Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Karla Sever

Implementacija komplementarnog filtra za procjenu orijentacije modela ADCS sustava temeljena na gradijentnom spustu

Zagreb, 2023.

Ovaj rad izrađen je na Zavodu za komunikacijske i svemirske tehnologije Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu pod vodstvom doc. dr. sc. Josipa Lončara i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2022./2023.

Sadržaj

Poglavlje 1

Uvod

Precizno i pouzdano određivanje lokacije i orijentacije umjetnih satelita složen je postupak od velike važnosti za moderne svemirske misije. Određivanje trenutne orijentacije je temelj za postupke u navigiranju orbitirajućeg satelita kroz svemir poput korekcije trenutne orijentacije i održavanja ispravne orijentacije satelita. Zbog kritične važnosti navedenih zadataka za ishode misije, sateliti unutar svoje strukture sadržavaju zaseban sustav čiji je zadatak određivanje trenutne orijentacije, njezina korekcija te održavanje. Naziv tog sustava je ADCS (*Attitude Determination and Control System*).

ADCS sustav sastoji se od hardvera i softvera koji su povezani i čiji je rad međusobno ovisan. Određivanje orijentacije softverski je ostvarena funkcionalnost koja se temelji na multi-senzorskim mjerenjima obrađenih nekom od tehnika senzorske fuzije (eng. *sensor fusion*). Spomenute tehnike kombiniraju podatke više zasebnih senzora u jedinstvenu informaciju veće točnosti. Jedna od najčešće korištenih tehnika senzorske fuzije je Kalmanov filtar koji se zbog svoje preciznosti i efikasnosti koristi u mnogim područjima primjene: od navigacijskih primjena u autonomnim vozilima ([1, 2, 3, 4, 5]), dronovima ([6, 7]) i robotici ([8, 9]) do sustava za obradu signala u radarima ([10, 11]) i sonarima ([12, 13]). Ipak, zbog svoje kompleksne implementacije i izvedbe koja predstavlja značajno opterećenje resursa računalnog sustava, upotreba Kalmanovog filtra nije uvijek najbolji izbor. U *low-power* primjenama iz tog se razloga često poseže za komplementarnim filtrom - implementacijski jednostavnijom tehnikom senzorske fuzije koja postavlja relativno male zahtjeve na količinu potrebne procesorske snage za izvođenje algoritma.

Ovaj pisani rad donosi detaljan prikaz programske implementacije komplementarnog filtra temeljenog na iterativnom optimizacijskom algoritmu gradijentnog spusta (eng. *gradient descent*) u svrhu određivanja orijentacije prototipa ADCS sustava, kao i demonstraciju kvalitete rada navedenog algoritma.

Radi boljeg razumijevanja važnosti ADCS sustava za satelitske misije, u drugom ćemo poglavlju rada opisati ulogu, smještaj i strukturu ADCS sustava. Prikazat ćemo konkretan prototip sustava koji smo izradili i na kojem se provodilo testiranje rada implementiranog algoritma.

Kako bismo razjasnili osnovna matematička načela implementiranog algoritma, potrebno je razumjeti na koje se načine orijentacija objekta uopće može matematički iskazati. Reprezentacija orijentacije Eulerovim kutovima i kvaternionima objašnjena je stoga u trećem poglavlju ovog rada.

Nakon toga, u četvrtom poglavlju donosimo matematičku pozadinu algoritma gradijentnog spusta, njegove karakteristike te važnost odabira njegovih parametara. U ovom ćemo poglavlju dati i ulomke implementiranog koda.

Detaljno objašnjenje komplementarnog filtera i njegove programske implementacije bit će opisano u petom poglavlju.

Implementirani algoritam verificirali smo simulacijski pomoću softverski generiranih mjerenja senzora i eksperimentalno na vlastitom prototipu ADCS sustava kojeg razvijamo na Zavodu za komunikacijske i svemirske tehnologije Fakulteta elektrotehnike i računarstva u Zagrebu. Cjelokupan postupak verifikacije algoritma s pripadajućim objašnjenjima rezultata opisan je u šestom poglavlju.

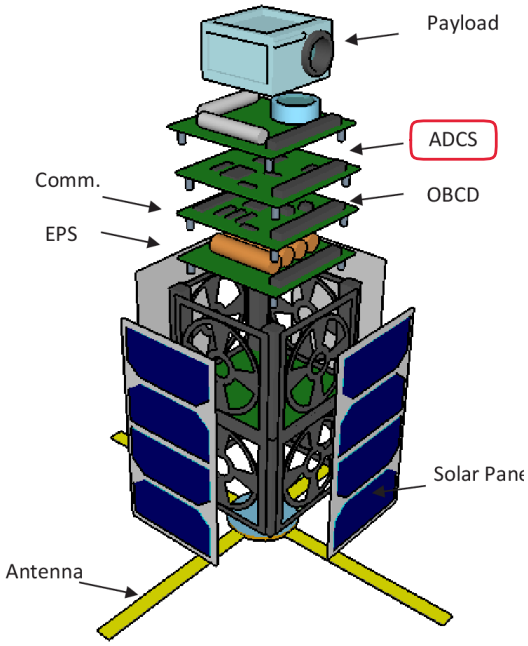
U sedmom poglavlju ovog rada osvrnut ćemo se na postignute rezultate te izložiti konačne zaključke.

Poglavlje 2

ADCS

2.1 Osnovne informacije o ADCS sustavu

Sustav za određivanje i kontrolu pozicije i orijentacije (eng. *Attitude Determination and Control System*; ADCS) satelitski je podsustav koji ima ključnu važnost za veliki broj suvremenih svemirskih misija. Njegova je uloga određivanje trenutne orijentacije satelitske platforme i njezinih instrumenata, te ukoliko nije ispravno orijentirana, njezinu korekciju. Prikaz smještaja ADCS sustava na raščlanjenoj strukturi minijaturnog satelita (tzv. *CubeSat*-a) prikazan je slikom 2.1.



Slika 2.1: Prikaz smještaja ADCS sustava u raščlanjenoj strukturi *CubeSat*-a, malog satelita ([14]).

Primjerice, ADCS sustavi satelita namijenjenih snimanju Zemljine površine najčešće moraju osigurati da je kamera usmjerena prema točki na Zemlji koja se nalazi direktno ispod satelita (tzv. *nadir-pointing*). Prikaz takvog usmjeravanja satelita ilustriran je slikom 2.2.

Slika 2.2: Sateliti koji koriste *nadir-pointing* usmjeravanje.

Osim održavanja ispravne orijentacije satelita tijekom misije, ADCS je zadužen i za *detumbling* – inicijalnu stabilizaciju satelita nakon njegovog otpuštanja u orbitu. *Detumbling* predstavlja kritičnu točku u misiji satelita - ukoliko se ne izvrši njegovo pravilno pozicioniranje u orbiti te orijentacija mjernih senzora i komunikacijskih antena, satelit ne može započeti s korisnim radom.

2.2 Struktura ADCS sustava

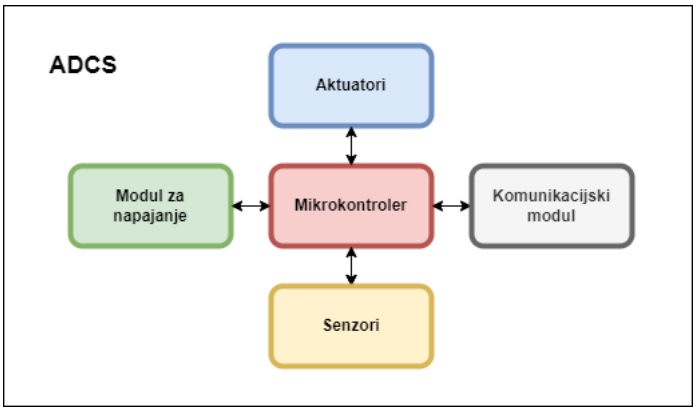
ADCS sustav se sastoji od senzora i aktuatora koji omogućavaju satelitu održavanje stabilne orijentacije u orbiti. Senzori ADCS-a prikupljaju podatke iz okoline satelita. Na temelju sakupljenih podataka, upravljački algoritmi ADCS-a estimiraju položaj satelita u orbiti i njegovu orijentaciju. Neke od često korištenih senzora koji se koriste kao dio ovog podsustava su: magnetometar, žiroskop, *star-tracker*, senzor horizonta i sl. S druge strane, aktuatori omogućuju ADCS-u da kontrolira položaj i orijentaciju satelita. Primjeri u praksi često upotrebljavanih aktuatora su zamašnjaci, magnetorkeri, potisnici, i sl. Konkretan odabir senzora i aktuatora korištenih u dizajnu ADCS podsustava ovisi o ciljevima i budžetu misije.

Prototip ADCS sustava, zajedno s pripadajućim testnim postavom, izradili smo samostalno na Zavodu za komunikacijske i svemirske tehnologije Sveučilišta u Zagrebu u edukativne svrhe. Cjelokupni se sustav sastoji od: tiskane pločice, postolja za elektroniku, kugle za testiranje i zračnog ležaja (slika 2.3). Na njemu je provedeno testiranje komplementarnog filtra čija je implementacija opisana u ovom radu.

Slika 2.3: Prototip ADCS sustava s pripadajućim testnim postavom.

2.2.1 ADCS PCB

Tiskana pločica (eng. *Printed Circuit Board*; PCB) sadržava upravljačku elektroniku ADCS-a. Pojednostavnjeni shematski prikaz njezine strukture prikazan je sljedećom blok-shemom (slika 2.4):

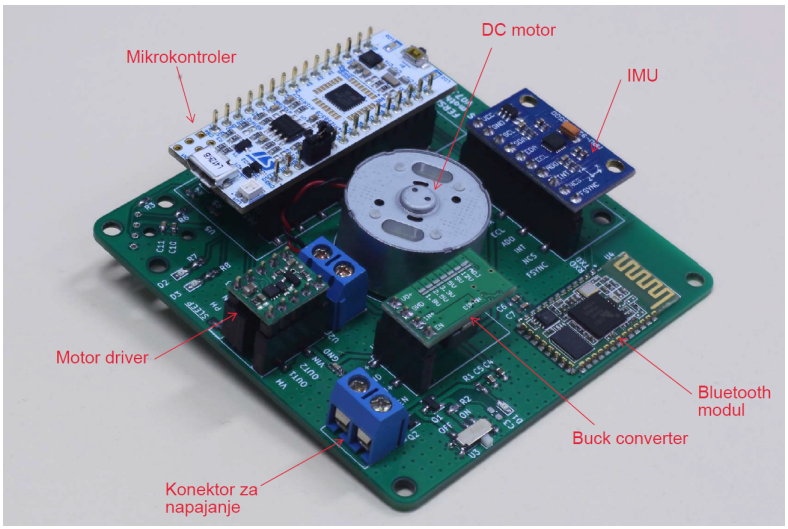


Slika 2.4: Blok-shema tiskane pločice ADCS-a.

Ključne komponente koje taj PCB sadržava su:

- senzori - inercijalna mjerna jedinica (IMU) i inkrementalni enkoder
- aktuator - zamašnjak pogonjen DC motorom
- bluetooth modul - korišten za komunikaciju s vanjskim računalom
- modul za napajanje
- mikrokontroler

Prikaz tiskane pločice ADCS-a s naznačenim ključnim komponentama dan je slikom 2.5.



Slika 2.5: Tiskana pločice ADCS-a s označenim komponentama.

Mikrokontroler ADCS-a središnja je komponenta ADCS-a. Na njemu se izvode algoritmi koji su zaduženi za prikupljanje informacija sa senzora, njihovu obradu, tumačenje te naposlijetku korištenje tih podataka u svrhu izračuna orijentacije satelita. Na temelju tih podataka, mikrokontroler upravlja aktuatorima satelita s ciljem korekcije orijentacije satelita. Pored toga, mikrokontroler je zadužen i za upravljanje komunikacijom – internom komunikacijom komponenata koje ga sačinjavaju te eksternom komunikacijom podsustava s centralnim računalom satelita.

2.2.2 Inercijalna mjerna jedinica

Za implementaciju koda razvijenog u sklopu ovog pisanog rada posebno nam je važan detaljniji osvrt na inercijalnu mjernu jedinicu (eng. *Inertial Measurement Unit*; IMU). IMU je lako dostupan, kompaktan, lagan i jeftin senzorski modul na koji se, u većoj ili manjoj mjeri, oslanja sustav za određivanje orijentacije velike većine satelita i svemirskih letjelica. Sastoji se od triju

troosnih senzora: akcelerometra, žiroskopa i magnetometra.

Akcelerometar je inercijski senzor kojim se mjeri translacijska akceleracija objekta. Njime je moguće postići dugoročnu pouzdanost u procjeni apsolutne orijentacije. Međutim, podatci akcelerometra sadržavaju visoku razinu šuma.

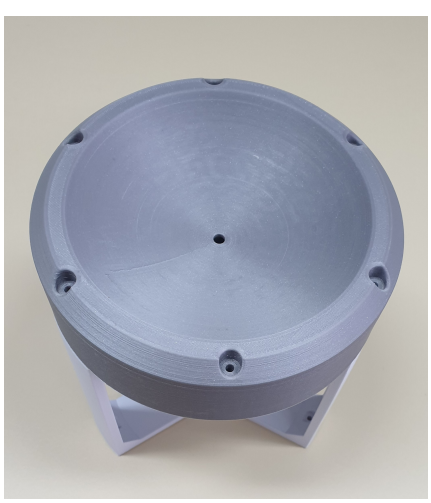
Žiroskop je, kao i akcelerometar, inercijski senzor. Koristi se za dobivanje informacije o kutnoj brzini objekta. Za razliku od akcelerometra, podatci dobiveni žiroskopom sadržavaju izuzetno nisku razinu šuma. No, njime je zbog akumulacije pogreške ipak moguće postići samo kratkoročnu pouzdanost procjene orijentacije relativne u odnosu na početno stanje.

Magnetometar, posljednji senzor IMU jedinice, mjeri jakost magnetskog polja Zemlje duž trio osi senzora izraženu u mikroteslama (μT). Zbog svoje jednostavnosti, neograničenog vidnog polja, niske cijene i potrošnje te funkcioniranja u sjeni Zemlje ovaj je senzor vrlo popularan za primjene na Zemljinoj površini te u niskoj Zemljinoj orbiti (eng. *Low-Earth Orbit*; LEO). Nedostatak korištenja magnetometra jest njegova nepreciznost te izražena osjetljivost na magnetske smetnje iz okoline senzora. Postoje dvije osnovne vrste distorzija koje utječu na ispravnost mjerenja magnetometra: distorzije uzrokovane djelovanjem permanentnog magnetskog polja magnetiziranog materijala čiji je položaj fiksiran u odnosu na magnetometar (tzv. *hard-iron distortion*) te distorzije uzrokovane prisustvom materijala koji stvaraju grešku u mjerenju magnetskog polja Zemlje, ali oni sami ne stvaraju magnetsko polje (tzv. *soft-iron distortion*). Utjecajem navedenih distorzija unosi se greška u mjerenje magnetskog polja, a njezino se uklanjanje vrši postupkom kalibracije.

Rad gore navedenih triju senzora IMU jedinice u navigacijskim se primjenama najčešće međusobno kombinira i nadopunjuje kroz algoritme senzorske fuzije, skupine algoritama u koju spada komplementarni filtar čiju će izvedba biti opisana u narednim poglavljima ovog rada.

2.3 Mehanički postav za testiranje ADCS sustava

Za ispitivanje ADCS sustava u laboratorijskim uvjetima potrebno je postići uvjete bliske onima u kojima bi satelit trebao u konačnici raditi, što nije jednostavan zadatak. U našem slučaju,



(a)

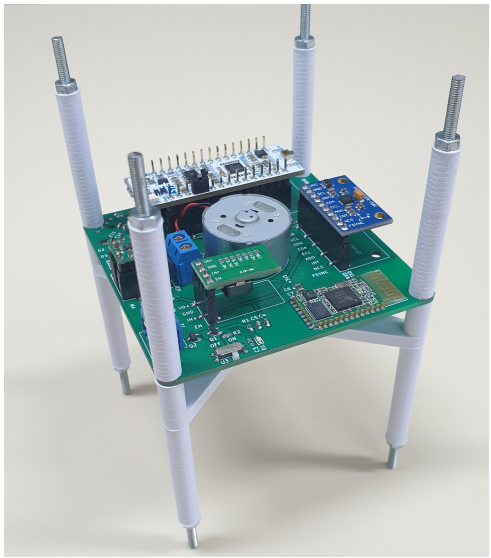
(b)

Slika 2.6: Zračni ležaj i kugla za testiranje.

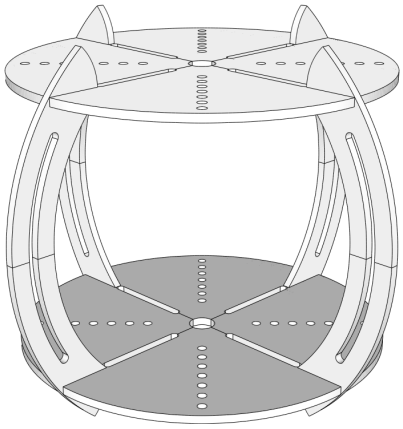
djelomično smo emulirali uvjete u LEO orbiti korištenjem testnog postava koji se sastoji od zračnog ležaja, postolja za elektroniku te kugle za testiranje.

Zračni ležaj centralni je element mehaničkog postava za ispitivanje ADCS sustava. To je posebna vrsta sferičnog kliznog ležaja vrlo malog trenja u kojem su površine kugle za testiranje i ležaja pri gibanju odvojene slojem stlačenog zraka. Time je omogućeno ispitivanje sustava za tri stupnja slobode gibanja satelita.

Upravljačka elektronika ADCS sustava fiksirana je unutar kugle za testiranje pomoću postolja i držača prikazanih na slikama ispod.



(a)



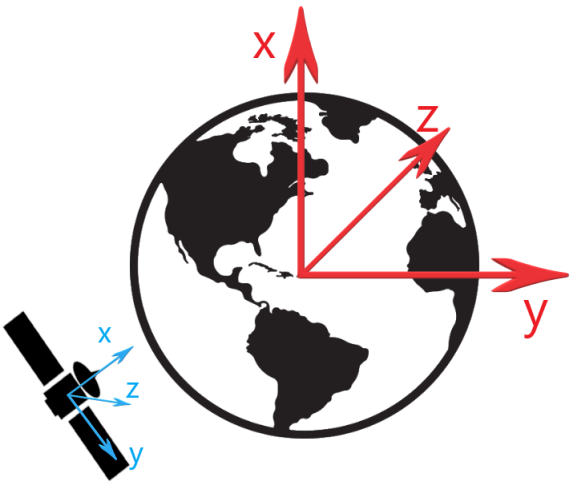
(b)

Slika 2.7: Postolje za elektroniku.

Poglavlje 3

Matematički opis orijentacije

Orijentaciju objekta u trodimenzionalnom prostoru iskazujemo kao rotaciju između lokalnog koordinatnog sustava objekta i referentnog inercijalnog koordinatnog sustava. Lokalni koordinatni sustav u našem je slučaju vezan uz model ADCS-a, dok je inercijalni koordinatni sustav vezan uz Zemlju te je definiran kao NED (eng. *North-East-Down*) koordinatni sustav - x, y i z-os definirane su u smjeru sjevera, istoka i središta Zemlje.

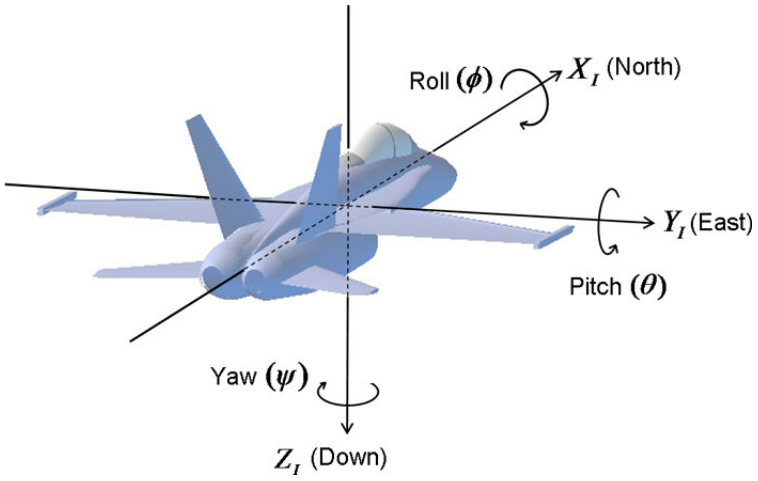


Slika 3.1: Inercijalni koordinatni sustav Zemlje i koordinatni sustav satelita.

Kako bismo mogli računalno odrediti orijentaciju objekta, potrebno je definirati način matematičkog zapisa rotacije između dvaju koordinatnih sustava. U praksi, dva najčešća pristupa uključuju upotrebu Eulerovih kutova ([15]) i kvaterniona ([16]). Ovo poglavlje objašnjava oba pristupa te detaljnije razmatra njihove prednosti i nedostatke.

3.1 Eulerovi kutovi

Orijentaciju objekta u prostoru moguće je opisati pomoću triju rotacijskih kutova koji predstavljaju zakret tog objekta oko triju međusobno okomitih osi koordinatnog sustava. Postoji više načina definiranja i primjene Eulerovih kutova, no najčešće se koriste tri rotacijska kuta označena s ϕ, θ i ψ koji redom definiraju zakret oko x-osi ili nagib (eng. *roll*), zakret oko y-osi ili poniranje (eng. *pitch*) i zakret oko z-osi, odnosno smjer (eng. *yaw*) objekta.



Slika 3.2: Eulerovi kutovi označeni na modelu aviona ([17]).

Svaka rotacija oko pojedine osi može se izraziti pomoću rotacijske matrice. Takve matrice čine posebnu vrstu kvadratnih matrica koje su ortogonalne, tj. čiji je inverz jednak transponiranoj rotacijskoj matrici.

$$R^{-1} = R^T \tag{3.1}$$

$$RR^T = R^T R = I, \tag{3.2}$$

a determinanta im je jednaka 1

$$det(R) = 1. \tag{3.3}$$

Eulerove kutove dakle možemo izraziti u obliku rotacijskih matrica na sljedeći način. Rotacijska matrica koja opisuje zakret oko x-osi koordinatnog sustava tijela za iznos ϕ prikazana je jednadžbom (3.4), rotacijska matrica koja opisuje zakret oko y-osi koordinatnog sustava tijela za iznos θ prikazana je jednadžbom (3.5), a rotacijska matrica koja opisuje zakret oko z-osi

koordinatnog sustava tijela za iznos ψ prikazana je jednadžbom (3.6).

$$R(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.4}$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.5}$$

$$R(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Budući da orijentaciju tijela želimo izraziti u trodimenzionalnom prostoru, rotacijske matrice (3.4) - (3.6) su dimenzija 3×3. Pritom je svaki stupac rotacijske matrice jedinični vektor koji opisuje rotaciju oko jedne osi koordinatnog sustava.

Rotacijske matrice primjenjuju se slijedno jedna za drugom postupkom množenja. Množenjem dviju ili više rotacijskih matrica dobiva se nova rotacijska matrica. Drugim riječima, rotaciju oko jedne osi možemo kombinirati s rotacijom oko druge osi te tako dobiti ukupnu rotaciju oko obje osi. Napomenimo kako je pritom redoslijed rotacija važan budući da množenje matrica nije komutativna operacija. Konvencionalni redoslijed rotacija u aeronautici glasi $z-y-x$: objekt se prvo rotira oko z-osi, potom oko y-osi i naposlijetku oko x-osi. Konačna matrica R_i^b transformira inercijalni koordinatni sustav u lokalni sustav tijela, prolazeći pritom kroz dva pomoćna koordinatna sustava v_1 i v_2.

$$R_i^b(\phi,\theta,\psi) = R_{v_2}^b(\phi) R_{v_1}^{v_2}(\theta) R_i^{v_1}(\psi) \tag{3.7}$$

Množenjem elemenata matrica dobivamo jedinstvenu matricu (3.8) čiji su elementi međusobno zavisni te su izraženi kao funkcije Eulerovih kutova.

$$R_i^b(\phi,\theta,\psi) = \begin{bmatrix} c(\psi)c(\theta) & c(\theta)s(\psi) & -s(\theta) \\ c(\psi)s(\phi)s(\theta) - c(\phi)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta) & c(\theta)s(\phi) \\ s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) & c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi) & c(\phi)c(\theta) \end{bmatrix} \tag{3.8}$$

Ovakvim redoslijedom množenja rotacijskih matrica jednoznačno smo odredili rotaciju objekta oko sve tri osi koordinatnog sustava.

Iako intuitivan i jednostavan, opis orijentacije Eulerovim kutovima ipak nije široko upotrebljavan u navigacijskim aplikacijama zbog nedostatka koji se očituje u gubitku stupnja slobode rotacije, tzv. *gimbal locku*. Ovaj fenomen nastaje poklapanjem dvaju ili više gimbal prstena u rotacijskom sustavu, što dovodi do stvaranja singulariteta u sustavu. Za ranije spomenutu $z-y-x$ rotacijsku sekvencu, gimbal lock će se javiti kada je $\theta = \pm 90^\circ$ te će rotacija objekta u tom slučaju biti omogućena samo oko z-osi inercijalnog koordinatnog sustava. Pojava *gimbal locka* moguća je i za druge redoslijede rotacija, no pod drugim uvjetima. Postoje različiti pristupi za izbjegavanje ovog problema, a najčešći je pristup u kojem se za opis orijentacije koriste kvaternioni.

3.2 Kvaternioni

Kvaternioni su matematički objekti koji se upotrebljavaju za opis rotacije u trodimenzionalnom prostoru ([18, 16]). Možemo ih opisati kao poopćene kompleksne brojeve koji se sastoje od četiri parametara: jedne realne jedinice, tj. skalara i tri međusobno ortogonalne imaginarne jedinice i, j i k

$$q = q_1 + q_2 i + q_3 j + q_4 k \tag{3.9}$$

Rotacijski kvaternion određen je kutom rotacije α te pripadajućim jediničnim vektorom $n = (n_x, n_y, n_z)$ koji predstavlja os rotacije. Primijetimo da se rotacijski kvaternion može zapisati kao četvero-elementni vektor.

$$q = \cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right) n = \begin{bmatrix} \cos(\alpha/2) \\ n_x \sin(\alpha/2) \\ n_y \sin(\alpha/2) \\ n_z \sin(\alpha/2) \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}. \tag{3.10}$$

Rotaciju u 3D prostoru možemo izraziti kao množenje rotacijskim kvaternionom, odnosno kao množenje rotacijskom matricom 3.11 koja ima strukturu sličnu matrici 3.8, ali njezini su elementi sada izraženi pomoću elemenata kvaterniona umjesto Eulerovih kutova.

$$R_i^b(q) = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2q_3 + q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 + q_1q_3) & 2(q_3q_4 - q_1q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix} \tag{3.11}$$

Osim ranije spomenute činjenice da upotrebom rotacijskih kvaterniona izbjegavamo mogućnost pojave *gimbal locka*, kvaternioni posjeduju još nekoliko prednosti u odnosu na Eulerove kutove. Primjerice, kvaternioni omogućuju jedinstveni prikaz orijentacije objekta i ne ovise o redoslijedu primijenjenih rotacija kao što je to slučaj kod Eulerovih kutova. Drugim riječima, korištenjem kvaterniona nećemo se susresti s problemom višestrukih mogućih rješenja koji se javljaju kod Eulerovih kutova koji nisu jedinstveni jer postoji više načina za izražavanje iste rotacije. Nadalje, kvaternioni zahtijevaju manje računalne memorije za pohranu. Osim toga, usporedbom jednadžbi (3.11) i (3.8) uočavamo kako kvaternioni ne zahtijevaju korištenje trigonometrijskih funkcija što može značajno smanjiti vrijeme potrebno za obradu podataka. To ih čini računalno učinkovitijom opcijom. Zbog svega navedenog jasno je da kvaternioni predstavljaju pouzdaniji i efikasniji način za računalno opisivanje rotacije objekta u trodimenzionalnom prostoru u odnosu na Eulerove kutove.

Poglavlje 4

Algoritam gradijentnog spusta

4.1 Temeljne karakteristike algoritma

Gradijentni spust je iterativni optimizacijski algoritam koji se koristi za pronalaženje lokalnog minimuma zadane funkcije cilja J (eng. *cost function*). Kako bismo osigurali pronalazak minimuma funkcije cilja, potrebno je da je ta funkcija konveksna. U suprotnom, algoritam gradijentnog spusta može zapeti u lokalnom minimumu ili u stacionarnoj točki kvazi-konveksne funkcije.

Algoritam točku minimuma pronalazi krećući od neke proizvoljno zadane radne točke p_0. Sljedeća radna točka pronalazi se pomoću gradijenta zadane funkcije u trenutnoj radnoj točki prema sljedećoj relaciji:

$$q_{n+1} = q_n - \mu \nabla J(q_n), \tag{4.1}$$

Ta je relacija temeljna jednadžba algoritma gradijentnog spusta. U njoj, $\nabla J(q_n)$ predstavlja gradijent funkcije cilja u promatranoj radnoj točki, dok μ predstavlja stopu učenja (eng. *learning rate*). Stopa učenja definirana je kao pozitivni skalar koji određuje veličinu koraka kojim se u gradijentnom spustu spuštamo prema minimumu funkcije cilja. Stopa učenja time direktno utječe na konvergenciju algoritma. Ukoliko odaberemo premalenu vrijednost stope učenja, algoritam će konvergirati prema minimumu funkcije cilja vrlo sporo. U suprotnom slučaju, odabirom prevelike vrijednosti stope učenja algoritam može početi divergirati. Vrijednosti ovog parametra iz tog se razloga odabire eksperimentalno s ciljem pronalaženja minimuma funkcije cilja u što manje koraka. Osim o stopi učenja, izvedba algoritma gradijentnog spusta ovisi i o

početnom kvaternionu q_0 te o maksimalnom dopuštenom broju iteracija N_{max}.

Početni kvaternion definiramo pretpostavljanjem njegove vrijednosti. S obzirom da nije moguće unaprijed poznavati orijentaciju objekta, bilo koja odabrana vrijednost rotacijskog kvaterniona predstavlja valjanu pretpostavku. U daljnjim iteracijama, kako se funkcija cilja približava svojem minimumu, vrijednost se inicijalnog kvaterniona ažurira te postaje sve bliža optimalnoj vrijednosti. Što je veća razlika vrijednost inicijalnog kvaterniona od vrijednosti lokalnog minimuma funkcije cilja, algoritmu će biti potrebno više vremena za konvergenciju. Odnosno, ukoliko vrijednost inicijalnog kvaterniona nije značajno različita u odnosu na vrijednost lokalnog minimuma funkcije cilja, algoritam će brže konvergirati do optimalne vrijednosti.

Definiranje maksimalnog dopuštenog broja iteracija N_{max} najjednostavniji je način terminiranja izvođenja algoritma gradijentnog spusta. Time ujedno sprječavamo i zapinjanje algoritma u beskonačnoj petlji u slučaju divergencije. Alternativno, može se pratiti promjena vrijednosti funkcije cilja u posljednje dvije iteracije, ∇J ([19]).

$$|\Delta J(q_n)| = |J(q_n) - J(q_{n-1})| \tag{4.2}$$

Padom vrijednosti ∇J ispod unaprijed postavljene vrijednosti ∇J_{max} zaustavlja se izvođenje algoritma. Terminaciju algoritma nije moguće provesti čekanjem da vrijednost funkcije cilja padne na nulu u točki minimuma jer se zbog inherentnih nesavršenosti senzora taj uvjet možda nikada neće ispuniti. Još jedan mogući kriterij terminacije temeljen je na parametru G_{max} ([20]), pri čemu je G skalar definiran kao

$$G = (\nabla J(q))^T \nabla J(q). \tag{4.3}$$

Iterativni se proces zaustavlja kada vrijednost G_n padne ispod unaprijed proizvoljno definirane vrijednosti G_{max}. Što je G_{max} manji, to je zahtjev na točnost estimiranog kvaterniona veći. Kako bi se postigao optimalan rad algoritma gradijentnog spusta i ostvarila što veća točnost procjene uz što je moguće manji broj iteracija, potrebno je odabrati optimalni skup gore spomenutih i opisanih parametara ([20]).

4.2 Implementacija gradijentnog spusta u svrhu određivanja orijentacije

Budući da gradijentni spust omogućuje iterativni pronalazak točke minimuma zadane funkcije cilja, u našoj implementaciji primijenili smo ga kako bismo pronašli optimalni rotacijski kvaternion koji opisuje orijentaciju ADCS sustava, odnosno satelita. Implementirani algoritam sastoji se od sljedećih koraka:

1. Definiranje funkcije J i njezinog gradijenta
2. Odabir vrijednosti početnog koraka p_0 i stope učenja μ
3. Izračun gradijenta funkcije u promatranoj točki $\nabla J(q_n)$
4. Izračun nove radne točke pomoću jednadžbe (4.1)
5. Ponavljanje koraka 3. i 4. dok se ne zadovolji kriterij terminacije algoritma.

U nastavku teksta detaljno ćemo objasniti svaki korak gore izloženog pseudokoda, te prikazati kako je pojedini korak realiziran u programskom kodu. Svi dijelovi programskog koda prikazani u ovom poglavlju implementirani su u Matlabu kao dio objektno orijentirane klase imena *GradientDescent*. Ova klasa nasljeđuje funkcionalnosti osnovne klase *handle*, čime joj je omogućeno korištenje određenih metoda za upravljanje objektima. Nakon definiranja klase slijedi definicija atributa ove klase, varijabli koje definiraju stanje objekta klase, te konstruktora. Konstruktor je posebna metoda u Matlabu koja ima isto ime kao i klasa koju definira (vidljivo na primjeru 4.1), i koristi se za stvaranje objekata te klase. Unutar konstruktora definiralni smo između ostaloga i korak učenja μ, maksimalni broj iteracija N_{max} te parametar G_{max}.

```
classdef GradientDescent < handle

    properties
        mu                      % Step size
        Nmax                    % Maximum number of iterations
        Gmax                    % Termination criterion
        a_ref                   % Referent gravitational acceleration in inertial frame
        m_ref                   % Referent magnetic field in inertial frame
        a_i                       % Normalized gravitational acceleration in inertial frame
        m_i                     % Normalized magnetic field in inertial frame
        q_initial               % Initial quaternion
        q_iterated              % Quaternion estimated through gradient descent iterative process
        NOI                       % Number of iterations
        A                       % Magnetometer calibration matrix
        b                       % Magnetometer calibration vector
        GD_history_flag         % History flag (store or not the value of cost function)
        J_history               % Value of cost function for each iteration
        view_angles             % Azimut and elevation for 3D plot
    end

    methods

    function obj = GradientDescent()
        obj.a_ref = [0, 0, 9.81]';                  % Reference vector of gravitational acceleration
        obj.m_ref = [22.2, 1.7, 42.7]';             % Reference vector of the magnetic field in Zagreb
```

```
            obj.a_i = obj.a_ref/norm(obj.a_ref);            % Normalizing the referent vector
            obj.m_i = obj.m_ref/norm(obj.m_ref);            % Normalizing the referent vector
            obj.mu = 0.07;                                  % Learning rate
            obj.Nmax = 20;                                  % Algorithm termination criterion
            obj.Gmax = 3e-2;                                % Termination criterion
            obj.q_initial = [1 0 0 0]';                     % Initial quaternion
            obj.NOI = 0;                                    % Number of iteratoins initialization
            obj.A = eye(3,3);               % Default state - magnetometer is not calibrated
            obj.b = zeros(1,3);             % Default state - magnetometer is not calibrated
            obj.GD_history_flag = false;    % History flag (store or not the value of cost function)
            obj.view_angles = [-37.5, 30];
        end

% ...
```

Primjer 4.1: Početni ulomak definicije klase gradijentnog spusta.

4.2.1 Definiranje funkcije cilja

Za pouzdano određivanje orijentacije potrebno je poznavati vektorske reprezentacije barem dviju fizikalnih veličina izraženih u koordinatnom sustavu tijela i inercijalnom koordinatnom sustavu. U implementiranom kodu koristimo fizikalne veličine Zemljina gravitacijskog ubrzanja i jakosti magnetskog polja. Podatke za ove veličine dobili smo putem senzora IMU jedinice, konkretno akcelerometra i magnetometra. Vrijednosti koje su izmjerene senzorima smještenim na ADCS sustavu izražene su u lokalnom koordinatnom sustavu. Te vrijednosti usporedili smo s referentnim vektorima Zemljina gravitacijskog ubrzanja (6.1) i jakosti magnetskog polja (6.2) čije su vrijednosti izražene u inercijalnom koordinatnom sustavu. Kako bismo tu usporedbu mogli izvršiti, transformirali smo zapis referentnih vektora u koordinatni sustav tijela pomoću rotacijske matrice iskazane relacijom (3.11). U idealnim uvjetima, oduzimanjem vektora izmjerenih vrijednosti od referentnih vektora dobili bismo nulu. Međutim, u stvarnosti rezultat oduzimanja neće biti nula zbog nesavršenosti senzora, već će biti jednak vektorima pogrešaka e_1 i e_2. Vektor e_1 predstavlja vektor pogreške mjerenja akceleracije

$$e_a(q) = R_i^b a^i - a^b, \tag{4.4}$$

dok vektor e_2 predstavlja vektor pogreške mjerenja magnetskog polja

$$e_m(q) = R_i^b m^i - m^b. \tag{4.5}$$

Funkciju cilja smo stoga definirali kao funkciju vektora pogrešaka na sljedeći način:

$$\begin{aligned} J(q) = J(e_1, e_2) &= {e_a}^T e_a + {e_m}^T e_m \\ &= e_{1x}^2 + e_{1y}^2 + e_{1z}^2 + e_{2x}^2 + e_{2y}^2 + e_{2z}^2. \end{aligned} \tag{4.6}$$

Uočimo kako je funkcija cilja zapravo definirana kao funkcija rotacijskog kvaterniona $J(q)$. Naš je cilj gradijentnim spustom pronaći kvaternion koji minimizira ovako definiranu funkciju cilja u smislu problema najmanjih kvadrata.

Programska implementacija izračuna funkcije cilja prikazana je primjerom 4.2. Funkcija *find_cost*() kao argumente prima vektore mjerenja akceleracije i jakosti magnetskog polja izražene u koordinatnom sustavu tijela te kao rezultat vraća vrijednost funkcije cilja.

```
function J = find_cost(obj, a_b, m_b, q)
        % Calculating the rotation matrix for the given quaternion
        R = obj.q2R(q);

        % Calculating the error vectors
        e_a = R*obj.a_i - a_b;
        e_m = R*obj.m_i - m_b;

        % Calculating the value of cost function
        J = e_a'*e_a + e_m'*e_m;
end
```

Primjer 4.2: Programska implementacija funkcije *find_cost*() koja izračunava funkciju cilja.

Ova funkcija koristi *q2R*() funkciju (prikazanu primjerom 4.3) koja iz kvateriona koji joj se zadaje kao argument računa rotacijsku matricu (3.11) potrebnu za izračunavanje vektora pogrešaka.

```
function R = q2R(~, q)
        % Calculating rotation matrix elements for the given quaternion
        r11 = q(1)^2+q(2)^2-q(3)^2-q(4)^2;
        r12 = 2*(q(2)*q(3)+q(1)*q(4));
        r13 = 2*(q(2)*q(4)-q(1)*q(3));

        r21 = 2*(q(2)*q(3)-q(1)*q(4));
        r22 = q(1)^2-q(2)^2+q(3)^2-q(4)^2;
        r23 = 2*(q(3)*q(4)+q(1)*q(2));

        r31 = 2*(q(2)*q(4)+q(1)*q(3));
        r32 = 2*(q(3)*q(4)-q(1)*q(2));
        r33 = q(1)^2-q(2)^2-q(3)^2+q(4)^2;

        % Assembling the rotation matrix
        R = [r11,r12,r13;
        r21,r22,r23;
        r31,r32,r33];
end
```

Primjer 4.3: Programska implementacija funkcije *q2R*() koja iz elemenata trenutno promatranog kvaterniona računa rotacijsku matricu *R*.

4.2.2 Izračun gradijenta funkcije cilja

Kako bismo mogli primijeniti temeljnu formulu gradijentnog spusta (4.1), potrebno je izračunati gradijent funkcije cilja. Time se nameće zahtjev diferencijabilnosti i konveksnosti na funkciju cilja, što za našu definiranu funkciju cilja vrijedi. Ona je definirana kao funkcija više varijabli te ju je zato potrebno parcijalno derivirati po svakoj varijabli (q_1, q_2, q_3, q_4). Prilikom izračunu parcijalne derivacije po određenoj nezavisnoj varijabli, ostale nezavisne varijable smatramo konstantama. U nastavku su prikazane izračunate parcijalne derivacije matrice R_i^b (3.11):

$$M_1(q_n) = \frac{\partial}{\partial q_1} R_i^b = 2 \begin{bmatrix} q_1 & q_4 & -q_3 \\ -q_4 & q_1 & q_2 \\ q_3 & -q_2 & q_1 \end{bmatrix} \tag{4.7a}$$

$$M_2(q_n) = \frac{\partial}{\partial q_2} R_i^b = 2 \begin{bmatrix} q_2 & q_3 & q_4 \\ q_3 & -q_2 & q_1 \\ q_4 & -q_1 & -q_2 \end{bmatrix} \tag{4.7b}$$

$$M_3(q_n) = \frac{\partial}{\partial q_3} R_i^b = 2 \begin{bmatrix} -q_3 & q_2 & -q_1 \\ q_2 & q_3 & q_4 \\ q_1 & q_4 & -q_3 \end{bmatrix} \tag{4.7c}$$

$$M_4(q_n) = \frac{\partial}{\partial q_4} R_i^b = 2 \begin{bmatrix} -q_4 & q_1 & q_2 \\ -q_1 & -q_4 & q_3 \\ q_2 & q_3 & q_4 \end{bmatrix} \tag{4.7d}$$

Gore zapisane matrice funkcija su trenutno promatranog kvaterniona. Konačan zapis gradijenta funkcije cilja za kvaternion q_n vrši se prema jednadžbi

$$\frac{\partial}{\partial q_i} J(q) = 2{e_a}^T \left(\frac{\partial}{\partial q_i} R_i^b \right) a^i + 2{e_m}^T \left(\frac{\partial}{\partial q_i} R_i^b \right) m^i \tag{4.8}$$

glasi

$$\nabla J(q_n) = 2 \begin{bmatrix} e_a{}^T M_1 a^i(q_n) + e_m{}^T M_1 m^i(q_n) \\ e_a{}^T M_2 a^i(q_n) + e_m{}^T M_2 m^i(q_n) \\ e_a{}^T M_3 a^i(q_n) + e_m{}^T M_3 m^i(q_n) \\ e_a{}^T M_4 a^i(q_n) + e_m{}^T M_4 m^i(q_n) \end{bmatrix} \tag{4.9}$$

Implementacija izračuna gradijenta funkcije cilja prikazana je primjerom (4.4) na kojem su vidljivi svi koraci opisani u prethodnom tekstu.

```
function grad_J = find_gradient(obj, a_b, m_b, q)
        % Initialization of the gradient vector
        grad_J = zeros(4,1);

        % Calculating the rotation matrix for the given quaternion
        R = obj.q2R(q);

        % Calculating the error vectors
        e_a = R*obj.a_i - a_b;
        e_m = R*obj.m_i - m_b;

        % Calculating and storing the value of cost function
        if obj.GD_history_flag == true
        J = e_a'*e_a + e_m'*e_m;
        obj.store_cost(J);
        end

        % Rotation matrix partial derivative with respect to q(1)
        M1 = 2*[ q(1)  q(4) -q(3);
        -q(4)  q(1)  q(2);
        q(3) -q(2)  q(1)];

        % Rotation matrix partial derivative with respect to q(2)
        M2 = 2*[ q(2)  q(3)  q(4);
        q(3) -q(2)  q(1);
        q(4) -q(1) -q(2)];

        % Rotation matrix partial derivative with respect to q(3)
        M3 = 2*[-q(3)  q(2) -q(1);
        q(2)  q(3)  q(4);
        q(1)  q(4) -q(3)];

        % Rotation matrix partial derivative with respect to q(4)
        M4 = 2*[-q(4)  q(1)  q(2);
        -q(1) -q(4)  q(3);
        q(2)  q(3)  q(4)];

        % Calculating the elements of the cost function gradient
        grad_J(1) = 2*(e_a'*M1*obj.a_i  +  e_m'*M1*obj.m_i);
        grad_J(2) = 2*(e_a'*M2*obj.a_i  +  e_m'*M2*obj.m_i);
        grad_J(3) = 2*(e_a'*M3*obj.a_i  +  e_m'*M3*obj.m_i);
        grad_J(4) = 2*(e_a'*M4*obj.a_i  +  e_m'*M4*obj.m_i);
end
```

Primjer 4.4: Programska implementacija funkcije *find_gradient*() koja pomoću mjerenja akcelerometra i magnetometra računa gradijent funkcije cilja J.

4.2.3 Primjena algoritma gradijentnog spusta

U dosadašnjem opisu implementacije koda opisali smo kako smo zadali početne parametre algoritma te kako smo implementirali izračun funkcije cilja i njezinog gradijenta. U koraku koji slijedi, potrebno je povezati pomoćne funkcije u jednu glavnu funkciju koja će primjenjivati algoritam gradijentnog spusta na podatke akcelerometra i magentometra. Prikaz implementacije te funkcije dan je primjerom (4.5).

```
function q_estimated = apply_gradient_descent(obj, a_b, m_b)
        % Setting up the variables
        obj.NOI = 0;                    % Number of iterations
        obj.q_iterated = [];
        obj.J_history = [];
        q_estimated = obj.q_initial;

        % Calibrate magnetometer
        m_b = obj.calibrate(m_b);

        % Normalizing the measured body-frame vectors
        a_b = a_b/norm(a_b);
        m_b = m_b/norm(m_b);

        for i = 1 : obj.Nmax
                % Calculating gradient for the i-th iteration
                grad_J = obj.find_gradient(a_b, m_b, q_estimated);

                % Calculating the scalar representation of the gradient
                G = grad_J'*grad_J;

                % Checking if the iterative process should be terminated
                if  G < obj.Gmax
                        break;
                else
                        % Applying the gradient decent algorithm
                        q_estimated = q_estimated - obj.mu*grad_J;

                        % Normalizing the estimated quaternion
                        q_estimated = q_estimated/norm(q_estimated);

                        % Setting the number of iterations
                        obj.NOI = i;

                        if i == obj.Nmax && obj.GD_history_flag == true
                                % Calculating final cost function value
                                J = obj.find_cost(a_b, m_b, q_estimated);
                                obj.store_cost(J);
                        end
                end
                obj.q_iterated = q_estimated;
        end
end
```

Primjer 4.5: Programska implementacija funkcije *apply_gradient_descent*().

U funkciji *apply_gradient_descent*() prvo se provodi inicijalizacija varijabli potrebnih za njezin rad, a nakon toga se provodi kalibracija podataka magnetometra. Glavni cilj kalibracijskog postupka je pronalazak matrice A i vektora b koji će transformirati elipsoid koji opisuje mjerenja magnetometra u centraliziranu sferu prema formuli:

$$\mathbf{m_{calibrated}} = (\mathbf{m_{raw}} - \mathbf{b})\mathbf{A} \tag{4.10}$$

gdje je $\mathbf{m_{calibrated}}$ vektor kalibriranih mjerenja magnetometra, $\mathbf{A}$ transformacijska matrica kojom modeliramo utjecaj *soft-iron* distorzija, $\mathbf{m_{raw}}$ vektor nekalibriranih mjerenih podataka magnetometra, $\mathbf{b}$ skupni *bias* vektor kojim modeliramo *hard-iron* distorziju magnetskog polja. Programska implementacija funkcije koja se koristi za kalibraciju magnetometra prikazana je primjerom 4.6.

```
function m_calibrated = calibrate(obj, m_raw)
        m_raw = m_raw';                          % Convert to raw vector
        m_calibrated = (m_raw - obj.b)*obj.A;    % Perform calibration
        m_calibrated = m_calibrated';            % Convert back to column vector
end
```

Primjer 4.6: Programska implementacija funkcije *calibrate*() kojom se eliminiraju distorzije u izmjerenim podatcima magnetometra.

U nastavku funkcije *apply_gradient_descent*() provodi se normalizacija mjerenih vektora akceleracije i magnetskog polja s ciljem osiguranja jednake duljine svih vektora. Potom slijedi iterativni postupak koji procjenjuje optimalnu vrijednost rotacijskog kvaterniona, a završava kada se postigne uvjet $G_n < G_{max}$ ili kada se dosegne maksimalni broja iteracija N_{max}. U svakoj iteraciji provodi se normalizacija estimiranog kvaterniona zbog toga što se korištenjem njegove jedinične duljine određivanje orijentacije iz vektorskih promatranja može formulirati kao strogo konveksni problem ([21]).

Poglavlje 5

Komplementarni filter

Kao što je ranije spomenuto, komplementarni filtar je jednostavna metoda senzorske fuzije koja je prikladna za implementaciju na uređajima niske potrošnje. Njegova je implementacija ostvarena kao kombinacija niskopropusnog i visokopropusnog filtra koji imaju istu graničnu frekvenciju (eng. *cut-off frequency*). Niskofrekvencijsko filtriranje primjenjuje se na rotacijski kvaternion estimiran iz podataka akcelerometra i magnetometra algoritmom gradijentnog spusta, kao što je opisano u prethodnom poglavlju. Time smanjujemo visokofrekvencijske komponente šuma koje su posljedica korištenja akcelerometra i magnetometra. Ovaj kvaternion označit ćemo sa q_{gd}. Na sličan način, visokofrekvencijsko se filtriranje primjenjuje na rotacijski kvaternion estimiran vremenskom integracijom mjerenja kutne brzine. Time se eliminira vremenski akumulirana pogreška žiroskopa. Ovaj ćemo kvaternion označiti sa q_w. Kombinaciju niskopropusno i visokopropusno filtriranih kvaterniona možemo zapisati kao otežanu sumu koja predstavlja komplementarni filtar.

$$q = Kq_{\omega} + (1-K)q_{gd}. \tag{5.1}$$

U gornjoj jednadžbi, faktor K nazivamo faktorom skaliranja ili pojačanjem komplementarnog filtra. On može poprimiti vrijednost iz intervala $[0,1]$, što znači da će uvijek vrijediti sljedeća relacija:

$$K + (1-K) = 1 \tag{5.2}$$

Definiranjem vrijednosti koeficijenta K određujemo kojoj metodi procjene kvaterniona više vjerujemo. Tu odluku donosimo prethodnim poznavanjem prednosti i ograničenja pojedine

metode. Budući da smo u prethodnom tekstu detaljno opisali implementaciju, prednosti i mane algoritma gradijentnog spusta, preostaje nam detaljnije objasniti određivanje kvaterniona metodom integracije kutne brzine.

5.1 Određivanje rotacijskog kvaterniona integracijom kutne brzine

Ova je metoda računanja orijentacije u navigacijskim primjenama još poznata pod nazivom *dead reckoning*. Koristi se za određivanje kuta zakreta objekta temeljem vremenske integracije kutne brzine zabilježene žiroskopom. Ovom metodom nije moguće odrediti apsolutnu orijentaciju objekta, već samo njegov zakret u odnosu na neku početnu poznatu orijentaciju. Vremensku promjenu rotacijskog kvaterniona q_i^b, kojeg ćemo u daljnjem tekstu radi jednostavnosti označavati kao q, u diskretnom je vremenu moguće odrediti na sljedeći način:

$$\dot{q_n} = \frac{d}{dt} q_n = \frac{1}{2} S(\omega_n^b) q_{n-1}. \tag{5.3}$$

U gornjem izrazu, $S(\omega^b)$ je matrica dimenzija 4×4 na čijoj se dijagonali nalaze nule, a ostali elementi matrice izraženi su kao elementi vektora mjerenja kutne brzine:

$$S(\omega^b) = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}. \tag{5.4}$$

Pored toga, uočavamo kako je za izračun $\dot{q_n}$ potrebno poznavati vrijednost rotacijskog kvaterniona iz prethodnog koraka q_{n-1}. Izračunom vremenske promjene rotacijskog kvaterniona imamo sve potrebne podatke za određivanje vrijednosti estimiranog kvaterniona u trenutnom koraku:

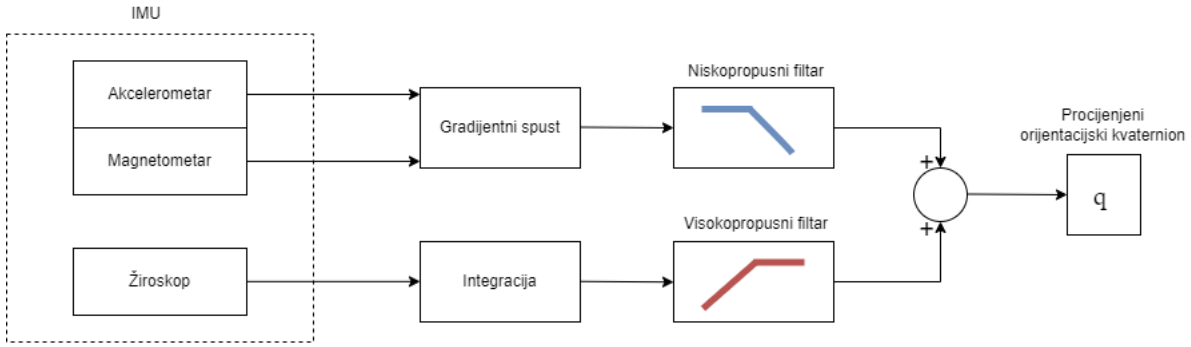
$$q_n = q_{n-1} + \Delta T \dot{q_n} = q_{n-1} + \frac{\Delta T}{2} S(w_n^b) q_{n-1}. \tag{5.5}$$

Gornjom jednadžbom definirali smo numeričko integriranje promjene kvaterniona po vremenu. U jednadžbi za izračun pretpostavlja se konstantan iznos parametra ΔT koji predstavlja vremenski interval između dva mjerenja kutne brzine. Njegova je vrijednost jednaka recipročnoj

vrijednosti frekvencije uzorkovanja korištenog senzora.

5.2 Implementacija klase komplementarnog filtra

Blok dijagramom prikazanim na slici 5.1 prikazana je pojednostavljena struktura komplementarnog filtra temeljenog na algoritmu gradijentnog spusta.



Slika 5.1: Blokovska shema koja ilustrira rad implementiranog komplementarnog filtra.

U prethodnom smo poglavlju opisali implementaciju klase *GradientDescent* koja estimira rotacijski kvaternion pomoću podataka dobivenih iz akcelerometra i magnetometra, te nam sada preostaje opisati klasu *ComplementaryFilter*. Ta je klasa definirana kao podklasa klase *GradientDescent*, kao što je prikazano primjerom 5.1. Klasa *ComplementaryFilter* ovakvom definicijom može, uz svoja definirana svojstva i metode, koristiti i svojstva i metode svoje nadklase *GradientDescent*.

```
classdef ComplementaryFilter < GradientDescent

    properties
        K;                          % Complementary filter gain
        dT;                         % Time between samples
        q_previous;
        q_estimated;
        propagation_type;           % Predictive
        first_sample_flag;
        CF_history_flag;
        q_history;
        NOI_history;
    end

    methods

    function obj = ComplementaryFilter()
        obj@GradientDescent();
        obj.K = 0.5;            % Scaling coefficient
        obj.dT = 0.1;           % The time between two measurement samples
        obj.q_previous = [1 0 0 0]';
        obj.q_estimated = [];
        obj.propagation_type = 'predictive';
        obj.first_sample_flag = true;
        obj.CF_history_flag = false;
```

```
            obj.q_history = [];
            obj.NOI_history = [];
        end

    %...
```

Primjer 5.1: Početni ulomak definicije klase komplementarnog filtra koja nasljeđuje od klase gradijentnog spusta.

Nakon definiranja klase slijedi definicija atributa ove klase, varijabli koje definiraju stanje objekta klase, te konstruktora. Unutar konstruktora inicijalizirane su vrijednosti svojstava objekta koji se stvara njegovim pozivom. Od važnijih inicijaliziranih svojstava izdvajamo pojačanje komplementarnog filtra K čija je vrijednost postavljena na 0.5, što znači da jednako vjerujemo objema procjenama. Osim K, postavljena je i vrijednost dT koja definira vremenski interval između dva mjerenja kutne brzine u trajanju od 100 milisekundi. Još jedno važno svojstvo definirano u konstruktoru klase *ComplementaryFilter* je propagacijska metoda inicijalnog kvaterniona gradijentnog spusta (!) koja je definirana kao prediktivna. To znači da se procjena orijentacije i informacija o kutnoj brzini trenutnog vremenskog koraka koriste kako bi se predvidjela vrijednost rotacijskog kvaterniona u sljedećem koraku. Ta se predikcija potom koristi kao početni kvaternion u sljedećem iterativnom procesu.

Na primjeru 5.2 prikazana je implementacija funkcije $apply\_complementary\_filter()$", koja predstavlja najvažniju funkciju u cjelokupnom programskom kodu. Njena važnost proizlazi iz toga što se njome izračunava optimalna vrijednost rotacijskog kvaterniona, što je ključan element koji nas zanima. Po ulasku u funkciju, prvo se provjerava je li argument funkcije prvi uzorak mjerenja senzora. Ukoliko jest, kvaternion se procjenjuje pozivom funkcije $apply\_gradient\_descent()$ koja je implementirana u klasi *GradientDescent*. Inicijalni kvaternion za gradijentni spust u tom je slučaju postavljen na $q\_initial = [1000]'$, sukladno definiciji koja je prisutna u konstruktoru „GradientDescent“. U slučaju da argument funkcije nije prvi uzorak mjerenja senzora, inicijalni se kvaternion za gradijentni spust računa funkcijom $q\_gyro()$ (prikazanom primjerom 5.3), koja integrira podatke o kutnoj brzini. Nakon toga, poziva se funkcija $apply\_gradient\_descent()$ kako bi se izračunao optimalni rotacijski kvaternion. Konačna procjena rotacijskog kvaterniona vrši se primjenom osnovne jednadžbe komplementarnog filtra (4.1). U oba slučaja, prije nego što se vrijednost estimiranog kvaterniona vrati kao rezultat, vrši se normiranje kvaterniona kako bi se očuvala konveksnost problema.

```
function estimation = apply_complementary_filter(obj, a_b, m_b, w)
        if obj.first_sample_flag == true
                obj.first_sample_flag = false;
                estimation = obj.apply_gradient_descent(a_b, m_b);
        else
                % Propagation_type: predictive
                q_gyro = obj.q_from_gyro(w);
                obj.q_initial = q_gyro;
                q_gd = obj.apply_gradient_descent(a_b, m_b);
                estimation = obj.K*q_gyro + (1 - obj.K)*q_gd;
        end

        estimation = estimation/norm(estimation);
        obj.q_estimated = estimation;
        obj.q_previous = estimation;

        if obj.CF_history_flag == true
                obj.q_history = [obj.q_history, estimation];
                obj.NOI_history = [obj.NOI_history, obj.NOI];
        end
end
```

Primjer 5.2: Programska implementacija funkcije *apply_complementary_filter*() kojom se izračunava traženi optimalni orijentacijski kvaternion kombiniranjem procjena dobivenih gradijentnim spustom i integriranjem mjerenja kutne brzine.

```
function q_gyro = q_from_gyro(obj, w)
        % Matrix used for calculation of the quaternion time change
        Omega = [   0, -w(1), -w(2), -w(3);
        w(1),     0,  w(3), -w(2);
        w(2), -w(3),     0,  w(1);
        w(3),  w(2),  -w(1),  0 ];

        % Calculating the time change (rate of change) of the
        % quaternion from the angular velocity
        q_dot = 0.5*Omega*obj.q_previous;

        % Intergating (accumulating) the quaternion change and adding
        % it to the previous quaternion.
        q_gyro = obj.q_previous + obj.dT*q_dot;

        % Normalizng the calculated quaternion
        q_gyro = q_gyro/norm(q_gyro);
end
```

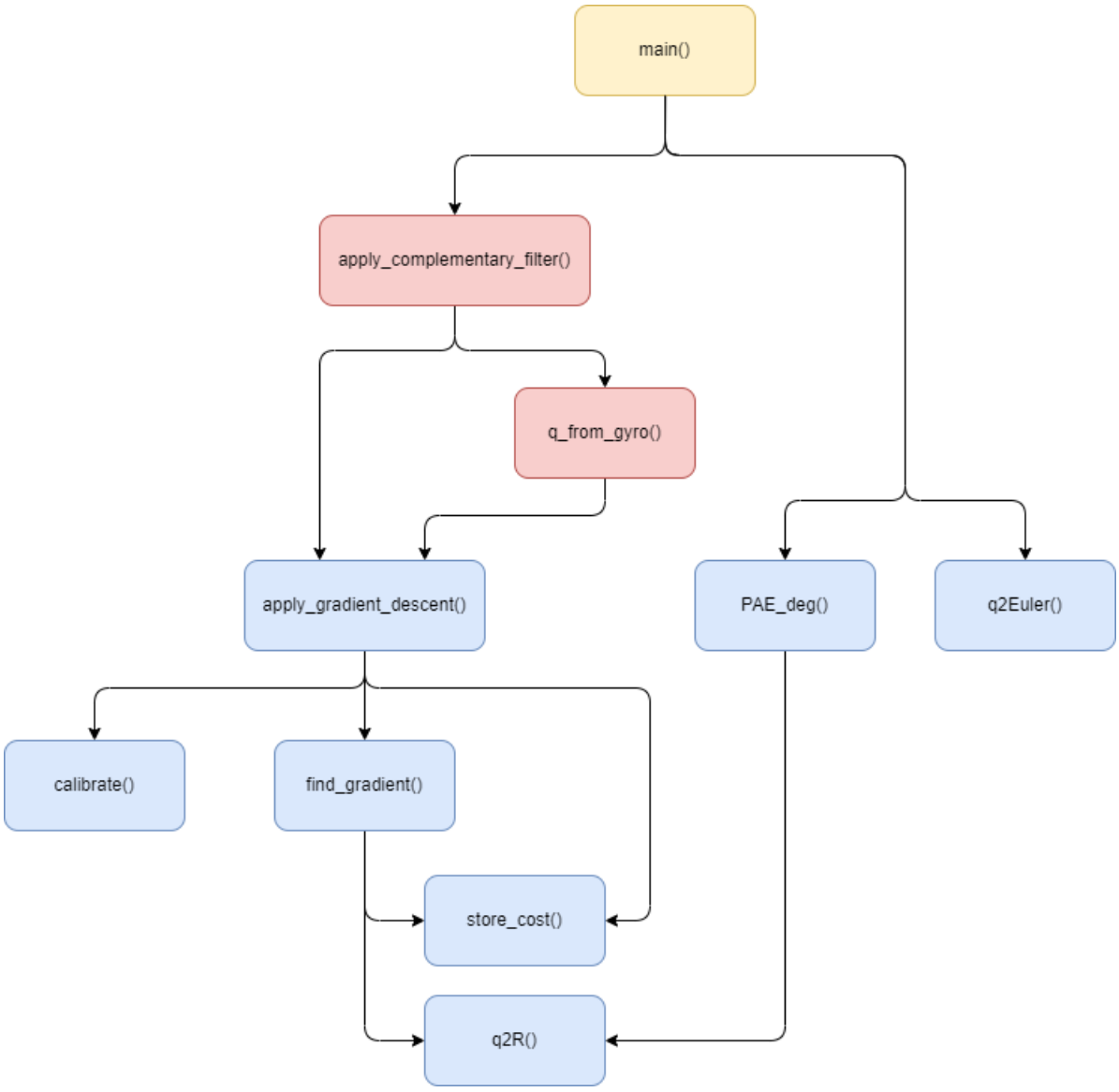
Primjer 5.3: Programska implementacija funkcije *q_from_gyro*() koja integriracijom mjerenja žiroskopa estimira vrijednost kvaterniona.

Ulomak glavne Matlab skripte prikazan je primjerom 5.4. Ona se sastoji od četiri dijela: u prvom dijelu koda učitavaju se mjerenja iz zasebnih .txt datoteka, dok se u drugom dijelu inicijaliziraju varijable potrebne za rad koda. Treći dio skripte sadrži for petlju u kojoj se izračunavaju rotacijski kvaternioni, pripadajući Eulerovi kutovi i pogreška glavnog kuta (eng. *Principal Angle Error*; PAE). Četvrti dio skripte sadrži kod za iscrtavanje grafova koji će biti analizirani u sklopu sljedećeg poglavlja. Kod četvrtog dijela nije prikazan na slici 5.4 zbog svojeg obujma.

```
clear all;
close all;
clc;

%% Load data

% Load artificial IMU data
load 'simulation_data_dynamic_noisy.txt'
S = simulation_data_dynamic_noisy;
clear simulation_data_dynamic_noisy;

% Load artificial ground truth quaternions
load 'simulation_data_dynamic_true_quaternion.txt'
q_true = simulation_data_dynamic_true_quaternion';
clear simulation_data_dynamic_true_quaternion;

%% Variable initialization

a = S(:,1:3)';  % Accelerometer data
m = S(:,4:6)';  % Magnetometer data
w = S(:,7:9)';  % Gyroscope data

fs = 10;        % Sensor sampling frequency
dT = 1/fs;      % Period

T = 1000;
t = linspace(0, T, T*fs + 1);

% Predictive propagation method
est_predictive = ComplementaryFilter();
est_predictive.propagation_type = 'predictive';
est_predictive.CF_history_flag = true;

% PAE
PAE_predictive = [];
PAE_limited = [];

% Euler angles
Euler_predictive = [];
Euler_limited = [];
Euler_true = [];

%% Data calculation

for i = 1:length(a)
        q_predictive = est_predictive.apply_complementary_filter(a(:,i), m(:,i), w(:,i
            ));
        PAE_predictive = [PAE_predictive, est_predictive.PAE_deg(q_predictive, q_true
            (:,i))];
        Euler_predictive = [Euler_predictive, est_predictive.q2Euler(est_predictive.
            q_history(:,i))];
        Euler_true = [Euler_true, est_predictive.q2Euler(q_true(:,i))];
end

%% Graph plotting

% ...
```

Primjer 5.4: Programska implementacija *main* funkcije.

Konačno, hijerarhija funkcija unutar programskog koda kao cjeline prikazana je blokovskim dijagramom na slici 5.2. Crveni blokovi blokovskog dijagrama označavaju funkcije klase *ComplementaryFilter*, plavo obojani blokovi označavaju funkcije klase *GradientDescent*, a strelice daju informaciju o tome koje sve funkcije određena funkcija poziva.



Slika 5.2: Blokovska shema koja prikazuje hijerarhiju funkcija u programskom kodu.

Poglavlje 6

Verifikacija i evaluacija performansi algoritma

Rad komplementarnog filtra čija je implementacija temeljena na algoritmu gradijentnog spusta testiran je simulacijski i eksperimentalno. U narednom ćemo tekstu objasniti oba pristupa te dobivene rezultate.

6.1 Rezultati dobiveni smulacijskom verifikacijom algoritma

Simulacijska verifikacija rada algoritma provedena je korištenjem umjetno generiranih mjerenja IMU senzora, uključujući troosni akcelerometar, troosni žiroskop i troosni magnetometar. Kako bi se što vjernije simulirali stvarni uvjeti mjerenja, umjetno generiranim podacima dodan je Gaussov bijeli šum. Ovim pristupom omogućeno je testiranje performansi algoritma u uvjetima koji su što bliži stvarnim uvjetima mjerenja, uzimajući u obzir inherentne nepravilnosti i šumove stvarnih senzora.

Kao što je detaljno objašnjeno u ranijim poglavljima, implementirani algoritam komplementarnog filtra težinski kombinira rotacijski kvaternion izračunat algoritmom gradijentnog spusta na temelju mjerenja akcelerometra i magnetometra s rotacijskim kvaternionom dobivenim iz kutne brzine, kako bi se izračunao optimalni rotacijski kvaternion. Pritom je od ključne važnosti ispravno postavljanje parametara algoritma. Važni parametri gradijentnog spusta uključuju korak učenja μ, maksimalni broj iteracija N_{max} te kriterij terminacije koji određuje preciznost procijenjenog kvaterniona, G_{max}. Vrijednost koraka učenja odredili smo eksperimentalnim putem

s ciljem postizanja brze konvergencije prema optimalno procijenjenom kvaternionu u što manjem broju koraka, istodobno pazeći da algoritam ne završi u divergenciji. Vrijednost parametra korištena u simulacijama tako iznosi $\mu = 0.07$. Maksimalni broj iteracija je parametar čijim postavljanjem možemo spriječiti da algoritam zaglavi u beskonačnoj petlji u slučaju divergencije. Korištenje ovog parametra kao samostalnog kriterija terminacije izvođenja algoritma nije preporučljivo jer ne osigurava optimalan rad algoritma. Vrijednost koju smo odabrali u našoj simulaciji iznosi $N_{max} = 20$. Kriterij kojim terminiramo iteracijski proces u trenutku kada procijenjeni kvaternion dosegne korisnički zahtijevanu preciznost je G_{max}. Što je njegova vrijednost manja, to je veća zahtijevana preciznost. Vrijednost parametra G_{max} postavili smo na 3×10^{-2}. Posljednji parametar algoritma koji smo trebali odabrati jest pojačanje komplementarnog filtra K. Odabirom njegove vrijednosti odabiremo kojoj ćemo metodi procjene više vjerovati – gradijentnom spustu ili integraciji podataka žiroskopa. U ovoj smo simulaciji vrijednost parametra K postavili na 0.5 čime smo odredili jednaku razinu povjerenja za obje metode procjene. Konačno, referentna vrijednost vektora gravitacijske akceleracije postavljena je na

$$a^i = \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^T \text{ m/s}^2, \tag{6.1}$$

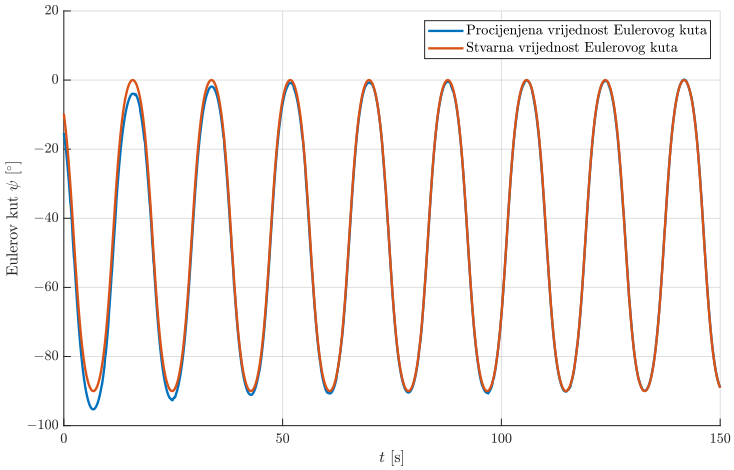
dok je referentna vrijednost vektora jakosti magnetskog polja Zemlje postavljena na

$$m^i = \begin{bmatrix} 22.2 & 1.7 & 42.7 \end{bmatrix}^T \ \mu\text{T}. \tag{6.2}$$

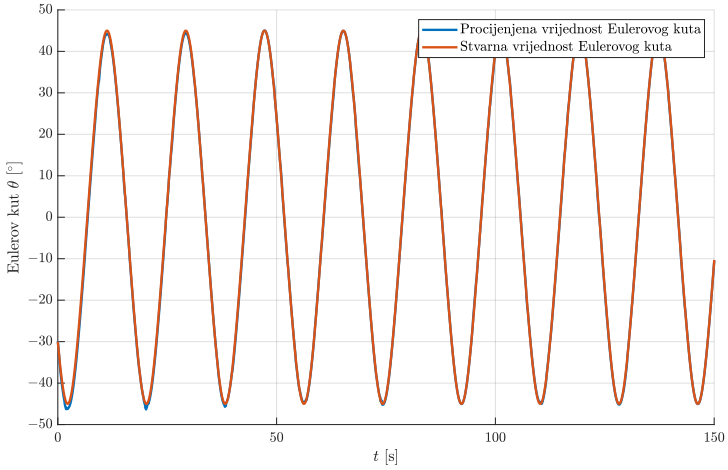
.

Ta je vrijednost magnetskog polja preuzeta iz Svjetskog magnetskog modela (eng. *World Magnetic Model*) ([22]) za grad Zagreb.

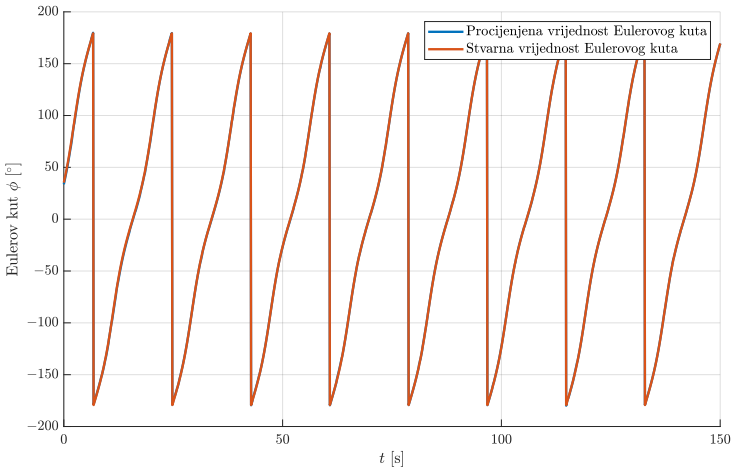
U provedenoj simulaciji pretpostavlja se rotacija objekta oko proizvoljno odabranog vektora brzinom od 20 stupnjeva po sekundi. Za početne vrijednosti kvaterniona i kuta zakreta odabrani su $q_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$ i $\alpha_0 = 45°$. Grafovi prikazani slikom 6.1 prikazuju vrijednosti Eulerovih kutova koje su izračunate iz rotacijskih kvaterniona pomoću funkcije $q2Euler()$ (njezina je implementacija prikazana primjerom 6.1).
Uočavamo kako izračunati Eulerovi kutovi izvrsno prate stvarne (eng. *ground truth*) vrijednosti te da je zbog toga na većem dijelu grafova nemoguće razlikovati dvije krivulje.



(a) Eulerov kut ψ.



(b) Eulerov kut θ.



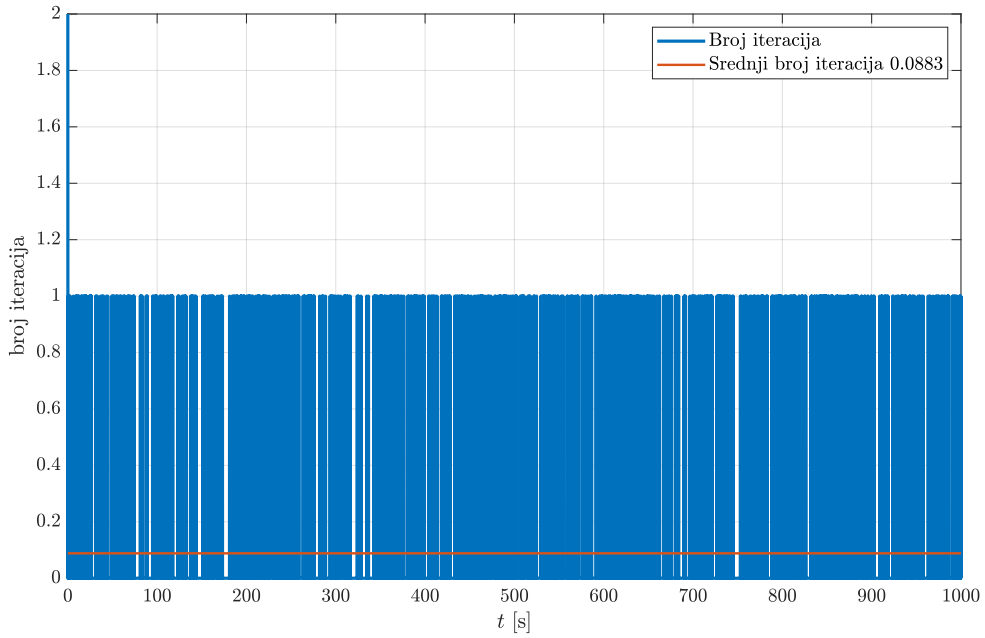
(c) Eulerov kut ϕ.

Slika 6.1: Usporedba procijenjenih Eulerovih kuteva s *ground-truth* vrijednostima Eulerovih kuteva. Parametri simualcije su: $\mu = 0.07$, $N_{max} = 20$, $G_{max} = 3 \times 10^{-2}$ i $K = 0.5$.

```
function euler_angles = q2Euler(~, q)
        % Calculating Euler angles (z-y-x, psi-theta-phi) from quaternion
        psi = atan2(2*(q(2)*q(3)+q(1)*q(4)), q(1)^2+q(2)^2-q(3)^2-q(4)^2);
        theta = asin(-2*(q(2)*q(4)-q(1)*q(3)));
        phi = atan2(2*(q(3)*q(4)+q(1)*q(2)), q(1)^2-q(2)^2-q(3)^2+q(4)^2);

        % Converting to degrees and storing in column vector
        euler_angles = rad2deg([psi, theta, phi]');
end
```

Primjer 6.1: Programska implementacija funkcije *q2Euler*() koja iz primljenog kvaterniona računa odgovarajuće vrijednosti Eulerovih kuteva.

Za istu simulaciju generirali smo grafove koji opisuju broj iteracija i pogrešku glavnog kuta (PAE). Na slici 6.2 vidimo da je srednji broj iteracija manji od 1, što implicira da, prema našem kriteriju zaustavljanja definiranom na ranije opisan način, algoritam u nekim koracima ne primjenjuje niti jednu iteraciju gradijentnog spusta. To je moguće zahvaljujući prediktivnom određivanju početnog kvaterniona u gradijentnom spustu.

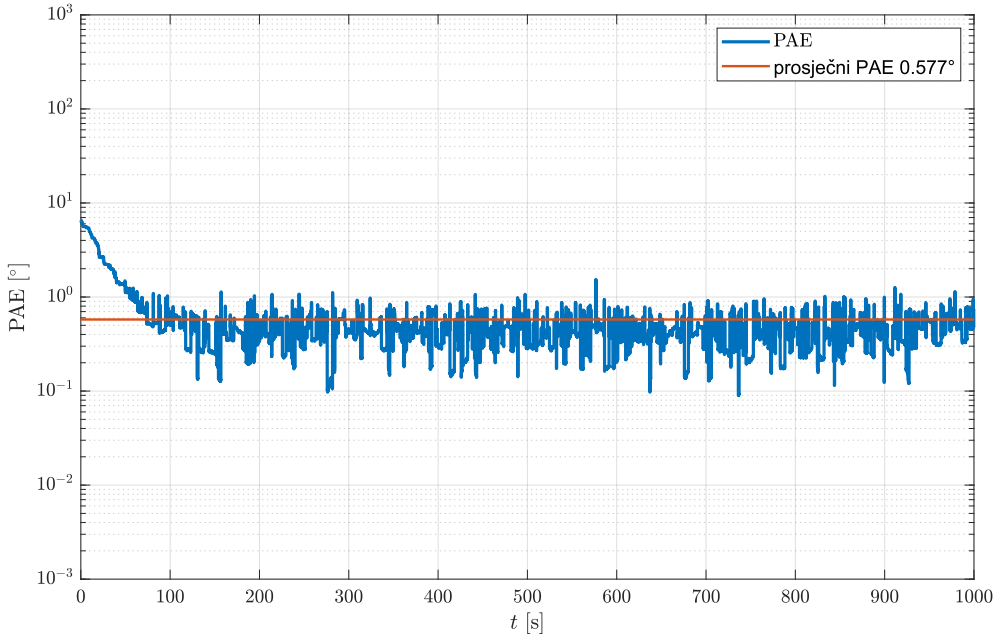


Slika 6.2: Broj iteracija.

Kvalitetu postignutih rezultat potvrđuje i PAE prikazan slikom 6.3. *Principle Angle Error* ili PAE mjera je koja se često koristi u navigacijskim primjenama za procjenu greške u određivanju kutne razlike između dvije orijentaciji. PAE se definira kao najmanji kut između dvaju vektora koji reprezentiraju dvije orijentacije. Za svaki uzorak, PAE se izračunava prema relaciji

$$PAE = \arccos\left(\frac{1}{2}\left(Tr(R^T\hat{R}) - 1\right)\right). \tag{6.3}$$

gdje je $R^T \hat{R}$ korektivna rotacijska matrica sačinjena od stvarne matrice R i estimirane rotacijske matrice $\hat{R}$. Što je manja vrijednost PAE-a, to je veća preciznost određivanja kutne razlike između orijentacija. Budući da prosječni PAE u našoj simulaciji iznosi 0.577°, te da smo očito dosegnuli razinu šuma (eng. *noise floor*), zaključujemo još jednom kako smo ispravno odabrali parametre algoritma (prije svega, kriterij terminacije G_{max}).



Slika 6.3: *Principle Angle Error* (PAE).

Programska implementacija izračuna PAE ostvarena je u sklopu klase *GradientDescent* te je prikazana primjerom 6.2.

```
function PAE = PAE_deg(obj, q_estimated, q_true)
        R_true = obj.q2R(q_true);
        R_estimated = obj.q2R(q_estimated);
        R_corrective = R_true'*R_estimated;

        PAE = acos(0.5*(trace(R_corrective)-1));
        PAE = rad2deg(PAE);
end
```

Primjer 6.2: Programska implementacija funkcije *PAE_deg()* koja računa vrijednost PAE u stupnjevima.

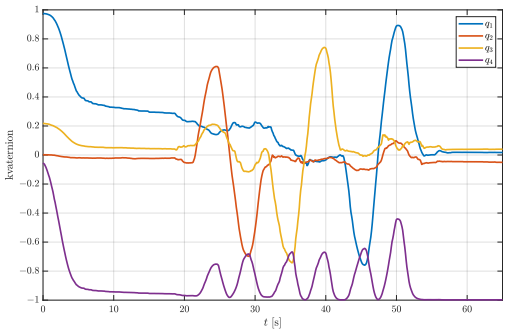
6.2 Rezultati dobiveni eksperimentalnom verifikacijom algoritma

Eksperimentalna verifikacija algoritma provedena je na prototipu ADCS sustava i pripadajućem testnom postavu koje smo detaljno opisali u drugom poglavlju ovog rada. Za razliku od simulacijske verifikacije, podatci senzora pomoću kojih se izvodi algoritam nisu generirani softverskim putem, već su snimljeni realnim senzorima IMU MPU9250 modula. Mjerenja su uzorkovana frekvencijom od 10 Hz te su na računalo na kojem se izvodi kod komplementarnog filtra poslana Bluetooth vezom. Za početnu vrijednosti kvaterniona odabrana je $q_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$, kao i u simulacijskoj verifikaciji sustava. Vrijednosti parametara μ, N_{max}, G_{max} i K također odgovaraju vrijednostima koje smo definirali u simulacijskoj verifikaciji sustava. Tijekom eksperimentalne verifikacije nismo koristili zračni ležaj, već samo ADCS sustav koji je bio postavljen u kuglu za testiranje. Takav smo sustav zakrenuli za 90° oko svake osi u pozitivnom i negativnom smjeru. Podatke koje smo snimili u ovom eksperimentu upotrijebili smo za izračun procijenjenog orijentacijskog kvaterniona, Eulerovih kutova, broja potrebnih iteracija za svaki uzorak mjerenja i početnu vrijednost funkcije cilja u svakom koraku. Izračunati podatci prikazani su grafovima na slici 6.4. Grafovi 6.4a i 6.4b vrlo jasno prikazuju tri slijedne rotacije oko x, y i z-osi, dok nam grafovi 6.4c i 6.4d prikazuju rezultate očekivane od realnog scenarija. Uočimo kako je na grafu 6.4c prosječan broj iteracija značajno veći u odnosu na prosječan broj iteracija dobiven simulacijskom verifikacijom algoritma. Povećan broj iteracija posljedica je ne samo uporabe realnih senzora, koji imaju svoje nedostatke i ograničenja, već i odabira početnog kvaterniona q_0 čija se pretpostavljena vrijednost značajno razlikovala od vrijednosti *ground truth* kvaterniona. Navedenu razliku u vrijednostima primjećujemo i na grafu 6.4d, gdje se vidi da inicijalna vrijednost funkcije cilja poprima maksimalnu vrijednost u početnom trenutku eksperimenta, a zatim se postupno smanjuje. Takvo je ponašanje očekivano ponašanje jer smo u algoritmu upotrijebili prediktivnu propagacijsku metodu inicijalnog kvaterniona koji se koristi u gradijentnom spustu. Naposljetku, udaljenost pretpostavljene vrijednosti početnog kvaterniona od njegove stvarne vrijednosti vidimo i na dugačkoj tranzijentnoj pojavi prisutnoj na početku grafa 6.4a.

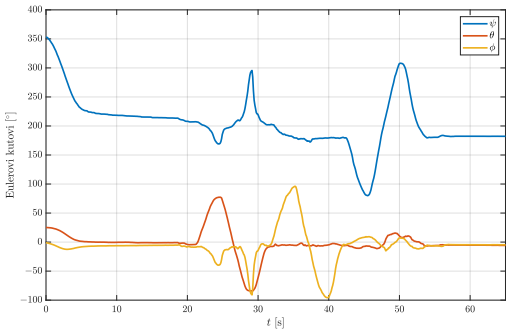
Čitatelje upućujemo na snimku[1] eksperimentalne verifikacije sustava koja prikazuje zakretanje

[1] https://youtu.be/5-WwKcSYmwY

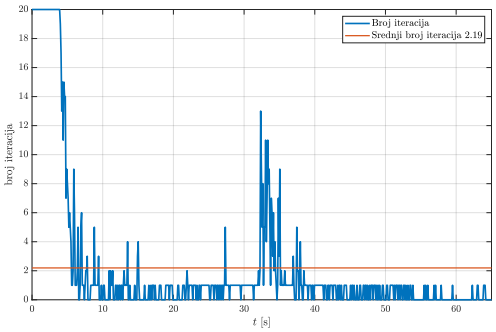
ADCS sustava u 3D prostoru, usporedno s animacijom koja se temelji na procjeni orijentacije dobivenoj korištenjem implementiranog algoritma. Na taj način se kvaliteta ostvarene procjene orijentacije jasno i nedvosmisleno očituje.



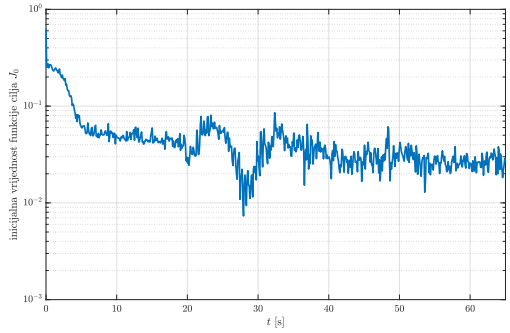
(a) Procijenjeni rotacijski kvaternion.



(b) Procijenjeni Eulerovi kutovi.



(c) Potreban broj iteracija za svaki set mjerenja.



(d) Inicijalna vrijednost funkcije cilja J_0 u svakom koraku.

Slika 6.4: Eksperimentalni rezultati dobiveni sljedećim parametrima algoritma: $\mu = 0.07$, $q_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$, $N_{max} = 20$, $G_{max} = 3 \times 10^{-2}$, i $K = 0.5$.

Poglavlje 7

Zaključak

U radu je predstavljena implementacija komplementarnog filtra temeljena na algoritmu gradijentnog spusta, čija je svrha određivanje orijentacije modela satelitskog ADCS sustava. Sustav se sastoji upravljačke elektronike i testnog postava koje smo samostalno izradili. Implementirani komplementarni filtar težinski zbraja procjenu orijentacijskog kvaterniona dobivenu algoritmom gradijentnog spusta i procjenu orijentacijskog kvaterniona dobivenu iz mjerenja kutne brzine te tako ostvaruje njihovo niskopropusno, odnosno visokopropusno filtriranje. Niskopropusni i visokopropusni filtar pritom imaju istu vrijednost granične frekvencije čime se iskorištavaju dobre karakteristike senzora, odnosno smanjuje se utjecaj njihovih negativnih karakteristika. Na taj način niskopropusnim filtriranjem možemo eliminirati visokofrekvencijske komponente šuma u mjerenjima akcelerometra i magnetometra, dok se visokofrekvencijskim filtriranjem uklanja akumulirano odstupanje žiroskopa. Pažljivim odabirom vrijednosti parametara algoritma moguće je značajno poboljšati ili unazaditi njegovu izvedbu. Neki od najvažnijih parametara gradijentnog spusta su korak učenja μ, maksimalni dozvoljeni broj iteracija N_{max}, kriterij terminacije algoritma G_{max} te početni kvaternion $\mathbf{q_0}$. Kod primjene komplementarnog filtra ključno je odabrati adekvatnu vrijednost koeficijenta skaliranja K, budući da njome definiramo razinu pouzdanosti u određenu metodu određivanja kvaterniona. Kvalitetu implementacije algoritma, kao i odabranih vrijednosti parametara testirali smo simulacijskim i eksperimentalnim putem. Simulacijska verifikacijska temeljila se na softverski generiranim mjerenjima IMU senzora, te je dala izvrsne rezultate koji se očituju u dobrom poklapanju estimiranih Eulerovih kutova sa stvarnim Eulerovim kutovima, srednjem broju iteracija koji je manji od 1, te niskoj vrijednosti pogreške glavnog kuta. Eksperimentalnom verifikacijom dan je uvid u rad algoritma na realnom sustavu koji podatke pribavlja iz realnih senzora IMU jedinice. Zbog nesavršenosti

i osjetljivosti tih senzora na vanjske čimbenike, srednji je broj iteracija u eksperimentalnoj verifikaciji nešto veći nego u simulacijama. Unatoč tome, kvaliteta procjenjivanja orijentacije implementiranim algoritmom zadovoljavajuće je točnosti, što je ilustrirano na realnom ADCS sustavu videosnimkom eksperimentalnog testiranja.

Poglavlje 8

Zahvale

Zahvaljujem svojem mentoru doc. dr. sc. Josipu Lončaru na poticaju i prilici da surađujem s njim u izradi znanstvenih radova, te na dragocjenoj pomoći pruženoj tijekom izrade ovog rada.

Literatura

[1] V. Subramanian, T. Burks, and W. Dixon, "Sensor fusion using fuzzy logic enhanced kalman filter for autonomous vehicle guidance in citrus groves," *Transactions of the ASABE*, vol. 52, no. 5, pp. 1411–1422, 2009.

[2] W. Farag, "Kalman-filter-based sensor fusion applied to road-objects detection and tracking for autonomous vehicles," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 235, no. 7, pp. 1125–1138, 2021.

[3] A. Werries, J. Dolan, *et al.*, "Adaptive kalman filtering methods for low-cost gps/ins localization for autonomous vehicles," tech. rep., Carnegie-Mellon University, 2016.

[4] F. A. Ghaleb, A. Zainal, M. A. Rassam, and A. Abraham, "Improved vehicle positioning algorithm using enhanced innovation-based adaptive kalman filter," *Pervasive and Mobile Computing*, vol. 40, pp. 139–155, 2017.

[5] Y. Liu, X. Fan, C. Lv, J. Wu, L. Li, and D. Ding, "An innovative information fusion method with adaptive kalman filter for integrated ins/gps navigation of autonomous vehicles," *Mechanical Systems and Signal Processing*, vol. 100, pp. 605–616, 2018.

[6] M. Gowda, J. Manweiler, A. Dhekne, R. R. Choudhury, and J. D. Weisz, "Tracking drone orientation with multiple gps receivers," in *Proceedings of the 22nd annual international conference on mobile computing and networking*, pp. 280–293, 2016.

[7] K. Yoshimura, B. M. Nguyen, H. Fujimoto, M. Kawanishi, and T. Narikiyo, "Dual-rate altitude control of drones based on double-layer kalman filter," in *8th IEEJ International Workshop on Sensing, Actuation, Motion Control, and Optimization*, pp. 485–488, 2022.

[8] R. Negenborn, "Robot localization and kalman filters," *Utrecht Univ., Utrecht, Netherlands, Master's thesis INF/SCR-0309*, 2003.

[9] T. D. Larsen, K. L. Hansen, N. A. Andersen, and O. Ravn, "Design of kalman filters for mobile robots; evaluation of the kinematic and odometric approach," in *Proceedings of the 1999 IEEE international conference on control applications (Cat. No. 99CH36328)*, vol. 2, pp. 1021–1026, IEEE, 1999.

[10] K. Ramachandra, *Kalman filtering techniques for radar tracking*. CRC Press, 2018.

[11] J. B. Pearson and E. B. Stear, "Kalman filter applications in airborne radar tracking," *IEEE Transactions on Aerospace and Electronic systems*, no. 3, pp. 319–329, 1974.

[12] D. R. Kumar, "Hybrid unscented kalman filter with rare features for underwater target tracking using passive sonar measurements," *Optik*, vol. 226, p. 165813, 2021.

[13] L. Jetto, S. Longhi, and G. Venturini, "Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 2, pp. 219–229, 1999.

[14] R. Darbali-Zamora, N. Cobo Yepes, E. Ortiz-Rivera, E. Aponte, and A. Rincon, "Applying hol/pbl to prepare undergraduate students into graduate level studies in the field of aerospace engineering using the puerto rico cubesat project initiative," 10 2018.

[15] F. Gonçalves, T. Ribeiro, A. F. Ribeiro, G. Lopes, and P. Flores, "A recursive algorithm for the forward kinematic analysis of robotic systems using euler angles," *Robotics*, vol. 11, no. 1, p. 15, 2022.

[16] J. L. Marins, X. Yun, E. R. Bachmann, R. B. McGhee, and M. J. Zyda, "An extended kalman filter for quaternion-based orientation estimation using marg sensors," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, vol. 4, pp. 2003–2011, IEEE, 2001.

[17] R. Labs, "Understanding euler angles," 2018.

[18] J. C. Chou and M. Kamel, "Finding the position and orientation of a sensor on a robot manipulator using quaternions," *The international journal of robotics research*, vol. 10, no. 3, pp. 240–254, 1991.

[19] K. Sever, I. Indir, I. Vnučec, and J. Lončar, "Evaluation of gradient descent algorithm for attitude estimation," in *2021 International Symposium ELMAR*, pp. 131–134, IEEE, 2021.

[20] K. Sever, L. M. Golušin, and J. Lončar, "Optimization of gradient descent parameters in attitude estimation algorithms," *Sensors*, vol. 23, no. 4, p. 2298, 2023.

[21] J. Wu, Z. Zhou, M. Song, H. Fourati, and M. Liu, "Convexity analysis of optimization framework of attitude determination from vector observations," in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 440–445, IEEE, 2019.

[22] National Centers for Environmental Information, "Geomagnetic Models and Software." `https://www.ngdc.noaa.gov/geomag`, 2022. [Online; accessed 28-November-2022].

Sažetak

Karla Sever, *Implementacija komplementarnog filtra za procjenu orijentacije modela ADCS sustava temeljena na gradijentnom spustu*

Točno i pouzdano određivanje orijentacije umjetnih satelita složen je i iznimno važan postupak u modernim svemirskim misijama. O njemu ne ovise samo daljnji postupci kao što su korekcija trenutne orijentacije i održavanje ispravne orijentacije, već i uspjeh cijele misije za koju su usmjerenost mjernih i komunikacijskih sustava satelita važni. Iz tog razloga, sateliti u svoju strukturu imaju ugrađen zaseban sustav po imenu ADCS (eng. *Attitude Determination and Control System*). U ovom radu, prikazana je praktična realizacija algoritma za određivanje orijentacije ADCS sustava temeljena na komplementarnom filtru, tehnici senzorske fuzije prikladnoj za primjenu na sustavima niske energetske potrošnje. Implementirani komplementarni filtar kombinira dva rotacijska kvaterniona određena korištenjem podataka iz različitih senzora IMU jedinice i dviju različitih metoda kako bi se procijenio optimalni rotacijski kvaternion koji opisuje zakret satelitskog koordinatnog sustava u odnosu na Zemljin koordinatni sustav. U radu je izložena detaljna matematička pozadina implementiranog komplementarnog filtra, kao i iterativnog optimizacijskog algoritma gradijentnog spusta koji se nalazi u njegovoj pozadini. Osim toga, prikazani su ulomci programskog koda na kojima je objašnjena implementacija algoritma te važnost ispravnog odabira njegovih parametara. Učinkovitost i kvaliteta rada algoritma provjereni su simulacijom i eksperimentalno, pri čemu je eksperimentalna provjera provedena na vlastitom prototipu ADCS sustava kojeg smo razvili na Fakultetu elektrotehnike i računarstva. Predstavljeni je algoritam u oba postupka verifikacije pokazao iznimno dobru točnost procjene orijentacije ADCS sustava.

Ključne riječi: komplementarni filtar, gradijentni spust, određivanje orijentacije, ADCS

Summary

Karla Sever, *Gradient descent based implementation of a complementary filter for the orientation estimation of the ADCS system model.*

Accurate and reliable orientation determination of artificial satellites is a complex and extremely important process in modern space missions. It is not only crucial for further procedures such as correction of the current orientation and maintaining the correct orientation, but also for the success of the entire mission, for which the orientation of the measuring and communication systems of the satellite is important. For this reason, satellites have a built-in separate system in their structure called ADCS (Attitude Determination and Control System). This paper presents a practical implementation of the ADCS system's algorithm for attitude estimation based on a complementary filter, a sensor fusion technique suitable for low-power systems. The implemented complementary filter combines two rotational quaternions determined by using data from different IMU unit sensors and by two different methods in order to estimate the optimal rotational quaternion, which describes the rotation of the satellite coordinate system relative to the Earth coordinate system. The paper presents a detailed mathematical background of the implemented complementary filter, as well as an iterative optimization algorithm of gradient descent that is used in the filter. In addition, fragments of the program code are presented that explain the implementation of the algorithm and the importance of selecting its parameters correctly. The effectiveness and quality of the algorithm are verified through simulation and experimental verification, the latter of which was conducted on our own prototype ADCS system developed at the Faculty of Electrical Engineering and Computing. The presented algorithm showed exceptional accuracy in estimating the orientation of the ADCS system in both verification procedures.

Key words: complementary filter, gradient descent, orientation estimation, ADCS