Programska podrška sustava za određivanje i upravljanje orijentacijom satelita

Vnučec, Ivan

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:741398

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-03**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2346

PROGRAMSKA PODRŠKA SUSTAVA ZA ODREĐIVANJE I UPRAVLJANJE ORIJENTACIJOM SATELITA

Ivan Vnučec

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2346

PROGRAMSKA PODRŠKA SUSTAVA ZA ODREĐIVANJE I UPRAVLJANJE ORIJENTACIJOM SATELITA

Ivan Vnučec

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

DIPLOMSKI ZADATAK br. 2346

Pristupnik:	**Ivan Vnučec (0036507815)**
Studij:	Elektrotehnika i informacijska tehnologija
Profil:	Elektroničko i računalno inženjerstvo
Mentor:	doc. dr. sc. Josip Lončar

Zadatak: **Programska podrška sustava za određivanje i upravljanje orijentacijom satelita**

Opis zadatka:

Nanosateliti CubeSat formata postaju sve popularniji u komercijalnim i znanstvenim misijama. Broj lansiranih nanosatelita raste iz godine u godinu. Sustav za određivanje i upravljanje orijentacijom satelita jedan je od najvažnijih sustava odgovoran za stabilno i ispravno usmjeravanje satelita i njegovog korisnog tereta kao što su kamere, senzori, antene i slično. Zadatak ovog sustava je pomoću dostupnih senzora odrediti orijentaciju satelita te je automatski ispraviti s obzirom na željenu orijentaciju koristeći se dostupnim aktuatorima. Tema diplomskog rada je razvoj programske podrške za model satelita koja omogućuje pouzdano određivanje i upravljanje orijentacijom satelita i njegovom kutnom brzinom. U radu je potrebno opisati korištene senzore i aktuatore, način matematičkog zapisa orijentacije satelita te odabranu metodu upravljanja orijentacijom i kutnom brzinom satelita. Potrebno je implementirati programsku podršku za određivanje i upravljanje orijentacijom na ugradbenom računalnom sustavu te objasniti osnovne blokove programskog koda uz pripadne grafove i dijagrame toka. U radu je također potrebno razviti osnovni kinematički model satelita, opisati optimizaciju odabranog regulatora za upravljanje orijentacijom te izložiti rezultate eksperimentalne verifikacije implementiranih algoritama.

Rok za predaju rada: 27. lipnja 2022.

Posebno hvala mentoru Josipu Lončaru na razumijevanju, nesebičnoj pomoći i susretljivosti. Da nije bilo tebe Josipe, radio bih Web development.
Jednako tako hvala i Karli Sever, bez tvog truda moj softver ne bi imao hardver.

SADRŽAJ

1. Uvod **1**

1.1. Mali sateliti 1

1.1.1. Podjela 1

1.1.2. Metode lansiranja 3

1.1.3. Povijesni pregled misija 5

1.2. Korisni teret satelita 5

1.2.1. Vrste tereta 5

1.2.2. Važnost kontrolirane orijentacije 6

2. Opis orijentacije satelita **7**

2.1. Eulerovi kutovi 7

2.2. Kvaternioni 8

2.3. Rotacijska matrica 9

2.3.1. Prikaz pomoću Eulerovih kutova 9

2.3.2. Prikaz pomoću kvaterniona 11

2.3.3. Rotacija vektora pomoću rotacijske matrice 13

2.4. Jednadžba rotacijskog gibanja 13

2.4.1. Prikaz pomoću Eulerovih kutova 14

2.4.2. Prikaz pomoću kvaterniona 14

2.5. Usporedba Eulerovih kutova i kvaterniona 15

2.6. Pojednostavljeni model satelita 15

3. Sustav za određivanje i upravljanje orijentacijom satelita (ADCS) **17**

3.1. Uvod 17

3.1.1. Određivanje i upravljanje orijentacijom 17

3.1.2. Važnost ADCS sustava 18

3.2. Osnovni dijelovi 18

3.2.1. Računalo 18

3.2.2. Senzori orijentacije 19
3.2.3. Senzori kutne brzine 24
3.2.4. Aktuatori 25
3.3. Algoritmi za određivanje i upravljanje orijentacijom 29
3.3.1. Određivanje orijentacije 29
3.3.2. Upravljanje orijentacijom/kutnom brzinom 34
3.4. Razvijeni sustav 35
3.4.1. Tiskana pločica i sklopovlje 35
3.4.2. Izabrani senzori 39
3.4.3. Izabrani aktuatori 40

4. Programska podrška ADCS sustava **41**
4.1. Ugradbeno računalo 41
4.2. Organizacija projekta 41
4.3. Korištene biblioteke 42
4.4. Operacijski sustav i funkcionalnost 43
4.4.1. Operacijski sustav i dretve 44
4.5. Razvoj 50

5. Eksperimentalna verifikacija ADCS sustava **52**
5.1. Opis sustava i korištenih alata 52
5.2. Određivanje parametara 55
5.2.1. Pojednostavljeni model satelita 55
5.2.2. PID regulator 57
5.3. Rezultati verifikacije 59

6. Zaključak **63**

Literatura **64**

1. Uvod

1.1. Mali sateliti

Mali sateliti (engl. *Small satellites*) su skupina satelita koji su, u usporedbi s konvencionalnim satelitima, uvelike ograničeni prema masi, cijeni, kao i prema vremenu potrebnom da ih se razvije [21].

Primjene malih satelita nalazimo u svim granama svemirske industrije: u telekomunikaciji, navigaciji, opservaciji Zemlje, znanstvenim istraživanjima i dr.

Njihova zastupljenost na tržištu raste iz godine u godinu jer pružaju jeftin i brz razvoj uz relativno male tehničke zahtjeve [54]. Sve je to izravan rezultat revolucije u elektronici i računarstvu [21]. Primjenom većeg broja malih satelita povezanih u konstelaciju, moguće je primjerice pokriti veću površinu zemlje i tako stvoriti uvijek dostupni internet ili navigacijski sustav.

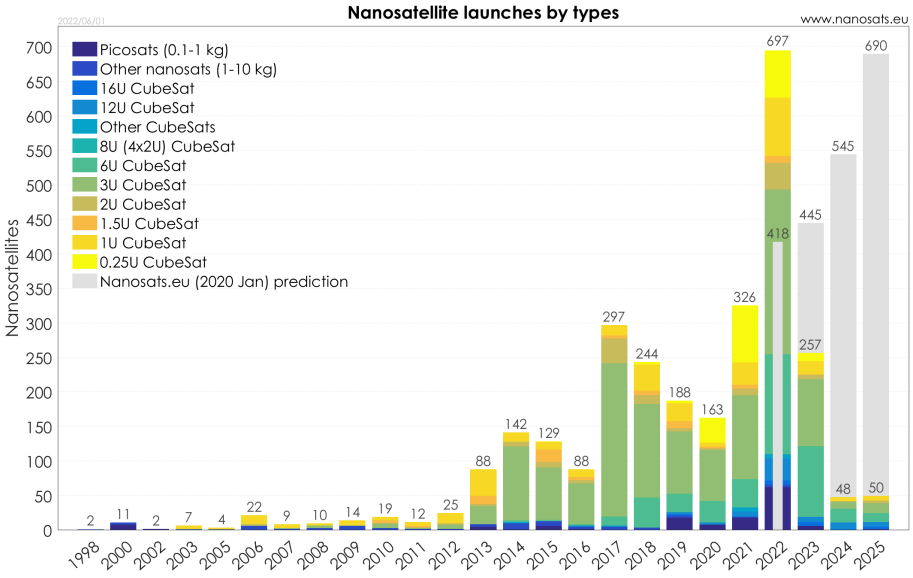
1.1.1. Podjela

Iako postoje više podjela satelita u ovisnosti od izvora do izvora, navest ćemo podjelu prema FAA (engl. *Federal Aviation Administration*) koja uključuje popis od satelita s najvećom masom pa sve do satelita s najmanjom masom [52]. Podjelu je moguće vidjeti u tablici 1.1.

Maleni sateliti teže manje od 1200 kilograma i cijena razvoja im je nešto manja od 30 milijuna GBP (britanskih funti), za izradu i lansiranje je potrebno od 2. do 3. godine. Mini sateliti teže do 600 kilograma i njihov razvoj traje oko 2. godine sa cijenom do 30 milijuna GBP. Micro sateliti imaju masu manju od 200 kilograma i koštaju manje od 10 milijuna GBP. Ispod Micro klase nalaze se Nano sateliti mase do 10 kilograma, te Pico i Femto sateliti mase manje od 1. kilograma [21]. Popularnost malih satelita raste iz godine u godinu što prikazuje graf na slici 1.1.

Tablica 1.1: Podjela malih satelita prema masi

Naziv kategorije satelita	Masa (kg)
Small	601 - 1200
Mini	201 - 600
Micro	11 - 200
Nano	1.1 - 10
Pico	0.09 - 1
Femto	0.01 - 0.1



Slika 1.1: Broj lansiranja različitih tipova malih satelita kroz godine [12].

Cubesat format

Cubesat format malih satelita osmišljen je 1999. godine suradnjom Cal Poly sveučilišta u San Luis Obispu i Stanford sveučilišta. Projekt je imao za zadatak standardizirati Cubesat format i definirati njegove specifikacije. Cubesat format definiran je kao kocka veličine 10 cm $\times$ 10 cm $\times$ 10 cm (oznaka 1U) [64]. Danas se mogu kombinirati tri (3U), šest (6U) ili više takvih blokova koji tvore veći Cubesat satelit. 1U, 2U i 3U formati prikazani su na slici 1.2.

Netom nakon standardizacije Cubesat formata, 1U format postao je najpopularniji format za razvoj i lansiranje koji je pružao jednostavne funkcije kao što su: fotografiranje Zemlje, telekomunikacija te niz jednostavnijih mjerenja koje se trebaju prenijeti na zemlju [64].

Slika 1.2: Prikaz 1U, 2U i 3U Cubesat formata.

1.1.2. Metode lansiranja

Mali sateliti lansiraju se kao sekundarni teret u posebno izrađenim kavezima u kojima stane nekoliko takvih satelita. Za izbacivanje satelita u svemir koristi se opruga jer mali sateliti često nemaju propulziju. Nakon što je izdan izbacivački signal, mehanička opruga vrši silu na male satelite koji potom klizući šinama izlaze kroz vrata kaveza [64]. Prikaz kaveza možemo vidjeti na slici 1.3.

Let rakete koja nosi primarni i sekundarni teret osmišljen je tako da primarni teret dovede u planiranu orbitu i da ga tamo izbaci. Nakon primarnog tereta izbacuje se i sekundarni teret kao jedan ili više malih satelita. Lansiranja primarnog tereta danas su vrlo česta. Datumi lansiranja i orbita znaju se mjesecima unaprijed tako da korisnici sekundarnog tereta mogu planirati orbitu koju žele. Korisnici sekundarnog tereta mogu birati samo između ponuđenih orbita, ali ne mogu mijenjati orbitu ili datum lansiranja. Cijene lansiranja 1U Cubesat-a kreću se oko 90.000 eura, a za 3U Cubesat oko 200.000 eura. Jedna raketa može u svemir ponijeti 10 ili više Cubesat satelita [64].

Poslovanje lansiranja malih satelita organizirano je kroz pružatelje lansirnih usluga (engl. *Launch Services Provider - LSP*) koji zakupljuju prostor za sekundarni teret od poduzeća koje lansiraju rakete (engl. *Launch Logistics Provider - LLP*). Danas postoji više LSP-ova i stranka koja želi lansirati satelit ima mogućnost izbora po cijeni i kvaliteti usluge. Poduzeća koja lansiraju satelite u svemir nalaze se po cijelom svijetu [64].

Slika 1.3: Lansirni kavez Cubesat satelita [15].

Listu nekih pružatelja usluga lansiranja malih satelita moguće je vidjeti u tablici 1.2.

Tablica 1.2: Lista pružatelja usluga lansiranja malih satelita [64].

Ime	Lokacija	Web adresa
Innovative Solutions in Space	Nizozemska	`www.isispace.nl`
ECM Space	Njemačka	`www.ecm-space.de`
UT AeroSpace Studies	Kanada	`www.utias-sfl.net`
CubeCab (3U+)	SAD	`www.cubecab.com`
Nano Avionics	Litva	`https://nanoavionics.com/`
Space Flight (3U+)	SAD	`https://spaceflight.com/`

Postupak lansiranja započinje potpisivanjem ugovora pri kojem korisnik usluge odabire jedan od ponuđenih datuma lansiranja i mogućih orbita. Pri potpisivanju ugovora korisnik plaća oko 50% cijene usluge. Potpisivanje ugovora se događa barem dvanaest mjeseci prije planiranog lansiranja. Tijekom slijedećih dva do šest mjeseca korisnik mora obaviti sva testiranja satelita. Šest mjeseci prije lansiranja, korisnik dobiva potvrdu o mjestu na raketi i plaća dodatnih 20% ugovorene cijene. Dva mjeseca

prije lansiranja, korisnik predaje satelit pružatelju lansirnih usluga koji satelit prevozi do lansirne lokacije. Stigavši na lansirnu lokaciju, korisnik plaća dodatnih 20% cijene. Konačno, u trenutku lansiranja, korisnik plaća posljednjih 10% cijene. Ukoliko lansiranje ne uspije nema povrata sredstava. Spomenuto obročno plaćanje ne vrijedi za sve pružatelje lansirnih usluga već neki pružatelji imaju drugačije oblike plaćanja. Nakon uspješnog lansiranja pružatelj lansirnih usluga prati orbitu svakog lansiranog satelita i predaje korisniku informacije o orbiti u kojoj se satelit nalazi [64].

1.1.3. Povijesni pregled misija

nanosats.eu baza podataka Nano i Cubesat satelita [12] sadrži preko 2000 podataka o lansiranjima Cubesat satelita od 1998. godine pa sve do danas. Prvo lansiranje Cubesat satelita zbilo se 2003. godine iz grada Plesetska u Rusiji u sklopu *Eurockot Launch Services's Multiple Orbit* misije. Cubesat sateliti koji su pušteni u orbitu bili su sateliti nekoliko tvrtki iz Danske, Japana, Kanade i SAD-a [14].

Nekoliko godina kasnije, točnije 2012. godine, lansirano je 11 Cubesat satelita pomoću rakete Atlas V. To je do tada bio najveći broj istovremeno lansiranih satelita gdje je najveći Cubesat satelit bio 24U formata [47].

Od 2012. godine do danas lansirano je preko 1000 malih satelita što je vidljivo na slici 1.1.

1.2. Korisni teret satelita

Korisni teret satelita (engl. *payload*) je podsustav ili skupina podsustava koji imaju za cilj ispuniti neku zadaću (misiju). Primjerice, ako je cilj satelita fotografiranje Zemljine površine, onda će njegov koristan teret biti sustav za fotografiranje, ili ako želimo mjeriti Zemljino magnetsko polje, onda će korisni teret biti sustav za mjerenje magnetskog polja.

1.2.1. Vrste tereta

Ovisno o tipu misije, podsustave je moguće svrstati u nekoliko kategorija:

- Opservacija Zemlje
- Komunikacija
- Navigacija
- Znanost i tehnologija

1.2.2. Važnost kontrolirane orijentacije

Sustavi za opservaciju Zemlje sadrže sklopovlje koje promatra Zemljinu površinu ili atmosferu. Najčešće takav sustav sadrži kameru koja fotografira površinu u odabranom dijelu spektra. Na takvom sustavu osim kamere imamo i objektive velikih uvećanja zbog kojih je potrebno imati mogućnost preciznog usmjeravanja kamere. Također, sustavi za komunikaciju osim sklopovlja za obradu signala sadrže i antene. Kako bi pospješili prijem i odašiljanje, potrebno je precizno usmjeravanje antene satelita prema anteni na Zemlji [1].

Osim gore opisanih sustava kojima je važna kontrola orijentacije, navest ćemo još jedan primjer gdje nam je kontrola orijentacije važna. Naime, prilikom izbacivanja satelita iz samog prijevoznog sredstva u svemir, događa se nekontrolirana rotacija satelita (engl. *tumbling*). Kako bi satelit zaustavio rotaciju, aktivni sustav za kontrolu orijentacije ulazi u posebni način rada (engl. *detumbling*) u kojem će zaustaviti nekontroliranu rotaciju. Takav zahvat može potrajati tjednima ili mjesecima, ovisno o tipu satelita [64].

2. Opis orijentacije satelita

Kako bi matematički prikazali rotacijsko gibanje satelita, satelit je prvo potrebno modelirati kao kruto tijelo. Mi ćemo u nastavku navesti samo osnovne izraze potrebne za modeliranje satelita, a potpune izraze i izvode moguće je pronaći u popratnoj literaturi [2].

Ako govorimo o orijentaciji, važno je objasniti pojam referentnog sustava (engl. *reference frame*). Referentni sustav je skup od 3 međusobno ortogonalnih vektora prema kojima određujemo sve ostale vektore: bilo njihovu orijentaciju, bilo pomak. Jedan možda najpoznatiji primjer takvog referentnog sustava je Kartezijev referentni sustav gdje smo kao referentne vektore izabrali 3 jedinična vektora $\overrightarrow{\boldsymbol{i}}$, $\overrightarrow{\boldsymbol{j}}$ i $\overrightarrow{\boldsymbol{k}}$ prema kojima onda određujemo sve ostale vektore u prostoru.

Moguće je imati više referentnih sustava gdje je prelazak iz jednog u drugi referentni sustav ostvaren pomoću transformacija s tzv. rotacijskim matricama (o kojima ćemo govoriti kasnije). Referentne sustave najčešće označavamo kao npr. $\mathcal{F}_a$ ili $\mathcal{F}_b$. Rotacijsku matricu koja određuje orijentaciju $\mathcal{F}_a$ referentnog sustava naspram $\mathcal{F}_b$ referentnog sustava označavamo kao $\boldsymbol{C}_{ab}$.

Da bi definirali orijentaciju satelita prvo moramo definirati referentne sustave. Prvi referentni sustav kojeg definiramo je tzv. inercijski referentni sustav $\mathcal{F}_G$. U klasičnoj fizici, inercijski sustav označava referentni sustav koji ne posjeduje akceleraciju [53]. Drugi referentni sustav kojeg definiramo je tzv. referentni sustav tijela (*body*) $\mathcal{F}_b$ koji je nepomičan s obzirom na gibanje čvrstog tijela (satelita). Orijentacija satelita u referentnom sustavu tijela $\mathcal{F}_b$ naspram inercijskog referentnog sustava $\mathcal{F}_G$ označavamo rotacijskom matricom $\boldsymbol{C}_{bG}$.

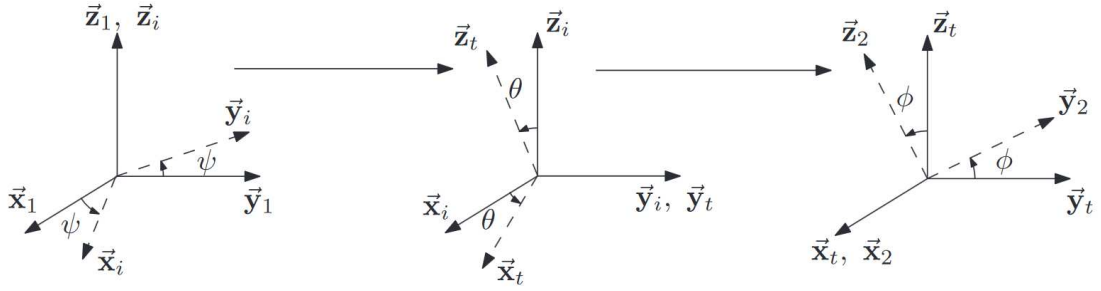
2.1. Eulerovi kutovi

Eulerovi kutovi predstavljaju 3 kuta između nekog odabranog vektora i ortogonalnih osi Kartezijevog koordinatnog sustava. Želimo li predstaviti rotaciju iz jednog vektora u drugi, to možemo napraviti na nekoliko načina. Jedan od načina navest ćemo u

sljedećem primjeru:

1. Rotacija za kut ψ oko originalne z-osi (engl. *yaw*).
2. Rotacija za kut θ oko tranzicijske y-osi (engl. *pitch*).
3. Rotacija za kut ϕ oko transformirane x-osi (engl. *roll*).

Uzmemo li za primjer 3 uzastopne rotacije: prve oko originalne z-osi, zatim druge oko tranzicijske y-osi i na kraju treće oko transformirane x-osi dobivamo rotaciju prikazanu na slici ispod:



Slika 2.1: Tri uzastopne rotacije oko z-osi, y-osi i x-osi za kut ψ, θ, i ϕ

2.2. Kvaternioni

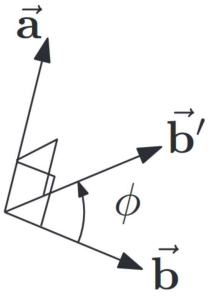
Kvaternion je četvero-dimenzionalni vektor s kojim je moguće opisati orijentaciju tijela. Kvaternion u usporedbi s Eulerovim kutovima posjeduje dvije prednosti (o kojima nešto više u nastavku): prva je da ne posjeduje singularnost, i druga da matematičke relacije modeliranja orijentacije više nisu trigonometrijske već algebarske prirode, što ih dodatno čini pogodnima za računanje.

Kvaternion u sebi sadrži dvije stvari: glavnu os rotacije (engl. *principal axis*) $\overrightarrow{\boldsymbol{a}}$ i kut rotacije ϕ kao što je vidljivo na slici 2.2. Zbog toga, razlikujemo vektorski i skalarni dio koje definiramo kao:

$$\begin{aligned} \boldsymbol{\epsilon} &= \overrightarrow{\boldsymbol{a}} \sin\tfrac{\phi}{2} = [\epsilon_x\ \epsilon_y\ \epsilon_z]\,, \\ \eta &= \cos\tfrac{\phi}{2} \end{aligned} \tag{2.1}$$

U praksi postoje dvije interpretacije kvaterniona u ovisnosti o tome je li skalarni dio prvi element kvaterniona ili posljednji. U našem radu pretpostavit ćemo oblik kvaterniona gdje je skalarni dio prvi, a vektorski drugi:

$$\boldsymbol{q} = [\eta\ \boldsymbol{\epsilon}] = [q_1\ q_2\ q_3\ q_4] \tag{2.2}$$



Slika 2.2: Prikaz rotacije vektora $\overline{b}$ oko glavne osi rotacije $\overline{a}$ za kut ϕ.

2.3. Rotacijska matrica

Rotacijska matrica opisuje transformaciju vektora iz originalnog u rotirani za neku poznatu rotaciju. U nastavku ćemo navesti rotacijske matrice koje određujemo pomoću Eulerovih kutova i pomoću kvaterniona i za svaku ćemo dati primjer rotacije vektora.

2.3.1. Prikaz pomoću Eulerovih kutova

Prije nego što ćemo navesti prikaz rotacijske matrice pomoću Eulerovih kutova prvo ćemo navesti kako izgledaju rotacijske matrice kada rotiramo vektore samo oko glavnih osi, a zatim ćemo navesti izraz rotacijske matrice za tzv. 3-2-1 uzastopnu rotacijsku sekvencu.

Rotacije oko glavnih osi

Bez ulaženja u detalje, navest ćemo redom kako izgledaju rotacijske matrice za rotacije oko z, y i x osi. Grafičke prikaze tih rotacija moguće je vidjeti na slici 2.3.

Rotaciju oko z-osi definiramo kao:

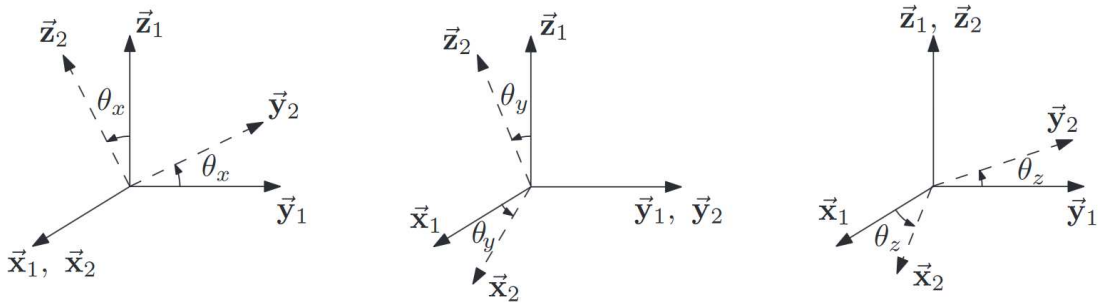
$$\boldsymbol{C}_z(\theta_z) = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

Rotacija oko y-osi:

$$\boldsymbol{C}_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix}. \tag{2.4}$$

Rotacija oko x-osi:

$$\boldsymbol{C}_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{bmatrix}. \tag{2.5}$$



Slika 2.3: Prikaz rotacija oko glavnih osi.

3-2-1 rotacijska sekvenca

Rotacijska sekvenca koja je jako česta u zrakoplovno-svemirskoj industriji označava se kao 3-2-1 orijentacijska sekvenca (engl. *3-2-1 attitude sequence*). Takvo ime je uzeto zato što je prva rotacija oko originalne z-osi (oznaka 3), zatim slijedi rotacija oko tranzicijske y-osi (oznaka 2) i na kraju rotacija oko transformirane x-osi (oznaka 1).

3-2-1 rotacijsku matricu moguće je prikazati pomoću Eulerovih kutova kao:

$$\begin{aligned} \boldsymbol{C}_{21}(\phi,\theta,\psi) &= \boldsymbol{C}_x(\phi)\boldsymbol{C}_y(\theta)\boldsymbol{C}_z(\psi) \\ &= \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{bmatrix}, \end{aligned} \tag{2.6}$$

gdje radi preglednosti pišemo $s_b = \sin(b)$ i $c_b = \sin(b)$.

Primijetimo kako smo 3-2-1 rotacijsku matricu dobili tako što smo izmnožili pojedinačne rotacijske matrice oko glavnih osi (vidi jednadžbe 2.3, 2.4 i 2.5). Važno je napomenuti kako u slučaju rotacija ne vrijedi komutativnost i da je redoslijed umnoška matrica vrlo bitan.

Singularnost Eulerovih kutova

Spomenuta mana reprezentacije rotacije pomoću Eulerovih kutova je tzv. singularnost. Može se pokazati da za svaku parametrizaciju rotacije može doći do singulariteta. Za

primjer 3-2-1 sekvence, singularitet nastaje onda kada je kut $\theta = \pm 90°$. Za takav slučaj rotacijska matrica postaje:

$$\boldsymbol{C}_{21}(\phi, 90°, \psi) = \begin{bmatrix} 0 & 0 & -1 \\ \sin(\phi - \psi) & \cos(\phi - \psi) & 0 \\ \cos(\phi - \psi) & -\sin(\phi - \psi) & 0 \end{bmatrix}. \tag{2.7}$$

Fizikalno zbog singulariteta, prva i treća rotacija 3-2-1 sekvence odvijaju se oko jedne te iste osi. U tom slučaju tzv. *roll* i *yaw* kutevi (ϕ i ψ) su zapravo jednaki kutovi i nije ih moguće jednoznačno odrediti (vidi sljedeće poglavlje). Izvan singulariteta je moguće jednoznačno odrediti sva tri Eulerova kuta.

Određivanje Eulerovih kutova iz rotacijske matrice

Ako rotacijsku matricu označimo kao:

$$\boldsymbol{C} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}, \tag{2.8}$$

iz jednadžbe 2.6 moguće je odrediti Eulerove kutove kao:

$$\begin{aligned} \phi &= \tan^{-1}(c_{23}/c_{33}), \\ \theta &= -\sin^{-1}(c_{13}), \\ \psi &= \tan^{-1}(c_{12}/c_{11}). \end{aligned} \tag{2.9}$$

Iznimno je važno da prilikom definiranja rotacije pomoću Eulerovih kutova naglasimo o kojoj sekvenci rotacije se radi (npr. 3-2-1) jer rotacijske sekvence ne komutiraju!

2.3.2. Prikaz pomoću kvaterniona

Rotacijsku matricu pomoću kvaterniona označavamo kao:

$$\boldsymbol{C} = (2\eta^2 - 1)\mathbf{1} + 2\boldsymbol{\epsilon}\boldsymbol{\epsilon}^T - 2\eta\boldsymbol{\epsilon}^\times, \tag{2.10}$$

gdje je $\boldsymbol{\epsilon}^\times$ definiran kao:

$$\boldsymbol{\epsilon}^\times = \begin{bmatrix} 0 & -\epsilon_z & \epsilon_y \\ \epsilon_z & 0 & -\epsilon_x \\ -\epsilon_y & \epsilon_x & 0 \end{bmatrix}. \tag{2.11}$$

Napominjemo da prilikom računanja rotacijske matrice kvaternion $\boldsymbol{q}$ prvo normaliziramo na jediničnu duljinu jer tako garantiramo da kasnije množenje rotacijske matrice i vektora neće rezultirati promjenom duljine vektora već samo promjenu smjera. Kvaternion normaliziramo na sljedeći način:

$$\boldsymbol{q}_{norm} = \frac{\boldsymbol{q}}{||\boldsymbol{q}||} = \frac{\boldsymbol{q}}{\sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}} \tag{2.12}$$

Ako li raspišemo rotacijsku matricu definiranu u jednadžbi 2.2 dobivamo izraz [11] koji je pogodniji za računanje:

$$\boldsymbol{C} = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2q_3 + q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3q_4 + q_1q_2) \\ 2(q_2q_4 + q_1q_3) & 2(q_3q_4 - q_1q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix} \tag{2.13}$$

Određivanje kvaterniona iz rotacijske matrice

Pomoću rotacijske matrice definirane u jednadžbi 2.8 moguće je odrediti sva 4 člana kvaterniona. Prvo određujemo skalarni član kao:

$$\eta = \pm\frac{(\text{trace}[\boldsymbol{C}] + 1)^{\frac{1}{2}}}{2}. \tag{2.14}$$

Predznak skalarnog člana nije bitan jer on fizikalno samo označava smjer rotacije. Pozitivan ili negativan predznak daju istu rotaciju.

Jednom kada smo odredili skalarni član η, moguće je odrediti vektor $\boldsymbol{\epsilon}$. Može se pokazati da za $\eta \neq 0$ vrijedi:

$$\begin{aligned} \epsilon_x &= \tfrac{c_{23}-c_{32}}{4\eta}, \\ \epsilon_y &= \tfrac{c_{31}-c_{13}}{4\eta}, \\ \epsilon_z &= \tfrac{c_{12}-c_{21}}{4\eta}. \end{aligned} \tag{2.15}$$

Za slučaj kada je $\eta = 0$, vrijedi da je kut zakretanja jednak $\phi = \pm\pi$. Može se pokazati kako je moguće izračunati elemente $\boldsymbol{\epsilon}$ vektora kao:

$$\begin{aligned} |\epsilon_x| &= \left(\tfrac{c_{11}+1}{2}\right)^{\frac{1}{2}}, \\ |\epsilon_y| &= \left(\tfrac{c_{22}+1}{2}\right)^{\frac{1}{2}}, \\ |\epsilon_z| &= \left(\tfrac{c_{33}+1}{2}\right)^{\frac{1}{2}}. \end{aligned} \tag{2.16}$$

Za odabir predznaka svakog elementa $\boldsymbol{\epsilon}$ vektora čitatelja se savjetuje da pogleda u izvor [2].

2.3.3. Rotacija vektora pomoću rotacijske matrice

U nastavku ćemo navesti izraze za rotaciju vektora pomoću rotacijskih matrica. Prvi primjer odnosi se na rotaciju vektora oko z-osi, a drugi primjer odnosi se na 3-2-1 rotaciju.

Rotacija oko z-osi

Želimo li vektor $\overrightarrow{\boldsymbol{v}} = [x\ y\ z]^T$ rotirati oko z-osi za npr. 30 stupnjeva, prvo računamo rotacijsku matricu pomoću jednadžbe 2.3:

$$\boldsymbol{C}_z(30^\circ) = \begin{bmatrix} \cos 30^\circ & \sin 30^\circ & 0 \\ -\sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.17}$$

zatim pomnožimo vektor $\overrightarrow{\boldsymbol{v}}$ i matricu $\boldsymbol{C}_z$ gdje dobivamo rotirani vektor $\overrightarrow{\boldsymbol{v}}'$:

$$\begin{aligned} \overrightarrow{\boldsymbol{v}}' &= \boldsymbol{C}_z(30^\circ)\overrightarrow{\boldsymbol{v}} \\ &= \begin{bmatrix} \cos 30^\circ & \sin 30^\circ & 0 \\ -\sin 30^\circ & \cos 30^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}. \end{aligned} \tag{2.18}$$

3-2-1 rotacija

Želimo li vektor $\overrightarrow{\boldsymbol{v}}$ definiran u prošlom primjeru rotirati prvo oko z-osi za 30 stupnjeva, zatim oko tranzicijske y-osi za 20 stupnjeva i na kraju oko transformirane x-osi za 10 stupnjeva, prvo računamo rotacijsku matricu pomoću jednadžbe 2.6 kao:

$$\boldsymbol{C}_{21}(10^\circ, 20^\circ, 30^\circ) = \boldsymbol{C}_x(10^\circ)\boldsymbol{C}_y(20^\circ)\boldsymbol{C}_z(30^\circ), \tag{2.19}$$

zatim pomnožimo vektor $\overrightarrow{\boldsymbol{v}}$ i matricu $\boldsymbol{C}_{21}$ gdje dobivamo rotirani vektor $\overrightarrow{\boldsymbol{v}}'$:

$$\overrightarrow{\boldsymbol{v}}' = \boldsymbol{C}_{21}(10^\circ, 20^\circ, 30^\circ)\overrightarrow{\boldsymbol{v}}. \tag{2.20}$$

2.4. Jednadžba rotacijskog gibanja

Želimo li definirati jednadžbe rotacije satelita, važno se je prvo prisjetiti definicija referentnog sustava tijela $\mathcal{F}_b$ i inercijskog referentnog sustava $\mathcal{F}_G$. Orijentaciju između ta

dva referentna sustava moguće je opisati orijentacijskom matricom $\boldsymbol{C}_{bG}$ i to na barem dva načina: pomoću Eulerovih kutova ili pomoću kvaterniona (vidi prethodna poglavlja).

Kako je $\mathcal{F}_b$ referentni sustav nepomičan s obzirom na tijelo satelita, kutnu brzinu referentnog sustava tijela $\mathcal{F}_b$ s obzirom na $\mathcal{F}_G$ inercijskog referentnog sustava označavamo kao:

$$\boldsymbol{\omega}_{bG} = \left[\omega_1 \ \omega_2 \ \omega_2\right]^T. \tag{2.21}$$

2.4.1. Prikaz pomoću Eulerovih kutova

Jednadžbu rotacijskog gibanja pomoću Eulerovih kutova definiramo kao:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \boldsymbol{\omega}_{bG}. \tag{2.22}$$

2.4.2. Prikaz pomoću kvaterniona

Jednadžbu rotacijskog gibanja pomoću kvaterniona i kutne brzine definiramo kao:

$$\begin{aligned} \dot{\boldsymbol{\epsilon}} &= \tfrac{1}{2}(\eta\mathbf{1} + \boldsymbol{\epsilon}^{\times})\boldsymbol{\omega}_{bG}, \\ \dot{\eta} &= -\tfrac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{\omega}_{bG}. \end{aligned} \tag{2.23}$$

Radi lakšeg računanja, raspišemo li jednadžbu 2.23 pomoću definicije kvaterniona u jednadžbi 2.2 dobiti ćemo relaciju za rotacijsko gibanje [11] kao:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{S}(\boldsymbol{\omega}_{bG})\boldsymbol{q} \tag{2.24}$$

gdje je matrica $\boldsymbol{S}(\boldsymbol{\omega}_{bG})$ definirana kao:

$$\boldsymbol{S}(\boldsymbol{\omega}_{bG}) = \begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix}. \tag{2.25}$$

2.5. Usporedba Eulerovih kutova i kvaterniona

Kao što smo već pisali u poglavlju 2.3.1, dva su problema prikazivanja orijentacije pomoću Eulerovih kutova. Prvi problem je singularitet gdje za neke kutove koje reprezentiraju istu orijentaciju nije moguće jednoznačno odrediti Eulerove kutove iz rotacijske matrice. Drugi problem je što za opisivanje orijentacije koristimo trigonometrijske funkcije koje je kompleksnije izračunati na računalima.

Prikaz orijentacije kvaternionom rješava nas problema singulariteta zbog dodatnog četvrtog člana. Kvaternionom smo također riješili problem računanja trigonometrijskih funkcija jer jednadžbe više ne sadrže trigonometrijske funkcije nego obične algebarske izraze.

S druge strane, pozitivna strana prikaza orijentacije pomoću Eulerovih kutova je njihova intuitivnost jer svaki Eulerov kut označava kut rotacije oko pojedine osi. Informacija o orijentaciji kod kvaterniona skrivena je unutar kombinacije skalarnog i vektorskog člana i na prvi pogled izgleda nelogično. Skalarni član ne označava izravno kut zakreta oko osi rotacije već njegov kosinus kuta, a vektorski dio sadrži skaliranu os rotacije sinusom kuta rotacije (vidi jednadžbu 2.1).

2.6. Pojednostavljeni model satelita

Radi jednostavnosti prilikom razvijanja programske podrške fokusirali smo se na određivanje i kontrolu orijentacije samo oko jedne osi.

Eksperimentalno se može pokazati (vidi poglavlje 5.2.1) kako Laplaceov model kutne brzine satelita oko jedne osi pogonjen zamašnjakom možemo modelirati kao sustav prvog reda:

$$H(s) = \frac{Y(s)}{X(s)} = K\frac{1/\tau}{s + 1/\tau}, \tag{2.26}$$

gdje je $Y(s)$ kutna brzina oko jedne osi, $X(s)$ je funkcija pobude zamašnjaka, a K i τ su koeficijenti prijenosne funkcije. Parametar K mogli bismo okarakterizirati kao koeficijent pretvorbe mjernih jedinica ulaza $X(s)$ i izlaza $Y(s)$, a parametar τ kao koeficijent tromosti sustava: što je sustav tromiji to je parametar τ veći, i obratno. Mjerna jedinica ulazne funkcije $X(s)$ je postotak *Duty Cycle*-a s kojim pogonimo elektromotor zamašnjaka (vidi poglavlje 3.3.2). Mjerna jedinica izlaza $Y(s)$ je radijan po sekundi (rad/s). Mjerna jedinica parametra K je radijan po sekundi po postotku DC zamašnjaka, a jedinica za tromost sustava τ je sekunda.

Drugi pojednostavljeni model sustava opisuje nam orijentaciju satelita oko jedne osi (kut). Dobili smo ga integriranjem jednadžbe 2.26 koja opisuje kutnu brzinu satelita jer vrijedi da je integral kutne brzine jednak kutu. Model orijentacije satelita u Laplaceovoj domeni definiramo kao:

$$E(s) = H(s)\ \frac{1}{s} = K\frac{1/\tau}{s + 1/\tau}\ \frac{1}{s}. \tag{2.27}$$

Primijetimo kako je model orijentacije satelita oko jedne osi sustav drugog reda.

3. Sustav za određivanje i upravljanje orijentacijom satelita (ADCS)

U ovom poglavlju navest ćemo definiciju ADCS (engl. *Attitude Determination and Control system*) sustava. Navest ćemo osnovne dijelove sustava. Navest ćemo najčešće korištene senzore i aktuatore, objasniti njihove karakteristike i navesti prednosti i mane. Također ćemo opisati metode i algoritme za određivanje i kontrolu orijentacije. Na koncu ćemo opisati razvijeno sklopovlje i navesti odabrane senzore i aktuatore.

3.1. Uvod

ADCS sustav je sustav koji određuje orijentaciju satelita pomoću podataka iz senzora na temelju kojih sustav aktuatorima vrši korekciju orijentacije u željeni položaj i tu orijentaciju održava.

3.1.1. Određivanje i upravljanje orijentacijom

Senzori koje koristi ADCS sustav za određivanje orijentacije najčešće ne daju direktno podatak o orijentaciji već se orijentacija određuje (estimira) pomoću tzv. estimacijskih algoritama. Problem senzora općenito je njihova neodređenost koja onda doprinosi grešci estimaciji orijentacije. Zbog neodređenosti senzora razvijeni su estimacijski algoritmi koje uzimaju u obzir neodređenost senzora i tako smanjuju neodređenost estimirane orijentacije.

Jednom kada smo odredili orijentaciju i ako ona nije jednaka željenoj, slijedi postupak korekcije (kontrole). Postupak korekcije započinje računanjem razlike željene i trenutno estimirane orijentacije. Regulator orijentacije pomoću te razlike izračunava optimalnu korekciju koju je potrebno izvršiti kako bi došli u željenu orijentaciju. Korekciju orijentacije izvršavaju mehanički ili električni aktuatori. Aktuatori stvaraju korekcijski moment koji satelit na koncu dovode u željenu orijentaciju. Postignuvši

željenu orijentaciju, ADCS sustav će je nastojat održavati.

Opisana estimacija i kontrola orijentacije ADCS sustava događa se u stvarnom vremenu kroz cijeli misiju satelita. Ni u jednom trenu se ne smije dogoditi da satelit posjeduje neželjenu ili nekontroliranu orijentaciju jer to može značiti kraj misije.

3.1.2. Važnost ADCS sustava

NASA-ino istraživanje pokazalo je da je ADCS sustav zaslužan za 23% pogrešaka na navigacijsko-kontrolnim sustavima (engl. *Guidance, Navigation and Control - GN&C*), i da je velika većina tih anomalija u konačnici uzrokovala kraj misije [8]. U nastavku ćemo navesti jedan primjer greške na ADCS sustavu.

Naime, svemirska letjelica *Lewis* [46] lansirana je 1997. godine s predviđenim trajanjem misije od 5 godina. Nakon postizanja uspješne orbite, satelit je započeo s normalnim radom i nije ga više bilo potrebno aktivno kontrolirati. U normalnom radu solarni paneli su bili upereni prema Suncu kako bi prikupili maksimalnu količinu električne energije. Zbog greške prilikom rada na jednom od aktuatora, stvarao se je neprimjetno mali konstantni moment oko jedne osi koju senzori nisu mogli detektirati. Nakon nekoliko dana neprestane rotacije satelit se je počeo nekontrolirano rotirati i solarni paneli s vremenom nisu prikupili dovoljno energije. Jednom kada su inženjeri napokon uočili nekontroliranu rotaciju već je bilo prekasno jer se je baterija satelita potpuno ispraznila i komunikacija sa satelitom zauvijek prekinuta [19].

3.2. Osnovni dijelovi

Generalno, osnovni dijelovi ADCS sustava su: računalo, senzori i aktuatori. U nastavku ćemo zasebno opisati svaki dio i navesti neke karakteristike.

3.2.1. Računalo

Računalo je zaslužno za prikupljanje i obradu podataka koje dolaze sa senzora i za kontrolu aktuatora. Prikupljanje podataka sa senzora vrši se preko komunikacijskih kanala (sabirnica). Obrada podataka zasniva se na računanju zahtjevnih estimacijskih algoritama koji na koncu daju podatak o trenutnoj orijentaciji. Na temelju razlike trenutne i željene orijentacije, računalo računa/modulira kontrolni signal koji se dalje šalje aktuatorima preko predviđenih sabirnica.

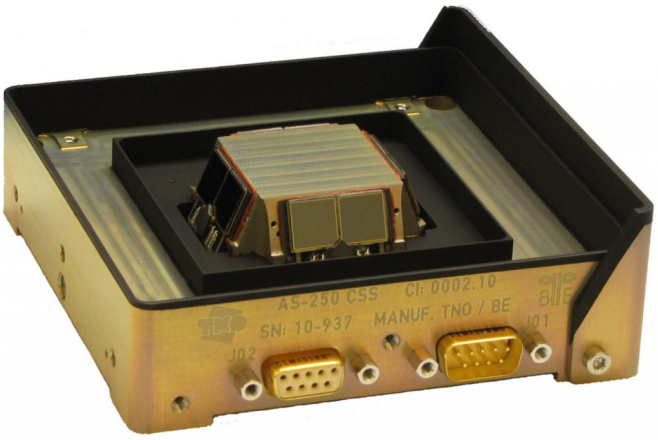
Neki od zahtjeva ADCS računala su: mala potrošnja, otpornost na radijaciju/kozmičke zrake, brzina izvođenja operacija, pouzdanost i dr. Pri odabiru računala važno je ako je računalo već korišteno u nekim prijašnjim misijama.

3.2.2. Senzori orijentacije

Senzori orijentacije ili orijentacijski senzori elektroničke su naprave iz kojih je moguće dobiti podatak o orijentaciji. Podatak o orijentaciji najčešće ne dolazi direktno sa senzora već se orijentacija estimira matematičkim algoritmima koji kao svoj rezultat daju estimiranu orijentaciju [2]. Za precizno određivanje orijentacije nekada je potrebno uzeti podatke iz više senzora odjednom. Takav način prikupljanja podataka zovemo engl. *sensor fusion*. U nastavku ćemo navesti najkorištenije senzore, njihove karakteristike i način rada.

Senzor Sunca

Senzore Sunca dijelimo u dvije skupine: analogne i digitalne. Analogni senzori Sunca rade na principu solarnih ćelija, a podatak o smjeru tzv. vektora sunca moguće je dobiti iz generirane struje ćelija. Slika 3.1 prikazuje industrijsku izvedbu jednog takvog senzora.

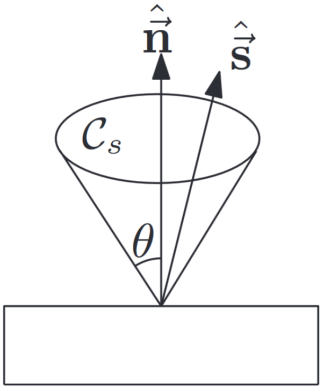


Slika 3.1: Prikaz industrijske izvedbe senzora Sunca tvrtke Bradford [7].

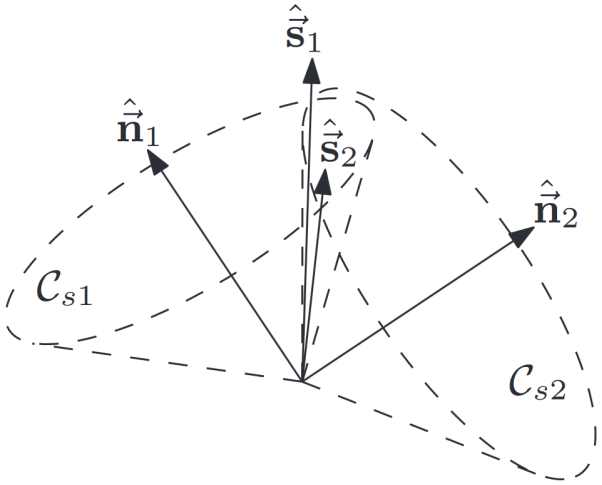
Struja iz solarnih ćelija ovisi o kutu θ što označava kut između normale senzora $\vec{\boldsymbol{n}}$ i vektora nadolazeće zrake Sunca $\vec{\boldsymbol{v}}$ (vidi sliku 3.2). Vezu struje ćelija i i kuta θ definiramo kao:

$$i(\theta) = i(0)\cos\theta. \tag{3.1}$$

Senzor Sunca ima konačni kut gledanja u obliku stošca. Sa samo jednim senzorom Sunca nije moguće jednoznačno odrediti vektor Sunca $\hat{\vec{v}}$ već se primjenjuje paradigma *sensor fusion*-a gdje dodamo još jedan senzor Sunca (vidi sliku 3.3) [2].



Slika 3.2: Prikaz modela senzora Sunca.



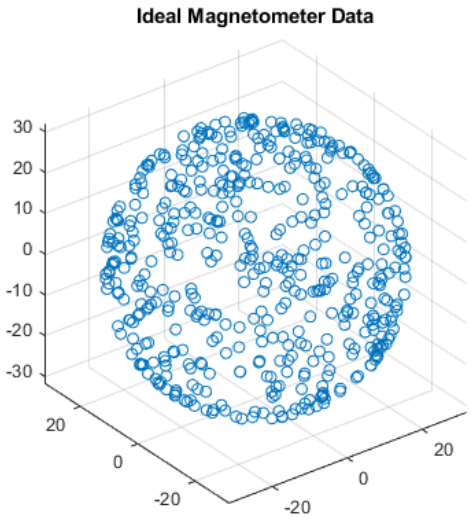
Slika 3.3: Prikaz para senzora Sunca.

Troosni magnetometar

Troosni magnetometar mjeri vektor lokalnog magnetskog polja Zemlje u koordinatnom sustavu senzora.

Magnetometri su relativno neprecizni senzori i za pouzdanija mjerenja potrebno ih je kalibrirati [2]. Kalibracija ima za cilj smanjiti utjecaj tzv. engl. *Hard* i *Soft* efekata. Prije nego objasnimo spomenute efekte objasniti ćemo postupak prikupljanja podataka u svrhu kalibracije magnetometra.

Kalibracija magnetometra započinje prikupljanjem podataka iz senzora i to tako što ćemo magnetometar rotirati u svim smjerovima pazeći da ne zanemarujemo određen položaj. Nakon što smo prikupili dovoljno podataka, možemo prikazati izmjerene vektore magnetskog polja kao skupinu točaka gdje svaka točka prikazuje vrh vektora očitane vrijednosti magnetskog polja. Prikaz očitanih vrijednosti idealnog magnetometra raspoređeni su po idealnoj sferi radijusa duljine vektora magnetskog polja sa središtem u centru koordinatnog sustava (vidi sliku 3.4).

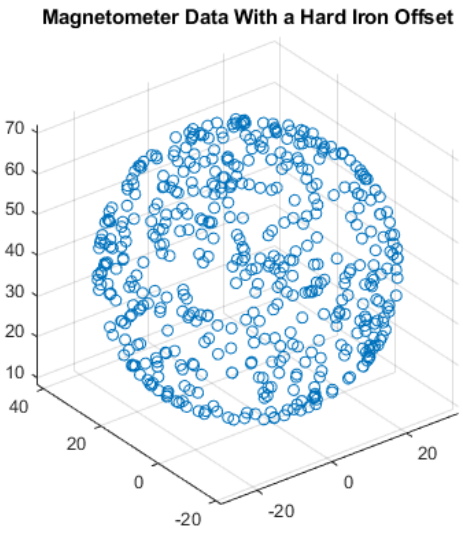


Slika 3.4: Točkasti prikaz mjerenja vektora Zemljinog magnetskog polja pomoću idealnog magnetometra. Vrijednosti mjerenja su u jedinici μT.

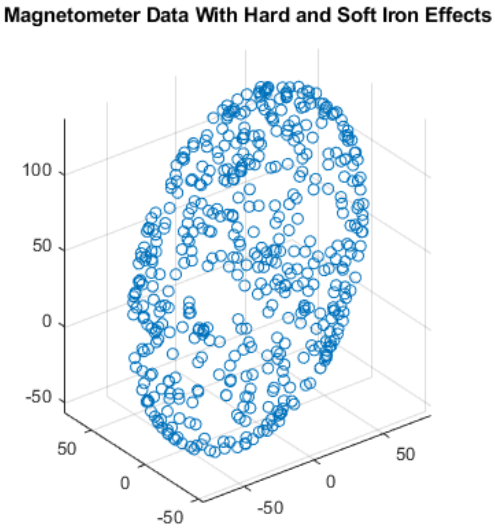
Nakon što smo opisali postupak prikupljanja podataka, slijedi objašnjenje *Hard* i *Soft* efekata. *Hard* efekt je greška magnetometra koja utječe na mjerenja magnetskog polja i koja se manifestira kao translacija ishodišta rezultantne sfere mjerenja za neki fiksni vektor (vidi sliku 3.5). Možemo slobodno reći kako su mjerenja pristrana odnosno da imaju tzv. engl. *bias*. Uzrok *Hard* efekta su najčešće stacionarne interferencije magnetskih polja kojeg želimo mjeriti i magnetskih polja okolnih metalnih dijelova. Magnetska interferencija dolazi ili od samog magnetometra ili od okolnog elektroničkog sklopovlja.

Soft efekti nastaju zbog predmeta u neposrednoj blizini magnetometra koji uzrokuju distorziju mjerenog lokalnog magnetskog polja. *Soft* efekt, kolokvijalno rečeno, razvlači i kosi sferu mjerenja koja zatim oblikom postaje nalik nakošenom elipsoidu (vidi sliku 3.6).

Modeliranje spomenutih efekata opisujemo u sljedećoj jednadžbi:



Slika 3.5: Točkasti prikaz mjerenja vektora Zemljinog magnetskog polja pomoću magnetometra s tzv. *Hard* efektom. Vrijednosti mjerenja su u jedinici μT.



Slika 3.6: Točkasti prikaz mjerenja vektora Zemljinog magnetskog polja pomoću magnetometra s tzv. *Hard* i *Soft* efektima. Vrijednosti mjerenja su u jedinici μT.

$$(\boldsymbol{x} - \boldsymbol{b})\boldsymbol{R}(\boldsymbol{x} - \boldsymbol{b})^T = \beta^2, \tag{3.2}$$

gdje je $\boldsymbol{R}$ matrica tipa 3×3 određuje oblik elipsoida (npr. za sferu jedinična matrica, a za elipsoid pozitivno definitna), vektor $\boldsymbol{b}$ tipa 1×3 definira središte elipsoida, a skalar β predstavlja duljinu vektora magnetskog polja.

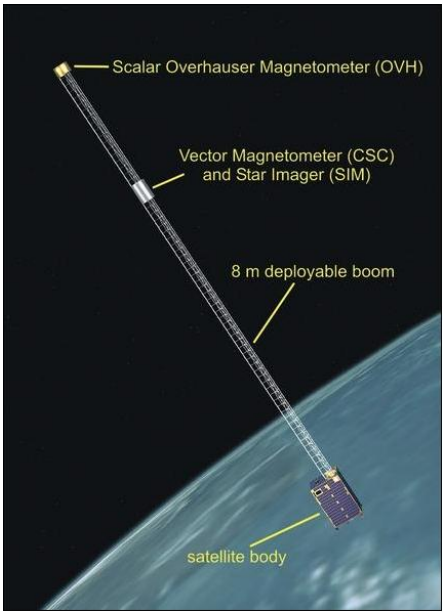
Kalibraciju magnetometra vršimo pomoću sljedeće relacije:

$$\boldsymbol{m} = (\boldsymbol{x} - \boldsymbol{b})\boldsymbol{A}, \tag{3.3}$$

gdje novo-definirana matrica $\boldsymbol{A}$ tipa 3×3 transformira oblik elipsoida u pravilnu sferu (eliminira utjecaj *Soft* efekta), a definirani vektor $\boldsymbol{b}$ translatira elipsoid u ishodište koordinatnog sustava (eliminira utjecaj *Hard* efekta).

Problem kalibracije magnetometra uključuje problem pronalaska $\boldsymbol{A}$ i $\boldsymbol{b}$ parametara. Parametre je moguće pronaći pomoću `magcal` funkcije unutar MATLAB alata [41]. Kalibraciju magnetometra vršimo u programskom kodu nakon svakog primitka nekalibrirane vrijednosti magnetometra. Programski isječak kalibracije moguće je vidjeti ovdje [28]. Dodatna objašnjenja u vezi kalibracije magnetometra mogu se pronaći ovdje [40].

Zbog spomenutih vanjskih utjecaja na vrijednosti mjerenja magnetometra, magnetometar se može fizički udaljiti od izvora smetnji pomoću konstrukcije nazvane engl. *boom* (vidi sliku 3.7) te tako smanjiti pogreške mjerenja [2].



Slika 3.7: Prikaz tzv. *boom*-a na čijem vrhu se nalazi magnetometar. Izvor Ørsted misija [13].

Pratioci zvijezda (engl. *Star Trackers*)

Pratioci zvijezda u usporedbi sa ostalim senzorima pružaju najpreciznija mjerenja orijentacije iz dva razloga: prvi je što su svjetlosni izvori zvijezda točkaste prirode i tako pružaju precizna mjerenja, i drugi, što se zvijezde zbog svoje udaljenosti od senzora doimaju kao stacionarni izvori neovisno o položaju satelita unutar orbite.

Slika 3.8: Izvedba pratioca zvijezda tvrtke Redwire [50].

Pomoću pratioca zvijezda moguće je pratiti vektor smjera samo jedne zvijezde i tako, u ovisnosti o položaju senzora na satelitu, orijentacijskom matricom dobiti podatak smjera zvijezde u referentnom sustavu satelita. Uporabom podataka iz ostalih senzora (*sensor fusion*) moguće je jednoznačno odrediti orijentaciju satelita.

U praksi pratioci zvijezda prate više zvijezda odjednom ne bi li smanjili neodređenost orijentacije. Jednom kada je senzor odredio orijentaciju, svako sljedeće mjerenje zasniva se na praćenju istih zvijezda i metodom usrednjavanja moguće je postignuti veću preciznost. Neodređenost je dodatno moguće smanjiti upotrebom tzv. zvjezdanih kataloga [31] koji sadrže podatke o položaju poznatih zvijezda i zviježđa.

Zbog svega navedenoga, pratioci zvijezda su najprecizniji senzori orijentacije od svih do sad navedenih senzora, ali s druge strane i najskuplji, najkompleksniji i najnepouzdaniji senzori [2].

3.2.3. Senzori kutne brzine

Senzori kutne brzine mjere kutnu brzinu referentnog sustava tijela naspram referentnog inercijskog sustava. Kao što ćemo vidjeti u nastavku rada, većina algoritama za procjenu orijentacije koriste podatak o kutnoj brzini satelita kao dodatnu korisnu informaciju.

Kutnu brzinu moguće je izračunati kao vremensku derivaciju orijentacije ali takav nam izračun unosi neodređenost u obliku visokofrekventnog (VF) šuma. Senzori kutne brzine nemaju VF izražen šum ali zato imaju problem sa tzv. *bias*-om - niskofrekventnim (NF) šumom.

Prednost senzora kutne brzine je što ne zahtijevaju eksterni izvor informacije. Primjerice, senzor Sunca uvijek mora imati prisutno Sunce ne bi li izračunali orijentaciju satelita ili magnetometar koji mora biti u prisutnosti izvora magnetskog polja.

U trenucima kada ne možemo direktno dobiti podatak o orijentaciji satelita, inte-

gracijom kutne brzine moguće je odrediti orijentaciju. Problem takvog načina određivanja orijentacije je spomenuti *bias* koji uzrokuje integracijsku grešku. Zbog toga se takav način određivanja orijentacije ograničava na kratke vremenske periode u kojima je greška integracije zanemariva.

Postoji više vrsta senzora kutne brzine, a tradicionalno su najpopularniji tzv. mehanički žiroskopi. Mane mehaničkih žiroskopa su mehanički pokretni dijelovi koji stvaraju vibracije, s vremenom im raste nepouzdanost i za rad koriste relativno puno električne energije. Zbog toga su nedavno u primjenu ušli tzv. laserski žiroskopi koji ne posjeduju pokretne dijelove [2].

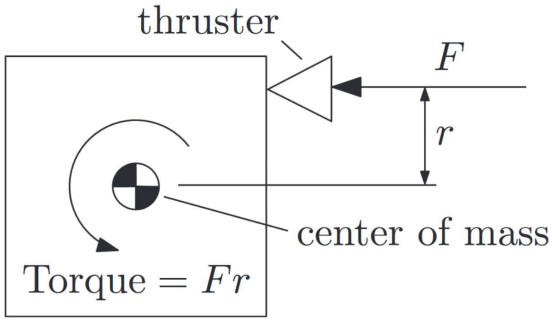
Osim mehaničkih i laserskih žiroskopa, postoje i tzv. *MEMS* žiroskopi koji pomoću tzv. Coriolisovog efekta mjere kutnu brzinu. Takvi senzori najčešće se ugrađuju u integrirane sklopove i u svom radu koriste nanostrukture koje omogućavaju mjerenje Coriolisovog efekta [32].

3.2.4. Aktuatori

Aktuatori su uređaji koji na temelju kontrolnog signala vrše korekciju orijentacije. Dijelimo ih u dvije skupine: oni koji mijenjaju kutnu količinu gibanja satelita i oni koji je ne mijenjaju. Potisnici i magnetorkeri spadaju u prvu skupinu, a zamašnjaci u drugu skupinu [2]. U nastavku slijedi opis svakog spomenutog aktuatora.

Potisnici (engl. *Thrusters*)

Potisnici su aktuatori koji rade na principu izbacivanja fluidne mase te tako stvaraju potisnu silu. Ako potisni vektor ne prolazi kroz centar mase satelita, u trenutku izbacivanja mase doći će do stvaranja momenta. Kao što se može vidjeti na slici 3.9, potisnik koji generira silu F na udaljenosti r od centra mase, uzrokuje moment T definiran kao $T = Fr$.



Slika 3.9: Princip rada potisnika.

Potisnik nije u stanju generirati silu F u oba smjera (naravni i suprotni smjer). Zbog toga je za stvaranje pozitivnog i negativnog momenta potrebno imati dva međusobno suprotno orijentirana potisnika. Sukladno tomu, za troosno upravljanje orijentacijom satelita potrebno je imati najmanje 6 potisnika [2].

Magnetorkeri (engl. *magnetorquers, magnetic torquers*)

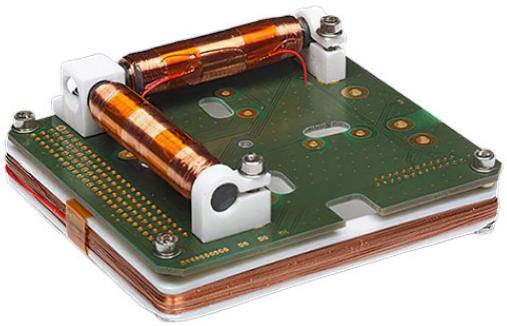
Magnetorkeri su aktuatori koji generiraju moment pomoću tzv. magnetske sprege svojstveno-generiranog magnetskog polja i magnetskog polja Zemlje. Magnetorker je u suštini zavojnica namotana oko zračne ili feromagnetske jezgre.

Magnetorker počiva na Amperovom zakonu koji kaže da prolaskom struje kroz namotaje zavojnice dolazi do stvaranja magnetskog polja (magnetski dipol). Međusobnom interakcijom generiranog dipola magnetorkera $\overrightarrow{m}$ i magnetskog dipola Zemlje $\overrightarrow{b}$ stvara se moment $\overrightarrow{T}$ jednak:

$$\overrightarrow{T} = \overrightarrow{m} \times \overrightarrow{b}. \tag{3.4}$$

Moguće je pokazati kako je vektorski produkt (također vektor) u jednadžbi 3.4 uvijek okomit na magnetski dipol Zemlje $\overrightarrow{b}$ i zbog toga je nemoguće generirati moment onda kada je položaj dipola magnetorkera usmjeren jednako kao i dipol magnetskog polja Zemlje.

Jedan magnetorker omogućuje upravljanje orijentacijom satelita samo u jednoj osi što znači da za troosno upravljanje orijentacijom tipično imamo 3 međusobno ortogonalna magnetorkera (vidi sliku 3.10).



Slika 3.10: Magnetorker tvrtke NanoAvionics [44].

Zbog relativno slabe jakosti Zemljinog magnetskog polja, teško je generirati momente dovoljne jakosti, pogotovo za satelite velike momente tromosti. Zbog toga se

kontrola orijentacije najčešće vrši pomoću zamašnjaka u sprezi sa magnetorkerima iz dva razloga: prvi razlog je već spomenuta nedovoljna jakost generiranog momenta od strane magnetorkera, a drugi razlog je problem tzv. zasićenja zamašnjaka (vidi kasnije poglavlje).

Valja skrenuti pažnju i na problem utjecaja magnetorkera na mjerenja senzora magnetskog polja. Ako prilikom vršenja korekcije orijentacije pomoću magnetorkera mjerimo magnetsko polje, doći će do magnetske interferencije magnetorkera i senzora magnetskog polja. Rješenje problema leži u tome da za vrijeme mjerenja magnetskog polja magnetorker ne vrši korekciju orijentacije [2].

U sklopu projekta razvijena je metoda optimalne parametrizacije magnetorkera koja će za dani magnetski moment parametrizirati magnetorker tako da posjeduje minimalnu masu uz minimalni potrošak snage [22].

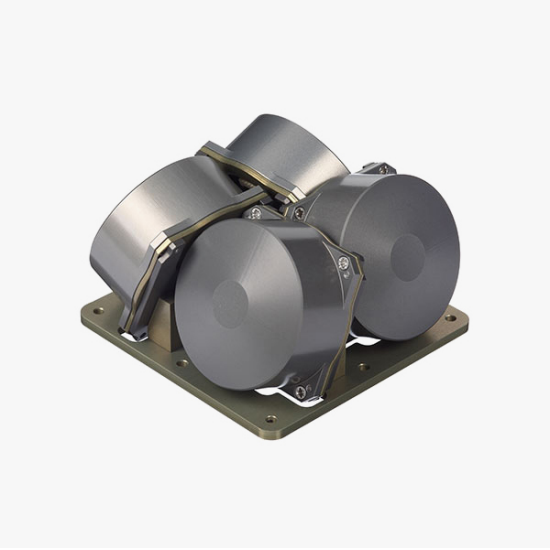
Zamašnjaci (engl. *Momentum wheels*)

Zamašnjaci su mehanički uređaji koji rade na principu očuvanja kutne količine gibanja. Zamašnjak se sastoji od rotirajuće mase i pogona (elektromotora). Naime, kada elektromotorom vršimo moment u jednom smjeru, zbog zakona očuvanja kutne količine gibanja, zamašnjak će generirati jednak moment po iznosu ali u suprotnom smjeru ne bi li ukupna kutna količina gibanja satelita ostala ista (vidi sliku 3.11). Na taj način moguće je rotirati satelit u jednu ili u drugu stranu. U nominalnom stanju zamašnjak ne posjeduje kutnu brzinu. Zamašnjaci kao aktuatori predstavljaju najprecizniju kontrolu orijentacije [2].

Slika 3.11: Zamašnjak za Cubesat tvrtke Rocket Lab [51].

Problem zamašnjaka manifestira se s vremenom. Da bi objasnili problem zamašnjaka, pretpostavimo da satelitu želimo održavati jednaku orijentaciju bez obzira na vanjske parazitne momente/smetnje (koji god oni bili). U trenutku vanjske smetnje, želimo li satelitu održati jednaku orijentaciju, zamašnjakom upravljamo na način da ćemo generirati jednaki moment po iznosu ali u suprotnom smjeru. Zamašnjak će se

zarotirati u istu stranu kao i vanjski moment smetnje. Jednom kada smo pomoću zamašnjaka uveli jednaku kutnu količinu gibanja kao što ju je uvela vanjska smetnja, zamašnjaku moramo konstantno održavati kutnu brzinu. Bilo kakva daljnja promjena kutne brzine zamašnjaka uzrokuje promjenu kutne količine gibanja i samim time promjenu orijentacije satelita. Ako se s vremenom smetnje akumuliraju doći će do pojave tzv. saturacije zamašnjaka gdje će se zamašnjak nastojati okretati sve većom i većom brzinom sve dok ne dosegne maksimalnu moguću brzinu - saturacija. Drugi suptilni problem je taj što rotirajućem zamašnjaku konstantno moramo održavati kutnu brzinu što unosi negativnu energetsku računicu. Desaturacija je postupak s kojim se želi smanjiti kutna brzina zamašnjaka. Postupak desaturacije uključuje istovremeno usporavanje kutne brzine zamašnjaka i kompenzaciju novonastalog momenta pomoću nekih drugih aktuatora (npr. magnetorkera). Treći problem zamašnjaka leži u činjenici da su zamašnjaci mehanički uređaji s rotirajućim dijelovima te se očekuje da će s vremenom oni prestati raditi. Iz tog razloga umjesto 3 zamašnjaka za mogućnost troosnog upravljanja uzimamo 4. karakteristično položena zamašnjaka (vidi sliku 3.12) gdje je jedan zamašnjak redundantan. Pomoću takvog položaja moguće je imati troosno upravljanje orijentacijom s gdje jedan od zamašnjaka ne radi [33].



Slika 3.12: Skup od 4 zamašnjaka tvrtke NanoAvionics gdje je jedan zamašnjak redundantan [45].

3.3. Algoritmi za određivanje i upravljanje orijentacijom

U ovom dijelu navest ćemo algoritme i metode za određivanje i upravljanje orijentacijom satelita.

3.3.1. Određivanje orijentacije

Određivanje orijentacije je postupak u kojem iz odabranih orijentacijskih senzora (vidi poglavlje 3.2.2) prikupljamo podatke koje zatim obrađujemo unutar posebno razvijenih algoritama čiji rezultat računanja daje podatak o orijentaciji satelita.

Orijentacija satelita nije jednoznačno određena već u njoj postoje neodređenosti (greške). Takve neodređenosti najviše proizlaze iz neodređenosti orijentacijskih senzora. Druge, manje utjecajnije neodređenosti proizlaze iz konačne preciznosti aritmetičkih operacija računala na kojima se računaju takvi algoritmi. Zbog toga vrlo često govorimo o estimaciji orijentacije jer je podatak o orijentaciji više ili manje gruba procjena. Kako bi smanjili neodređenosti procjene orijentacije, najčešće uzimamo podatke iz više neovisnih senzora zajedno - *Sensor fusion*. *Sensor fusion* će smanjiti neodređenosti pojedinih senzora i u konačnici ćemo dobiti bolji rezultat procjene nego što bismo dobili korištenjem samo jednog senzora. Algoritmi za estimaciju orijentacije vrlo su često razvijeni tako da optimalno procjenjuju orijentaciju s obzirom na unaprijed definiran uvjet.

Najpoznatiji problem estimacije orijentacije satelita nazivamo engl. *Wahba's Problem* iz 1965. godine [61]. Nekoliko algoritama je razvijeno za potrebe rješavanja *Wahba's* problema i neke od njih navest ćemo u nastavku poglavlja.

U svrhu estimacije orijentacije važno je shvatiti nekoliko stvari. Algoritmi koje ćemo spomenuti, za svoj rad koriste podatke iz 2 različita troosna senzora položena u referentni sustav tijela (satelita). Za svaki od ta dva senzora moraju unaprijed biti poznate njihove tzv. referentne vrijednosti - vrijednosti vektora unutar inercijskog referentnog sustava. Iz razlika izmjerenih vrijednosti i njihovih referentnih vrijednosti moguće je dobiti podatak o orijentaciji. Za lakše shvaćanje pojma referentnog vektora navest ćemo primjer akcelerometra - uređaja koji, u ovom slučaju, mjeri akceleraciju Zemljine sile teže. Kao dogovor uzet ćemo primjer gdje je referentni vektor akcelerometra položen u referentni sustav kojemu Z komponenta gleda prema središtu Zemlje (X i Y komponenta za ovaj primjer nisu važne). U tom slučaju vrijedi da je vrijednost referentnog vektora jednaka $[0\ 0\ -9.81]\, m/s^2$. Dalje, uzmimo da je

u jednom trenutku orijentacija senzora takva da gleda u suprotnom smjeru od središta Zemlje. U tom ćemo slučaju s akcelerometra pročitati vrijednost mjerenja iznosa $[0\ 0\ +9.81]\, m/s^2$. Iz usporedbe vrijednosti referentnog vektora i vektora mjerenja možemo lako zaključiti kako je orijentacija akcelerometra suprotna od orijentacije referentnog vektora i na taj način pretpostaviti orijentaciju.

Algoritme za estimaciju orijentacije dijelimo u dvije skupine: bezmemorijske i memorijske [11]. Bezmemorijski algoritmi ne uzimaju u obzir prošlo stanje orijentacije, već podatak o orijentaciji dobivaju isključivo iz trenutno izmjerenih vrijednosti. S druge strane, memorijski algoritmi će na temelju prošlog stanja orijentacije i trenutno izmjerenih vrijednosti estimirati orijentaciju. Takav način estimacije je superiorniji jer je stanje u neposrednoj prošlosti ili isto, ili vrlo slično trenutnom stanju.

Jedan popularni bezmemorijski algoritam naziva se QUEST algoritam. Neki od memorijskih algoritama naziva su: REQUEST, Optimal-REQUEST, Komplementarni filtar i dr. O njima nešto više u nastavku.

Wahba problem

Kako bismo naveli i objasnili estimacijske algoritme, prvo je potrebno da definiramo Wahba problem.

Ako su nam dani dva skupa vektora sa $n \geq 2$ elemenata $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \cdots, \boldsymbol{r}_n\}$ i $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n\}$, Wahba problem je problem pronalaženja orijentacijske matrice A koja transformira prvi set vektora u drugi na način koji najbolje minimizira srednju kvadratnu pogrešku definiranu kao:

$$\sum_{i=1}^{n} ||\boldsymbol{b}_i - A\boldsymbol{r}_i||^2, \tag{3.5}$$

gdje za matricu A vrijedi $A^T A = \boldsymbol{I}_3$ i $\det(A) = 1$.

REQUEST algoritam

REQUEST algoritam je jedan od algoritama koji rješava Wahba problem pronalaska orijentacijske matrice A. REQUEST algoritam je preteča QUEST algoritma koji je uzimao u obzir samo n trenutno izmjerenih vektora i na temelju njih računao orijentacijsku matricu [57]. REQUEST algoritam ide korak dalje i uvodi rekurziju.

Na temelju već dobivene orijentacijske matrice u prijašnjem k-tom koraku i novo izmjerenih $\boldsymbol{b}_i$ i $\boldsymbol{r}_i$ vektora senzora orijentacije, REQUEST algoritam će izračunati orijentacijsku matricu u $k+1$ koraku [6].

Osim izmjerenih $\boldsymbol{b}_i$ i $\boldsymbol{r}_i$ vektora, REQUEST algoritam uzima u obzir i trenutnu kutnu brzinu tijela pomoću koje točnije estimira orijentacijsku matricu.

Jedan problem REQUEST algoritma je što za estimaciju orijentacije zahtijeva par dodatnih fiksnih koeficijenata koji moraju biti eksperimentalno izabrani. Ti koeficijenti su zapravo težinski faktori koji govore kojim mjerenjima pridodajemo više važnosti: senzorima orijentacije ili mjerenju žiroskopa. Zbog tih fiksnih parametara REQUEST algoritam je suboptimalan [9]. Problem suboptimalnosti rješava Optimal-REQUEST algoritam kojeg ćemo navesti u nastavku.

Valja još spomenuti kako REQUEST algoritam nije općeniti algoritam koji će na izlazu dati estimaciju spomenute matrice A, već je to algoritam koji je specifično razvijen da na svom izlazu daje podatak o orijentaciji u obliku kvaterniona.

Optimal-REQUEST algoritam

Optimal-REQUEST algoritam je preteča REQUEST algoritma. Kao što smo već napomenuli ranije, REQUEST algoritam sadrži empirički izabrane koeficijente zbog kojih je takav algoritam suboptimalan. Optimal-REQUEST ide korak dalje: algoritam će s obzirom na propagaciju greške na senzorima orijentacije i žiroskopu sam mijenjati spomenute koeficijente na optimalan način. Metoda promijene koeficijenata zasniva se na tzv. Kalmanovom filtru [9].

Nećemo ulaziti u detalje implementacije već ćemo samo navesti kako smo u sklopu ovog projekta razvili Optimal-REQUEST algoritam u MATLAB kodu, te smo ga pretvorili u C kod pomoću MATLAB Coder alata [38]. Cijeli programski kod dostupan je ovdje [30] uz popratnu dokumentaciju ovdje [25].

Gradijentni spust (engl. *Gradient descent*)

Ako posjedujemo dva para vektora mjerenja $(\boldsymbol{v}_1^i,\ \boldsymbol{v}_1^b)$ i $(\boldsymbol{v}_2^i,\ \boldsymbol{v}_2^b)$ gdje $\boldsymbol{v}_i^b$ označava vektor definiran u referentnom sustavu tijela $\mathcal{F}_b$, a $\boldsymbol{v}_i^i$ označava vektor definiran u inercijskom referentnom sustavu $\mathcal{F}_i$, moguće je definirati tzv. rotacijsku matricu $\mathcal{R}_i^b$ (vidi jednadžbu 2.13) koja transformira vektor $\boldsymbol{v}^i$ u vektor $\boldsymbol{v}^b$.

Cilj estimacijskog algoritma je izračunati orijentacijsku matricu tako da smanjimo pogrešku estimacije. Pogrešku estimacije $\boldsymbol{e_i} = [e_{x,i}\ e_{y,i}\ e_{z,i}]^T$ za svaki par mjerenih vektora definiramo kao:

$$\begin{aligned} \boldsymbol{e}_1 &= \mathcal{R}_i^b \boldsymbol{v}_1^i - \boldsymbol{v}_1^b, \\ \boldsymbol{e}_2 &= \mathcal{R}_i^b \boldsymbol{v}_2^i - \boldsymbol{v}_2^b. \end{aligned} \tag{3.6}$$

Nadalje, definirat ćemo tzv. funkciju cilja. Funkcija cilja u gradijentnom spustu je parametar kojeg želimo maksimalno smanjiti. Funkciju cilja definiramo kao:

$$J(\boldsymbol{q}_i^b) = \boldsymbol{e}_1^T \boldsymbol{e}_1 + \boldsymbol{e}_2^T \boldsymbol{e}_2 \tag{3.7}$$

Konačno, algoritam gradijentnog spusta definiramo kao:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n - \alpha \nabla J(\boldsymbol{q}_n), \tag{3.8}$$

gdje je gradijent funkcije cilja $\nabla J(\boldsymbol{q}_n)$ definiramo kao:

$$\nabla J(\boldsymbol{q}_n) = \begin{bmatrix} 2\boldsymbol{e}_1^T \boldsymbol{M}_1 \boldsymbol{v}_1^i(\boldsymbol{q}_n) + 2\boldsymbol{e}_2^T \boldsymbol{M}_1 \boldsymbol{v}_2^i(\boldsymbol{q}_n) \\ 2\boldsymbol{e}_1^T \boldsymbol{M}_2 \boldsymbol{v}_1^i(\boldsymbol{q}_n) + 2\boldsymbol{e}_2^T \boldsymbol{M}_2 \boldsymbol{v}_2^i(\boldsymbol{q}_n) \\ 2\boldsymbol{e}_1^T \boldsymbol{M}_3 \boldsymbol{v}_1^i(\boldsymbol{q}_n) + 2\boldsymbol{e}_2^T \boldsymbol{M}_3 \boldsymbol{v}_2^i(\boldsymbol{q}_n) \\ 2\boldsymbol{e}_1^T \boldsymbol{M}_4 \boldsymbol{v}_1^i(\boldsymbol{q}_n) + 2\boldsymbol{e}_2^T \boldsymbol{M}_4 \boldsymbol{v}_2^i(\boldsymbol{q}_n) \end{bmatrix}, \tag{3.9}$$

i gdje su sve parcijalne derivacije rotacijske matrice $\mathcal{R}_i^b$ definirane kao:

$$\boldsymbol{M}_1 = \frac{\partial \mathcal{R}_i^b}{\partial q_1} = 2\begin{bmatrix} q_1 & q_4 & -q_3 \\ -q_4 & q_1 & q_2 \\ q_3 & -q_2 & q_1 \end{bmatrix}, \quad \boldsymbol{M}_2 = \frac{\partial \mathcal{R}_i^b}{\partial q_2} = 2\begin{bmatrix} q_2 & q_3 & q_4 \\ q_3 & -q_2 & q_1 \\ q_4 & -q_1 & -q_2 \end{bmatrix},$$

$$\boldsymbol{M}_3 = \frac{\partial \mathcal{R}_i^b}{\partial q_3} = 2\begin{bmatrix} -q_3 & q_2 & -q_1 \\ q_2 & q_3 & q_4 \\ q_1 & q_4 & -q_3 \end{bmatrix}, \quad \boldsymbol{M}_4 = \frac{\partial \mathcal{R}_i^b}{\partial q_4} = 2\begin{bmatrix} -q_4 & q_1 & q_2 \\ -q_1 & -q_4 & q_3 \\ q_2 & q_3 & q_4 \end{bmatrix}. \tag{3.10}$$

Procjena orijentacijskog kvaterniona započinje odabirom inicijalnog kvaterniona $\boldsymbol{q}_0$. Za inicijalni kvaternion najčešće se uzima proizvoljna vrijednost (ako drugačije nije moguće) npr. $\boldsymbol{q}_0 = [1,\ 0,\ 0,\ 0]^T$. Svaki sljedeći korak uključuje potraživanje mjerenih vektora iz orijentacijskih senzora i računanje procjene orijentacije jednadžbom 3.8.

Problem opisane procijene orijentacijskog kvaterniona metodom gradijentnog spusta je što takva procjena sadrži visokofrekventne komponente šuma. Takav šum moguće je filtrirati niskopropusnim filtrom, međutim, postoji bolji način filtriranja. Naime, kao što je bilo rečeno u poglavlju 3.2.3, orijentaciju je moguće izračunati iz mjerenog vektora kutne brzine (vidi jednadžbu 2.24) relacijom [11]:

$$\boldsymbol{q}_k = \boldsymbol{q}_{k-1} + \Delta T \dot{\boldsymbol{q}}_k = \boldsymbol{q}_{k-1} + \frac{\Delta T}{2} \boldsymbol{S}(\boldsymbol{\omega}_{bG}) \boldsymbol{q}_{k-1}, \tag{3.11}$$

gdje je ΔT proteklo vrijeme od prošlog računanja.

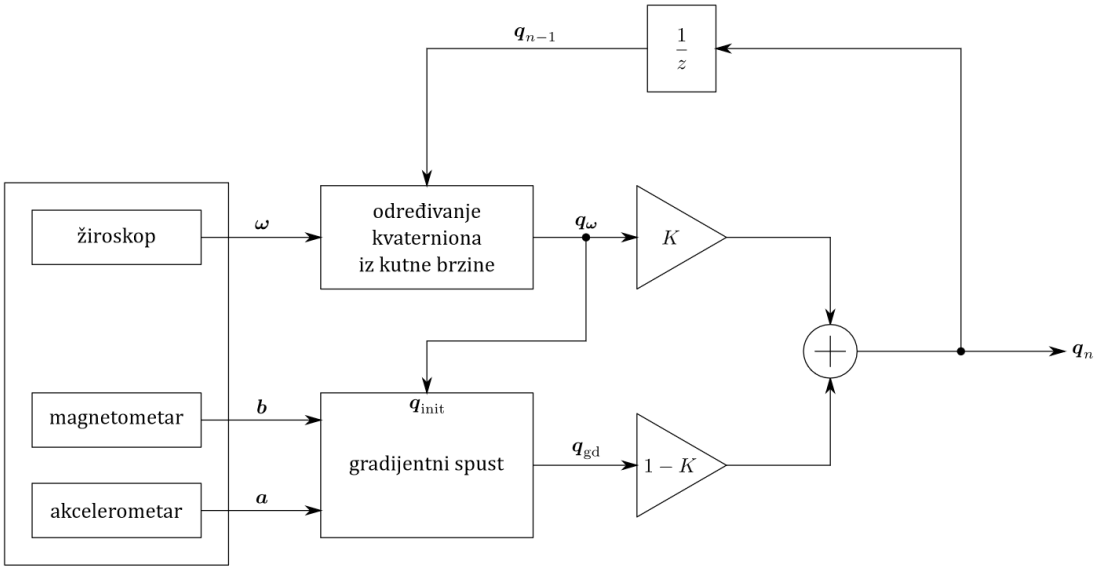
Problem takvog načina određivanja orijentacije je akumulirana greška. Takvu akumuliranu grešku moguće je filtrirati visokopropusnim filtrom.

Želimo li riješiti problem visokofrekventne komponente šuma orijentacijskog kvaterniona i akumulirane greške kutne brzine pokušat ćemo spojiti obje metode procijene orijentacije u jednu metodu. Tako ćemo u konačnici dobiti procjenu koja rezultira manjom greškom. U tu svrhu uvest ćemo tzv. komplementarni filtar [11].

Komplementarni filtar

Komplementarni filtar zasniva se na ideji gdje izlaz filtra ovisi o dvije ulazne vrijednosti i to na način da filtru sa težinskim faktorom $K \in [0,\ 1]$ odredimo hoće li izlaz ovisiti o prvom, odnosno o drugom ulaznom parametru.

Cijela logika računanja orijentacijskog kvaterniona može se opisati dijagramom na slici 3.13. Obratimo pažnju na to kako s obzirom na težinski faktor K, izlaz filtra ovisi o jednom ili o drugom ulaznom parametru. Ako vrijedi da je $K = 1$, izlaz komplementarnog filtra (procijenjena orijentacija) ovisi isključivo o algoritmu određivanja orijentacije pomoću vektora kutne brzine. Obrnuto, ako je $K = 0$, izlaz filtra isključivo ovisi o algoritmu procijene orijentacije pomoću gradijentnog spusta [56]. Parametar K određuje se eksperimentalnim putem ili simulacijom.



Slika 3.13: Prikaz komplementarnog filtra za određivanje orijentacije satelita pomoću gradijentnog spusta i vektora kutne brzine [11].

3.3.2. Upravljanje orijentacijom/kutnom brzinom

Korišteni aktuatori

U našem radu za kontrolu orijentacije upotrijebili smo zamašnjak. Zamašnjak je pogonjen elektromotorom. Elektromotor se upravlja pomoću upravljivog H-mosta koji omogućuje promjenu smjera. Brzinom elektromotora upravlja se pomoću pulsno širinske modulacije (engl. *Pulse Width Modulation - PWM*). Rotirajuća masa izrađena je na 3D printeru.

Razlog zašto smo odabrali zamašnjak za upravljanje orijentacijom je u tome što je to bio najjednostavniji aktuator za izvedbu. Magnetorker bi zahtijevao izradu posebnog kaveza koji zamjenjuje Zemljino magnetsko polje što dodatno komplicira testni sustav.

PWM modulacija generira se u sklopovlju mikroupravljača.

Korišteni algoritmi

U našem sustavu implementirane su dvije regulacijske petlje: prva koja upravlja kutnom brzinom satelita i druga koja upravlja orijentacijom. Radi jednostavnosti obje regulacijske petlje reguliraju orijentaciju/kutnu brzinu samo oko jedne osi.

Upravljanje orijentacijom satelita prvo započinje određivanjem orijentacije. Jednom kada je orijentacija određena, upravljački PWM signal upravlja elektromotorom. Isto vrijedi i za upravljanje kutnom brzinom satelita.

Za obje regulacijske petlje koristimo PID regulator oblika:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}, \tag{3.12}$$

gdje je $u(t)$ upravljački izlaz regulatora, $e(t)$ razlika željene i trenutne veličine koje reguliramo, K_p, K_i i K_d koeficijenti PID regulatora.

Za implementaciju PID regulatora u računalu pogodan je diskretizirani oblik. Diskretizirani oblik PID regulatora [5] definiramo kao:

$$u(k) = K_P e(k) + K_I \sum e(k) + K_D(e(k) - e(k-1)), \tag{3.13}$$

gdje su koeficijenti diskretiziranog regulatora jednaki:

$$\begin{aligned} K_P &= K_p, \\ K_I &= \tfrac{K_p T}{T_i}, \\ K_D &= \tfrac{K_p T_d}{T}, \end{aligned} \tag{3.14}$$

i gdje je parametar T jednak vremenskom periodu pozivanja regulacijske petlje.

Razliku trenutne i željene vrijednosti $e(k)$ dovodimo na ulaz diskretiziranog PID regulatora koji zatim na izlazu daje upravljački signal $u(k)$. Upravljački signal PID regulatora je u intervalu $u(k) \in [0, 100]$ što nam direktno daje tzv. *Duty cycle* (DC) PWM-a. DC računamo kao [42]:

$$\text{DC} = \frac{T_{on}}{T_{on} + T_{off}} \times 100\,[\%]\,, \tag{3.15}$$

gdje je parametar T_{on} jednak vremenu visokog stanja PWM signala, a parametar T_{off} jednak vremenu niskog stanja PWM signala (odnosno 0). Zbroj vremena visokog i niskog stanja PWM-a daje nam period PWM signala $T_{\text{PWM}} = T_{on} + T_{off}$. Ako vrijedi da je $T_{on} = T_{\text{PWM}}$ to znači da je srednja vrijednost PWM signala jednaka naponu napajanja PWM-a. Vrijedi i obrnuto, ako je $T_{off} = T_{\text{PWM}}$ onda srednja vrijednost PWM signala iznosi 0.

Odabir PID koeficijenata za obje regulacijske petlje opisan je u poglavlju 5.2.2.

3.4. Razvijeni sustav

U ovom poglavlju bit će opisana razvijena tiskana pločica i sklopovlje. Također, navest ćemo izabrane orijentacijske senzore i aktuatore.

Cijeli ADCS sustav nalazi se unutar polusferične kugle koja je položena na zračni ležaj prikazan na slici 3.14.

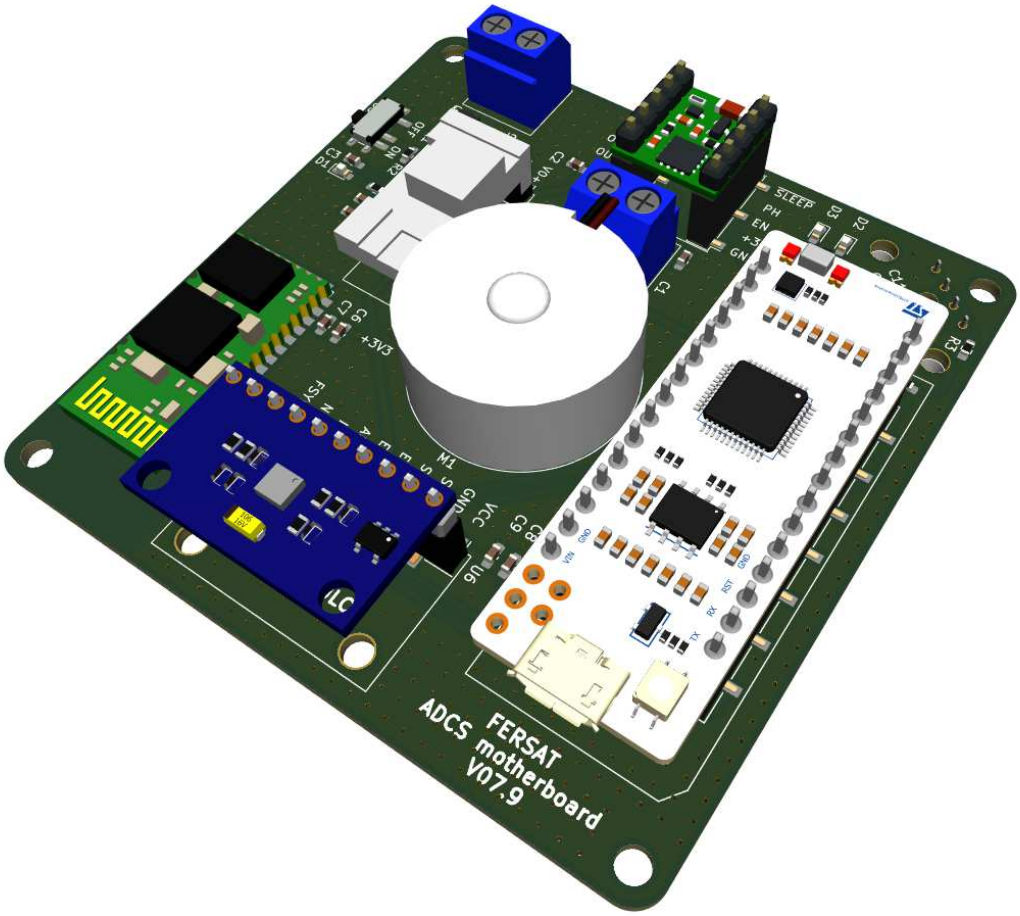
3.4.1. Tiskana pločica i sklopovlje

Tiskana pločica sa sklopovljem prikazana na slici 3.15 i sastoji se od nekoliko dijelova:

- Baterija,
- Napajanje,
- Pogonska elektronika zamašnjaka,
- Elektromotor,
- Bluetooth modul,
- Inkrementalni enkoder,
- IMU modul,
- Nucleo pločica.

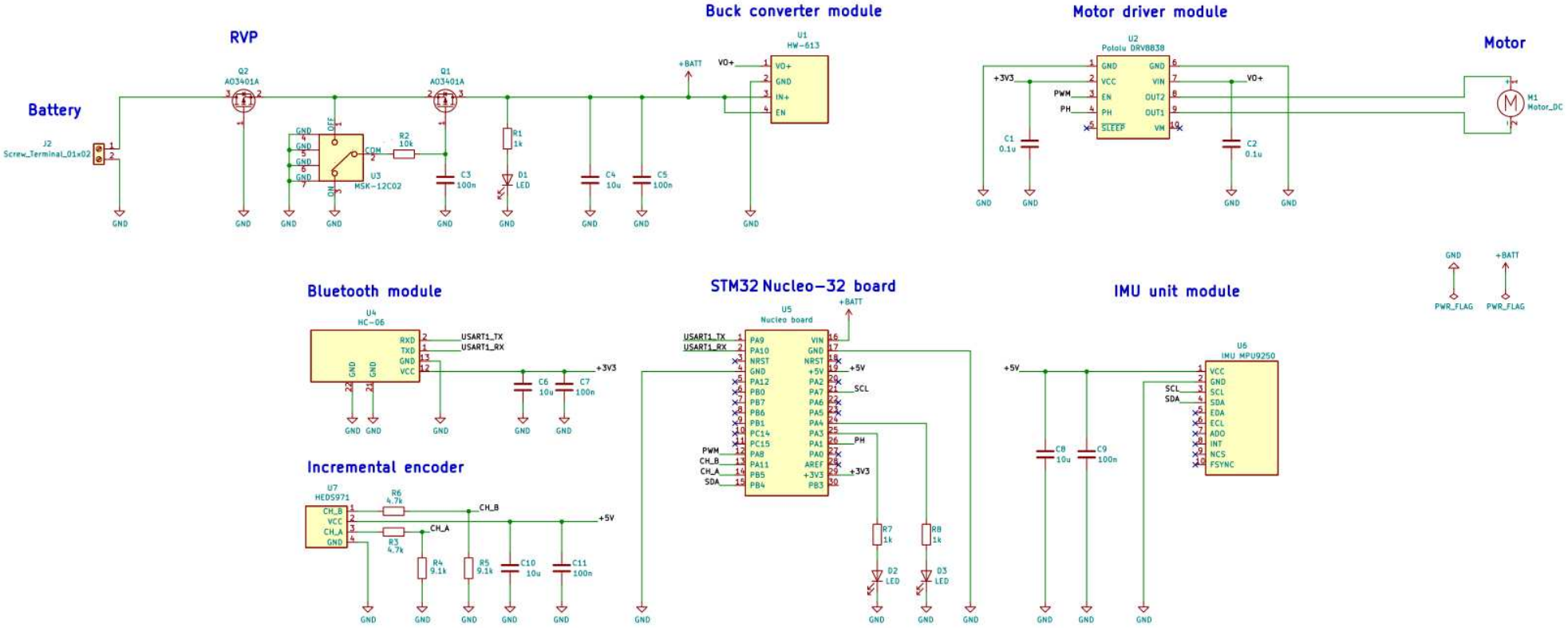
Slika 3.14: Prikaz ADCS sustava sa pripadajućim zračnim ležajem.

Shema tiskane pločice nalazi se na slici 3.16. Napon sa baterija dolazi do napajačkog sklopovlja (oznaka `Buck converter module`) koji smanjuje napon do razine pogodne za napajanje sklopovlja. Pogonska elektronika zamašnjaka (`Motor driver module`) oznake Pololu DRV8838 [49] je mali H-most modul za upravljanje elektromotorom zamašnjaka. Elektromotor je male snage i nalazi se u sredini tiskane pločice (`Motor`). Na tiskanoj pločici se također nalazi i Bluetooth modul oznake *HM-06* (`Bluetooth module`). Bluetooth modul nalazi se na samom rubu tiskane pločice radi bolje transmisije signala. Osim Bluetooth modula tu je i inkrementalni enkoder (`Incremental encoder`) pomoću kojeg je moguće izmjeriti kutnu brzinu zamašnjaka. Funkcionalnost inkrementalnog enkodera u trenutku pisanja rada još nije implementirana. Nadalje, na pločici se nalazi i IMU modul (`IMU unit module`) oznake *MPU9250* koji na sebi sadrži IMU (engl. *Inertial Measurement Unit*) senzor. Na kraju, pločica sadrži i utor za Nucleo pločicu (`NUCLEO-L412KB`) [58] koju je moguće jednostavno izvaditi i natrag staviti. Radi lakšeg razvijanja programske podrške na pločici se nalazi nekoliko statusnih LED dioda.



Slika 3.15: 3D prikaz tiskane pločice ADCS sustava.

Na Nucleo pločici morali smo napraviti nekoliko preinaka. Naime, naišli smo na problem postavljanja točne brzine prijenosa UART-a (engl. *baudrate*) na `UART1` periferiji (vidi [59]). `UART1` periferiji smo postavili standardnu brzinu prijenosa od 115200 bits/s da bi na izlazu dobili brzinu prijenosa od oko 108000 bits/s. Razlog tomu je što je Nucleo pločica tvornički postavljena da koristi oscilator unutar mikroupravljača koji je vrlo neprecizan. Da bi konfigurirali pločicu da koristi vanjski oscilator, morali smo fizički odlemiti kratkospojnike `SB5` i `SB7` i kratko spojiti kratkospojnik `SB17` (vidi [58]). Svatko tko želi koristiti programsku podršku opisanu u ovom radu mora napraviti jednake promjene na Nucleo pločici.

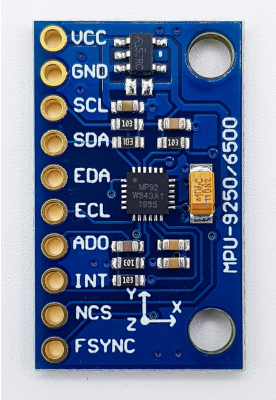


Slika 3.16: Shema tiskane pločice ADCS sustava.

3.4.2. Izabrani senzori

Zbog prirode algoritama za procjenu orijentacije (vidi poglavlje 3.3.1), potrebno je imati 2 neovisna troosna senzora. U praksi bi to npr. mogli biti senzor Sunca i magnetometar. U našem slučaju, radi jednostavnosti izvedbe testnog sustava i samog testiranja algoritma za estimaciju, izabrali smo akcelerometar kao prvi, odnosno magnetometar kao drugi senzor.

Oba troosna senzora (plus troosni žiroskop) nalaze se u jedinstvenom kućištu IMU senzora oznake *MPU9250* [23]. U našem sustavu koristimo Arduino modul sa MPU9250 senzorom prikazanom na slici 3.17.



Slika 3.17: Arduino modul sa MPU9250 senzorom [62].

Akcelerometar

Akcelerometar mjeri trenutnu akceleraciju na tijelo/senzor. Problem korištenja akcelerometra kao orijentacijskog senzora u satelitu koji orbitira oko svemirskog tijela je taj što na takav sustav ne djeluje nikakva rezultantna sila. Zbog odsutnosti rezultantne sile ne postoji akceleracija koju bi akcelerometar mjerio.

Drugi problem je što je akcelerometar senzor koji posjeduje relativno puno visokofrekventnog šuma, ali zato ne posjeduje tzv. *bias*.

Magnetometar

Magnetometar nam za podatak daje vektor Zemljinog magnetskog polja. Pomoću dostupnih alata na internetu (vidi [48]), moguće je odrediti referentni vektor Zemljinog magnetskog polja za odabrano mjesto na Zemlji. Magnetometar je povoljan orijentacijski senzor zato što ga je moguće koristiti u Zemljinoj orbiti. Ako bi znali podatak o referentnom vektoru magnetskog polja Zemlje za svaku točku orbite i imali izmjerenu

vrijednost Zemljinog magnetskog polja, mogli bismo pomoću estimacijskog algoritma izračunati orijentaciju satelita u orbiti.

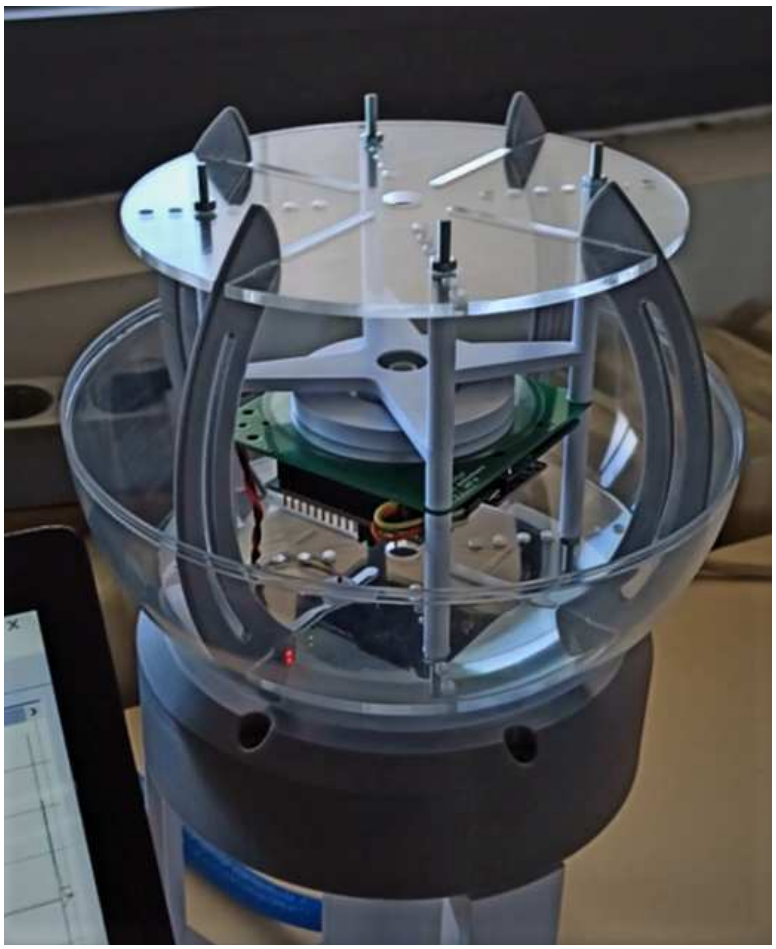
Žiroskop

Kao što smo ranije spomenuli, za uspješniju estimaciju orijentacije koristimo žiroskop. Unutar IMU senzora nalazi se troosni žiroskop.

3.4.3. Izabrani aktuatori

Za kontrolu orijentacije izradili smo zamašnjak kojeg je moguće vidjeti na slici 3.18. Elektromotor zamašnjaka položen je na tiskanu pločicu ADCS sustava. Rotirajuću masu zamašnjaka izradili smo pomoću 3D printera.

Zamašnjak smo izabrali zato što je to aktuator kojeg je relativno lako izraditi, dijelovi su lako dostupni, za pogon je potrebna samo električna energija koja dolazi iz baterije i pogonska elektronika elektromotora je jednostavne izvedbe.



Slika 3.18: Prikaz zamašnjaka sa ADCS sustavom.

4. Programska podrška ADCS sustava

U ovom poglavlju opisati ćemo programsku podršku ADCS sustava i opisat ćemo ugradbeno računalo koje koristimo za računanje i kontrolu aktuatora. Objasnit ćemo organizaciju programske podrške, navest ćemo korištene biblioteke, operacijski sustav, opisati funkcionalnost programa i u konačnici, ukratko opisati postupak razvijanja programske podrške. Napominjemo kako sljedeći opis sustava vrijedi isključivo za stanje sustava u trenutku pisanja ovog rada i moguće je da u trenutku čitanja postoje određene razlike.

Cijela programska podrška nalazi se na GitHub repozitoriju autora ovdje [27]. Programska podrška verzionirana je pomoću Git [16] programa za verzioniranje. Objašnjenja i opisi programske podrške vrijede do `f4d98b3` *commit*-a [26].

4.1. Ugradbeno računalo

Ugradbeno računalo koje koristimo je već gotov sustav na *NUCLEO-L412KB* razvojnoj pločici [58] STM proizvođača. Takav gotov sustav omogućuje nam jednostavno i brzo razvijanje programske podrške.

Nucleo razvojna pločica se sastoji od *STM32L412KBU6U* mikroupravljača niske potrošnje [59] koji sadrži tzv. jedinicu za računanje aritmetike pomičnog zareza (engl. *Floating point unit - FPU*). FPU nam omogućava hardversko ubrzanje računanja algoritama. Osim mikroupravljača, pločica sadrži i tzv. *ST-LINK* sučelje za jednostavno *debugiranje* i pohranu programa. Na samoj pločici nalazi se i upravljiva LED dioda.

4.2. Organizacija projekta

Programski kod podijeljen je u nekoliko direktorija i poddirektorija. Pogled iz glavnog direktorija sa opisom svakog poddirektorija prikazan je u tablici 4.1.

Nepotrebno je ulaziti u detalje svakog direktorija već ćemo u nastavku opisati samo što se nalazi unutar `src` direktorija koji sadrži programsku podršku ADCS sustava. Pogled iz `src` direktorija i opis svakog direktorija moguće je vidjeti je u tablici 4.2.

Tablica 4.1: Pogled iz glavnog direktorija projekta.

Naziv	Tip	Opis
cmake	Direktorij	Pomoćne CMake datoteke
docs	Direktorij	Popratnu dokumentaciju
.git	Direktorij	Git verzioniranje
.github	Direktorij	GitHub CI server
scripts	Direktorij	Pomoćne Shell skripte
src	Direktorij	Programska podrška
tests	Direktorij	Testovi programske podrške
.vscode	Direktorij	Pomoćne datoteke za VSCode editor
.clang-format	Datoteka	Konfiguracija za clang-format
CMakeLists.txt	Datoteka	CMake konfiguracija projekta
.gitignore	Datoteka	Imena direktorija/datoteka koje Git ne verzionira
.gitmodules	Datoteka	Imena i poveznice (linkovi) na eksterne Git repozitorije
LICENSE	Datoteka	Licenca projekta
Makefile	Datoteka	GNU Make komande za olakšano prevođenje (kompajliranje)
README.md	Datoteka	Opis projekta

4.3. Korištene biblioteke

U sklopu programske podrške koristimo i nekoliko biblioteka:

- comp_filt,
- mpu9250,
- optimal_request,
- printf,
- zs040.

Neke biblioteke su samostalni GitHub repozitoriji i u projekt su dodani kao tzv. Git submoduli. Primjer takvih biblioteka su: `mpu9250` koja upravlja MPU9250 IMU senzorom, `optimal_request` biblioteka koja sadrži kod za Optimal-REQUEST algo-

Tablica 4.2: Pogled iz `src` direktorija.

Naziv	Tip	Opis
bsp	Direktorij	engl. *Board support package*
core	Direktorij	Postavljanje osnovnih periferija mikroupravljača (clock, memorija itd.)
drivers	Direktorij	Drajveri za periferiju (I2C, UART, PWM, tajmeri itd.)
libs	Direktorij	Eksterne biblioteke (Bluetooth, MPU9250, komplementarni filtar itd.)
linker	Direktorij	Linker datoteke
mcu	Direktorij	Drajveri za periferiju (ali za naš mikroupravljač)
middlewares	Direktorij	Programski kod dretvi/aplikacije
rtos	Direktorij	Datoteke RTOS-a
utils	Direktorij	tzv. *Error handling*, matematičke funkcije
main.c	Direktorij	Main (glavna) funkcija
main.h	Direktorij	Main (glavna) funkcija

ritam, `printf` biblioteka koja sadrži jednostavniju verziju klasične `printf` funkcije i `zs040` biblioteka koja upravlja Bluetooth modulom. `comp_filt` biblioteka je jedina biblioteka koja nije u zasebnom GitHub repozitoriju i sadrži programski kod za Komplementarni filtar.

Svrha eksternih GitHub repozitorija (submodula) je u tome što to mogu biti samostalni repozitoriji koje netko drugi može preuzeti i ukomponirati u svoj projekt neovisno o tome na kojem sustavu se oni pokreću. Netko tko želi iskoristiti npr. OptimalREQUEST biblioteku u svojem projektu može ju jednostavno dodati u svoj repozitorij kao Git submodul.

4.4. Operacijski sustav i funkcionalnost

Zahtjev ADCS sustava je da u realnom vremenu obavlja nekoliko stvari istovremeno:

1. Estimacija orijentacije,
2. Kontrola orijentacije,
3. Komunikacija.

Za mogućnost istovremenog (paralelizam) izvršavanja nekoliko stvari istovremeno, odlučili smo se za operacijski sustav (OS). Zbog toga što se ADCS sustav odvija u

realnom vremenu izabrali smo operacijski sustav u stvarnom vremenu (engl. *Real Time Operating System - RTOS*).

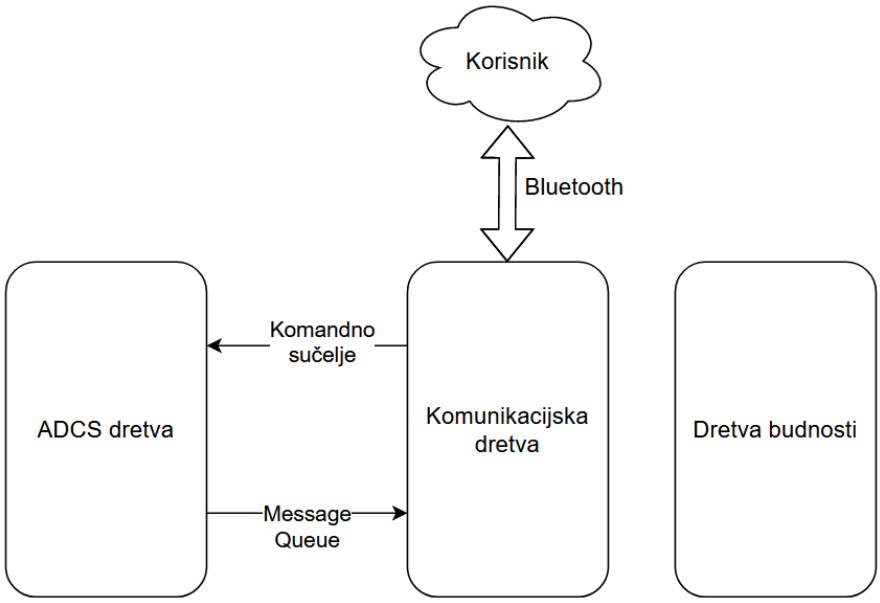
4.4.1. Operacijski sustav i dretve

Zbog popularnosti i široke zajednice, odlučili smo se za *FreeRTOS* operacijski sustav. *FreeRTOS* je besplatni operacijski sustav otvorenog koda koji sadrži podršku za mikroupravljač kojeg koristimo [60].

Sustav smo podijelili u nekoliko dretvi koje se izvršavaju paralelno:

1. Dretva budnosti (engl. *Alive thread*),
2. Dretva za estimaciju i kontrolu orijentacije,
3. Dretva za komunikaciju.

Shematski prikaz svih dretvi sa međusobnom komunikacijom moguće je vidjeti na slici 4.1. Detaljan opis svake dretve slijedi u nastavku.



Slika 4.1: Prikaz svih dretvi s međusobnim komunikacijskim kanalima.

Dretva budnosti

Dretva budnosti je jednostavna dretva u kojoj se periodički pali i gasi LED statusna dioda na samoj Nucleo pločici. Dretva budnosti nam omogućuje vizualnu provjeru rada

sustava. Ako iz nekog razloga izvršavanje sustava prestane, korisnik će na oko vidjeti kako statusna LED dioda ne treperi. To je znak kako je sustav otišao u neodređeno stanje. Pseudokod dretve budnosti nalazi se u nastavku:

```
while(True) {
    promijeni_stanje_statusne_ledice();
    čekaj_100_milisekundi();
}
```

Dretva za estimaciju i kontrolu orijentacije

Dretva za estimaciju i kontrolu orijentacije (ADCS dretva) sadrži programski kod koji periodički estimira trenutnu orijentaciju i na temelju toga upravlja regulacijskim petljama. Pseudokod ADCS dretve nalazi se u nastavku:

```
pričekaj_3_sekunde_za_stabilizaciju_satelita();

inicijaliziraj_imu();
inicijaliziraj_komplementarni_filtar();
inicijaliziraj_zamašnjak();
inicijaliziraj_regulacijske_petlje();

while(True) {
  imu_mjerenje = izmjeri_imu_vrijednosti();

  vrsta_regulacije = potraži_vrstu_regulacije();
  if (vrsta_regulacije == REGULACIJA_ORIJENTACIJE) {
    kvaternion = estimiraj_orijentaciju(imu_mjerenje);

    izmjereni_kut = izračunaj_eulerov_kut_z(kvaternion);
    željeni_kut = potraži_željeni_eulerov_kut();
    reguliraj_orijentaciju_oko_z_osi(željeni_kut, izmjereni_kut);

    izlaz_regulatora = potraži_izlaz_pid_regulatora_orijentacije();

    pošalji_preko_bluetootha(željeni_eulerov_kut,
      izmjereni_eulerov_kut,
      izlaz_regulatora);

  } else if (vrsta_regulacije == REGULACIJA_KUTNE_BRZINE) {
    željena_kutna_brzina = potraži_željenu_kutnu_brzinu();
    reguliraj_kutnu_brzinu_oko_z_osi(željena_kutna_brzina,
      imu_mjerenje.žiroskop.z);
```

```
29
30      izlaz_regulatora = potraži_izlaz_pid_regulatora_kutne_brzine();
31      pošalji_preko_bluetootha(željena_kutna_brzina,
32        imu_mjerenje.žiroskop.z,
33        izlaz_regulatora);
34
35    } else if (vrsta_regulacije == BEZ_REGULACIJE) {
36        kvaternion = estimiraj_orijentaciju(imu_mjerenje);
37        kutovi = izračunaj_eulerove_kutove(kvaternion);
38        željeni_kut = potraži_željeni_eulerov_kut();
39        zamašnjak = potraži_duty_cycle_zamašnjaka();
40
41        pošalji_preko_bluetootha(kutovi,
42          imu_mjerenje.žiroskop,
43          zamašnjak);
44    }
45
46    čekaj_100_milisekundi();
47  }
```

Pogledamo li pseudokod ADCS dretve vidjet ćemo da na samom početku imamo čekanje od 3 sekunde ne bi li se satelit smirio. Razlog tomu je što na samom početku inicijalizacije sustava imamo kalibraciju žiroskopa. Kalibracija žiroskopa nastoji pronaći konstantni bias pod uvjetom da je satelit u mirovanju, odnosno da je kutna brzina za svaku os jednaka 0. Tih 3 sekunde čekanja nam omogućuje da nakon ponovnog pokretanja sustava, imamo vremena satelit negdje položiti i pričekati da prođe vrijeme kalibracije.

Nakon 3 sekunde čekanja slijedi inicijalizacija IMU jedinice, komplementarnog filtra, zamašnjaka i obje regulacijske petlje.

Nakon inicijalizacije slijedi beskonačna petlja gdje se prvo potražuje mjerenje s IMU jedinice. IMU jedinica dat će podatke o vektorima akceleracije, magnetskog polja i kutne brzine za svaku os.

Jednom kada smo dobili vrijednosti mjerenja, slijedi potraživanje globalnog stanja regulacije. Postoje 3 globalna stanja regulacije:

1. Regulacija orijentacije oko z-osi,
2. Regulacija kutne brzine oko z-osi,
3. Bez regulacije.

Sva 3 stanja zapravo dolaze iz komunikacijske dretve koja omogućuje da korisnik u bilo koje vrijeme odabere jedan od tih stanja (popis komandi u tablici 4.3). Inicijalno stanje regulacije nakon svakog ponovnog pokretanja sustava je stanje bez regulacije.

Stanje regulacije orijentacije započinje estimacijom orijentacije. Funkcija koja estimira orijentaciju kao podatak prima mjerenja iz IMU jedinice (vektor akceleracije, magnetskog polja i kutne brzine), a na izlazu daje procijenjeni kvaternion. Funkcija za estimaciju orijentacije u pozadini poziva komplementarni filtar koji estimira kvaternion. Nakon što smo dobili kvaternion, računamo Eulerov kut oko z-osi. Nadalje, potražujemo željeni Eulerov kut koji zapravo dolazi također iz komunikacijske dretve jer korisnik u svakom trenutku može promijeniti željeni kut. Pomoću izračunatog (estimiranog) kuta i željenog kuta pozivamo funkciju za regulaciju orijentacije. U pozadini ta funkcija poziva PID regulacijsku petlju za kontrolu orijentacije (kuta). Na samom kraju dretve Bluetooth-om se šalju podaci korisniku o vrijednosti željenog (referentnog) Eulerovog kuta, trenutnog (estimiranog) Eulerovog kuta i izlazu PID regulatora. Te tri vrijednosti važne su nam u proučavanju odziva satelita prilikom regulacije orijentacije.

Stanje regulacije kutne brzine započinje pozivanjem funkcije regulacijske petlje za regulaciju kutne brzine. Funkcija koja regulira kutnu brzinu kao podatak prima mjerenja iz žiroskopa (samo z-os) i željenu kutnu brzinu. Željenu kutnu brzinu potražujemo iz komunikacijske dretve koja omogućava da korisnik u svakom trenutku promijeni željenu kutnu brzinu. Funkcija za regulaciju kutne brzine u pozadini poziva funkciju za PID regulaciju kutne brzine. Pri samom kraju dretve šaljemo korisniku podatke o vrijednosti željene (referentne) kutne brzine, trenutne (izmjerene) kutne brzine i izlaz PID regulatora.

Stanje bez regulacije započinje estimiranjem orijentacije, računanjem Eulerovih kutova i čitanjem vrijednosti Duty Cycle-a zamašnjaka (trenutne i referente). Sve to na kraju zajedno pošaljemo korisniku. Ti podaci su nam važni za snimanje odziva satelita na vanjske pobude i računanje koeficijenata pojednostavljenog modela (vidi poglavlje 5.2.1).

Na samom kraju dretve slijedi čekanje od 100 milisekundi. Obratimo pažnju i na to da obje regulacijske petlje ovise o periodu izvođenja ADCS dretve.

Dretva za komunikaciju

Dretva za komunikaciju sadrži programski kod koji je omogućuje komunikaciju između korisnika i ADCS sustava. Komunikacija sa satelitom odvija se u oba smjera:

od korisnika prema sustavu i od sustava prema korisniku. Komunikacija se vrši preko Bluetooth-a. Bluetooth komunikacija nam omogućuje jednostavno pristupanje satelitu pomoću računala ili pametnog telefona koji sadrži Bluetooth sučelje.

Komunikacijska dretva omogućava slanje podataka o trenutnom stanju satelita kao što su: trenutna orijentacija, kutna brzina, izlaz regulatora i sl. Druga bitna funkcionalnost dretve je mogućnost upravljanja satelitom od strane korisnika pomoću komandnog sučelja. Popis svih podržanih komandi upravljanja satelitom prikazani su u tablici 4.3.

Tablica 4.3: Popis komandi, njihov opis i primjer korištenja. **Napominjemo kako svaka komanda u primjeru mora završavati sa '\n' ASCII znakom.**

Komanda	Opis	Primjer korištenja
echo	Vraća nazad korisniku sve primljeno	`echo(ADCS)`
help	Ispisuje sve podržane komande	`help()`
reset	Ponovno pokretanje sustava	`reset()`
set_sending	Omogućuje/onemogućuje slanje poruka	`set_sending(1)`, (ili `0`)
set_reg_angvel	Pokreće regulaciju kutne brzine	`set_reg_angvel()`
set_reg_attitude	Pokreće regulaciju orijentacije	`set_reg_attitude()`
set_reg_no_reg	Onemogućuje regulaciju	`set_reg_no_reg()`
set_pid_p	Postavlja P član aktivnog PID regulatora	`set_pid_p(3.14)`
set_pid_i	Postavlja I član aktivnog PID regulatora	`set_pid_i(15)`
set_pid_d	Postavlja D član aktivnog PID regulatora	`set_pid_d(92.2)`
set_pid_v	Postavlja V član aktivnog PID regulatora	`set_pid_v(6.67)`
set_ref_angle_z	Postavlja ref. kut z-osi (stupnjevi)	`set_ref_angle_z(30)`
set_ref_angvel_z	Postavlja ref. kutnu brzinu z-osi (rad/s)	`set_ref_angvel_z(-1.2)`
set_rw_duty_cycle_z	Postavlja Duty cycle zamašnjaka	`set_rw_duty_cycle_z(99)`

Obratimo pažnju kako svaka komanda, da bi je satelit primio, mora završavati sa ASCII znakom `\n`, odnosno novim redom (engl. *new line, line feed - LF*). Također, komunikacijska dretva će ignorirati bilo koju drugu komandu koja nije opisana u tablici 4.3.

Svi parametri ADCS sustava koje želimo poslati preko Bluetooth sučelja moraju proći kroz komunikacijsku dretvu. Kako ADCS dretva jedina ima informaciju o npr. trenutnoj orijentaciji, morali smo implementirati način komunikacije između dretvi. Način komunikacije između dretvi implementiran je pomoću tzv. *Message Queue*-a [35].

Pseudokod komunikacijske dretve nalazi se u nastavku:

```
inicijaliziraj_bluetooth();

while (True) {
    if (provjeri_ima_li_poruka_za_slanje()) {
        poruka = potraži_poruke_od_drugih_dretvi();
        pošalji_poruku_preko_bluetootha(poruka);
    }

    if (provjeri_ima_li_poruka_od_korisnika()) {
        poruka = potraži_primljenu_poruku();

        komanda, arg = potraži_komandu_i_argument_iz_poruke(poruka);

        for (podržana_komanda : podržane_komande) {
            if (podržana_komanda == komanda) {
                funkcija = potraži_funkciju(komanda);
                funkcija(arg);
            }
        }
    }

    raspusti_dretvu();
}
```

Napominjemo kako funkcije za provjeru poruka za slanje i funkcije provjere primljenih korisničkih poruka nisu blokirajuće. Te funkcije provjeravaju ima li poruka koje se trebaju poslati korisniku ili ima li poruka koje je korisnik poslao sustavu. Na kraju beskonačne petlje nalazi se funkcija koja raspušta izvršavanje dretve i daje procesorsko vrijeme drugim dretvama. Komunikacijska dretva u normalnom stanju čeka poruke za slanje na način da ne oduzima procesorsko vrijeme ostalim dretvama jer je komunikacijska dretva najmanjeg prioriteta - izvršava se odvija jedino onda kada su sve ostale dretve u stanju čekanja. Primanje korisničkih komandi odvija se pomoću UART prekidne rutine gdje na svaki primljeni znak s Bluetooth modula, mikroupravljač odlazi u prekidnu rutinu i puni spremnik znakova koji sadrži sve primljene znakove. Jednom kada komunikacijska dretva dobije procesorsko vrijeme ona će provjeriti ima li primljenih znakova u spremniku znakova i sačinjavaju li oni korisničku komandu. Na temelju toga će komunikacijska dretva ići u daljnju obradu tih komandi.

4.5. Razvoj

U ovom dijelu će ukratko biti objašnjeno na koji način se razvija programska podrška ADCS sustava. Programska podrška sustava napisana je u C programskom jeziku standarda C99 [24]. Sva programska logika nalazi se unutar već spomenutog `src` direktorija, a testovi se razvijaju unutar `tests` direktorija.

Razvoj počinje tzv. kloniranjem projekta koji se nalazi na GitHub repozitoriju [27] pomoću Git programa. Git će klonirati sve datoteke koje se nalaze na repozitoriju i pohraniti ih lokalno na disk razvojne mašine. Napominjemo kako nije nužno klonirati projekt pomoću Git programa već se program može jednostavno preuzeti s GitHub repozitorija i ručno spremiti na disk. Autor preporučuje kloniranje Git-om jer je to jednostavniji postupak.

Napominjemo kako autor programske podrške za razvoj programske podrške koristi *Ubuntu 21.10* operacijski sustav (OS) jer je na tom OS jednostavnije instalirati sve razvojne alate (vidi `README.md` datoteku za upute). Razvoj je moguć i na Windows OS što je autor također koristio. Instaliranje svih alata za Ubuntu OS moguće je napraviti jednostavnim pokretanjem GNU Make komande `make install_deps`.

Za pomoć pisanju programskog koda autor se odlučio za *VSCode* tekstualni uređivač [43]. VSCode pruža mogućnost jednostavnijeg potraživanja programskog koda, bojanje programskih linija i podrške za debugiranje. Napominjemo kako korisnik može koristiti bilo koji drugi tekstualni uređivač.

Jednom kada smo implementirali željenu funkcionalnost, slijedi postupak prevođenja (engl. *compilation*). Za prevođenje koristimo *GNU Arm Embedded Toolchain* prevodilac verzije *v10.3-2021.10* [4]. Kao graditelj sustava (engl. *build system*) koristimo *CMake* verzije *v3.18.4* [36]. Pomoću CMake-a je moguće jednostavnije postaviti sustav za prevođenje neovisno o tome koju vrstu operacijskog sustava koristimo na razvojnoj mašini (Windows, Ubuntu i sl.). Prilikom prvog prevođenja potrebno je postaviti projekt pomoću Make programa tako što pokrenemo komandu `make setup_cmake`. Nakon postavljanja projekta moguće je pokrenuti prevođenje pomoću komande `make build`, ili kraće `make`.

Za provjeru rada programske podrške (debugiranje), koristimo *OpenOCD* [63] program koji u pozadini otvori GDB [18] server na kojeg se je moguće spojiti sa VSCode-om. Debugiranje započinjemo unutar VSCode-a tako što otvorimo *Run and Debug* karticu i pokrenemo debugiranje pritiskom na zelenu strelicu. VSCode će pokrenuti prevođenje i pohranu programa na mikroupravljač pod uvjetom da je Nucleo pločica priključena na razvojno računalo pomoću USB kabela. Postavimo li zaustavnu točku

(engl. *breakpoint*) unutar npr. glavne *main* funkcije, program bi morao na trenutak prestati s izvođenjem i korisniku dati pristup vrijednostima svih varijabli koje program koristi unutar *main* funkcije. Pritiskom na tipku nastavi (engl. *Continue*), program bi trebao nastaviti s izvođenjem.

Ako postupkom debugiranja vidimo neko neželjeno ponašanje programa, debugiranje se može prekinuti i grešku možemo prepraviti. Jednom kada smo prepravili grešku možemo opet na isti način pokrenuti postupak debugiranja.

Kada smo zadovoljili sve zahtjeve programske podrške, preporučuje se da se programski kod formatira. Formatiranje programskog koda je važno ako isti sustav uređuje više osoba istovremeno. Pretpostaviti je da svaka osoba ima svoj način pisanja programske podrške i zbog toga može doći do otežane čitljivosti koda. Da bi tome pribjegli koristimo *ClangFormat* program za formatiranje programskog koda [37]. Formatiranje programskog koda pokreće se komandom `make clang_format`.

Jednom kada smo uspješno napravili sve postupke, programski kod je moguće verzionirati pomoću Git programa postupkom koji je objašnjen ovdje [17] ili preinake poslati direktno autoru projekta (kontakt se nalazi na GitHub stranici projekta).

Kao zanimljivost navest ćemo broj linija programskog koda za svaki poddirektorij unutar `src` direktorija. Broj linija programskog koda nalazi se u tablici 4.4.

Tablica 4.4: Broj linija koda za svaki direktorij unutar `src` direktorija.

Direktorij	C99	Assembly
drivers	63330	0
libs	6776	0
rtos	13229	0
middlewares	1444	0
core	633	283
mcu	367	0
bsp	125	0
utils	53	0
top_dir	30	0

5. Eksperimentalna verifikacija ADCS sustava

U ovom poglavlju objasnit ćemo testni sustav za eksperimentalnu verifikaciju programske podrške i navesti postupke i rezultate eksperimentalne verifikacije.

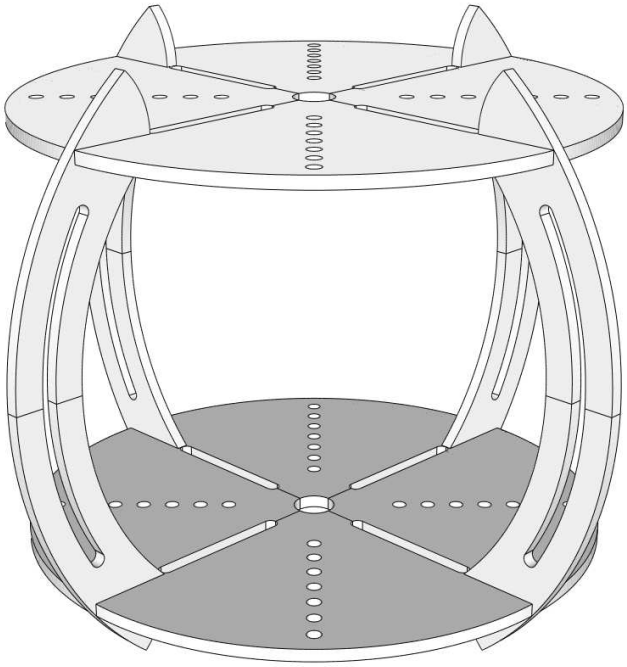
5.1. Opis sustava i korištenih alata

Verifikacijski sustav sastoji se od nekoliko dijelova. Prvi i najvažniji dio je tzv. zračni ležaj sa pripadajućom plastičnom kuglom.

Zračni ležaj sastoji se od 3D isprintane polusferične podloge koja se napaja stlačenim zrakom koji dolazi iz kompresora zraka. Stlačeni zrak dolazi na površinu zračnog ležaja i stvara tanki film zračne struje koja smanjuje trenje između ležaja i plastične kugle. Pomoću zračnog ležaja dobivamo uvjete koji vladaju u orbiti oko Zemlje gdje nema otpora zraka koji bi stvarao parazitni moment na satelit. Plastična kugla sastoji se od vanjske ljuske i plastičnog držača satelita. Plastična vanjska ljuska sastavljena je od dvije polusfere. Plastični držač izrađen je od laserski rezanog plexiglasa i prikazan je na slici 5.1. Posebnim navojnim odstojnicima satelit je pričvršćen na plastični držač [55].

Za komuniciranje sa satelitom koristimo računalo ili pametni mobilni telefon koji sadrže Bluetooth sučelje. Za klasično komandno sučelje na Windows računalu koristimo program *Termite* [10] prikazan na slici 5.2, a na Android mobilnom telefonu aplikaciju *Serial Bluetooth Terminal* [34] prikazanu na slici 5.3.

Podešavanje *Termite* programa je vrlo jednostavno. Postupak započinje spajanjem računala i satelita preko Bluetooth sučelja čiji postupak nećemo objašnjavati jer ovisi verziji Windows operacijskog sustava. Nakon što smo se uspješno priključili računalom na Bluetooth satelita, odabiremo pripadajući COM port unutar *Termite* aplikacije. Ako je sve uspješno obavljeno u programu je moguće vidjeti podatke koje dolaze sa satelita u obliku običnog teksta. Jednako tako unutar *Termite* programa moguće je slanje



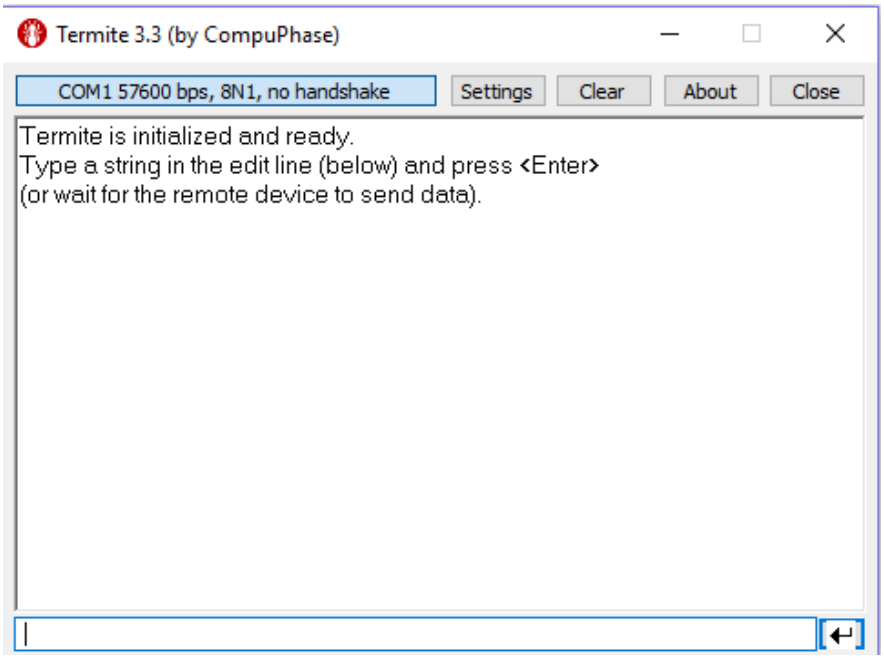
Slika 5.1: Prikaz plastičnog držača satelita [55].

komandi prema satelitu. Važno je napomenuti kako svaka komanda mora završavati sa $'\backslash n'$ znakom što je moguće podesiti kao postavku unutar *Termite* programa.

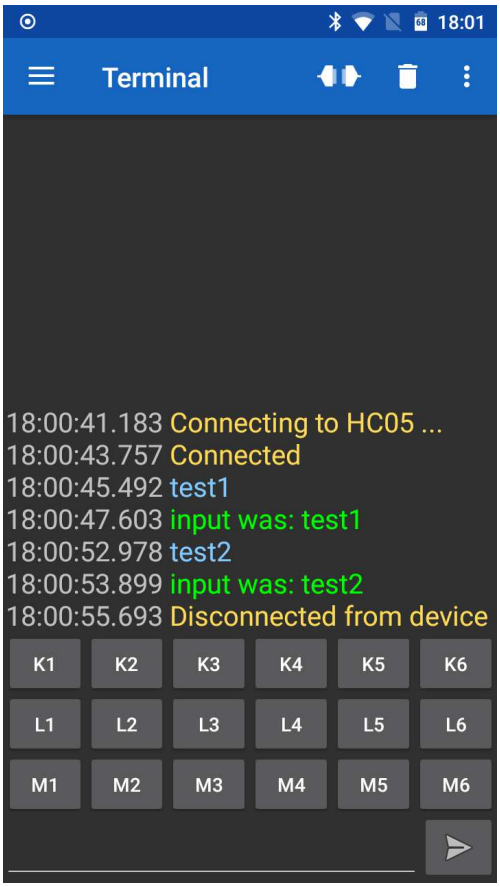
Podešavanje *Serial Bluetooth Terminal* Android mobilne aplikacije i spajanje sa satelitom vrlo je jednostavno i neće biti objašnjeno. Nakon uspješnog spajanja sa satelitom, unutar aplikacije se pojavljuju podaci sa satelita. Slanje komandi prema satelitu jednostavno upisuju u za to predviđeni prozor unutar aplikacije. Napominjemo kako je u postavkama aplikacije potrebno podesiti da svaka komanda završava sa $'\backslash n'$ znakom.

Želimo li grafički prikazivati vrijednosti koje dolaze sa satelita, to možemo učiniti sa *SerialPlot* programom [20]. *SerialPlot* program pruža različite mogućnosti: primanje i slanje komandi, grafičko prikazivanje vrijednosti, spremanje podataka u CSV datoteku i još mnogo toga. Program je besplatan i otvorenog koda. Sučelje *SerialPlot* programa prikazano je na slici 5.4.

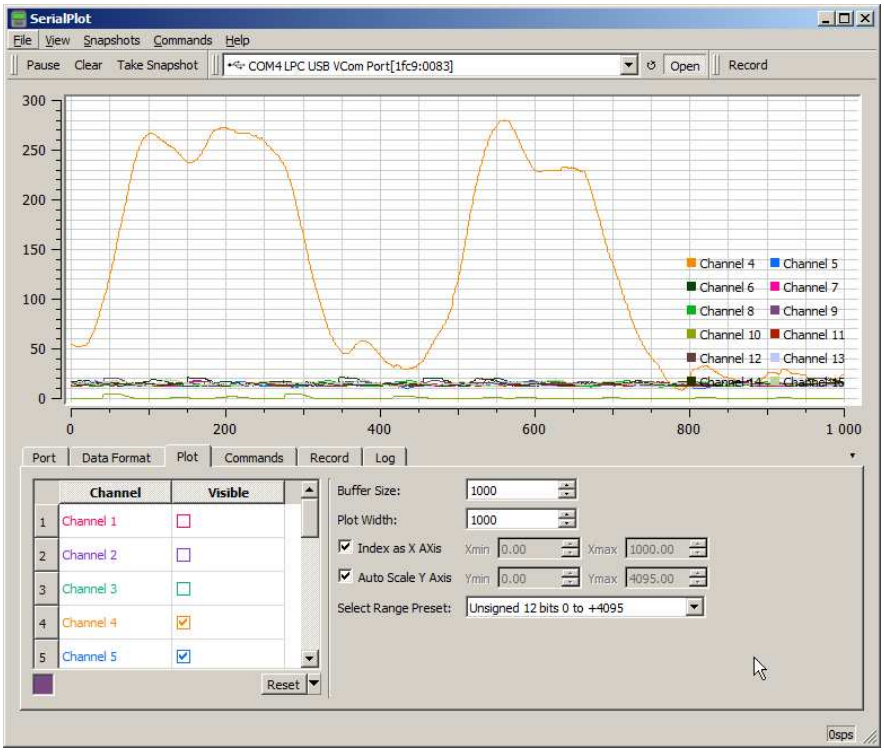
Postupak spajanja *SerialPlot* programa i satelita praktično je isto kao i spajanje sa Termite programom i neće biti ovdje objašnjeno. Prilikom uspješnog spajanja na ekranu se pojavljuju grafičke vrijednosti sa satelita. U programu je moguće odabrati automatsku detekciju broja podataka u ovisnosti o tome na koji način šaljemo vrijednosti. U našem slučaju, vrijednosti šaljemo kao cijele brojeve odvojene znakom zareza (,). Takvu postavku moramo odabrati i unutar *SerialPlot* programa. Svaku pojedinačnu krivulju moguće je sakriti ili prikazati odabirom tzv. zelene kvačice unutar programa.



Slika 5.2: Prikaz *Termite* aplikacije za pristup COM komandnom sučelju [10].



Slika 5.3: Prikaz *Serial Bluetooth Terminal* Android aplikacije [34].



Slika 5.4: Prikaz *SerialPlot* aplikacije za grafički prikaz vrijednosti sa COM sučelja [20].

5.2. Određivanje parametara

5.2.1. Pojednostavljeni model satelita

Kao što smo već pisali u poglavlju 2.6, pojednostavljeni kinematički model satelita (jednadžba 2.26) definirali smo kao:

$$H(s) = \frac{Y(s)}{X(s)} = K\frac{1/\tau}{s + 1/\tau}, \tag{5.1}$$

gdje je $Y(s)$ kutna brzina sustava u radijanima po sekundi (rad/s), $X(s)$ predstavlja Duty Cycle zamašnjaka u postotnom omjeru (%), a koeficijente K i τ želimo odrediti.

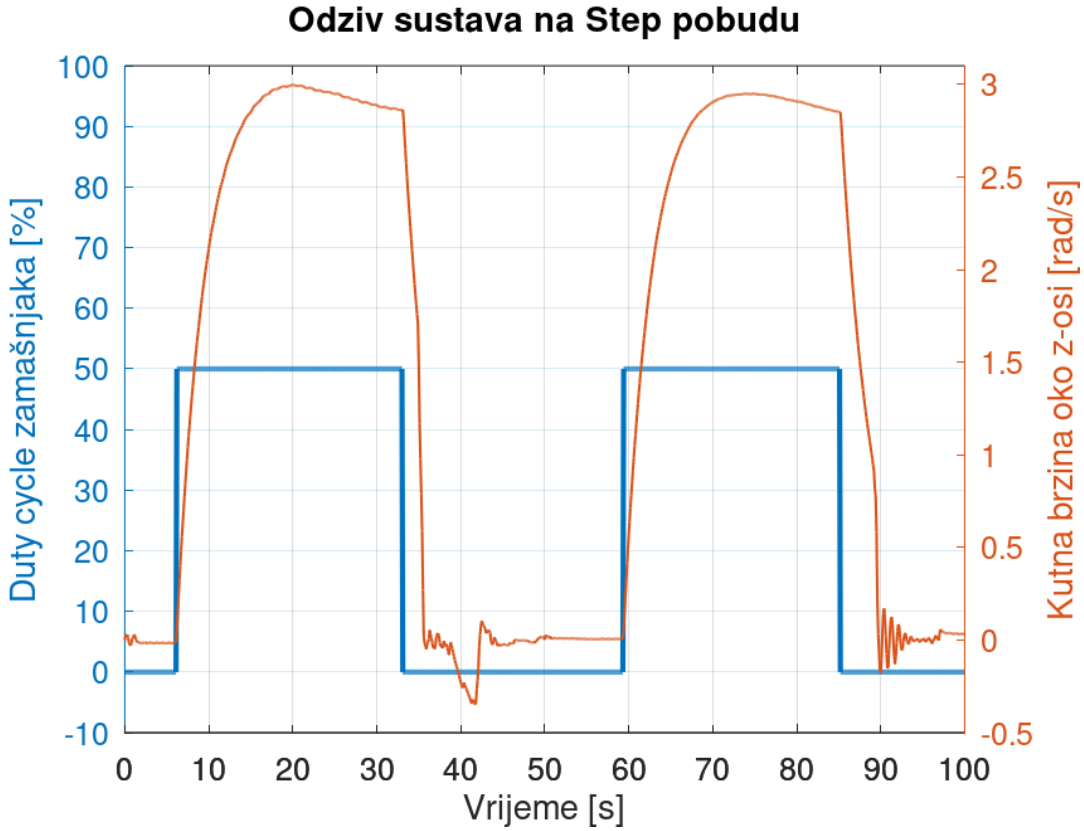
Jednom kada smo odredili te koeficijente, moguće je izračunati odziv sustava $Y(s)$ za bilo koju vrstu pobude $X(s)$. Poznavajući model sustava, jednostavnije je razviti regulacijske petlje s kojima reguliramo neke odabrane veličine (npr. kutnu brzinu).

Postupak određivanja parametara započinje pokretanjem ADCS sustava. Nakon što se ADCS sustav pokrenuo i kada je uspješno prošla sva inicijalizacija, satelit postavljamo na zračni ležaj tako da satelit u potpunosti miruje. Nakon što smo uspostavili sve uvjete za testiranje, možemo krenuti dalje na snimanje odziva sustava na tzv. skokovitu (engl. *Step*) pobudu.

Snimanje sustava na skokovitu pobudu započinje pokretanjem programa *SerialPlot*

(vidi poglavlje 5.1) pomoću kojeg je vrlo lako moguće snimiti sve vrijednosti u CSV format datoteke. Snimljene vrijednosti zatim je lako analizirati pomoću matematičkih alata kao što je MATLAB. Prilikom pokretanja *SerialPlot* programa odaberemo COM port na kojemu je moguće dobiti podatke iz satelita. Namjestimo li automatsko primanje svih parametara na ekranu ćemo vidjeti 3 vrste signala: estimirani Eulerovi kutovi, mjerenje sa žiroskopa i Duty Cycle zamašnjaka. Pomoću mjerenja žiroskopa i vrijednosti DC zamašnjaka moguće je odrediti kinematički model satelita.

Odziv sustava na dvije skokovite pobude prikazan je na slici 5.5. Oko 7. sekunde poslali smo satelitu naredbu da zavrti zamašnjak sa 0% na 50% DC nakon koje kutna brzina počinje naglo rast do nekih 3 rad/s. Primijetimo kako oko 20. sekunde kutna brzina počinje blago padati. Razlog tomu su blage nepravilnosti zračnog ležaja koje uzrokuju parazitni moment koji usporava satelit. Modeliranje parazitnog momenta nismo uzimali u obzir radi jednostavnosti jednadžbi. Postupak snimanja odziva na skokovitu pobudu napravili smo još jednom od 60. do 85. sekunde.



Slika 5.5: Odzivi sustava na skokovitu pobudu sa DC zamašnjaka na 50%.

Funkciju skokovite pobude u Laplaceovoj domeni zapisujemo kao:

$$S(s) = \frac{\mathrm{DC}}{s}, \tag{5.2}$$

gdje je DC parametar jednak Duty Cycle-u zamašnjaka.

Koristeći jednadžbu 2.26 i jednadžbu 5.2, odziv na skokovitu pobudu satelita u Laplaceovoj domeni možemo zapisati kao:

$$Y(s) = H(s)S(s) = K\frac{1/\tau}{s+1/\tau}\,\frac{\text{DC}}{s}, \tag{5.3}$$

ili u vremenskoj kao domeni:

$$y(t) = (h * s)(t) = K \cdot \text{DC} \cdot (1 - e^{-\frac{t}{\tau}}), \tag{5.4}$$

Pomoću MATLAB alata i pripadajućih MATLAB skripti koje je moguće pronaći unutar `docs/step_response` direktorija [29], moguće je na jednostavan način dobiti K i τ parametre kinematičkog modela. Primjer dobivanja parametara preko MATLAB skripti moguće je vidjeti na slici 5.6. Napisana skripta prvo učitava već snimljene podatke (vidi sliku 5.5) i izlučuje trenutak eksponencijalnog rasta kutne brzine. Na temelju tog vremenskog odsječka skripta će kao rezultat dati K i τ koeficijente. Za naš sustav i pobudu od $\text{DC} = 50\%$ dobili smo sljedeće vrijednosti koeficijenata:

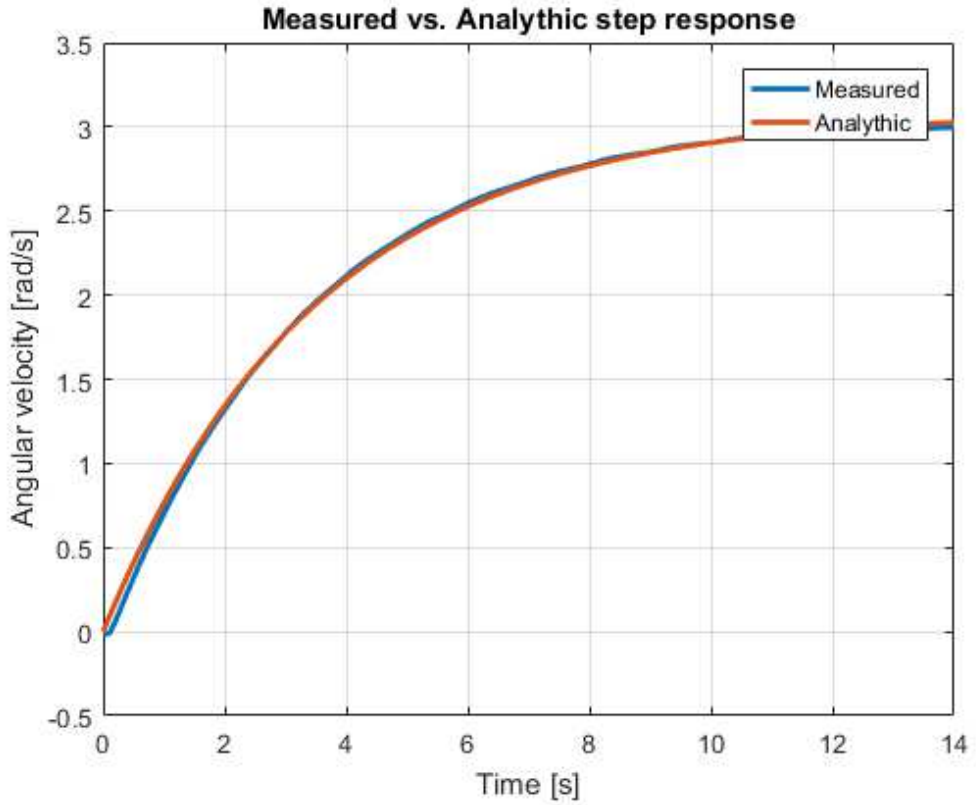
$$\begin{aligned} K &= 0.0616\ \text{rad/s/\%}\ , \\ \tau &= 3.50\ \text{s}\ . \end{aligned} \tag{5.5}$$

Spomenute MATLAB skripte, osim estimacije parametara kao rezultat daju i kinematičke modele. Prvi kinematički model odnosi se na modeliranje kutne brzine (vidi jednadžbu 2.26), a drugi na modeliranje orijentacije/kuta (jednadžba 2.27). Pomoću tih modela nalazimo parametre PID regulatora (vidi sljedeće poglavlje).

5.2.2. PID regulator

Regulator kutne brzine

Pomoću spomenuta dva modela možemo eksperimentalno odrediti koeficijente PID regulatora. Otvorimo *PID Tuner* MATLAB aplikaciju [39] u kojemu prvo unosimo prvi kinematički model - model kutne brzine satelita. Podešavanjem kliznika na vrhu prozora moguće je PID regulator ugoditi na željeni odziv u ovisnosti o tome želimo li brži odziv s relativno velikim izdizanjem/prekoračenjima (engl. *overshoot*) ili sporiji odziv sa blažim izdizanjem. U drugom prozoru moguće je otvoriti graf na kojemu je prikazan izlaz regulatora (engl. *controller effort*). Pomoću takvog grafa možemo provjeriti da nam izlaz iz regulatora ne prelazi DC zamašnjaka jer se u programski izlaz regulatora ograničava na 100% DC zamašnjaka što nam uvodi nelinearnosti u sustav.



Slika 5.6: Prikaz grafa odziva na skokovitu pobudu i grafa funkcije kojoj smo pronašli nepoznate parametre.

Za regulaciju kutne brzine dobili smo sljedeće vrijednosti koeficijenata kontinuiranog PID regulatora:

$$\begin{aligned} K_p &= 42.5\,, \\ K_i &= 14.8\,, \\ K_d &= 0.0\,. \end{aligned} \tag{5.6}$$

Obraćamo pažnju kako su ovo koeficijenti kontinuiranog PID regulatora. Koeficijente diskretiziranog PID regulatora moguće je dobiti pomoću jednadžbi 3.14. Vremensko pozivanje regulacijske petlje kutne brzine je $T_d = 100$ ms

Regulator orijentacije

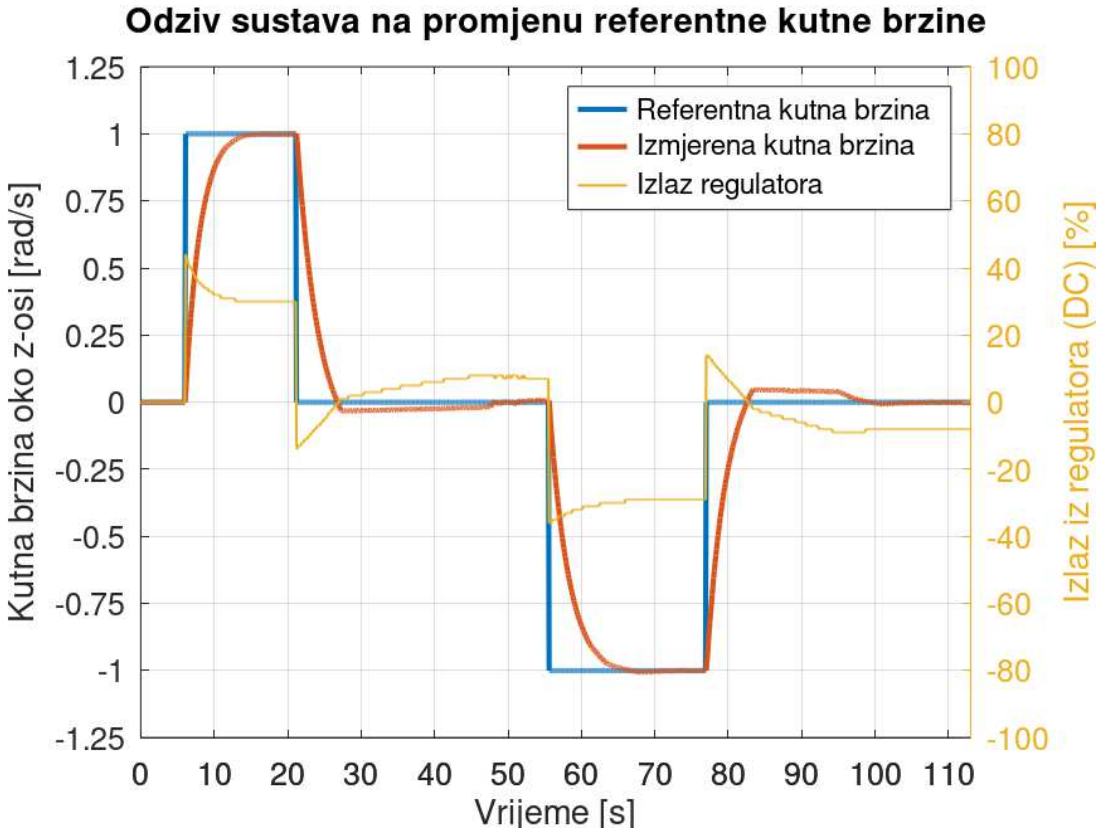
Jednaki postupak određivanja koeficijenata PID regulatora vrijedi i za regulaciju orijentacije, odnosno kuta. Skripta će na izlazu, osim modela kutne brzine, dati i model orijentacije. Taj model možemo također učitati u *PID Tuner* aplikaciju i po potrebi ugoditi koeficijente PID regulatora. Za regulaciju orijentacije dobili smo sljedeće vrijednosti koeficijenata PID regulatora:

$$\begin{aligned} K_p &= 50.0\ , \\ K_i &= 3.0\ , \\ K_d &= 100.0\ . \end{aligned} \tag{5.7}$$

Obraćamo pažnju kako su ovo koeficijenti kontinuiranog PID regulatora. Koeficijente diskretiziranog PID regulatora moguće je dobiti pomoću jednadžbi 3.14. Vremensko pozivanje regulacijske petlje orijentacije je $T_d = 100$ ms

5.3. Rezultati verifikacije

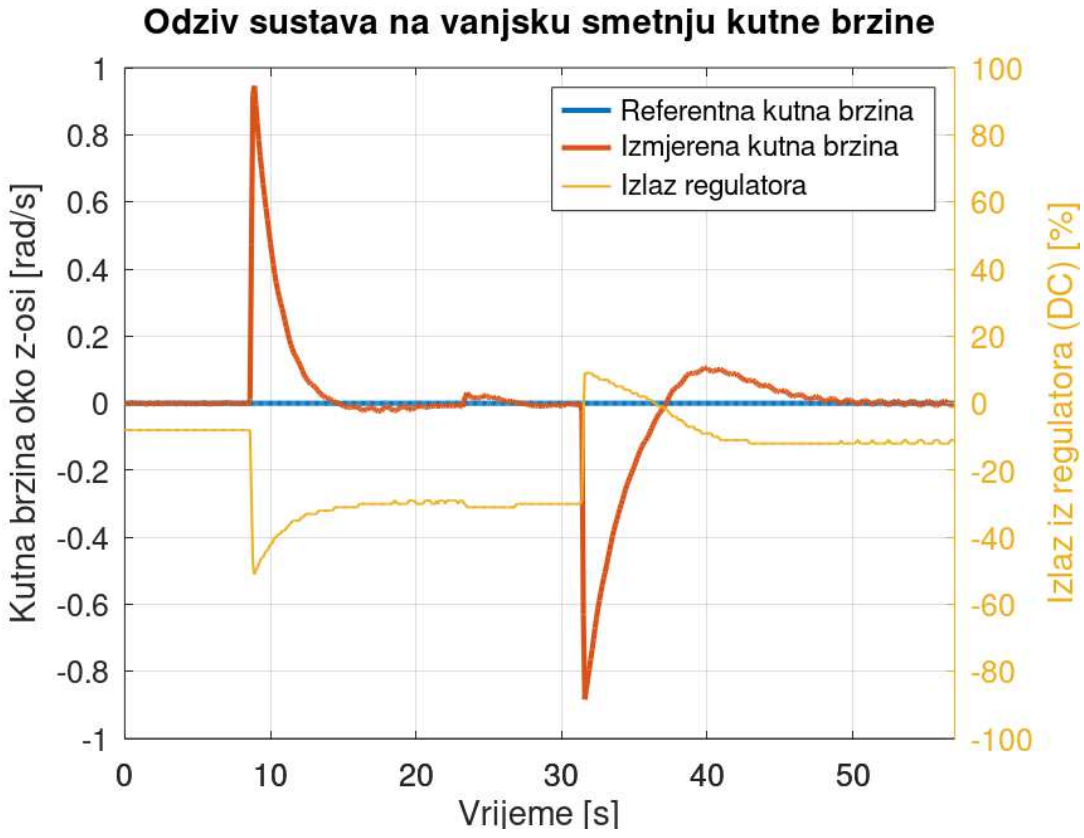
Rezultate parametrizacije PID regulatora snimili smo prilikom eksperimentalne verifikacije sustava. Na slici 5.7 prikazan je odziv sustava na promjenu željene kutne brzine u nekoliko navrata. U prvom navratu korisnik je u 5. sekundi komandom zadao željenu kutnu brzinu od 1 rad/s. Možemo vidjeti kako kutna brzina naglo počinje rasti sve dok satelit ne dosegne željenu brzinu od 1 rad/s oko 20. sekunde. Neposredno nakon toga satelitu je poslana naredba za referentnu brzinu od 0 rad/s i to se može jasno vidjeti naglim ali kontroliranim padom do 0 rad/s.



Slika 5.7: Odzivi sustava na promjenu referentne kutne brzine.

Jednaki postupak izveli smo u 55. sekundi gdje smo satelitu zadali kutnu brzinu od 1 rad/s ali sada u suprotnom smjeru. Žuto obojana linija na grafu predstavlja vrijednosti izlaza regulatora. Izlaz regulatora direktno utječe na DC zamašnjaka. Negativna vrijednost označava suprotni smjer vrtnje zamašnjaka. Primijetimo kako graf izlaza regulatora ne prelazi $\pm 100\%$. Primijetimo još i činjenicu kako oko 30. i 85. sekunde postoji blago nadvišenje s obzirom na referentnu kutnu brzinu koje traje relativno dugo. Uzrok tog nadvišenja je mala greška referentne i željene kutne brzine koja dolazi na ulaz PID regulatora. PID regulator množi tu malu grešku sa K članom ali to i dalje nije dovoljno za značajniju korekciju. O utjecaju D člana nema govora jer je sustav u stacionarnom stanju i ne postoje visokofrekventne komponente na koje bi D član reagirao. Primijetimo kako je izlaz PID regulatora u početku relativno mali i da s vremenom raste. To je upravo utjecaj I člana koji s vremenom integrira malu grešku i koji će nakon dovoljno vremena postati toliko značajan da će vršiti korekciju kutne brzine.

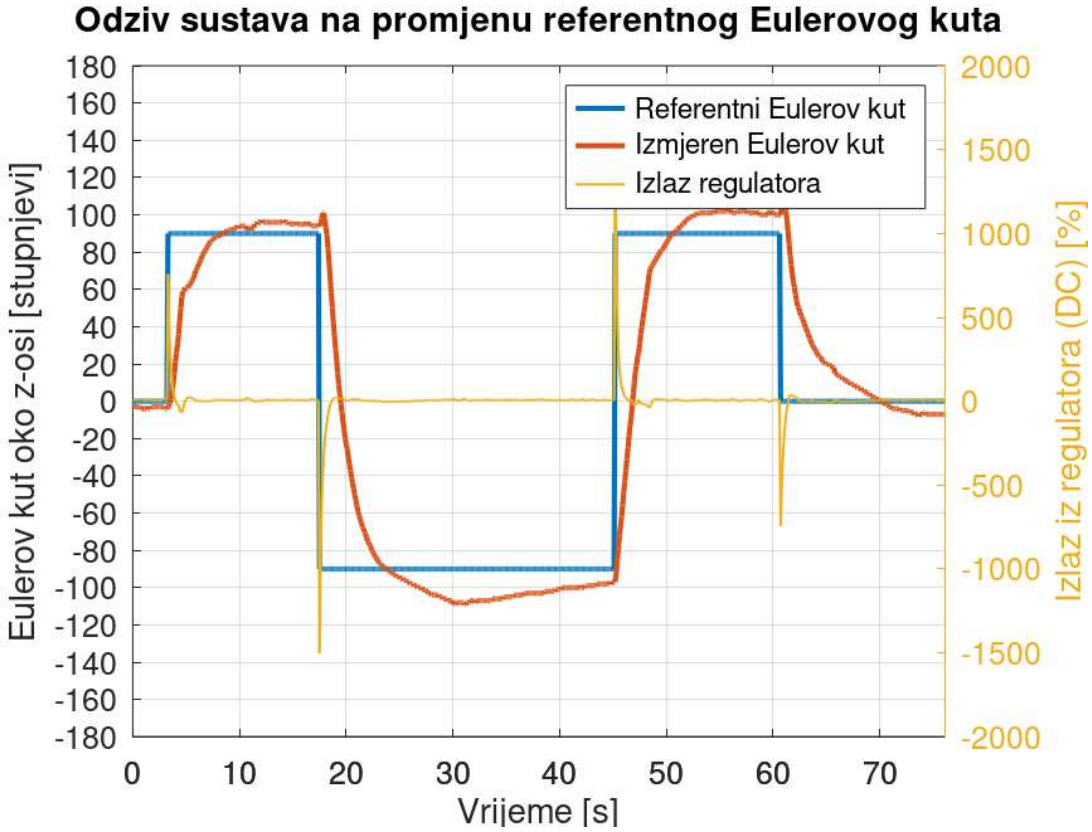
Na slici 5.8 vidimo odziv regulatora na vanjske smetnje gdje smo u nekoliko navrata ručno pomaknuli satelit i tako mu doveli vanjsku smetnju. Dva su trenutka smetnje: prvi oko 10. sekunde i drugi oko 32. sekunde. Prva smetnja ima različitu orijentaciju u odnosu na drugu zbog simetričnosti regulacije.



Slika 5.8: Odzivi sustava na vanjsku smetnju kutne brzine.

Vidimo kako je referentna kutna brzina cijelo vrijeme postavljena u 0 rad/s. Primijetimo kako regulator vrlo brzo vrši korekciju i da je tranzijentno vrijeme vrlo kratko. Zanimljivo je još jednom primijetiti manifestacije nepravilnosti zračnog ležaja. Vidljivo je na samom početku grafa da izlaz regulatora nije 0% nego ima neku određenu vrijednost. To se događa zato što zračni ležaj daje konstantni moment kojeg regulator pokušava kompenzirati. Uočimo kako oko 40. sekunde postoji znatno nadvišenje kutne brzine što je u svakom slučaju negativna pojava. Daljnja eksperimentalna verifikacija mogla bi ukazati postojanje boljih parametara PID regulatora koji bi dali bolju regulacijsko ponašanje.

Nadalje, rezultate ugađanja PID regulatora za regulaciju kuta možemo vidjeti na slici 5.9 gdje je korisnik u 3. sekundi zadao referentni kut od 90 stupnjeva. Vidimo kako od tog trenutka imamo nagli skok iz estimiranog kuta do 90 stupnjeva u 16. sekundi. Nakon toga korisnik zadaje naredbu za referentnu orijentaciju od -90 stupnjeva. Primijetimo kako orijentacija počinje naglo padati i prelazi u drugu stranu. Na grafu je vidljivo kako postoji relativno veliko nadvišenje.

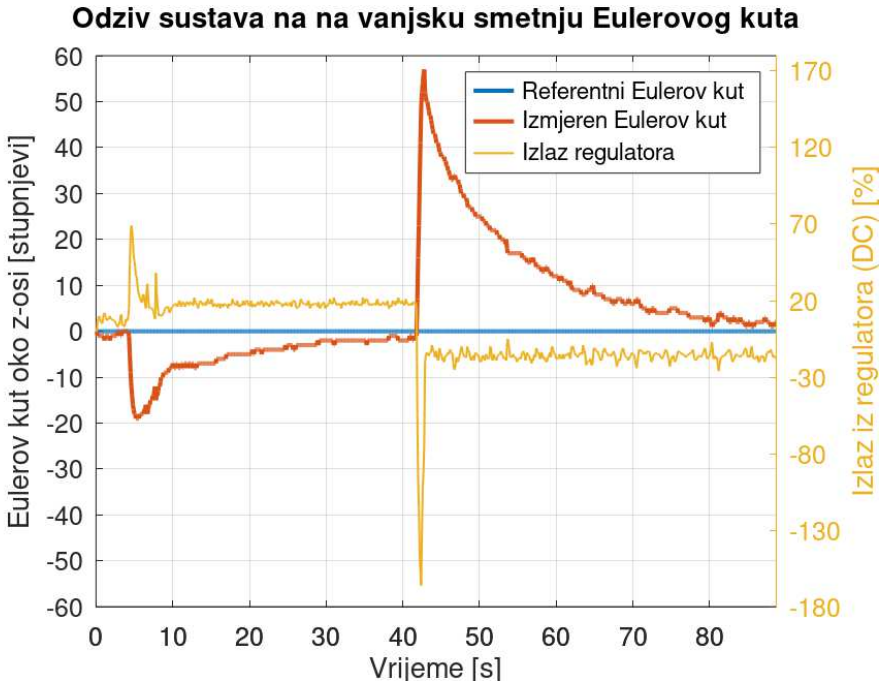


Slika 5.9: Odzivi sustava na promjenu referentnog Eulerovog kuta.

Uzrok tog nadvišenja je u početku relativno velika izlazna komponenta PID regulatora gdje PID regulator zamašnjaku zadaje naredbu brzine vrtnje od 1500% DC-a.

To je naravno nemoguće jer DC zamašnjaka može maksimalno biti 100%. Razlog toliko velikog izlaza leži u relativno velikom D članu koji zapravo i mora biti velik jer mora kompenzirati veliku integralnu komponentu kinematičkog modela za orijentaciju satelita (jednadžba 2.27). Bez obzira što regulator traži od zamašnjaka da se zavrti nemogućim brzinama, sustav je tako razvijen da će *odrezati* (engl. *clamp*) vrijednosti više/niže od $\pm 100\%$ DC zamašnjaka. Na grafu je vidljivo kako regulatoru treba relativno dugo vrijeme kompenzacije greške. Razlog tomu je što su K i P komponente relativno malog iznosa te mora proći neko vrijeme sve dok I komponenta PID regulatora ne postane dovoljno velika kako bi kompenzirala nadvišenje. Tijekom verifikacije testirali smo različite kombinacije PID parametara no ni jedna nije rezultirala zadovoljavajućim rezultatima. Pretpostavljamo kako se sustavi 2. reda teško reguliraju konvencionalnim PID regulatorom (kojeg mi koristimo), već se rabe druge inačice PID regulatora prikladnije regulaciju sustava 2. reda [3].

Na slici 5.10 možemo vidjeti odziv regulatora na vanjske smetnje gdje smo ručno sustav pomaknuli iz referentnog položaja. Prvi put u 5. sekundi pomaknuli smo sustav za otprilike -20 stupnjeva. Zbog visokofrekventne komponente smetnje, D član u početku prevladava sve dok I član s vremenom ne postane dominantniji. Drugu vanjsku smetnju generirali smo u 42. sekundi gdje smo satelit ručno pomakli za oko 57 stupnjeva. Regulator je kompenzirao grešku za oko 50 sekundi. I ovdje je jasno vidljiv negativan utjecaj nepravilnosti zračnog ležaja.



Slika 5.10: Odzivi sustava na vanjsku smetnju Eulerovog kuta.

6. Zaključak

U sklopu ovog rada uspješno smo razvili programsku podršku sustava za određivanje i kontrolu orijentacije satelita (ADCS). Eksperimentalnom verifikacijom pokazali smo kako sustav vrlo dobro procjenjuje orijentaciju pomoću komplementarnog filtra metodom gradijentnog spusta. Pomoću dodatnih alata uspješno smo pronašli koeficijente pojednostavljenog modela kutne brzine i orijentacije satelita, te smo uspješno odredili parametre PID regulatora za regulaciju kutne brzine i orijentacije oko jedne osi. Eksperimentalno smo pokazali kako pomoću zamašnjaka i regulacijskih petlji možemo vrlo dobro regulirati kutnu brzinu oko jedne osi bez neželjenih prekoračenja kutne brzine od one zadane. Također smo pokazali kako regulacija orijentacije nije zadovoljavajuća s trenutno izabranim modelom PID regulatora, već planiramo implementirati regulator koji će biti prikladniji regulaciji orijentacije. Jednom kada postignemo zadovoljavajuće upravljanje kutnom brzinom i orijentacijom oko jedne osi, sustav planiramo nadograditi regulacijama oko sve tri osi.

LITERATURA

[1] Alén Space. Q&a on cubesat payloads: What can you put in a small satellite? https://info.alen.space/cubesat-payloads-what-can-you-put-in-a-small-satellite, 2020. [pristupljeno 28-Lipanj-2022].

[2] James R. Forbes Anton H. de Ruiter, Christopher Damaren. *Spacecraft Dynamics and Control: An Introduction*. Wiley, 2013. ISBN 9781118342367.

[3] Mituhiko Araki i Hidefumi Taguchi. Optimal-request algorithm for attitude determination. *International Journal of Control, Automation, and Systems*, 1(4), 2003.

[4] Arm Limited. Arm gnu toolchain downloads. https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/downloads, 2022. [pristupljeno 28-Lipanj-2022].

[5] Atmel Corporation. *AVR221: Discrete PID controller*, 2006.

[6] I. Y. Bar-Itzhack. Request - a recursive quest algorithm for sequential attitude determination. *Journal of Guidance, Control, and Dynamics*, 19(5):1034–1038, 1996. doi: 10.2514/3.21742.

[7] Bradford Engineering BV. *Coarse Sun Sensor Datasheet*, 1975.

[8] Eric Stoneking Brent Robertson. *Satellite GN&C Anomaly Trends*. American Astronautical Society, 2003.

[9] I. Y. B.-I. Y. O. D. Choukroun. Optimal-request algorithm for attitude determination. *Journal of Guidance, Control, and Dynamics*, 27(3):418–425, 2014.

[10] CompuPhase. Termite: a simple rs232 terminal. https://www.compuphase.com/software_termite.htm, 2022. [pristupljeno 28-Lipanj-2022].

[11] Dubravko Babić, Ana Babić, Josip Lončar, Josip Vuković. Predavanja na predmetu uvod u svemirske tehnologije. https://www.fer.unizg.hr/predmet/uust, 2022. [pristupljeno 28-Lipanj-2022].

[12] Erik Kulu. World's largest database of nanosatellites. https://www.nanosats.eu, 2022. [pristupljeno 28-Lipanj-2022].

[13] ESA Earth Observation Portal. Ørsted geomagnetic mission. https://directory.eoportal.org/web/eoportal/satellite-missions/o/oersted, 2002. [pristupljeno 28-Lipanj-2022].

[14] EUROCKOT Launch Services GmbH. Eurockot successfully launches mom - rockot hits different orbits. https://web.archive.org/web/20100303013451/http://www.eurockot.com/alist.asp?cnt=20040706&main=5, 2010. [pristupljeno 28-Lipanj-2022].

[15] EXOLAUNCH GmbH. Serdeployment adapters. www.ecm-space.de, 2016. [pristupljeno 28-Lipanj-2022].

[16] Git. Službene stranice git programa. https://git-scm.com. [pristupljeno 28-Lipanj-2022].

[17] GitHub. Git guides. https://github.com/git-guides/git-commit, 2022. [pristupljeno 28-Lipanj-2022].

[18] GNU. Gdb - gnu development tools. https://man.archlinux.org/man/community/arm-none-eabi-gdb/arm-none-eabi-gdb.1.en, 2022. [pristupljeno 28-Lipanj-2022].

[19] Christopher D. Hall. *When Spacecraft Wont Point*. American Astronautical Society, 2004.

[20] Hasan Yavuz Özderya. Serialplot. https://github.com/hyOzd/serialplot, 2022. [pristupljeno 28-Lipanj-2022].

[21] Hrvatski vojnik. Satelitska mikrorevolucija. https://hrvatski-vojnik.hr/satelitska-mikrorevolucija/, 2006. [pristupljeno 28-Lipanj-2022].

[22] Ivan Indir, Karla Sever, Ivan Vnučec, i Josip Lončar. *Design and Optimization of Air Core Magnetorquers for Attitude Control of LEO Nanosatellites*, 2021.

[23] InvenSense Inc. *MPU-9250 Product Specification*, 2016.

[24] *ISO/IEC 9899:1999*. ISO, IEC, 1999.

[25] Ivan Vnučec. Implementacija optimal-request algoritma. https://github.com/IvanVnucec/Optimal-REQUEST/blob/master/docs/optimal_request_croatian.pdf, 2021. [pristupljeno 28-Lipanj-2022].

[26] Ivan Vnučec. Verzija programske podrške adcs sustava u trenutku pisanja rada. https://github.com/IvanVnucec/cubesat-adcs/tree/f4d98b3425eeb4909aece4a200a083e3d3f88e7a, 2022. [pristupljeno 28-Lipanj-2022].

[27] Ivan Vnučec. Github repozitorij programske podrške adcs sustava. https://github.com/IvanVnucec/cubesat-adcs, 2022. [pristupljeno 28-Lipanj-2022].

[28] Ivan Vnučec. Isječak programskog koda za kalibraciju magnetometra. https://github.com/IvanVnucec/cubesat-adcs/blob/f4d98b3425eeb4909aece4a200a083e3d3f88e7a/src/middlewares/adcs/adcs_imu.c#L74, 2022. [pristupljeno 28-Lipanj-2022].

[29] Ivan Vnučec. Matlab skripte za pronalaženje koeficijenata modela. https://github.com/IvanVnucec/cubesat-adcs/blob/06a5a8e6566654c79b3c06791dd24b9fb783fe0e/docs/step_response/import_csv_data_and_fit.m, 2022. [pristupljeno 28-Lipanj-2022].

[30] Ivan Vnučec. Programski kod optimal-request algoritma. https://github.com/IvanVnucec/Optimal-REQUEST, 2022. [pristupljeno 28-Lipanj-2022].

[31] G. Wang J. Li i X. Wei. *Generation of Guide Star Catalog for Star Trackers*, 2018.

[32] Jeff Watson. Mems gyroscope provides precision inertial sensing in harsh, high temperature environments. https://www.analog.com/en/technical-articles/mems-gyroscope-provides-precision-inertial-sensing.html. [pristupljeno 28-Lipanj-2022].

[33] Jaehyun Jin, Sangho Ko, i Chang-Kyung Ryoo. Fault tolerant control for satellites with four reaction wheels. *Control Engineering Practice*, 16(10):1250–1258, 2008. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2008.02.001. URL https://www.sciencedirect.com/science/article/pii/S0967066108000233.

[34] Kai Morich. Serial bluetooth terminal. https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=hr&gl=US, 2022. [pristupljeno 28-Lipanj-2022].

[35] Keil. Cmsis-rtos api v2 - message queue. https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__Message.html. [pristupljeno 28-Lipanj-2022].

[36] Kitware. Cmake releases - v3.18.4. https://github.com/Kitware/CMake/releases/tag/v3.18.4, 2020. [pristupljeno 28-Lipanj-2022].

[37] LLVM. Llvm 12.0.0. https://github.com/llvm/llvm-project/releases/tag/llvmorg-12.0.0, 2021. [pristupljeno 28-Lipanj-2022].

[38] MathWorks. Matlab coder - help center. https://www.mathworks.com/products/matlab-coder.html, 2022. [pristupljeno 28-Lipanj-2022].

[39] MathWorks. Pid tuner - help center. https://www.mathworks.com/help/control/ref/pidtuner-app.html, 2022. [pristupljeno 28-Lipanj-2022].

[40] MathWorks Help Center. Magnetometer calibration. https://www.mathworks.com/help/fusion/ug/magnetometer-calibration.html, 2022. [pristupljeno 28-Lipanj-2022].

[41] MathWorks Help Center. magcal function documentation. https://www.mathworks.com/help/nav/ref/magcal.html, 2022. [pristupljeno 28-Lipanj-2022].

[42] Myo Maung, Maung Latt, i Chaw Nwe. Dc motor angular position control using pid controller with friction compensation. *International Journal of Scientific and Research Publications (IJSRP)*, 8, 11 2018. doi: 10.29322/IJSRP.8.11.2018.p8321.

[43] Microsoft. Download visual studio code. https://code.visualstudio.com/download, 2022. [pristupljeno 28-Lipanj-2022].

[44] NanoAvionics. *Magnetorquers MTQ3X Datasheet*, 2019.

[45] NanoAvionics. Cubesat reaction wheels control system satbus 4rw0. https://nanoavionics.com/cubesat-components/cubesat-reaction-wheels-control-system-satbus-4rw, 2022. [pristupljeno 28-Lipanj-2022].

[46] NASA. Lewis spacecraft failure board report released. https://carlkop.home.xs4all.nl/lewis.html, 1998. [pristupljeno 28-Lipanj-2022].

[47] *NROL-36 Features Auxiliary Payloads*. National Reconnaissance Office, 2012.

[48] NOAA. Magnetic field calculators. https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml#igrfwmm, 2022. [pristupljeno 28-Lipanj-2022].

[49] Pololu Corporation. Drv8838 single brushed dc motor driver carrier. https://www.pololu.com/product/2990, 2022. [pristupljeno 28-Lipanj-2022].

[50] Redwire Corporation. Spectratrac star tracker. https://redwirespace.com/products/spectratrac/?rdws=nnn.xffxcv.tfd&rdwj=44045. [pristupljeno 28-Lipanj-2022].

[51] Rocket Lab. Data sheet for reaction wheel 3mnms rw-0.003. https://www.rocketlabusa.com/assets/Uploads/RL-RW-0.003-Data-Sheet.pdf, 2022. [pristupljeno 28-Lipanj-2022].

[52] Ademir L. Xavier Jr. Rui C. Botelho A. S. A unified satellite taxonomy proposal based on mass and size. *Advances in Aerospace Science and Technology*, 4(4), 2019.

[53] Bruce A. Sherwood Ruth W. Chabay. *Matter and Interactions, Volume 1: Modern Mechanics*. Wiley, 2019. ISBN 9781119625209.

[54] *Small Satellite Market Intelligence Report*. Satellite Applications Catapult Limited, 2021.

[55] Karla Sever, Nikša Ovčina, Filip Božić, Dino Cindrić, i Josip Lončar. *Low Cost 3D-Printed Air Bearing for CubeSat ADCS Testing*.

[56] Karla Sever, Ivan Indir, Ivan Vnučec, i Josip Lončar. *Evaluation of Gradient Descent Algorithm for Attitude Estimation*, 2021.

[57] M.D. Shuster i S.D. Oh. "three-axis attitude determination from vector observations. *Journal of Guidance and Control*, 4(1):70–77, 1981.

[58] *UM1956 User manual*. STMicroelectronics, 2018.

[59] STMicroelectronics. *RM0394 Reference manual*, 2018.

[60] STMicroelectronics. Stm32cube mcu full package for the stm32l4 series. `https://github.com/STMicroelectronics/STM32CubeL4`, 2022. [pristupljeno 28-Lipanj-2022].

[61] G. Wahba. A least squares estimate of spacecraft attitude. *SIAM Review*, 7(3): 409, 1965.

[62] Wolfgang Ewald. Mpu9250 - 9-axis sensor module - part 2. `https://wolles-elektronikkiste.de/en/mpu9250-9-axis-sensor-module-part-2`, 2021. [pristupljeno 28-Lipanj-2022].

[63] xpack-dev-tools. xpack openocd v0.11.0-4. `https://github.com/xpack-dev-tools/openocd-xpack/releases`, 2022. [pristupljeno 28-Lipanj-2022].

[64] Zavod za komunikacijske i svemirske tehnologije. Studentski satelit fersat - sažetak projekta. `https://www.fer.unizg.hr/zkist/FERSAT/projekt`, 2018. [pristupljeno 28-Lipanj-2022].

Programska podrška sustava za određivanje i upravljanje orijentacijom satelita

Sažetak

Precizna orijentacija satelita preduvjet je za uspješnu provedbu misije, stoga je sustav za određivanje i kontrolu orijentacije (ADCS) jedan od najvažnijih sustava satelita. U ovom radu prezentirana je programska podrška ADCS sustava koja omogućuje: sakupljanje i obradu podataka iz orijentacijskih senzora, određivanje orijentacije, upravljanje aktuatorima te održavanje željene orijentacije. Na kraju rada prezentirani su i komentirani rezultati eksperimentalne verifikacije sustava.

Ključne riječi: Orijentacija satelita; estimacija orijentacije; kontrola orijentacije; ADCS; Cubesat

Software for Satellite Attitude Determination and Control System

Abstract

Accurate satellite orientation is a prerequisite for a successful mission, therefore the Attitude Determination and Control System (ADCS) is one of the most important satellite systems. In this thesis, the ADCS system software is presented, which enables the collection and processing of the orientation sensors data, attitude determination, actuator control and finally, control of the desired orientation. In the end, ADCS experimental verification results are presented.

Keywords: Satellite orientation; orientation estimation; orientation control; ADCS; Cubesat