



VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

ANTANO GUSTAIČIO AVIACIJOS INSTITUTAS

AVIACIJOS TECHNOLOGIJŲ KATEDRA

Monika Nepaitė

**DVIEJŲ ORLAIVIŲ KONFLIKTO PLOKŠTUMOJE SPRENDIMO
TYRIMAS**

RESEARCH OF AIRCRAFT PAIR PLANE CONFLICT RESOLUTION

Baigiamasis magistro darbas

Skrydžių valdymo studijų programa, valstybinis kodas 601H41002

Aeronautikos inžinerijos studijų kryptis

Vilnius, 2020

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ANTANO GUSTAIČIO AVIACIJOS INSTITUTAS
AVIACIJOS TECHNOLOGIJŲ KATEDRA

TVIRTINU
Katedros vedėjas

(Parašas)

doc. dr. Vytautas Rimša
(Vardas, pavardė)

2020-06-01

(Data)

Monika Nepaitė

DVIEJŲ ORLAIVIŲ KONFLIKTO PLOKŠTUMOJE SPRENDIMO TYRIMAS
RESEARCH OF AIRCRAFT PAIR PLANE CONFLICT RESOLUTION

Baigiamasis magistro darbas

Skrydžių valdymo studijų programa, valstybinis kodas 601H41002

Aeronautikos inžinerijos studijų kryptis

Vadovas

prof. habil. dr. Marijonas Bogdevičius
(Moksl. laipsnis/pedag. vardas, vardas, pavardė)

(Parašas)

(Data)

Konsultantas

Vytautas Rafanavičius
(Moksl. laipsnis/pedag. vardas, vardas, pavardė)

(Parašas)

(Data)

Lietuvių kalbos konsultantas lekt. Regina Žukienė

(Moksl. laipsnis/pedag. vardas, vardas, pavardė)

(Parašas)

(Data)

Vilnius, 2020

Vilniaus Gedimino technikos universitetas
Antano Gustaičio aviacijos institutas
Aviacijos technologijų katedra

ISBN ISSN
Egz. sk.
Data-.....-.....

Vientisosios studijų **Skrydžių valdymo** programos magistro baigiamasis darbas

Pavadinimas **Dviejų orlaivių konflikto plokštumoje sprendimo tyrimas**

Autorius **Monika Nepaitė**

Vadovas **Marijonas Bogdevičius**

Kalba: lietuvių

Anotacija

Baigiamajame magistro darbe nagrinėjamas dviejų orlaivių konflikto plokštumoje sprendimas sudarant nuoseklias trajektorijas ir minimizuojant trajektorijos ilgėjimą. Išnagrinėti galimi šios problemos sprendimo metodai bei algoritmai, jų privalumai bei trūkumai. Pateiktas naujas dviejų orlaivių konflikto plokštumoje sprendimo matematinis modelis, paremtas dinaminio programavimo ir Monte Karlo paieškos metodo kombinacija, jo sudarymo teorija bei taikymo pavyzdžiai įvairioms konfliktinėms situacijoms. Baigiamajame darbe nustatyti ir prie pasirinkto greičio nagrinėjami parametrai, turintys įtakos apskaičiuojamų išvengiamųjų trajektorijų ilgiui ir skaičiavimų trukmei. Išnagrinėjus teorinį algoritmo efektyvumą, pateikiamos baigiamojo darbo išvados ir rekomendacijos tolimesniems tyrimams.

Darbą sudaro 8 dalys: įvadas, orlaivių konfliktų sprendimo literatūros apžvalga, siūlomo dviejų orlaivių konflikto plokštumoje sprendimo matematinio modelio teorinė dalis, siūlomo matematinio modelio aprašymas, gautų rezultatų apibendrinimas ir rekomendacijos, išvados, literatūros sąrašas ir A priedas.

Darbo apimtis – 57 p. teksto be priedų, 26 iliustr., 15 lent., 45 bibliografiniai šaltiniai.

Atskirai pridedami darbo priedai.

Prasminiai žodžiai: Dinaminis programavimas, Monte Karlo paieška, orlaivių konfliktų sprendimas, trajektorijos ilgio optimizavimas.

Vilnius Gediminas Technical University
Antanas Gustaitis' Aviation Institute
Department of Aviation Technologies

ISBN	ISSN
Copies No.	
Date-.....-.....	

Integrated Studies **Air Traffic Control** study programme Master Graduation Thesis

Title	Research of aircraft pair plane conflict resolution
-------	---

Author	Monika Nepaitė
--------	----------------

Academic supervisor	Marijonas Bogdevičius
---------------------	------------------------------

Thesis language: Lithuanian

Annotation

This Master thesis studies aircraft pair conflict resolution in the plane by using smoothened trajectories and minimizing the lengthening of the resolution trajectory. Various possible methods and algorithms that can be used for solving such problem are reviewed and their pros and cons are discussed. New mathematical model for solving aircraft pair conflicts in the plane based on dynamic programming and Monte Carlo tree search is proposed, its theory and examples of application to different types of conflict geometry is provided. The paper also discusses discovered parameters that, at chosen speed, have an effect on the length of resolution trajectories and running time. Given the results, conclusions are provided and recommendations for further research are proposed.

The paper comprises 8 parts: introduction, aircraft conflict resolution literature review, theory of proposed mathematical model, description and explanation of proposed mathematical model, results of the research based on proposed model and recommendations for further research, conclusions, references and A appendix.

Thesis consists of: 57 p. text without appendixes, 26 pictures, 15 tables, 45 bibliographical references.

Appendixes included.

Keywords: Aircraft conflict resolution, dynamic programming, Monte Carlo tree search, trajectory length optimization.

TURINYS

SANTRUMPŲ SĄRAŠAS	8
ILIUSTRACIJŲ SĄRAŠAS	8
LENTELIŲ SĄRAŠAS	9
ĮVADAS	10
1. ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMUJE APŽVALGA	11
1.1. Orlaivių konfliktų sprendimo automatizavimo poreikis	11
1.2. Orlaivių trajektorijų nustatymas	12
1.3. Konfliktų sprendimo plokštumoje metodai	15
1.3.1. Orlaivių kurso keitimo metodai	15
1.4. Vėjo įtaka orlaivio pozicijos neapibrėžtumui	21
1.5. Pirmojo skyriaus išvados	22
2. ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMUJE MATEMATINIO MODELIO TEORINĖ DALIS	23
2.1 Dinaminis programavimas	23
2.2 Markovo sprendimų priėmimo procesas	24
2.3 Monte Karlo paieškos algoritmas	26
2.4 <i>Python</i> programavimo kalba	28
3. ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMUJE MATEMATINIS MODELIS	30
3.1 Tiriama situacija ir bendroji sprendimo logika	30
3.2 Tyrimo prielaidos ir apribojimai	32
3.3 Orlaivių konflikto sprendimo algoritmas	33
4. ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMUJE REZULTATAI	40
4.1 Orlaivių konfliktų sprendimo rezultatai	40
4.1.1. Priešpriešinis konfliktas	41
4.1.2. Stataus trajektorijų susikirtimo kampo konfliktas	42
4.1.3. To paties maršruto konfliktas	45
4.1.4. Smailaus trajektorijų susikirtimo kampo konfliktas	47
4.1.5. Vėjo įtaka konflikto išvengiamajai trajektorijai	50
4.2. Algoritmo ir jo sprendimo efektyvumo tyrimo rezultatai	51
4.2. Tyrimo rezultatų apibendrinimas ir rekomendacijos	57
IŠVADOS	62
LITERATŪROS SĄRAŠAS	63
A PRIEDAS	67

SANTRUMPŲ SĄRAŠAS

ADS-B – automatinė stebėjimo sistema (angl. *Automatic dependent surveillance – broadcast*)

KT – mazgas (angl. *Knot*)

MCTS – Monte Karlo paieškos algoritmas (angl. *Monte Carlo tree search*)

NM – jūrmylė (angl. *Nautical mile*)

TCAS – orlaivių susidūrimo vengimo sistema (angl. *Traffic collision avoidance system*)

ILIUSTRACIJŲ SĄRAŠAS

1.1 pav. Dviejų tiesių trajektorijos	13
1.2 pav. Orlaivių konflikto sprendimas keičiant kursą	18
2.1 pav. Markovo elementų sąryšis	25
2.2 pav. MCTS algoritmo iteracija	27
3.1 pav. Valdomo orlaivio ir orlaivio įsibrovėlio konfliktas	30
3.2 pav. Veiksmo pasirinkimas ir trajektorijos numatymas	31
3.3 pav. Valdomo orlaivio ir orlaivio įsibrovėlio konflikto sprendimas	31
3.4 pav. Bendroji algoritmo schema	36
3.5 pav. Taško z atšakų z' radimas	37
3.6 pav. Veiksmų simuliacija iki galutinės būsenos	38
3.7 pav. Atlygio priskyrimas tirtiems taškams	39
4.1 pav. Priešpriešinio konflikto sprendimo trajektorijos	41
4.2 pav. Atstumo tarp orlaivių kitimas skrydžio metu (priešpriešinis konfliktas)	42
4.3 pav. Stataus trajektorijų susikirtimo kampo konflikto sprendimo trajektorijos, kai: a) orlaivių greičiai vienodi; b) orlaivis įsibrovėlis 20 % greitesnis už valdomą orlaivį	44
4.4 pav. Atstumo tarp orlaivių kitimas skrydžio metu (stataus trajektorijų susikirtimo kampo konfliktas)	44
4.5 pav. To paties maršruto konflikto sprendimo trajektorijos	46
4.6 pav. Atstumo tarp orlaivių kitimas skrydžio metu (to paties maršruto konfliktas)	46
4.7 pav. Smailaus trajektorijų susikirtimo kampo konflikto sprendimo trajektorijos, kai: a) orlaivių greičiai vienodi; b) valdomas orlaivis greitesnis 25 %; c) valdomas orlaivis greitesnis 50 %	48
4.8 pav. Atstumo tarp orlaivių kitimas skrydžio metu (smailaus trajektorijų susikirtimo kampo konfliktas)	49
4.9 pav. Vėjo įtaka stataus trajektorijų susikirtimo kampo konflikto sprendimo trajektorijoms	51
4.10 pav. Sprendimo skaičiavimų trukmės priklausomybė nuo simuliacijų gylio ir kiekio	52
4.11 pav. Vidutinio trajektorijų ilgio pokyčio priklausomybė nuo simuliacijų gylio ir laiko žingsnio	53

4.12 pav. Skaičiavimų trukmės priklausomybė nuo laiko žingsnio ir simuliacijų gylio	54
4.13 pav. Vidutinio trajektorijos ilgio pokyčio priklausomybė nuo orlaivių greičių ir laiko žingsnių.....	54
4.14 pav. Vidutinio trajektorijų ilgių pokyčio priklausomybė nuo posūkio vertės	55
4.15 pav. Skirtingo leistino posūkio kampo trajektorijos, kai: a) priešpriešinis konfliktas; b) stataus trajektorijų susikirtimo kampo konfliktas; c) to paties maršruto konfliktas; d) smailaus trajektorijų susikirtimo kampo konfliktas	57

LENTELIŲ SĄRAŠAS

3.2 lentelė. Pagrindinės algoritmo įvestys ir jų žymėjimas.....	34
3.3 lentelė. Algoritmo parametrai ir jų žymėjimas	35
4.1 lentelė. Brėžinių legenda.....	40
4.2 lentelė. Standartiniai algoritmo nustatymai	40
4.3 lentelė. Priešpriešinio konflikto įvestys	41
4.4 lentelė. Priešpriešinio konflikto sprendimo rezultatai.....	42
4.5 lentelė. Stataus trajektorijų susikirtimo kampo konflikto įvestys	43
4.6 lentelė. Stataus trajektorijų susikirtimo kampo konflikto sprendimo rezultatai	44
4.7 lentelė. To paties maršruto konflikto įvestys	45
4.8 lentelė. To paties maršruto konflikto sprendimo rezultatai.....	46
4.9 lentelė. Smailaus trajektorijų susikirtimo kampo konflikto įvestys	47
4.10 lentelė. Smailaus trajektorijų susikirtimo kampo konflikto sprendimo rezultatai	49
4.11 lentelė. Vėjo įtakos išvengiamajai trajektorijai rezultatai	50
4.12 lentelė. Siūlomo modelio palyginimas su standartiniu sprendimu.....	58
4.13 lentelė. Parametrų įtaka konflikto išvengiamajai trajektorijai.	60

IVADAS

Temos aktualumas. Kasmet augantis skrydžių skaičius kelia vis daugiau iššūkių Europos aviacijai. Remiantis Eurokontrolės (2019) tyrimais, stabili ekonomika, industrijos tendencijos bei kultūrinis regiono užimtumas lems beveik 2 % metinį skrydžių skaičiaus augimą per ateinančius 5 metus. Augant skrydžių skaičiui, tačiau nekeičiant oro erdvės pralaidumo, didės ir orlaivių konfliktų tikimybė. Skrydžių vadovų naudojami metodai yra įgalūs spręsti orlaivių konfliktus, tačiau nėra pakankamai efektyvūs ar optimalūs. Be to, pastarieji metodai nebeatitinka aukštų reikalavimų, keliamų orlaivių degalų sąnaudoms, bei laiko apribojimų, būdingų naujai Europos orlaivių srautų politikai, paremtai 4D trajektorijų technologija. Dėl šių priežasčių iškyla orlaivių konfliktų sprendimo automatizavimo reikiamybė. 4D trajektorijos, kartu su laisvojo skrydžio koncepcija galėtų išspręsti oro eismo valdymo efektyvumo problemą, tačiau tam reikalingos naujos orlaivių konfliktų sprendimo sistemos, kuriuos šiuo metu yra dar ankstyvoje vystymosi stadijoje.

Problema. Siekiant sumažinti degalų bei skrydžio laiko sąnaudas ir komunikavimą tarp skrydžių vadovo ir piloto orlaivių konflikto sprendimo metu, reikalinga realaus laiko konflikto sprendimo sistema, galinti užtikrinti tiek saugumą, tiek kuo optimalesnes orlaivių trajektorijas. Dabartinis orlaivių konflikto sprendimas remiasi ribotomis skrydžių vadovo instrukcijomis, todėl skrydis tampa neefektyvus, orlaivių trajektorijos neracionaliai ilgėja, o tai lemia skrydžių vėlavimus maršrute ir atvykimo aerodrome. Todėl nauja orlaivių konflikto sprendimo sistema privalo užtikrinti saugų atstumą kiek įmanoma trumpinant orlaivio trajektorijos pailgėjimą.

Tyrimo objektas. Dviejų orlaivių konflikto sprendimas plokštumoje esant trajektorijos ilgio ribojimams.

Darbo tikslas – išspręsti dviejų orlaivių konfliktą plokštumoje efektyviai keičiant skrydžio parametrus ir užtikrinant minimalų įmanomą trajektorijos pailgėjimą.

Darbo hipotezė. Remiantis dinaminiu programavimu orlaivių konflikto sprendimo metu orlaivį galima nuosekliai grąžinti į jo pradinę trajektoriją mažinant trajektorijos ilgėjimą.

Darbo uždaviniai:

1. Atlikti mokslinių darbų ir tyrimo metodų analizę nagrinėjama tematika;
2. Sudaryti matematinį dviejų orlaivių konflikto sprendimo modelį;
3. Sukurti programinę įrangą modeliui realizuoti panaudojant *Python* programinę kalbą;
4. Atlikti sukurto matematinio modelio parametrų analizę;
5. Pateikti sprendimo pavyzdžius esant skirtingoms konfliktinėms situacijoms;
6. Pateikti apibendrintas tyrimo išvadas ir rekomendacijas.

Darbo metodika. Darbo uždaviniams įgyvendinti bus naudojami matematinis modeliavimas, skaitiniai tyrimai pagal sukurta matematinį modelį ir lyginamoji analizė.

1. ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMOJE APŽVALGA

1.1. Orlaivių konfliktų sprendimo automatizavimo poreikis

Pagrindinis efektyvaus oro srautų valdymo tikslas yra stebėti eismą, užtikrinant skrydžių saugą bei mažinant vėlavimų kiekį ir jų įvykimo riziką. Remiantis standartiniais orlaivių skirstymo reikalavimais, regiono skrydžių valdymo erdvėje orlaiviai privalo būti atskirti vienas nuo kito 5 NM atstumu horizontalioje plokštumoje. Taigi, aplink orlaivį yra sukuriamą saugi zona. Sprendžiant konfliktus, t. y. užkertant kelią suartėjimui pažeidus minimalaus orlaivių skirstymo intervalus, pilotui yra suteikiamos instrukcijos pakeisti tam tikrus skrydžio parametrus. Skrydžio parametrai, naudojami konflikto sprendimui plokštumoje, yra horizontalus orlaivio greitis ir kursas.

Visų pirma, šie pokyčiai turi įtakos sunaudojamo kuro kiekiui ir vėlavimams. Nors vėlavimų ir kuro išlaidos nėra didesnės už skrydžių vadovų išlaikymo kaštus, vėlavimai sutrikdo apskaičiuotą skrydžių srautą apkraudami oro erdvę, o maksimalus oro erdvės pralaidumo užpildymas lemia pasikeitusį skrydžių vadovų darbo krūvį (Lehouillier et al., 2016). Pasak M. Radanovičiaus ir kolegų (2017), didelis eismo intensyvumas apsunkina skrydžių vadovo gebėjimą saugiai ir laiku išskirstyti orlaivius, o padidėjęs aptiktų konfliktų skaičius turi įtakos bendram oro erdvės saugumui. Kai kurių autorių teigiama (Donciu & Pleter, 2009), kad centralizuotas skrydžių valdymas, paremtas žodinėmis instrukcijomis, apskritai yra pernelyg apsunkintas ir itin ribojamas žmogiškųjų faktorių. Dėl šių priežasčių centralizuoto skrydžių valdymo efektyvumas yra žemas ir neužkerta kelio konfliktinėms situacijoms, todėl konfliktų aptikimo ir sprendimo sistemos yra būtinos oro eismo vadybai. Iš dalies ar visiškai automatizuotas sprendimų priėmimas padėtų įgyvendinti proaktyvų skrydžių valdymą, sumažintų darbo krūvį bei galėtų apdoroti daugiau ir kompleksiškesnių konfliktų sprendimo būdų (Cafieri & Durand, 2014; Lehouillier et al., 2016; Wang et al., 2019).

Deja, dabartinės skrydžių valdymo sistemos, naudojamos centralizuotam skrydžių valdymui, nėra įgalios spręsti konfliktų. Užsienio autorių teigimu (Trapsilawati et al., 2016), šiuo metu moderniose sistemose diegiamos konfliktų aptikimo funkcijos yra tik įspėjamosios, todėl skrydžių vadovas privalo pats išspręsti konfliktą prieš įsijungiant orlaivių susidūrimo sistemai TCAS. Tyrėjų nuomone (Durand et al., 1996), taip yra dėl to, kad aukštus automatizavimo ir optimizavimo standartus atitinkančią sistemą yra itin sunku sukurti. Nauja konfliktų sprendimo sistema turi numatyti įvairius neapibrėžtumus, pavyzdžiui, nenumatytus orlaivio greičio pokyčius, parengti instrukcijas laiku, bei būti aprūpinta orlaivio charakteristikų duomenų baze

Dėl šių priežasčių, konflikto išvengimo atsakomybė gali būti atiduodama pilotui, taikant laisvojo skrydžio koncepciją (Tomlin et al., 1998). Viena pirmųjų šios koncepcijos sistemų yra TCAS, tačiau kai kurių autorių (Radanovic et al., 2017) teigimu, nors TCAS sistema puikiai veikia sprendžiant orlaivių poros konfliktus, visgi ji yra skirta žemo intensyvumo oro erdvėms ir nėra

pritaikyta spręsti daugiau nei dviejų orlaivių konfliktus. Be to, TCAS konfliktų sprendimai yra taikomi, kai laiko iki konflikto yra likę labai mažai, todėl sprendimai dažnai būna drastiški ir neoptimalūs. Naujos kartos konflikto sprendimo sistema galėtų pati parinkti pačią optimaliausią konflikto sprendimo trajektoriją ir tiesiogiai perduotų šią informaciją kitam orlaiviui ADS-B ryšiu, taip atsisakant žodinės komunikacijos ir sutaupant laiko bei degalų (Donciu & Pleter, 2009; Tomlin et al., 1998).

1.2. Orlaivių trajektorijų nustatymas

Sprendžiant orlaivių konfliktus gaunamos naujos trajektorijos, kurios gali būti tiek tiesės, tiek kreivės. Teorinės iš dviejų ar daugiau tiesių sudarytos trajektorijos yra dažnai naudojamos centralizuotam skrydžių valdymui, nes tai yra vienas paprasčiausių būdų apibūdinti orlaivių judėjimą konflikto sprendimo metu ir gali būti nesudėtingai perteikiamos skrydžių vadovo išduodamomis instrukcijomis. Trajektorijos, sudarytos iš tiesių buvo naudojamos keleto mokslininkų darbuose (Bilimoria, 2000; Erzberger, 2005; Geser & Munoz, 2002; Paielli, 2003; Park, Oh, & Tahk, 2008). Šiuose darbuose remtasi reliatyvaus greičio vektoriumi ir minimalaus atstumo skaičiavimais.

Atlikus paprastus trigonometrinius skaičiavimus greičių vektorių trikampiui, galima apskaičiuoti reliatyvaus greičio vektoriaus dydį bei kryptį. Kampas tarp reliatyvaus greičio vektoriaus ir tiesės tarp orlaivių, reikalingas išspręsti konfliktui, gaunamas remiantis formule (Erzberger, 2005):

$$\beta = \pm \sin^{-1}(R_m/S_0) \quad (1.1)$$

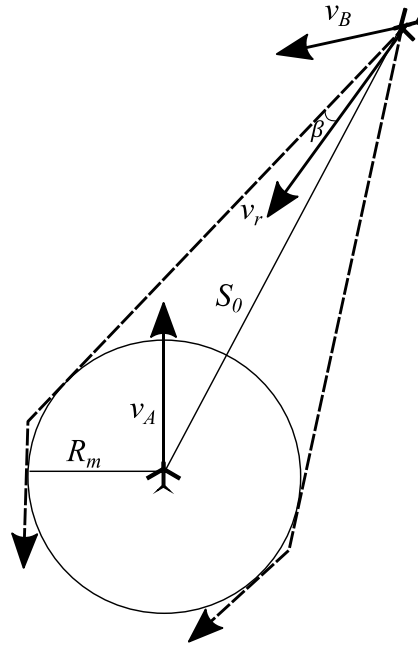
Čia: R_m – mažiausias leidžiamas atstumas tarp orlaivių jūrmylėmis; S_0 – pradinis atstumas tarp orlaivių.

Apskaičiavus reliatyvaus greičio vektoriaus krypties pokytį, galima apskaičiuoti naują orlaivio kursą konflikto sprendimui (1.2) ir atstumą, po kurio orlaivis gali grįžti į maršrutą (1.3). Gautos dvi tiesės yra aplink vieną iš orlaivių apibrėžto separacijos apskritimo liestinės. Dviejų tiesių trajektorijos taikymo konflikto sprendimui pavyzdys grafiškai pavaizduotas 1.1 pav. (brūkšninėmis linijomis žymimos galimos dvi konflikto sprendimo trajektorijos).

$$\alpha_A^* = \beta + \sin^{-1}\left[\frac{v_B}{v_A} \sin(\alpha_B - \beta)\right] \quad (1.2)$$

$$d = v_A \frac{S_0(\cos\beta + \sin\beta \tan\mu)}{v_R^*} \quad (1.3)$$

Čia: α_A^* - naujas orlaivio A kursas laipsniais; β - reliatyvaus greičio vektoriaus kryptis, išsprendžianti konfliktą, laipsniais; v_A - orlaivio A greitis mazgais; v_B - orlaivio B greitis mazgais; α_B - pradinis orlaivio B kursas laipsniais; μ - pradinės reliatyvaus greičio vektoriaus krypties ir reliatyvaus greičio vektoriaus krypties, išsprendžiančios konfliktą, skirtumas laipsniais; v_R^* - naujas reliatyvus greitis mazgais.



1.1 pav. Dviejų tiesių trajektorijos (Sudaryta autoriaus, remiantis Erzberger (2005))

Vis dėlto, tiesių metodas nėra pakankamai realus, nes gali lemti staigius orlaivio skrydžio parametrų pokyčius atkarpų galuose. Užsienio autoriai (Menon et al., 1999) aptaria galimybę šiuos pokyčius sušvelninti interpoliuojant, tačiau tai gali į skaičiavimus įvesti papildomus orlaivio pozicijos neapibrėžtumus. Tam pritaria ir R. A. Paielli's (2003) teigdamas, kad remiantis trajektorijomis, sudarytomis iš tiesių, dažnai nėra atsižvelgiama į orlaivio kurso keitimo dinamiką. Siekiant patikslinti orlaivio posūkio modelį, mokslininkas siūlo į skaičiavimus įtraukti posūkio vėlavimą (1.4) ir laiko sutaupymą suapvalinant trajektoriją, tačiau pastebi, kad pastarasis laiko sutaupymas realybėje gali būti atmetamas dėl mažos įtakos visam skrydžio laikui.

$$t_d = v^2 / g \tan 20^\circ |\tan \Delta\psi / 2| / v \quad (1.4)$$

Čia: v - orlaivio greitis mazgais; g - laisvojo kritimo pagreitis m/s^2 ; $\Delta\psi$ - orlaivio kurso pokytis laipsniais.

Dviejų tiesių trajektorija yra paprasta, nesudėtingai aprašoma, tačiau ne itin optimali trajektorija. Siekiant trajektorijos optimizavimo, jos pailgėjimo trumpinimo ir skaičiavimų

supaprastinimo, trajektoriją patogiau suskaidyti į mažesnes atkarpas, t. y., naudoti ne dvi, bet daugiau tiesių (Menon et al., 1999). Kitas būdas trajektorijai aprašyti ir sumažinti parametrų pokyčius yra kreivės. Vienos žinomiausių kreivių trajektorijų yra Dubinso trajektorijos. Žinant pradinę ir galutinę orlaivio poziciją bei kursą, galima sudaryti 4 maršrutus, paremtus skrydžio tiesiai ir posūkių į kairę bei dešinę kombinacijomis. Trumpiausias atstumas tarp dviejų taškų randamas palyginus atstumus tarp tangento apskritimų.

Dubinso trajektorijos yra gana plačiai tyrinėjamos bepiločių orlaivių skrydžiams ir stacionarių kliūčių išvengimui (Kikutis, Stankūnas, Rudinskas, & Masiulionis, 2017; Lugo-Cardenas, Flores, Salazar, & Lozano, 2014; Shanmugavel, Tsourdos, Zbikowski, & White, 2005). R. Kikučio ir jo kolegų (2017) teigimu, norint rasti orlaivio kurso pokytį α , kuris reikalingas Dubinso apskritimo centro koordinatei apskaičiuoti, reikia rasti atstumą nuo kliūties centro k bei Dubinso apskritimo (trajektorijos kreivės) spindulį R . Tai apskaičiuojama remiantis formulėmis:

$$k = \frac{R_{kl} \sin(180 - RB - \chi)}{\sin(RB)} \quad (1.5)$$

$$R = k \cdot \operatorname{tg} \frac{\gamma}{2} \quad (1.6)$$

$$\alpha = \operatorname{arctg} \frac{R}{k} \quad (1.7)$$

Čia: R_{kl} – mažiausias atstumas kliūties apskridimui; RB – reliatyvus kampas, rodantis kliūties padėtį orlaivio atžvilgiu; χ – kampas tarp atstumo k ir tiesės, jungiančios kliūtį su tašku, sudarančiu k bei kliūties apsauginės zonos sankirtą; γ – orlaivio posūkio apskritiminės trajektorijos lanko kampas.

Vis dėlto, Dubinso trajektorijų pritaikymas komercinių orlaivių skrydžiams ir konfliktų sprendimui nėra pakankamai ištirtas. Viena iš to priežasčių gali būti kreivės netolydumas trajektorijos pokyčio taškuose, kuris užsienio autorių (Shanmugavel et al., 2005) teigimu yra gana nepraktiškas fiksuoto sparno orlaiviams. Mokslininkų nuomone, šis netolygumas gali būti švelninamas interpoliuojant. Kitas būdas, leidžiantis išvengti kreivės netolydumų, yra Dubinso trajektorijos tiesių keitimas Bezjė kreivėmis (Askari, Mortazavi, Talebi, & Motamedi, 2016). Mokslininkai ištyrė, kad dėl kreivės netolydumo orlaivio skrydis tampa ne toks tikslus, kai skrydžio maršrutas yra Dubinso trajektorija. Nors pastaroji trajektorija yra trumpiausias maršrutas tarp taškų ir trumpesnė už Bezjė kreives, Bezjė kreivės yra praktiškesnės ir 2,6 % tikslesnės, be to, naudingos tokioms trajektorijoms, kurios yra ribojamos apskaičiuoto atvykimo į tašką laiko.

Apžvelgus galimas orlaivių trajektorijas galima teigti, kad dažniausiai sprendžiant konfliktus naudojamas trajektorijos tipas yra tiesės. Trajektorija sudaryta iš tiesių yra paprastai apskaičiuojama ir tradiciškai naudojama, kai trajektorijos ilgio optimizavimas nėra prioritetas. Nors Dubinso trajektorija yra trumpesnė, tiek ši, tiek tiesių trajektorijos lemia tam tikrus skrydžio parametrų keitimo

netolygumus. Interpoliavimas nėra pats geriausias netolygumų švelninimo metodas, nes gali lemti papildomą orlaivio pozicijos neapibrėžtumą, todėl turėtų būti naudojami kiti metodai, tokie kaip skrydžio dinamikos skaičiavimai (tiesių trajektorija) ar Bezjė kreivės (Dubinsko trajektorija). Vis dėlto svarbu nepamiršti, kad naudojant papildomus netolygumų švelninimo metodus iškyla trajektorijos pailgėjimo tikimybė, į kurią reikia atsižvelgti.

1.3. Konfliktų sprendimo plokštumoje metodai

Idealioje situacijoje egzistuoja keli orlaivių konflikto sprendiniai. Galutinis sprendimas priimamas atsižvelgiant į orlaivio charakteristikas, oro sąlygas, trajektorijos pailgėjimą, laiką ir kitus lemiamus veiksnius. Pavyzdžiui, nesudėtingame konflikte yra geriau keisti orlaivio kursą nei aukštį, nes tokiu atveju bus sunaudota mažiau kuro (Cafieri, 2014). Dėl šios priežasties, tokių situacijų neįmanoma spręsti tiesiogiai panaudojant Būlio algebrą, nes loginių kintamųjų yra daug. Tai patvirtina ir kiti autoriai (Durand et al., 1996), teigdami, kad vadovaujantis klasikiniais metodais negalima apskaičiuoti optimalaus sprendimo, nes jie netinkami realiam trajektorijų numatymui bei planavimui.

Kaip minėta anksčiau, orlaivių konfliktus plokštumoje galima spręsti dviem būdais. Vis dėlto, S. Cafieri'o ir R. Omheni'o (2016) manymu, greičio keitimas gali išspręsti labai mažai konfliktų, vien dėl Eurokontrolės rekomenduojamų ar vietinių greičio pokyčio ribojimų, todėl nėra pakankamai efektyvus konfliktų sprendimo būdas. Tam pritaria ir J. Krozelis ir M. Petersas (1997), teigdami, kad net ir nesant greičio pokyčio ribojimų, didelis greičio pokytis, galintis išspręsti konfliktą, užtrunka per ilgai arba yra visiškai nepraktiškas (ypač kai atstumas tarp orlaivių mažesnis nei 40 NM), todėl kurso keitimas yra žymiai efektyvesnis konflikto sprendimo būdas. Tai pabrėžia ir J. K. Kucharas ir L. C. Yangas (2000), teigdami, kad kurso keitimas yra vienas populiariausių orlaivių konfliktų sprendimo metodų dėl savo universalumo, nes įprastinėmis skrydžio sąlygomis orlaivis visada gali pasukti (priešingai greičio keitimui, kuris gali būti neįmanomas dėl orlaivio svorio ar kitų aplinkybių). Dėl šių priežasčių šiame darbe bus tiriamas tik orlaivio kurso keitimas.

Nors įvairūs įrankiai yra naudojami valdymo teorijoje ir automatizavimo bei optimizavimo procesuose jau daugelį metų, jų pritaikymo orlaivių konfliktų sprendimui galimybės pradėtos tyrinėti visai neseniai (Lovato et al., 2018). Šiame poskyryje nagrinėjamas įvairių matematinių modelių bei algoritmų pritaikymas tiek centralizuotam, tiek laisvojo skrydžio orlaivių konfliktų sprendimui plokštumoje, pateikiami šių tyrimų metodika, jos privalumai bei trūkumai, ir rezultatai.

1.3.1. Orlaivių kurso keitimo metodai

A. Sathyano ir jo kolegų (2015) pasiūlytas genetinės neraiškiosios logikos modelis, taikytas bepiločiams orlaiviams, buvo pritaikytas 5 komercinių orlaivių konfliktų sprendimui 70 NM spindulio erdvėje (Sathyan et al., 2017), nes buvo pastebėta, kad algoritmas gali gana efektyviai rasti

tiesiausius maršrutus tarp taškų. Galimi tyrimo sprendiniai buvo apriboti vienu susidūrimo išvengiančiu kurso pokyčiu vienam orlaiviui ir maksimaliu 30° posūkio kampu. Į šį tyrimą taip pat buvo įtrauktas žemiausio lygio neapibrėžtumas, galimas bet kuriuo laiko momentu. Autorių tikslas buvo sukurti neraiškiosios logikos sistemą, kuri galėtų parinkti saugius nuo konfliktų maršrutus su mažiausiu posūkio kampu. Sukurtoje sistemoje buvo panaudotas paslėptas sluoksnis įvesčių sujungimui, taip sumažinant parametrų (įvesčių), skirtų neraiškiųjų išvadų sistemai, skaičių. Neraiškiosios logikos sistemos ciklas buvo kartojamas 15 kartų kas 15 s, kaskart įvertinant orlaivių pozicijas ir konfliktų tikimybę. Taip pat aplink kiekvieną orlaivį buvo nustatyta neapibrėžtumo erdvė, kuri buvo padidinta 2,5 NM spinduliu (puse saugaus atstumo tarp orlaivių) visomis kryptimis. Tyrimo rezultatuose nurodoma, kad neraiškiosios logikos sistema gali rasti konflikto sprendimus 120 kartų greičiau nei genetinis algoritmas, bet pastarasis geba rasti šiek tiek geresnius sprendimus. Autoriai pabrėžia, kad šis modelis gali būti pritaikomas ir didesniam orlaivių skaičiui ar aukštesniam neapibrėžtumo lygiui, tačiau tai lemtų didesnes neapibrėžtumo erdves aplink orlaivius ir didesnę konfliktų tikimybę.

N. Durando ir jo kolegų (1996) darbe nagrinėjamas genetinio algoritmo panaudojimas orlaivio kurso keitimui. Tyrimo metu parengtas algoritmas sprendžia konfliktus 2D erdvėje, atsižvelgiant į greičio neapibrėžtumus. Mokslininkų teigimu, atstumo tarp dviejų trajektorijų nustatymas yra gana sudėtingas, todėl siekiant supaprastinti skaičiavimus bei patį algoritmą buvo parinkti šie orlaivių trajektorijų tipai:

1. Orlaivis nėra nukrypęs nuo maršruto (orlaivis skrenda pirmine trajektorija);
2. Orlaivis yra nukrypęs į dešinę;
3. Orlaivis yra nukrypęs į kairę;
4. Orlaivis yra nukrypęs į dešinę skrydžiui paralelia trajektorija;
5. Orlaivis yra nukrypęs į kairę skrydžiui paralelia trajektorija.

Šie trajektorijų tipai buvo kryžminami algoritmo pagalba, sprendiniai mutuojami ir lyginami su pradine situacija. Be to, genetinis algoritmas buvo patobulintas dirbtinio grūdinimo metodo pagalba, t. y. lyginant pradines trajektorijas su naujai išvestomis. Tokiu būdu galima gauti žymiai daugiau konflikto sprendimų, o tai suteikia galimybę rasti tinkamą konflikto sprendimą pakankamai greitai. Genetinio algoritmo pabaigoje sprendiniai papildomai apdorojami laiptiniu algoritmu, siekiant išgryninti geriausius sprendimus atitinkamiems nuokrypio lygiams. Šiame darbe tyrinėti tiek susikertantys, tiek maršrutai, kai orlaiviai skrenda vienas paskui kitą, ir buvo nustatyta, kad modelis geba priimti sprendimą, dėl kurio vienas ar keli orlaiviai turės nukrypti nuo pradinio maršruto ilgam laikui, jeigu tai padarius būtų galima išvengti dar didesnių nukrypimų ar vėlavimų.

Matematinis programavimas taip pat buvo taikomas šiai orlaivių konfliktų sprendimo procedūrai. S. Cafieri'o ir R. Omheni'o (2016) tyrime mokslininkai ieškojo mažiausio kurso pokyčio

orlaivių atskyrimo sąlygai (1.8) patenkinti. Mokslininkai tyrė 4 orlaivių, atsitiktinai išdėstytų ant pasirinkto spindulio apskritimo lanko ir skrendančių į apskritimo centrą, konfliktą. Dėl konfliktinės zonos vietos ir vienodų orlaivių greičių pasirinkimo, ši problema yra itin simetrinė, todėl sunkiai išsprendžiama ir nerealistiška. Dėl šios priežasties, mokslininkai savo sukurta skaičiavimų principą vėliau pritaikė orlaiviams, skrendantiems atsitiktinėmis trajektorijomis. Šiame tyrime buvo daroma prielaida, kad orlaivis skrenda pagal tolyginio judėjimo dėsnius, o orlaivio i abscisė ir ordinatė laiko momentu t aprašoma formulėmis 1.9 ir 1.10:

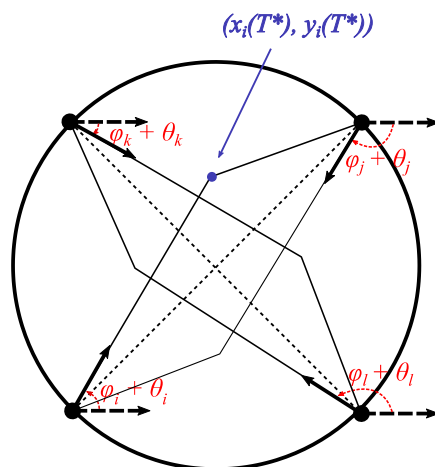
$$(x_i(t) - x_j(t))^2 + (y_i(t) - y_j(t))^2 \geq d^2 \quad \forall t \quad (1.8)$$

$$x_i(t) = x_i^0 + \cos(\phi_i + \theta_i)v_i t \quad (1.9)$$

$$y_i(t) = y_i^0 + \sin(\phi_i + \theta_i)v_i t \quad (1.10)$$

Čia: d – minimalus horizontalus orlaivių atskyrimas jūrmylėmis; $x_i(t)$ ir $y_i(t)$ – orlaivio i koordinatės laiko t momentu; θ_i – kurso pokyčio, išsprendžiančio konfliktinę situaciją, kampas orlaiviui i laipsniais; ϕ_i – orlaivio i vektoriaus projekcijos kampas laipsniais; v_i – pradinis orlaivio greitis mazgais.

Toliau mokslininkai apskaičiavo optimalų kurso pokytį θ^* (čia ir toliau žvaigždute žymima optimali vertė). Jų teigimu, orlaivio kurso pokyčiai turėtų būti kuo mažesni, išlaikant orlaivį kuo arčiau suplanuoto maršruto, todėl šis apribojimas buvo pasirinktas kaip pagrindinė tikslo funkcija. Tyrimo metu paaiškėjo, kad šio uždavinio sprendime didžiausių sunkumų kėlė netiesinė trigonometrinių funkcijų prigimtis, todėl, autorių nuomone, kintamųjų aprašymas tiesinio programavimo metodu, nors ir yra įmanomas, visgi nėra praktiškas. Svarbu nepamiršti, kad išvengus konflikto, orlaivį reikia grąžinti į jo pirminį maršrutą. Žinant orlaivio maršruto tašką ant oro erdvės ribos, galima apskaičiuoti naują trajektoriją grįžimui į maršrutą (Alonso-Ayuso et al., 2016). Pastarųjų mokslininkų pasiūlytą neapribotą kvadratinio programavimo metodą S. Cafieris ir R. Omhenis (2016) pritaikė orlaivių poroms. Šiuo metodu nustatomas optimalus laiko intervalas T^* , po kurio orlaivis gali grįžti į savo maršrutą. Taip šis metodas tampa pritaikoms laisvųjų maršrutų erdvėje, duodant pilotui leidimą skristi tiesiai į tašką, kai žinomos tų taškų koordinatės ir kurso pokytis, reikalingas tokiam manevrui atlikti. Mokslininkų siūlomas modelis grafiškai pavaizduotas 1.2 pav.



1.2 pav. Orlaivių konflikto sprendimas keičiant kursą (Cafieri & Omheni 2016)

Vis dėlto orlaivio kurso pokytis yra neiškilusis uždavinys, turintis daug lokalių minimumų. Be to, binarinių kintamųjų ir sinuso bei kosinuso netiesiškumas gali turėti įtakos globalaus sprendimo paieškos spartai ir gebai apdoroti informaciją. Dėl šios priežasties užsienio mokeslininkai (Cafieri & Omheni, 2016) numato galimybę konfliktų sprendimo algoritme kombinuoti kurso bei greičio keitimą, pirmiau taikant orlaivio greičio pokytį ir maksimizuojant orlaivių konfliktų sprendimą, remiantis pastarąja procedūra, kiekį. Tokiu būdu į modelį taip pat būtų galima įtraukti ir vienas į kitą skrendančių orlaivių konfliktus. Galiausiai, šiame darbe nebuvo atsižvelgta į dar kelis svarbius aspektus, tokius kaip greičio įtaka posūkio spartai (Wang et al., 2015) ar galima vėjo nuonaša.

Matematinis programavimas buvo naudojamas ir kitų užsienio autorių tyrimuose (Hu et al., 2002; Raghunathan et al., 2004). Šiuose darbuose optimalios trajektorijos buvo gaunamos panaudojus kinematinį orlaivio modelį ir jį pritaikius tikslo funkcijos minimizavimui. J. Hu ir jo kolegos (2002) konflikto sprendimą apibrėžė kaip trajektoriją, sudarytą iš dviejų atkarpų, pasirenkant tas atkarpas, kurių ilgių suma būtų pati mažiausia. Svarbu pažymėti, kad šiuose tyrimuose orlaivių greitis buvo nekintamas. Mokslininkų teigimu, įvedus daugiau apribojimų (maksimalaus kurso pokyčio, aerodinaminių kintamųjų ir t.t.), modelis gali būti patobulintas atmetant nerealias trajektorijas ir taip trumpinant skaičiavimams reikalingą laiką. P. K. Menono ir jo kolegų (1999) siūlomame konfliktų sprendimo remiantis laisvojo skrydžio koncepcija modelyje taip pat buvo naudojamas matematinis programavimas, tačiau šiame tyrime pagrindinis tikslas buvo ne optimali trajektorija, bet optimalus orlaivio techninių parametrų (variklio traukos, posvyrio, apkrovos ir kt.) naudojimas ir valdymas. Priešingai nei S. Cafieri'o ir R. Omheni'o (2016) tyrime, pastarųjų trijų tyrimų apskaičiuotos naujos trajektorijos yra optimesnės, nes yra sudarytos iš daugiau tiesių atkarpų arba kreivių.

E. Hoffmano ir jo kolegų (2001) tyrime orlaivių konfliktai buvo sprendžiami dinaminio programavimo pagalba. Mokslininkų teigimu, konflikto sprendimas yra nuoseklus procesas, todėl būtent dinaminis programavimas gali geriausiai apdoroti sprendimo būdus ir jų pasekmes kiekvieno

žingsnio metu. Šiame tyrime mokslininkai diskretizavo laiką ir erdvę, taip leisdami algoritmui ieškoti sprendimo laiko momentu. Taip pat buvo pasiūlyti įvairūs apribojimų modeliai, tokie kaip kurso pokyčių skaičiaus, nuokrypio nuo pradinės trajektorijos, kurso pokyčio dydžio ribojimai. Buvo pastebėta, kad taikant nuokrypio nuo pradinės trajektorijos apribojimą, algoritmas vėlindavo orlaivio kurso keitimą taip padidindamas pradinį kurso pokytį. Be to, išsprendus konfliktą orlaivis neskrisdavo tiesiai į artimiausią pradinio maršruto tašką, todėl grįžimo į atitinkamą pradinio maršruto tašką apribojimas algoritmui yra būtinas. Pasiūlytas algoritmas sugebėjo išspręsti 56 % konfliktų, tačiau tik 41 % skrydžių grįžo į savo maršrutą nenurodant papildomų grįžimo sąlygų. Dinaminio programavimo privalumus optimalių trajektorijų skaičiavimui aptarė ir kiti mokslininkai (Miyazava et al., 2013). Pastarieji mokslininkai pabrėžė dinaminių programų gebėjimą taupyti laiką, reikalingą pakartotinių skaičiavimų konvergavimui. Tyrime skrydžių erdvė buvo suskaidyta į mažesnius plotus ir ieškoma, kuriame iš jų galimas konfliktas. Vis dėlto, mokslininkai konfliktą sprendė tik orlaivių aukščio keitimu. Šiame darbe taip pat buvo atsižvelgta ir į galimą vėjo įtaką prie trijų orlaivio būsenos kintamųjų (koordinatų ir kurso) pridėdant trajektorijos kampą. Y. Miyazavos ir jo kolegų (2013) nuomone dinaminį programavimą galima taikyti ir dviejų judančių objektų konflikto sprendimui ne tik skaidant plokštumą pasirinkto tankio tinkleliu, bet ir taikant laiko žingsnius. Tai nurodoma ir kitų mokslininkų tyrimuose (Omer & Farges, 2013; Raghunathan et al., 2004), kuriuose teigiama, kad laiką suskaidant į baigtinius žingsnius (intervalus) ir skaičiavimų algoritmą taikant kiekvienam žingsniui, nebaigtinę problemą galima paversti baigtine netiesine problema.

Vis dėlto, kai kurių autorių teigiama (Luo et al., 2017), kad dinaminę problemą skaidant į paprastesnes subproblemas, sprendimo radimas gali būti gana sudėtingas. Jiems pritaria ir J. Omeras (2015), teigdamas, kad norint išlaikyti pageidautiną tikslumą, dinaminio programavimo modelyje laiko horizontas turi būti skaidomas į labai daug žingsnių, o tai lemia didelį kiekį kintamųjų bei ribojimų ir potencialiai ilgus skaičiavimus. Vis dėlto, aukšto kompleksiško problemoms spręsti dinaminis programavimas gali būti tinkamas, nes juo gali būti apjungiamas tiesiniu programavimu sprendžiamų problemų sprendimų optimalumas ir netiesinio programavimo galimybė apdoroti realią netiesinių ribojimų geometriją (Omer & Farges, 2013). Užsienio tyrėjų nuomone (Kochenderfer & Chryssanthacopoulos, 2011), dinaminio programavimo panaudojimas orlaivių konfliktams spręsti yra tinkamas, tačiau turi būti tobulinamas papildomomis priemonėmis siekiant palengvinti erdvės ar laiko diskretizavimą ir trumpinant skaičiavimų laiką. Mokslininkai siūlo hibridinį dinaminio programavimo modelį, kuriame dinaminis algoritmas yra papildomas sustiprintu mokymusi, pavyzdžiui, multi-agentų sistema (angl. *MAS – Multi-agent system*). Tokia papildoma sistema padeda spręsti itin sudėtingas problemas ir algoritmui suteikia taip galimybę kaupti informaciją apie geriausius sprendimus ir remtis jais, priimant sprendimus ateityje, taip paspartinant skaičiavimus. Algoritmas su MAS sprendžiamą problemą decentralizuoja, t. y. nėra vieno sprendimą priimančio

agento-lyderio, o problemos objektai (orlaiviai) išsaugo savo autonomiškumą. Mokslininkų siūlomas Markovo sprendimų priėmimo modelis pagerina saugumo ir efektyvumo rodiklius, tačiau kai kurių užsienio autorių teigiama (Yang & Wei, 2018), kad pastarąjį modelį naudojant erdvės diskretizavimui galimi informacijos praradimai ir sprendimo netikslumai. Dėl šios priežasties, nors pastarasis modelis tinkamas problemai aprašyti, mokslininkai siūlo dinaminį algoritmą papildomai stiprinti atsitiktinės paieškos metodais, kurie geba rasti sudėtingų problemų sprendinį, paremtą nuosekliu sprendimų priėmimu, be didelio kiekio įvesčių.

Lyginant aptartus modelius galima teigti, kad matematiniu programavimu bei genetiniu algoritmu paremti kurso keitimo modeliai gali lemti didesnius laiko, reikalingo sprendimui apskaičiuoti, kaštus nei neraiškiosios logikos sistema, tačiau pastarosios sistemos apskaičiuojami sprendimai yra daug mažiau optimalūs ar tikslūs nei genetinio algoritmo sprendimai. Visgi, genetiniui algoritmui būdinga itin didelės kompiuterinės atminties būtinybė, nes algoritmo gebėjimas spręsti problemą priklauso nuo populiacijos dydžio (Chaitanan et al., 2014), be to, geriausias sprendimas kartais gali būti tiesiog atmestas. S. Cafieris (2017) pastebi, kad norint spręsti sudėtingesnes problemas bei įtraukti tvarumo ir ekonomiškumo tikslus ar orlaivio judėjimo erdvėje neapibrėžtumus, matematinio programavimo metodai turi būti tobulinami arba naudojami kartu su kitais optimizavimo metodais. Tam pritaria ir kiti mokslininkai (Chaudhuri et al., 2015), savo tyrime siūlydami neraiškiasias funkcijas. Autorių nuomone, būtent neraiškiosios funkcijos gali būti panaudojamos matematiniame programavime, siekiant apdoroti laiko kintamuosius ir situacijai būdingus neapibrėžtumus.

Norint sumažinti laiko kaštus bei supaprastinti skaičiavimus, dinaminis programavimas yra geriausias metodas, nes problema yra skaidoma į mažesnes. Be to, kurso pokyčio ieškojimas nustatytu laiko žingsniu leidžia orlaivį nuosekliai grąžinti į jo pradinį maršrutą taip mažinant nuokrypį. Galiausiai, daugelyje aptartų tyrimų nebuvo atsižvelgta į vėjo įtaką orlaivio trajektorijai. Būtent dinaminis programavimas gali geriausiai atsižvelgti į vėjo atsiradimą laiko momentu ir pakeisti trajektoriją, reikalingą konflikto sprendimui (tuo tarpu kiti metodai apskaičiuoja galutinį konflikto sprendimą neatsižvelgdami į galimą vėjo atsiradimą, arba jį įtraukia į skaičiavimus kaip atsitiktinį greičio kintamąjį, todėl apskaičiuota trajektorija nėra pakankamai tiksli ar optimali visu skrydžio metu). Apibendrinant galima daryti išvadą, kad trajektorijos, sumodeliuotos remiantis dinaminio programavimu, gali būti sudarytos iš daugiau atkarpų (taip išlaikant orlaivį kuo arčiau pradinės trajektorijos), todėl gali būti pakankamai tikslūs, tačiau papildomas metodas kursų pokyčių sušvelninimui atkarpų galuose yra būtinas. Kita vertus, dinaminiam programavimui reikalingi papildomi modeliai, kurie galėtų sutrumpinti skaičiavimų trukmę.

1.4. Vėjo įtaka orlaivio pozicijos neapibrėžtumui

Pasak užsienio tyrėjų (Hu et al., 2003), oro masių maišymasis ir vėjo įtaka orlaivio pozicijai dažnai yra ignoruojami. Autoriai savo tyrime teigia, kad vėjo pokyčiai turi nenuginčijamą įtaką konfliktų tikimybei ir privalo būti įtraukti į naujų konfliktų aptikimo ir sprendimo sistemų kūrimą. Tam pritaria ir S. Chaimatananas ir jo kolegos (2014), savo tyrime minintys ne tik vėją, bet ir kelis kitus faktorius, tokius kaip oro temperatūra, turinčius įtakos orlaivio pozicijos neapibrėžtumui. Kaip ir A. Sathyan ir kolegų (2017) modelyje, šiame tyrime numatoma padidinta saugumo zona aplink orlaivį. Greičio neapibrėžtumas tyrime atspindimas orlaivį modeliuojant kaip uždara 3D voką, kurio dydis laikui bėgant gali kisti. Mokslininkai pabrėžia, kad neapibrėžtumų įtraukimas į modeliavimą reiškia ilgesnį laiką, reikalingą sistemai ieškant sprendimo. Kiti autoriai (Rey et al., 2012) siūlo greičio neapibrėžtumą algoritme modeliuoti kaip atsitiktinį greičio kintamąjį nustatytame paskirstyme, centruotame ties pradine orlaivio greičio pokyčio instrukcija. Tokiu atveju greičiai paverčiami intervalais, kurie sukuria naujus orlaivių trajektorijų susikirtimo laikus.

Neapibrėžtumų įtaka sprendžiant orlaivių konfliktus nagrinėjama N. Durand ir jo kolegų (1996) vykdytame tyrime. Siekiant įtraukti neapibrėžtumus į modeliavimą, orlaiviai yra modeliuojami kaip taškas laiko momentu, kuris vėliau virsta atkarpa neapibrėžtumo kryptimi (greičio vektoriumi). Atstumai tarp šių atkarpų yra nuolatos matuojami ir lyginami su žinomu minimaliu saugiu atstumu. Priešingai nei anksčiau minėtų mokslininkų modeliuose, šiame tyrime nėra papildomos saugumo zonos aplink orlaivius. Deja, autorių teigimu, dėl greičio neapibrėžtumų sistemos numatytas konfliktas nebūtinai yra realus, todėl paskubintas konflikto sprendimas gali nukreipti orlaivį į kitą konfliktinę situaciją. Be to, bandymu buvo įrodyta, kad net ir išvengus naujo konflikto, ankstyvas konflikto sprendimas reikštų staigesnį ir didesnį kurso pakeitimą. Šį neapibrėžtumų fenomeną, esant būtinybei, siūloma spręsti padidinant matavimų dažnį.

Galima daryti išvadą, kad aptartuose tyrimuose naudojami atsitiktiniai dydžiai turi didelę įtaką tolimesniems skaičiavimams ir lemia itin netikslas trajektorijas. Vis dėlto, žinant vėjo kryptį ir greitį, šie parametrai gali būti lengvai įtraukiami į orlaivio judėjimo skaičiavimą, remiantis formule (Girardet, Lapasset, Delahaye, & Rabut, 2014):

$$\begin{cases} \dot{x}(t) = v \sin(\theta(t)) + W_x(x, y) \\ \dot{y}(t) = v \cos(\theta(t)) + W_y(x, y) \end{cases} \quad (1.11)$$

Čia: (x, y) – orlaivio pozicija; θ – orlaivio kurso kampas laipsniais; v – orlaivio greitis mazgais; $W_x(x, y)$ – vėjo rytinė dedamoji; $W_y(x, y)$ – vėjo šiaurinė dedamoji.

Esant reikiamybei, orlaivio greičio žemės atžvilgiu skaliarinė išraiška gali būti apskaičiuojama (Hurter et al., 2014):

$$v_g = v_t \cos(\alpha - \theta) + W \cos(\varphi - \theta) \quad (1.12)$$

Čia: v_g – orlaivio greitis žemės atžvilgiu mazgais; v_t – orlaivio tikrasis greitis mazgais; α – orlaivio kurso kampas laipsniais; θ – trajektorijos kampas laipsniais; W – vėjo greitis mazgais; φ – vėjo kryptis laipsniais.

Apžvelgus mokslininkų siūlomus būdus greičio neapibrėžtumui dėl vėjo įtakos įtraukti į orlaivių konfliktų sprendimo modelį galima teigti, kad nekintamo dydžio saugumo zonos apibrėžimas aplink orlaivius nėra pats geriausias metodas, nes jis neatspindi pačios neapibrėžtumo esmės – kitimo laikui bėgant. Atsitiktinio greičio vektoriaus modeliavimas taip nėra realybę atspindintis metodas, nes vėjo įtaka trajektorijos kampui gali būti didesnė nei orlaivio greičiui. Taigi, norint gauti kiek įmanoma tikslesnes trajektorijas ir žinant vėjo kryptį bei greitį, vienas geriausių metodų yra vėjo sudedamųjų įtraukimas į pozicijų skaičiavimą laiko momentu.

1.5. Pirmojo skyriaus išvados

Atlikus literatūrinę analizę, buvo padarytos šios išvados:

1. Automatizuojant orlaivių konfliktų sprendimą šio proceso optimizavimo problema išlieka aktuali.
2. Mokslininkai sutaria, kad klasikiniai metodai šiai problemai spręsti nėra tinkami ir iš apžvelgtų darbų matoma, kad nėra matematinio modelio, kuris spręstų orlaivio konfliktą ir stengtųsi palaipsniui grąžintų orlaivį į pradinį maršrutą nepažeisdamas saugaus atstumo.
3. Didžioji dalis tyrimų neatsižvelgia į vėjo įtaką skrydžio trajektorijai konflikto sprendimo metu arba ją modeliuoja per daug netiksliai.
4. Atsižvelgus į aptartą literatūrą, nutarta sukurti orlaivių konflikto plokštumoje sprendimo algoritmą, kuris spręsdamas konfliktą nuosekliai ieškos galimų kurso pokyčių orlaivio grąžinimui į pradinį maršrutą taip minimizuojant trajektorijos ilgėjimą.
5. Algoritmas bus kuriamas remiantis dinaminio programavimu, nes šis metodas geriausiai atspindi dinaminę problemos pobūdį, geba reaguoti į situacijos pokyčius realiu laiku, bei bus ieškoma būdo, kaip patobulinti dinaminį algoritmą siekiant supaprastinti skaičiavimus.
6. Vėjo įtaką nutarta modeliuoti jo dedamąsias įtraukiant į orlaivio pozicijos skaičiavimus laiko momentu dėl kuo tikslesnės trajektorijos skaičiavimo.

2 ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMOJE MATEMATINIO MODELIO TEORINĖ DALIS

2.1 Dinaminis programavimas

Bet kokia optimizavimo problema visuomet turi tam tikrą tikslą – pasirinkto problemos parametro vertės minimizavimą ar maksimizavimą. Šių parametrų komponentės dar vadinamos problemos kintamaisiais, o tikslas yra aprašomas matematine tikslo funkcija. Aprašant kintamuosius itin svarbu atitikti apibrėžtas suformuluoto uždavinio sąlygas. Atitinkantys sąlygas kintamieji sudaro leistiną sritį, apibrėžtą ribojimų funkcijų reikšmėmis. Atsižvelgus į šiuos reikalavimus, minimizavimo uždavinys aprašomas formule 2.1 (Žilinskas, 2005). Analogiškai aprašomas ir maksimizavimo uždavinys.

$$F = \min_{X \in A} f(X) \quad (2.1)$$

Čia: X – parametro komponentės; A – leistinoji sritis; $f(X)$ – tikslo funkcija.

Dinaminis programavimas yra tiek matematinio optimizavimo, tiek programavimo metodas. Šio metodo esmė yra sudėtingos problemos skaidymas į paprastesnes subproblemas, t. y. sprendimo žingsnių sekos sudarymas. Kadangi tokia seka remiasi žingsniais laiko momentu, algoritmas turi saugoti sprendimų priėmimo informaciją ir stebėti situacijos pokyčius bėgant laikui. Norint priimti teisingą sprendimą, reikalinga informacija apie dabartinę situaciją, vadinama sistemos būsena. Būsenos vertės laiko momentu gali būti randamos atliekant atbulinius skaičiavimus, aprašomus Belmano lygtimi. Šie skaičiavimai galimi diskretaus laiko optimizavimo problemoms. Belmano lygtis aprašo dviejų vertės funkcijų santykį:

$$V(x) = \max_{a \in \Gamma(x)} \{F(x, a) + \beta V(T(x, a))\} \quad (2.2)$$

Čia: $V(x)$ – vertės funkcija; a – galimas veiksmas; Γ – veiksmų leistinoji sritis; $F(x, a)$ – atlygio funkcija pasirinkus veiksmą a ; $T(x, a)$ – naujos būsenos funkcijas pasirinkus veiksmą a ; β – nuolaidos faktorius.

Remiantis dinaminio programavimo teorija, šio tyrimo svarbiausia tikslo funkcija yra aprašoma formule:

$$\min F = k_1 \frac{d(t)}{7} + k_2 \gamma \quad (2.3)$$

Čia: k_1 ir k_2 – svertiniai koeficientai; $d(t)$ – atstumas tarp orlaivių laiko momentu, NM; γ – kampas tarp pradinės orlaivio trajektorijos ir orlaivio kurso, laipsniais.

Ši tikslo funkcija atitinka darbo tikslą, nes aprašo trajektorijos ilgio ir nuokrypio nuo pradinės trajektorijos kampo minimizavimą. Ši formulė taip pat tiesiogiai siejasi su atlygio vertės laiko momentu formule, kuri prioritetizuoja kuo mažesnę atstumo nuo tikslo būsenos.

Suskaidžius sudėtingą problemą į mažesnes iškyla problemų pasikartojimo galimybė. Vienas dinaminio programavimo privalumų yra gebėjimas išsaugoti sprendimus taip paspartinant skaičiavimus. Egzistuoja du problemos sprendimo ir rezultatų saugojimo būdai:

- Viršus-apačia (angl. *top-down*) – subproblemų sprendimai yra saugomi lentelėje. Sprendžiant naują subproblemą, lentelė yra patikrinama ir jei joje sprendimas yra randamas, jis yra tiesiogiai pritaikomas. Priešingu atveju, subproblema išsprendžiama ir jos sprendimas išsaugomas lentelėje.
- Apačia-viršus (angl. *bottom-up*) – subproblemos yra sprendžiamos pirmiausia ir jų sprendimai yra panaudojami didesnių problemų sprendimams. Mažųjų subproblemų sprendimai taip pat gali būti saugomi lentelėje ir naudojami didesnių problemų sprendimų apskaičiavimui.

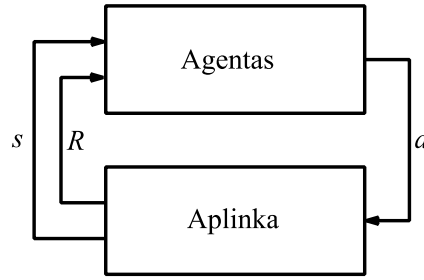
Šiame tyrime taikomas apačia-viršus rezultatų saugojimo būdas, aprašomas kaip atgalinio sklidimo procesas Monte Karlo paieškoje.

2.2 Markovo sprendimų priėmimo procesas

Markovo sprendimų priėmimo procesas yra vienas paprasčiausių sprendimo priėmimo matematinių modelių. Šis procesas yra sudarytas iš elementų:

- Aplinka, kurioje veikia agentai;
- Agentų būsenos s ;
- Galimi agentų veiksmai a ;
- Agento būsenos pokytis T ;
- Agento veiksmo įvertinimas ir atlygis R .

Šio modelio veikimo principas yra paremtas agentu patalpinimu į pasirinktą aplinką, kurioje jis gali atlikti iš anksto aprašytus veiksmus pasirinktu laiko žingsniu. Šie veiksmai ir jų pasekmės yra įvertinamos atsižvelgiant į agento būsenos pokytį aplinkoje. Šis pokytis yra siunčiamas agentui ir aplinkai kartu su atinkamu veiksmo įvertinimu. Markovo elementų sąryšis matomas 2.1 pav. Markovo proceso tikslas sprendžiant problemą yra optimalios strategijos π (angl. *policy*), kuri maksimizuočių kaupiamąjį atlygį, radimas.



2.1 pav. Markovo elementų sąryšis (Sudaryta autoriaus, 2020)

Šiame tyrime agentu yra laikomas orlaivis, o jo būseną aprašoma orlaivio pozicija, greičiu ir kursu. Orlaivio kursas ir greitis patys savaime nėra itin svarbūs orlaivio būsenai laiko momentu aprašyti, tačiau yra būtini būsenos pokyčiui įvertinti, todėl yra laikomas orlaivio būsenos dalimi.

Galimi orlaivio veiksmai apibūdinami galimu skrydžio parametro pokyčiu, šiame tyrime – orlaivio kurso pokyčiu iš leistinosios veiksmų erdvės A . Orlaiviui atlikus veiksmą, jo būsenos pokyčiui įvertinti naudojamas paprasčiausias orlaivio kinematinis modelis, aprašomas:

$$\begin{aligned}\dot{x} &= v \cos \varphi \\ \dot{y} &= v \sin \varphi \\ \dot{\varphi} &= a\end{aligned}\tag{2.4}$$

Čia: a – pasirinkta veiksmo vertė iš A veiksmų erdvės, laipsniais; v – orlaivio greitis mazgais.

Įvertinus būseną orlaiviui galima suteikti atlygį, aprašoma atlygio R funkcija:

$$R(s) = \begin{cases} 1, & \text{jeigu } s \text{ yra tikslo būseną} \\ 0, & \text{kitais atvejais} \end{cases}\tag{2.5}$$

Taigi, funkcija skiria nuobaudą ($R = 0$), jeigu orlaivis per daug suartėja su kitu orlaiviu arba palieka tiriamą plokštumą prieš pasiekdamas savo tikslo poziciją ir užbaigia skaičiavimų procesą suteikdama apdovanojimą ($R = 1$), kai orlaivis pasiekia galutinį maršruto tašką. Visi tarpiniai apdovanojimai, kurių vertės mažesnės už 1, skatina orlaivį kuo greičiau pasiekti savo tikslą, nes orlaivis yra veikiamas bendrojo agento tikslo gauti kuo didesnę apdovanojimą. Atlygio vertė laiko momentu, kai orlaivis nėra pasiekęs nei vienos iš 3 galimų galutinių būsenų, apskaičiuojama (Yang & Wei, 2018):

$$\tilde{R}(s) = 1 - \frac{d(o,g)}{\max d(o,g)}\tag{2.6}$$

Čia: $d(o,g)$ – atstumas tarp orlaivio ir jo tikslo pozicijos, NM; $\max d(o,g)$ – maksimalus atstumas tarp orlaivio ir jo tikslo pozicijos, NM. Maksimalus galimas atstumas pasirinktoje tyrimo plokštumoje yra plokštumą apibrėžiančio stačiakampio įžambinės ilgis.

Markovo modelyje itin svarbus ir laiko kintamasis. Nors laiko žingsnis savaime nėra laikomas proceso elementu, jis yra esminis dinaminio programavimo bruožas, leidžiantis pasirinktu tikslumu vertinti būsenos pokyčius ir rezultatus. Racionalus laiko žingsnio pasirinkimas turi didelę įtaką galutiniam rezultatui. Nuo jo priklauso ne tik skaičiavimų trukmė ir jos derinimas su leistinu maksimaliu sprendimo priėmimo laiko limitu, bet ir simuliacijų, kurias gali atlikti algoritmas, skaičiui. Pastarasis skaičius, kartu su simuliacijų gyliu yra svarbūs kriterijai algoritmo sustiprintam mokymuisi. Simuliacijų gylis, nurodantis kokį kiekį simuliacijos žingsnių reikia atlikti ieškant geriausio sprendimo ir apibrėžiantis, kada gautas rezultatas tenkina jam taikomus reikalavimus, taip pat gali ženkliai padidinti ar sumažinti skaičiavimų laiko kaštus. Dėl šių priežasčių modeliuojant tyrimo problemą yra itin svarbu atsižvelgti kaip kiekvienas proceso elementas ir kintamasis paveiks tiek laiko, tiek ir vėliau gauto sprendimo optimalumą.

2.3 Monte Karlo paieškos algoritmas

Monte Karlo paieškos algoritmas (angl. *MCTS – Monte Carlo Tree Search*) yra heuristinis paieškos algoritmas, taikomas sprendimų priėmimo procesams ir itin tinkamas spręsti problemoms, ribojamoms laiko. Šis algoritmas remiasi Monte Karlo metodo atsitiktinumo principu ir išplečia paiešką atlikdamas atsitiktinius skaičiavimus. Šio algoritmo esmė ištirti kuo daugiau įmanomų veiksmų ir jų galutinių rezultatų, galiausiai pasirenkant geriausią. Vienas didžiausių algoritmo privalumų yra jo pritaikymas problemoms su didele paieškos erdve. Apjungiant dinaminį programavimą su šiuo paieškos algoritmu, būtų galima ištirti daugiau įvairių modelių sutrumpinant skaičiavimų laiką skrydžio metu, dinaminio programavimo skaičiavimus galima atlikti iš anksto ir išsaugoti jų rezultatus, kuriuos vėliau galima panaudoti paieškos algoritme (Kochenderfer & Chryssanthacopoulos, 2011).

MCTS algoritmas atlieka daug simuliacijų pasirinktoje dabartinėje agento būsenoje įvertindamas tolimesnes būsenos-veiksmo vertes Q . Pastarosios vertės apibūdina gauto atlygio sumą, kai tam tikroje būsenoje s pasirenkamas veiksmas a . Pagrindiniai iteracijų žingsniai MCTS algoritme yra šie:

- Pasirinkimas (angl. *selection*) – atliekant skaičiavimus pasirenkamas taškas (angl. *node*) z , kuris maksimizuoja algoritmo rezultatą (Yang & Wei, 2018):

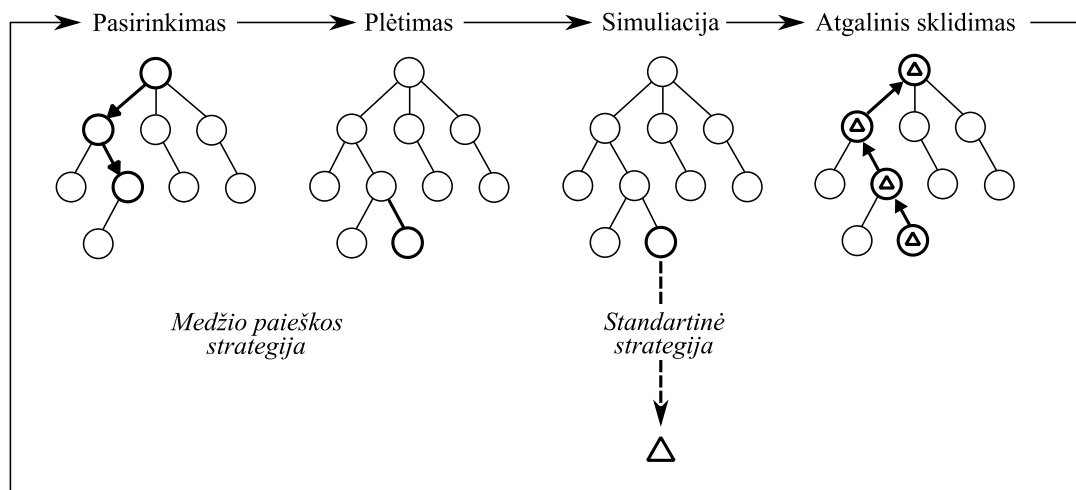
$$MCTS = \overline{X_z} + 2C \sqrt{\frac{2 \ln n}{n_z}} \quad (2.7)$$

Čia: $\bar{X}_j$ – normalizuota būsenos-veiksmo vertė taške z ; n – skaičius, nurodantis kiek kartų buvo aplankyta dabartinės būsenos taškas; n_z – skaičius, nurodantis kiek kartų buvo aplankytas būsenos taškas z ; C – teigiama konstanta.

Pirmoji formulės dalis nurodo išnaudojimą (angl. *exploitation*), paremtą turima informacija apie situaciją laiko momentu, antroji formulės dalis aprašo tyrinėjimą (angl. *exploration*), skirtą rinkti papildomai informacijai apie galimus veiksmus.

- Plėtimas (angl. *expansion*) – remiantis galimais veiksmais ir paieškos medžio strategija į paiešką pridedama daugiau taškų.
- Simuliacija – naujai pridėtiems taškams atliekami skaičiavimai pagal standartinę strategiją siekiant gauti naujus rezultatus.
- Atgalinis sklidimas (angl. *backpropagation*) – gauti simuliacijos rezultatai yra pritaikomi simuliacijoje naudotiems taškams taip atnaujinant jų statistines vertes.

Viena MCTS algoritmo iteracija pavaizduota 2.2 pav.



2.2 pav. MCTS algoritmo iteracija (Sudaryta autoriaus, remiantis Yang ir Wei (2018))

Orlaivių konfliktų sprendime MCTS paieškos medžio taškais yra laikomos orlaivio būsenos, t. y. orlaivio pozicijos laiko momentu. Šie taškai tarpusavyje yra susiję orlaivio kurso keitimo veiksmais iš veiksmų erdvės A , dėl šios priežasties kiekvienas būsenos taškas gali turėti tik 3 tolimesnius būsenos taškus. Remiantis šia logika sudarytas MCTS algoritmas šiai problemai spręsti pavaizduotas pav.

Pasirinkimo žingsnyje algoritmas parenka tolimesnį tašką pagal formulę 2.7. Šioje formulėje išnaudojimo išraiška $\bar{X}_j$ išvedama iš formulės (Yang & Wei, 2018):

$$\bar{X}_j = Q(z)/N(z) \quad (2.8)$$

Čia: $N(z)$ – skaičius, nurodantis kiek kartų šis būsenos taškas buvo aplankytas anksčiau; $Q(z)$ – visų tirtų trajektorijų, kertančių šį būsenos tašką, atlygių suma.

Tyrinėjimo išraiška priklauso nuo C konstantos. Svarbu pabrėžti, kad žemesnė C vertė reikš didesnės svarbos suteikimą taškų išnaudojimui, o aukštesnė suteiks didesnę svarbą taškų tyrinėjimui. Remiantis mokslininkų tyrimų rezultatais (Kocsis & Szepesvari, 2006), šiame tyrime $C = 1/\sqrt{2}$. Aptikus tašką, kuris dar nėra įtrauktas į skaičiavimus, jam suteikiama kuo didesnė įmanoma vertė, taip leidžiant algoritmui bent kartą įvertinti visus taškus.

Plėtimo žingsnyje pridedant naujus būsenos taškus, jų aplankymo skaičius lygus 1, o kaupiamasis atlygis lygus 0. Tolimesniame simuliacijos žingsnyje, naujai pridėto būsenos taško vertė nustatoma simuliuojant trajektoriją iki vienos iš galutinių būsenų. Šiame žingsnyje naudojamas atsitiktinis veiksmo parinkimas ir tolimesnė orlaivio būsena nustatoma pagal kinematinį modelį. Pasiekus galutinę būseną, trajektorija yra įvertinama. Šiame žingsnyje svarbu atlikti daug simuliacijų, ištiriant didelį kiekį galimų veiksmų ir būsenų taškų, nes tik tokiu atveju bus galima pasirinkti veiksmus ir trajektorijas, pasiekiančias tikslo būseną ir sukaupiančias didesnę vertę. Atlikus simuliaciją iki vienos galutinės būsenos, galutinis atlygis turi būti priskiriamas naujai pridėtiems būsenos taškams ir atgalinio sklidimo žingsnyje taip pridedamas ir pirminiams būsenos taškams.

2.4 *Python* programavimo kalba

Python yra interpretuojanti, aukšto lygio programavimo kalba, sukurta G. van Rossumo 1990 m. Vienas esminių šios programavimo kalbos bruožų yra platus jos panaudojimas. Per kelis savo gyvavimo dešimtmečius, *Python* buvo taikoma grafiniame dizaine, versle, moksle kompiuterinių žaidimų bei operacinių sistemų kūrimo, įvairių prototipų vystyme ir kt. Dažnai lyginama su kitomis programavimo kalbomis, *Python* pasižymi keletu privalumų:

- Trečiųjų šalių modulių pritaikymas – *Python* geba naudotis įvairiomis programavimo kalbomis parašytais moduliais. Dėl šios priežasties, *Python* programavimo kalba gali sąveikauti su kitomis programavimo kalbomis ir platformomis taip išplečiant jos panaudojimo galimybes.
- Išsamios pagalbinės bibliotekos – *Python* programavimo kalba remiasi standartine pagalbine biblioteka, kurioje įtraukta didelė dalis dažniausiai naudojamų programinių užduočių, tokių kaip interneto protokolai, veiksmai su skaičių eilutėmis, operacinių sistemų sąsajos ir kt. Be to, *Python* kalbai būdingos ir vartotojui draugiškos duomenų struktūros. Naudojantis pagalbinėmis bibliotekomis ir įdiegtomis struktūromis galima reikšmingai sutrumpinti rašomo kodo ilgį.

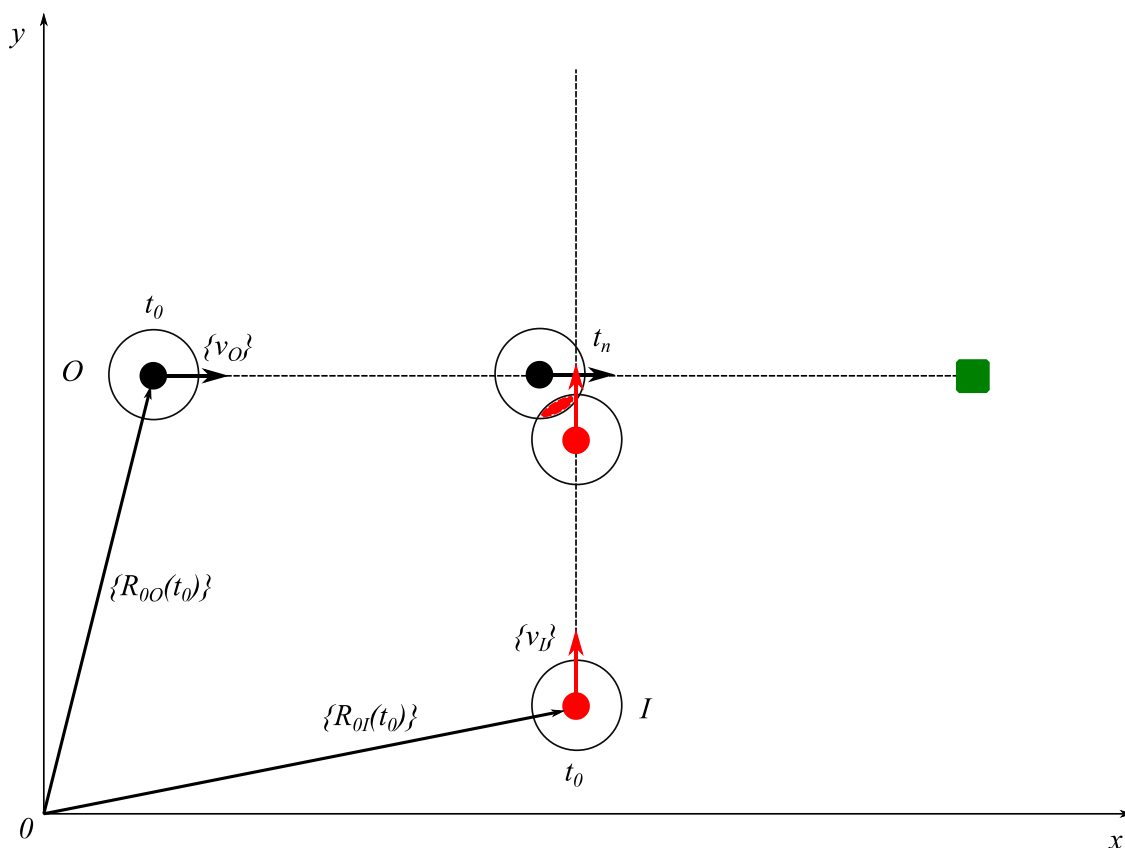
- Atviras kodas – kadangi *Python* kalba yra atviro kodo programavimo kalba, ja galima naudotis laisvai ir nemokamai.
- Sintaksės paprastumas – *Python* sintaksė remiasi aiškiomis ir paprastomis loginėmis taisyklėmis, ja parašytas kodas yra lengvai skaitomas ir suprantamas, o programavimo pagrindai nesunkiai įsisavinami. Dėl šios priežasties *Python* programavimo kalba yra itin populiari tarp neprofesionalių programuotojų.
- Produktyvumas ir greitis – *Python* pasižymi teksto apdorojimo ir patobulinto valdymo galimybėmis, taip pat ir savo vidinio testavimo sistema. Šios savybės kartu su paprasta sintakse gerina kodo greitį bei produktyvumą, todėl *Python* programavimo kalba itin tinkama aukšto kompleksiškumo problemoms spręsti.

Aukšto lygio simuliacijoms ir dinaminio programavimo apdorojamiems informacijos kiekiams būtina programavimo kalba, gebanti paprastai ir greitai atlikti skaičiavimus. Dėl šios priežasties ir anksčiau išvardintų *Python* privalumų, būtent ši programavimo kalba buvo pasirinkta tyrimo problemai spręsti.

3 ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMOJE MATEMATINIS MODELIS

3.1 Tiriama situacija ir bendroji sprendimo logika

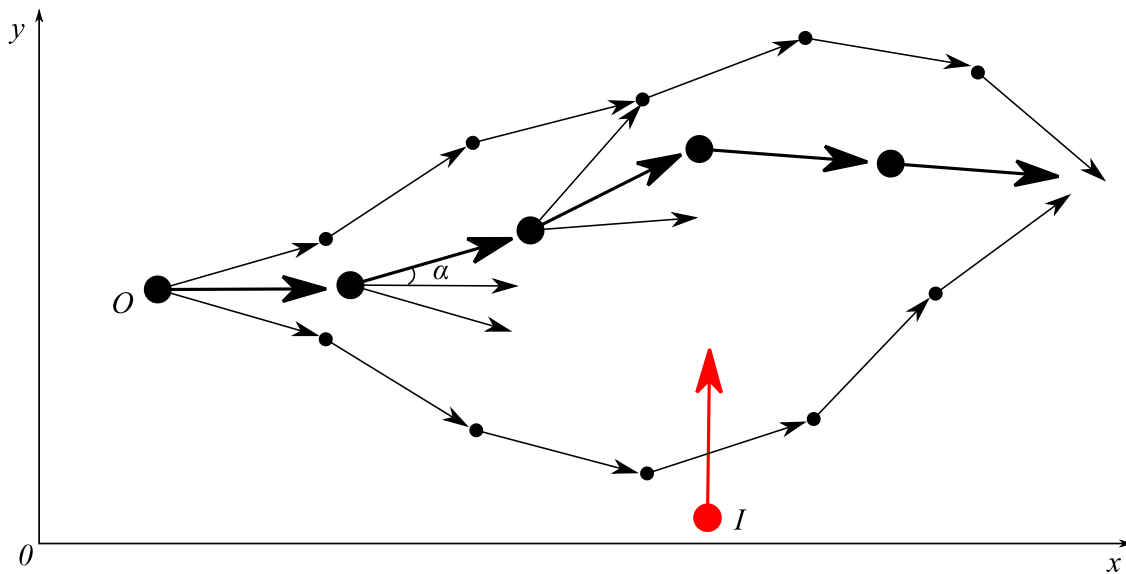
Algoritmu tiriamos dviejų orlaivių konfliktinės situacijos. 3.1 pav. vaizduojami valdomo orlaivio (žymimas O , laiko vieneto vektorius juodos spalvos) ir orlaivio įsibrovėlio (žymimas I , laiko vieneto vektorius raudonos spalvos) maršrutai. Valdomo orlaivio tikslas yra pažymėtas žaliu kvadratu, o orlaivio įsibrovėlio tikslas nėra žinomas, tačiau remiamasi prielada, kad orlaivis įsibrovėlis išlaikys savo pradinį kursą visos tiriamos skrydžio dalies metu. Aplink orlaivius apibrėžti apskritimai žymi separacinę zoną lygios pusei minimalios separacijos (toliau tyrime – 3,5 NM). Nesiėmus jokių išvengiamųjų veiksmų, laiko momentu t_n minimali separacija bus prarasta, t. y. bus mažesnė nei 7 NM, ir orlaivių separaciniai apskritimai persidengs, nepriklausomai nuo pasirinkto orlaivių konflikto tipo (susikertančių maršrutų, orlaivių, skrendančių vienas paskui kitą ir t. t.).



3.1 pav. Valdomo orlaivio ir orlaivio įsibrovėlio konfliktas (Sudaryta autoriaus, 2020).

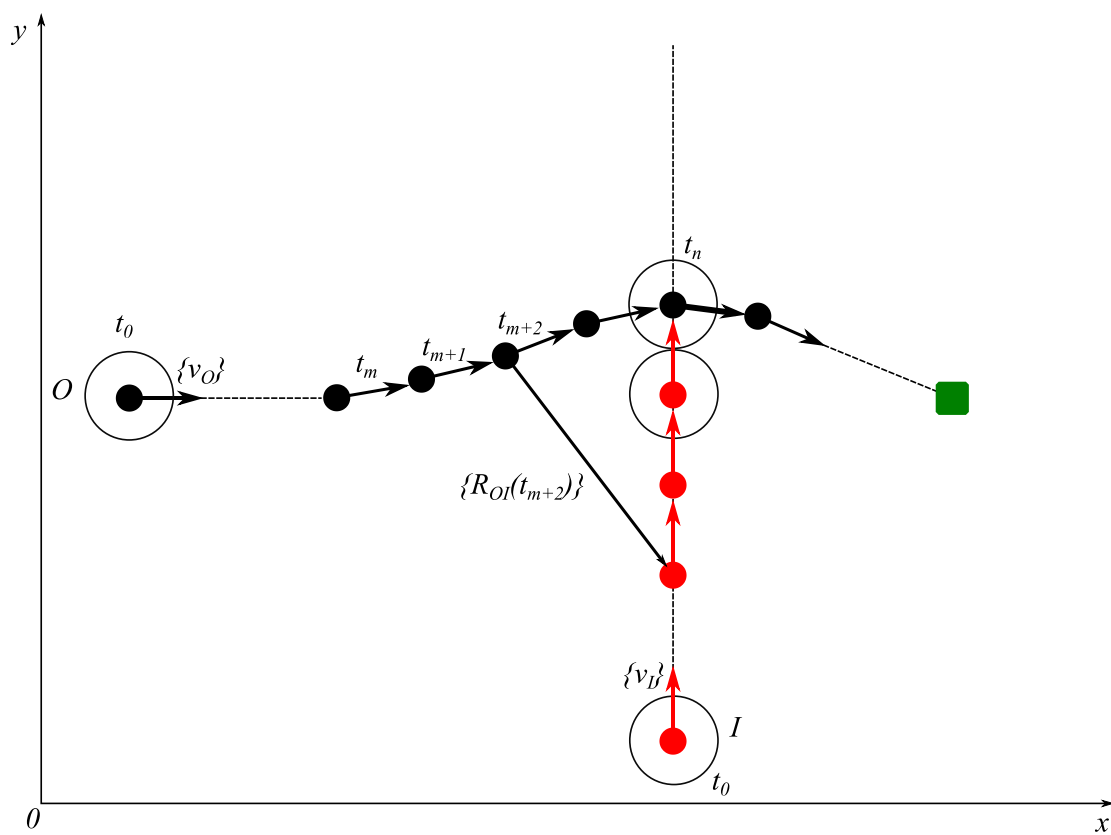
Algoritmo apskaičiuojamas sprendimas yra paremtas būsimos trajektorijos ilgio vertinimu pasirinktu žingsnių kiekiu į priekį, taip parenkant geriausią sprendimą laiko momentu. Šis principas matomas 3.2 pav. Paryškinta numatyta trajektorija yra trumpiausia, todėl pirmas veiksmas a yra skristi tiesiai ($\alpha = 0^\circ$). Tolimesniame žingsnyje algoritmas vėl atsižvelgia į tris galimus veiksmus,

perskaičiuoja numatomą trajektoriją, jei reikia, pakeičia ankstesnio ciklo metu numatytą veiksmą ir priima sprendimą, pavyzdžiui, sukti į kairę ($\alpha = -1^\circ$).



3.2 pav. Veiksmo pasirinkimas ir trajektorijos numatymas (Sudaryta autoriaus, 2020)

Taikant pastarąjį trajektorijų tyrimo principą, apskaičiuojama galutinė išvengiamoji trajektorija. Supaprastintas tokio algoritmo sprendimo pavyzdys matomas 3.3 pav.



3.3 pav. Valdomo orlaivio ir orlaivio įsibrovėlio konflikto sprendimas (Sudaryta autoriaus, 2020).

Algoritmo pasirinktu laiko momentu t_m pradedami išvengiamieji manevrai ir valdomas orlaivis atlieka vieną iš galimų veiksmų. Kadangi algoritmas yra dinaminis, tolimesni valdomo orlaivio veiksmai yra atliekami pasirinktu laiko žingsniu. Taigi, laiko momentu t_{m+1} algoritmas įgyvendina kitą racionalų sprendimą remiantis aukščiau aprašytu trajektorijos numatymo principu. Šiame tyrime racionali laikomas toks sprendimas, kuris užtikrina saugų atstumą tarp orlaivių (šis atstumas nuolatos tikrinamas $\{R_{OI}\}$) ir mažina valdomo orlaivio nukrypimą nuo pradinės trajektorijos bei naujos trajektorijos ilgį. Laiko momentu t_n , kai atstumas tarp orlaivių yra mažiausias, paveiksle 3.3. matomas konflikto sprendimas, nes separacijos apskritimai nepersidengia, vadinasi, išlaikoma minimali 7 NM separacija. Valdomo orlaivio veiksmai dinamiškai renkami tol, kol orlaivis gali nebetrunkdomas skristi tiesiai į savo tikslą. Svarbu pabrėžti, kad nors paveiksle pavaizduotas tik vienas konflikto sprendimas, remiantis apžvelgta mokslinė literatūra, visada egzistuoja du galimi konflikto sprendimai. Vis dėlto, priklausomai nuo pasirinktų algoritmo veikimo parametrų, algoritmas parenka geresnį (trumpesnės trajektorijos) sprendimą, o kai abu sprendimai identiški – sprendimas parenkamas atsitiktinai. Plačiau algoritmo veikimo principas aprašomas 3.4. skyrelyje.

3.2. Tyrimo prielaidos ir apribojimai

Atliekant šį tyrimą remiamasi šiomis prielaidomis ir apribojimais:

1. Orlaiviai skrenda nuspėjamai ir dalinasi informacija ADS-B ryšiu. Šio tyrimo tikslas yra ištirti paprasčiausią situaciją, todėl galimas orlaivių pozicijos neapibrėžtumas ir nenumatyti skrydžio parametrų pokyčiai nėra nagrinėjami.
2. Orlaiviai savo pradinuose maršrutuose skrenda tiesiai ir pastoviu greičiu. Siekiant supaprastinti algoritmą, darbe tiriami ne pilni orlaivių maršrutai, o tik jų tiesaus skrydžio atkarpos, nes tokiose atkarpose orlaivių konfliktai yra lengviausiai nustatomi, todėl tinkami siūlomo modelio efektyvumo tyrimui.
3. Siūlomas modelis numatomas vidutinio nuotolio orlaivių konfliktams spręsti. Orlaiviai vienas kitą pastebi 200 NM ir mažesniu atstumu. Vidutinė viršutinė ADS-B ryšio veikimo riba yra ~250 NM priklausomai nuo vietovės reljefo, todėl siekiant universalumo šiame tyrime pasirinktas kiek mažesnis veikimo atstumas. Mažiausias atstumas tarp orlaivių, kai valdomas orlaivis privalo pradėti išvengiamuosius manevers yra 40 NM. Priklausomai nuo orlaivių greičių, 40 NM atstumas suteikia mažiausiai 5 min iki orlaivių atskyrimo pažeidimo. Šis laiko tarpas yra pakankamas išvengti konfliktui ir neiššaukia skrydžių valdymo sistemų vidutinio nuotolio orlaivių konflikto įspėjimo, t. y. išvengiamieji manevrai bus apskaičiuoti ir pradėti vykdyti prieš skrydžių vadovui imantis veiksmų.

4. Tyrime nenagrinėjami orlaivių greičių ekstremumai, t. y. itin lėti arba labai greiti (pavyzdžiui, naikintuvai) orlaiviai, nes norima ištirti modelio pritaikymą įprastiniams komerciniams orlaiviams, skrendantiems regiono skrydžių valdymo erdvėje.
5. Tik vienas orlaivis gali atlikti išvengiamuosius manevrus. Šis apribojimas susiaurina tiriamą erdvę.
6. Galimas tik horizontalus konflikto sprendinys, t. y. orlaiviai skrenda tame pačiame aukštyje ir gali keisti tik kursą. Šio apribojimo tikslas – susiaurinti tyrimą, be to, apžvelgtoje mokslinėje literatūroje teigiama, kad kiti konflikto sprendiniai yra neefektyvūs.
7. Galimi kurso pokyčiai yra apriboti leistinąja kurso pokyčių erdve. Šis apribojimas susiaurina tiriamą erdvę ir sutrumpina skaičiavimų laiką.
8. Atlikus pasirinktą kiekį veiksmų, orlaivis privalo skristi tiesiai pasirinktą laiko tarpą. Tai leidžia algoritmą supaprastinti ir pritaikyti jo skaičiavimus ilgus atstumus bei dideliu greičiu skrendantiems orlaiviams.
9. Konflikto sprendimo metu orlaivių greitis išlieka pastovus. Šis apribojimas susiaurina tiriamą erdvę sumažindamas galimų veiksmų skaičių ir sutrumpina skaičiavimų laiką.
10. Darom prielaida, kad orlaivis iškart pasisuka reikiamu kursu, t. y. tyrime neatsižvelgiama į posūkio spindulį, nes tiriamos posūkio vertės yra labai mažos.
11. Geriausiu sprendimu laiko momentu visada laikomas skrydis tiesiai. Valdomam orlaiviui artėjant prie savo tikslo, visi galimi veiksmai (skrydis tiesiai, posūkis į kairę arba dešinę) gauna vienodą atlygį, todėl siekiant išvengti beprasmių posūkių, turi būti pasirenkamas skrydis tiesiai. Nepaisant to, kad orlaivį viso skrydžio metu norima palaipsniui grąžinti į maršrutą, pastarasis principas taikomas visame algoritme, nes jo supaprastinimui reikalinga bendra skaičiavimų logika.
12. Orlaiviai laikomi ne konflikto būsenoje, kai atstumas tarp jų yra didesnis arba lygus 7 NM. Šis atstumas yra minimali regiono erdvės separacija su papildoma 2 NM saugumo zona, taip kompensuojant galimus skaičiavimų netikslumus.
13. Galimos navigacinės paklaidos ir autopiloto reakcijos vėlavimas į sprendimą nėra tiriamos, nes darbo tikslas yra ištirti siūlomo modelio pritaikymo galimybę, o ne pilną įgyvendinimą.
14. Parametrų įtaka apskaičiuojamų trajektorijų ilgiui tiriami priėmimo vieno pasirinkto greičio, kuriuo skrenda abu orlaiviai. Vienodas orlaivių greitis pasirinktas dėl to, kad maži orlaivių greičių skirtumai (iki 50 kt) neturi įtakos trajektorijų ilgiui, o didesni greičių skirtumai yra gana reti.

3.3. Orlaivių konflikto sprendimo algoritmas

Siūlomo modelio algoritmas sukurtas panaudojant pagrindinius X. Yango ir P. Wei'o (2018) siūlomo Monte Karlo paieškos pritaikymo bepiločių orlaivių skrydžiams modelio principus, juos

adaptuojant trajektorijos ilgio optimizavimui ir komercinių orlaivių skrydžiams viršutinėje skrydžių valdymo erdvėje. Algoritme naudojamas paieškos medis yra sudarytas iš taškų, kuriuos aprašo to taško būseną. Paieškos medžio taškai (toliau – taškai) yra numeruojami, o jiems priklausančios būsenos sudaromos iš abiejų tiriamų orlaivių pozicijų, jų greičių projekcijų ir tikslo pozicijos. Siekiant geriau susieti algoritmo paaiškinimą su jo programiniu kodu (žr. A priedą), šiame skyrelyje bus naudojami tiek taško, tiek būsenos terminai, tačiau būtina pabrėžti, kad šiame tyrime pastarieji terminai yra glaudžiai susiję, nes kiekvienas taškas yra aprašomas konkrečia būseną, o kiekviena būseną yra priskiriama konkrečiam taškui.

Norint aprašyti pradinę būseną, reikalingos algoritmo įvestys. Pagrindinės algoritmo įvestys yra orlaivių (prie parametro žymėjimo atitinkamai pridedant orlaivio žymenį O arba I) bei vėjo parametrai, taip pat valdomo orlaivio tikslo taško pozicija. Šių įvesčių žymėjimas nurodytas 3.2 lentelėje.

3.2 lentelė. Pagrindinės algoritmo įvestys ir jų žymėjimas

Įvesties pavadinimas	Įvesties žymėjimas
Orlaivio pozicija	x, y
Orlaivio greičio projekcijos	v_x, v_y
Orlaivio kursas	φ
Vėjo greičio projekcijos	Wv_x, Wv_y
Vėjo kursas	γ
Tikslo taško pozicija	Gx, Gy

3.2 lentelėje nurodytos pradinės įvestys (be vėjo) sudaro pradinę būseną s_0 , kuri priklauso pradiniam taškui z_0 . s_0 aprašomas taip:

$$(O_x, O_y, O_{vx}, O_{vy}, O_\varphi, I_x, I_y, I_{vx}, I_{vy}, I_\varphi, G_x, G_y) \quad (3.2)$$

Visos tolimesnės būsenos aprašomos tuo pačiu principu ir sudaromos iš pagal kinematinį modelį perskaičiuotų įvesčių. Atliekant skaičiavimus su vėju, jo dedamosios įtraukiamos į kinematinį modelį. Visose būsenose tikslo pozicija nekinta.

Prieš pristatant algoritmo veikimo principą, svarbu apibrėžti, kas yra orlaivio galutinė būseną. Algoritmo skaičiavimų metų valdomo orlaivio priklausymas tiriamai plokštumai, atstumas tarp orlaivių bei atstumas tarp valdomo orlaivio ir jo tikslo yra nuolatos tikrinami. Atlikus tam tikrą kiekį veiksmų, valdomas orlaivis gali pasiekti vieną iš dviejų galutinių būsenų (žymimų galiniais taškais):

- Konflikto būseną – būseną, kai atstumas tarp orlaivių tampa mažesnis nei minimalus atskyrimo atstumas.

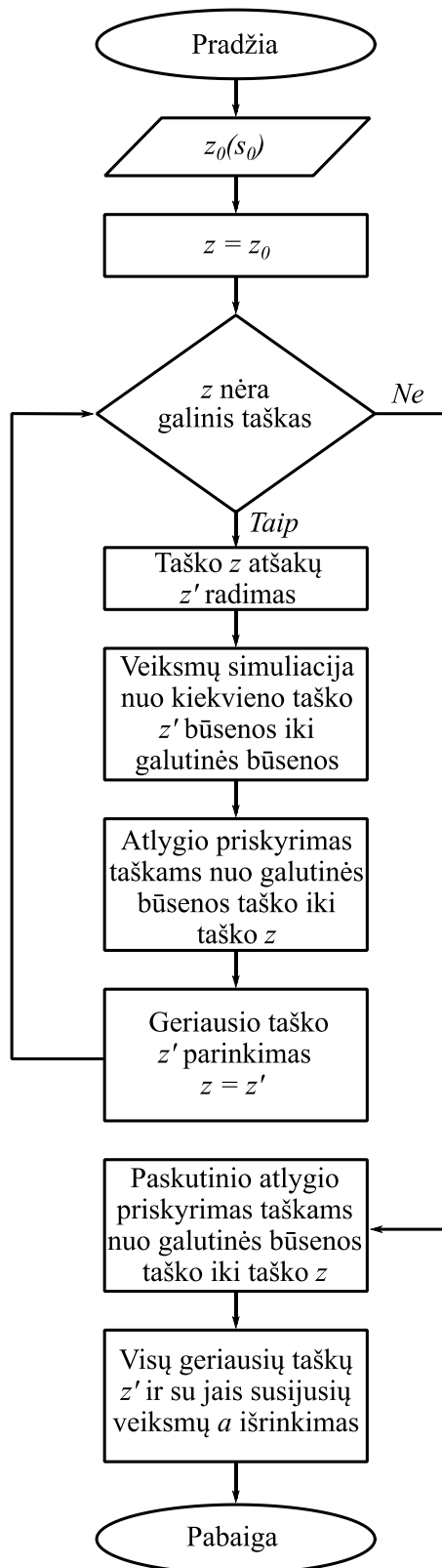
- Tikslo būseną – būseną, kai orlaivis pasiekia savo galutinį tikslą. Orlaivis laikomas pasiekęs tikslą, jei atstumas tarp orlaivio ir tikslo yra mažesnis nei separacinio apskritimo spindulys. Egzistuoja ir trečia galutinė būseną – paklydimo būseną. Ši būseną pasiekama tuomet, kai orlaivis išskrenda iš apibrėžtos tiriamosios erdvės. Vis dėlto, šiame darbe tiriami Lietuvos oro erdvėje esantys orlaivių konfliktai, kai orlaivių maršrutų taškai yra ant valstybės sienų, todėl tikimybė, jog valdomas orlaivis išskris iš tiriamosios oro erdvės nepasiekęs savo tikslo yra ganėtinai maža. Dėl didelės tiriamosios erdvės nėra poreikio paklydimo būsenos įtraukti į algoritmą, tačiau ji gali būti paranki nagrinėjant mažesnes oro erdves.

Siekiant supaprastinti bendrąją schemą, joje naudojami parametrų žymėjimai, nurodyti 3.3 lentelėje. Algoritme naudojami maksimalūs simuliacijų kiekis bei gylis yra nustatomi iš anksto ir gali būti keičiami pagal poreikį.

3.3 lentelė. Algoritmo parametrai ir jų žymėjimas

Parametro pavadinimas	Parametro žymėjimas
Pradinis paieškos medžio taškas	z_0
Tiriamas paieškos medžio taškas	z
Tolimesnis paieškos medžio taškas	z'
Pradinė būseną	s_0
Tiriama būseną	s
Tolimesnė būseną	s'
Simuliacijų kiekis	K
Maksimalus simuliacijų kiekis	k
Simuliacijų gylis	P
Maksimalus simuliacijų gylis	p
Pasirinktas veiksmas	a
Paieškos medžio taško aplankymų skaičius	N
Paieškos medžio tašką kertančių trajektorijų atlygių suma	Q

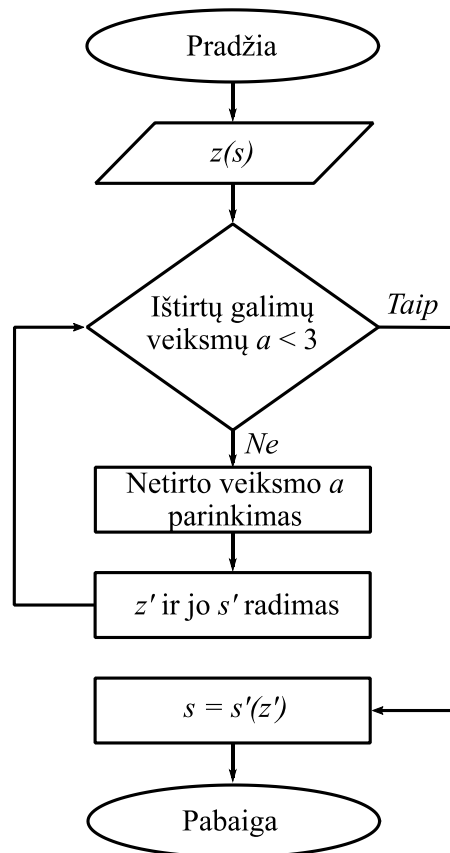
Bendroji algoritmo schema pateikiama 3.4 pav. kitame puslapyje. Į algoritmą įvedama pradinė situacija – orlaivių maršrutų pradžios taškai. Šią situaciją aprašo pradinė būseną s_0 , sudaryta pagal anksčiau paaiškintą principą. Pradinei būsenai suteikiamas atitinkamas pradinis taškas z_0 ir algoritmas nustato šį tašką kaip tiriamą tašką z bei pradeda trajektorijų tyrimo skaičiavimus. Pirmiausia patikrinama, ar šis taškas nėra galinis taškas, t. y. ar šį tašką aprašanti būseną nėra galutinė būseną, apskaičiuojant atstumus tarp valdomo orlaivio ir orlaivio įsibrovėlio bei tarp valdomo orlaivio ir jo tikslo taško. Jeigu ši sąlyga nėra įgyvendinama (orlaiviai susidūrė arba tikslas yra pasiektas), algoritmas apskaičiuoja būsenų ir jų taškų atlygius, išrenka geriausią trajektoriją ir skaičiavimų tikslas yra užbaigiamas.



3.4 pav. Bendroji algoritmo schema (Sudaryta autoriaus, 2020)

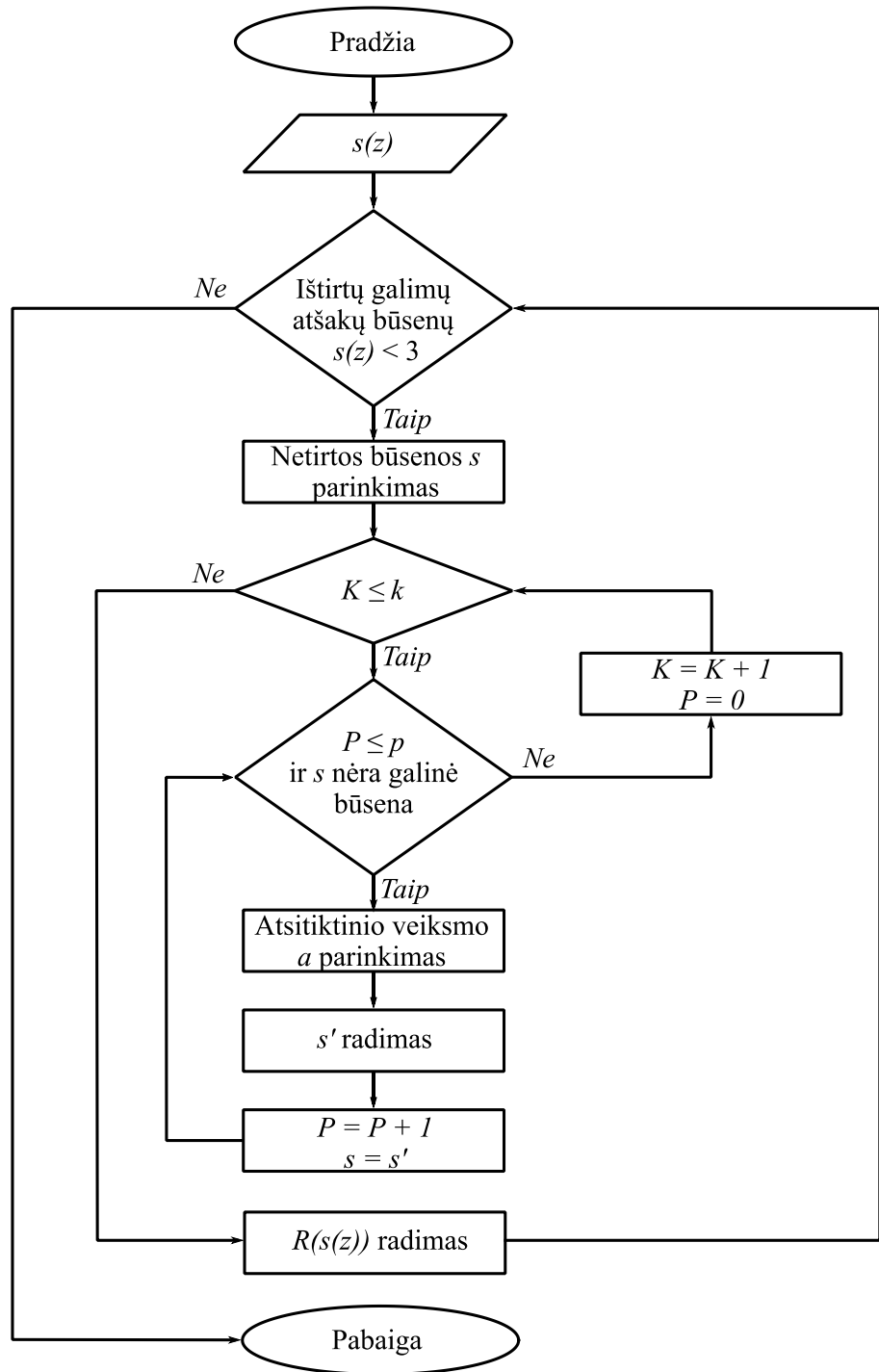
Jeigu tiriamas taškas atitinka iškeltą sąlygą, algoritmas tikrina, ar buvo ištirti visi trys šiame taške leidžiami atlikti veiksmai. Jeigu taškas nėra iki galo ištirtas, pasirenkamas vienas iš netirtų veiksmų ir apskaičiuojamas tolimesnis taškas z' . Pastarasis taškas yra taško z atšaka paieškos medyje.

Šis ciklas kartojamas tol, kol taškas tampa visiškai ištirtas, t. y. apskaičiuojami visi trys atšakos taškai ir jų būsenos (3.5 pav.). Būsenos s' nustatomos kaip tiriamos būsenos s tolimesniam jų tyrimui.



3.5 pav. Taško z atšakų z' radimas (Autorius 2020).

Toliau atliekama veiksmų simuliacija. Pasirenkamas vienas iš atšakos taškų ir patikrinama, ar nepasiektas maksimalus simuliacijų kiekis. Jei ši sąlyga patenkinama, patikrinama, ar pasirinkto taško būsena s nėra galutinė ir ar nėra pasiektas maksimalus simuliacijų gylis. Būsenai atitikus abi sąlygas pasirenkamas atsitiktinis veiksmas iš nustatytos veiksmų erdvės, apskaičiuojama tolimesnė būsena s' , nustatoma simuliacijos gylis ir apskaičiuotoji būsena s' nustatoma kaip tiriamą būsena s . Veiksmai su būsenomis atliekami tol, kol pasiekama viena iš galutinių būsenų arba išnaudojamas visas simuliacijų gylis. Tokia veiksmų simuliacija atliekama vis iš naujo tol, kol išnaudojamas visas galimas simuliacijų kiekis, taip patikrinant kuo daugiau įmanomų atlikti veiksmų ir galimų trajektorijų, prasidedančių nuo tiriamo atšakos taško. Tuomet apskaičiuojamas taško z atšakos taško z' atitinkančios tirtosios būsenos atlygis R . Šis veiksmų simuliacijos ciklas kartojamas tol, kol patikrinamos visų atšakos taškų simuliacijos ir gaunami atlygiai kiekvienam atšakos taškui (3.6 pav).

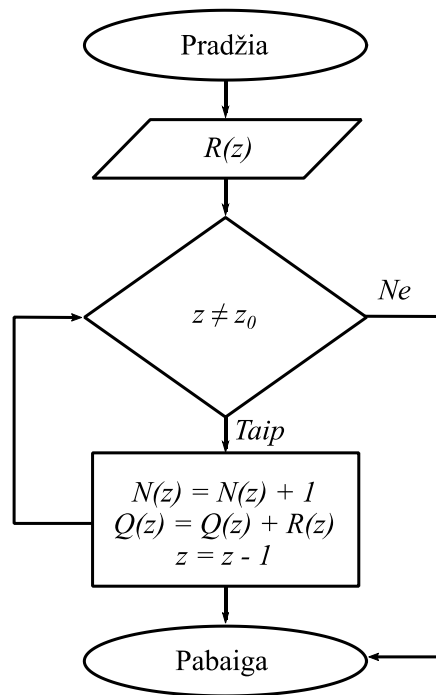


3.6 pav. Veiksmų simuliacija iki galutinės būsenos (Sudaryta autoriaus, 2020)

Šie atlygiai atgaliniu sklidimu yra priskiriami visiems tyrimo grandinėje nuo galutinės būsenos iki pirmųjų pradinio taško z_0 atšakų buvusiems taškams ir jų būsenoms, perskaičiuojant tų taškų aplankymo skaičiaus bei suminio atlygio vertes (3.7 pav). Tuomet iš tirtojo taško atšakų pasirenkamas vienas atšakos taškas z' , kuris maksimizuoja 2.7 formulės funkcijos vertes:

$$\operatorname{argmax}_{z' \in z \text{ atšakos}} \frac{Q(z')}{N(z')} + C \sqrt{\frac{2 \ln N(z)}{N(z')}} \quad (3.3)$$

Čia: z' – atšakos taškas; z – tiriamas taškas; z_0 – pradinis taškas; N – skaičius, nurodantis kiek kartų atitinkamas taškas buvo aplankytas anksčiau; Q – visų tirtų trajektorijų, kertančių atitinkamą tašką, atlygių suma.



3.7 pav. Atlygio priskyrimas tirtiems taškams (Sudaryta autoriaus, 2020)

Pasirinktas atšakos taškas z' nustatomas kaip tiriamas taškas z , nustatoma simuliacijų skaičiaus vertė ir patikrinama, ar šis taškas nėra galinis taškas ir ar nėra pasiektas maksimalus simuliacijų kiekis. Jeigu tiriamas taškas atitinka abi sąlygas, visas skaičiavimų ciklas pradedamas iš naujo. Šis ciklas kartojamas tol, kad viena iš pradinių sąlygų yra neįgyvendinama. Tuomet algoritmas apskaičiuoja paskutiniojo tirta taško būsenos atlygį, jį priskiria visiems tirtiems taškams pagal anksčiau paaiškintą principą ir apskaičiuoja galutinį rezultatą. Šis rezultatas taip pat maksimizuoja 2.7 formulės funkcijos vertę nebeteikdamas svarbos papildomos informacijos rinkimui ($C = 0$), o z' yra visų pradinio taško z_0 atšakų taškai:

$$\operatorname{argmax}_{z' \in z_0 \text{ atšakos}} \frac{Q(z')}{N(z')} \quad (3.4)$$

Gautas rezultatas yra taškų ir jų būsenų rinkinys. Kadangi veiksmai, atvedantys į šiuos taškus, yra žinomi, galima nubraižyti galutinę išvengiamąją trajektoriją.

4 ORLAIVIŲ KONFLIKTO SPRENDIMO PLOKŠTUMOJE REZULTATAI

4.1. Orlaivių konfliktų sprendimo rezultatai

Šiame poskyryje apžvelgiami orlaivių konflikto sprendimo rezultatai keturiems konfliktų tipams – priešpriešiniams, to paties maršruto, stataus bei smailaus trajektorijų susikirtimo kampo konfliktams. Siekiant aiškesnio vizualaus konfliktų sprendimo vaizdavimo brėžiniuose, konfliktams, kuriuose orlaivių greičiai skiriasi, parinkti didesni orlaivių greičių skirtumai (25-50 %), tačiau algoritmas veikia ir prie mažų greičių skirtumų. Tirtose situacijose nagrinėjami įvairūs orlaivių greičiai, būdingi skrydžiams regiono skrydžių valdymo erdvėje (250-500 kt). Trajektorijų brėžiniuose naudojami žymėjimai nurodyti 4.1 lentelėje, o standartiniai algoritmo nustatymai nurodyti 4.2 lentelėje

4.1 lentelė. Brėžinių legenda

Žymėjimas brėžinyje	Žymėjimo reikšmė
	Valdomo orlaivio tikslas
	Valdomo orlaivio maršruto pradžia
	Valdomo orlaivio pradinio maršruto trajektorija
	Valdomo orlaivio išvengiamojo maršruto trajektorija (kai $\alpha = 5^\circ$)
	Valdomo orlaivio išvengiamojo maršruto trajektorija (kai $\alpha = 10^\circ$)
	Valdomo orlaivio išvengiamojo maršruto trajektorija (kai $\alpha = 15^\circ$)
	Valdomo orlaivio alternatyvaus išvengamojo maršruto trajektorija
	Įsibrovėlio orlaivio maršruto pradžia
	Įsibrovėlio orlaivio maršruto trajektorija
	Įsibrovėlio orlaivio judėjimo kryptis

4.2 lentelė. Standartiniai algoritmo nustatymai

Nustatymas	Nustatymo vertė
Separacinio apskritimo spindulys	3,5 NM
Posūkio kampas	5°
Laiko žingsnis	1 min
Maksimalus simuliacijų kiekis	800
Maksimalus simuliacijų gylis	4

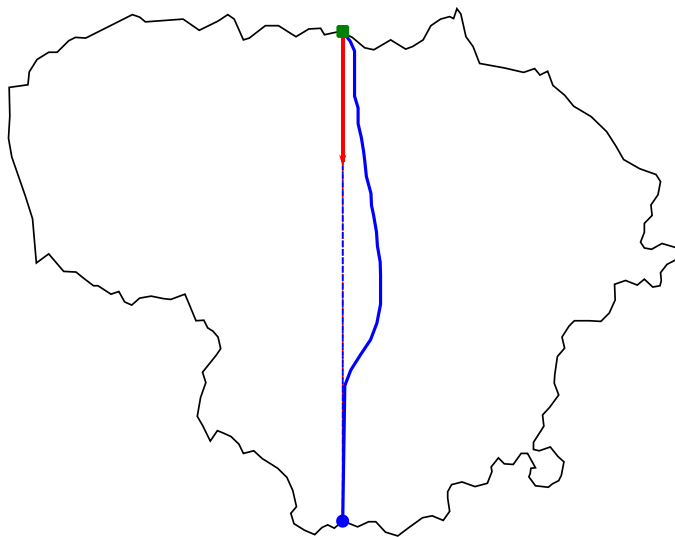
Šie standartiniai nustatymai naudoti visame tyrime, išskyrus smailaus trajektorijų susikirtimo kampo konflikto tyrimą. Algoritmo nustatymai pastarajam konfliktui nurodyti atitinkamame skyrelyje. Algoritmo nustatymų (taip pat ir kitų parametrų) įtaka tiek algoritmo skaičiavimų, tiek apskaičiuojamo sprendimo efektyvumui plačiau nagrinėjama 4.2. poskyryje.

6.1.1. Priešpriešinis konfliktas

Tiriamas standartinio priešpriešinio konflikto įvestys, aprašančios valdomo orlaivio ir orlaivio įsibrovėlio skrydžių parametrus, nurodytos 4.3 lentelėje. Priešpriešinis konfliktas visuomet turi du identiškus galimus sprendimo variantus ir apskaičiuotoji konflikto išvengiamoji trajektorija priklauso nuo algoritmo atsitiktinai parinkto pirmojo posūkio krypties (kairės arba dešinės). 4.1 pav. matoma viena iš galimų priešpriešinio konflikto sprendimo trajektorijų. Dėl mažo trajektorijų susikirtimo kampo galima daryti prielaidą, kad priešpriešiniai konfliktai yra išsprendžiami nesudėtingai, nuokrypis nuo pradinės trajektorijos mažas, todėl ir trajektorijos pailgėjimas dažniausiai būna nedidelis.

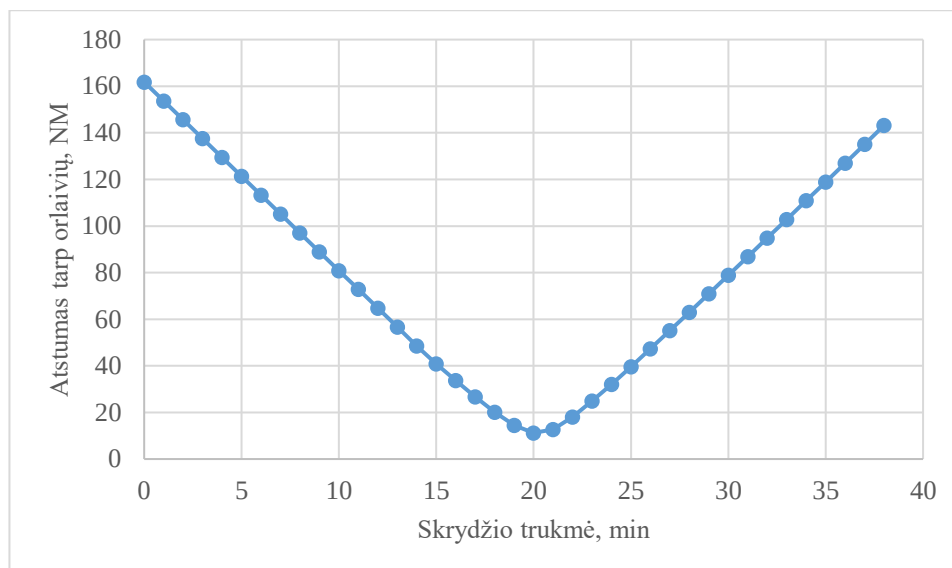
4.3 lentelė. Priešpriešinio konflikto įvestys

Konflikto įvestis	Konflikto įvesties vertė
Valdomo orlaivio tikslas	56° 14' 15"N 24° 0' 56"
Valdomo orlaivio maršruto pradžia	53° 43' 9"N 23° 54' 51"E
Valdomo orlaivio pradinis kursas	360°
Valdomo orlaivio greitis	300 kt
Įsibrovėlio orlaivio maršruto pradžia	56° 14' 15"N 24° 0' 56"
Įsibrovėlio orlaivio kursas	180°
Įsibrovėlio orlaivio greitis	300 kt



4.1 pav. Priešpriešinio konflikto sprendimo trajektorijos (Sudaryta autoriaus, 2020)

Atstumo tarp orlaivių kitimas priešpriešinio konflikto sprendimo metu pavaizduotas 4.2 pav., o sprendimo rezultatų duomenys pateikti 4.4 lentelėje.



4.2 pav. Atstumo tarp orlaivių kitimas skrydžio metu (priešpriešinis konfliktas) (Sudaryta autoriaus, 2020)

4.4 lentelė. Priešpriešinio konflikto sprendimo rezultatai

Pradinės trajektorijos ilgis, NM	Išvengiamosios trajektorijos ilgis, NM	Trajektorijų ilgių skirtumas, NM	Skrydžio laikų skirtumas, min	Trajektorijos ilgio ir skrydžio laiko pokytis, %
151,24	154,49	3,24	0,6	2,14%

Atstumo tarp orlaivių kitimas skrydžio metu yra parabolė, kuri įrodo nuoseklią išvengiamąją trajektoriją, o mažiausias atstumas tarp orlaivių yra 8 NM. Remiantis mažiausiu atstumu tarp orlaivių galima daryti išvadą, kad algoritmas, ribojamas nustatymų, sukuria papildomą saugumo zoną, t. y. padidina minimalų atskyrimo atstumą. Šis atstumas gali būti sumažinamas iki norimo 7 NM atskyrimo pakeičiant algoritmo nustatymus, tačiau tai turėtų įtakos skaičiavimų trukmei. Kita vertus, atstumo skirtumas yra ganėtinai mažas, todėl galima teigti, kad algoritmas randa gerą sprendimą. Trajektorijos ilgio ir skrydžio laiko pokytis yra 2,14 %, tuo tarpu standartinio skrydžių vadovo sprendimo pokytis yra 2 % (skirtumas 0,14 %). Ši situacija buvo pakartotinai tirta, kai orlaivių greičiai dvigubai didesni. Gautos trajektorijos panašios į anksčiau aprašytos konfliktinės situacijos trajektorijos, o galutinis trajektorijos ilgio ir skrydžio laiko pokytis 2,5 %, todėl daroma išvada, kad vienodai didinant ar mažinant abiejų orlaivių greičius trajektorijų ilgio pokytis keičiasi nežymiai, todėl kitų konfliktų tipų tyrimuose šis aspektas nėra nagrinėjamas.

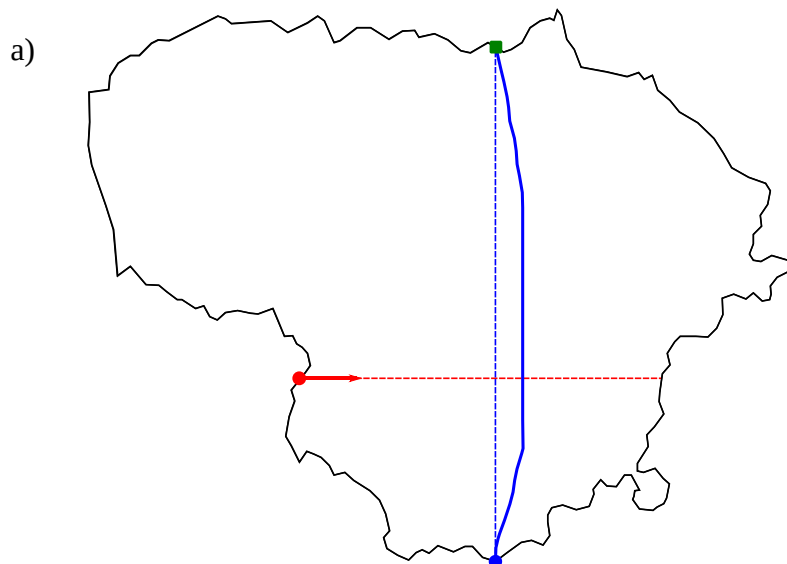
4.1.2. Stataus trajektorijų susikirtimo kampo konfliktas

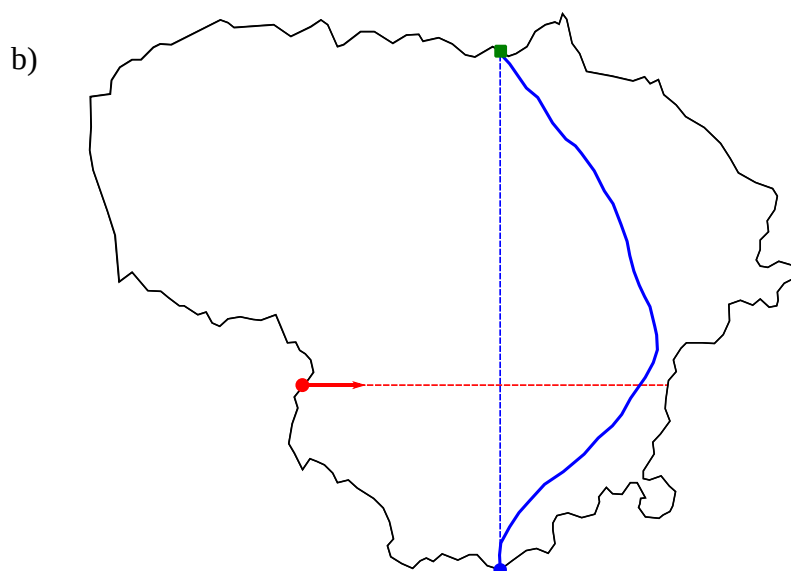
Tiriamo standartinio stataus trajektorijų susikirtimo kampo konflikto įvestys, aprašančios valdomo orlaivio ir orlaivio įsibrovėlio skrydžio parametrus, nurodytos 4.5 lentelėje. Šio tipo konfliktams būdingi du skirtingi sprendimai, aprašomi skirtingo ilgio trajektorijomis. Šie sprendimai

priklauso nuo orlaivių pozicijų, kai vienas iš orlaivių yra trajektorijų susikirtimo taške. Jeigu valdomas orlaivis pasiekia trajektorijų susikirtimo tašką anksčiau nei orlaivis įsibrovėlis, trumpesnė išvengiamoji trajektorija visuomet bus ta trajektorija, kurią sekdamas valdomas orlaivis kirs orlaivio įsibrovėlio kelią priešais. Priešingu atveju, trumpesnė išvengiamoji trajektorija yra ta, kurią sekdamas valdomas orlaivis aplenktų orlaivį įsibrovėlį iš paskos. Dviejų tirtų situacijų trajektorijos pavaizduotos 4.3 pav. Situacija, kai valdomas orlaivis yra greitesnis už orlaivį įsibrovėlį, nėra tiriamą, nes tokiu atveju konflikto nebelieka ir valdomas orlaivis gali skristi tiesiai į savo tikslą netrukdomas. Tiriant šio tipo konfliktą daroma prielaida, kad algoritmas parinks trumpesnę išvengiamąją trajektoriją kaip galutinį sprendimą.

4.5 lentelė. Stataus trajektorijų susikirtimo kampo konflikto įvestys

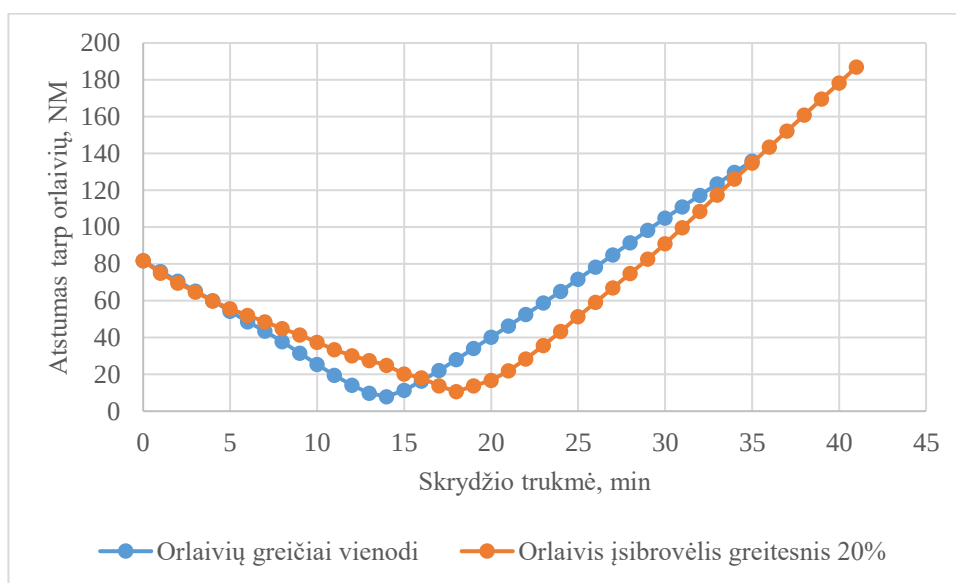
Konflikto įvestis	Konflikto įvesties vertė
Valdomo orlaivio tikslas	56° 17' 06"N 24° 29' 33"E
Valdomo orlaivio maršruto pradžia	53° 56' 38"N 24° 23' 00"E
Valdomo orlaivio pradinis kursas	360°
Valdomo orlaivio greitis	300 kt
Įsibrovėlio orlaivio maršruto pradžia	54° 45' 08"N 22° 30' 23"E
Įsibrovėlio orlaivio kursas	090°
Įsibrovėlio orlaivio greitis	a) 300 kt b) 360 kt





4.3 pav. Stataus trajektorijų susikirtimo kampo konflikto sprendimo trajektorijos, kai: a) orlaivių greičiai vienodi; b) orlaivis įsibrovėlis 20 % greitesnis už valdomą orlaivį (Sudaryta autoriaus, 2020)

Tyrimo rezultatai pateikti 4.4 pav. ir 4.6 lentelėje.



4.4 pav. Atstumo tarp orlaivių kitimas skrydžio metu (stataus trajektorijų susikirtimo kampo konfliktas) (Sudaryta autoriaus, 2020)

4.6 lentelė. Stataus trajektorijų susikirtimo kampo konflikto sprendimo rezultatai

	Pradinės trajektorijos ilgis, NM	Išvengiamosios trajektorijos ilgis, NM	Trajektorijų ilgių skirtumas, NM	Skrydžio laikų skirtumas, min	Trajektorijos ilgio ir skrydžio laiko pokytis, %
a)	140,61	141,87	1,26	0,3	0,9%
b)		173,08	32,47	6,5	22,84%

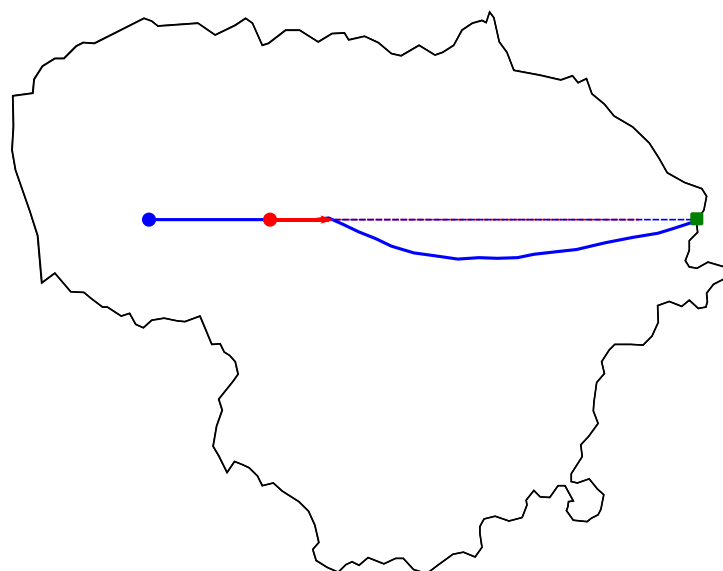
Mažiausias nustatytas atstumas tarp orlaivių skrydžio metu, kai orlaivių greičiai yra vienodi, yra 7 NM, o kai orlaivis įsibrovėlis greitesnis – 8 NM. Galima daryti išvadą, kad sudėtingesnėje situacijoje algoritmas prideda papildomą saugumo zoną. Įprastoje situacijoje, kai orlaivių greičiai vienodi, algoritmo apskaičiuojama trajektorija pailgėja nežymiai (0,9 %). Išvengiamosios trajektorijos ilgio pokytis padidėja beveik 20 kartų, kai orlaivis įsibrovėlis yra greitesnis už valdomą orlaivį. Tokį staigų trajektorijos ilgio padidėjimą lemia konflikto simetriškumas. Konflikto simetriškumas reiškia tai, kad abu orlaiviai trajektorijų susikirtimo taške atsidurtų tuo pačiu metu. Šiuo atveju anksčiau aprašyti trumpesnės trajektorijos radimo principai nebegalioja, todėl realiose situacijose skrydžių vadovai tokių simetrinių konfliktų horizontalioje plokštumoje nesprensdžia, nes sprendimo geometrija yra daug sudėtingesnė ir orlaivio nukrypimas nuo pradinės trajektorijos žymiai didesnis, kai neįmanoma nustatyti nei vieno orlaivio pirmenybės. Abu galimi simetrinio konflikto sprendimai yra panašūs, todėl algoritmo sprendimas priklauso nuo pirmo atsitiktinio ištirto posūkio. Lyginant apskaičiuotųjų trajektorijų ilgio pokyčius su atitinkamais standartinio sprendimo rezultatais, trajektorijos, kai orlaivių greičiai vienodi, ilgio pokytis yra 0,5 % didesnis (standartinio sprendimo pokytis – 0,4 %), o kai orlaivis įsibrovėlis greitesnis ir konfliktas simetrinis palyginti neįmanoma, nes standartinis sprendimas nėra taikomas.

4.1.3. To paties maršruto konfliktas

Tiriamas standartinio to paties maršruto konflikto įvestys, aprašančios valdomo orlaivio ir orlaivio įsibrovėlio skrydžių parametrus, nurodytos 4.7 lentelėje. To paties maršruto konfliktas, kaip ir priešpriešinį konfliktas, turi du identiškus galimus sprendimo variantus. Kadangi šie du konfliktų tipai yra itin panašūs, to paties maršruto konflikto išvengiamosios trajektorijos apskaičiavimui galioja tas pats algoritmo atsitiktinio pasirinkimo principas. 4.5 pav. matoma viena iš galimų to paties maršruto konflikto sprendimo trajektorijų. Dėl minėto panašumo, daroma prielaida, kad gauti rezultatai bus artimi priešpriešinio konflikto rezultatams.

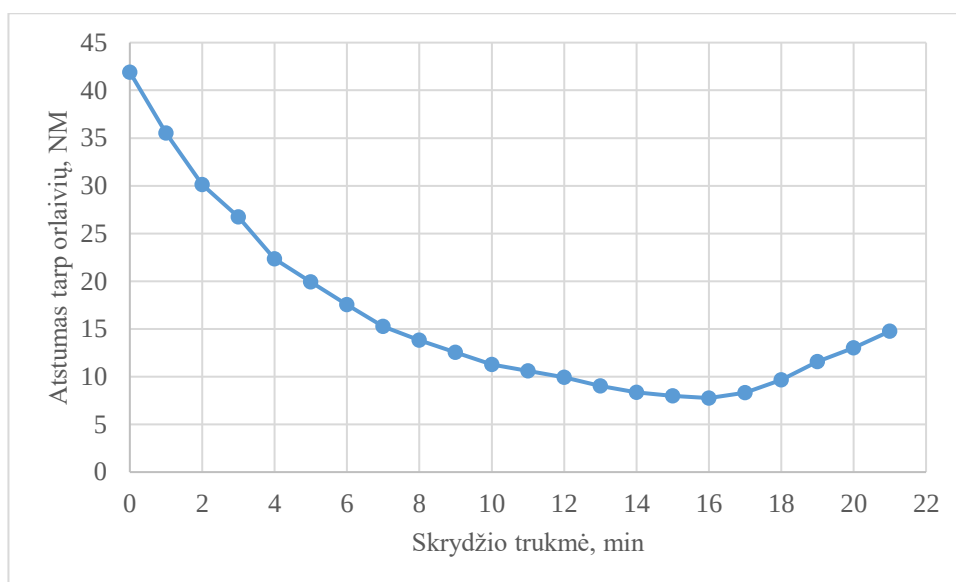
4.7 lentelė. To paties maršruto konflikto įvestys

Konflikto įvestis	Konflikto įvesties vertė
Valdomo orlaivio tikslas	55° 30' 57"N 26° 27' 20"E
Valdomo orlaivio maršruto pradžia	55° 30' 57"N 22° 07' 51"E
Valdomo orlaivio pradinis kursas	090°
Valdomo orlaivio greitis	430 kt
Įsibrovėlio orlaivio maršruto pradžia	55° 30' 57"N 23° 04' 10"E
Įsibrovėlio orlaivio kursas	090°
Įsibrovėlio orlaivio greitis	290 kt



4.5 pav. To paties maršruto konflikto sprendimo trajektorijos (Sudaryta autoriaus, 2020)

To paties maršruto konflikto sprendimo rezultatai pateikiami 4.6 pav. ir 4.8 lentelėje.



4.6 pav. Atstumo tarp orlaivių kitimas skrydžio metu (to paties maršruto konflikto) (Sudaryta autoriaus, 2020)

4.8 lentelė. To paties maršruto konflikto sprendimo rezultatai

Pradinės trajektorijos ilgis, NM	Išvengiamosios trajektorijos ilgis, NM	Trajektorijų ilgių skirtumas, NM	Skrydžio laikų skirtumas, min	Trajektorijos ilgio ir skrydžio laiko pokytis, %
147,07	149, 9	2,83	0,4	1,93 %

4.6 pav. grafike matyti, kad parabolė yra plokštesnė lyginant su priešpriešinio konflikto sprendimo rezultatais. Šį skirtumą paaiškina konflikto tipas, nes minimalus 7 NM atstumas tarp

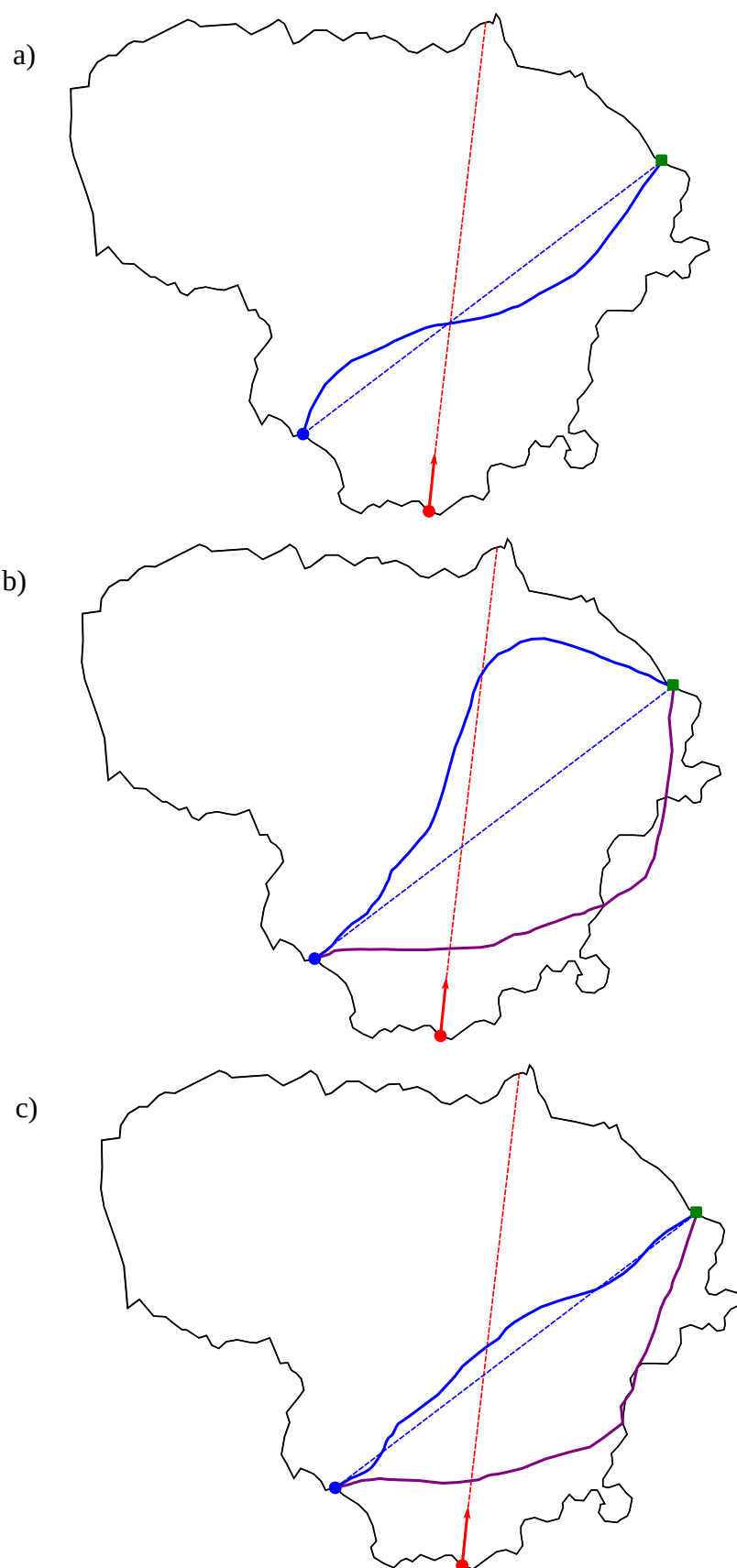
orlaivių yra palaikomas skrendant lygiagrečiai tol, kol atsiranda galimybė pasukti orlaivį į tikslą. Trajektorijos ilgio pokytis nuo apskaičiuotojo priešpriešinio konflikto išvengiamosios trajektorijos ilgio pokyčio skiriasi 0,21 %, todėl anksčiau pateikta prielaida dėl rezultatų panašumo yra teisinga. To paties maršruto trajektorijos ilgio pokytis lyginant su standartiniu sprendimu yra 0,51 % (standartinio sprendimo trajektorijos ilgio pokytis – 1,42 %) didesnis.

4.1.4. Smailaus trajektorijų susikirtimo kampo konfliktas

Tiriamos smailaus trajektorijų susikirtimo kampo konflikto įvestys, aprašančios valdomo orlaivio ir orlaivio įsibrovėlio skrydžio parametrus, nurodytos 4.9 lentelėje. Šio tipo konfliktams, kaip ir stataus trajektorijų susikirtimo kampo konfliktams, būdingi du skirtingi sprendimai, aprašomi skirtingo ilgio trajektorijomis. Šio tipo konflikto išvengiamųjų trajektorijų apskaičiavimui galioja tie patys principai. Vis dėlto, smailaus trajektorijų susikirtimo kampo konfliktai yra specifiniai konfliktai, kurie geometriškai išsprendžiami daug sudėtingiau nei kitų tipų konfliktai. Dėl šios priežasties skrydžių vadovai niekada neišduoda horizontalių manevrų instrukcijų, t. y. standartinis sprendimas netaikomas, todėl tiriant galimą automatinį tokio konflikto sprendimą daroma prielaida, kad trajektorijos ilgio pokytis smailaus kampo konfliktams bus didesnis nei kitiems konflikto tipams. Taip pat šio konflikto sprendimo tyrimo metu plačiau nagrinėjama algoritmo nustatymų įtaka išvengiamųjų trajektorijų ilgiui. Trijų tirtų situacijų trajektorijos pavaizduotos 4.7 pav. Brėžiniuose mėlyna spalva žymima trajektorija, apskaičiuota, kai $p = 10$, o alternatyvi violetinė trajektorija gaunama, kai $p = 8$. Situacija, kai orlaivis įsibrovėlis yra greitesnis už valdomą orlaivį, nėra tirama, nes tokioje situacijoje konflikto nebūna ir valdomas orlaivis gali skristi netrukdomas.

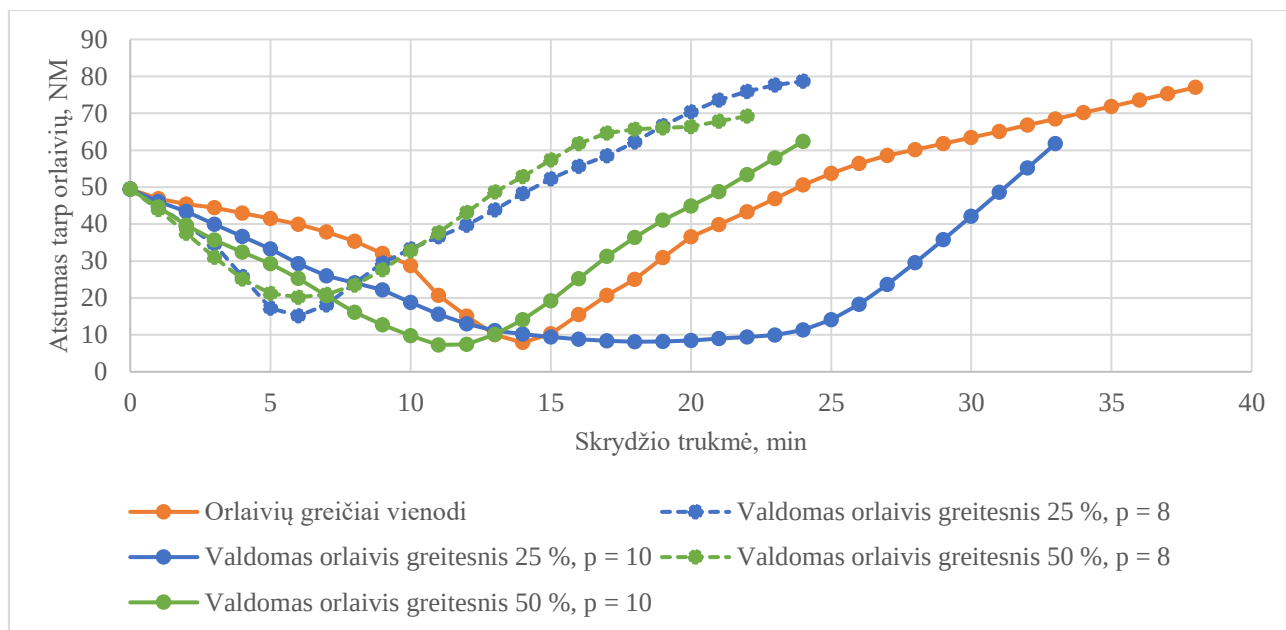
4.9 lentelė. Smailaus trajektorijų susikirtimo kampo konflikto įvestys

Konflikto įvestis	Konflikto įvesties vertė
Valdomo orlaivio tikslas	55° 51' 44"N 26° 34' 13"E
Valdomo orlaivio maršruto pradžia	54° 10' 06"N 22° 58' 32"E
Valdomo orlaivio pradinis kursas	045°
Valdomo orlaivio greitis	a) 280 kt b) 350 kt c) 420 kt
Įsibrovėlio orlaivio maršruto pradžia	54° 10' 06"N 24° 23' 00"E
Įsibrovėlio orlaivio kursas	005°
Įsibrovėlio orlaivio greitis	a) 280 kt b) 280 kt c) 280 kt



4.7 pav. Smailaus trajektorijų susikirtimo kampo konflikto sprendimo trajektorijos, kai: a) orlaivių greičiai vienodi; b) valdomas orlaivis greitesnis 25 %; c) valdomas orlaivis greitesnis 50 % (Sudaryta autoriaus, 2020)

Atstumo tarp orlaivių kitimas smailaus orlaivių trajektorijų susikirtimo kampo konflikto sprendimo metu esant skirtingiems gyliams bei orlaivių greičiams pavaizduoti 4.8 pav., o sprendimo rezultatų duomenai pateikiami 4.10 lentelėje.



4.8 pav. Atstumo tarp orlaivių kitimas skrydžio metu (smailaus trajektorijų susikirtimo kampo konfliktas) (Sudaryta autoriaus, 2020)

4.10 lentelė. Smailaus trajektorijų susikirtimo kampo konflikto sprendimo rezultatai

	Simuliacijų gylis	Pradinės trajektorijos ilgis, NM	Išvengiamosios trajektorijos ilgis, NM	Trajektorijų ilgių skirtumas, NM	Skrydžio laikų skirtumas, min	Trajektorijos ilgio ir skrydžio laiko pokytis, %
a)	10	160,14	169,35	9,21	1,9	5,75 %
b)	8		191	30,86	5,2	19,2 %
	10		179,25	19,11	3,3	11,85 %
c)	8		188,53	28,39	4	17,68 %
	10		162,6	2,46	0,4	1,56 %

Tyrimo metu paaiškėjo, kad standartinio simuliacijų gylio nustatymo ($p = 4$) neužtenka smailaus trajektorijų susikirtimo kampo konfliktui išspręsti, nes nustatius tokį gylį algoritmas neįvertina pakankamo kiekio galimų trajektorijų ir reikalingų posūkių, o pastarųjų reikia didelio kiekio dėl anksčiau aptarto geometrijos sudėtingumo. Eksperimentiniu būdu nustatyta, kad mažiausias simuliacijų gylis, galintis užtikrinti išvengiamosios trajektorijos apskaičiavimą, yra dvigubai didesnis nei standartinis, t. y. $p = 8$. Šio gylio užtenka užtenka apskaičiuoti ilgesnio trajektorijos ilgio pokyčio trajektoriją. Vis dėlto, norint gauti geresnę, t. y. trumpesnę trajektoriją, mažiausias reikalingas simuliacijų gylis yra $p = 10$. 4.8. pav. matoma, kad pastarasis gylis tarp orlaivių išlaiko atstumą,

artimesnį minimaliam leistinam atstumui (7-8 NM). Taip pat paveiksle matoma, kad situacijoje, kai valdomas orlaivis greitesnis už orlaivį įsibrovėlį 25 %, parabolė yra plokštesnė, taigi, orlaiviai dalį laiko skrenda lygiagrečiai.

Vertinant trajektorijų ilgių pokyčius galima teigti, kad prielaida, jog pokyčiai bus didesni lyginant su kitų konfliktų rezultatais, yra teisinga. Lyginant vienodo orlaivių greičio situacijos rezultatus su atitinkamais kitų konfliktų rezultatais smailaus trajektorijų susikirtimo kampo konflikto trajektorijos ilgio pokytis vidutiniškai yra 4,1 % didesnis. Pasirenkus didesnę simuliacijų gylį, išvengiama trajektorija, kai valdomas orlaivis greitesnis 25 %, yra trumpesnė 7,35 %, o kai greičiai skiriasi 50 % - 16,12 %. Be to, pastarosios trajektorijos leidžia orlaiviui likti tos pačios valstybės oro erdvėje (žr. 4.7 b) pav.). Šių trajektorijų palyginti su standartiniu sprendimu neįmanoma dėl anksčiau išvardintų priežasčių.

4.1.5. Vėjo įtaka konflikto išvengiamajai trajektorijai

Vėjo įtakos rezultatų iliustravimui šiame skyrelyje pasirinkta vidutinio geometrijos sudėtingumo situacija – stataus trajektorijų susikirtimo kampo orlaivių konfliktas. Nagrinėjamos situacijos orlaivių maršrutų taškai bei kursai yra tokie patys, kaip ir 4.1.2. skyrelyje (žr. 4.5 lentelę), išskyrus orlaivių greičius – šiame tyrime orlaiviai skrenda 450 kt greičiu. Tyrime daroma prielaida, kad orlaivio įsibrovėlio greitis ir kryptis tyrimo metu nesikeis (orlaivis įsibrovėlis skrenda išilgai vėjo krypties, todėl tai neturi įtakos orlaivio trajektorijai)

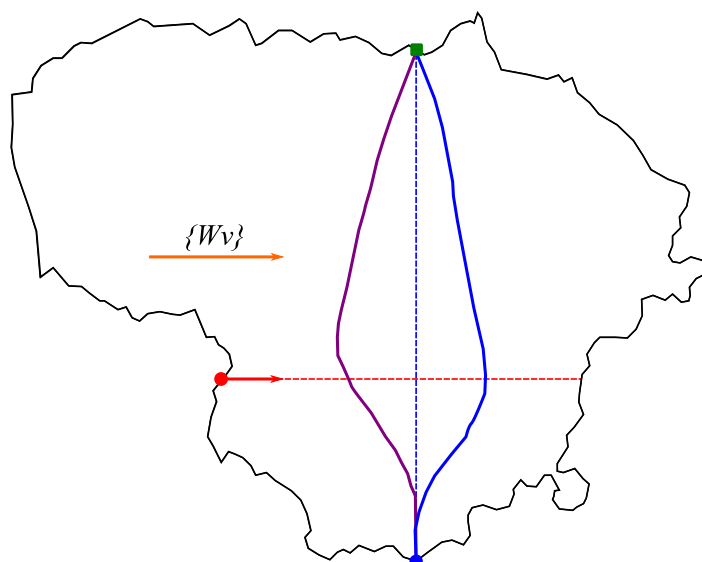
Kaip pavyzdys šiame tyrime nagrinėjamas šoninis vėjas. Jo įtaka apskaičiuotai išvengiamajai trajektorijai tirama esant dviem skirtingiems vėjo kursams bei greičiams. Šio tyrimo rezultatai pateikiami 4.11 lentelėje.

4.11 lentelė. Vėjo įtakos išvengiamajai trajektorijai rezultatai

Vėjo greitis, kt	Vėjo kryptis, °	Pradinės trajektorijos ilgis, NM	Išvengiamosios trajektorijos ilgis, NM	Trajektorijų ilgių skirtumas, NM	Skrydžio laikų skirtumas, min	Trajektorijos ilgio ir skrydžio laiko pokytis, %
25	090°	140,61	144,74	4,13	0,5	3 %
	270°		148,33	7,72	1	5,5 %
50	090°		143,86	3,25	0,4	2,4 %
	270°		148,58	7,97	1,1	5,7 %

Iš lentelėje pateiktų rezultatų matyti, kad esant silpnesniai vėjui ($W_{v1} = 50$ kt), trajektorijos ilgio pokytis lyginant su trajektorijoms be vėjo įtakos padidėja ~3 kartus. Galima teigti, kad rezultatai yra realistiški, nes situacijoje, kai vėjas pučia valdomą orlaivį tolyn nuo orlaivio įsibrovėlio (šioje

situacijoje – kai vėjas pučia iš vakarų), gaunama ilgesnė trajektorija. Tuo tarpu eksperimentų metu tiriant stipresnį vėją ($Wv_2 = 100$ kt) buvo pastebėta, kad esant didelei orlaivio nuonašai nuo pradinės trajektorijos laikui bėgant, algoritmas parenka geresnį variantą. 4.9 pav. matomas tokio sprendimo pavyzdys – kai vėjas pučia iš vakarų, mėlyna linija žymi orlaivio trajektoriją esant silpnesniam vėjui, o violetinė linija žymi trajektoriją esant dvigubai stipresniam vėjui (oranžinė rodyklė – vėjo kryptis).

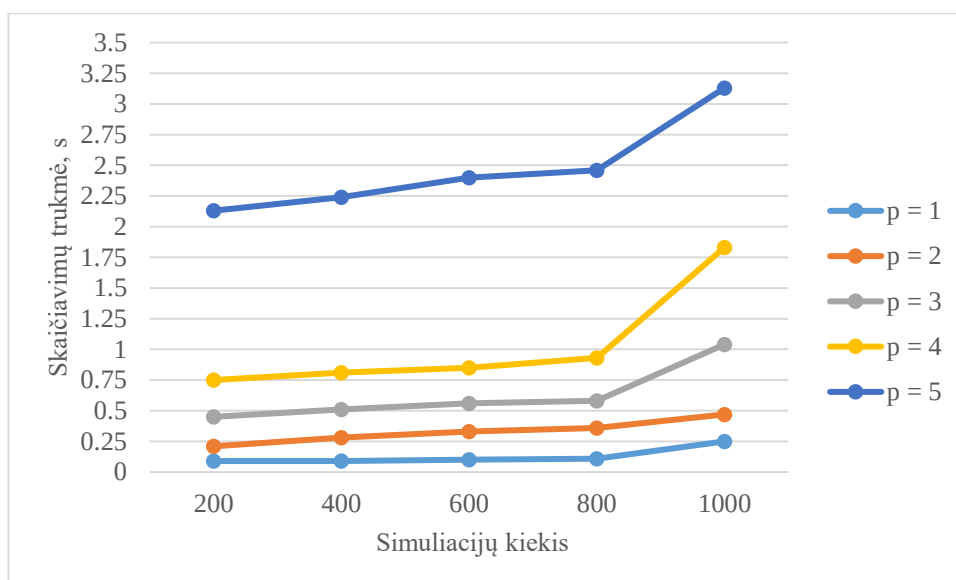


4.9 pav. Vėjo įtaka stataus trajektorijų susikirtimo kampo konflikto sprendimo trajektorijoms (Sudaryta autoriaus, 2020)

Kaip matyti paveiksle, esant stipresniam vėjui algoritmas parenka tokią išvengiamąją trajektoriją, kurioje valdomas orlaivis aplenkia orlaivį iš paskos, t. y. nebesilaikoma standartinių konflikto sprendimo taisyklių (žr. 4.1.2. skyrelį). Lyginant abiejų vėjų greičių rezultatus galima daryti išvadą, kad nepriklausomai nuo vėjo stiprumo, išvengiamųjų trajektorijų ilgiai labai panašūs (vidutinis skirtumas $\sim 0,5$ %). Ši trajektorijų tolygumą lemia algoritmo trajektorijos numatymo savybė, kuri leidžia parinkti geriausią sprendimą nesilaikant standartinių skrydžių valdymo taisyklių.

4.2. Algoritmo ir jo sprendimo efektyvumo tyrimo rezultatai

Remiantis 3 skyriuje aprašytu algoritmo veikimo principu galima teigti, kad paties algoritmo efektyvumą nulemia du kintamieji – simuliacijų kiekis k ir gylis p . Kadangi algoritmas veikia realiu laiku, jo skaičiavimų trukmė gali turėti įtakos sprendimų įgyvendinimui. Efektyvumas buvo tiriamas sprendžiant atsitiktinius konfliktus išskyrus smailaus trajektorijų susikirtimo kampo konfliktus. Efektyvumui tirti pasirinkti penki simuliacijų gyliai ir kiekiai. Šio tyrimo rezultatai pavaizduoti 4.10 pav.

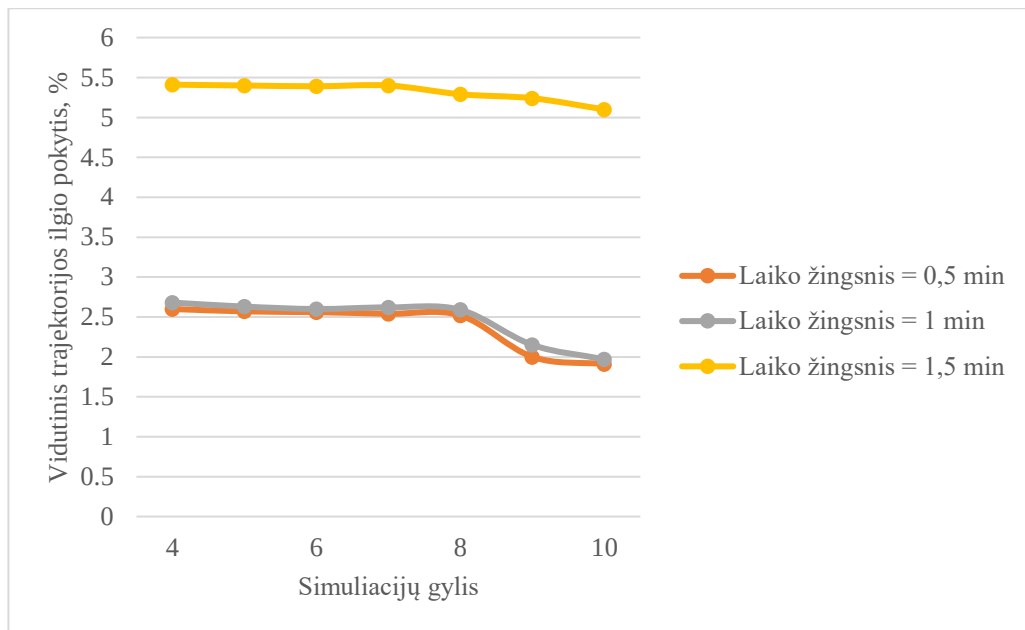


4.10 pav. Sprendimo skaičiavimų trukmės priklausomybė nuo simuliacijų gylio ir kiekio (Sudaryta autoriaus, 2020)

Remiantis 4.1. poskyrio eksperimentiniais rezultatais nustatyta, kad norint išspręsti didžiąją dalį konfliktų mažiausias reikalingas simuliacijų gylis yra $p = 4$. Efektyvumo tyrimo rezultatų grafike matyti, kad taikant pasirinktą simuliacijų gylį didžiausias galimas simuliacijų kiekis, užtikrinantis, kad skaičiavimų trukmė neišaugtų dvigubai, yra $k = 800$.

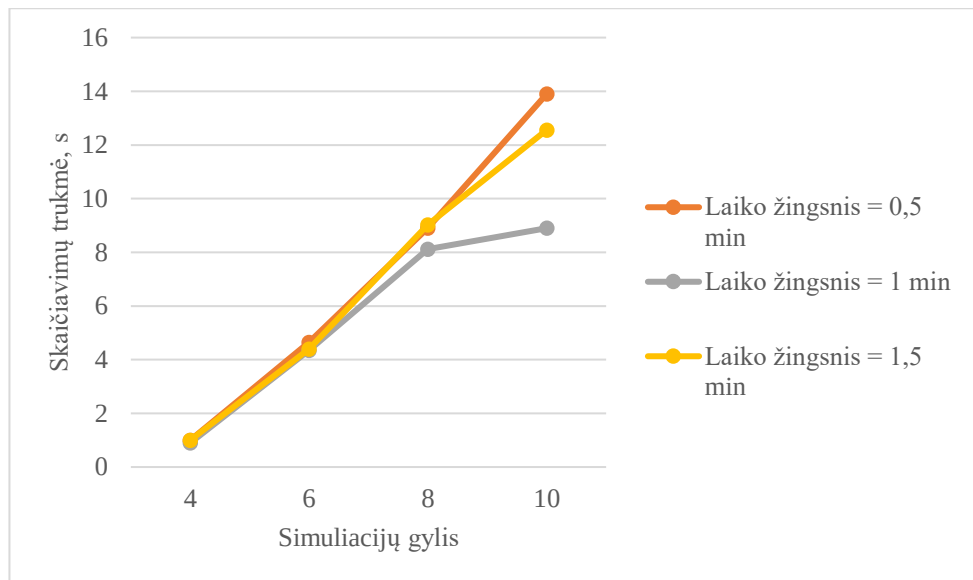
Kiti du faktoriai, turintys įtakos skaičiavimų trukmei ir algoritmo efektyvumui, yra leistinų veiksmų kiekis laiko momentu bei laiko žingsnis (nurodantis, kas kiek laiko orlaivis gali atlikti posūkį). Dėl pirmojo faktoriaus tyrimas orientuotas tik į kurso keitimą, kai orlaivis laiko momentu gali atlikti tik vieną iš trijų veiksmų – skristi tiesiai, sukti į kairę arba sukti į dešinę. Leistino posūkio maksimali absoliutinė vertė gaunama iš pasirinktos orlaivio veiksmų erdvės, kuri yra sudaroma remiantis galimais orlaivio veiksmais laiko momentu. Tyrimas atliekamas laikantis logiškos prielaidos, kad mažesnė orlaivių veiksmų erdvė lems trumpesnę skaičiavimų laiką nei didesnė, todėl veiksmų erdvė yra papildomai susiaurinama pasirenkant konkrečias leistinas veiksmų (posūkių) vertes. Antrasis faktorius glaudžiai siejasi ir su algoritmo sprendimo efektyvumu.

Algoritmo sprendimo efektyvumą nurodo trajektorijos ilgio pokytis, kuris priklauso nuo anksčiau minėtojo laiko žingsnio, simuliacijų gylio, orlaivių greičių ir leistinų posūkio verčių. 4.11 pav. pateikiamas grafikas atspindi trajektorijos ilgio pokyčio priklausomybę nuo simuliacijų gylio ir laiko žingsnio, kai orlaivių greičiai vienodi (400 kt). Šiame grafike pateikiamas trajektorijos ilgio pokyčio vidurkis visiems konfliktų tipams, nes algoritmas turi veikti kuo tolygiau nepriklausomai nuo galimos konfliktinės situacijos.



4.11 pav. Vidutinio trajektorijų ilgio pokyčio priklausomybė nuo simuliacijų gylio ir laiko žingsnio (Sudaryta autoriaus, 2020)

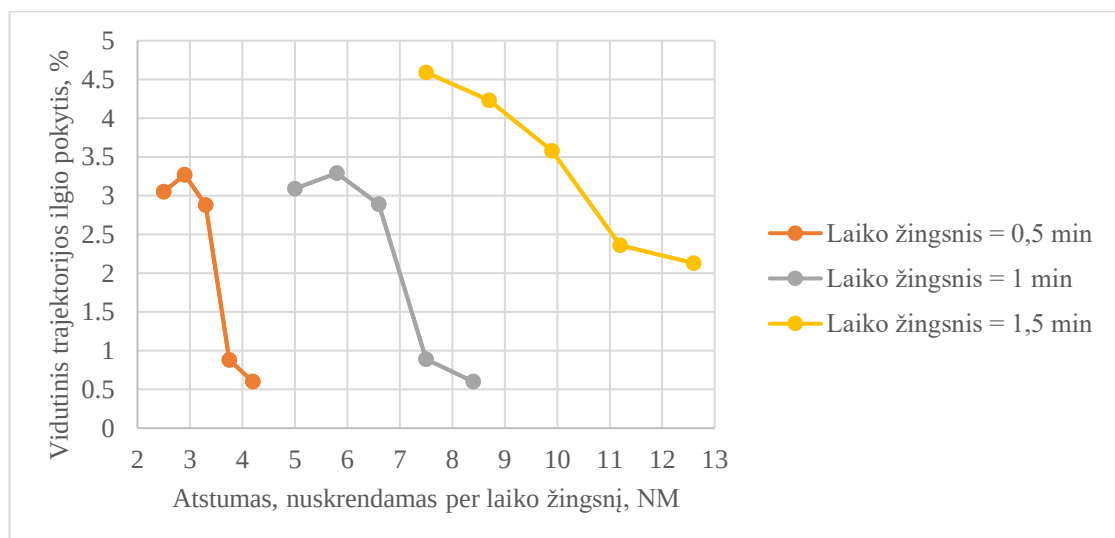
Kadangi orlaiviai skrenda didelius atstumus, nagrinėti mažesnių nei 0,5 min laiko žingsnių nėra prasmės (itin išauga skaičiavimų kaštai). Didesni nei 1,5 min laiko žingsniai taip pat nebuvo tyrinėti, nes tokie žingsniai nebėra efektyvūs sprendžiant konfliktus, kai orlaiviai vienas nuo kito yra nutolę mažiausiu leistinu atstumu išvengiamiesiems manevrams pradėti, t. y. kai pradinis atstumas tarp orlaivių yra artimas 40 NM – tokiu atveju galima situacija, kad algoritmas nesugebės rasti jokios išvengiamosios trajektorijos, nes orlaiviai nuskris per didelius atstumus. Be to, dideli laiko žingsniai gali lemti tokią situaciją, kurioje algoritmas nebesugeba grąžinti orlaivio į jo tikslą ir orlaivis pradeda suktis orbitoje. Esant vienodiems orlaivių greičiams simuliacijų gylio įtaka yra nežymi – iki 0,5 %, ši tendencija būdinga ir paprastos geometrijos skirtingų greičių konfliktams, išskyrus simetrinius konfliktus (simuliacijų gylio įtaka sudėtingos geometrijos konfliktams, kai orlaivių greičiai skiriasi, aptarta 4.1.4. skyrelyje). Remiantis grafiku daroma išvada, kad didesni nei 1 min laiko žingsniai yra neefektyvūs, nes lemia dvigubai didesnius trajektorijų ilgio pokyčius. Taip pat galima teigti, kad 0,5 min ir 1 min laiko žingsniai yra lygiaverčiai ir apskaičiuoja panašaus optimalumo trajektorijas (didžiausias skirtumas tarp trajektorijų ilgio pokyčių yra 0,8 %). Kadangi rezultatai yra panašūs, siekiant pasirinkti geresnį laiko žingsnį, atliktas papildomas laiko žingsnio įtakos skaičiavimų trukmei tyrimas, kurio rezultatai pavaizduoti 4.12 pav.



4.12 pav. Skaiciavimų trukmės priklausomybė nuo laiko žingsnio ir simuliacijų gylio (Sudaryta autoriaus, 2020)

Galima teigti, kad esant mažesniems gyliams, skaičiavimų trukmė nepriklauso nuo pasirinkto laiko žingsnio. Vis dėlto, kaip matyti iš konfliktų sprendimo rezultatų, kai kuriems konfliktų tipams būtinas didesnis simuliacijų gylis. Kai $p > 8$, 1 min dydžio laiko žingsnis yra pranašesnis ir sutrumpina skaičiavimų trukmę ~4,5 s lyginant su kitais nagrinėtais laiko žingsniais.

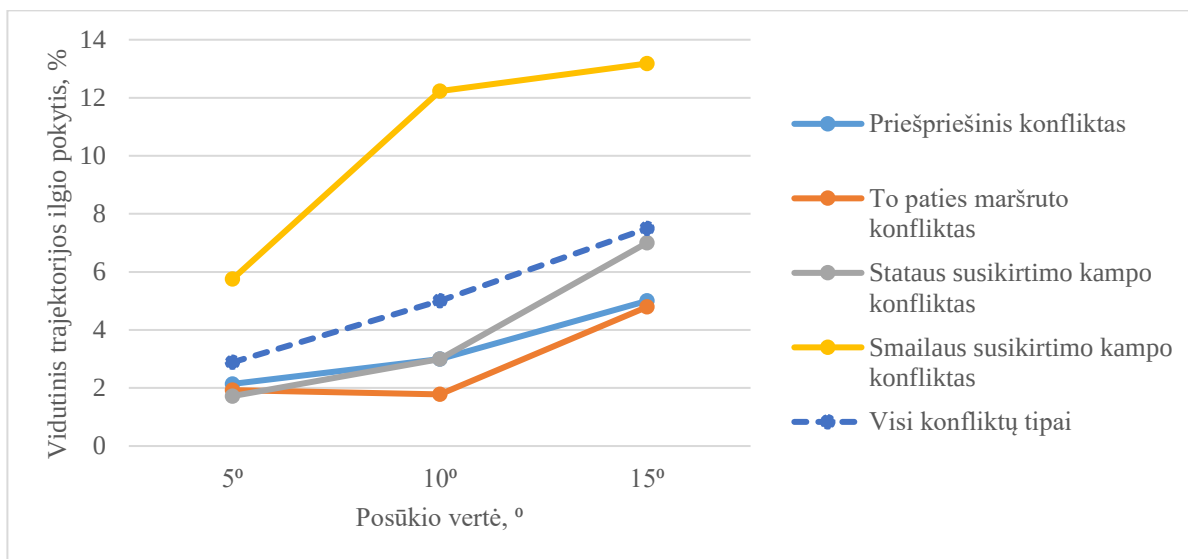
Laiko žingsnio pasirinkimas svarbus trajektorijos ilgio pokyčiui ir keičiant orlaivių greičius. Atliktame tyrime nagrinėti orlaivių greičiai nuo 300 kt iki 500 kt, nes šis greičių diapazonas yra būdingas tiriamai oro erdvei. 4.13 pav. matomi šio tyrimo rezultatai. Siekiant apjungti tris atskirus grafikus, šiame grafike naudojamas ir X ašyje žymimas naujas dydis – atstumas, nuskrendamas per laiko žingsnį. Šis dydis yra orlaivio greičio ir laiko žingsnio sandauga.



4.13 pav. Vidutinio trajektorijos ilgio pokyčio priklausomybė nuo orlaivių greičių ir laiko žingsnių (Sudaryta autoriaus, 2020)

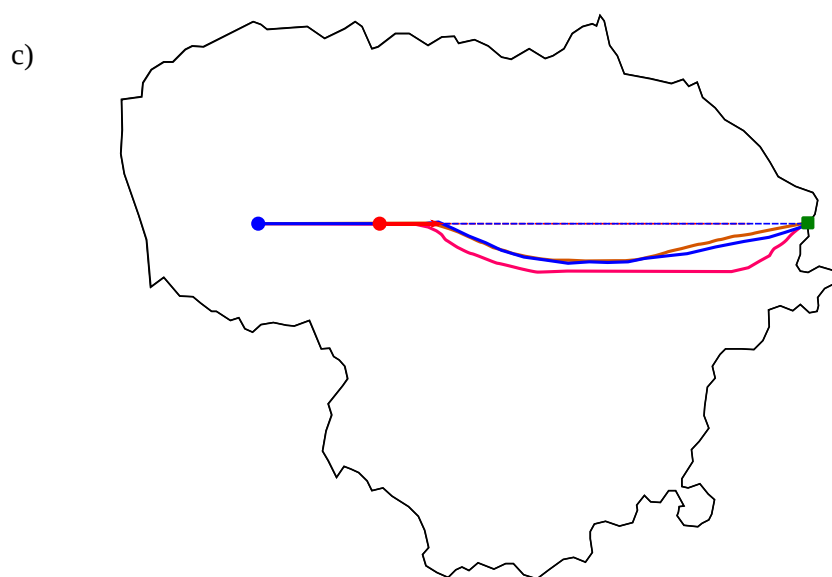
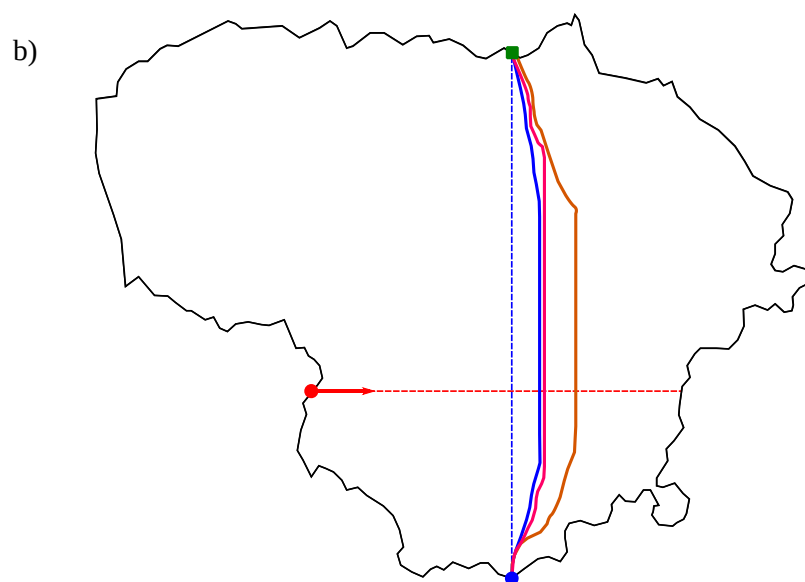
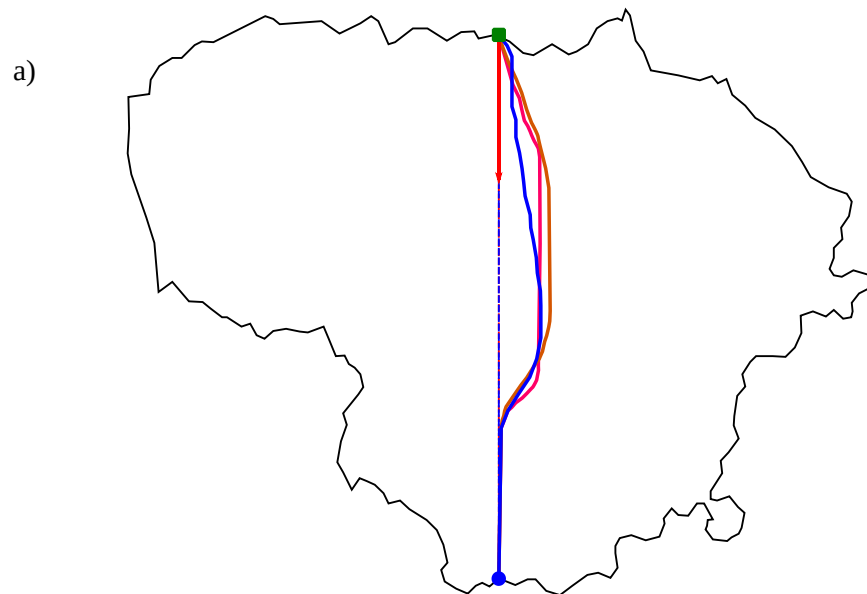
Aukščiau pateikti rezultatai atspindi panašias tendencijas, kurios buvo aptiktos ankstesniuose laiko žingsnio tyrimuose – 0,5 min ir 1 min laiko žingsniai lemia beveik identiškus trajektorijos ilgio pokyčius, tuo tarpu 1,5 min laiko žingsnis lemia 1-1,5 % didesnį trajektorijos ilgio pokytį lyginant su mažesniais laiko žingsniais, kai orlaivių greičiai yra atitinkamai vienodi. Šiuo tyrimu taip pat aptikta naujas įtakos galutiniam sprendimui šaltinis – orlaivių greičiai. Kai tiriami greičiai priklauso pirmajai greičių diapazono daliai (300-400kt), trajektorijos ilgio pokytis auga, tačiau tiriant didesnius greičius trajektorijos ilgio pokytis drastiškai mažėja (~2 kartus). Tokius rezultatus galima aiškinti tuo, kad esant dideliems greičiams, orlaiviai pajuda didesnius atstumus per pasirinktą laiko žingsnį, todėl sumažėja poreikis dažnai keisti orlaivio kursą, t. y. konfliktas išsprendžiamas su mažesniu veiksmų (posūkių) kiekiu.

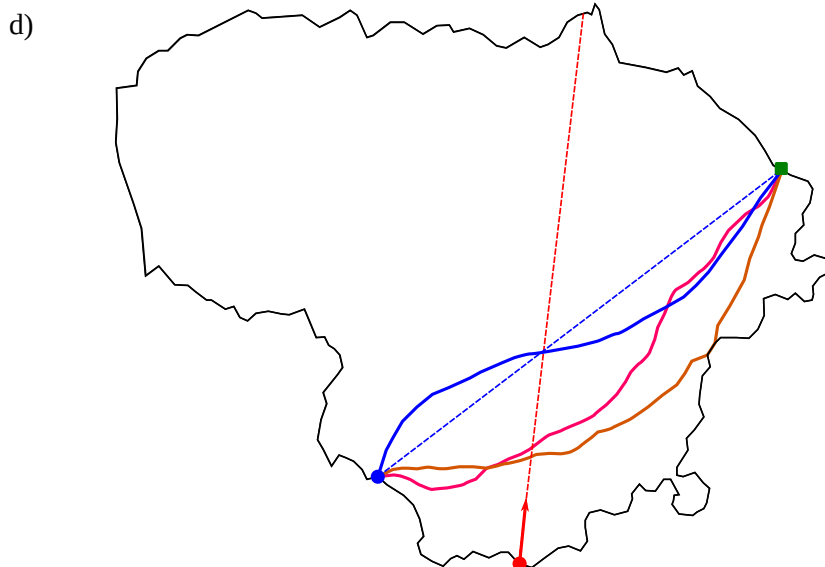
Remiantis aukščiau apžvelgtais rezultatais, pasirinktas 1 min laiko žingsnis ir atliktas papildomas algoritmo sprendimo efektyvumo tyrimas. Sprendimo efektyvumui tyrime buvo nagrinėjamos trys posūkio vertės - 5°, 10°, 15°. Didesnės posūkio vertės nebuvo tirtos, nes gali lemti orlaivio įvedimą į sukimąsi orbitoje. Sprendimo efektyvumui įvertinti tirti visi konfliktų tipai, kai orlaivių greičiai yra vienodi ir lygūs 400 kt. Šis apribojimas pasirinktas remiantis eksperimentų rezultatais, kurie rodo, kad kai orlaivių greičiai skiriasi iki 50 kt, trajektorijų ilgių pokyčiai kinta iki 0,5 %, todėl į tai galima neatsižvelgti, o didesni greičių skirtumai lemia specifines situacijas ir yra gana reti. Šių konfliktų trajektorijų pokyčio priklausomybė nuo leistinos posūkio vertės matoma 4.14 pav.



4.14 pav. Vidutinio trajektorijų ilgių pokyčio priklausomybė nuo posūkio vertės (Sudaryta autoriaus, 2020)

Skirtingų sprendimo trajektorijų pavyzdžiai keturiems konfliktų tipams pavaizduoti 4.15 pav.





4.15 pav. Skirtingo leistino posūkio kampo trajektorijos, kai: a) priešpriešinis konfliktas; b) stataus trajektorijų susikirtimo kampo konfliktas; c) to paties maršruto konfliktas; d) smailaus trajektorijų susikirtimo kampo konfliktas (Sudaryta autoriaus, 2020)

Iš trajektorijų pokyčio tyrimo rezultatų matyti, kad 5° ir 10° posūkiai lemia panašius rezultatus, išskyrus smailaus trajektorijų susikirtimo kampo konfliktus. Pastarojo tipo konflikte 10° ir 15° posūkiai lemia 2 kartus didesnę trajektorijos pailgėjimą. Vidutinis visų konfliktų tipų trajektorijos pokytis tiesiškai priklauso nuo pasirinktos posūkio vertės. Siekiant užtikrinti kiek įmanoma panašesnį sprendimo efektyvumą visiems konfliktų tipams ir remiantis vidutinio pokyčio rezultatais, galima teigti, kad rekomenduojama posūkio vertė yra 5° , nes ji užtikrina mažiausią trajektorijos ilgio pokytį, taigi, ir tolygesnę trajektoriją.

4.3. Tyrimo rezultatų apibendrinimas ir rekomendacijos

Šiame poskyryje pateikiami apibendrinti tiriamosios dalies rezultatai, jų palyginimas su standartiniu skrydžių vadovo sprendimu, nurodomi algoritmo privalumai ir trūkumai, įvairių parametrų įtaka gaunamiems rezultatams ir pateikiamos rekomendacijos tolimesniems tyrimams.

Tiriamosios dalies rezultatų ir siūlomo modelio charakteristikų palyginimas su skrydžių vadovo taikomo standartinio (neautomatizuoto) sprendimo rezultatais pateikiami žemiau esančioje 4.12 lentelėje. Lyginant trajektorijų ilgių pokyčius daroma prielaida, kad skrydžių vadovo parinktas sprendimas yra optimalus, t.y. užtikrina minimalų orlaivių atskyrimą ir laiku pasuka valdomą orlaivį į jo tikslą. Pabrėžtina, kad standartinis sprendimas stataus orlaivių trajektorijų susikirtimo kampo konfliktams, kai orlaivis įsibrovėlis yra greitesnis, įprastose situacijose yra taikomas, tačiau šiame darbe tirtas būtent simetrinis konfliktas, todėl rezultatų palyginti negalima būtent dėl šios konflikto ypatybės.

4.12 lentelė. Siūlomo modelio palyginimas su standartiniu sprendimu

Sprendimo tipas Charakteristika		Dinaminio modelio sprendimas	Standartinis sprendimas
Vidutinis trajektorijos ilgio pokytis	Priešpriešinis konfliktas	2,14 %	2 %
	To paties maršruto konfliktas	1,93 %	1,42 %
	Stataus kampo konfliktas	0,9 % (kai greičiai vienodi) 22,84 % (kai įsibrovėlis greitesnis)	0,4 % (kai greičiai vienodi) NETAIKOMAS DĖL SIMETRIŠKUMO
	Smailaus kampo konfliktas	5,75 % (kai greičiai vienodi) 11,85 % – 19,2 % (kai valdomas greitesnis 25 %) 1,56 % – 17,68 % (kai valdomas greitesnis 50 %)	NETAIKOMAS
Trumpesnės trajektorijos parinkimas		PRIKLAUSO NUO ALGORITMO NUSTATYMŲ	TAIP
Galimybė spręsti simetrinius konfliktus		TAIP	NE
Galimybė užtikrinti pasirinktą minimalų atskyrimą		PRIKLAUSO NUO ALGORITMO NUSTATYMŲ	NE
Galimybė patikrinti kelis sprendimo variantus		TAIP	NE
Vėjo įtakos įtraukimas į trajektorijos parinkimą		TAIP	NE
Matematinis manevrų parinkimas ir manevrų geometrija		SUDĖTINGA	PAPRASTA
Išvengiamosios trajektorijos tolygumas		DIDELIS	MAŽAS

Apibendrinant rezultatus galima daryti išvadą, kad algoritmas orlaivių konfliktus sprendžia pakankamai efektyviai – pasirinkus optimalius algoritmo nustatymus minimalus atstumas tarp orlaivių išlaikomas 7-8 NM, o išvengiamosios trajektorijos savo ilgiu nuo standartinių sprendimų skiriasi nežymiai (ne daugiau nei 1,32 %). Svarbu pabrėžti tai, kad realybėje skrydžių vadovo sprendimas nebūtinai yra pakankamai optimalus, todėl išlieka tikimybė, kad siūlomas modelis gali parinkti geresnius sprendimus. Algoritmo parinktoms trajektorijoms būdingas trajektorijos tolygumo privalumas. Be to, algoritmas gali parinkti horizontalius maneuvrus smailaus trajektorijų susikirtimo

kampo konfliktui išspręsti. Minėtojo tipo konfliktams spręsti standartinis skrydžių vadovo sprendimas dažniausiai yra aukščio keitimas. Nors algoritmo apskaičiuojamos trajektorijos šiems konfliktams yra ilgesnės nei kitiems konfliktų tipams, kai aukščio keitimas nėra pageidaujamas, algoritmas yra pranašesnis už skrydžių vadovą. Naudojant rekomenduotus algoritmo nustatymus sprendimas apskaičiavimas užtrunka ne ilgiau nei 9 s. Lyginant su literatūros analizėje apžvelgtais modeliais, siūlomas algoritmas randa sprendimą 1 s greičiau nei matematinio programavimo modelis (ir apskaičiuoja daug sudėtingesnę bei tolygesnę trajektoriją). Dar vienas algoritmo privalumų yra gebėjimas ištirti didelį kiekį galimų trajektorijų – ši charakteristika itin reikšminga pučiant vėjui, kai algoritmas priklausomai nuo vėjo parametrų gali užtikrinti vienodą trajektorijos ilgio pokytį. Vis dėlto, algoritmas turi ir trūkumų. Vienas pagrindinių algoritmo trūkumų yra jo priklausomybė nuo parinktų nustatymų. Norint išspręsti sudėtingą konfliktą arba naudoti itin didelį kiekį žingsnių, skaičiavimų trukmė auga eksponentiškai, todėl keičiant algoritmo nustatymus svarbu pasirinkti, kam teikiama pirmenybė – trajektorijos ilgiui, tolygumui ar skaičiavimų trukmei. Naudojant simuliacijų gylį $p \geq 20$, algoritmas nebegali apskaičiuoti net ir paprasčiausio sprendimo. Sprendimas taip pat ne visada apskaičiuojamas pasirinkus itin didelį laiko žingsnį ar posūkio vertę, didinant šiuos nustatymus didėja priverstinio orlaivio skrydžio orbita tikimybė. Kitas algoritmo trūkumas yra atsitiktinumo įtaka apskaičiuojamiems sprendimams. Nors ši algoritmo charakteristika turi ir privalumų (ženkliai sutrumpina skaičiavimų trukmę, leidžia ištirti daug skirtingų trajektorijų), ji taip pat gali lemti apskaičiuoto sprendimo neoptimalumą. Visuomet išlieka tikimybė, kad algoritmas vien dėl atsitiktinumo nerado ir nepatikrino geriausios iš galimų trajektorijų ir galutinė išvengiamaoji trajektorija yra ne itin tolygi (žr. 4.7. a) pav.).

Lyginant siūlomą adaptuotą dinaminį Monte Karlo modelį su anksčiau minėtu X. Yango ir P. Wei'o (2018) modeliu, kurio idėjos buvo panaudotos sprendžiant išsikeltą problemą galima teigti, kad skaičiavimų trukmė žymiai skiriasi. Kadangi mokslininkų tyrime buvo naudotos tik žemo lygio simuliacijos, galima palyginti tik vieną sutampantį gylį $p = 4$: keičiant simuliacijų kiekius skaičiavimų trukmės kitimo tendencija yra ta pati, tačiau 10 kartų mažesnė nei siūlomo adaptuoto modelio. Tai galima aiškinti papildomų apribojimų (trajektorijos ilgio, nuokrypio kampo) ir skirtingo atlygio (orlaivių susidūrimui skiriama didesnė nuobauda, todėl algoritmas privalo tirti ir atmesti daugiau netinkamų trajektorijų) įtraukimu į siūlomą modelį. Kitų rezultatų palyginti neįmanoma, nes skiriasi tyrimų tikslai ir objektai – mokslininkų tyrime nagrinėjama, kaip kinta orlaivių susidūrimų ar suartėjimų skaičius didinant įsibrovėlių skaičių bei kaip dažnai valdomi orlaiviai pasiekia savo tikslą ir visiškai nenagrinėjamas trajektorijos ilgio aspektas, tuo tarpu šiame tyrime orlaivių suartėjimas arba tikslo nepasiekimas yra draudžiamas.

Algoritmo skaičiavimų sparta priklauso nuo 3 parametrų: simuliacijų gylio, kiekio, laiko žingsnio ir orlaivių greičių. Naudojant žemus simuliacijų gylius ($p < 4$) algoritmas sprendimą randa

per greičiau nei 1 s, toliau kiekvienas papildomas gylis skaičiavimų trukmę dvigubina. Ši tendencija būdinga ir simuliacijų kiekiui (skaičiavimų trukmė pradeda dvigubėti, kai $k > 800$). Skaičiavimų trukmė nuo laiko žingsnio priklauso tiesiškai, išskyrus 1 min laiko žingsnį, kuris grafiko kreivę suploksština esant didesniems gyliams. Didėjant orlaivių greičiams orlaivis tikslą pasiekia greičiau, todėl yra mažiau žingsnių ir skaičiavimų trukmė tiesiškai mažėja.

Algoritmo apskaičiuotas sprendimas yra priklausomas taip pat nuo 3 parametrų: simuliacijų gylis, laiko žingsnio ir leidžiamo posūkio. Taip pat apskaičiuotų trajektorijų ilgio pokytis priklauso ir orlaivių greičių, tačiau pastarieji nėra algoritmo parametrai (nustatymai), ir jų įtaka šiame tyrime nėra pilnai ištirta. Simuliacijų kiekio įtaka tirtų situacijų trajektorijoms dėl mažos leistinų veiksmų erdvės ir nedidelių gylių laikoma nežymia. Minėtųjų parametrų ir orlaivių greičių įtaka išvengiamajai trajektorijai pateikta 4.13 lentelėje. Pabrėžtina, kad simetriniams, kaip retiems, sudėtingiems ir mažai tyrinėtiems konfliktams parametrų įtaka čia nėra nagrinėjama ir apibendrinama.

4.13 lentelė. Parametrų įtaka konflikto išvengiamajai trajektorijai.

Parametras	Situacija		Parametro įtaka trajektorijos ilgio pokyčiui
Simuliacijų gylis	Konflikto geometrija paprasta ir orlaivių greičiai:	Vienodi	NEDIDELĖ (skirtumas iki 0,5 %)
		Skirtingi	NEDIDELĖ (skirtumas iki 0,5 %)
	Konflikto geometrija sudėtinga ir orlaivių greičiai:	Vienodi	NEDIDELĖ (skirtumas iki 0,5 %)
		Skirtingi	DIDELĖ (skirtumas iki 15 %)
Orlaivių greičiai	300-400 kt		ILGIO POKYTIS DIDĖJA
	400-500 kt		ILGIO POKYTIS MAŽĖJA (~2 kartus)
Laiko žingsnis	Iki 1 min		NEDIDELĖ (skirtumas iki 0,8 %)
	Virš 1 min		DIDELĖ (~2 kartus didėja)
Leistina posūkio vertė	Didinama kas 5°		ILGIO POKYTIS DIDĖJA (~2 %)

Apibendrinant pateiktus rezultatus galima teigti, kad didžiausią įtaką sprendimui simuliacijų gylis turi tuomet, kai tiriama konflikto geometrija itin sudėtinga ir orlaivių greičiai skiriasi 50 kt ir daugiau. Orlaivių greičių proporcingas didinimas iš pradžių lemia nežymų trajektorijos ilgio didėjimą, o vėliau drastišką kritimą, tuo tarpu didesnis nei 1 min laiko žingsnis lemia trajektorijos ilgio padvigubėjimą. Trajektorijos ilgio pokytis nuo leistinos posūkio vertės priklauso tiesiškai ir didėja vidutiniškai 2 % kas 5°.

Apžvelgus visus tyrimo rezultatus, siūlomo modelio privalumus ir trūkumus galima teigti, kad siūlomas dinaminis orlaivių konfliktų sprendimo modelis yra gana perspektyvus ir vertas tolimesnių tyrimų. Šiame tyrime mažai nagrinėti simetriniai konfliktai. Viena nagrinėta situacija įrodo algoritmo galimybę spręsti tokio tipo konfliktus, tačiau tolimesni simetrinių konfliktų optimizavimo tyrimai yra

būtinai siekiant išsiaiškinti algoritmo galimybes ir ribas šioje srityje. Taip pat tyrime nenagrinėti galimi orlaivių greičių pokyčiai vietoj kurso pokyčių arba leistinos veiksmų erdvės didinimas bei šių aspektų įtaka algoritmo veikimui. Be to, didelį potencialą turi ir šio modelio taikymas daugiau nei dviejų orlaivių konflikto sprendimui. Nors panašūs modeliai yra naudoti kelių bepiločių orlaivių konfliktų sprendimams, vis dėlto trajektorijos ilgio ar kitų parametrų optimizavimas tokiose situacijose nėra nagrinėtas. Papildomi tyrimai reikalingi ir siekiant išsiaiškinti parametrų įtaką esant skirtingiems greičiams, nes šiame darbe simuliacijų gylis ir posūkio vertės įtaka buvo tiriama orlaiviams skrendant vienu 400 kt greičiu. Taip pat reikalinga papildoma parametrų jautrumo analizė. Norint geriau įvertinti algoritmo privalumus lyginant su žmogaus priimamais sprendimais, rekomenduojama atlikti lyginamąją analizę atsižvelgiant į konfliktų išvengiamąsias trajektorijas realioje aplinkoje (kai konfliktą sprendžia skrydžių vadovas).

IŠVADOS

1. Apžvelgus mokslinę literatūrą pastebėta, kad klasikiniai metodai tiriamai problemai spręsti yra netinkami, nuoseklių trajektorijų taikymas dviejų orlaivių konfliktų sprendime nėra pakankamai ištirtas ir tyrimuose beveik nenagrinėjamas tokių trajektorijų ilgio minimizavimas.
2. Sukurtas dviejų orlaivių konflikto plokštumoje sprendimo modelis paremtas dinaminio programavimo ir Monte Karlo paieškos metodo kombinacija. Šiame modelyje nustatyti 3 parametrai, turintys įtaką išvengiamųjų trajektorijų ilgiui, tačiau reikalingi papildomi tyrimai siekiant įvertinti jų įtaką prie skirtingų orlaivių greičių. Siūlomo modelio parenkamos trajektorijos yra sudarytos iš didelio tiesių atkarpų kiekio, todėl orlaivis nuosekliai išvengia konflikto ir yra pažingsniui grąžinamas į maršrutą.
3. Remiantis tyrimų rezultatais, rekomenduojami simuliacijų gyliai yra 4-10, kurie gali būti pasirenkami priklausomai nuo sprendžiamo konflikto tipo. Simuliacijų gylis, tinkamas visiems konflikto tipams ir užtikrinantis trumpesnės trajektorijos parinkimą, yra $p = 10$.
4. Nustatyta, kad optimalūs algoritmo parametrai, užtikrinantys kuo mažesnę trajektorijos ilgio pokytį ir skaičiavimų trukmę, kai orlaivių greičiai yra vienodi ($v = 400$ kt), yra šie: simuliacijų gylis $p = 10$, laiko žingsnis 1 min, posūkio vertė $\alpha = 5^\circ$.
5. Pastebėta, kad orlaivių greičiai turi įtakos trajektorijų ilgiui ir trumpiausios trajektorijos gaunamos, kai $v > 400$ kt, tačiau reikalingi papildomi tyrimai orlaivių greičių ir algoritmo parametrų sąveikai įvertinti.
6. Tyrimo metu pastebėta, kad sudėtingos geometrijos konfliktai yra itin jautrūs algoritmo nustatymams (parametrms). Nustatyta, kad didžiausią įtaką tokių konfliktų sprendimui daro simuliacijų gylis (iki 15 %) ir laiko žingsnis (trajektorijos ilgio pokytis dvigubėja didinant laiko žingsnį).
7. Sukurtas programinis kodas, sudarytas *Python* programavimo kalba, kuriuo gali naudotis mokslininkai bei tyrėjai tolimesniuose tyrimuose.

LITERATŪROS SĄRAŠAS

- Alonso-Ayuso, A., Escudero, L. F., & Martín-Campo, F. J. (2016). Exact and Approximate Solving of the Aircraft Collision Resolution Problem via Turn Changes. *Transportation Science*, 50(1), 263–274. <https://doi.org/10.1287/trsc.2014.0557>
- Askari, A., Mortazavi, M., Talebi, H., & Motamedi, A. (2016). A new approach in UAV path planning using Bezier–Dubins continuous curvature path. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 230(6), 1103–1113. <https://doi.org/10.1177/0954410015603415>
- Bilimoria, K. (2000). A geometric optimization approach to aircraft conflict resolution. *18th Applied Aerodynamics Conference*. <https://doi.org/10.2514/6.2000-4265>
- Cafieri, S. (2014). Maximizing the number of solved aircraft conflicts through velocity regulation. *MAGO 2014, 12th Global Optimization Workshop*. Prieiga per internetą: <https://hal-enac.archives-ouvertes.fr/hal-01018051>
- Cafieri, S. (2017). Mixed-Integer Nonlinear Optimization in Air Traffic Management: Aircraft Conflict Avoidance. In *Advances and Trends in Optimization with Engineering Applications* (pp. 293–301). <https://doi.org/10.1137/1.9781611974683.ch22>
- Cafieri, S., & Durand, N. (2014). Aircraft deconfliction with speed regulation: new models from mixed-integer optimization. *Journal of Global Optimization*, 58(4), 613–629. <https://doi.org/10.1007/s10898-013-0070-1>
- Cafieri, S., & Omheni, R. (2016). Mixed-integer nonlinear programming for aircraft conflict avoidance by sequentially applying velocity and heading angle changes. *European Journal of Operational Research*, 260(1), 283–290. <https://doi.org/10.1016/J.EJOR.2016.12.010>
- Chaimatanan, S., Delahaye, D., & Mongeau, M. (2014). A Hybrid Metaheuristic Optimization Algorithm for Strategic Planning of 4D Aircraft Trajectories at the Continental Scale. *IEEE Computational Intelligence Magazine*, 9(4), 46–61. <https://doi.org/10.1109/MCI.2014.2350951>
- Chaudhari, A., Chatterjee, D., & Rajput, R. (2015). *Fuzzy Mixed Integer Linear Programming for Air Vehicles Operations Optimization*. Prieiga per internetą: <https://arxiv.org/ftp/arxiv/papers/1503/1503.04222.pdf>
- Donciu, B., & Pleter, O. T. (2009). Free Flight vs. Centralized Air Traffic Management. *INCAS BULLETIN*, 3(4), 67–75. <https://doi.org/10.13111/2066-8201.2011.3.4.7>
- Durand, N., Alliot, J.-M., Noailles, J. (1996). *Automatic aircraft conflict resolution using Genetic Algorithms*. 289–298. <https://doi.org/10.1145/331119.331195i>
- Erzberger, H. (2005). *Automated Conflict Resolution For Air Traffic Control*. Prieiga per internetą: <https://ntrs.nasa.gov/search.jsp?R=20050242942>

EUROCONTROL Seven-Year Forecast Network Manager nominated by the European Commission EUROCONTROL. (2019). Prieiga per internetą: <https://www.eurocontrol.int/sites/default/files/content/documents/official-documents/forecasts/eurocontrol-7-year-forecast-february-2019-main-report.pdf>

Geser, A., & Munoz, C. (2002). A geometric approach to strategic conflict detection and resolution [ATC]. *Proceedings. The 21st Digital Avionics Systems Conference*, 1, 6B1-1-6B1-11. <https://doi.org/10.1109/DASC.2002.1067985>

Girardet, B., Lapasset, L., Delahaye, D., & Rabut, C. (2014). Wind-optimal path planning: Application to aircraft trajectories. *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, 1403–1408. <https://doi.org/10.1109/ICARCV.2014.7064521>

Hoffman, E., Bonnans, F., Blin, K., & Zeghal, K. (2001). Conflict resolution in presence of uncertainty - A case study of decision making with dynamic programming. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. <https://doi.org/10.2514/6.2001-4235>

Hu, J., Prandini, M., & Sastry, S. (2002). Optimal Coordinated Maneuvers for Three-Dimensional Aircraft Conflict Resolution. *Journal of Guidance, Control, and Dynamics*, 25(5), 888–900. <https://doi.org/10.2514/2.4982>

Hu, J., Sastry, S., & Prandini, M. (2003). Aircraft Conflict Detection in Presence of Spatially Correlated Wind Perturbations, *AIAA Guidance, Navigation and Control Conference*. <https://doi.org/10.2514/6.2003-5339>

Hurter, C., Alligier, R., Gianazza, D., Puechmorel, S., Andrienko, G., & Andrienko, N. (2014). Wind parameters extraction from aircraft trajectories. *Computers, Environment and Urban Systems*, 47, 28–43. <https://doi.org/10.1016/J.COMPENVURBSYS.2014.01.005>

Kikutis, R., Stankūnas, J., Rudinskas, D., & Masiulionis, T. (2017). Adaptation of Dubins Paths for UAV Ground Obstacle Avoidance When Using a Low Cost On-Board GNSS Sensor. *Sensors*, 17(10), 2223. <https://doi.org/10.3390/s17102223>

Kochenderfer, M. J., & Chryssanthacopoulos, J. P. (2011). *Robust Airborne Collision Avoidance through Dynamic Programming*. (ATC-371). Prieiga per internetą: http://www.ll.mit.edu/mission/aviation/publications/publication-files/atc-reports/Kochenderfer_2011_ATC-371_WW-21458.pdf

Kocsis, L., & Szepesvári, C. (2006). Improved Monte-Carlo search. *Univ. Tartu, Estonia, Tech. Rep 1*.

Krozel, J., & Peters, M. (1997). Strategic conflict detection and resolution for free flight. *Proceedings of the 36th IEEE Conference on Decision and Control*, 2, 1822–1828. <https://doi.org/10.1109/CDC.1997.657844>

Kuchar, J. K., & Yang, L. C. (2000). A review of conflict detection and resolution modeling

methods. *IEEE Transactions on Intelligent Transportation Systems*, 1(4), 179–189. <https://doi.org/10.1109/6979.898217>

Lehouillier, T., Soumis, F., Omer, J., & Allignol, C. (2016). Measuring the interactions between air traffic control and flow management using a simulation-based framework. *Computers & Industrial Engineering*, 99, 269–279. <https://doi.org/10.1016/J.CIE.2016.07.025>

Lovato, A., Fontes, C., Embiruçu, M., & Kalid, R. (2018). A fuzzy modeling approach to optimize control and decision making in conflict management in air traffic control, *Computers & Industrial Engineering* 115: 167–189. <https://doi.org/10.1016/J.CIE.2017.11.008>

Lugo-Cardenas, I., Flores, G., Salazar, S., & Lozano, R. (2014). Dubins path generation for a fixed wing UAV. *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, 339–346. <https://doi.org/10.1109/ICUAS.2014.6842272>

Luo, B., Liu, D., Huang, T., Yang, X., & Ma, H. (2017). Multi-step heuristic dynamic programming for optimal control of nonlinear discrete-time systems. *Information Sciences*, 411, 66–83. <https://doi.org/10.1016/J.INS.2017.05.005>

Menon, P. K., Sweriduk, G. D., & Sridhar, B. (1999). Optimal Strategies for Free-Flight Air Traffic Conflict Resolution. *Journal of Guidance, Control, and Dynamics*, 22(2), 202–211. <https://doi.org/10.2514/2.4384>

Miyazawa, Y., Wickramasinghe, N. K., Harada, A., & Miyamoto, Y. (2013). Dynamic Programming Application to Airliner Four Dimensional Optimal Flight Trajectory. *AIAA Guidance, Navigation, and Control (GNC) Conference*. <https://doi.org/10.2514/6.2013-4969>

Omer, J. (2015). A space-discretized mixed-integer linear model for air-conflict resolution with speed and heading maneuvers. *Computers Operations Research*, 58, 75–86. <https://doi.org/10.1016/j.cor.2014.12.012>

Omer, J., & Farges, J.-L. (2013). Hybridization of Nonlinear and Mixed-Integer Linear Programming for Aircraft Separation With Trajectory Recovery. *IEEE Transactions on Intelligent Transportation Systems*, 14(3), 1218–1230. <https://doi.org/10.1109/TITS.2013.2257758>

Paielli, R. A. (2003). Modeling Maneuver Dynamics in Air Traffic Conflict Resolution. *Journal of Guidance, Control, and Dynamics*, 26(3), 407–415. <https://doi.org/10.2514/2.5078>

Park, J.-W., Oh, H.-D., & Tahk, M.-J. (2008). UAV collision avoidance based on geometric approach. *2008 SICE Annual Conference*, 2122–2126. <https://doi.org/10.1109/SICE.2008.4655013>

Radanovic, M., Javier, F., Nieto, S., Angel, M., Eroles, P., Koca, T., & Nieto, S. (2017). *Self-Reorganized Supporting Tools for Conflict Resolution in High-Density Airspace Volumes*. Prieiga per internetą: <https://www.researchgate.net/publication/319109538>

Raghunathan, A. U., Gopal, V., Subramanian, D., Biegler, L. T., & Samad, T. (2004). Dynamic Optimization Strategies for Three-Dimensional Conflict Resolution of Multiple Aircraft. *Journal of*

Guidance, Control, and Dynamics, 27(4), 586–594. <https://doi.org/10.2514/1.11168>

Rey, D., Rapine, C., Constans, S., & Fondacci, R. (2010). *A Mixed Integer Linear Model for Potential Conflict Minimization by Speed Modulations*. Retrieved from <https://hal.archives-ouvertes.fr/hal-00520052/>

Sathyan, A., Ernest, N. D., & Cohen, K. (2015). An Efficient Genetic Fuzzy Approach to UAV Swarm Routing. *Unmanned Systems*, 04(02), 117–127. <https://doi.org/10.1142/s2301385016500011>

Sathyan, A., Ernest, N., Lavigne, L., Cazaurang, F., Kumar, M., & Cohen, K. (2017). *A Genetic Fuzzy Logic Based Approach to Solving the Aircraft Conflict Resolution Problem*. (December 2016). <https://doi.org/10.2514/6.2017-1751>

Shanmugavel, M., Tsourdos, A., Zbikowski, R., & White, B. (2005). Path Planning of Multiple UAVs Using Dubins Sets. *AIAA Guidance, Navigation, and Control Conference and Exhibit*. <https://doi.org/10.2514/6.2005-5827>

Tomlin, C., Pappas, G. J., & Sastry, S. (1998). Conflict resolution for air traffic management: a study in multiagent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4), 509–521. <https://doi.org/10.1109/9.664154>

Trapsilawati, F., Chen, C. H., & Khoo, L. P. (2016). An investigation into conflict resolution and trajectory prediction AIDS for future air traffic control. *Advances in Transdisciplinary Engineering*, 4, 503–512. <https://doi.org/10.3233/978-1-61499-703-0-503>

Wang, X., Fu, M., Ma, H., & Yang, Y. (2015). Lateral control of autonomous vehicles based on fuzzy logic. *Control Engineering Practice*, 34, 1–17. <https://doi.org/10.1016/J.CONENGPRAC.2014.09.015>

Wang, Z., Wu, M., Wen, X., & Jiang, X. (2019). An Aircraft Conflict Resolution Model Based on Geometric Optimization. *Journal of Physics: Conference Series*, 1168(2), 022034. <https://doi.org/10.1088/1742-6596/1168/2/022034>

Yang, X., & Wei, P. (2018). Autonomous On-Demand Free Flight Operations in Urban Air Mobility using Monte Carlo Tree Search. *International Conference on Research in Air Transportation (ICRAT)*, Barcelona, Spain, (August).

A PRIEDAS

Orlaivių konflikto sprendimo algoritmas

```
SCALAR = 1 / math.sqrt(2.0)
```

```
intruder_size = 1
```

```
state_size = intruder_size * 4 + 5 + 2
```

```
action_size = 3
```

```
RADIUS = 10
```

```
no_frame = 5
```

```
tick = 30
```

```
flexibility_angle = 1
```

```
np.set_printoptions(precision=2)
```

Būsena:

```
def __init__(self, state=np.zeros(state_size, dtype=float), collide_wall=False,  
collide_intruder=False,
```

```
reach_goal=False, prev_action=None, depth=0):
```

```
self.state = state # pozicijos, orlaivių greičiai, tikslo pozicija
```

```
self.prev_action = prev_action
```

```
self.depth = depth
```

```
def next_state(self, a):
```

```
return PROCEED(self.state, a, self.depth)
```

```
def terminal(self):
```

```
if self.reach_goal or self.collide_intruder or self.collide_wall or self.depth ==  
search_depth:
```

```
return True
```

```
return False
```

```
def reward(self):
```

```
if self.collide_wall:
```

```
r = 0
```

```
elif self.collide_intruder:
```

```
r = 0
```

```
elif self.reach_goal:
```

```

        r = +1
    else:
        r = 1 - self.dist_goal() / 1200.0
    return r

@property
def ownx(self): # valdomo orlaivio pozicija x
    return self.state[intruder_size * 4 + 0]
@property
def owny(self): # valdomo orlaivio pozicija y
    return self.state[intruder_size * 4 + 1]
@property
def goalx(self): # tikslo pozicija x
    return self.state[intruder_size * 4 + 5]
@property
def goaly(self): # tikslo pozicija y
    return self.state[intruder_size * 4 + 6]
def dist_goal(self): # atstumas tarp valdomo orlaivio ir tikslo
    return metric(self.ownx, self.owny, self.goalx, self.goaly)
def dist_intruder(self): # atstumas tarp valdomo orlaivio ir orlaivio įsibrovėlio
    distance = 5000
    for i in range(intruder_size):
        intrux = self.state[4 * i + 0]
        intruy = self.state[4 * i + 1]
        current_dist = metric(intrux, intruy, self.ownx, self.owny)
        if current_dist < distance:
            distance = current_dist
    return distance
def __hash__(self):
    this_str = str(self.state) + str(self.collide_wall) + str(self.collide_intruder) +
str(self.reach_goal)
    return int(hashlib.md5(this_str.encode('utf-8')).hexdigest(), 16)
def __eq__(self, other): # ar būsenos vienodos
    if hash(self) == hash(other):
        return True

```

```

        return False

def __repr__(self):
    s = "prev action: %d, own: (%f, %f), goal: (%f, %f), dist goal: %f, dist intruder: %f depth:
%d" \

        % (self.prev_action,
            self.ownx,
            self.owny,
            self.goalx,
            self.goaly,
            self.dist_goal(),
            self.dist_intruder(),
            self.depth)

    return s

```

Paieškos taškas:

```

def __init__(self, state, parent=None):
    self.visits = 0 # kiek kartų aplankytas taškas
    self.reward = 0.0 # sukauptas taško atlygis
    self.reward / self.visits # taško vidurkis
    self.state = state # būseną susijusi su tašku
    self.children = [] # taško palikuonys
    self.parent = parent # pradinis taškas
    self.untried_action = [0, 1, 2] # kuris iš veiksmų netirtas
def add_child(self, child_state):
    child = Node(child_state, self)
    self.children.append(child)
def update(self, reward): # atlygio grąžinimas
    self.reward += reward # kaupiamo atlygio padidėjimas
    self.visits += 1 # aplankymo padidėjimas
# jeigu visi veiksmai išbandyti
def fully_expanded(self):
    if len(self.children) == action_size:
        return True
    return False
def __repr__(self):

```

```

s = "Node: children: %d; visits: %d; reward: %.4f; p_action: %s, state: (%.2f, %.2f); " \
    "goal: (%.2f, %.2f), dist: %.2f, dist2: %.2f" \
    % (len(self.children),
       self.visits,
       self.reward,
       self.state.prev_action,
       self.state.ownx,
       self.state.owny,
       self.state.goalx,
       self.state.goaly,
       self.state.dist_goal(),
       self.state.dist_intruder())

return s

def MCTSSEARCH(budget, root):
    for _ in range(int(budget)):
        # if iter % 10000 == 9999:
        #     print("simulation: %d" % iter)
        #     print(root)
        front = TREEPOLICY(root)
        reward = DEFAULTPOLICY(front.state)
        BACKUP(front, reward)
    return BESTCHILD(root, 0)

def TREEPOLICY(node):
    while node.state.terminal() == False:
        if len(node.children) < 3:
            return EXPAND(node)
        else:
            node = BESTCHILD(node, SCALAR)
    return node

def EXPAND(node):
    random_action = random.choice(node.untried_action) # pasirinkti netirtą veiksmą
    node.untried_action.remove(random_action) # ištrinti veiksmą iš netirtų veiksmų
    new_state = node.state.next_state(random_action) # nustatyti kitą veiksmą
    node.add_child(new_state) # pridėti taško palikuonį
    return node.children[-1]

```

```

def BESTCHILD(node, scalar):
    bestscore = -10
    bestchildren = []
    for c in node.children:
        exploit = c.reward / c.visits
        explore = math.sqrt(2 * math.log(node.visits) / float(c.visits))
        score = exploit + scalar * explore
        if score == bestscore:
            bestchildren.append(c)
        if score > bestscore: # this is the best child
            bestchildren = [c]
            bestscore = score
    if len(bestchildren) == 0:
    if len(bestchildren) > 1: # jeigu yra keli geriausi taškai
        for node in bestchildren:
            if node.state.prev_action == 1:
                return node
    if scalar == 0:
        r = random.choice(bestchildren)
        return r
    return random.choice(bestchildren)

def DEFAULTPOLICY(state):
    while state.terminal() == False: # simuliuoti iki galutinės būsenos
        random_action = random.randint(0, 3)
        state = state.next_state(random_action)
    return state.reward()

def BACKUP(node, reward):
    while node is not None:
        node.visits += 1
        node.reward += reward
        node = node.parent
    return

```

```

def PROCEED(s, a, d):
    state = copy.deepcopy(s)
    collide_wall = False
    collide_intruder = False
    reach_goal = False
    delta_direction = (a - 1) * flexibility_angle # kurso pokytis
    for _ in range(no_frame): # kartojuama tiek, kiek no_frames
        for i in range(intruder_size): # atnaujinti įsibrovėlio poziciją
            i_x += i_vx
            i_y += i_vy
            state[4 * i + 0] += state[4 * i + 2]
            state[4 * i + 1] += state[4 * i + 3]
    direction = state[intruder_size * 4 + 4] + delta_direction
    rad = direction * math.pi / 180
    wx = -0.2 * math.sin(np.pi) # vėjas x
    wy = -0.2 * math.cos(np.pi) # vėjas y
    ownvx = -2 * math.sin(rad) + wx
    ownvy = -2 * math.cos(rad) + wy
    state[intruder_size * 4 + 2] = ownvx
    state[intruder_size * 4 + 3] = ownvy
    state[intruder_size * 4 + 4] = direction
    state[intruder_size * 4 + 0] += ownvx
    state[intruder_size * 4 + 1] += ownvy
    # dabartinės valdomo orlaivio ir tikslo pozicijos
    ownx = state[intruder_size * 4 + 0]
    owny = state[intruder_size * 4 + 1]
    goalx = state[intruder_size * 4 + 5]
    goaly = state[intruder_size * 4 + 6]
    # ar orlaiviai susidūrė
    for i in range(intruder_size):
        x = state[4 * i + 0]
        y = state[4 * i + 1]
        if metric(x, y, ownx, owny) < 32:
            collide_intruder = True

```

```

    # ar pasiektas tikslas
    if metric(ownx, owny, goalx, goaly) < 32:
        reach_goal = True
    return State(state, collide_wall, collide_intruder, reach_goal, a, d + 1)
# atstumas tarp taškų (x1,y1) ir (x2,y2)
def metric(x1, y1, x2, y2):
    dx = x1 - x2
    dy = y1 - y2
    return math.sqrt(dx * dx + dy * dy)

```

Valdomas orlaivis:

```

DroneSprite(pygame.sprite.Sprite):
    def __init__(self, position):
        pygame.sprite.Sprite.__init__(self)
        self.src_image = pygame.image.load('images/drone.png')
        self.rect = self.src_image.get_rect()
        self.image = self.src_image
        self.position = position # pradinė pozicija
        self.speed = 2.2 # greitis px
        self.direction = 0 # pradinis kursas
        self.rad = 0
        vx = -self.speed * math.sin(self.rad)
        vy = -self.speed * math.cos(self.rad)
        self.velocity = (vx, vy) # greičio projekcijos
        self.radius = RADIUS
        self.delta_direction = 0
    def update(self, deltat):
        self.direction += self.delta_direction # naujas kursas
        self.direction %= 360
        self.rad = self.direction * math.pi / 180
        vx = -self.speed * math.sin(self.rad)
        vy = -self.speed * math.cos(self.rad)
        self.velocity = (vx, vy) # naujos greičio projekcijos

```

```

x = self.position[0] + vx
y = self.position[1] + vy
self.position = (x, y) # nauja pozicija
self.image = pygame.transform.rotate(self.src_image, self.direction)
self.rect = self.image.get_rect()
self.rect.center = self.position

```

Orlaivis įsibrovėlis:

PadSprite(pygame.sprite.Sprite):

```

def __init__(self, position, speed, direction):
    pygame.sprite.Sprite.__init__(self)
    self.src_image = pygame.image.load('images/intruder.png')
    self.image = self.src_image
    self.rect = self.src_image.get_rect()
    self.position = position
    self.speed = speed
    self.direction = direction
    self.rad = self.direction * math.pi / 180
    vx = -self.speed * math.sin(self.rad)
    vy = -self.speed * math.cos(self.rad)
    self.velocity = (vx, vy)
    self.radius = RADIUS

def update(self):
    x, y = self.position
    x += self.velocity[0]
    y += self.velocity[1]
    self.position = (x, y)
    self.image = pygame.transform.rotate(self.src_image, self.direction)
    self.rect = self.image.get_rect()
    self.rect.center = self.position

```

Tikslas:

GoalSprite(pygame.sprite.Sprite):

```

def __init__(self, position):
    pygame.sprite.Sprite.__init__(self)
    self.image = pygame.image.load('images/goal.png')

```

```

        self.rect = self.image.get_rect()
        self.position = position
        self.radius = 16
        self.rect = self.image.get_rect()
        self.rect.center = self.position
    def update(self):
        self.rect.center = self.position
def get_state(pads, own, goal):
    state_list = []
    for i in range(intruder_size):
        state_list.append(pads[i].position[0])
        state_list.append(pads[i].position[1])
        state_list.append(pads[i].velocity[0])
        state_list.append(pads[i].velocity[1])
    for i in range(1):
        # valdomo orlaivio informacija
        state_list.append(own.position[0])
        state_list.append(own.position[1])
        state_list.append(own.velocity[0])
        state_list.append(own.velocity[1])
        state_list.append(own.direction)
    # tikslo informacija
    state_list.append(goal.position[0])
    state_list.append(goal.position[1])
    return np.array(state_list)
# atstumas tarp orlaivių
def dist(a, b):
    dx = a.position[0] - b.position[0]
    dy = a.position[1] - b.position[1]
    return math.sqrt(dx * dx + dy * dy)
def reset_intruder(intruder):
    while dist(ownship, intruder) < min_dist:
        intruder.position = (150,470) # orlaivio įsibrovėlio pradinės pozicija
    intruder.speed = 2.2
    intruder.direction = 270 # įsibrovėlio kursas

```

```

intruder.rad = intruder.direction * math.pi / 180
vx = -intruder.speed * math.sin(intruder.rad)
vy = -intruder.speed * math.cos(intruder.rad)
intruder.velocity = (vx, vy)
pads = []
goal = GoalSprite((250,100)) # tikslo pozicija
for _ in range(intruder_size):
    pads.append(PadSprite((150,470), 2.2, 270))
rect = screen.get_rect()
print(rect.center)
ownship = DroneSprite((250,560)) # valdomo orlaivio pradinė padėtis
ownship.direction = 0 # kursas
car_group = pygame.sprite.RenderPlain(ownship)
pad_group = pygame.sprite.RenderPlain(*pads)
goal_group = pygame.sprite.RenderPlain(goal)

budget = 800 # simuliacijų kiekis
search_depth = 4 # simuliacijų gylis
time_step = 0 # laikas
simulate = True

import time
current_time_list = []
#koordinatės
#east_south 53.5070160000000000, 29.545377000000000
#west_south 53.270272613213860, 22.03723140182686
#west_north 56.863367665464665, 22.03723140182686
#east_north 57.099970796890275, 29.545377000000000
df_own = pd.DataFrame(index=np.arange(0, 10), columns=('dates','Lon', 'Lat', 'color') )
df_intr = pd.DataFrame(index=np.arange(0, 10), columns=('dates','Lon', 'Lat', 'color') )
df_final = pd.DataFrame(index=np.arange(0, 10), columns=('dates','Lon', 'Lat', 'color') )

while simulate:
    time_step += 1
    # pradinis laiko momentas

```

```

        next_time      =      datetime.datetime(2017,      11,      28,      23,      55,
59)+datetime.timedelta(minutes=(time_step-1)*0.5)
        current_state = get_state(pads, ownship, goal)
        # dabartinė būsena paverčiama tašką iš px į koordinates
        for intruder in pad_group:
            intr_lat  =  56.863367665464665  -  (intruder.position[1])*(56.863367665464665-
53.270272613213860)/height
            intr_lon  =  22.03723140182686  +  (intruder.position[0])*(29.545377000000000-
22.03723140182686)/width
            df_intr.loc[time_step-1] = [next_time, intr_lon, intr_lat, 'red']
            own_lat  =  56.863367665464665  -  (ownship.position[1])*(56.863367665464665-
53.270272613213860)/height
            own_lon  =  22.03723140182686  +  (ownship.position[0])*(29.545377000000000-
22.03723140182686)/width
            df_own.loc[time_step-1] = [next_time, own_lon, own_lat, 'blue']
            current_node = Node(State(current_state))
            print(next_time.strftime("%Y-%m-%dT%H:%M:%S"))
            # skrydžio laikas
            current_time = int(round(time.time() * 1000))
            # veiksmo parinkimas
            action = UCTSEARCH(budget, current_node).state.prev_action
            # skrydžio laikas
            time_after = int(round(time.time() * 1000))
            # skaičiavimų trukmė
            current_time_list.append(time_after - current_time)
            # kurso perskaičiavimas
            ownship.delta_direction = (action - 1) * flexibility_angle
            for _ in range(no_frame):
                deltat = clock.tick(tick)
                screen.fill((255, 255, 255))
                for intruder in pad_group:
                    x = intruder.position[0]
                    y = intruder.position[1]
                    if dist(ownship, intruder) < ownship.radius + intruder.radius:
                        collide_intruder = True

```

```

        ownship.collision_intruder += 1
    elif x > width - 4 or x < 4 or y > height - 4 or y < 4:
        intruder.position = ownship.position
        reset_intruder(intruder)
# patikrinti ar tikslas pasiektas
    if dist(ownship, goal) < ownship.radius + goal.radius:
        simulate = False
# skrydžių simuliavimas
    car_group.update(deltat)
    pad_group.update()
    goal_group.update()
    pad_group.draw(screen)
    car_group.draw(screen)
    goal_group.draw(screen)
    time_display(time_step)
    collision_wall(ownship.collision_wall)
    collision_intruder(ownship.collision_intruder)
    pygame.display.flip()
if (dist(ownship, goal) < ownship.radius + goal.radius):
    print('budget: ', budget)
    print('search depth: ', search_depth)
    print('wall: ', ownship.collision_wall)
    print('intruder: ', ownship.collision_intruder)
    print('intruder speed: ', intruder.speed, " px/time_step")
    print('ownship speed: ', ownship.speed, " px/time_step")
    print("Total time algorithm: ",sum(current_time_list), 'milliseconds')
    df_own.to_csv('output_own.csv',index=False)
    df_intr.to_csv('output_intr.csv',index=False)
    df_final = pd.concat([df_own, df_intr])
    df_final.to_csv('output_final.csv',index=False)
    ownship.collision_wall = 0
    ownship.collision_intruder = 0
    ownship.collision_goal = 0

simulate = False

```