načrtovati svojo ploščico s senzorji. Zanimiva bi bila tudi primerjava uporabe Kalman filtra ter primerjava rezultatov algoritma za izračun rotacije z referenčnim sistemom (recimo Animazoo obleko, drugega podobnega sistema, Kinecta oz. podobnega sistema za 3D zajem podatkov). Pri uporabi

podatkov na robotu Nao bi bilo dobro izboljšati tudi upravljanje gibov, ki jih robot ne more doseči. Trenutno se prevelike oz. premajhne vrednosti kotov enostavno ignorirajo, vendar to povzroči nekatere težave v robnih položajih.

Vmesnik za zajem podatkov in algoritem za izračun rotacij bi lahko uporabili tudi za veliko drugih aplikacij, kot zgolj za upravljanje robotske roke. Lahko bi ga uporabili kot igralno konzolo za različne igre, predvsem tiste, ki zahtevajo natančno vodenje roke. Teoretično bi bil uporaben tudi pri analizi športnikov v različnih panogah. Kot sem že omenil, se podobni sistemi že uporabljajo na brezpilotnih letalih in samouravnoteževalnih robotih, v prihodnosti pa imam cilj razviti sistem dopolniti s sistemom GPS ter ga uporabiti za beleženje različnih parametrov (naklon, hitrost, pospešek, lokacija ipd.) ob vožnji z motornim kolesom.

Dodatek A

Tabele karakteristik in nastavitev vmesnika za zajem podatkov

Mikrokrmilnik	ATmega328P
Napetost	3.3 V ali 5 V (možnost nastavitve)
Frekvenca procesorja	8 MHz (pri 3.3 V) ali 16 MHz (pri 5 V)
Digitalni vhodno/izhodni pini	14
Analogni vhodni pini	8
Spomin	Flash (16 kB), SRAM (1 kB) in EEPROM (1 kB)
Komunikacije	I^2C, SPI, UART TTL
Dimenzije	43 mm x 18 mm

Tabela A.1: Arduino Pro Mini - karakteristike.

Izhod	16 bitov za vsako os
Komunikacije	I^2C in SPI
Filtri	Vgrajena visokoprepustni in nizkoprepustni filter
Resolucije	±250, ±500 in ±2000 °/s
Hitrost podatkov	100, 200, 400 in 800 Hz
Druge lastnosti	Ultra stabilen glede na temperaturo in čas
	Omogoča prekinitve

Tabela A.2: Žiroskop - karakteristike.

Resolucija	±250 °/s	±500 °/s	±2000 °/s
Občutljivost	8,75 m°/s	17,50 m°/s	70 m°/s

Tabela A.3: Žiroskop - občutljivost pri določenih resolucijah.

Hitrost podatkov	100 Hz
Resolucija	±250 °/s

Tabela A.4: Žiroskop - nastavitve.

Izhod	13 bitov za vsako os
Komunikacije	I^2C in SPI
Resolucija	±16g
Občutljivost	4mg/LSB
Hitrost podatkov	od 0.1 do 3200 Hz
Druge lastnosti	Odporen na udarce do 10.000g
	Omogoča prekinitve
	Možnost detekcije prostega pada

Tabela A.5: Pospeškometer - karakteristike.

Hitrost podatkov	100 Hz
Resolucija	±16g

Tabela A.6: Pospeškometer - nastavitve.

Izhod	12 bitov za vsako os
Komunikacije	I^2C
Resolucija	od ± 0.88 Ga do ±8.1 Ga
Občutljivost	od 0.73 mGa/LSB (pri resoluciji ±0.88 Ga)
	do 4.35 mGa/LSB (pri resoluciji ±8.1 Ga)
Hitrost podatkov	od 0.75 do 75 Hz (izjemoma do 160 Hz)
Druge lastnosti	Omogoča natančnost od 1 do 2 stopinji

Tabela A.7: Magnetometer - karakteristike.

Hitrost podatkov	15 Hz
Resolucija	±1.3 Ga
Občutljivost	0.92 mGa/LSB

Tabela A.8: Magnetometer - nastavitve.

Dodatek B

Osnove rotacijskih matrik

Rotacijska matrika je matrika, ki opisuje rotacijo v Evklidskem prostoru. Vse uporabljene operacije se izvajajo v prostoru $\mathbb{R}^3$, zato so tudi uporabljene matrike velikosti 3×3. Za rotacijske matrike velja, da so ortogonalne (torej velja $A^{-1} = A^T$), njihova determinanta pa je enaka 1. Uporabljamo desnosučni koordinatni sistem - to je pomembno zaradi smeri rotacije. Če gledamo v smeri osi, okrog katere rotiramo, so v desnosučnem koordinatnem sistemu pozitivne rotacije v obratni smeri urinega kazalca.

Naslednje rotacijske matrike rotirajo okrog x, y in z osi v tri dimenzionalnem prostoru:

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

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

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

Vsaka izmed rotacijskih matrik rotira okrog ene koordinatne osi. Če želimo dobiti rotacijsko matriko, ki rotira okrog vseh treh osi, moramo med

sabo pomnožiti posamezne rotacijske matrike, pri tem pa je zelo pomemben vrstni red.

Naj bo: $fromEuler("XYZ",\ \alpha,\ \beta,\ \gamma) = R_x(\alpha) \times R_y(\beta) \times R_z(\gamma)$ (prvi parameter naj določa vrstni red rotacijskih matrik). Pri tem je potrebno biti pozoren, saj se pri rotiranju objekta z zgornjo matriko najprej izvede rotacija okrog z-osi, nato okrog y-osi ter na koncu okrog x-osi (kadar gledamo globalni (fiksni) koordinatni sistem objekta, ki se med rotacijami ne rotira poleg objekta). Lahko pa si predstavljamo rotacije okrog lokalnih koordinatnih osi, ki se med rotiranjem rotirajo zraven objekta - so fiksne glede na objekt. V tem primeru pa se bo najprej izvedla rotacija okrog x-osi ter na koncu okrog z-osi.

Naj bo: $fromEuler("XYZ",\ M)$ funkcija, ki vrne trojico kotov $(\alpha,\ \beta,\ \gamma)$, da velja $R_x(\alpha) \times R_y(\beta) \times R_z(\gamma) = M$. Tedaj velja:
$toEuler("XYZ",\ fromEuler("XYZ",\ \alpha,\ \beta,\ \gamma)) = (\alpha,\ \beta,\ \gamma)$.

Množenje matrik ni komutativna operacija, torej $A \times B \neq B \times A$. Naj bosta A in B rotacijski matriki in naj nam A določa trenutno rotacijo nekega objekta. Ta objekt lahko zarotiramo z rotacijsko matriko B (naj bo $B = R_x(\alpha) \times R_y(\beta) \times R_z(\gamma)$). Z množenjem $A \times B$ rotiramo objekt okrog njegovega lokalnega koordinatnega sistema, najprej okrog osi x ter na koncu okrog osi z (glede na njegov lokalni koordinatni sistem, ki se bo med rotiranjem tudi rotiral). Z množenjem $B \times A$ pa ta objekt rotiramo okrog globalnega (fiksnega) koordinatnega sistema, najprej okrog osi z ter na koncu okrog osi x (glede na globalni koordinatni sistem).

Dodatek C

Navodila za namestitev

Na priloženem elektronskem mediju lahko najdete datoteke, potrebne za namestitev. Namestitev lahko poenostavite z uporabo priloženih datotek za virtualizacijo operacijskega sistema Windows XP s programom VMware, kjer so že naloženi potrebni gonilniki za strojno opremo, programski paket za upravljanje robota Nao in ostale datoteke, potrebne za delovanje. Datoteke za virtualni operacijski sistem se nahajajo na elektronskem mediju v mapi *WindowsXP*, potrebne datoteke za uporabo produkta diplomske naloge pa se v virtualnem operacijskem sistemu nahajajo na namizju v mapi *Diploma*. V primeru uporabe virtualnega operacijskega sistema nadaljujte pri dodatku D, kjer so navodila za uporabo.

Vmesnik za zajem podatkov ima možnost komunikacije z računalnikom z brezžično povezavo bluetooth in žično povezavo preko povezave USB. Za uporabo povezave bluetooth potrebuje vaš računalnik podporo bluetooth s primernimi gonilniki. Lahko uporabite tudi povezavo USB (v tem primeru potrebujete gonilnike FTDI za virtualizacijo naprav USB kot standardna serijska vrata COM, dostopne na *http://www.ftdichip.com/Drivers/VCP.htm*).

Za povezavo na robota Nao je potreben modul *pynaoqi* za programski jezik *Python*, dostopen na *http://www.aldebaran-robotics.com/documentation /dev/python/install_guide.html*.

Nato si lahko snamete mapo *Skripte* s priloženega elektronskega medija,

v kateri se nahajajo potrebni programi, ter sledite navodilom za uporabo v dodatku D.

Dodatek D

Navodila za uporabo

V primeru uporabe virtualnega sistema se vse potrebne datoteke nahajajo v mapi *Diploma* na namizju, sicer pa so datoteke na priloženem elektronskem mediju v mapi *Skripte*. Za uporabo morate pravilno nastaviti nastavitve v datoteki $Config.py$. Najprej morate nastaviti vrata serijske povezave. V primeru uporabe vmesnika bluetooth morate na vašem sistemu vzpostaviti povezavo z vmesnikom za zajem podatkov (naprava se imenuje $Sensors$, geslo za združitev naprave pa je 1234). S tem se vam na računalniku ustvari virtualna serijska povezava in ugotoviti morate njeno ime. Pri žični povezavi USB se prav tako ustvari virtualna serijska povezava (edina razlika v primerjavi s povezavo bluetooth je, da ni potrebno predhodno združevanje naprav). V primeru uporabe žične povezave USB morate z elektronskega vmesnika za zajem podatkov odstraniti modul za povezavo bluetooth, stikalo za baterijsko napajanje pa mora biti na Off.

Uporabniki operacijskega sistema Windows lahko najdete ime naprave s programom *Upravljatelj naprav* (angl. *Device Manager*). Vrata se nahajajo v vrstici *Vrata (COM & LPT)* (angl. *Ports (COM & LPT)*) in se imenujejo $COMx$ (kjer je x določeno zaporedno število). V operacijskem sistemu Linux se vrata imenujejo */dev/ttyUSBx*, kjer je x prav tako določeno zaporedno število. Ime virtualnih vrat spremenite v datoteki $Config.py$, kjer so označena s spremenljivko $serial\_port$.

Za upravljanje robota Nao (pravega robota ali pa robota v simulatorju), morate v datoteki *Config.py* nastaviti tudi spremenljivko *nao_ip_port* (nastavitev IP povezave in vrat za robota). Za upravljanje robota poženite *Nao.py* (z ukazom *python Nao.py*).

Samo za izpisovanje vrednosti s senzorjev lahko uporabite *PrintData.py*. S programom *PlotDataBoth.py* lahko tudi grafično prikazujete rotacijo obeh senzorjev - za to je potrebna knjižnica *pyprocessing*, ki je dostopna na *https://code.google.com/p/pyprocessing/*.

Navodila za kalibracijo senzorjev so v dodatku E.

Dodatek E

Navodila za kalibracijo

Podrobni postopki kalibracije senzorjev so opisani v poglavju 3.5. Za kalibracijo pospeškometra in magnetometra nisem naredil posebnih programov in je potrebno narediti kalibracijo ročno. Za pomoč naj služi program *PrintData.py*, ki izpisuje različne podatke s senzorjev. Spodaj sta opisana postopka kalibracije magnetometra ter postopek kalibracije začetnega položaja roke. Rezultate kalibracij je potrebno vnesti v $Config.py$.

E.1 Kalibracija magnetometra

Teoretične podrobnosti kalibracije magnetometra so opisana v poglavju 3.5.3. Kalibracijo magnetometra moramo narediti v dveh korakih in sicer prvo moramo zajeti podatke ter jih nato procesirati. Za zajem podatkov uporabite program *SaveMagnetometerData.py*. Najprej bo program zajel podatke z magnetometra za nadlaket in jih shranil v datoteko *upperarm.txt* ter nato podatke z magnetometra za podlaket in jih shranil v datoteko $forearm.txt$. Kalibracija bo uspešna, če boste med zajemanjem podatkov rotirali senzor v čim več različnih smeri - tako bo aproksimacija elipsoida na dobljene podatke v naslednjem koraku bolj natančna. Datoteki $forearm.txt$ in *upperarm.txt* se shranita v podmapo $Kalibracija$. V tej podmapi se nahaja tudi $Matlab$ program *magnetometer_calibration.m* za procesiranjem dobljenih podatkov.

Uporabite lahko tudi program *magneto12.exe* (dostopen na https://sites.google.com/site/sailboatinstruments1/) - oba programa dajeta podobne rezultate. Pri uporabi *magneto12.exe* je potrebno v okence *Norm of Magnetic or Gravitational field* vnesti vrednost 470.

Z obema programoma dobimo kalibracijsko matriko (v programu *magneto12.exe* je to zgornja matrika, v programu *magnetometer_calibration.m* pa spodnja matrika - *Ellpisoid comp evecs*) ter vrednosti za translacijo (v programu *magneto12.exe* so to vrednosti pri *Combine bias (b)*, v programu *magnetometer_calibration.m* pa vrednosti pri *Ellipsoid center*). Rezultate vnesemo v datoteko $Config.py$. Translacijske vrednosti vnesemo v spremenljivki *upperarm_calib* in $forearm\_calib$ (odvisno od tega, za kateri senzor smo izračunali podatke) in sicer v slovar z indeksi *'mX'*, *'mY'* in *'mZ'*. Vrednosti kalibracijske matrike pa vnesemo v spremenljivki *upperarm_calibM* in $forearm\_calibM$.

E.2 Kalibracija začetnega položaja

Kalibracija začetnega položaja je opisana v poglavju 4.2.4. Za ta namen sem naredil program $CalibratePositionForNao.py$. Program zabeleži dvesto meritev ter izpiše kalibracijske vrednosti, ki jih moramo vnesti v datoteko $Config.py$ in sicer v spremenljivke, ki so pod komentarjem *Kalibracija začetnega položaja*.

Literatura

[1] Aldebaran Robotics, "Overview — NAO Software 1.14.3 documentation," NAO dokumentacija, apr. 2013. Dostopno na: http://www.aldebaran-robotics.com/documentation/.

[2] Analog Devices, "Digital Accelerometer," ADXL345 dokumentacija, 2013.

[3] Best Microcontroller Projects, "An I2C Tutorial," *best-microcontroller-projects.com*. Dostopno na: http://www.best-microcontroller-projects.com/i2c-tutorial.html.

[4] Shane Colton, "The Balance Filter," *Chief Delphi white paper*, jun. 2007.

[5] E. Denti, R. Galatolo in F. Schettini, "An AHRS Based on a Kalman Filter for the Integration of Inertial, Magnetometric and GPS Data," v *Proceedings of the 27th Congress of the International Council of Aeronautical Sciences (ICAS)*, Nica, Francija, sep. 2010.

[6] Jay Esfandyari, Roberto De Nuccio in Gang Xu, "Introduction to MEMS gyroscopes," *electroiq.com*, nov. 2010. Dostopno na: http://www.electroiq.com/articles/stm/2010/11/introduction-to-mems-gyroscopes.html.

[7] Honeywell International, "3-Axis Digital Compass IC HMC5883L," HMC5883L dokumentacija, feb. 2013.

[8] Mark Looney, "A simple calibration for MEMS gyroscopes," *EDN Europe*, strani 28-31, julij 2010.

[9] Yury Petrov, *Elipsoid fit*, jul 2009. Dostopno na: http://www.mathworks.com/matlabcentral/fileexchange/24693-ellipsoid-fit/.

[10] Starlino, "A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications," *starlino.com*. Dostopno na: http://www.starlino.com/imu_guide.html.

[11] STMicroelectronics, "MEMS motion sensor: ultra-stable three-axis digital output gyroscope," L3G4200D dokumentacija, dec. 2010.

[12] Texas Instruments, "High speed CMOS logic analog multiplexers/demultiplexers," 74HC4051 dokumentacija, nov. 1997 [Revidirana feb. 2011].

[13] VectorNav Technologies, "Calibration," vectornav.com. Dostopno na: http://www.vectornav.com/support/library?id=86.

[14] T.S. Yoo, S.K. Hong, H.M. Yoon in S. Park, "Gain-Scheduled Complementary Filter Design for a MEMS Based Attitude and Heading Reference System," *Sensors*, 2011, 11(4):3816-3830.