Vilniaus universitetas

Fizikos fakultetas

Cheminės fizikos institutas

Augustas Jurevičius

INERCINĖS NAVIGACINĖS SISTEMOS IR JOS BANDYMO STENDO MODELIŲ TYRIMAS

Pagrindinių studijų baigiamasis darbas

Moderniųjų technologijų fizikos ir vadybos studijų programa

Studentas	Augustas Jurevičius
Leista ginti	2023-05-26
Darbo vadovas	dr. Liudas Tumonis
Recenzentas	dr. Gytis Sliaužys
Insituto vadovas	dr. Darius Abramavičius

Vilnius 2023

Turinys

Įvadas 2
1. Inercinės navigacijos sistemos modelis 3
1.2. Kampinė padėtis 4
1.3. INS akselerometrų matuojamų dydžių interpretacija 8
1.4. INS matuojamų kampinių greičių ir tiesinių pagreičių integravimas 9
2. Matematinis bandymų platformos su pritvirtinta INS modelis 13
2.1. Matematinio INS testavimo platformos modelio apibrėžimas 13
2.2. INS, įtvirtintos ant testavimo platformos, modelis 15
3. Programos aprašas 23
4. Rezultatai 25
Literatūros sąrašas 35
Summary 36
Priedai 37

Įvadas

Žmonės niekados nebuvo pernelyg sėslūs padarai. Anksčiau ar vėliau visada atsirada reikiamybė migruoti, trauktis nuo plėšrūnų, bėgti nuo gamtos stichijų ar, apibendrintai, keliauti. Tam buvo reikalinga bent apytiksliai žinoti erdvinę ir kampinę padėtį – buvimo vietą ir kryptį, kuria keliaujama [2]. Nuo pasikliovimo aplinkine flora ir fauna, žvaigždžių stebėjimo, žodinių instrukcijų perdavimo ar žemėlapių nagrinėjimo evoliucija ir proveržiai moksle privedė prie tikslaus navigacijos uždavinio apibrėžimo – nustatyti junančio kūno erdvinę ir kampinę padėtį duotu laiko momentu - ir jo įgyvendinimo. Vienas iš plačiausiai naudojamų prietaisų navigacijos uždaviniui spręsti – inercinės navigacijos sistemos (INS). Sistemą sudaro bent vienas kompiuteris, trys tarpusavyje ortogonalūs akselerometrai ir trys ortogonalūs giroskopai, kartais – magnetomerai ar altimetrai. Pagrindinis INS privalumas – kad šios sistemos veikia visiškai autonomiškai, nepriklausomai nuo palydovinių ar radio signalų, taigi tokios sistemos nepamainomos laivyboje ar orlaivystėje. Tiesa, INS turi ir trūkumų. Kadangi kompiuteris turi atlikti integravimą (vieną kartą orientacijos atnaujinimui, du kartus pozicijos nustatymui), betkokios paklaidos, atsiradusios matavimuose ar netinkamai parinktame skaitiniame integravimo metode, gana greitai ir stipriai išauga. Vertinti jutiklių paklaidoms ar skaitinių algoritmų netikslumams, kuriami bandymo stendai. Jie leidžia kontroliuoti bandymo sąlygas (tiksliai valdyti erdvinę ir kampinę padėtis). Todėl reikalingi inercinių navigacijos sistemų modeliai ir jų stendai bei kontroliuojama aplinka siekiant testuoti individualius INS jutiklius ar visą sistemą, įvertinti triukšmą bei jo priežastis.

Šio darbo tikslas buvo sukurti ateityje planuojamo INS bandymų stendo kinematikos matematinį modelį ir sumodeliuoti ant tokio stendo pritvirtinto INS jutiklių matuojamų dydžių kitimą, kurį bus galima palyginti su tikros sistemos matavimo duomenimis. Darbui atlikti buvo išsikelti šie uždaviniai:

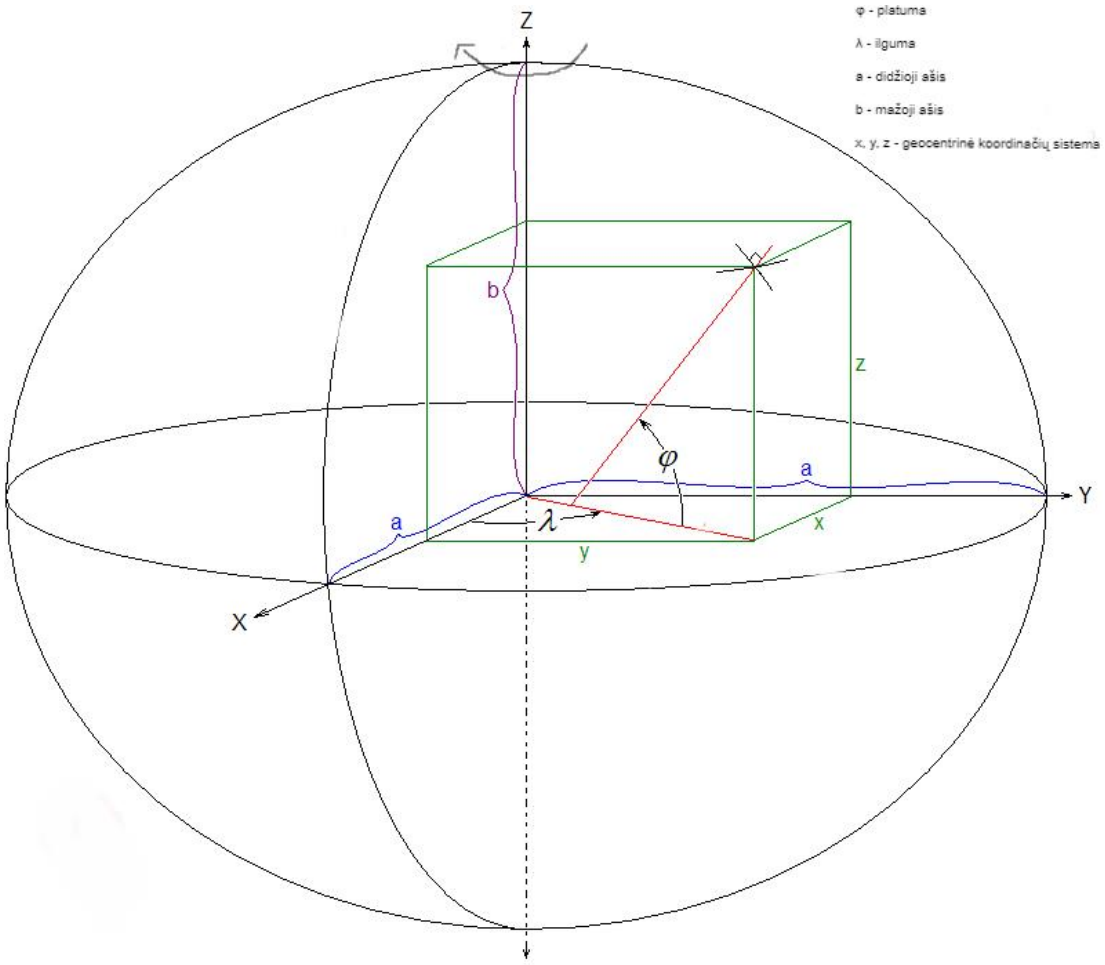
- Išnagrinėti fiksuotos INS veikimo principus, matematinį aparatą.
- Sudaryti matematinį parametrinį stendo kinematikos modelį.

1. Inercinės navigacijos sistemos modelis

Navigacija – tai kūno erdvinės ir kampinės padėties ir judėjimo dydžių, tokių kaip judėjimo ir sukimosi greitis bei pagreitis nustatymas pasirinktos koordinačių sistemos atžvilgiu. Šiame skyriuje bus pristatytos pagrindinės inercinės navigacijos sąvokos, metodai ir matematiniai kinematiniai parametrai.

1.1. Kūno padėties apibrėžimas

Norint išspręsti navigacijos uždavinį – apibrėžti kūno erdvinei ir kampinei padėčiai, būtina pasirinkti atitinkamą atskaitos koordinačių sistemą, kadangi INS giroskopų ir akselerometrų duomenys pateikiami su jutikliu susietoje koordinačių sistemoje. Galimų atskaitos koordinačių sistemų pasirinkimų yra be galo daug, tačiau populiariausios yra geocentrinė inercinė arba geocentrinė koordinačių sistemos.



1 pav. Geocentrinė koordinačių sistema. Adaptuota pagal [8].

Geocentrinės ir geocentrinės inercinės koordinačių sistemų ašys nurodytos (1 pav.) Esminis skirtumas tarp šių sistemų – geocentrinės inercinės koordinačių sistemos *XY* plokštuma sutampa su pusiaujo plokštuma, bet kartu su ja nesisuka; jos ašys yra orientuotos dangaus sferos šviesulių atžvilgiu.

Kadangi geocentrinė inercinė koordinačių sistema nesisuka kartu su Žemės rutuliu, yra patogi naudoti sudaryti judėjimo lygtis objektams kosmose. Geocentrinė sistema sukasi kartu su Žemės rutuliu, dėl to yra patgoesnė naudoti, kai reikia apskaičiuoti judėjimą kūnų ant Žemės paviršiaus. Paveiksle (1 pav.) taip pat pavaizduota sukimosi kryptis aplink *Z* ašį, nukreiptą link šiaurės ašigalio. Tai geocentrinės koordinačių sistemos teigiamo posūkio kryptis – pasisukimas bus teigiamas, jei *XY* plokštuma suksis pagal laikrodžio rodyklę, stebint iš Žemės centro link šiaurės ašigalio.

1.2. Kampinė padėtis

Kampinei padėčiai nustatyti yra keletas būdų: krypties kosinusų matrica (DCM), Eulerio kampai ir kvaternionai.

Tam tikrus vektorius kaip greičiai, pagreičiai ar transliacijos vektoriai gali būti transformuoti tarp dviejų atskaitos sistemų, naudojant 3x3 krypties kosinusų matricą [4]. Turint tam tikrą vektorių Q_b INS jutiklio atskaitos sistemoje, į betkokią *n* atskaitos sistemą jį galimą pasukti skaliarine sandauga iš krypties kosinusų matricos:

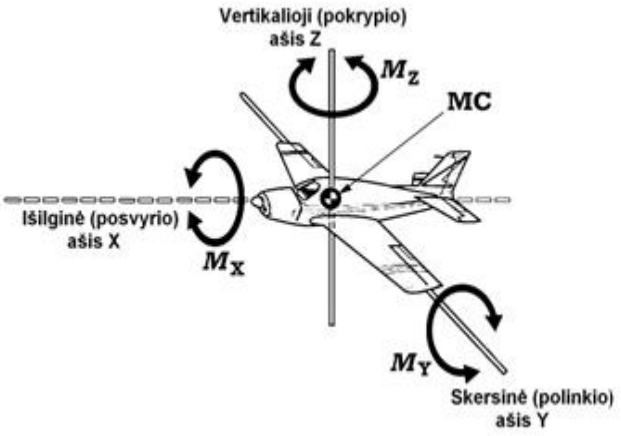
$$Q_b C = \begin{bmatrix} Q_x \\ Q_y \\ Q_z \end{bmatrix} \begin{bmatrix} c_{xx} & c_{xy} & c_{xz} \\ c_{yx} & c_{yy} & c_{yz} \\ c_{zx} & c_{zy} & c_{zz} \end{bmatrix} \tag{1}$$

Krypties kosinusų matrica, kuri iš esmės yra sukimo matrica, nebūtinai yra simetriška. Matricos stulpeliai nusako jutiklio atskaitos sistemos ašių transformaciją į pasirinktą sistemą, o eilutės – atskaitos sistemos ašių transformaciją į jutiklio atskaitos sistemą. Taigi, kiekvienas krypties kosinusų matricos komponentas yra kosinusas kampo tarp jutiklio ir jutiklio ir pasirinktosios atskaitos sistemų. Visi vektoriai (matricos stulpeliai) yra ortogonalūs tarpusavyje ir jų ilgiai lygūs vienetui. Pagal (1), paveikus vektorių krypties kosinusų matrica, jo komponentai bus lygūs:

$$\begin{aligned} Q_{nx} &= r_{xx}Q_{bx} + r_{xy}Q_{by} + r_{xz}Q_{bz} \\ Q_{ny} &= r_{yx}Q_{bx} + r_{yy}Q_{by} + r_{yz}Q_{bz} \\ Q_{nz} &= r_{zx}Q_{bx} + r_{zy}Q_{by} + r_{zz}Q_{bz} \end{aligned} \tag{2}$$

Iš (1) gali atrodyti, kad krypties kosinusų matrica turi devynis nepriklausomus komponentus. Iš tiesų tėra trys nepriklausomi komponentai dėl šešių ortogonalumo ir normalizacijos sąlygų: visi vektoriai, aprašomi stulepiais matricoje, yra taprusavyje statmeni ir jų visų ilgis lygus vienetui. Sukimo matrica, skirta transformuoti vektorius iš vienos koordinačių sistemos į kitą nebūtinai turi būti sudaryta iš krypties kosinusų.

Dar plačiai naudojami Eulerio kampai. Šie kampai yra Eulerio rotacijos teoremos pasėkmė: jei standus kūnas, turintis fiksuotą tašką patiria betkokį poslinkį, tas poslinkis lygus sukimui aplink ašį, einančią per tą fiksuotą tašką. Tai reiškia, kad dvi pasukimai taip pat sudaro posūkį.



2 pav. Eulerio kampai.

Kaip matoma (2 pav.) yra trys Eulerio kampai [7], kuriais vienareikšmiškai apibrėžiama kūno kampinė padėtis (orientacija). Kampas tarp išilginės orlaivio ašies ir horizonto vadinamas polinkio kampu (ang. Pitch) ir yra žymimas graikiška raide $\boldsymbol{\theta}$. Šis kampas teigiamas, kai išilginė ašis nukreipta į viršų nuo horizontalios plokštumos ir neigiamas, kai išilginė ašis nukreipta žemyn horizonto atžvilgiu. Polinkio kampas nurodo orlaivio kampinę padėti horizonto atžvilgiu, bet ne jo judėjimo kryptį. Kampas tarp skersinės orlaivio ašies ir horizonto vadinasmas posvyrio kampu (ang. Roll). Jis teigiamas, kai, žiūrint iš *Z* koordinačių sistemos pradžios, krypsta pagal laikrodžio rodyklę ir neigiamas, kai krypsta prieš laikrodžio rodyklę. Posvyrio kampas žymimas graikiška raide $\boldsymbol{\phi}$. Galiausiai, posūkio kampas apie vertikalią ašį vadinamas pokrypiu (ang. Yaw). Žymimas graikiška raide $\boldsymbol{\psi}$.

Kaip ir kypties kosinusų matricos atveju, naudojant Euerio kampus taip pat sudaromos sukimo matricos transformuoti vektoriams iš vienos atskaitos sistemos į kitą, Vienos koordinačių sistemos transformaciją į kitą galima atlikti trimis pasukimais apie kiekvieną koordinačių sistemos ašį, pavyzdžiui, pilnam sukimui atlikti reikia koordinačių sistemą pasukti kampu ψ aplink z ašį, kampu θ aplink y ašį ir kampu φ apie x ašį. Šis būdas aprašyti sukimus patogus, nes tokius duomenis teiktų jutikliai, įtaisyti tarp trijų ortogonalių giroskopų laisvai besisukančioje inercinėje navigacijos sistemoje. Matematiškai sukimą aplink x, y ir z ašis Eulerio kampais galima aprašyti trijomis sukimo matricomis:

$$C_1 = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$C_2 = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{4}$$

$$C_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{5}$$

kur C_1 aprašo posūkį kampu ψ apie z ašį, C_2 – posūkį kampu θ apie y ašį ir C_3 – posūkį kampu ϕ apie x ašį. Antrasis sukimas kitoks, nei likę du, nes gali vykti tik rėžiuose $\theta \in [0, \pi]$. Atskaitos koordinačių sistemos transformacija į kūno koordinačių sistemą gali būti išreiškiama kaip (3), (4) ir (5) matricų sandauga:

$$C_n^b = C_3 C_2 C_1 \tag{6}$$

Taip pat galima atlikti ir atvirkščią transformaciją – iš kūno koordinačių sistemos į atskaitos koordinačių sistemą:

$$C_b^n = C_n^{bT} = C_1^T C_2^T C_3^T \tag{7}$$

Galutinė krypties kosinusų matrica, išreikšta Eulerio kampais:

$$C_b^n = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \quad (8)$$

Pagrindinė problema, orientacijos atnaujinimui naudojant Eulerio kampus – kardaninis užsirakinimas. Tai reiškinys, kai kūno judėjimo metu dvi iš trijų giroskopo veikimo plokštumų pasidaro lygiagrečios ir prarandamas vienas laisvės laipsnis, pakeičiant sistemos veikimą trimatėje erdvėje į veikimą dvimatėje, vykstantis dėl posūkio kampu θ, kuri vyksta rėžiuose $\theta \in [0, \pi]$. Patogesnis metodas apibūdinti posūkiams, kuris neturi šios problemos – kvaternionai. Tai skaičių sistema, pirmą kartą paminėta William Rowan Hamilton [1]. Kvaternionai yra kompleksinių skaičių papildymas – tai skaičius, turintis vieną realią dalį ir tris menamas dalis. Viena pagrindinių kvaternionų savybių – menamų dalių kvadratai ir menamų dalių tarpusiavio sandauga yra lygios -1:

$$q = r + xi + yj + zk;\ i^2 = j^2 = k^2 = ijk = -1 \quad (9)$$

Kvaternionus galima naudoti apibūdinti posūkiams, nes jie iliustruoja ašies-kampo reprezentaciją – sukimas trimatėje erdvėje gali būti apibūdinamas dviem dydžiais – vienetiniu vektoriumi, nusakančiu sukimo ašį ir kampu θ, nusakančiu pasukimo dydį apie tą ašį. Vienetinis kvaternionas turi išraišką:

$$q = \begin{bmatrix} q_w q_x q_y q_z \end{bmatrix}^T \quad (10)$$

kur:

$$q_w = \cos\left(\frac{\alpha}{2}\right) \quad (11)$$

$$q_x = \sin\left(\frac{\alpha}{2}\right)\cos(\beta_x) \quad (12)$$

$$q_y = \sin\left(\frac{\alpha}{2}\right)\cos(\beta_y) \quad (13)$$

$$q_z = \sin\left(\frac{\alpha}{2}\right)\cos(\beta_z) \quad (14)$$

čia: α – posūkio kampas, (rad); $\cos(\beta_x)$, $\cos(\beta_y)$ ir $\cos(\beta_z)$ – krypties kosinusai kampų tarp sukimosi ašies ir kiekvienos koordinačių ašies atitinkmai. Tada ortogonalią sukimo matricą galima išreikšti per kvaternijonus tokiu būdu [1]:

$$R = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_w q_y + q_x q_z) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 - q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_w q_x + q_y q_z) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \quad (15)$$

1.3. INS akselerometrų matuojamų dydžių interpretacija

Inercinės navigacijos sistemos gali būti laisvai besisukančios (plaukiančios) arba fiksuotos (strapdown). Pagrindinis skirtumas tarp jų – fiksuotos sistemos jutikliai yra įtaisyti nejudamai, o laisvai besisukančios sistemos jutikliai įtaisyti giroskopais stabilizuotoje platformoje, ir skrydžio metu išlaiko savo padėtį inercinių ašių atžvilgiu. Išmatuoti platformos posūkio kampai kūno atžvilgiu naudojami kampinei kūno padėčiai nustatyti, o akselerometrų matavimo rezultatų nebereikia transformuoti į inercinę atskaitos sistemą: jie būna sutapdinti su KS ašimis ir naudojami tiesiogiai. Šiame darbe yra modeliuojama fiksuota inercinės navigacijos sistema. Kaip minėta, fiksuota INS turi tris tarpusavyje statmenus akselerometrus ir giroskopus. Kadangi dėl mikroelektronikos pažangos, visa INS tėra kelių mm dydžio, dažniausiai įrenginiuose tvirtinamos nejudamai įrenginio atžvilgiu [6]. Jie taip pat turi jundačias dalis, bet dėl jų konstrukcijos specifikos, veikimui nebūtina atskira platforma. Akselerometrai fiksuoja specifinę jėgą (jėgą, kurią „jaučia“ akselerometras), veikiančią judantį kūną, trijose kūno koordinačių sistemos ašyse [2]:

$$\boldsymbol{f} = \frac{d^2\boldsymbol{r}}{dt^2} - \boldsymbol{g} \quad (16)$$

čia: $\boldsymbol{f}$ – specifinė jėga, (N); $\boldsymbol{r}$ – koordinatė; $\boldsymbol{g}$ – gravitacijos vektoriaus dedamoji, (m/s^2).

Reikia paminėti, kad akselerometras negeba atskirti judėjimo su pagreičiu $\frac{d^2\boldsymbol{r}}{dt^2}$ nuo gravitacijos jėgos pagreičio $\boldsymbol{g}$**.** Kad tai atlikti, akselerometro išmatuotas pagreitis turi būti transformuotas iš su kūnu susietos koordinačių sistemos į pasirinktą KS. Tai galima atlikti kosinusų matrica paveikus specifinės jėgos vektorių (2):

$$f_x^i = c_{11} f_{bx} + c_{12} f_{by} + c_{13} f_{bz} \quad (17)$$

$$f_y^i = c_{21}f_{bx} + c_{22}f_{by} + c_{23}f_{bz} \tag{18}$$

$$f_z^i = c_{31}f_{bx} + c_{32}f_{by} + c_{33}f_{bz} \tag{19}$$

Kadangi kūnas juda, jo padėtis Žemės centro atžvilgiu kinta, taigi gravitacijos vektoriaus dedamoji taip pat nėra pastovi:

$$\boldsymbol{g}_x = -\frac{xgR^2}{l^3} \tag{20}$$

$$\boldsymbol{g}_y = -\frac{ygR^2}{l^3} \tag{21}$$

$$\boldsymbol{g}_z = -\frac{(z-R)gR^2}{l^3} \tag{22}$$

čia: $\boldsymbol{g}_i$ – gravitacijos vektoriaus dedamoji i ašyje, (m/s^2); g – gravitacijos konstanta, (m/s^2); R – Žemės spindulys, (m); l – vektoriaus ilgis, (m).

$$l = \sqrt{(x^2 + y^2 + (z-R)^2)} \tag{23}$$

Iš (16) matoma, jog norint iš INS akselerometrų duomenų gauti judėjimo pagreitį, reikia iš specifinės jėgos atimti gravitacijos vektoriaus dedamąją.

1.4. INS matuojamų kampinių greičių ir tiesinių pagreičių integravimas

Orientacijos uždavinys įgyvendinamas „dead reckoning“ metodu – sistemai užduodama pradinė padėtis ir pradinė orientacija. Iš pasinaudojus išmatuotais kinematinais dydžiais (pagreičiais, judėjimo

greičiais) apskaičiuojama nauja padėtis ir greitis sekančią laiko akimitrką pridedant apskaičiuotą pokytį prie senųjų verčių.

Tam, kad krypties kosinusų matrica pastoviai tiksliai nusakytų kūno orientaciją, ji vis atnaujinama laiko atžvilgiu. Krypties kosinusų matricos kitimo greitis gali būti nusakomas taip [2]:

$$\dot{C_b^n} = \lim_{\delta t \to 0} \frac{\delta C_b^n}{\delta t} = \lim_{\delta t \to 0} \frac{C_b^n(t+\delta t) - C_b^n(t)}{\delta t} \tag{26}$$

čia: $C_b^n(t)$ ir $C_b^n(t+\delta t)$ – krypties kosinusų matricos laiko momentais t ir $t+\delta t$. $C_b^n(t+\delta t)$ dedamoji gali būti perrašyta kaip:

$$C_b^n(t+\delta t) = C_b^n(t)A(t) \tag{27}$$

čia: $A(t)$ – krypties kosinusų matricas laiko momentais t ir laiko momentu $t+\delta t$ susiejanti matrica. Pasukimams nedideliais kampais $A(t)$ gali būti perrašoma kaip:

$$A(t) = [\mathrm{I} + \delta\Psi] \tag{28}$$

kur I – vienetinė matrica, o:

$$\delta\Psi = \begin{bmatrix} 0 & -\delta\psi & \delta\theta \\ \delta\psi & 0 & -\delta\phi \\ -\delta\theta & \delta\phi & 0 \end{bmatrix} \tag{29}$$

kur $\delta\psi$, $\delta\theta$ ir $\delta\phi$ yra kūno koordinačių sistemos sukimo pokyčiai išreikšti nedideliais Eulerio kampais apie z, y ir x ašis atitinkamai. Pertvarkius lygtis gaunama, kad:

$$\lim_{\delta t \to 0} \frac{\delta\Psi}{\delta t} = \Omega_{nb}^b \tag{30}$$

To pasekoje:

$$\dot{C_b^n} = C_b^n \Omega_{nb}^b \tag{31}$$

kur

$$\Omega_{nb}^{b} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{32}$$

Naudojant (29) lygtį, integruojamos kampinių greičių vertės išmatuotos INS jutikliais ir kiekvienu laiko momentu atnaujinama kūno kampinė padėtis. Viena iškylanati kliūtis – INS atiduodami duomenys visad yra kūno koordinačių sistemoje, kokias jėgas, pagreičius ar sukimą „jaučia“ pats jutiklis. Taigi, reikalingi metodai kaip šiuos duomenis perkelti į norimą atskaitos sistemą.

Kaip ir kosinusų matricos atveju, kad būtų korektiškai atnaujinama kampinė padėtis, Eulerio kampai taip pat turi būti pastoviai atnaujinami. Eulerio kampų kitimo greitį, jutiklio matuojamus kampinius sukimosi greičius ir sukimo matricas C_1, C_2 ir C_3 sieja išraiška:

$$\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + C_3 \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + C_3 C_2 \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{33}$$

INS giroskopai teikia momentinio kampinio greičio duomenis, kurie turi būti integruojami orientacijai nustatyti. Tam reikia iš gaunamų kampinių greičių su kūnu susijusiose koordinačių sistemoje sukonstruoti krypties kosinusų matricą, kuri nusakytų kūno orientaciją [2]. Žinant orientaciją pradiniu laiko momentu ir naudojantis (31) išreiškiami pokyčiai kiekvienam krypties kosinusų matricos komponentui:

$$\begin{array}{lll} \dot{c_{11}} = c_{12}\omega_z - c_{13}\omega_y & \dot{c_{12}} = c_{13}\omega_x - c_{11}\omega_z & \dot{c_{13}} = c_{11}\omega_y - c_{12}\omega_x \\ \dot{c_{21}} = c_{22}\omega_z - c_{23}\omega_y & \dot{c_{22}} = c_{23}\omega_x - c_{21}\omega_z & \dot{c_{23}} = c_{21}\omega_y - c_{22}\omega_x \\ \dot{c_{31}} = c_{32}\omega_y - c_{33}\omega_y & \dot{c_{32}} = c_{32}\omega_x - c_{31}\omega_z & \dot{c_{33}} = c_{31}\omega_y - c_{32}\omega_x \end{array} \tag{34}$$

Galiausiai, kiekvienu laiko momentu kiekvienas krypties kosinusų matricos komponentas atnaujinamas pagal:

$$c_{ij}(t) = c_{ij}(t - \Delta t) + \dot{c_{ij}}dt \tag{35}$$

čia: $c_{ij}(t)$ – krypties kosinusų matricos elemento vertė laiku *t*, $c_{ij}(t - \Delta t)$ – krypties kosinusų matricos komponento vertė laiko momentu $(t - \Delta t)$, $\dot{c_{ij}}dt$ – krypties kosinusų matricos komponento pokyčio ir laiko žingsnio sandauga

Taip pat, kiekvienu laiko momentu atnaujinus krypties kosinusų matricos narius, projekcijos visose ašyse yra ortogonalizuojamos ir ortonormuojamos [4]. Normalizacija atliekama kiekvienos ašies dedamasias dalinant iš viso vektoriaus ilgio, naudojantis (23). Ortogonalumo nuokrypiai ištaisomi naudojant paklaidą:

$$\varepsilon = XY = X^T Y = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \end{bmatrix} \begin{bmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{bmatrix} \tag{36}$$

Tada paklaida koreguojama *X* ir *Y* ašims:

$$\begin{aligned} X_{ort} &= X - \frac{\varepsilon}{2} Y \\ Y_{ort} &= Y - \frac{\varepsilon}{2} X \end{aligned} \tag{37}$$

Z ašies ortogonalumas ištaisomas naudojant *X* ir *Y* vektorinę sandaugą, kadangi jei *X* ir *Y* yra ortonormuoti, jų vektorinės sandaugos rezultatas – *XY* plokštumai ortonormuotas vektorius.

Siekiant nustatyti taško padėtį ir judėjimo greitį, turi būti atlikti ir dar du integravimo žingsniai: pirmojo metu gaunamas kūno greičio kitimas pasirinktoje KS, INS pagreitį transformuojant iš jutiklio į pasirinktąją KS pasinaudojant 1.3 skyriuje aprašytu būdu, antrojo metu – integruojant greitį gauamas kūno padėties kitimas:

$$v(t) = v_0 + \int_0^t a(t)dt \tag{24}$$

$$r(t) = r_0 + \int_0^t v(t)dt \tag{25}$$

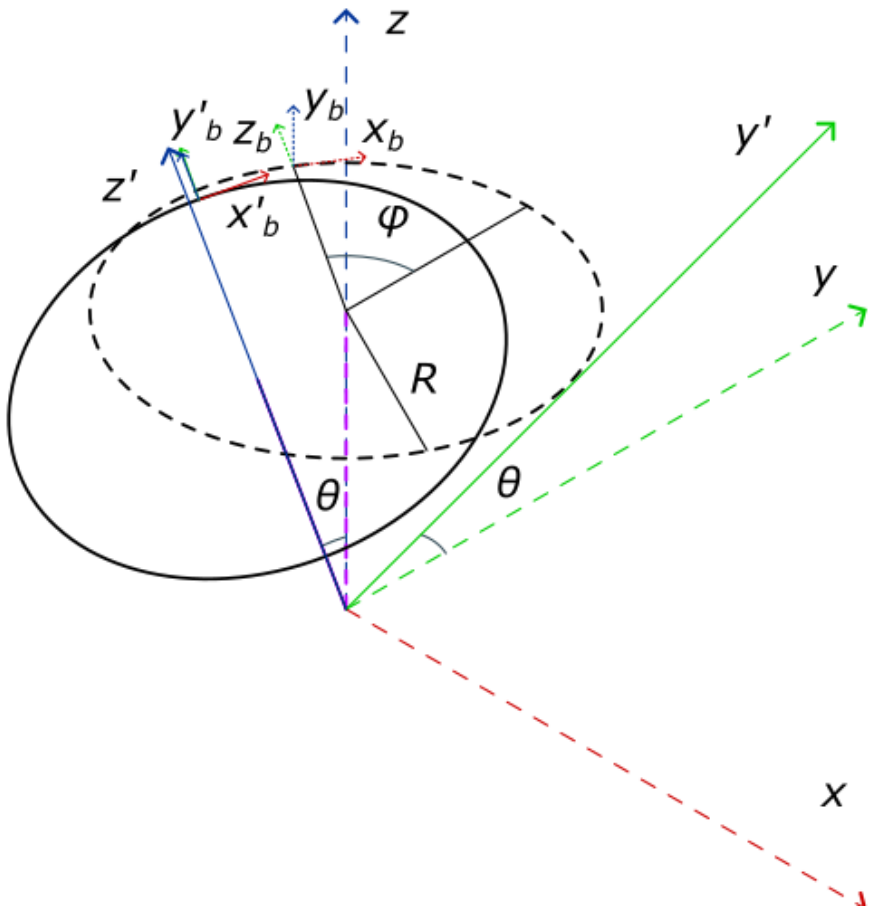
čia: $v(t)$ – greitis, (m/s); v_0 – pradinis greitis, (m/s); a – pagreitis, (m/s^2); $r(t)$ – koordinatė , r_0 –pradinė koordinatė.

2. Matematinis bandymų platformos su pritvirtinta INS modelis

Šiame skyriuje pristatomas INS testavimo platforma, bei matematiškai aprašomas jos kinematinis modelis, paaiškinant pačio modelio judėjimo dėsningumus ir išvedant parametrines pačios INS judėjimo lygtis.

2.1. Matematinio INS testavimo platformos modelio apibrėžimas

Incercinės navigacijos sistemos tyrimui sukurtas prietaiso, skirto tikrinti INS veikimui, matematinis modelis. Žinant teorinius prietaiso judėjimo parametrus, išvestos parametrinės lygtys, nusakančios taško, esančio ant prietaiso, judėjimo trajektoriją. Vėliau, naudojantis sukurta judėjimo trajektorija ir orientacija, apskaičiuojami INS jutikliais matuojami dydžiai, jų kitimo dėsningumai, bandoma atkurti originalią taško judėjimo trajektoriją bei kampinę padėtį.



3 pav. INS tyrimui skirto prietaiso matematinis modelis.

Kaip matoma (3 pav.), apskritiminė platforma sukasi aplink Z' ašį kampuΦ. Ši apskritiminė platforma nutolusi nuo x ašies per h. Besisukanti platforma taip pat siūbuoja apie išilginę ašį (X ašį), kampu Θ. Naudojantis šiuo modeliu analitiškai aprašoma ant platformos įtvirtintos INS erdvinė ir kampinė padėtys. Kaip matoma (3 pav.), matematiniame modelyje taško ant apskritiminės platformos judėjimas susideda iš dviejų dalių: sukimosi ant apskritiminės platformos kampu Φ ir siūbavimo kartu su pagrindinė platforma kampu Θ. Apskritiminės platformos atveju, pasitelkiant polinę koordinačių sistemą, judėjimą galima aprašyti lygčių sistema:

$$\begin{aligned} x &= r\cos(\Phi) \\ y &= r\sin(\Phi) \end{aligned} \tag{38}$$

čia: r – atstumas nuo apskritiminės platformos centro iki jutiklio, (m); Φ – posūkio kampas, (rad); Posūkio kampas gaunamas iš apskritiminės platfromos kampinio greičio:

$$\Phi = \omega_\Phi t = 2\pi f_\Phi t \tag{39}$$

čia: ω_Φ – apskritiminės platformos kampinis sukimosi greitis, (s^{-1}); f_Φ – apskritiminės platformos sukimosi dažnis, (Hz).

Besisukanti platforma svyruoja apie x ašį. Todėl taškas, esantis y-z plokštumoje turi būti pasuktas kampu θ. Tam pasitelkiama dvimatė sukimo matrica:

$$R = \begin{bmatrix} \cos(\Theta) & -h\sin(\Theta) \\ \sin(\Theta) & h\cos(\Theta) \end{bmatrix} \tag{40}$$

čia: Θ – pagrindinės platformos posvyrio kampas x ašies atžvilgiu (rad); h – jutiklio aukštis nuo centrinės siūbavimo ašies, (m). Šis kampas apskaičiuojamas pagal:

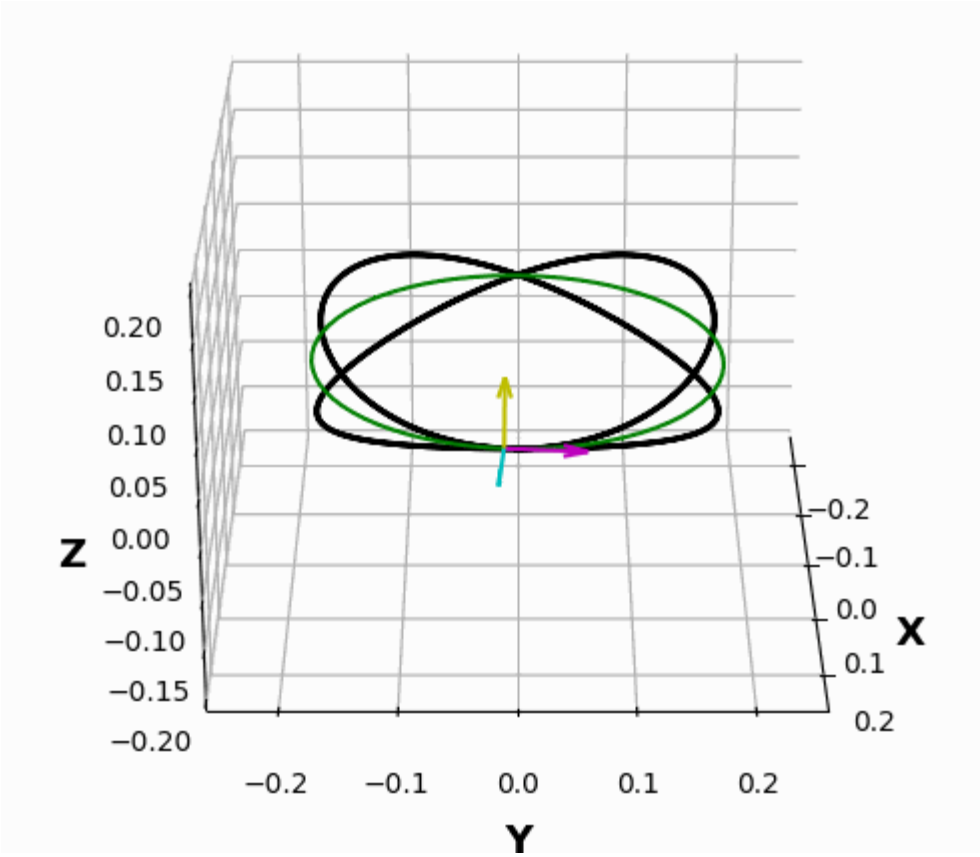
$$\Theta = A\sin(2\pi f_\Theta t) \tag{41}$$

čia: A – pagrindinės platformos siūbavimo amplitudė, (rad); f_θ – pagrindinės platformos siūbavimo dažnis, (Hz). Taško judėjimas platformoje gaunamas atlikus šias transformacijas: pasisuka apskritiminė platforma, o paskui pasvyra pagrindinė platforma. Pasinaudojus anksčiau paminėtomi lygtimis (35) ir (37), galutinė taško judėjimo ant platformos trajektorija aprašoma lygčių sistema:

$$\begin{aligned} x &= r\cos(\Phi) \\ y &= r\sin(\Phi)\cos(\Theta) - h\sin(\Theta) \end{aligned} \tag{42}$$

$$z = r\sin(\Phi)\sin(\Theta) + h\cos(\Theta)$$

Pagal anksčiau apibrėžtas parametrines lygtis, apskaičiuota judančio taško trajektorija, su pradinėmis sąlygomis: r = 18 cm, h = 1 cm, $A = \pi/8$, dt = 0,005s ir kintančiu apskritiminės platformos sukimosi, bei pagrindinės platformos siūbavimo dažnių santykiu $k = \frac{f_\Phi}{f_\Theta}$. Pasirinktas k sanytkis 2.



4 pav. Trajektorijos, gautos iš parametrinių lygčių (42) su parametru k=2 pavyzdys

2.2. INS, įtvirtintos ant testavimo platformos, modelis

Prie platformos pritvirtinta ir su pradinėmis jos ašimis x, y, z sutapdinta INS nejuda besisukančios platformos atžvilgiu. Inercinės sistemos atžvilgiu, jos judėjimą galima išskaidyti į du etapus: judėjimą nepakrypusioje plokštumoje aplink Z' ašį XY plokštumoje ir judėjimą platformai pakrypstant aplink X ašį ZY plokštumoje Užduotis – sudaryti 3x3 matricą, kurios eilutės nurodytų INS ašis, o stulpeliai – INS ašių projekcijas į atskaitos pasirinktą inercinę koordinačių sistemą:

$$C = \begin{bmatrix} x_{\mathrm{bx}} & x_{\mathrm{by}} & x_{\mathrm{bz}} \\ y_{\mathrm{bx}} & y_{\mathrm{by}} & y_{\mathrm{bz}} \\ z_{\mathrm{bx}} & z_{\mathrm{by}} & z_{\mathrm{bz}} \end{bmatrix} \tag{43}$$

INS judant apskritimu kol nepakreipta plokštuma, judėjimas aprašomas sukimo matrica XY plokštumoje. Tariant, kad aplink Z ašį vyksta posūkis kampu α ir remiantis (40), pakoreguojamos dvi pirmos (43) matricos eilutės:

$$C = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ z_b x & z_b y & z_b z \end{bmatrix} \tag{44}$$

Jutiklio ašių x_b ir y_b projekcijos į Z ašį lygios 0 šio sukimo metu. Kitas žingsnis – pasukti ašis aplink X ašį YZ plokštumoje kampu β. Taikant (40), tik pakeičiant ženklus, gaunama sukimo matrica apie X ašį:

$$R = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \tag{45}$$

Kadangi posvyris vyksta apie X ašį, kinta tik Y ir Z ašių komponentai. Sudauginus atitinkamus (44) ir(45) komponentus, su INS susietos koordinačių sistemos ašių projekcijos užrašomos stulpeliu-vektoriumi:

$$x_b = \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha)\cos(\beta) \\ \sin(\alpha)\sin(\beta) \end{bmatrix}, \tag{46}$$

$$y_b = \begin{bmatrix} -\sin(\alpha) \\ \cos(\alpha)\cos(\beta) \\ \cos(\alpha)\sin(\beta) \end{bmatrix} \tag{47}$$

$$z_b = \begin{bmatrix} 0 \\ -\sin(\beta) \\ \cos(\beta) \end{bmatrix} \tag{48}$$

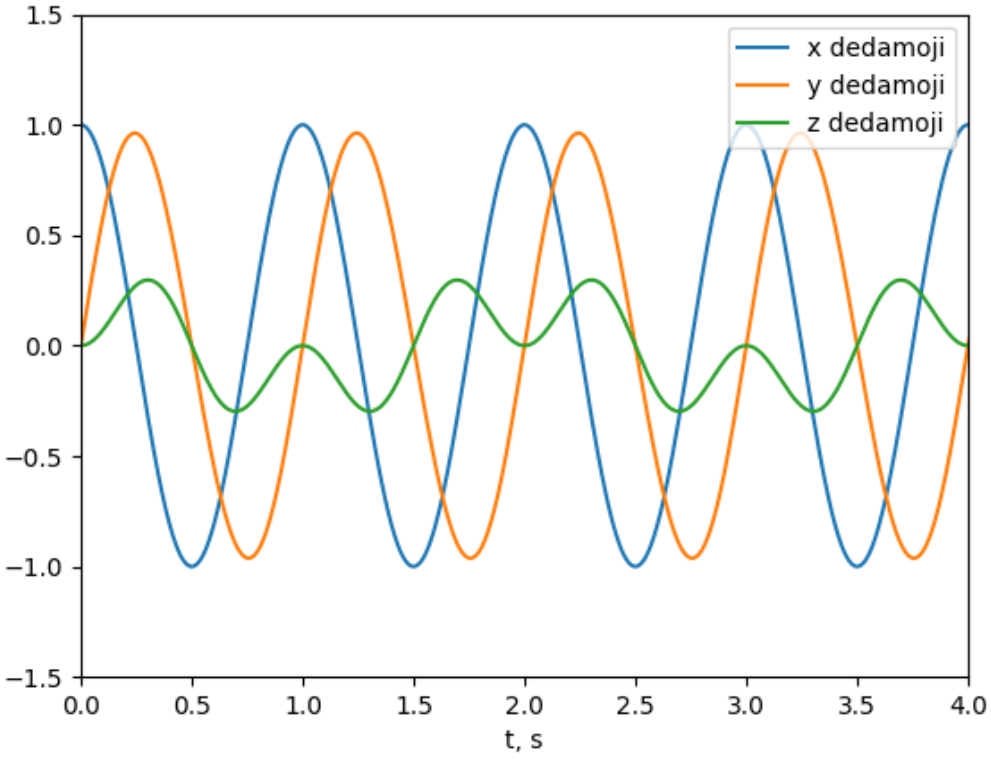
Tada užpildyta (43) matrica įgauna šį pavidalą:

$$C = \begin{bmatrix} \cos(\alpha) & \sin(\alpha)\cos(\beta) & \sin(\alpha)\sin(\beta) \\ -\sin(\alpha) & \cos(\alpha)\cos(\beta) & \cos(\alpha)\sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix} \tag{49}$$

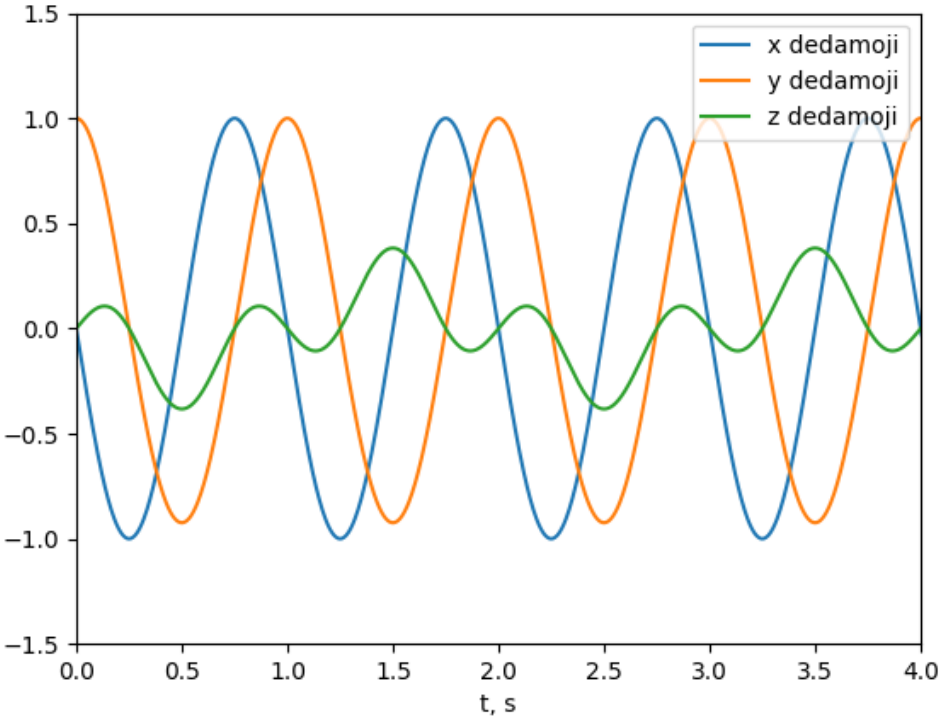
Iš parametrinių judėjimo lygčių, remiantis (49), sukurta krypties kosinusų matrica:

$$C = \begin{bmatrix} \cos(\varphi) & \sin(\varphi)\cos(\theta) & \sin(\varphi)\sin(\theta) \\ -\sin(\varphi) & \cos(\varphi)\cos(\theta) & \cos(\varphi)\sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{52}$$

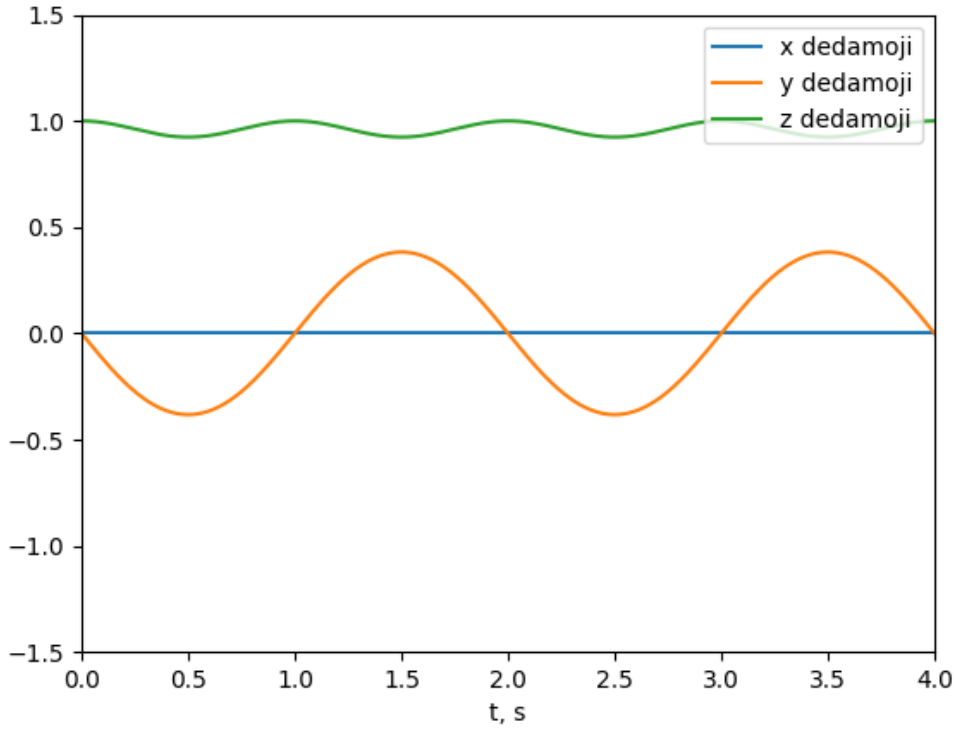
Apskaičiavus pagal parametrines judėjimo lygtis sukonstruotos krypties kosinusų matricos narius pagal pradines sąlygas, įvardytas 2.1 skyriuje, grafiškai pavaizduota su jutikliu susijusios koordinačių sistemos projekcijų į inercinę koordiačių sistemą kosinusų kitimas laike:



5 pav. INS Xb koordinačių kitimas laike: mėlyna spalva žymi projekciją į x ašį, oranžinė žymi projekciją į y ašį ir žalia – projekciją į z ašį.



6 pav. INS Yb koordinačių kitimas laike: mėlyna spalva žymi projekciją į x ašį, oranžinė žymi projekciją į y ašį ir žalia – projekciją į z ašį.



7 pav. INS Zb koordinačių kitimas laike: mėlyna spalva žymi projekciją į x ašį, oranžinė žymi projekciją į y ašį ir žalia – projekciją į z ašį.

Šia (45) krypties kosinusų matrica pasinaudota apskaičiuoti atitinkamus Eulerio kampus, išreiškiančius INS orientaciją, o iš pastarųjų – kampinius greičius, kuriuos matuotų taip įtvirtintos INS giroskopai.

$$\phi = \arctan2\left[\frac{c_{32}}{c_{33}}\right] \tag{53}$$

$$\theta = \arcsin[-c_{31}] \tag{54}$$

$$\psi = \arctan2\left[\frac{c_{21}}{c_{11}}\right] \tag{55}$$

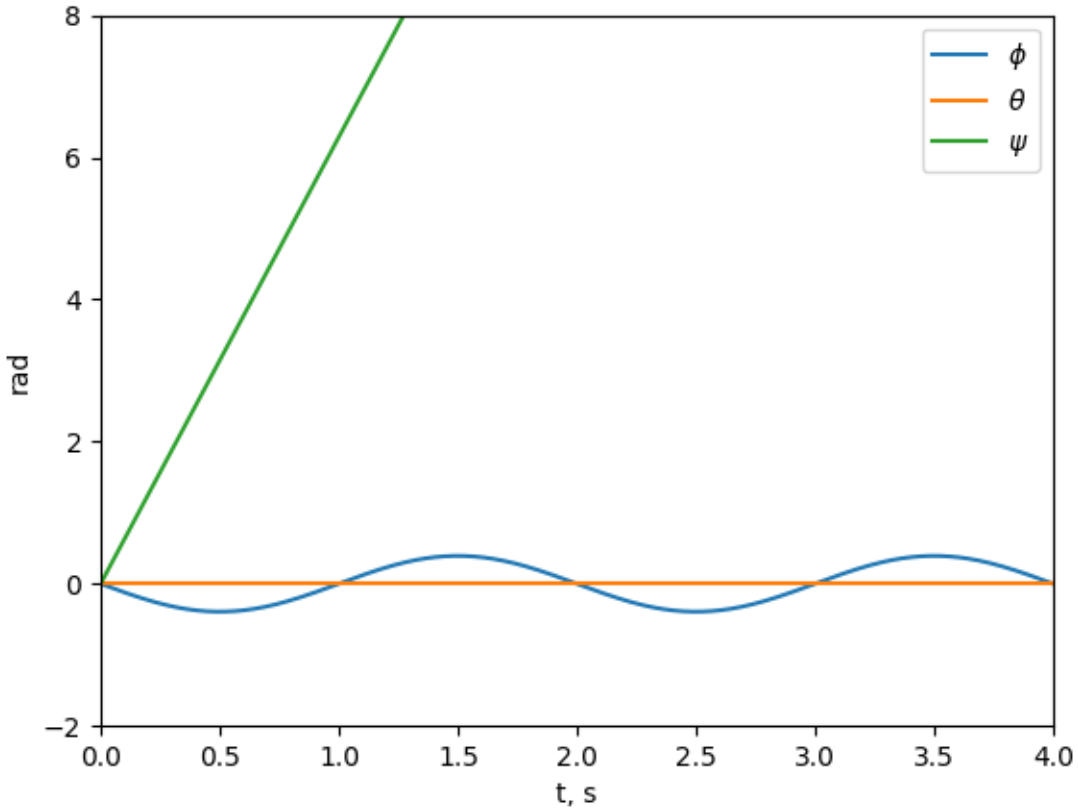
čia: ϕ – sukimosi kampas apie *X* ašį, (rad); θ – sukimosi kampas apie *Y* ašį, (rad); ψ – sukimosi kampas apie *Z* ašį, (rad). Įrašius atitinkamas iš parametrinių lygčių sudarytos krypties kosinusų matricos komponentų vertes, galutinė analizinė Eulerio kampų išraiška tampa:

$$\phi = \arctan2[\tan(\Theta)] = \Theta \tag{56}$$

$$\theta = \arcsin[0] = 0 \tag{57}$$

$$\psi = \arctan2[\tan(\Phi)] = \Phi \tag{58}$$

Naudojant minėtas pradines sąlygas, grafiškai atvaizduotas Eulerio kampų kitimas laike, pagal (49), (50) ir (51),:



8 pav. Eulerio kampų kitimo nuo laiko priklausomybė

Toliau, naudojantis (33) išvedamos lygtys kampiniams greičiams apskaičiuoti. Įstačius pradines sąlygas, gaunamos analitinės INS matuojamų kampinių greičių išraiškos:

$$\omega_x = -2\pi f_\theta \cos(2\pi f_\theta t) A \tag{59}$$

$$\omega_y = -2\pi \sin(A \sin(2\pi f_\theta t)) f_\varphi \tag{60}$$

$$\omega_z = 2\pi \cos(A \sin(2\pi f_\theta t)) f_\varphi \tag{61}$$

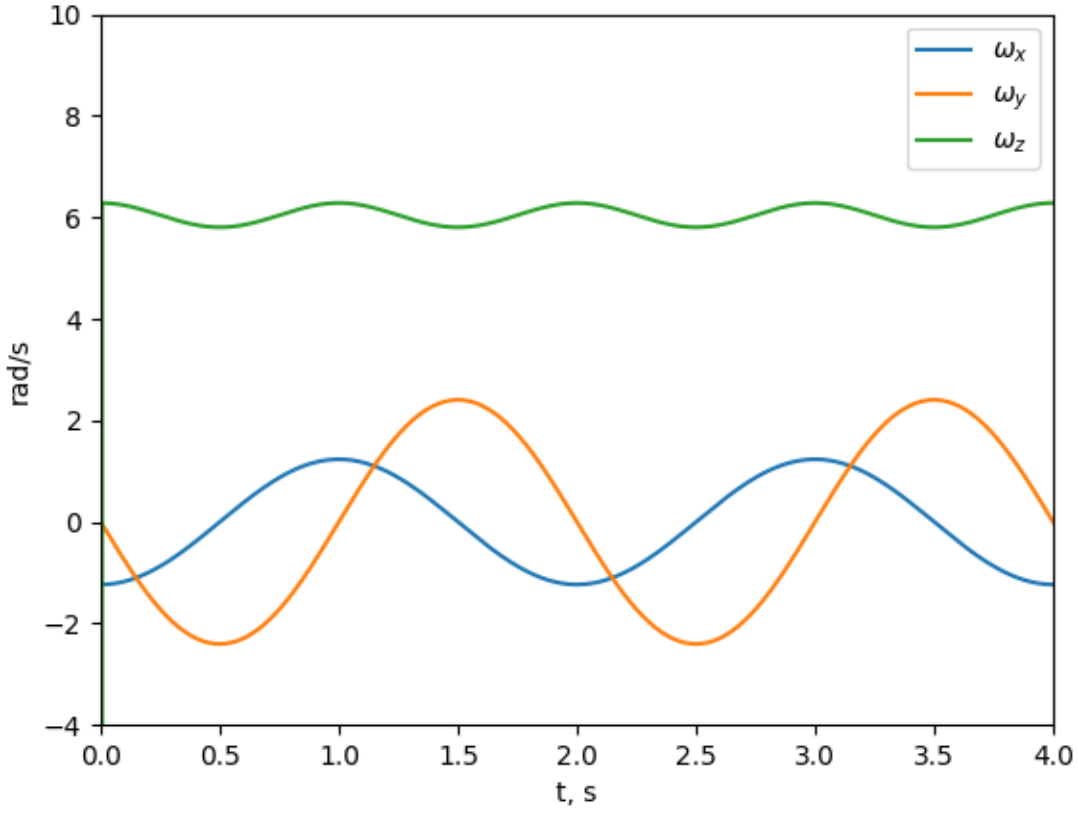
Naudojantis (52)-(54) ir pradinėmis sąlygomis: r = 0,18m, h = 0,01m, dt = 0,005s, k = 2, apskaičiuoti INS giroskopų matuojami kampiniai greičiai:

$$\omega_x = -\left(\frac{\pi^2}{2}\right) \cos(3\pi t) \tag{62}$$

$$\omega_y = -\pi \sin\left(\frac{\pi}{6} \sin(3\pi t)\right) \tag{63}$$

$$\omega_z = \pi \cos\left(\frac{\pi}{6} \sin(3\pi t)\right) \tag{64}$$

Gautos kampinių greičių vertės pavaizduojamos grafiškai:



9 pav. Kampinių greičių priklausomybė nuo laiko

Iš parametrinių taško judėjimo lygčių galima apskaičiuoti INS duomenis – tiesinius pagreičius ir kampinius greičius. Iš pradžių apskaičiuojamas taško greitis ir pagreitis:

$$v(t) = \frac{r(t + \Delta t) - r(t - \Delta t)}{2\Delta t} \tag{65}$$

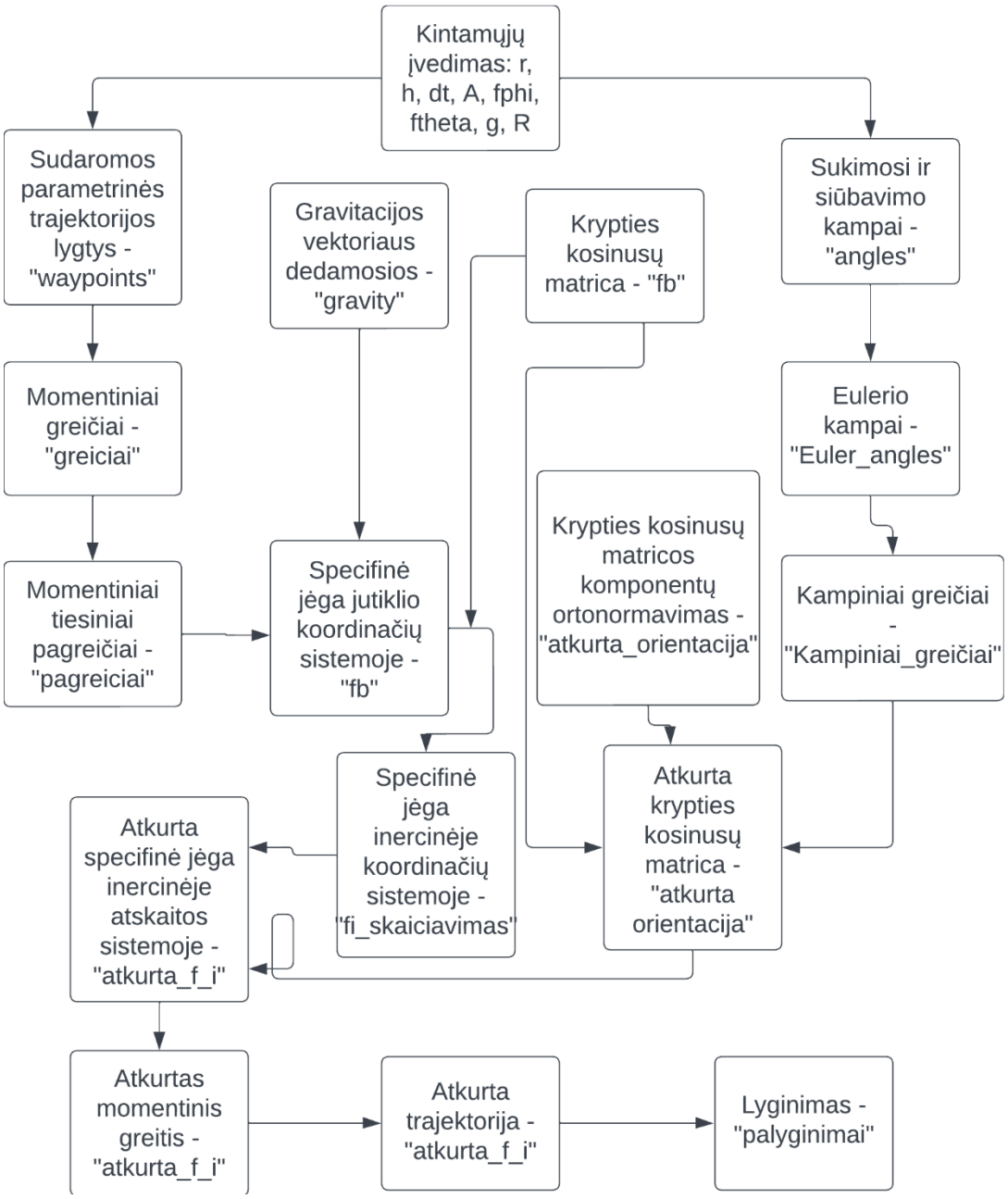
$$a(t) = \frac{v(t + \Delta t) - v(t - \Delta t)}{2\Delta t} \tag{66}$$

čia: $v(t)$ – momentinis greitis, (m/s); $r(t + \Delta t)$ – koordinatė laiko momentu $t + \Delta t$, $r(t - \Delta t)$ – koordinatė laiko momentu $t - \Delta t$, $a(t)$ – momentinis pagreitis, (m/s^2); $v(t + \Delta t)$ – greitis laiko momentu $t + \Delta t$, (m/s); $v(t - \Delta t)$ – greitis laiko momentu $t - \Delta t$, (m/s).

Remiantis (65) ir (66) ir pradinėmis sąlygomis, aprašytomis 2.1 skyriuje, grafiškai pavaizduojami iš parametrinių lygčių apskaičiuoti INS matuojami pagreičiai ir iš jų apskaičiuoti greičiai.

3. Programos aprašas

Šiame skyriuje pristatoma ir paaiškinama Python programavimo kalba parašyta programa, skirta INS modeliui aprašyti ir tyrimui atlikti (žr. Priedai).

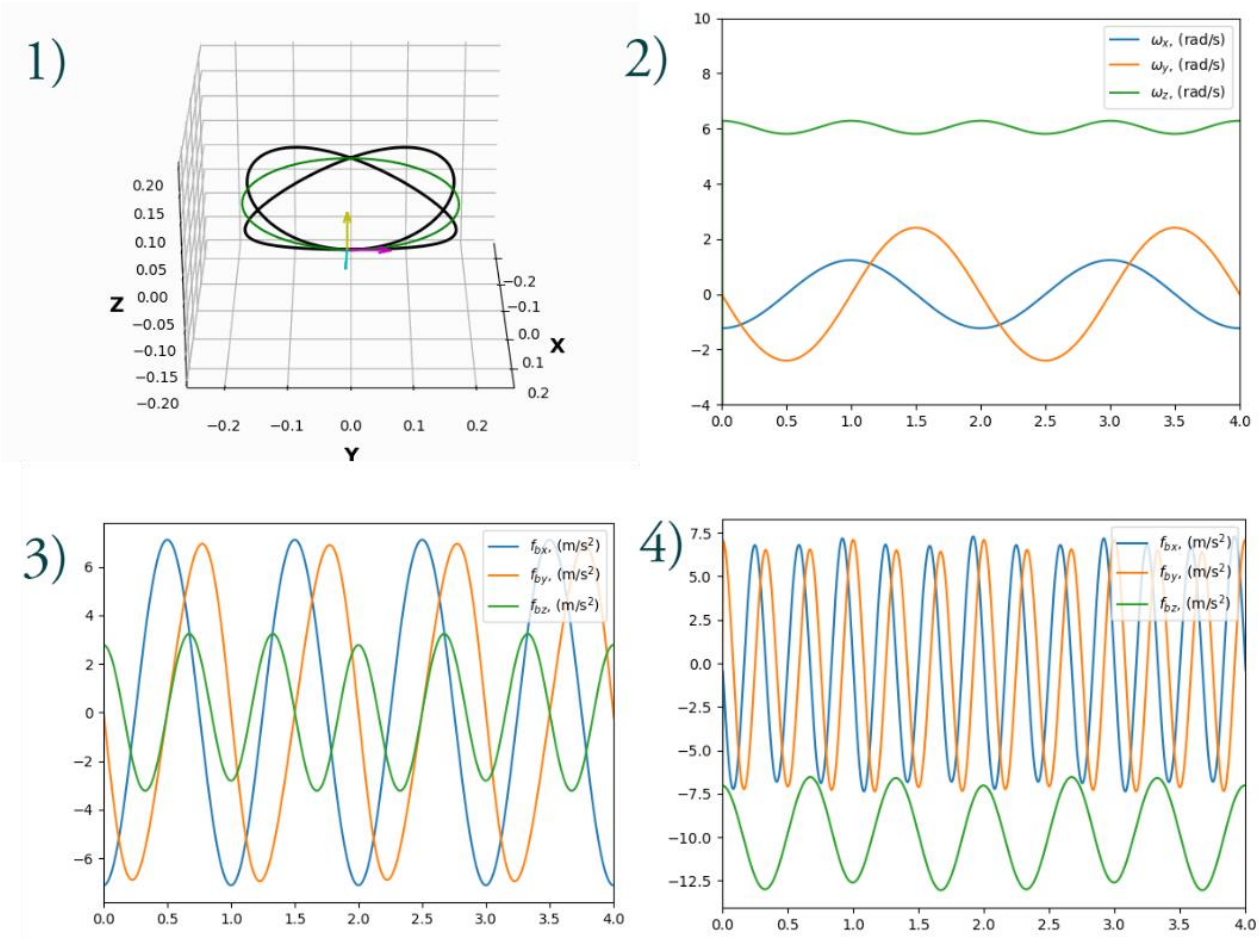


10 pav. Programos, skirtos aprašyti INS modelį ir tirti jo veikimą, diagrama

Iš pradžių programoje įvedami kintamieji – INS modelio parametrai, kurie bus naudojami tolimesniems skaičiavimams. Sudaromi φ ir θ kampų masyvai pagal (39) ir (40) atitinkamai. Pagal (38), (40) ir (42) sudaromos parametrinės INS modelio judėjimo lygtys, pagal kurias vėliau sudaroma ir krypties kosinusų matrica. Naudojantis Φ ir Θ kampais, apskaičiuojami Eulerio kampai: posvyrio kampas (ang. Roll) pagal (56), polinkio kampas (ang. Pitch) pagal (57) ir pokrypio kampas (ang. Yaw) pagal (58). Iš Eulerio kampų apskaičiuojami kampiniai greičiai apie *X*, *Y*, ir *Z* ašis, atitinkamai pagal (59), (60) ir (61). Iš parametrinių lygčių apskaičiavus koordinates, pagal (65) apskaičiuojamas momentinis greitis kiekvienoje su jutikliu susietos koordinačių sistemos ašyje ir taip pat, pagal (66), momentiniai pagreičiai. Kadangi jutiklio atžvilgiu matuojamas momentinis pagreitis yra neatskiriamas nuo gravitacijos sukeliamo pagreičio, kiekvienai su jutikliu susijusiai koordinačių ašiai apskaičiuojama gravitacijos vektoriaus dedamoji: *X* ašiai pagal (20), *Y* ašiai pagal (21), *Z* ašiai pagal (22). Iš momentinių pagreičių atėmus gravitacijos vektoriaus dedamąsias, gaunama specifinė jėga jutiklio koordinačių sistemoje, apskaičiuojama pagal (16) ir transformuojama į atskaitos koordinačių sistemą pagal (17), (18) ir (19). Iš paramterinių lygčių (42) sudėliojama krypties kosinusų matrica. Iki šio žingsnio programa vykdė skaičiavimus, su tikslu imituoti idealios INS matuojamus dydžius. Toliau programa iš šių dydžių atkuria pradinę taško judėjimo trajektoriją bei kampinę padėtį. Iš pradžių atkuriama krypties kosinusų matrica. Kiekvieno komponento pradine verte priskiriama tokia pat kaip ir krypties kosinusų matricos, sudėtos iš parametrinių lygčių. Pagal (34) apskaičiuojamas kiekvieno komponento pokytis per laiko žingsnį ir pagal (35) tas pokytis dauginamas iš laiko žingsnio ir pridedamas prie senos vertės. Siekiant mažinti skaitinio metodo netikslumus, kiekviename laiko žingsnyje matricos komponentai yra normuojami ir atstatomas jų ortogonalumas atitinkamai pagal (36) ir (37). Galiausiai pradedama atnaujinti trajektorija. Kaip ir atkuriamos krypties kosinusų matricos atveju, atkuriamiems dydžiams priskiriamos pradinės vertės: momentinių greičių atveju pradinės vertės lygios iš parametrinių lygčių apskaičiuotų momentinių greičių pradinėms vertėms. Koordinačių pradinės vertės $x(0) = r$, $y(0) = 0$, ir $z(0) = h$. Atkūrimas prasideda nuo specifinės jėgos inercinėje atskaitos sistemoje atkūrimo. *X* ašyje pagal (17), *Y* ašyje pagal (18) ir *Z* ašyje pagal (19). Specifinė jėga integruojama pagal (24) ir gaunamas momentinis greitis, kuris darkart integruojamas ir gaunamos koordinatės. Galiausiai, programa palygina specifines jėgas, momentinius greičius ir koordinates atkurtas algoritmo ir apskaičiuotas iš parametrinių lygčių.

4. Rezultatai

Pasirinktos pradinės sąlygos: A = π/8, r = 0,18m, h = 0,01m, dt = 0,005s, k = 2. Gauti pagreičiai ir kampiniai greičiai atvaizduojami grafiškai



12 pav. INS modelio sintetinamų duomenų rezultatai, kai A = π/8: 1) taško nubrėžta trajektorija; 2) Kampinių greičių priklausomybės nuo laiko; 3) specifinės jėgos kūno atskaitos sistemoje priklausomybė nuo laiko; 4) specifinės jėgos inercinėje atskaitos sistemoje priklausomybė nuo laiko..

Su šiais duomenimis iš esmės buvo sumodeliuota sintetinė inercinė navigacijos sistema, kuri, keičiant pradinius parametrus pateikia kampinių greičių ir tiesinių pagreičių vertes. Sekanti darbo užduotis – vykdyti atvirkštinį navigacijos uždavinį – atstatyti taško trajektoriją ir kampinę padėtį. Kaip jau aptarta, šiuos duomenis teks integruoti. Iš pradžių atkuriama krypties kosinusų matrica pagal (32), pradinėms vertėms naudojant tas pačias vertes kaip pradinės vertės iš parametrinių lygčių sudarytoje

krypties kosinusų matricoje. Pasirinkta, kad platforma nesvyruotų, o vyktų tik sukimasis apskritimine platforma. Naudojantis specifinės jėgos ir gravitacijos vektoriaus dedamosios skirtumu – pagreičiu, pagal (22), (23) ir (24) formules atkuriama specifinė jėga inercinėje atskaitos koordinačių sistemoje. Prie gautų specifinės jėgos dedamųjų pridėjus gravitacijos vektoriaus dedamąsias, integruojamos tiesinio greičio dedamosios:

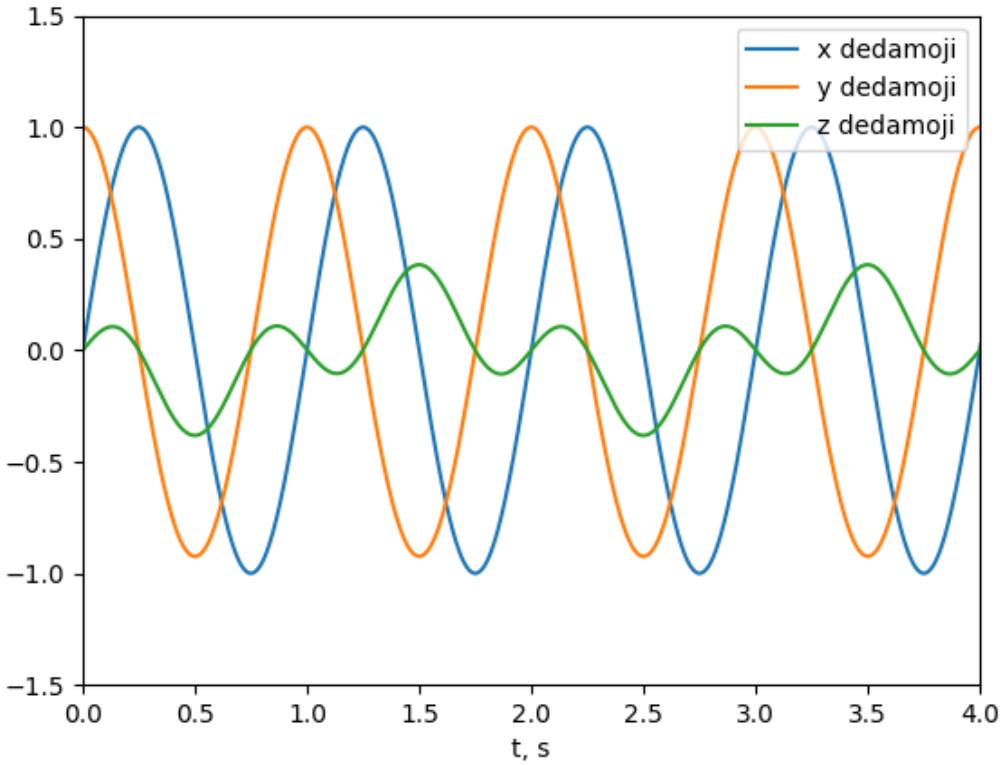
$$v_x = v_x(0) + (f_{ix} + g_x)dt \tag{67}$$

čia: v_x – momentinis greitis X ašyje, (m/s); $v_x(0)$ – momentinis greitis X ašyje praeitu laiko momentu, (m/s); f_{ix} – specifinė jėga inercinėje koordinačių sistemoje, (N); g_x – gravitacijos vektoriaus dedamoji X ašyje, (m/s^2); dt – laiko žingsnis, (s). Tokiu pačiu metodu tiesinio greičio dedamosios apskaičiuojamos ir kitoms ašims. Tam, kad gauti koordinatę, greitis dar kartą integruojamas, naudojantis (30):

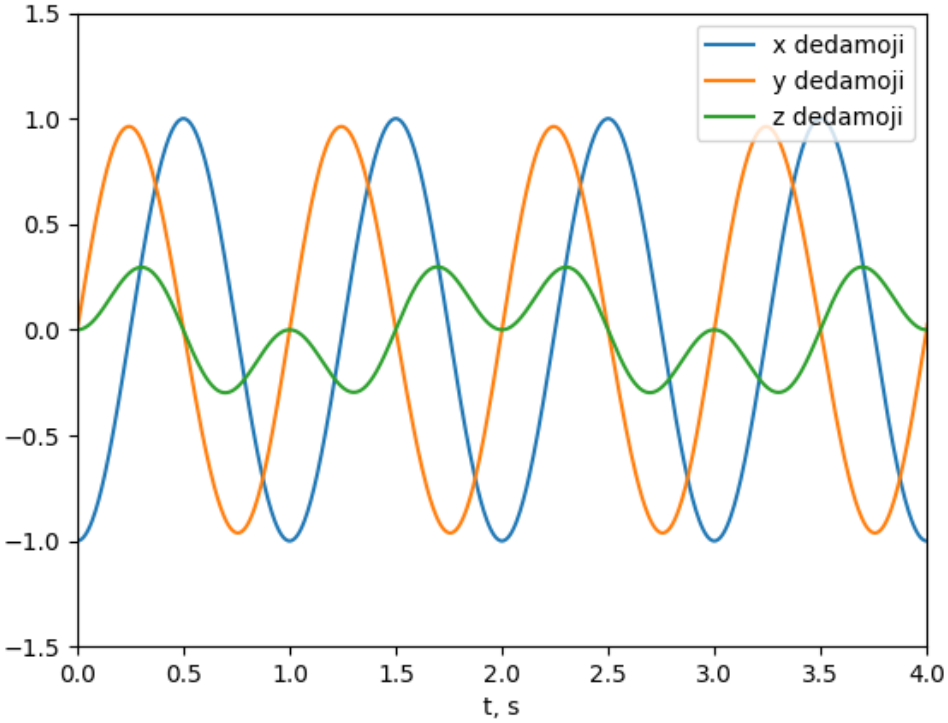
$$x = x(0) + v_x dt \tag{68}$$

čia: x – koordinatė esamu laiko momentu, $x(0)$ – koordinatė praeitu laiko momentu, v_x – momentinis tiesinis greitis X ašyje, (m/s); dt – laiko žingsnis, (s). Analogiškai apskaičiuojamos koordinatės *y* ir *z*.

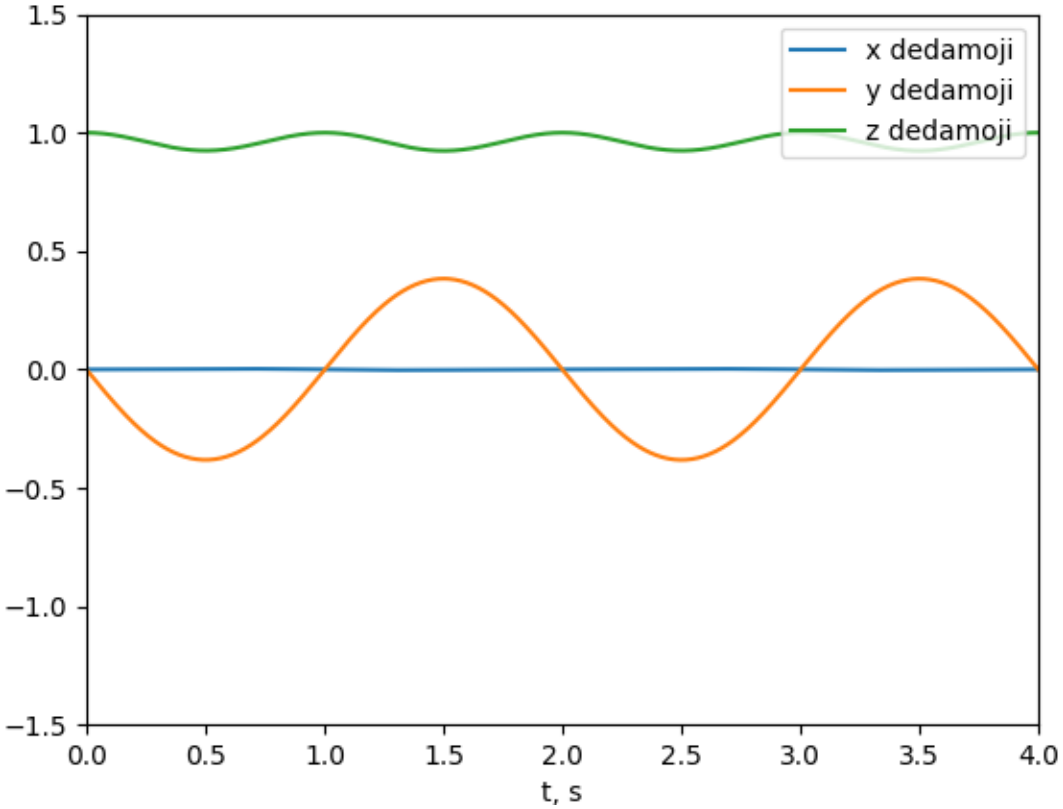
Naudojant minėtas pradines sąlygas, atkūrti krypties kosinusų matricos komponentai, tiesiniai pagreičiai, greičiai bei koordinatės. Rezultatai vaizduojami grafiškai.



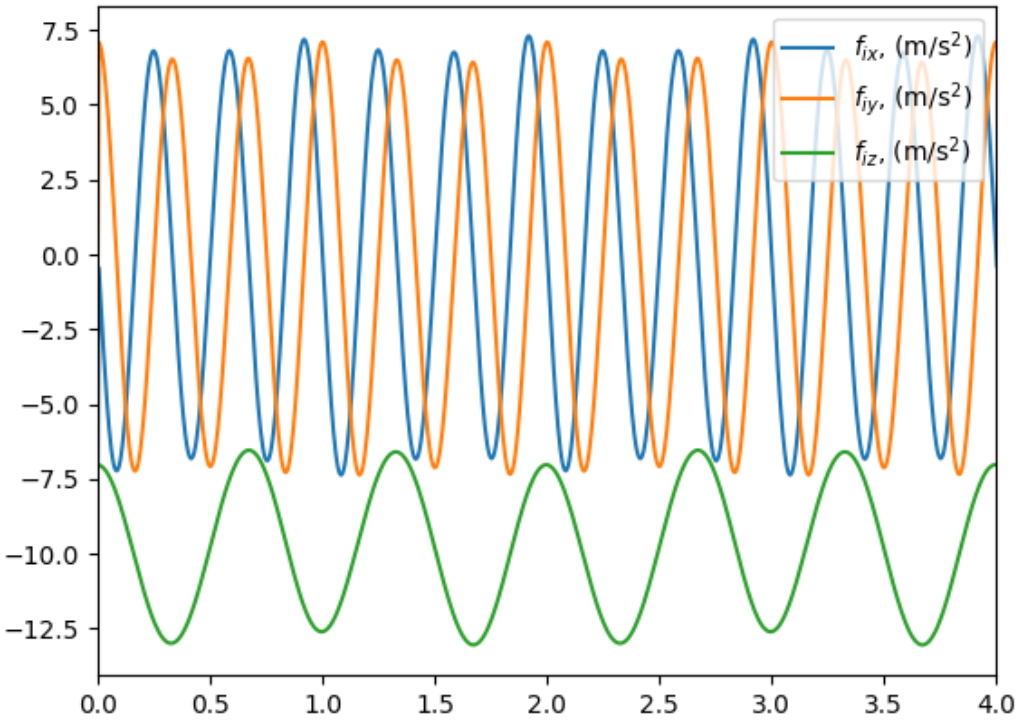
13 pav. INS atkurto Xi koordinačių kitimas laike: mėlyna spalva žymi projekciją į x ašį, oranžinė žymi projekciją į y ašį ir žalia – projekciją į z ašį.



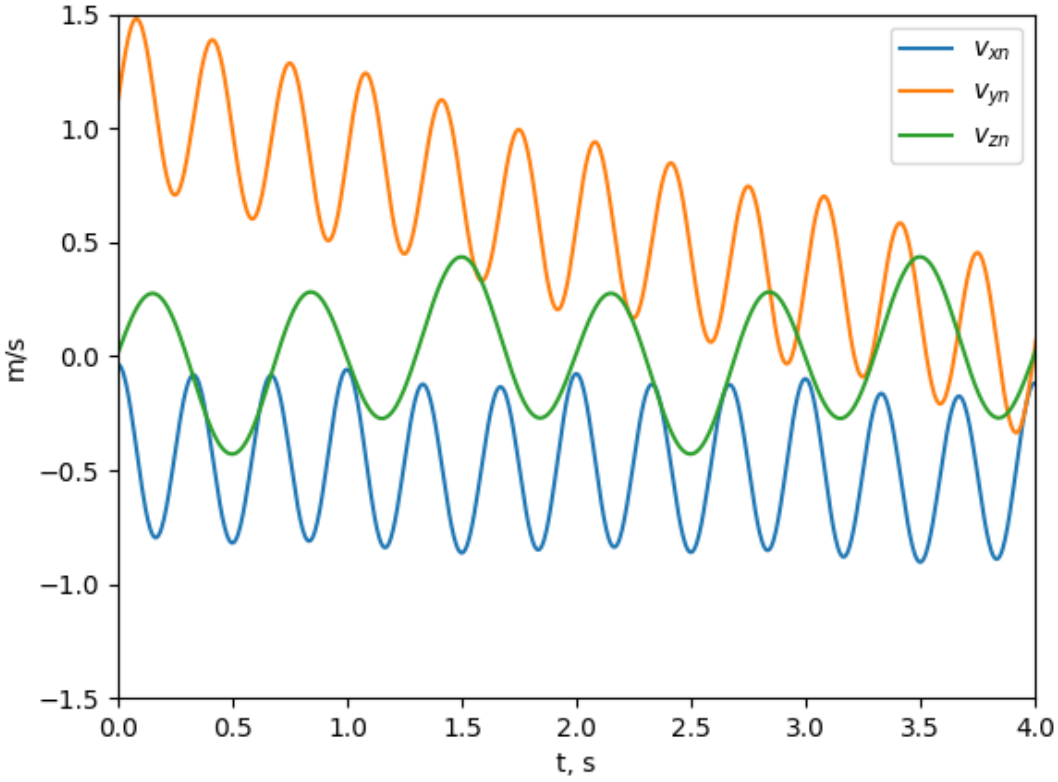
14 pav. INS atkurto Yi koordinačių kitimas laike: mėlyna spalva žymi projekciją į x ašį, oranžinė žymi projekciją į y ašį ir žalia – projekciją į z ašį.



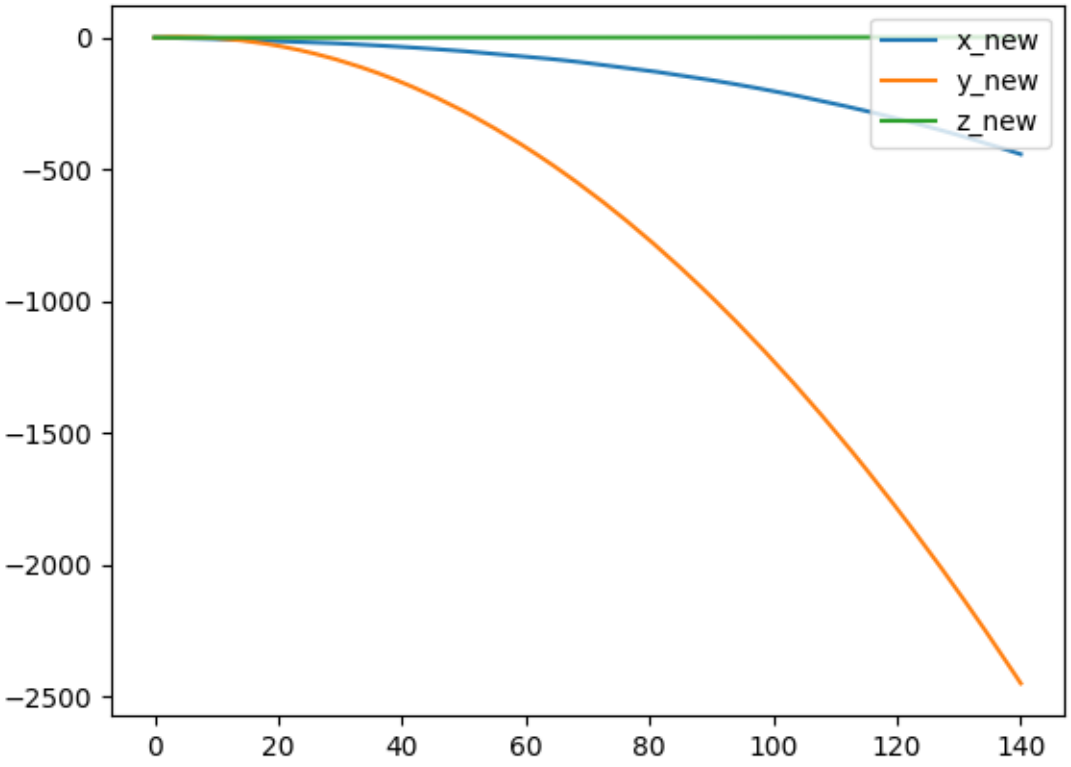
15 pav. INS atkurto Zi koordinačių kitimas laike: mėlyna spalva žymi projekciją į x ašį, oranžinė žymi projekciją į y ašį ir žalia – projekciją į z ašį.



16 pav. Atkurtos specifinės jėgos incercinėje atskaitos sistemoje priklausomybė nuo laiko visoms trims ašims.

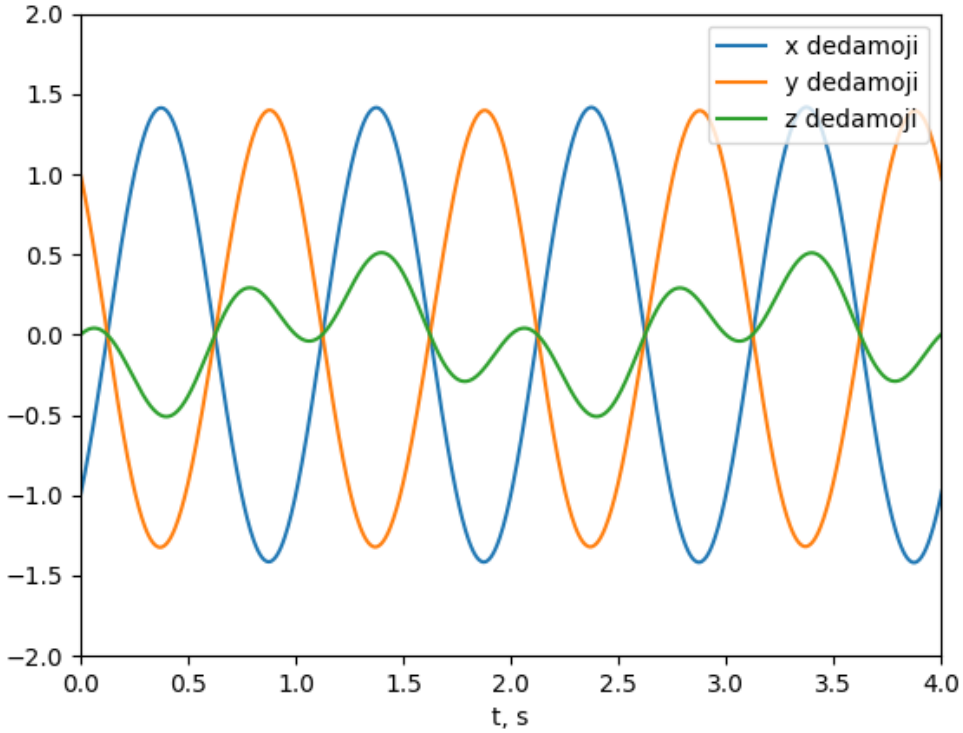


17 pav. Atkurtų momentinių greičių priklausomybės nuo laiko visoms trims ašims.

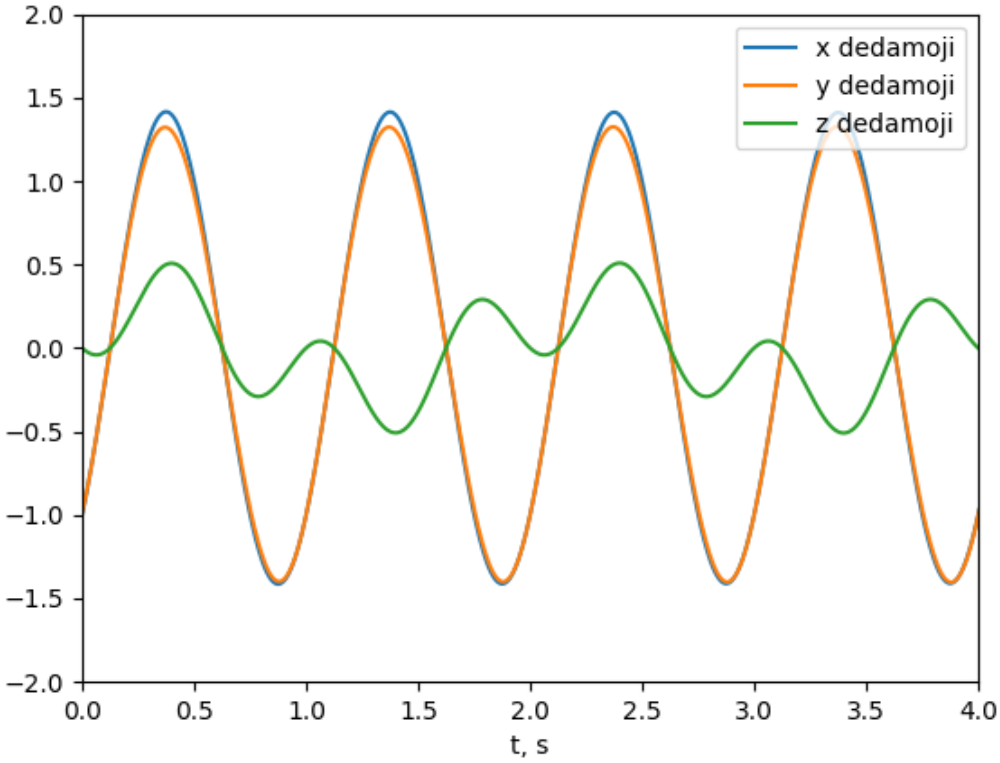


18 pav. Atkurtų koordinačių priklausomybė nuo laiko.

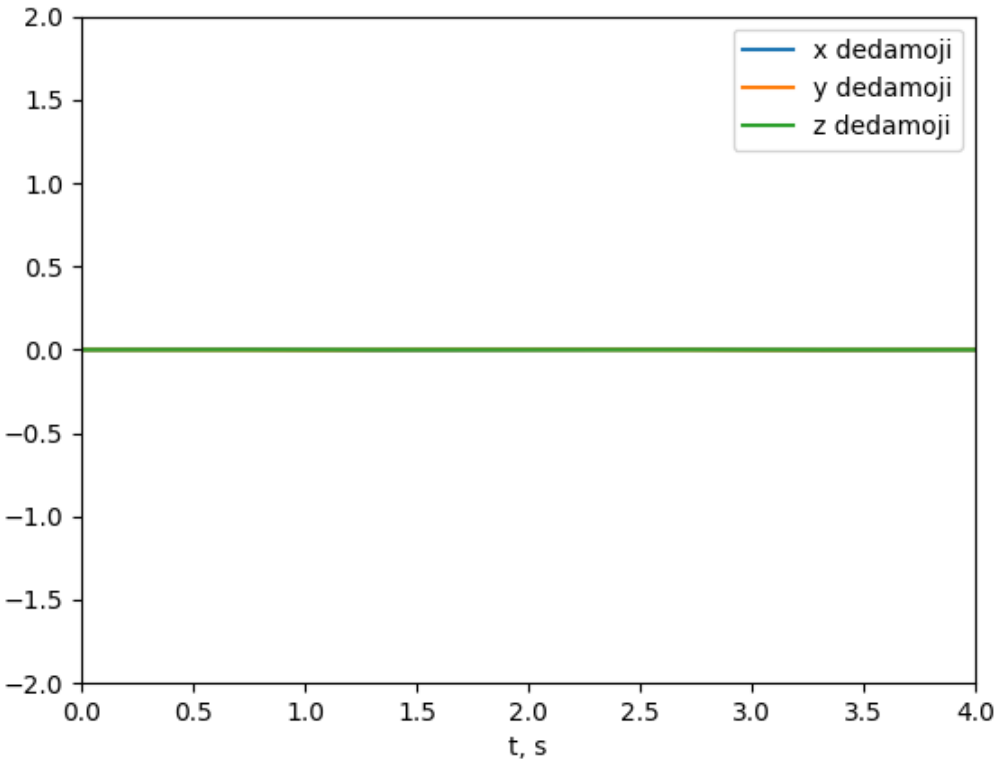
Atkūrus krypties kosinusų matricą, naudojantis kampiniais greičiais ir suintegravus atkurtą specifinę jėgą inercinėje koordinačių sistemoje – grafiškai pavaizduojami gautų rezultatų ir teorinių verčių skirtumai su minėtomis pradinėmis sąlygomis.



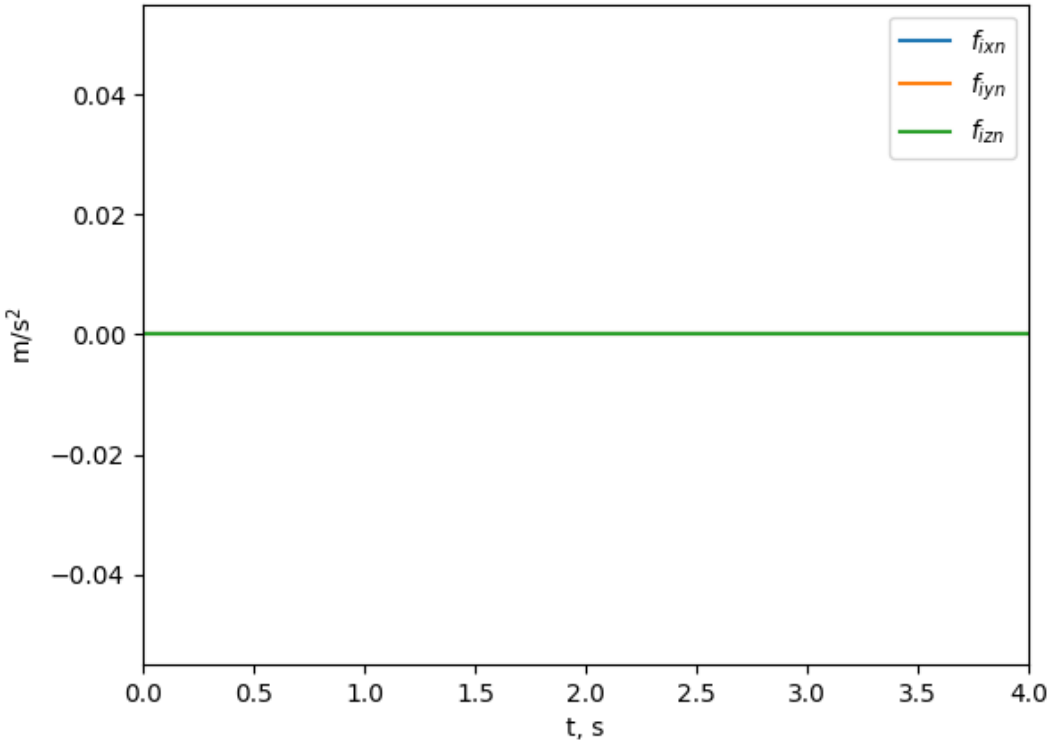
19 pav. Atkurtos krypties kosinusų matricos X ašies projekcijos verčių ir teorinės X ašies projekcijos verčių skirtumas.



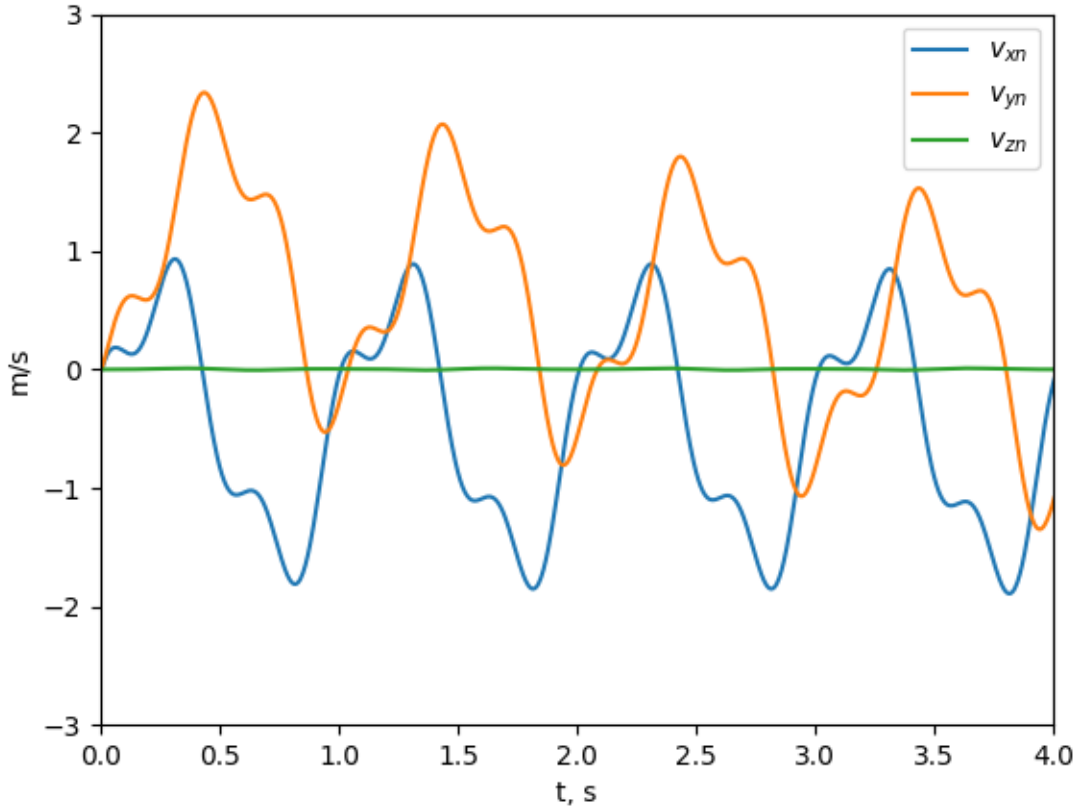
20 pav. Atkurtos krypties kosinusų matricos Y ašies projekcijos verčių ir teorinės Y ašies projekcijos verčių skirtumas.



21 pav. Atkurtos krypties kosinusų matricos Z ašies projekcijos verčių ir teorinės X ašies projekcijos verčių skirtumas.



22 pav. Atkurtos specifinės jėgos inercinėje koordinačių sistemoje atkurtų verčių palyginimas su teorinėmis vertėmis.



23 pav. Iš specifinės jėgos inercinėje koordinačių sistemoje suintegruotų ir iš parametrinių lygčių apskaičuotų greičio verčių skirtumas.

Grafiškai nelygintos atkurtos trajektorijos ir parametrinėmis lygtimis aprašytos trajektorijos vertės dėl iškilusių didelių integracijos netikslumų.

Išvados

Sumodeliavus inercinės navigacijos sistemos stendo modelį darytinos tokios išvados:

1. Modeliuojant inercinę navigacijos sistemą labai svarbu nustatyti kokios koordinačių sistemos atžvilgiu daromas modelis ir kaip modeliuojamos INS koordinačių ašys orientuotos atskaitos sistemos atžvilgiu.
2. Pasirinktas laiko žingsnis ir pasirenkamos integravimo schemos turi didelę įtaką atkuriamų duomenų tikslumui. Kuo mažiau pasirenkama narių išskleidus Teiloro eilutę integruojant – tuo labiau reiks mažinti laiko žingsnį tarp iteracijų.
3. Eulerio kampai nėra optimalus metodas atkuriant INS modelio kampinę padėtį dėl kardaninio užsirakinimo, kurį reikia spręsti skaitiniais metodais.
4. Skaitinio metodo netobulumai atkuriant trajektoriją ir kampinę padėtį smarkiai išauga ir turi didelę įtaką galutiniam duomenų atkūrimui.
5. Šiame darbe sukurtas teorinis stendo modelis yra tinkamas modeliuoti ir analizuoti inercinės navigacijos sistemos veikimą.

Literatūros sąrašas

[1] Henderson, D.M., McDonnell Douglas Tech. Services Co., Inc. *Shuttle program. Euler angles, quaternions and transformation matrices working relationships* Technical memorandum, Work of the US Gov. ID 19770024290

[2] David H. Titterton and John L. Weston *Strapdown Inertial Navigation Technology – 2nd Edition* The Institution of Electrical Engineers. 2 – asis leid., The institution of Electrical Engineers, United Kingdom, The American Institute of Aeronautics and Astronautics, USA, 2004. ISBN 0 86341 358 7

[3] Oliver J. Woodman *An introduction to inertial navigation* Technical report Number 696, University of Cambrige Computer Laboratory, 2007, ISSN 1476-2986

[4] William Premerlani and Paul Bizard *Direction Cosine Matrix IMU: Theory*, 2009, <https://wiki.paparazziuav.org/w/images/e/e5/DCMDraft2.pdf> [žiūrėta 2023-05-20]

[5] Attitude transformations <https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-attitudetran> [žiūrėta 2023-05-20]

[6] Bob Scannell *Precision MEMS Sensors Enable New Navigation Applicaitons* Technical article MS-2151, <https://www.analog.com/media/en/technical-documentation/tech-articles/MS-2151.pdf>

[7] Antanas Savickas *Avionika. Orlaivių padėties erdvėje nustatymo priemonės* 1 dalis, 2014-04-04,

< http://www.elektronika.lt/teorija/ivairi/44240/avionika-orlaiviu-padeties-erdveje-nustatymo-priemones/> [žiūrėta 2023-05-22]

[8] Dr. John Stryjewski *Coordinate Transformations* Abstract 2020-08-19

Investigation of models of inertial navigation system and its test stand

Augustas Jurevičius

Summary

Inertial navigation systems(INS) are an integral part of everyday life. Submarines, aircraft, satelites, military rocket systems, even smart devices like a phone all use inertial navigation systems to acurately calculate the position and orientation. Unfortunately, INS that can truly function independently from any other sensor or system is expensive to manufacture due to the cost of accurate sensors and difficult to replace if broken. Therefore it is important to create various models of inertial navigation systems to test algorithms that reduce the dependancy on hardware and focus on producing a clean signal with software.

The goal of this thesis is the investigation of a test stand for an inertial navigation system, comprised of a horizontal platform, suspended on ball bearings and a circular platform that sits on he horizontal platform and holds the strapdown inertial navigation system. By mathematicaly examining the motion of a point on the circular platform, a set of parametric equations was derived to describe the trajectory of the moving INS. From the trajectory, linear accelerations were calculated. Then, a direction cosine matrix was constructed to visualise the orientation of the INS and its propagation with time. A set of Euler angles was calculated out of the components of the direction cosine matrix and, using the Euler angles, angular speeds were calculated.

These calculations provided the exact same type of data that a physical inertial navigation system would, hence creating a synthetic INS. Then, procedures and algorithms were written to recreate the trajectory and orientation from the linear acceleration and angular speed data of the moving INS. After the recreation of the orientation it was found that the selection of the reference coordinate system used has a substantial influence on the values of the components of the direction cosine matrix. Since a positive rotation is considered clock-wise looking from the origin of the Z axis, selecting the Z axis direction to be opposite would result in a couter clock-wise direction yielding incorect results. Further analysis showed that the reconstructed direction cosine matrix is heavily involved in the recreation of the trajectory of the INS. Therefore in order to accurately recreate the position of the inertial navigation system, the orientation must be known precisely.

Priedai

Kodas, įgyvendinantis matematinį INS modelį sintetinių duomenų sukūrimui ir trajektorijos bei kampinės padėties atkūrimui Python programavimo kalba.

```
import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image


r = 0.18 # atstumas nuo jutiklio iki apskritiminės platformos centro
h = 0.01 # apskritiminės platformos aukštis nuo siūbavimo ašies X
dt = 0.005 # laiko žingsnis
time = np.arange(0, 140, dt) # laiko masyvas
Amp = np.pi/8 # siūbavimo aplink X ašį amplitudė
fphi = 1 # apskritiminės platformos sukimosi dažnis
ftheta = 0.5 # siūbavimo aplink X ašį dažnis
g = 9.8 # laisvojo kritimo pagreitis
R= 6378137.0 # Žemės spindulys
pics = []
C1=[]
C2=[]


# funkcija, koreguojanti kampų vertes, kad jo būtų nuo -pi iki pi rėžiuose
def normalize_angle(angle):
    result = np.fmod(angle + np.pi, np.pi*2.0)
    if (result <= 0.):
        return result + np.pi
    return result - np.pi
# funkcija, koreguojanti kampų vertes, kad jo būtų nuo 0 iki 2pi rėžiuose
def normalize_angle1(result):
    if (result < -np.pi):
        return result + 2.0*np.pi
    if (result > np.pi):
        return result - 2.0*np.pi
    return result

# paimta (adaptuota) is ros angles bibliotekos
```

```
def angdist(froma, toa):
    return normalize_angle1(toa-froma)
# procedūra, skirta stebėti funkcijose atliktų pakeitimų poveikį kuriamai ir
atkuriamai trajektorijai
def showWP(t, x, y, z, phi, theta):

    fig = plt.figure('WP')
    w = fig.add_subplot(111, projection='3d')

    w.azim = 0
    w.dist = 10
    w.elev = 25

    w.set_xlabel('X', fontweight='bold', fontsize=14)
    w.set_ylabel('Y', fontweight='bold', fontsize=14)
    w.set_zlabel('Z', fontweight='bold', fontsize=14)
    w.set_xlim([-.25, 0.25])
    w.set_ylim([-0.25, 0.25])
    w.set_zlim([-0.2, 0.2])

    w.plot(x[:t+2], y[:t+2], z[:t+2], '-r', linewidth=2)
    #w.plot(x_new[:t+2], y_new[:t+2], z_new[:t+2], '-k', linewidth=2)
    # orig. body koord. sistema
    w.quiver(x[t], y[t], z[t], 0.07 * np.cos(phi[t]), 0.07 * np.sin(phi[t]) *
np.cos(theta[t]),
             0.07 * np.sin(phi[t]) * np.sin(theta[t]), color='r')  # raudona/x

    w.quiver(x[t], y[t], z[t], 0.07 * -np.sin(phi[t]), 0.07 * np.cos(phi[t]) *
np.cos(theta[t]),
             0.07 * np.cos(phi[t]) * np.sin(theta[t]), color='g')  # zalia/y

    w.quiver(x[t], y[t], z[t], 0, -0.07 * np.sin(theta[t]), 0.07 * np.cos(theta[t]),
color='b')  # melyna/z

    # circle
    cx = []
    cy = []
    cz = []
    cp = 72
    for p in range(cp):
        cx.append(r * np.cos(np.pi * 2. * p / (cp - 1)))
        cy.append(r * np.sin(np.pi * 2. * p / (cp - 1)) * np.cos(theta[t]) - h *
np.sin(theta[t]))
        cz.append(r * np.sin(np.pi * 2. * p / (cp - 1)) * np.sin(theta[t]) + h *
np.cos(theta[t]))

    w.plot(cx, cy, cz, color='k')
```

```
    plt.show()
# funkcija, apskaičiuojanti apskritiminės platformos sukimosi kampo phi ir siūbavimo
kampo theta masyvus
def angles(A, fp, ft, t):
    phi = 2. * np.pi * fp * t
    #theta = (2*np.pi * ft * t)
    theta = A * np.sin(2*np.pi * ft * t)

    return  phi, theta

phi, theta = angles(Amp, fphi, ftheta, time)
# funkcija, pagal parametrines lygtis apskaičiuojanti x, y ir z koordinačių masyvus
def waypoints(r, h, phi, theta):
    x = r * np.cos(phi)
    y = r * np.sin(phi) * np.cos(theta) - h * np.sin(theta)
    z = r * np.sin(phi) * np.sin(theta) + h * np.cos(theta)

    return x, y, z

x, y, z = waypoints(r, h, phi, theta)
# funkcija, iš phi ir theta kampų apskaičiuojanti Eulerio kampus
def Euler_angles(phi, theta):
    #ZYX.
    roll = []
    pitch = []
    yaw = []
    for i in range(len(theta)):
        roll.append(-theta[i])
        pitch.append(0.)
        yaw.append(phi[i])

    return roll, pitch, yaw

roll, pitch, yaw = Euler_angles(phi, theta)
# procedūra, skirta patikrinti kode atliktų pakeitimų poveikį krypties kosinusų
matricai duotu laiko momentu t
def Cbi(t, phi, theta):

    c11 = np.cos(phi[t])
    c12 = np.sin(phi[t])*np.cos(theta[t])
    c13 = np.sin(phi[t])*np.sin(theta[t])
    c21 = -np.sin(phi[t])
    c22 = np.cos(phi[t])*np.cos(theta[t])
    c23 = np.cos(phi[t])*np.sin(theta[t])
    c31 = 0
    c32 = -np.sin(theta[t])
```

```
    c33 = np.cos(theta[t])
    return [c11, c12, c13, c21, c22, c23, c31, c32, c33]
# procedūra, skirta patikrinti ar teisingai apskaičiuoti Eulerio kampai iš jų sudarant
krypties kosinusų matricą
def Ea2Cb(t, roll, pitch, yaw):
    sroll = np.sin(roll[t])
    croll = np.cos(roll[t])
    spitch = np.sin(pitch[t])
    cpitch = np.cos(pitch[t])
    syaw = np.sin(yaw[t])
    cyaw = np.cos(yaw[t])
    return [cpitch*cyaw, -croll*syaw+sroll*spitch*cyaw, sroll*syaw+croll*spitch*cyaw,
\
    cpitch*syaw, croll*cyaw+sroll*spitch*syaw, -sroll*cyaw+croll*spitch*syaw, \
    -spitch, sroll*cpitch, croll*cpitch]
# funkcija, iš Eulerio kampų apskaičiuojanti kampinius greičius
def Kampiniai_greiciai(roll, pitch, yaw, dt, theta):
    omegax = []
    omegay = []
    omegaz = []
    for t in range(len(theta)):

        omegax.append((angdist(roll[t-1], roll[(t+1)%len(roll)])*0.5/dt -
np.sin(pitch[t])*angdist(yaw[t-1],yaw[(t+1)%len(yaw)])*0.5/dt))
        omegay.append(((np.cos(roll[t])*angdist(pitch[t-1],
pitch[(t+1)%len(pitch)])*0.5/dt + np.sin(roll[t])*np.cos(pitch[t])*angdist(yaw[t-
1],yaw[(t+1)%len(yaw)])*0.5/dt)))
        omegaz.append((np.cos(roll[t])*np.cos(pitch[t])*angdist(yaw[t-1],
yaw[(t+1)%len(yaw)])*0.5/dt - np.sin(roll[t])*angdist(pitch[t-1],
pitch[(t+1)%len(pitch)])*0.5/dt))


    return omegax, omegay, omegaz

omegax, omegay, omegaz = Kampiniai_greiciai(roll, pitch, yaw, dt, theta)

for i in range(len(time)):
    C1.append(Cbi(i, phi, theta))
    C2.append(Ea2Cb(i, roll, pitch, yaw))
# funkcija, iš koordinačių apskaičiuojanti greičius
def greiciai(x, y, z, dt, time):

    vx_ = []
    vy_ = []
    vz_ = []
    n = len(time)
    for t in range(n):
```

```
        vx_.append((x[(t+1)%n] - x[t-1])*0.5/dt)
        vy_.append((y[(t+1)%n] - y[t-1])*0.5/dt)
        vz_.append((z[(t+1)%n] - z[t-1])*0.5/dt)

    return vx_, vy_, vz_

vx, vy, vz = greiciai(x, y, z, dt, time)
# funkcija, iš greičių apskaičiuojanti pagreičius
def pagreiciai(vx, vy, vz, dt, time):

    ax_ = []
    ay_ = []
    az_ = []
    n = len(time)
    for t in range(n):

        ax_.append((vx[(t+1)%n] - vx[t-1])*0.5/dt)
        ay_.append((vy[(t+1)%n] - vy[t-1])*0.5/dt)
        az_.append((vz[(t+1)%n] - vz[t-1])*0.5/dt)

    return ax_, ay_, az_

ax, ay, az = pagreiciai(vx, vy, vz, dt, time)
# funkcija, skirta apskaičiuoti gravitacijos vektoriaus dedamąsias, priklausomai nuo
koordinatės kitimo
def gravity(x, y, z, g, R):
    l_ = np.sqrt(x**2 + y**2 + (z-R)**2)
    return - x*g*R*R/(l_**3), - y*g*R*R/(l_**3), -(z-R)*g*R*R/(l_**3)
# funkcija, apskaičiuojanti specifinę jėgą su kūnu susietoje koordinačių sistemoje
def fb(phi, theta, x, y, z, xa, ya, za, g, R):

    c11 = np.cos(phi)
    c12 = np.sin(phi)*np.cos(theta)
    c13 = np.sin(phi)*np.sin(theta)
    c21 = -np.sin(phi)
    c22 = np.cos(phi)*np.cos(theta)
    c23 = np.cos(phi)*np.sin(theta)
    c31 = np.zeros(len(phi))
    c32 = -np.sin(theta)
    c33 = np.cos(theta)

    gx, gy, gz = gravity(x, y, z, g, R)
    ax_ = xa - gx
    ay_ = ya - gy
    az_ = za - gz

    fb_x = c11*ax_ + c21*ay_ + c31*az_
```

```
    fb_y = c12*ax_ + c22*ay_ + c32*az_
    fb_z = c13*ax_ + c23*ay_ + c33*az_

    return fb_x, fb_y, fb_z, c11, c12, c13, c21, c22, c23, c31, c32, c33, ax_, ay_,
az_

fb_x, fb_y, fb_z, c11, c12, c13, c21, c22, c23, c31, c32, c33, ax_nog, ay_nog, az_nog
= fb(phi, theta, x, y, z, ax, ay, az, g, R)

# funkcija, integruojanti kampinius greičius ir sudaranti atnaujintą kosinusų matricą
def atkurta_orientacija(c11, c12, c13, c21, c22, c23, c31, c32, c33, omegax, omegay,
omegaz, time):

    cxx_ = []
    cxy_ = []
    cxz_ = []
    cyx_ = []
    cyy_ = []
    cyz_ = []
    czx_ = []
    czy_ = []
    czz_ = []

    cxx_.append(c11[0])
    cxy_.append(c12[0])
    cxz_.append(c13[0])
    cyx_.append(c21[0])
    cyy_.append(c22[0])
    cyz_.append(c23[0])
    czx_.append(c31[0])
    czy_.append(c32[0])
    czz_.append(c33[0])

    lx = []
    ly = []
    lz = []
    error = []

    for t in range(len(time)):

        c11_dot = cxy_[-1]*omegaz[t] - cxz_[-1]*omegay[t]
        c12_dot = cxz_[-1]*omegax[t] - cxx_[-1]*omegaz[t]
        c13_dot = cxx_[-1]*omegay[t] - cxy_[-1]*omegax[t]
        c21_dot = cyy_[-1]*omegaz[t] - cyz_[-1]*omegay[t]
        c22_dot = cyz_[-1]*omegax[t] - cyx_[-1]*omegaz[t]
        c23_dot = cyx_[-1]*omegay[t] - cyy_[-1]*omegax[t]
        c31_dot = czy_[-1]*omegaz[t] - czz_[-1]*omegay[t]
```

```
        c32_dot = czz_[-1]*omegax[t] - czx_[-1]*omegaz[t]
        c33_dot = czx_[-1]*omegay[t] - czy_[-1]*omegax[t]

        cxx_.append(cxx_[-1] + c11_dot*dt)
        cxy_.append(cxy_[-1] + c12_dot*dt)
        cxz_.append(cxz_[-1] + c13_dot*dt)
        cyx_.append(cyx_[-1] + c21_dot*dt)
        cyy_.append(cyy_[-1] + c22_dot*dt)
        cyz_.append(cyz_[-1] + c23_dot*dt)
        czx_.append(czx_[-1] + c31_dot*dt)
        czy_.append(czy_[-1] + c32_dot*dt)
        czz_.append(czz_[-1] + c33_dot*dt)

        lx = np.sqrt((cxx_[-1]*cxx_[-1] + cxy_[-1]*cxy_[-1] + cxz_[-1]*cxz_[-1]))
        ly = np.sqrt((cyx_[-1]*cyx_[-1] + cyy_[-1]*cyy_[-1] + cyz_[-1]*cyz_[-1]))
        lz = np.sqrt((czx_[-1]*czx_[-1] + czy_[-1]*czy_[-1] + czz_[-1]*czz_[-1]))

        cxx_[-1] /= lx
        cxy_[-1] /= lx
        cxz_[-1] /= lx

        cyx_[-1] /= ly
        cyy_[-1] /= ly
        cyz_[-1] /= ly

        czx_[-1] /= lz
        czy_[-1] /= lz
        czz_[-1] /= lz

        error = ((cxx_[-1] * cyx_[-1]) + (cxy_[-1] * cyy_[-1]) + (cxz_[-1] * cyz_[-
1]))

        cxx_[t] -= error/2 * cyx_[t]
        cxy_[t] -= error/2 * cyy_[t]
        cxz_[t] -= error/2 * cyz_[t]

        cyx_[t] -= error/2 * cxx_[t]
        cyy_[t] -= error/2 * cxy_[t]
        cyz_[t] -= error/2 * cxz_[t]

        czx_[-1] = cxy_[-1] * cyz_[-1] - cxz_[-1] * cyy_[-1]
        czy_[-1] = cxz_[-1] * cyx_[-1] - cxx_[-1] * cyz_[-1]
        czz_[-1] = cxx_[-1] * cyy_[-1] - cxy_[-1] * cyx_[-1]


    return cxx_, cxy_, cxz_, cyx_, cyy_, cyz_, czx_, czy_, czz_
```

```
cxx, cxy, cxz, cyx, cyy, cyz, czx, czy, czz = atkurta_orientacija(c11, c12, c13, c21,
c22, c23, c31, c32, c33, omegax, omegay, omegaz, time)
# funkcija, apskaičiuojanti specifinę jėgą inercinėje atskaitos sistemoje
def fi_skaiciavimas(fb_x, fb_y, fb_z, cxx, cxy, cxz, cyx, cyy, cyz, czx, czy, czz,
time):

    fi_x_ = []
    fi_y_ = []
    fi_z_ = []

    for t in range(len(time)):

        fi_x_.append(cxx[t]*fb_x[t] + cxy[t]*fb_y[t] + cxz[t]*fb_z[t])
        fi_y_.append(cyx[t]*fb_x[t] + cyy[t]*fb_y[t] + cyz[t]*fb_z[t])
        fi_z_.append(czx[t]*fb_x[t] + czy[t]*fb_y[t] + czz[t]*fb_z[t])

    return fi_x_, fi_y_, fi_z_

fi_x, fi_y, fi_z = fi_skaiciavimas(fb_x, fb_y, fb_z, cxx, cxy, cxz, cyx, cyy, cyz,
czx, czy, czz, time)

restored_x_ini = r # pradinė atkuriamos x koordinatės vertė
restored_y_ini = 0 # pradinė atkuriamos y koordinatės vertė
restored_z_ini = h # pradinė atkuriamos z koordinatės vertė

restored_vx_ini = vx[0] # pradinė atkuriamo greičio x ašimi vertė
restored_vy_ini = vy[0] # pradinė atkuriamo greičio y ašimi vertė
restored_vz_ini = vz[0] # pradinė atkuriamo greičio z ašimi vertė
# funkcija, naudojanti atkurtą krypties kosinusų matricą atnaujintos specifinės jėgos
apskaičiavimui
def atkurta_f_i(x, y, z, cxx, cxy, cxz, cyx, cyy, cyz, czx, czy, czz, fb_x, fb_y,
fb_z, g, R, restored_x_ini, restored_y_ini, restored_z_ini, restored_vx_ini,
restored_vy_ini, restored_vz_ini, dt, time):

    fi_x_ = []
    fi_y_ = []
    fi_z_ = []

    vx_new = []
    vy_new = []
    vz_new = []

    vx_new.append(restored_vx_ini)
    vy_new.append(restored_vy_ini)
    vz_new.append(restored_vz_ini)

    x_new = []
```

```
    y_new = []
    z_new = []

    x_new.append(restored_x_ini)
    y_new.append(restored_y_ini)
    z_new.append(restored_z_ini)

    gx, gy, gz = gravity(x, y, z, g, R)

    for t in range(len(time)):

        gx, gy, gz = gravity(x_new[-1], y_new[-1], z_new[-1], g, R)

        fi_x_.append(cxx[t]*fb_x[t] + cxy[t]*fb_y[t] + cxz[t]*fb_z[t])
        fi_y_.append(cyx[t]*fb_x[t] + cyy[t]*fb_y[t] + cyz[t]*fb_z[t])
        fi_z_.append(czx[t]*fb_x[t] + czy[t]*fb_y[t] + czz[t]*fb_z[t])

        vx_new.append(vx_new[-1] + (fi_x_[-1] + gx)*dt)
        vy_new.append(vy_new[-1] + (fi_y_[-1] + gy)*dt)
        vz_new.append(vz_new[-1] + (fi_z_[-1] + gz)*dt)

        x_new.append(x_new[-1] + vx_new[-1]*dt)
        y_new.append(y_new[-1] + vy_new[-1]*dt)
        z_new.append(z_new[-1] + vz_new[-1]*dt)


    return fi_x_, fi_y_, fi_z_, vx_new, vy_new, vz_new, x_new, y_new, z_new

fi_x_new, fi_y_new, fi_z_new, vx_new, vy_new, vz_new, x_new, y_new, z_new =
atkurta_f_i(x, y, z, cxx, cxy, cxz, cyx, cyy, cyz, czx, czy, czz, fb_x, fb_y, fb_z, g,
R, restored_x_ini, restored_y_ini, restored_z_ini, restored_vx_ini, restored_vy_ini,
restored_vz_ini, dt, time)
fi_x_diff = []
fi_y_diff = []
fi_z_diff = []
for i in range(len(fi_x_new)):
    fi_x_diff.append(fi_x_new[i] - fi_x[i])
    fi_y_diff.append(fi_y_new[i] - fi_y[i])
    fi_z_diff.append(fi_z_new[i] - fi_z[i])

vx_diff = []
vy_diff = []
vz_diff = []
for i in range(len(fi_x_new)):
    vx_diff.append(vx_new[i] - vx[i])
    vy_diff.append(vy_new[i] - vy[i])
    vz_diff.append(vz_new[i] - vz[i])
```

```
x_diff = []
y_diff = []
z_diff = []
for i in range(len(fi_x_new)):
    x_diff.append(x_new[i] - x[i])
    y_diff.append(y_new[i] - y[i])
    z_diff.append(z_new[i] - z[i])
# funkcija, skirta lyginti teorinius ir atkurtus dydžius
def palyginimai(cxx, cxy, cxz, cyx, cyy, cyz, czx, czy, czz, c11, c12, c13, c21, c22,
c23, c31, c32, c33, fb_x, fb_y, fb_z, fi_x_new, fi_y_new, fi_z_new):

    figc1 = plt.figure('c11-c13')
    _c1_ = figc1.add_subplot()
    _c1_.set_xlim(0, 4)
    _c1_.set_ylim(-1.5, 1.5)
    _c1_.set_xlabel('t, s')
    _c1_.plot(time, c11, label='x dedamoji')
    _c1_.plot(time, c12, label='y dedamoji')
    _c1_.plot(time, c13, label='z dedamoji')
    _c1_.legend(loc=1)
    plt.show()
    figc1.savefig('c11-c13')
    plt.close(figc1)

    figc2 = plt.figure('c21-c23')
    _c2_ = figc2.add_subplot()
    _c2_.set_xlim(0, 4)
    _c2_.set_ylim(-1.5, 1.5)
    _c2_.set_xlabel('t, s')
    _c2_.plot(time, c21, label='x dedamoji')
    _c2_.plot(time, c22, label='y dedamoji')
    _c2_.plot(time, c23, label='z dedamoji')
    _c2_.legend(loc=1)
    plt.show()
    figc2.savefig('c21-c23')
    plt.close(figc2)

    figc3 = plt.figure('c31-c33')
    _c3_ = figc3.add_subplot()
    _c3_.set_xlim(0, 4)
    _c3_.set_ylim(-1.5, 1.5)
    _c3_.set_xlabel('t, s')
    _c3_.plot(time, c31, label='x dedamoji')
    _c3_.plot(time, c32, label='y dedamoji')
    _c3_.plot(time, c33, label='z dedamoji')
    _c3_.legend(loc=1)
```

```
plt.show()
figc3.savefig('c31-c33')
plt.close(figc3)

figcx = plt.figure('cxx-cxz')
_cx_ = figcx.add_subplot()
_cx_.set_xlim(0, 4)
_cx_.set_ylim(-1.5, 1.5)
_cx_.set_xlabel('t, s')
_cx_.plot(time, cxx[1:], label='x dedamoji')
_cx_.plot(time, cxy[1:], label='y dedamoji')
_cx_.plot(time, cxz[1:], label='z dedamoji')
_cx_.legend(loc=1)
plt.show()
figcx.savefig('cxx-cxz')
plt.close(figcx)

figcy = plt.figure('cyx-cyz')
_cy_ = figcy.add_subplot()
_cy_.set_xlim(0, 4)
_cy_.set_ylim(-1.5, 1.5)
_cy_.set_xlabel('t, s')
_cy_.plot(time, cyx[1:], label='x dedamoji')
_cy_.plot(time, cyy[1:], label='y dedamoji')
_cy_.plot(time, cyz[1:], label='z dedamoji')
_cy_.legend(loc=1)
plt.show()
figcy.savefig('cyx-cyz')
plt.close(figcy)

figcz = plt.figure('czx-czz')
_cz_ = figcz.add_subplot()
_cz_.set_xlim(0, 4)
_cz_.set_ylim(-1.5, 1.5)
_cz_.set_xlabel('t, s')
_cz_.plot(time, czx[1:], label='x dedamoji')
_cz_.plot(time, czy[1:], label='y dedamoji')
_cz_.plot(time, czz[1:], label='z dedamoji')
_cz_.legend(loc=1)
plt.show()
figcz.savefig('czx-czz')
plt.close(figcz)

figcxcomp = plt.figure('cx_comp')
_cxcomp_ = figcxcomp.add_subplot()
_cxcomp_.set_xlim(0, 4)
_cxcomp_.set_ylim(-2, 2)
```

```
_cxcomp_.set_xlabel('t, s')
_cxcomp_.plot(time, (cxx[1:] - c11), label='x dedamoji')
_cxcomp_.plot(time, (cxy[1:] - c12), label='y dedamoji')
_cxcomp_.plot(time, (cxz[1:] - c13), label='z dedamoji')
_cxcomp_.legend(loc=1)
plt.show()
figcxcomp.savefig('cx_comp')
plt.close(figcxcomp)

figcycomp = plt.figure('cy_comp')
_cycomp_ = figcycomp.add_subplot()
_cycomp_.set_xlim(0, 4)
_cycomp_.set_ylim(-2, 2)
_cycomp_.set_xlabel('t, s')
_cycomp_.plot(time, (cyx[1:] - c21), label='x dedamoji')
_cycomp_.plot(time, (cyy[1:] - c22), label='y dedamoji')
_cycomp_.plot(time, (cyz[1:] - c23), label='z dedamoji')
_cycomp_.legend(loc=1)
plt.show()
figcycomp.savefig('cy_comp')
plt.close(figcycomp)

figczcomp = plt.figure('cz_comp')
_czcomp_ = figczcomp.add_subplot()
_czcomp_.set_xlim(0, 4)
_czcomp_.set_ylim(-2, 2)
_czcomp_.set_xlabel('t, s')
_czcomp_.plot(time, (czx[1:] - c31), label='x dedamoji')
_czcomp_.plot(time, (czy[1:] - c32), label='y dedamoji')
_czcomp_.plot(time, (czz[1:] - c33), label='z dedamoji')
_czcomp_.legend(loc=1)
plt.show()
figczcomp.savefig('cz_comp')
plt.close(figczcomp)

figomega = plt.figure('omega')
_omega_ = figomega.add_subplot()
_omega_.set_xlim(0, 4)
_omega_.set_ylim(-4, 10)
_omega_.set_xlabel('t, s')
_omega_.set_ylabel('rad/s')
_omega_.plot(time, omegax, label='$\omega_x$')
_omega_.plot(time, omegay, label='$\omega_y$')
_omega_.plot(time, omegaz, label='$\omega_z$')
_omega_.legend(loc=1)
plt.show()
figomega.savefig('omega.png')
```

```
plt.close(figomega)

figeuler = plt.figure('euler')
_euler_ = figeuler.add_subplot()
_euler_.set_xlim(0, 4)
_euler_.set_ylim(-2, 8)
_euler_.set_xlabel('t, s')
_euler_.set_ylabel('rad')
_euler_.plot(time, roll, label='$\phi$')
_euler_.plot(time, pitch, label=r'$\theta$')
_euler_.plot(time, yaw, label='$\psi$')
_euler_.legend(loc=1)
plt.show()
figeuler.savefig('euler')
plt.close(figeuler)

figfb = plt.figure('fb')
_fb_ = figfb.add_subplot()
_fb_.set_xlim(0, 4)
_fb_.set_xlabel('t, s')
_fb_.set_ylabel('m/s$^2$')
_fb_.plot(time, fb_x, label='$f_{bx}$')
_fb_.plot(time, fb_y, label='$f_{by}$')
_fb_.plot(time, fb_z, label='$f_{bz}$')
_fb_.legend(loc=1)
plt.show()
figfb.savefig('fb.png')
plt.close(figfb)

figfi = plt.figure('fi')
_fi_ = figfi.add_subplot()
_fi_.set_xlim(0, 4)
_fi_.set_xlabel('t, s')
_fi_.set_ylabel('m/s$^2$')
_fi_.plot(time, fi_x, label='$f_{ix}$')
_fi_.plot(time, fi_y, label='$f_{iy}$')
_fi_.plot(time, fi_z, label='$f_{iz}$')
_fi_.legend(loc=1)
plt.show()
figfi.savefig('fi.png')
plt.close(figfi)

figfinew = plt.figure('finew')
_finew_ = figfinew.add_subplot()
_finew_.set_xlim(0, 4)
_finew_.set_xlabel('t, s')
_finew_.set_ylabel('m/s$^2$')
```

```
_finew_.plot(time, fi_x_new, label='$f_{ixn}$')
_finew_.plot(time, fi_y_new, label='$f_{iyn}$')
_finew_.plot(time, fi_z_new, label='$f_{izn}$')
_finew_.legend(loc=1)
plt.show()
figfinew.savefig('finew.png')
plt.close(figfi)

figficompare = plt.figure('ficomp')
_ficomp_ = figficompare.add_subplot()
_ficomp_.set_xlim(0, 4)
_ficomp_.set_xlabel('t, s')
_ficomp_.set_ylabel('m/s$^2$')
_ficomp_.plot(time, fi_x_diff, label='$f_{ixn}$')
_ficomp_.plot(time, fi_y_diff, label='$f_{iyn}$')
_ficomp_.plot(time, fi_z_diff, label='$f_{izn}$')
_ficomp_.legend(loc=1)
plt.show()
figficompare.savefig('ficomp.png')
plt.close(figficompare)

figv = plt.figure('v')
_v_ = figv.add_subplot()
_v_.set_xlim(0, 4)
_v_.set_ylim(-1.5, 1.5)
_v_.set_xlabel('t, s')
_v_.set_ylabel('m/s')
_v_.plot(time, vx, label='$v_{x}$')
_v_.plot(time, vy, label='$v_{y}$')
_v_.plot(time, vy, label='$v_{z}$')
_v_.legend(loc=1)
plt.show()
figv.savefig('v.png')
plt.close(figv)

figvnew = plt.figure('vnew')
_vnew_ = figvnew.add_subplot()
_vnew_.set_xlim(0, 4)
_vnew_.set_ylim(-1.5, 1.5)
_vnew_.set_xlabel('t, s')
_vnew_.set_ylabel('m/s')
_vnew_.plot(time, vx_new[1:], label='$v_{xn}$')
_vnew_.plot(time, vy_new[1:], label='$v_{yn}$')
_vnew_.plot(time, vz_new[1:], label='$v_{zn}$')
_vnew_.legend(loc=1)
plt.show()
figvnew.savefig('vnew.png')
```

```
plt.close(figvnew)

figvcomp = plt.figure('vcompare')
_vcomp_ = figvcomp.add_subplot()
_vcomp_.set_xlim(0, 4)
_vcomp_.set_ylim(-3, 3)
_vcomp_.set_xlabel('t, s')
_vcomp_.set_ylabel('m/s')
_vcomp_.plot(time, vx_diff, label='$v_{xn}$')
_vcomp_.plot(time, vy_diff, label='$v_{yn}$')
_vcomp_.plot(time, vz_diff, label='$v_{zn}$')
_vcomp_.legend(loc=1)
plt.show()
figvcomp.savefig('vcomp.png')
plt.close(figvcomp)

figx = plt.figure('koord')
_x_ = figx.add_subplot()
_x_.set_xlabel('t, s')
_x_.set_ylabel('r, m')
_x_.plot(time, x, label='x')
_x_.plot(time, y, label='y')
_x_.plot(time, z, label='z')
_x_.set_xlim(0, 4)
_x_.legend(loc=1)
plt.show()
figx.savefig('koord.png')
plt.close(figx)

figxnew = plt.figure('koordnew')
_xnew_ = figxnew.add_subplot()
_xnew_.set_xlabel('t, s')
_xnew_.set_ylabel('r, m')
_xnew_.plot(time, x_new[1:], label='x_new')
_xnew_.plot(time, y_new[1:], label='y_new')
_xnew_.plot(time, z_new[1:], label='z_new')
_xnew_.legend(loc=1)
plt.show()
figxnew.savefig('koordnew.png')
plt.close(figxnew)

figxcomp = plt.figure('koordcomp')
_xcomp_ = figxcomp.add_subplot()
_xcomp_.set_xlabel('t, s')
_xcomp_.set_ylabel('r, m')
_xcomp_.plot(time, x_diff, label='x_new')
_xcomp_.plot(time, y_diff, label='y_new')
```

```
    _xcomp_.plot(time, z_diff, label='z_new')
    _xcomp_.legend(loc=1)
    plt.show()
    figxcomp.savefig('koordcomp.png')
    plt.close(figxcomp)

    plt.show()

palyginimai(cxx, cxy, cxz, cyx, cyy, cyz, czx, czy, czz, c11, c12, c13, c21, c22, c23,
c31, c32, c33, ax, ay, az, fi_x, fi_y, fi_z)
# ciklas, skirtas brėžti trajektorijas pagal teorines ir atkurtas koordinačių bei
orientacijos vertes
for t in range(len(time[:400])):

    fig = plt.figure('Trajektorija')
    w = fig.add_subplot(111, projection='3d')

    w.azim = 0
    w.dist = 8
    w.elev = 25

    w.set_xlabel('X', fontweight='bold', fontsize=14)
    w.set_ylabel('Y', fontweight='bold', fontsize=14)
    w.set_zlabel('Z', fontweight='bold', fontsize=14)
    w.set_xlim([-.25, 0.25])
    w.set_ylim([-0.25, 0.25])
    w.set_zlim([-0.2, 0.2])

    w.plot(x[:t+1], y[:t+1], z[:t+1], 'k', linewidth=2)
    w.plot(x_new[:t+1], y_new[:t+1], z_new[:t+1], color='pink', linewidth=2)

    # orig. body koord. sistema


    w.quiver(x[t], y[t], z[t], 0.07 * np.cos(phi[t]), 0.07 * np.sin(phi[t]) *
np.cos(theta[t]),
             0.07 * np.sin(phi[t]) * np.sin(theta[t]), color='c')  # žydra/x

    w.quiver(x[t], y[t], z[t], 0.07 * -np.sin(phi[t]), 0.07 * np.cos(phi[t]) *
np.cos(theta[t]),
             0.07 * np.cos(phi[t]) * np.sin(theta[t]), color='m')  # magenta/y

    w.quiver(x[t], y[t], z[t], 0, -0.07 * np.sin(theta[t]), 0.07 * np.cos(theta[t]),
color='y')  # geltona/z

    # rokonstruota body koord. sistema
    w.quiver(x_new[t], y_new[t], z_new[t], 0.07 * cxx[t], 0.07 * cxy[t],
```

```
            0.07 * cxz[t], color='b')  # mėlyna/ x

    w.quiver(x_new[t], y_new[t], z_new[t], 0.07 * cyx[t], 0.07 * cyy[t],
            0.07 * cyz[t], color='r')  # raudona/ y

    w.quiver(x_new[t], y_new[t], z_new[t], 0.07 * czx[t], 0.07 * czy[t],
             0.07 * czz[t], color='g')  # žalia/ z

    # circle
    k = []
    cx = []
    cy = []
    cz = []
    cp = 72
    for p in range(cp):
        cx.append(r * np.cos(np.pi * 2. * p / (cp - 1)))
        cy.append(r * np.sin(np.pi * 2. * p / (cp - 1)) * np.cos(theta[t]) - h *
np.sin(theta[t]))
        cz.append(r * np.sin(np.pi * 2. * p / (cp - 1)) * np.sin(theta[t]) + h *
np.cos(theta[t]))

    w.plot(cx, cy, cz, color='g')

    print("ciklas:" + str(t))
    #plt.show()
    buf = BytesIO()
    plt.savefig(buf)
    buf.seek(0)
    im = Image.open(buf)
    pics.append(im)
    plt.close(fig)

pics[0].save('ATKURTIEJI.gif', save_all=True, append_images=pics[1:])

plt.show()
```