Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Saviorganizuojančio bepiločių orlaivių spiečiaus kontrolės algoritmo kūrimas

Baigiamasis magistro projektas

Arnas Pilvelis

Projekto autorius

Doc. Saulius Japertas

Vadovas

Kaunas, 2026

Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Saviorganizuojančio bepiločių orlaivių spiečiaus kontrolės algoritmo kūrimas

Baigiamasis magistro projektas

Aeronautikos inžinerija (6211EX024)

Arnas Pilvelis

Projekto autorius

Doc. Saulius Japertas

Vadovas

Prof. Laurencas Raslavičius

Recenzentas

Kaunas, 2026

Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Arnas Pilvelis

Saviorganizuojančio bepiločių orlaivių spiečiaus kontrolės algoritmo kūrimas

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Arnas Pilvelis

Patvirtinta elektroniniu būdu

Kaunas technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Magistro baigiamojo projekto užduotis

Išduota studentui (-ei) – Arnui Pilveliui

1. Projekto tema

Saviorganizuojančio bepiločių orlaivių spiečiaus kontrolės algoritmo kūrimas

(Lietuvių kalba)

Control Algorithm Development for a Self-organizing UAV Swarm

(Anglų kalba)

2. Projekto tikslas ir uždaviniai

Tikslas: Sukurti decentralizuotą spiečiaus valdymo algoritmą, užtikrinantį stabilų formavimąsi, nustatytos misijos atlikimą ir atsparumą trikdžiams, esant ribotai komunikacijai.
Uždaviniai:
1. Išnagrinėti dabartines saviorganizuojančio BO spiečiaus technologijas bei algoritmų charakteristikas;
2. Parinkti vieną matematinį modelį, kuris tinka saviorganizuojančio BO spiečiaus kontrolės algoritmo kūrimui;
3. Atlikti matematinio modelio analizę;
4. Pagal išnagrinėtą matematinį modelį sukurti BO spiečiaus algoritmą.
5. Atlikti saviorganizuojančio BO spiečiaus algoritmo testinę simuliaciją;
6. Įvertinti simuliacijos rezultatus ir parašyti išvadas;

3. Pagrindiniai reikalavimai ir sąlygos

Sklandus algoritmo veikimas pagal nustatytus plokščio BO spiečiaus parametrus.

4. Papildomi reikalavimai projektui, ataskaitai ir jos priedams

Pritaikymas įvairioms BO spiečiaus misijoms.

Arnas Pilvelis	2026-02-15
(Vardas, Pavardė)	*(Data)*
Saulius Japertas	2026-02-15
(Vardas, Pavardė)	*(Data)*

Krypties studijų programų vadovas	Artūras Keršys *(Vardas, Pavardė)*	2026-02-15 *(Data)*

Pilvelis, Arnas. Saviorganizuojančio bepiločių orlaivių spiečiaus kontrolės algoritmo kūrimas. Magistro baigiamasis projektas / vadovas doc. Saulius Japertas; Kauno technologijos universitetas, Mechanikos inžinerijos ir dizaino fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Aeronautikos inžinerija (E14), Inžinerijos mokslai.

Reikšminiai žodžiai: Bepiločių orlaivių spiečius, algoritmas, algoritmų kūrimas, spiečiaus valdymas, simuliacija.

Kaunas, 2026. 73 p.

Santrauka

Augant bepiločių orlaivių naudojimui įvairiose pramonės srityse, paieškos bei autonominio spiečiaus valdymo karinėse operacijose klausimas tampa vis aktualesnis. Vis dažniau norima užtikrinti, jog grupė dronų galėtų kuo savarankiškiau koordinuoti veiksmus ir vykdyti misijas be tiesioginio žmogaus įsikišimo. Šiame magistriniame darbe yra nagrinėjama bepiločių orlaivių spiečiaus kontrolės kūrimo problema. Teorinėje darbo dalyje yra apžvelgiama pagrindinės bepiločių orlaivių spiečiaus charakteristikos bei jų valdymo sistemų technologijos. Analizuojant „*Boids*", „*Vicsek"*, potencialių laukų, lyderio-sekėjų bei PSO, ACO ir SOMA optimizavimo metodų modelius buvo nustatyta, jog nė vienas iš jų nėra tinkamas siekiant užtikrinti norimą spiečiaus veikimą. „*Boids*" modelio stabilumas mažėja, kuomet spiečiai naudoja didelį kiekį agentų, „*Vicsek*" modelis dažnu atveju yra per paprastas, todėl jo veikimas remiasi į kitų modelių naudojimą. Potencialių laukų modelio kuriami lokalūs minimumai gali sustabdyti spiečiaus veikimą, kuomet spiečius randa pozicinį balansą, o lyderio-sekėjų modelis gali būti pažeistas, jeigu sugenda vienintelis spiečiaus lyderis. Dėl šių priežasčių atsiranda reikiamybė naudoti hibridinį valdymo sistemų modelį. Metodinėje dalyje išanalizuotas „*Fuzzy*" matematinis modelis, kuris yra taikomas pagrindiniam spiečiaus judėjimo valdymui. Modelio gebėjimas valdyti neapibrėžtas duomenų sistemas be reikiamybės tiksliai apibrėžti visos spiečiaus aplinkos puikiai tinka atlikti norimą algoritmą. Spiečiaus algoritmas yra sudaromas iš keturių skirtingų fazių, per kurias spiečius yra formuojamas į paieškai tinkamą formaciją, paleidžiamas plataus mąsto paieškai, suradus objektą subūriuojamas šalia naujai išrinkto lyderio bei vykdomas susidūrimas su rastu objektu. Fazių judėjimui naudojama trijų judėjimo valdymo komponenčių sistema, kurioje taikoma atskyrimo, lygiavimo bei kohentiškumo dedamosios. Šis algoritmas yra testuojamas atliekant dvi skirtingas simuliacijas „*MATLAB*" programoje, kuomet objektas yra arba randamas, arba ne. Simuliacijų metu yra parodomos sukurto algoritmo galimybės efektyviai ir stabiliai kontroliuoti spiečių glaudžiose bei plačiose jo formacijose ir atlikti greitus ir koordinuotus veiksmus artimose aplinkose. Taip pat yra apžvelgiami naudojamos sistemos trūkumai. Nors sukurtas algoritmas norimus spiečiaus tikslus įgyvendina, norint algoritmą naudoti realybėje reikia atsižvelgti į jo veikimą dvimatėje erdvėje ir įvertinti komunikacijos trikdžius bei realų duomenų perdavimą.

Pilvelis, Arnas. Control Algorithm Development for a Self-organizing UAV Swarm. Masters's Final Degree Project / supervisor assoc. prof. Saulius Japertas; Faculty of Mechanical Engineering and Design, Kaunas University of Technology.

Study field and area (study field group): Aeronautical Engineering (E14), Engineering Science.

Keywords: UAV Swarm, Algorithm, Algorithm Creation, Swarm Control, Simulation.

Kaunas, 2026. 73 pages.

Summary

With the growing use of unmanned aerial vehicles in various industrial fields, search and military operations, the issue of autonomous swarm control is becoming increasingly relevant. Increasingly, there is a desire to ensure that a group of drones can coordinate actions as independently as possible and carry out missions without direct human intervention. This master's thesis examines the problem of developing swarm control for unmanned aerial vehicles. The theoretical part of the work reviews the main characteristics of unmanned aerial vehicle swarms and the technologies of their control systems. Analyzing the models of "*Boids*", "*Vicsek*", potential fields, leader-follower, PSO, ACO and SOMA optimization methods, it was found that none of them are suitable for ensuring the desired performance of the swarm. The stability of the "*Boids*" model decreases when swarms use a large number of agents. "*Vicsek*" model is often too simple, so its operation relies on the use of other models. Local minimums created by the potential fields model can stop the swarm when the swarm finds positional balance, and the leader-follower model can break if the only leader of the swarm fails. For these reasons, the need to use a hybrid control system model arises. The methodological part analyzes the "Fuzzy" mathematical model, which is applied to the basic control of the swarm movement. The model's ability to handle uncertain data systems without the need to precisely define the entire swarm environment is well suited to perform the desired algorithm. The swarm algorithm is composed of four distinct phases, during which the swarm is formed into a formation which is designed for optimal search pattern, launched for a large-scale search, grouped around a newly elected leader once an object is found, and set for collision with the found object. A three-component motion control system is used for phase motion, which includes separation, alignment, and coherence components. This algorithm is tested by running two different simulations in the "*MATLAB*" program, where the object is either found or not. The simulations demonstrate the capabilities of the developed algorithm to effectively and stably control swarms in close and wide formations and to perform rapid and coordinated actions in close environments. The shortcomings of the used system are also reviewed. Although the developed algorithm implements the desired goals of the swarm, to use the algorithm in real life scenarios, it is necessary to evaluate its current operation in two-dimensional space and evaluate communication interference and real-world data transmission problems.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Įvadas

Bepiločiai orlaiviai yra ypatingai greitai auganti aviacijos sfera. Kuomet didžiajai visuomenės daliai bepiločių orlaivių prieinamumas tampa vis realesnis, proporcingai didėja ir BO technologijų išnaudojimas ir pritaikymas įvairiose rinkose bei kasdieniniame žmogaus gyvenime. Norint bepiločius orlaivius pritaikyti vis sudėtingesnėms užduotims, gali kilti didelių iššūkių, kai naudojamas vienas žmogaus kontroliuojamas dronas. Užduočių sudėtingumas akivaizdžiai mažėja, kuomet yra naudojamas didesnis kiekis tarpusavyje sujungtų ir bendraujančių bepiločių orlaivių. BO spiečiaus valdymas taip pat gali būti nelengva užduotis, kai jo suvaldymui reikia žmogaus indėlio bei veiksmų siunčiant juos realiu laiku, todėl saviorganizuojantys bepiločių orlaivių spiečiai yra aukštesnis BO užduočių vykdymo lygis.

Saviorganizuojančių BO spiečių užduotys gali būti labai įvairios, pavyzdžiui: topografinės informacijos kaupimas, karinė žvalgyba su spiečiaus saugojimusi tarpusavyje, taktiniai puolimai, vizualiniai pasirodymai danguje ir t.t. Norint šias užduotis atlikti tiksliai ir be nesklandumų, orlaiviams turi būti pateikti nurodymai bei taisyklės, kuriomis jie turi vadovautis. Daugumoje spiečių, ypač saviorganizuojančiuose spiečiuose, šie nurodymai pateikiami algoritmo formatu. Būtent tam tikrai užduočiai sukurtas algoritmas yra jungiamoji spiečiaus dalis, kuri leidžia jam bendrauti, tartis ir veikti kaip vienam organizmui. Spiečiaus algoritme dažniausia atsispindi šie parametrai - reagavimas į aplinką bei objektus, kurioje veikia BO spiečius, komunikacijos ir lyderiavimo grandys tarp atskirų bepiločių orlaivių, veiksmų ir sprendimų medis, kuris priklauso nuo sąlygų su kuriomis susiduria užduoties metu. Visi veiksniai ir galimybės turi būti tiksliai aprašytos pagal matematinius bei statistinius modelius, kurie sukuria efektyvias sąlygas ir padeda pasiekti optimalų spiečiaus veikimą.

Šio darbo tikslas yra sukurti decentralizuotą spiečiaus valdymo algoritmą, užtikrinantį stabilų formavimąsi, nustatytos misijos atlikimą ir atsparumą trikdžiams, esant ribotai komunikacijai.

Tikslui įgyvendinti buvo išsikelti tokie uždaviniai:
1. išnagrinėti dabartines saviorganizuojančio BO spiečiaus technologijas bei algoritmų charakteristikas;
2. parinkti vieną matematinį modelį, kuris tinka saviorganizuojančio BO spiečiaus kontrolės algoritmo kūrimui;
3. atlikti matematinio modelio analizę;
4. pagal išnagrinėtą matematinį modelį sukurti BO spiečiaus algoritmą;
5. atlikti saviorganizuojančio BO spiečiaus algoritmo testinę simuliaciją;
6. įvertinti simuliacijos rezultatus ir pateikti išvadas.

1. Teorinė dalis

1.1. Bepilotis orlaivis

Bepilotis orlaivis (BO), kitaip vadinamas dronu yra orlaivis, kuris neturi piloto ar kitaip aptarnauti galinčio žmogaus savo viduje. Jo valdymas vyksta per atstumą naudojant įvairų ryšį, kuriuo galima esant ant žemės perduoti komandas ore esančiam orlaiviui. Piloto nebuvimas orlaivyje yra itin svarbus, kadangi tai suteikia galimybes BO būti ir itin mažo dydžio. Vienas iš pagrindinių dronų kategorizavimo būdų yra būtent jų dydis. [1] Toliau bepiločiai orlaiviai gali būti smulkiau kategorizuojami pagal:

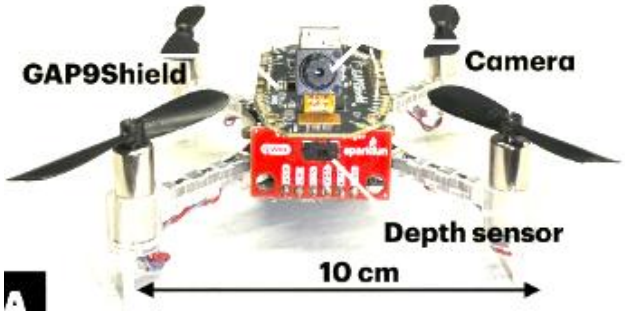
1. pakilimo būdą;
2. sparnų kiekį;
3. sparnų tipą;
4. skrydžio ilgį bei aukštį. [1]

Dronai gali būti smulkiai kategorizuoti. Dronų paskirstymą pagal jų dydį galima matyti pirmoje lentelėje.

1 lentelė. Bepiločių orlaivių kategorizavimas pagal dydį. [1]

Pavadinimas	**Bepiločio orlaivio dydis**
UAV (*unmaned air vehicle*)	< 35kg (Nepanešamas vieno žmogaus)
µUAV (*small UAV*)	> 35kg. (panešamas vieno žmogaus)
MAV (*Micro air vehicle*)	Sparnų ilgis 15-100 cm
NAV (*Nano air vehicle*)	Sparnų ilgis iki 15 cm
PAV (*Pico air vehicle*)	Sparnų ilgis iki 25 mm
SD *(Smart dust*)	Sparnų ilgis 1-3 mm

Žemiau galima pamatyti pavyzdį, kaip atrodo ir iš ko susidaro NAV dydžio bepilotis orlaivis: (Žr. 1 pav.)



1 pav. NAV dydžio bepiločio orlaivio pavyzdys [2]

1.2. Bepiločio orlaivio aerodinamika

Bet kokio tipo bepiločio orlaivio veikimui didelę įtaką daro jo aerodinaminės charakteristikos. Šiame poskyryje analizuojamas labiausiai paplitęs drono tipas – multi-rotorinis dronas, bei jo aerodinaminės savybės veikiant išoriniams veiksniams.

1.2.1. Multi-rotorinio bepiločio orlaivio aerodinaminės savybės

Kadangi multi-rotoriniai dronai geba vertikaliai sklandyti virš žemės paviršiaus, pastarieji turi didelį privalumą prieš fiksuoto sparno dronų modelius. Tačiau, dronui sklandant vertikaliai, šis susiduria su išoriniais aerodinaminiais reiškiniais, kurių poveikį bepiločiui orlaiviui apskaičiuoti yra sudėtinga. Oro srautai yra netiesūs bei nepastovūs, pvz. Multi-rotorinio drono propeleris gali susidurti su oro srautais, suformuotais jo paties arba kitų sparnų galūnių. Išorinių aerodinaminių reiškinių poveikį dronui apskaičiuoti yra dar sunkiau, kuomet atsiranda aerodinaminė sąveika tarp bepiločio orlaivio sparnų ir jo korpuso. [3]

1.3. Bepiločių orlaivių spiečius

Bepiločių orlaivių rinkai augant itin sparčiai, yra matoma ir jų galimybių bei misijų pokytis. Pavieniui valdomi dronai dažnai nesugeba įgyvendinti sudėtingesnių projektų arba tai reikalauja didelio antžeminių pilotų kiekio. Jų misijos įgyvendinimo greitį bei tikslumą lemia ribotas erdvės matymas bei gedimo rizikos. Taip pat, pavienio drono misijos mąstas proporcingai didina misijos įvykdymo laiką. Tokiais atvejais vis dažniau yra pasirenkama dronų spiečiaus galimybė, kuri leidžia vienam asmeniui valdyti žymiai didesnį kiekį dronų. Tokiu būdu yra eliminuojami misijos mąsto ribojamai. Taip pat pavienių orlaivių gedimas bepiločių orlaivių spiečiuje nebūtinai reiškia misijos nutraukimą ar vėlesnį jos kartojimą.[4]

Siekis suvaldyti didelį kiekį bepiločių orlaivių vienu metu iššaukia spiečiaus kontrolės problemas. Dažniausiai spiečiai yra sudaromi iš mažų dydžių bepiločių orlaivių. Tai lemia didesnę aplinkos įtaką:
1. spiečių formacijai;
2. nustatytos trajektorijos laikymuisi;
3. susidūrimams;
4. prognozuotam ir neprognozuotam spiečiaus manevravimui. [4]

1.3.1. Pagrindiniai bepiločio orlaivio skrydžio parametrai, reikalingi skrydžiui spiečiuje

Kuomet skrydžiui yra komplektuojamas bepilotis orlaivis, reikia atsižvelgti į daug skirtingų parametrų, tokių kaip masė, sparno tipas, korpuso dydis, variklio galingumas, baterijos talpa, apkrova, sukelta orlaiviui, trauka bei skrydžio laikas.

Viena iš svarbesnių drono savybių yra jo masė. Kuo svoris yra mažesnis, tuo efektyvesnis yra bepiločio orlaivio naudojimas, kadangi iškelti ir ore išlaikyti mažą svorį ilgą laiką yra gerokai lengviau bei yra sunaudojama mažiau energijos. Taip pat svorio apribojimai limituoti, kokio galingumo ir talpumo yra drono energijos šaltinis. [5]

Sparno tipas gali būti dvejopas - fiksuotas arba rotorinis. Fiksuoto sparno bepiločiai orlaiviai pasižymi itin didele ištverme, todėl gali padengti didelius plotus vieno skrydžio metu. Fiksuoto sparno bepiločiai orlaiviai taip pat yra paprastesnės konstrukcijos, turi didesnį pajėgumą nešti sunkesnį krovinį bei reikalauja minimalios techninės priežiūros. [6] Rotorinio sparno dronų privalumas yra jų gebėjimas tiek kilti, tiek leistis vertikaliai. Ši savybė leidžia atlikti operacijas ribotoje erdvėje,

nereikalaujant specialaus kilimo ar tūpimo ploto. Dėl prieš tai minėtų savybių rotorinio sparno dronai yra labai tinkami misijoms ankštose ar sudėtingose aplinkose. [6]

Bepiločio orlaivio korpuso matmenys lemia jo ergonomiką ir aerodinamines savybes. Didėjant drono gabaritams, mažėja jo manevringumas sudėtingose ar siaurose erdvėse: orlaivis gali netilpti tarp kliūčių, sunkiau įsilieti į numatytą spiečiaus formaciją ar nepasiekti suplanuoto atstumo pasirinktu maršrutu.

Variklio galia turi būti parinkta taip, kad atitiktų konkretaus bepiločio orlaivio poreikius. Per didelė galia lemia neproporcingai dideles energijos sąnaudas ir trumpesnį veikimo laiką, o per maža – gali net neleisti orlaiviui pakilti, nes variklis nesukuria pakankamos traukos.

Baterijos talpa tiesiogiai lemia drono veikimo trukmę, nes būtent ji aprūpina orlaivį visa reikalinga energija skrydžiui pasirinktu maršrutu. Renkantis bateriją būtina įvertinti planuojamą krovinį ir bendrą drono masę – nuo šių veiksnių priklauso energijos sąnaudų greitis ir faktinis baterijos išsikrovimo greitis. [7]

Apkrova yra įvairios droną veikiančios jėgos, kurios daro įtaką jo skrydžio svaybėms. Per didelė apkrova gali ne tik apsunktinti bepiločio orlaivio valdymą, bet ir padidinti sąnaudas. To pasekoje greičiau išsikrauna baterija, o skrydžio trukmė sumažėja. Taip pat, viršijus konstrukcines ribas gali pablogėti stabilumas bei suprastėti manevringumas.

Trauka apibūdina santykį tarp bendro drono svorio ir variklio generuojamos jėgos. Iš esmės, trauka yra rodiklis, kuris parodo, ar bepiločio orlaivio variklis sukuria pakankamą jėgą dronui pakilti, manevruoti ir stabiliai skristi. Kuo didensis traukos ir svorio santykis, tuo geresnės drono kilimo, pagreičio bei valdymo savybės.

Srydžio laikas yra esminis rodiklis siekiant užtikrinti, kad bepilotis orlaivis įveiktų numatytą misiją. Norint sėkmingai ją atlikti ar įveikti suplanuotą maršrutą, būtina turėti pakankamai laiko, kad misija būtų baigta be trikdžių.

1.3.2. Bepiločių orlaivių komunikavimo tarpusavyje sistemos

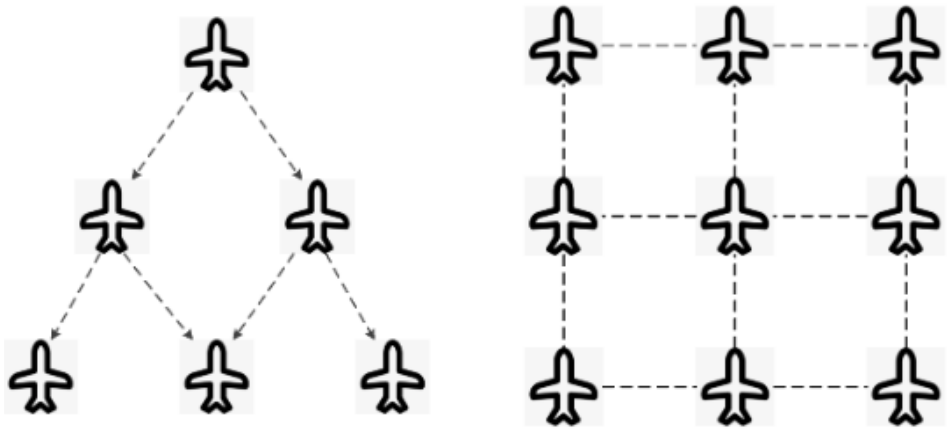
Komunikacija ir užduočių paskirstymas spiečiuje vyksta nuolat viso skrydžio metu. Šiam procesui užtikrinti taikomi du pagrindiniai metodai: *„FANET“ (Flying Ad-Hoc Network)* tipo tinklai ir infrastruktūrinė komunikacija.

Priešingai nei infrastruktūrinis būdas, *“FANET”* sistema nereikalauja antžeminio valdymo stoties. Visi spiečiaus dronai sudaro bendrą tinklą, kuriame jie tiesiogiai keičiasi informacija. Misijos metu kiekvienam orlaiviui yra priskiriami ryšio mazgai pagal maršrutizavimo algoritmą, tai leidžia užtikrinti tarpusavio komunikaciją realiu laiku. Tokiu būdu valdomas spiečius eliminuoja infrastruktūros keliamas rizikas, tačiau reikalauja, kad kiekvienas dronas turėtų tarpusavio ryšiui skirtą įrangą. Tai didina orlaivio masę, riboja ryšio nuotolį, nes siųstuvų galia priklauso nuo svorio apribojimų. Be to, tokio tipo ryšio sistemą sudėtinga įgyvendinti atliekant didelio tikslumo reikalaujančias užduotis [8].

Infrastruktūrinis metodas remiasi antžemine valdymo sistema, kuri tuo pačiu metu perduoda identišką informaciją visiems spiečiaus dronams. Šis būdas plačiai taikomas dėl galimybės siųsti komandas realiuoju laiku arba naudoti iš anksto parengtą komandų seką, tačiau pastaroji riboja spiečiaus autonomiškumą. Vis dėlto, kadangi visa sistema priklauso nuo vieno valdymo centro, pagrindinis trūkumas yra didelė rizika, jog valdymo sutrikimai ar centro gedimas gali sutrikdyti viso spiečiaus veikimą [8].

1.3.3. Bepiločių orlaivių spiečiaus formacijos ir jų kontrolė

Kiekvienas dronų spiečius turi savo formaciją, kuri yra priskirta pagal atitinkamą misiją bei valdymo sąlygas. Formacijos tipai priklauso nuo bepiločių orlaivių spiečiaus išsidėstymo bei pavienio drono ryšio su šalia skrendančiais spiečiaus dronais. Keli pagrindiniai tipai yra linijinis, koloninis, V formos, deimanto formos bei tinklelinis. Žemiau pateikta formacijų pavyzdžiai ir dronų tarpusavio ryšys. [4]



2 pav. BO spiečiaus formacijos pavyzdžiai [4]

2 paveiksle esančiuose pavyzdžiuose matoma, kad priklausomai nuo bepiločių orlaivių formacijos, tarpusavio ryšys gali drastiškai kisti. Trikampio formos formacijoje dronų ryšys veikia tik su viena linija žemiau esančiais orlaiviais. Vienoje linijoje esantys dronai nekomunikuoja tarpusavyje. Tinklelio formos formacijoje dronas funkcionuoja kaip figūros viršūnė, kuri siejasi su visais šalia esančiais bepiločiais orlaiviais.

Bepiločių orlaivių spiečiaus formacijų kontrolė yra skirstoma į tris pagrindinius valdymo būdus:
1. centralizuotą;
2. decentralizuotą;
3. hibridinį.

Centralizuotos spiečiaus kontrolės skiriamasis ženklas yra vienas kontrolės centras, pvz.: ant žemės esantis žmogus. Kontrolės centras surinkinėja visą informaciją ir pagal ją toliau priima sprendimus bei siunčia nurodymus atskiriems dronams. Šis kontrolės metodas pasižymi savo valdymo paprastumu, kadangi atskiri dronai patys neturi priimti sprendimų ir patys nereaguoja į aplinką. Iš vienos pusės, tai suteikia daugiau individualios kontrolės ir galimybę ant žemės esančiam žmogui reaguoti realiu laiku. [9] Kita vertus, šio valdymo tipo trūkumai atsiskleidžia norint valdymą implementuoti į didesnio spiečiaus agentų sistemą, kadangi nurodymų kiekis iš vieno kontrolės taško per trumpą laiko tarpą yra stipriai ribotas. Taip pat, vienas taškas gali susidurti su ryšio perdavimo problemomis, o tokiais atvejais spiečius pasidaro nevaldomas. [4]

Decentralizuota formacijų kontrolė remiasi į bepiločių orlaivių spiečiaus agentų tarpusavio bendravimą. Šiuo atveju antžeminis kontrolės taškas yra nereikalingas, kadangi spiečius skrenda pagal jiems iš anksto užduotą misiją bei algoritmą. Ši kontrolė panaikina spiečiaus mąsto problemą su kuria susiduriama centralizuotame valdyme bei leidžia dronams prisitaikyti tarpusavyje pagal aplink iškilusius sunkumus ar trikdžius. Šiuos decentralizuotos kontrolės privalumus šiek tiek atsveria

kiti faktoriai, pvz. spiečiaus dronų sudėtis ir dizainas. Decentralizuota kontrole valdomi dronai privalo turėti daugiau jutiklių bei daviklių, kurių pagalba jie galėtų tarpusavyje komunikuoti, tačiau tai augina agento dydį ir svorį. Prie šių problemų prisideda ir realaus testavimo sunkumai. [4]

Hibridinė spiečiaus formacijų kontrolė sujungia abiejų ankščiau minėtų valdymo būdo principus, todėl pasižymi ir bendru koordinavimo centru, ir aktyviu pačių agentų tarpusavio bendravimu. Šiuo atveju kontrolės centras siunčia nurodymus, o spiečiaus dronai, tarpusavyje besikeičiantys duomenimis, koreguoja savo judėjimą formacijoje ir priima savarankiškus sprendimus. Hibridinė kontrolė užtikrina misijos aiškumą ir kryptį, būdingą centralizuotai kontrolei, tuo pačiu išlaikydama decentralizuotam valdymui būdingą prisitaikymą prie aplinkos ir atsparumą vieno spiečiaus drono gedimui. Tačiau, verta paminėti, kad hibridinis valdymas reikalauja sudėtingesnės komunikacijos tarp agentų, kadangi šie turi nuolat keistis informacija, tuo tarpu kontrolės centrui tenka apdoroti kur kas didesnį duomenų srautą. [4]

1.4. Algoritmas

Algoritmas gali būti suprantamas kaip nuoseklių veiksmų rinkinys, kurio tikslas – pasiekti aiškiai apibrėžtą rezultatą. Tai struktūruota procedūra, leidžianti automatizuoti tam tikrus procesus ir užtikrinti, kad jie būtų atliekami sistemiškai. Kiekvienas algoritmas remiasi pradine užduotimi bei jai įvykdyti reikalingais duomenimis ar veiksmais. Kadangi veiksmų seka yra determinuota, pateikus tuos pačius įvesties duomenis visuomet gaunamas identiškas rezultatas.

Algoritmams būdingi keli esminiai požymiai:
1. veiksmai turi būti nedviprasmiški ir tiksliai apibrėžti;
2. pradiniai duomenys turi būti aiškiai nurodyti;
3. rezultatas turi būti konkretus ir patikrinamas;
4. veiksmų seka negali būti begalinė;
5. algoritmą turi būti įmanoma realizuoti esamomis priemonėmis;
6. jo veikimas nepriklauso nuo pateikimo kalbos ar formos.

Jeigu visos šios sąlygos yra įvykdytos, algoritmas laikomas korektišku ir tinkamu naudoti. [10]

Šiuolaikinėse technologinėse sistemose algoritmai taikomi beveik visur, kur reikia greito ir tiksliai apibrėžto veikimo, kurio žmogus mechaniškai atlikti nepajėgtų. Efektyviausias būdas tokius procesus įgyvendinti – kompiuteriui pateikti reikiamus duomenis ir aiškiai suformuluotas užduotis. Pavyzdžiui, Laitho Abualigaho ir Ali Diabato straipsnyje *„Advances in Sine Cosine Algorithm: A comprehensive survey“* nagrinėja sinuso–kosinuso algoritmą. Tai populiacijos pagrindu veikiantis optimizavimo metodas, kurio veikimas grindžiamas sinusinių ir kosinusinių funkcijų taikymu. Autoriai pažymi, kad optimizavimo procesas pradedamas nuo atsitiktinai sugeneruotų pradinių sprendinių (populiacijos), kurie vėliau iteratyviai vertinami ir tobulinami. [11]

1.5. Bepiločių orlaivių saviorganizacijos modeliai

Saviorganizacija dronų spiečiuose grindžiama prielaida, jog spiečiaus elgsena gali būti valdoma užduodant reikiamas lokalias taisykles ir tam tikrą informacijos keitimosi apimtį. Saviorganizuojantys spiečiaus modeliai yra lengvai prisitaikantys prie aplinkos ir lengvai plečiami nes kiekvienas agentas sprendimai priklauso nuo artimos jo aplinkos. Šiame poskyryje aptariami pagrindiniai saviorganizacijos principai ir klasikiniai modeliai, kurie tapo šiuolaikinių bepiločių orlaivių spiečių valdymo algoritmų pagrindu:

1. *„Boids“* modelis – trys taisyklės: atstūmimas, lygiavimasis ir trauka;
2. *„Vicsek“* modelis – agentai lygiuojasi pagal kaimynų kryptį;
3. potencialų laukų metodai – jėgų balansas tarp traukos ir atstūmimo;
4. optimizaciniai metodai – PSO, ACO, SOMA.

1.5.1. *„Boids“* modelis

Vienas pirmųjų ir vienas plačiausiai naudojamų saviorganizacijos modelių yra C. W. Reynoldso 1987 metais pristatytas *„Boids“* modelis, kuris stengiasi imituoti paukščių spiečiaus elgseną. Modelio judėnimas ir būriavimasis yra grindžiamas trimis lokaliomis taisyklėmis:

1. atstūmimas - spiečiaus agentas vengia per didelio priartėjimo prie kaimynų atstumiant kitus aplinkinius agentus, kad būtų išvengta susidūrimų;
2. lygiavimasis - agentas koreguoja savo judėjimo kryptį atsižvelgiant į šalia skrendančių kaimynų vidutinę kryptį;
3. trauka - agentas juda link kaimynų koordinačių centro, siekdamas palaikyti viso spiečiaus susibūriavimą. [12]

Šių trijų taisyklių kombinacija įgalina spiečių elgtis koordinuotai be pagrindinio centrinio valdymo. Bepiločių orlaivių spiečiaus kontekste *Boids* modelis yra naudojamas šiems tikslams įvykdyti: formacijos palaikymui, kolektyviniam judėjimui bei susidūrimų vengimui. Šis modelis yra ganėtinai paprastas, kas leidžia modeliui greitai veikti ir keisti spiečiaus dinamiką realių laiku, tačiau užduotims sunkėjant modelio tikslumas tampa prastesnis ypač planuojant tikslesnes judėjimo trajektorijas ar prisiderinime prie aplinkos.[13]

1.5.2. *„Vicsek“* modelis

Tamáso Vicseko *„Vicsek“* modelis buvo sukurtas 1995 metais ir yra vienas iš fundamentaliausių saviorganizacijos tyrimų rezultatų. Jo tikslas yra nagrinėti spiečiaus bendrą judėjimą minimalistinėse sąlygose. Šiame modelyje agentams yra priskiriamas vienas pastovus greitis, o judėjimo krypties skaičiavimuose atsižvelgiama į kaimynų kryptis ir pridedant triukšmo komponentę: (Žr. 1 formulę) [14]

$$\theta_i(t+1) = \langle \theta_j(t) \rangle r + \eta \quad (1)$$

čia $\langle \theta_j(t) \rangle r$ – kaimynų krypties vidurkis spinduliu r;

η – atsitiktinis triukšmas.

„Vicsek“ modelis teigia, jog į sistemą įtraukiant triukšmą galima netikėtai pereiti į tvarkingą spiečiaus formaciją, kurioje agentai juda į vieną kryptį. Bepiločių orlaivių spiečiuose *„Vicsek“* modelis tinkamas analizuojant kolektyvinio judėjimo stabilumą, triukšmo bei paklaidų poveikį bei kuriant decentralizuotus spiečiaus lygiavimo algoritmus. Tačiau *„Vicsek“* modelio trūkumas kitaip nei Boids modelis neapima agentų atstūmimo ar traukos komponentų todėl *„Vicsek“* modelis dažniausiai naudojamas kartu su kitais saviorganizacijos metodais. [15]

1.5.3. Potencialių laukų metodai

Potencialų laukų metodų veikimas yra paremtas tuo, jog kiekvienas agentas turi savo jėgų lauką, kurie keičia traukos bei atstūmimo komponentus ir jų formavimą spiečiuje. Veikiant bendrai traukos ir atstūmimo jėgoms bepilotis orlaivis juda nustatyto taikinio link, o potencialų laukų superpozicija sukuriama jėga nulemia bepiločio orlaivio judėjimo kryptį. [16] Šiame modelyje dažniausiai yra naudojamos šios jėgos:

1. atstūmimo jėga nuo per arti esančių agentų ar kliūčių;
2. traukos jėga link formacijos centro, lyderio ar užduoties taško;
3. papildomos jėgos, kurios atsiranda dėl papildomų nustatytų veiksnių kaip pvz., objektų vengimas, kelio sekimas, tikslų siekimas.

Potencialų laukai bepiločių orlaivių spiečiuose įgalina šiuos veiksmus:

1. generuoti sklandžias trajektorijas;
2. užtikrinti susidūrimų vengimą;
3. palaikyti formacijas;
4. integruoti aplinkos informaciją (pvz., kliūčių žemėlapius).

Pagrindinis modelio trūkumas – galimybė įstrigti lokaliuose minimumuose, kadangi agentas visada juda link mažiausios energijos taško, tačiau aplinkui agentą gali būti daugiau minimumų, kurie nėra optimali pozicija jam. Dėl šių priežasčių potencialių laukų metodai praktinėse sistemose dažnai derinami su optimizaciniais ar euristiniais metodais. [17]

1.5.4. Optimizaciniai metodai (PSO, ACO, SOMA)

Bepiločių orlaivių spiečių optimizaciniai metodai yra skirti ne tik palaikyti užduotas spiečiaus formacijas bet ir atlikinėti sunkesnes užduotis kaip tinkamo maršruto parinkimai, skirtingų spiečiaus tikslų paskirstymas atskiriems agentams ar agentų energijos eikvojimo optimizacija. Tokioms užduotims dažniausiai naudojami modeliai yra:

1. PSO (angl. Particle Swarm Optimization) – Jo elgsena yra pritaikyta pagal paukščių būrio judėjimą. Spiečiaus agentai keičia savo trajektoriją pagal geriausią tame žingsnyje atliktą individualų bei globalų sprendimą. Šis metodas spiečiuose naudojamas optimizuoti trajektorijas ara formacijų generavimui. [18]
2. ACO (angl. Ant Colony Optimization) – šio modelio elgsena paremta skruzdžių lizdo veikimo principais, kurios informacijai pranešti naudoja feromonus. Šis metodas dažniausiai naudojamas optimalių kelių paieškai ar teritorijos skautavimo misijai. [19]
3. SOMA (angl. Self-Organizing Migrating Algorithm) – Šis metodas paremtas populiacijos migracija. Jo naudojimas daugialaipsniškuose optimizacijos uždaviniuose parodo metodo efektyvumą. Bepiločių orlaivių spiečiuose dažnai naudojamas formacijų optimizavimui arba realaus laiko parametrų kaitai. [20]

Optimizaciniai metodų privalumas yra jų greitas prisitaikymas prie sudėtingų aplinkų, tačiau šie privalumai kenčia nuo pagrindinių metodų trūkumų t.y didelės duomenų apdorojimo kainos ir didelių išteklių reikalavimo. [21]

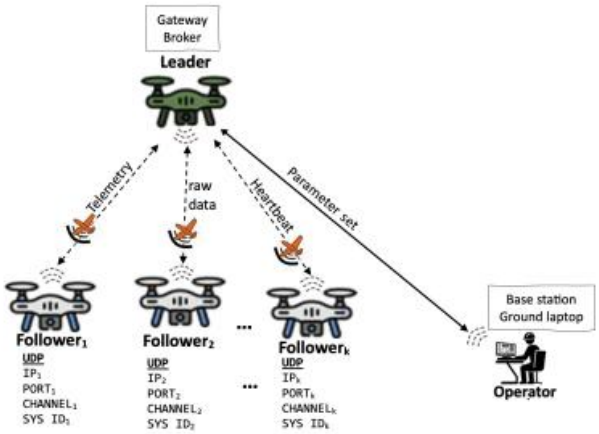
1.6. Bepiločių orlaivių spiečiaus veikimas pagal lyderio-sekėjų modelį

Bepiločių orlaivių suvaldymas gali sukelti nenorimų trikdžių. Tam kad spiečiuje atsirastų didesnė apibrėžtis, aplink kurią būtų galima modeliuoti tolimesnį spiečiaus judėjimą, reikia nustatyti spiečiaus atskaitos taškus. Lyderių ir sekėjų spiečiaus modelis yra vienas iš populiariausių valdymo metodų, kuris dinamiškai ir decentralizuotai leidžia nustatyti minėtus taškus. Spiečiuje yra sudaroma lyderio arba lyderių bei jų sekėjų hierarchija. Dronai lyderiai yra spiečiaus atskaitos taškai, kurie iš žemės kontrolės gauna pradinius misijos nurodymus. Pagal tai lyderiai gali:

1. nustatyti likusių sekėjų roles;
2. perduoti antžeminės kontrolės nurodymus sekėjams;
3. nustatyti spiečiaus agentų tarpusavio atstumus ir juos palaikyti;
4. užduoti norimą spiečiaus judėjimo greitį bei kryptį;
5. užduoti sekėjų išsidėstymo būdą. [22]

Sekėjų rolės nustatymas padeda sukurti optimalų spiečiaus judėjimą bei mąstymą nustatytai užduočiai. Spiečiaus agentų pakėlimas į orą ir pirminių duomenų perdavimas turi būti įgyvendinamas rankiniu būdu žmogaus esančio ant žemės, tačiau duomenys yra perduodami tik lyderiui. Spiečiaus atitinkamo atstumo palaikymas yra nustatomas lyderio, tačiau atstumo laikymasis vyksta dalinantis duomenimis šalia esantiems agentams. [23] Lyderio–sekėjų modelyje sekėjų greitis ir kryptis paprastai formuojami kaip kelių dalių formulė. Pirma dedamoji yra tiesiogiai sekant lyderio būseną, antra dedamoji yra gaunama iš lokalių kaimynų sinchronizacijos, o trečioji iš avarinio atstūmimo atstumui kritiškai sumažėjus. tokiu būdu sekėjo pageidaujamas greitis ir vektorius gaunamas kaip suma, kurią galima užrašyti kontrolės dėsniu.

3 paveiksle galima matyti schemą, kuria galima apibrėžti komunikaciją tarp antžeminės stotelės, spiečiaus lyderio bei jo sekėjų:



3 pav. Antžeminės stotelės, spiečiaus lyderio bei sekėjų komunikacijos schema [22]

Misijos metu spiečiaus lyderis taip pat gali pasikeisti, jeigu tai padeda misijos tikslui.

Šio modelio spiečiaus privalumai yra spiečiaus orientacijos stabilumas. Lyderio buvimas suteikia spiečiui pakankamai informacijos, kad spiečius galėtų išlaikyti nurodytą formaciją. Tai sumažina spiečiaus priklausomybę nuo aplinkinės informacijos rinkimo bei analizavimo, o tai nulemia mažesnę atskiro agento skaičiavimo apkrovą. Šios sąlygos sukuria geresnes galimybes kontroliuoti visą sistemą realiu laiku. Tačiau šis modelis turi ir trūkumų. Viena iš pagrindinių rizikų yra lyderio

gedimas ar ryšio praradimas su juo. Tai gali nulemti spiečiaus sugriuvimą, jeigu nėra atsarginio lyderio išrinkimo sistemos ar didesnio pradinio lyderių kiekio. [24] Taip pat lyderio turėjimas mažina spiečiaus savikontrolės lygį, kadangi priklausomybė nuo lyderio apriboja atskirų agentų prisitaikymą prie netikėtų situacijų.

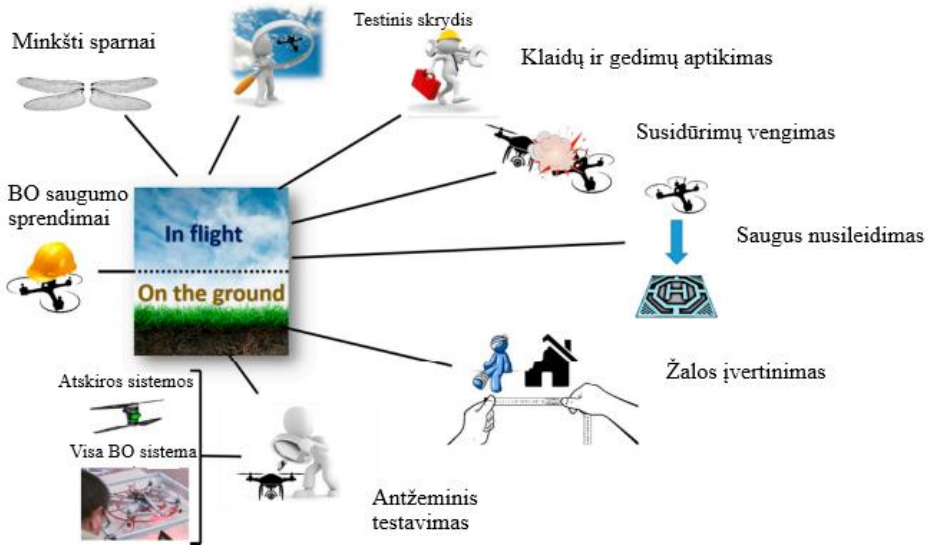
1.7. Bepiločių orlaivių ir jų spiečių saugos aspektai

Bepiločių orlaivių bei jų spiečių naudojimas įvairiose pramonės srityse, rekreacinėse veiklose, gelbėjimo ar karinėse operacijose suteikia daug naujų galimybių optimaliai vykdyti norimus tikslus. Tačiau kaip ir su visomis tobulėjančiomis sritimis lygiagrečiai turi tobulėti ir naudojimo saugumas bei naudojimo rizikos vertinimas. Nekontroliuojami bepiločiai orlaiviai arba techniškai neprižiūrėti dronai gali sukelti avarijas su skrendančiais pilotuojamais orlaiviais, netikėtai nukristi su apgadinti turtą ar sukelti žalą ant žemės esantiems žmonėms. [25] Norint užtikrinti saugumą reikia atlikti tam tikras antžemines procedūras: (Žr. 4 Pav.)

1. antžeminis bepiločio orlaivio bei jo mažesnių sistemų testavimas;
2. žalos, kurią gali padaryti to tipo dronas įvertinimas.

Kuomet dronas yra ore, reikia:

1. atlikti pirminį testinį skrydį;
2. užtikrinti saugų orlaivio nusileidimą;
3. vengti dronų bei kitų orlaivių susidūrimų.



4 pav. Bepiločio orlaivio saugumo bei rizikų galimybių sudedamosios dalys [25]

Dronų spiečiuose dažniausios avarijos pasitaiko dėl susidūrimų tarpusavyje. Pirminės susidūrimo rizikos atsiranda agentams tik įsijungus ir pradedant kilimo fazę jeigu kilimas vyksta tuo pačiu metu [26]. Šiais atvejais reikia integruoti susidūrimų vengimo sistemą, kuri veikia autonomiškai ir gali reguliuoti atstumą tarp agentų be žmogaus įsikišimo. Tokios pat sistemos padeda išvengti susidūrimų ir skrydžio metu ypač kai agentų greitis yra didelis o atstumai maži.

Šiuo metu auganti sfera dronų gamyboje, kuri yra susijus su naudijmo saugumu, yra minkštų, neaštrių ir nesunkių sparnų ar kitų detalių gamyba. Šio tipo technologijos kuria detales, kurios avarijų ar susidūrimo su žmogumi metu padarytų kuo mažesnę žalą. Žala dažniausiai priklauso nuo drono

greičio ir jėgos, kuria yra susiduriama su objektu. Greičio faktorių suvaldyti pavyksta nevisada, tačiau jeigu bepilotis orlaivis bus lengvesnis, jo susidūrimo jėga bus daug mažesnė. [27]

1.8. Dirbtinio intelekto naudojimo metodai bepiločių orlaivių spiečiaus valdyme

Šiuolaikiniuose bepiločių orlaivių spiečiuose dažniausiai yra reikalaujama, kad agentų kontrolė ir jų misijos vykdymas vyktų autonomiškai arba su minimalia žmogaus pagalba. Šiais atvejais labai didelę pagalbą gali suteikti dirbtinis intelektas. Vienas iš dirbtinio intelekto naudojimo bepiločių orlaivių spiečiui valdyti privalumas yra spiečiaus agento gebėjimas veikti bei rinkti norimus duomenis atskirai nuo spiečiaus, kas leidžia į turimą situaciją pažvelgti iš skirtingų pozicijų. [28] Dirbtinis intelektas, kuris yra pritaikytas dirbti su bepiločių orlaivių spiečiais gali prižiūrėti:

1. spiečiaus formaciją bei judėjimą;
2. užduočių paskirstymą bei jų optimizavimą;
3. vengti dronų bei kitų orlaivių susidūrimų;
4. spiečiaus komunikaciją, bei informacijos dalinimąsi tarpusavyje;
5. sprendimų priėmimą ir spiečiaus prisitaikymą prie netikėtų sąlygų realiu laiku. [29]

Spiečiaus formavimo ir judėjimo srityje naudojami saviorganizacijos, susispietimo ir potencialių laukų dirbtinio intelekto algoritmai, leidžiantys dronams autonomiškai organizuotis. Šie algoritmai įgalina kiekvieną droną stebėti kaimyninių dronų pozicijas bei greičius ir dinamiškai keisti savo elgesį, išlaikant pageidaujamą formaciją, kuomet keičiasi aplinkos sąlygos.

Užduočių paskirstymo sritis yra puikiai tinkanti išnaudoti dirbtinio intelekto privalumus. Jo metodai yra genetiniai algoritmai, skruzdžių kolonijų optimizacija bei sutvirtinto mokymosi modelis, leidžiantya efektyviai paskirstyti užduotis tarp atskirų dronų, atsižvelgiant į jų galimybes, energijos sąnaudas ir artumą prie tikslo. Taip užtikrinamas tolygus apkrovos paskirstymas ir maksimalus visos sistemos efektyvumas.

Navigacijos ir trajektorijų planavimo srityje dirbtinis intelektas taiko dirbtinių potencialų laukų ar tikimybinių kelių žemėlapių metodus, kurie užtikrina saugų judėjimą sudėtingoje aplinkoje, susidūrimų vengimą ir taisyklingą navigaciją. Dronai tarpusavyje dalijasi informacija apie kliūtis ir aplinkos ypatumus. Tokiu būdu spiečius kolektyviai planuoja sau efektyviausius judėjimo maršrutus.

Komunikacijos ir informacijos dalijimosi srityje dirbtiniu intelektu grindžiami susitarimo algoritmai, spiečiaus intelekto protokolai ir belaidžio ryšio technologijos. Tai užtikrina patikimą duomenų apsikeitimą tarp dronų realiuoju laiku, kas yra būtina sąlyga koordinuotiems veiksmams ir kolektyviniam sprendimų priėmimui. Šie patobulinimai labai stipriai padeda tokioms misijoms, kaip paieškos ir gelbėjimo misijos, arba tiksli agrokultūros priežiūra. [30]

Sprendimų priėmimo srityje dirbtinis intelektas gali analizuoti jutiklių renkamus duomenis naudodamas decentralizuotus sprendimų modelius, mašininio mokymosi algoritmus ar žaidimų teorijos algoritmą. Tai leidžia paprasčiau priimti sprendimus pagal iš anksto nustatytus tikslus, prisitaikant prie kintančių sąlygų. 5 paveiksle galima matyti, kaip spiečiaus autonominiai sprendimai gali būti naudojami sudėtingesnėse aplinkos sąlygose.

5 pav. Bepiločių orlaivių spiečius naudojamas paieškos ir gelbėjimo misijoje [31]

Apibendrinus visų algoritmų modelių analizę galima teigti, jog kiekvienas modelis turi savų privalumų bei trūkumų, tačiau nei vienas modelis vienu metu neužtikrina trijų spiečiaus esminių savybių – spiečiaus stabilumo, modelio efektyvumo auginant spiečiaus agentų kiekį bei trikdžių atsparumo. „*Boids“* modelis yra paprastas, tačiau dideliuose spiečiuose jo veikimas tampa neefektyvus. *„Vicsek“* modelio veikimas realiose sąlygose tinkamas būna tik tada, kai naudojama kitas valdymo modelis. Potencialių laukų modelio užstrigimas lokaliuose minimumuose trikdytų spiečiaus stabilumui. Lyderio-sekėjų modelį naudojantis spiečius gali staigiai prarasti kontrolę, jeigu pagrindinis komunikacijos taškas, t.y spiečiaus lyderis, sugenda ar praranda ryšį su kitais agentais.

2. Metodinė dalis

2.1. Bepiločių orlaivių spiečiaus efektyvumas karinėje aplinkoje

Remiantis šių dienų aktualijomis, BO bei jų spiečiai yra vienas iš pagrindinių strateginių temų karo srityje. Dronai su įtaisytomis vaizdo kameromis gali filmuoti ir rinkti naudingą žvalgybos informaciją. Kiti bepiločiai orlaiviai turi sprogstamuosius užtaisus, kurių misija – neutralizuoti norimus priešų objektus. Šiai karo sričiai augant itin sparčiai yra kuriamos ir apsauginės priemonės, padedančios apsisaugoti nuo priešų atakų, todėl pavieniai dronai po truputį tampa neefektyvūs. Dabartinėmis žiniomis, norint sunaikinti bet kokią karo mašiną, prireikia ne mažiau nei trijų dronų su sprogstamaisiais užtaisais. Tokiais atvejais bepiločių orlaivių spiečiai turi didelį pranašumą.[32]

2.2. Bepiločių orlaivių spiečiaus algoritmo misijos identifikavimas

Šio tyrimo objektas yra decentralizuotas savikontroliuojantis bepiločių orlaivių spiečius. Jo agentas naudoja vietinę komunikaciją bei "*Fuzzy*" matematinio modelio logiką. Šio algoritmo kūrimas prasideda nuo pagrindinės misijos parametrų, uždavinių bei tikslų apibrėžimo. Kuriamo algoritmo tikslas yra ieškoti ir surasti priešų objektą bei jį neutralizuoti. Šiam tikslui pasiekti yra užsibrėžiamos kritinės sąlygos, pagal kurias algoritmas turės funkcionuoti:

1. bepiločių orlaivių spiečius sudaromas iš 21 agento;
2. bepiločių orlaivių skrydis vyksta vienoje plokštumoje;
3. bepiločių orlaivių aptikimo sistema veikia 150 metrų spinduliu;
4. bepiločių orlaivių efektyvus skrydžio atstumas – 20 km;
5. bepiločiai orlaiviai nesusiduria nei su išorinėmis kliūtimis, nei tarpusavyje.

Taip pat algoritmai, kurių misija yra pakankamai komplikuota, gali turėti skirtingas elgsenos fazes. Kuriamo algoritmo nustatytos fazės yra:

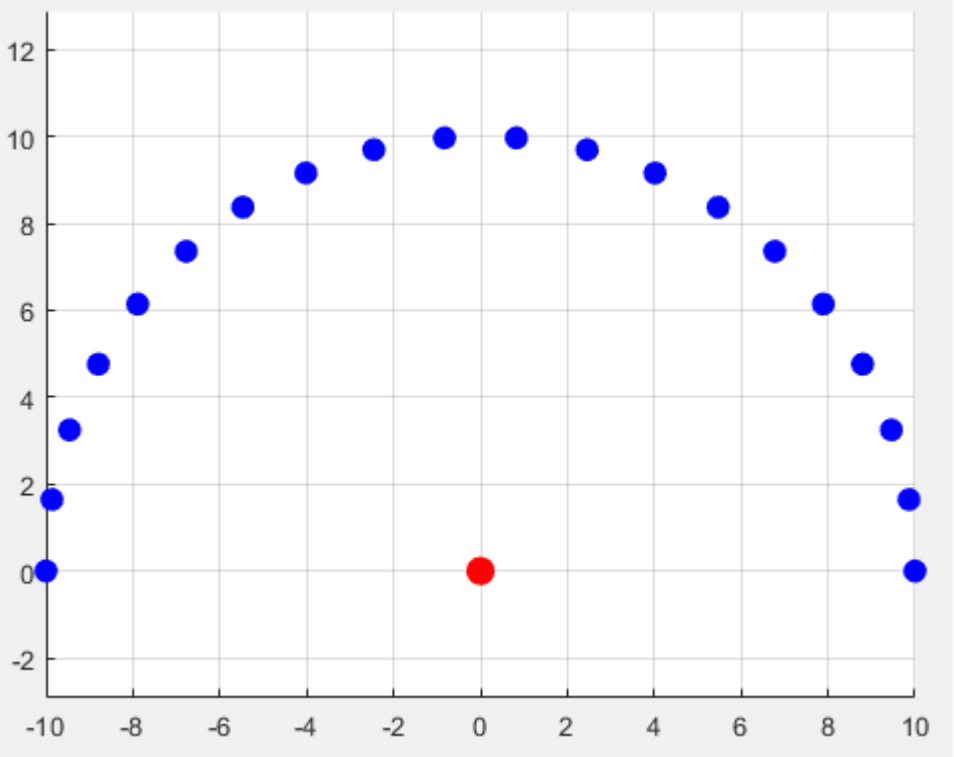
1. pradinės formacijos sudarymas;
2. formacijos išsisklaidymas ir paieška;
3. lyderio išsirinkimas, kuomet surandamas objektas;
4. spiečiaus susibūrimas ir sekimas;
5. spiečiaus susidūrimas su objektu.

2.2.1. Algoritmo kritinių sąlygų analizė

21 Agentas yra pasirenkamas dėl pradinio spiečiaus išsidėstymo. 1 Dronas yra pasirenkamas kaip pradinis lyderis. Jo tikslas yra suformuoti likusius agentus aplink save ir paruošti jų trajektoriją toliau vyksiančiai paieškai. 20 likusių agentų paieškos metu gali padengti pakankamai didelį paieškos plotą. Kuomet priešo objektas yra surandamas, agentų kiekis padidina objekto neutralizavimo tikimybę. Vienoje plokštumoje judantys dronai algoritmą padaro mažiau kompleksiniu, kadangi nereikia atsižvelgti į trečią dimensiją ir judėjimą joje. Bepiločių orlaivių objekto aptikimo sistemos dydis parenkamas 150 metrų, kadangi tai sukuria pakankamai didelį aptikimo plotą nesumažinant objekto aptikimo galimybių. Aptikimo sistema remiasi vaizdo kamerų renkama informacija, todėl apimant didesnį plotą, dėl vaizdo raiškos atsirastų galimybė neaptikti norimų objektų. Kadangi spiečiaus agentas turi sugebėti atlikti ir paieškos, susibūrimo ir sekimo, bei susidūrimo su objektu fazes, 20 km efektyvus skrydžio atstumas užtikrina pilną misijos įvykdymą. BO nesusidūrimas tarpusavyje bei su išorinėmis kliūtimis yra būtinas norint efektyviai vykdyti kiekvieną algoritmo fazę.

2.2.2. Algoritmo fazių analizė

Aukščiau įvardintos algoritmo fazės apibūdina misijos scenarijų. Pradinės formacijos sudarymas prasideda nuo lyderio ir likusių agentų pakilimo į oro erdvę. Lyderis yra naudojamas kaip apskritimo vidurys, aplink kurį rikiuojasi likę agentai.

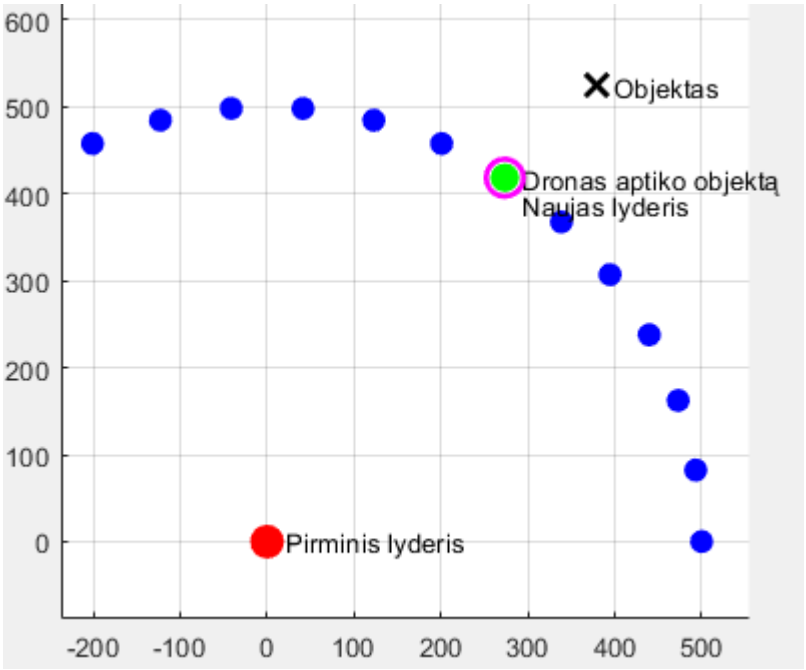


6 pav. Pavyzdinis pradinis spiečiaus išsirikiavimas

6 paveiksle matoma, kaip aplink lyderį yra suformuojama pusrutulio formacija. Ši formacija leidžia agentams judėti spindulio trajektorija link neištirtos teritorijos ir plokštumą pasidalinti laipsniškai. Šių agentų trajektorijos kampas skiriasi 9 laipsniais, o tai leidžia padengti didelį paieškos plotą. Taip pat dronai išsirikiuoja tokiu atstumu, kuris nekeltų rizikos tarpusavio susidūrimams.

Paieškos fazė paleidžia dronus judėti jiems priskirta trajektorija. Nuo pradinės formacijos jų skrydis tęsiasi tol, kol yra surandamas objektas arba objekto aptikimo sistemos spindulys nesusikerta su šalia skrendančio drono paieškos spinduliu. Šios fazės metu agentų tarpusavio bendravimas yra atstumas tarp dviejų šalia paleistų dronų.

Jeigu paieška įgyvendinama sėkmingai ir yra surandamas norimas objektas, objektą suradęs dronas tampa spiečiaus lyderiu. Pagal jo poziciją visas likęs spiečius pradeda savo judėjimą taikinio link. 8 paveiksle galima matyti momentą, kuomet atsitiktinis dronas suranda objektą ir yra priskiriamas nauju spiečiaus lyderiu:



7 pav. Objekto aptikimas ir naujo lyderio išrinkimas

7 paveiksle matoma, kad objektą radęs dronas yra priskiriamas nauju spiečiaus lyderiu, likę spiečiaus agentai baigia paieškos užduotį ir gauna naujus nurodymus skristi pas naują lyderį ir būriuotis aplink jį. Tuo pačiu metu lyderis nepaleidžia objekto iš savo aptikimo zonos. Jeigu objektas yra judantis, spiečiaus lyderis jį seka palaikydamas aptikimo atstumą. Šioje fazėje dronai turi atsižvelgti į atstumą tarp kiekvieno agento, kadangi visi agentai renkasi į ta pačią poziciją. Dronams susibūriavus, algoritmas pasikeičia į galutinę fazę.

BO Spiečius vienu metu pradeda judėti į objekto poziciją su tikslu susidurti su objektu. Kadangi tikslas susidurti yra tik su objektu, agentams taip pat reikia palaikyti bent minimalų tarpusavio atstumą. Sklandų susidūrimą su objektu taip pat galima įgyvendinti atskiriant agentų startavimo momentą trumpu laiko tarpu. Tokiu atveju mažėja galimybė dronams susidurti tarpusavyje.

Misija taip pat gali nepavykti, kuomet paieškos fazės metu nėra randamas norimas objektas. Tokiu atveju spiečiaus agentams yra liepiama grįžti pas pradinį lyderį. Misijos eiga taip pat gali kisti jeigu tuometinis spiečiaus lyderis sugenda, nukrenta arba yra prarandamas ryšys tarp jo ir kitų spiečiaus agentų. Tokiu atveju spiečius turi grįžti į paieškos fazę ir toliau ieškoti norimų objektų. Objektų paieškai tai nekelia papildomų iššūkių, kadangi atskiro agento paieškos spindulys didžiąją misijos laiko dalį kertasi su šalia skrendančiu agento paieškos spinduliu.

2.3. Bepiločių orlaivių spiečiaus saviorganizacijos sprendimai

Decentralizuotos kontrolės bepiločių orlaivių spiečius veikiantis plokštumoje informaciją apie judėjimą bei kitų agentų poziciją gauna tik iš lokalios sistemos. Agento pozicija yra apibrėžiama koordinačių sistemoje. Šis apibrėžimas yra paprastesnis, kadangi visas judėjimas vyksta plokštumoje. Taisyklės, kurios reikalingos agentams suvaldyti, yra paremtos matematinių modelių veikimu. Šiam algoritmui yra pasirenkamas "*Fuzzy"* matematinis modelis.

2.3.1. Bepiločių orlaivių spiečiaus matematinis modelis bei jo valdymo struktūra

Norint tiksliai apibrėžti bepiločių orlaivių spiečiaus saviorganizacijos algoritmą, būtina matematiškai aprašyti kiekvieno agento būseną bei jo sąveiką su aplinka. Kiekvieno spiečiaus agento koordinatės bei jo greitis atitinkamai aprašomas kaip $x_i(t)$ ir $v_i(t)$, kur t – laiko momentas. Tarkime, kad spiečių sudaro N bepiločių orlaivių, kurių kiekvieno būsena apibrėžiama padėties ir greičio vektoriais:

$$x_i(t) \in A^2, \quad v_i(t) \in A^2, \quad i = 1, \ldots, N \tag{2}$$

čia A – padėtis dvimatėje plokštumoje. Konkretaus UAV dinamika modeliuojama laiku pagal šias lygtis:

$$x_i(t+1) = x_i(t) + v_i(t)\Delta t \tag{3}$$

$$v_i(t+1) = v_i(t) + u_i(t)\Delta t \tag{4}$$

čia $u_i(t)$ – Algoritmo sugeneruotas valdymo signalas (pagreičio dedamoji);

Δt – laiko žingsnis.

Norint suvaldyti agentų greitį reikia priskirti maksimalų greitį v_{max}, kuris neleistų agentams pasiekti nerealistiskų greičių. Formulė tam pasiekti atrodytų taip:

$$\|v_i(t)\| \leq, v_{max} \tag{5}$$

Kuriamas algoritmas yra decentralizuoto tipo, todėl agentų sąveika turi būti tik su lokaliai esančiais kaimynais. Visą kaimynų sąveiką galime apibrėžti kaip:

$$N_i(t) = \left\|x_j(t) - x_i(t)\right\| < R, \qquad j \neq i \tag{6}$$

čia $N_i(t)$ - kaimynų aibė apibrėžiama kaip visi agentai j, kurių atstumas nuo agento i yra mažesnis už ryšio spindulį R.

Jeigu agentų padėtį apibrėžiame kaip:

$$x_j(t) = (x_j, y_i) \tag{7}$$

$$x_i(t) = (x_i, y_i) \tag{8}$$

tai, $$\left\|x_j(t) - x_i(t)\right\| = \sqrt{\left(x_j - x_i\right)^2 \left(y_j - y_i\right)^2} \tag{9}$$

Ši aibė yra dinamiška, kadangi ji kinta kiekvieno algoritmo ciklo metu, priklausomai nuo agentų judėjimo. Tai užtikrina, kad kiekvienas dronas reaguoja tik į artimiausius kaimynus, o ne į visą spiečių, kas atitinka decentralizuoto valdymo principą.

Agento greičio kintamumas priklauso nuo valdymo signalo. Jis susidaro iš trijų pagrindinių komponentų sumos:

$$u_i(t) = u_i^{ats}(t) + u_i^{lyg}(t) + u_i^{koh}(t). \tag{10}$$

čia $u_i^{ats}(t)$ - atskyrimo komponentė kuri apsirašo pagal žemiau esančią formulę:

$$u_i^{ats}(t) = \sum_{j=N_i} k_{ats} \frac{x_i(t)-x_j(t)}{\|x_i(t)-x_j(t)\|^2}, \quad (11)$$

čiak_{ats} - agentų atskyrimo koeficientas, kurį galima pasirinkti pagal kokiu atstumimu reaguos šalimai esantys agentai. Ši komponentė užtikrina saugų atstumą tarp agentų.

$u_i^{lyg}(t)$ yra lygiavimo komponentė. Ji užtikrina, kad spiečiaus agentai derina savo greitį su kaimynais. Ši komponentė yra apskaičiuojama pagal šią formulę:

$$u_i^{lyg}(t) = k_{lyg}\left(\frac{1}{|N_i(t)|}\sum_{j\in N_i(t)}\big(v_j(t) - v_i(t)\big)\right). \quad (12)$$

čiak_{lyg} - lygiavimo koeficientas, kuriuo kaip ir su atskyrimo koeficientu reguliuojama kaip stipriai kaimynai derinasi greitį su kaimynais.

Trečioji valdymo signalo dedamoji yra kohentiškumo komponentė (*angl. Cohesiveness)*, kuri priskiria agentams kiek kaimynai gali priartėti prie duoto UAV. Jos formulė yra:

$$u_i^{koh} = k_{koh}\left(\frac{1}{|N_i(t)|}\sum_{j\in N_i(t)}\big(x_j(t) - x_i(t)\big)\right) \quad (13)$$

čia k_{koh} - kohentiškumo koeficientas, kuriuo yra reguliuojama priartėjimo atstumo ribos.

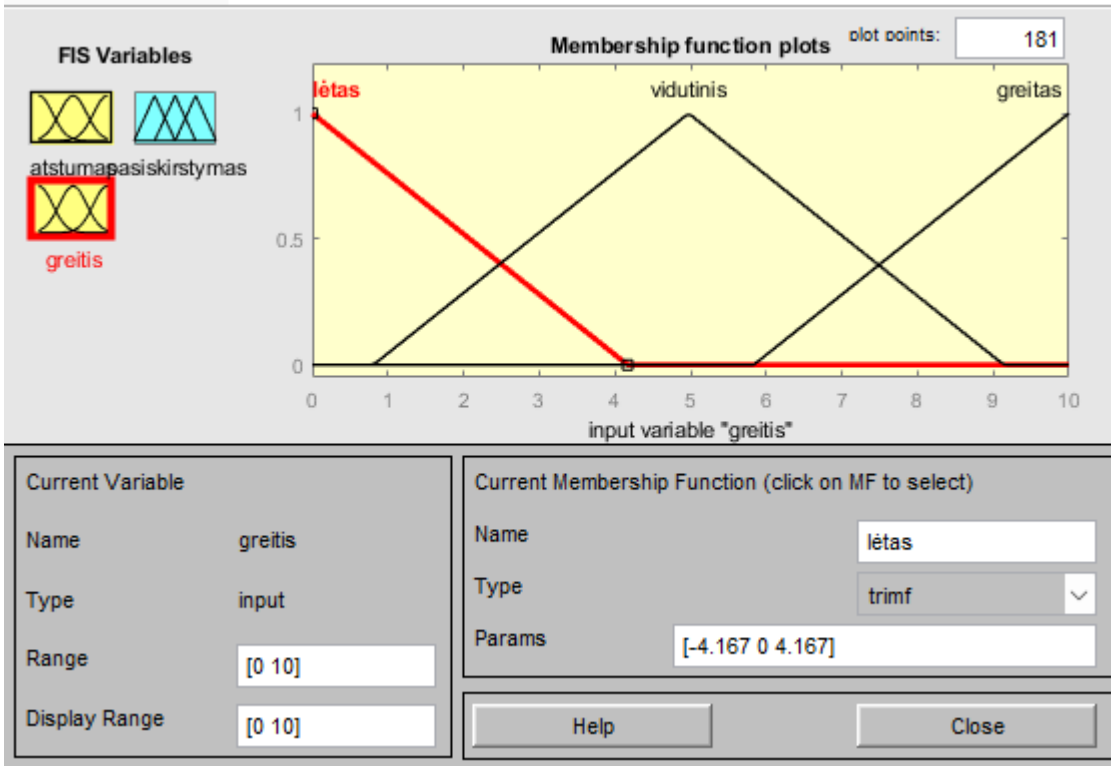
Ši komponentė traukia agentą link kaimynų centro, taip palaikant spiečiaus ryšį visos misijos metu. Šių trijų komponentų balansas lemia sklandų ir efektyvų spiečiaus judėjimą kiekvienoje misijos fazėje.

2.3.2. *„Fuzzy”* matematinio modelio veikimo principai

„Fuzzy” matematinis modelis yra apibūdinamas kaip neapibrėžtų ir neaiškių duomenų keitimas į konkrečius kontrolės sprendimus. Šis modelis itin gerai tinka decentralizuotai spiečiaus kontrolei, kadangi lokalūs duomenys gali būti trikdomi išorinių veiksnių ir būti netikslūs. Šio matematinio modelio loginė sprendimų seka nėra paremta *„TRUE”* ar *„FALSE”* sąlygomis, tačiau iš to iškyla galimybė apsibrėžti sau norimas logines narystes. [33]

„Fuzzy” matematinis modelis susideda iš trijų dalių:
1. pirminė duomenų įvestis;
2. taisyklių generavimas;
3. norimo rezultato išeities funkcijos apibrėžimas.

Viena iš kritinių algoritmo sąlygų yra nesusidūrimas su šalia skrendančiais agentais. Tokiu atveju norint palaikyti atstumą tarp agentų, *„Fuzzy”* matematiniame modelyje galima naudoti tokias duomenų įvestis kaip „mažas”, „vidutinis” ar „didelis” atstumas. Šalia atstumo taip pat įvedamos ir santykinio greičio įvestys, pvz. „lėtas”, „vidutinis” ar „greitas”. Prie šių įvesčių yra priskiriamos funkcijos, kurios pateikia apytikslę skaitinę reikšmę.



8 pav. Greičio duomenų įvestis "Fuzzy" matematiniame modelyje

8 paveiksle matoma, kaip atrodo greičio kintamasis *„Fuzzy"* matematiniame modelyje *„MATLAB"* programoje. Ši funkcija yra trikampinė, kuri parodo tikslius žodinių įvesčių skaitinius pikus. Ji yra tinkama tada, kuomet įvestis turi turėti optimalią reikšmę. [34] Taip pat dažnai naudojama trapezoidinės arba gausinio tipo funkcijos formos, kurios gali keisti kiek žodinė reikšmė kinta nuo skirtingos skaitinės ribos. Šios linijos parodo, kokią sprendimo įtaką turės kiekviena iš žodinių įvesčių skalėje nuo 0 iki 1. Šiuo atveju, jeigu drono greitis momente būtų 5 m/s, *„Fuzzy"* modelis toliau aprašytose taisyklėse imtų „vidutinis" greičio reikšmę kaip 1. Žodinių reikšmių vienoje duomenų įvestyje galima naudoti ir daugiau, tačiau nuo to priklauso taisyklių sudėtingumas bei agento operacinės sistemos apkrova.

Pagal šias įvestis ir jų norimas reikšmes toliau yra rašomos algoritmo taisyklės. Šio modelio taisyklės apibūdina kokią duomenų išvestį norima gauti pagal momente turimą situaciją. Pavyzdžiui, jeigu atstumas nuo kito agento yra „mažas", o greitis yra „didelis", tada krypties keitimo svarba yra didelė. Tokiu būdu yra apibrėžiamas agentų judėjimas skirtingomis misijos fazėmis. Taisyklės taip pat turi savo svorį, kuris keičia pagal taisykles gauto rezultato reikšmę. [35] 9 paveiksle matomas pavyzdys, kaip yra aprašomos taisyklės su dviem kintamaisiais.

Fuzzy Inference System (FIS) Plot | Membership Function (MF) Editor | Rule Editor | Control Surface ×

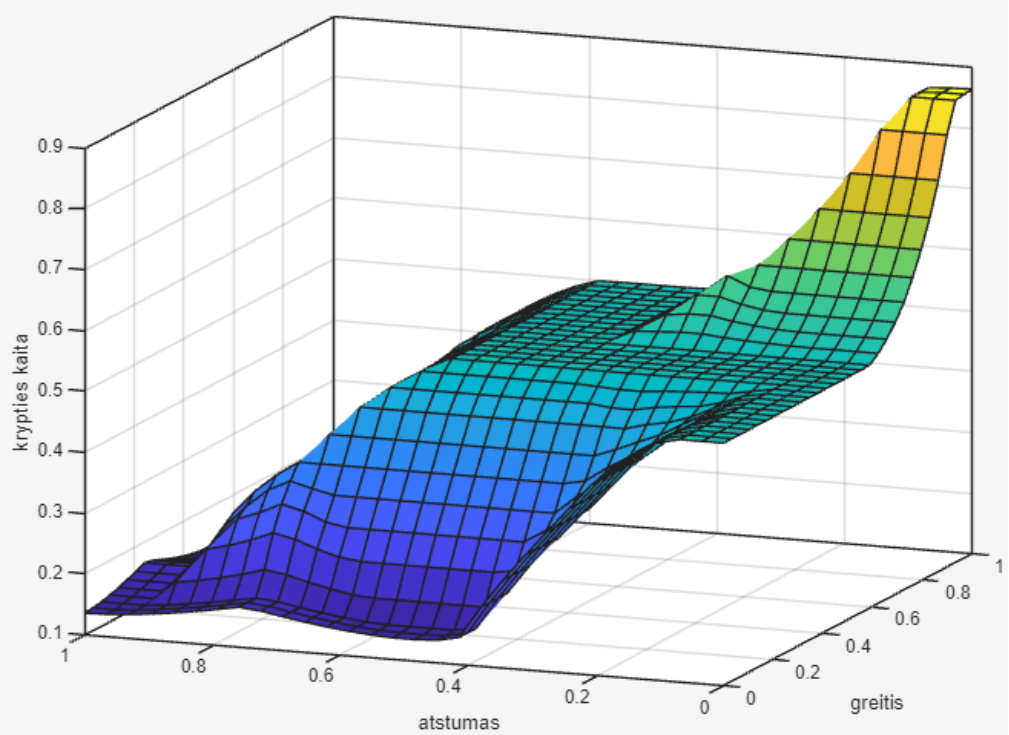
System: mamdanitype1

Add All Possible Rules | Clear All Rules

	Rule	Weight	Name
1	If greitis is lėtas and atstumas is mažas then krypties kaita is vidutinė	0.5	rule1
2	If greitis is vidutinis and atstumas is mažas then krypties kaita is vidutinė	0.8	rule2
3	If greitis is greitas and atstumas is mažas then krypties kaita is didelė	1	rule3
4	If greitis is lėtas and atstumas is vidutinis then krypties kaita is maža	1	rule4
5	If greitis is vidutinis and atstumas is vidutinis then krypties kaita is vidutinė	0.5	rule5
6	If greitis is greitas and atstumas is vidutinis then krypties kaita is vidutinė	0.8	rule6
7	If greitis is lėtas and atstumas is didelis then krypties kaita is maža	1	rule7
8	If greitis is vidutinis and atstumas is didelis then krypties kaita is maža	1	rule8
9	If greitis is greitas and atstumas is didelis then krypties kaita is maža	1	rule9

9 pav. *„Fuzzy“* modelio taisyklių generavimas pagal turimas įvestis ir išeitis

9 paveiksle matoma, kad taisyklių generavimo metu galima pasirinkti koks yra įvesčių tarpusavio ryšys, kadangi įvesčių sujungimo operatorius gali būti *„AND“* arba *„OR“*. Sujungimo etapas apibrėžia, kaip visi aktyvuoti *„Fuzzy”* taisyklių rezultatai yra integruojami į vieną bendrą išėjimo *„Fuzzy”* rinkinį. Kiekviena taisyklė, priklausomai nuo jos aktyvumo, generuoja dalinį išėjimo narystės funkcijos fragmentą, atitinkantį konkrečią valdymo komandą (pvz., posūkio kampo korekciją ar greičio pokytį). Šie fragmentai nėra vertinami atskirai. Vietoje to jie sujungiami į vieną bendrą *„Fuzzy”* rinkinį taikant maksimalumo operatorių, kuris kiekvienam išėjimo kintamojo taškui parenka didžiausią narystės laipsnį iš visų taisyklių. Toks sujungimo principas leidžia sistemai vienu metu atsižvelgti į kelias tarpusavyje konkuruojančias ar papildančias taisykles, išlaikant sklandų ir nuoseklų valdymo signalą. Maksimalumo operatorius užtikrina, kad stipriausiai aktyvuotos taisyklės turėtų didžiausią įtaką galutinei išėjimo funkcijai, o silpnesni signalai nebūtų prarasti, bet prisidėtų prie bendros formos.



10 pav. Krypties kaitos rezultatų pagal atstumo ir greičio įvestis 3D grafikas

10 paveiksle pateikti pagal nustatytas taisykles sugeneruoti krypties kaitos rezultatai. Matoma, jog skalėje nuo 0 iki 1 krypties kaita turi didžiausią reikšmę, kuomet greitis yra didžiausias bei atstumas yra mažiausias. Mažiausios reikšmės matomos, kai greitis yra mažas, o atstumas – didelis. Taip galima sumodeliuoti visus norimus kintamuosius bei jų rezultatus, kad spiečiaus judėjimas būtų kuo tikslesnis reikiamais misijos etapais.

2.3.3. Bepiločių orlaivių spiečiaus algoritmo trikdžių įvertinimas bei apžvalga

Realiomis sąlygomis bepiločių orlaivių spiečiaus jutikliai ir ryšio sistemos niekada neveikia idealiomis matematinėmis sąlygomis. Išoriniai veiksniai, kaip atmosferos trikdžiai, elektromagnetinės trukdos, specialiai kuriami ryšio trikdžiai bei blokai ar mechaniniai jutiklių netikslumas, gali iškraipyti agentų gaunamą informaciją apie kaimynų padėtis. Todėl matematiniame modelyje būtina įvertinti šiuos netikslumus.

Šiuo atveju visus trikdžius matematiškai galima įvertinti įvedant Gauso, kitaip vadinamą baltąjį/ atsitiktinį triukšmą, kuris yra pridedamas prie spiečiaus agentų ir jų kaimynų atstumų matavimų. Tokiu atveju kaimynų padėtys kito agento atžvilgiu būtų apskaičiuojama pagal šią formulę:

$$\widetilde{x_j}(t) = x_j(t) + \varepsilon_{ij}(t) \tag{14}$$

čia $\varepsilon_{ij}(t)$ – Gauso (baltasis) triukšmas, kuris įvertinamas naudojant Gauso skirstinį su nuliniu vidurkiu ir gaunama dispersija:

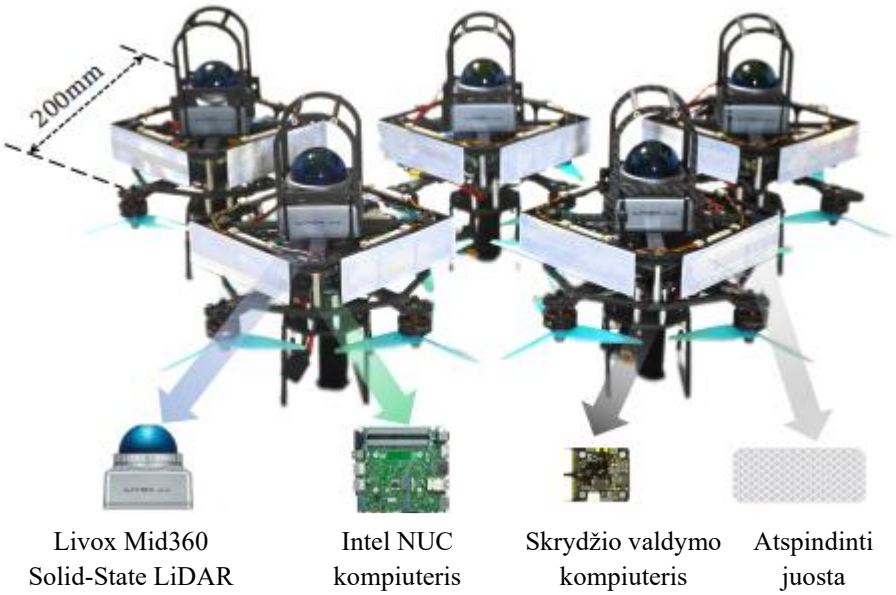
$$\varepsilon_{ij} \sim G(0, \sigma^2) \tag{15}$$

Formulėje dispersija σ^2 apibrėžia matavimo paklaidų sklaidą – kuo ji didesnė, tuo labiau iškreipta informacija apie kaimyno padėtį. Šis triukšmo modelis pasirinktas dėl to, kad jo panaudojimas ir

pritaikymas gali būti itin platus bei jis yra matematiškai pagrįstas ir naudojamas jutiklių triukšmui matuoti. Jis tinka šiam algoritmui, kadangi trikdžiai dažniausiai atsiranda iš daugelio mažų nepriklausomų šaltinių, kurių bendra įtaka pagal centrinę ribinę teoremą artėja į Gauso skirstinį. Gauso triukšmo modelis taip pat svarbus naudojant *„Fuzzy“* matematinį modelį, kadangi jis įvertina *„Fuzzy“* logikos duomenų neapibrėžtumą, todėl realiomis situacijomis atsiradus matavimo paklaidoms, spiečiaus sistema išlaikytų funkcinį stabilumą.

2.4. Bepiločių orlaivių spiečiaus saviorganizacijos algoritmo integracija

Pasirinkus bepiločių orlaivių spiečiaus saviorganizacijos algoritmą reikia apsibrėžti, kokiu būdu algoritmas bus integruojamas į spiečių. Kad saviorganizacinis algoritmas veiktų, viena iš svarbiausių detalių yra šalia skrendančių agentų suradimas ir apibrėžimas. Kiekvienas spiečiaus agentas algoritmo ciklo starto metu turi nuskenuoti savo aplinką ir surasti arčiausiai skrendančius agentus. Tai atlikti realioje situacijoje yra naudojama arba RGB kameros arba LiDAR jutikliai ir davikliai. (Žr 11 pav.) Agentui yra priskiriamas aptikimo spindulys, kuris aptinka artimus agentus reikalingus skaičiavimams.



11 pav. Dronų komplektacija naudojant LiDAR aptikimo sistemą [36]

„MATLAB” simuliacijos metu yra apskaičiuojami atstumai tarp aplinkinių agentų ir išrenkami artimiausi. Kiekvieno ciklo metu yra gaunama dinamiška duombazė, kuri kinta spiečiaus agentams keičiant pozicijas. Iš arčiausiai esančių dronų yra surenkama reikalinga informacija, t.y. atstumas, greitis, atstumas nuo spiečiaus lyderio ir kiti duomenys, kurie gali turėti įtakos spiečiaus trajektorijai. Kai daroma prielaida, kad agentų pozicijos įvardijimos *X* ir *Y* koordinačių sistemoje, atstumas tarp dviejų agentų apskaičiuojamas pagal formulę:

$$L = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \tag{16}$$

čia *L* – atstumas tarp dviejų spiečiaus agentų; *Xi, Xj* – dviejų agentų x koordinatės plokštumoje;

Yi bei *Yj* – dviejų agentų y koordinatės plokštumoje.

Rašant šios dalies algoritmą dalinis pseudokodas atrodytų taip:

Parametrai:
1. agentų X koordinatės;
2. agentų Y koordinatės;
3. dt = Laiko žingsni;
4. N = Agentų kiekis.

Kiekvieno laiko žingsnio ciklas:
1. jei reikia, nuskaityti arba atnaujinti pozicijas poz[i] ir greičius v[i];
2. kiekvienam agentui rasti kaimynus ir apskaičiuoti poveikį;
3. atnaujinti greitį ir poziciją agentui i pagal surinktą informaciją;
4. atnaujinti pozicijas visiems agentams.

Po duomenų surinkimo jie yra įvedami į *„Fuzzy“* įvestis ir susiejami su norimais rezultatais. Pagal anksčiau minėtas įvestis galima išgauti šiuos rezultatus: trajektorijos kampo keitimą bei greičio pokytį. Trajektorijos kampo keitimas reguliuoja atskiro agento orientaciją, o greičio pokytis reguliuoja judėjimo intensyvumą.

Tuo pačiu yra kuriamos „defuzzifikacijos“ taisyklės, pagal kurias įvestys bus koreliuojamos su rezultatais. Akivaizdus taisyklių sudarymo prioritetas yra spiečiaus saugumas, saugaus atstumo palaikymas bei greitas ir sklandus misijos fazės įvykdymas. Taisyklės, kurios yra susiję su susidurimų prevencija, užtikrina greitas trajektorijos korekcijas, kuomet agentai būriuojasi artimuose atstumuose arba skrenda dideliu greičiu. Formacijos taisyklės užtikrina, jog ir veikiant trajektorijos korekcijoms būtų palaikoma norima skrydžio formacija. Šiais atvejais *„Fuzzy“* logika puikiai tinka sudaryti sklandų taisyklių vykdymą net ir gaunant daug duomenų iš jutiklių arba veikiant neprognozuotiems išorės veiksniams. Aplinkos pokyčiai taip pat decentralizuotai persiduoda ir aplinkiniams agentams, kadangi vieno drono trajektorijos kaita priartina jį prie šalia skrendančio agento. Kitame algoritmo cikle šis pasikeitimas turi įtakos ateinančiam taisyklių vertinimui ir dronai gauna naujus nurodymus verčiančius pataisyti atstumą. Tai gali veikti ir atvirkščiai, t.y. jeigu agentas atsiranda per toli nuo šalia skrendančio drono, taisyklės verčia mažinti atstumą didinant greitį ar pakreipiant trajektoriją.

2.4.1. Bepiločių orlaivių spiečiaus kokybės saviorganizacijos vertinimo kriterijus

Norint įvertinti bepiločių orlaivių spiečiaus saviorganizacijos algoritmo efektyvumą, įvedamas kiekybinis vertinimo kriterijus E(t). Šiuo kriterijumi galima pamatuoti, kaip glaudžiai spiečius yra susispaudęs arba išsisklaidęs koordinačių sistemoje. Formulę apskaičiuoti šiam kriterijui galima aprašyti taip:

$$E(t) = \frac{1}{N}\sum_{i=1}^{N}\|x_i(t) - x_c(t)\|. \quad (17)$$

čia $x_c(t)$ – centrinė spiečiaus koordinatė. Ši vertė imama, kaip visų spiečiaus agentų koordinačių vidutinė reikšmė apskaičiuojama kaip:

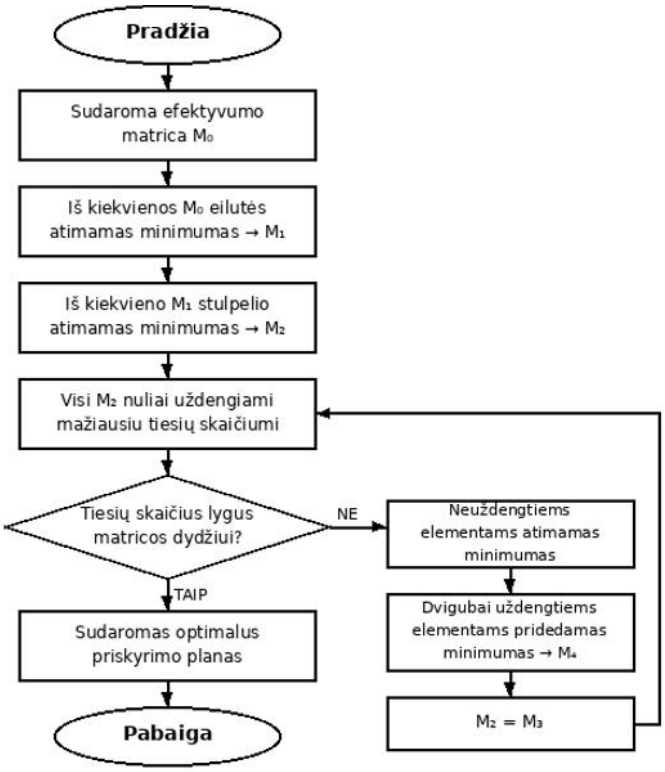
$$x_c(t) = \left(\frac{1}{N}\sum_{i=1}^{N} x_i(t),\ \frac{1}{N}\sum_{i=1}^{N} y_i(t)\right) \quad (18)$$

Pagal turimas vertes apskaičiuota E(t) reikšmė gali būti arba maža, arba didelė. Jeigu reikšmė yra maža, tai indikuoja, jog spiečius yra glaudžiai susibūriavęs. Atvirkščiai, didelė E(t) reikšmė parodo spiečiaus išsisklaidymą. Tai ne visada gali parodyti kuriamo algoritmo kokybę, kadangi 2-oje fazėje

bei 3-ios fazės pradiniuose žingsniuose spiečius yra specialiai išsisklaidęs, o likusiais momentais spiečius organizuojasi į artimą būrį.

2.4.2. Vengrų algoritmo veikimo principai

Pirmoje algoritmo fazėje bepiločių orlaivių spiečiaus agentai iš pradinės pakilimo pozicijos turi sustoti į puslankio formaciją aplink spiečiaus lyderį. Tai gali sukelti nenorimų trikdžių, kuomet spiečiaus agentai juda tik *„Fuzzy“*matematinio modelio principu. *„Fuzzy“* matematinis modelis apibrėžia tik agentų greitį ir trajektoriją. Be papildomų nurodymų, spiečius užtruktų žymiai ilgesnį laiką tinkamų vietų užėmimo etapui, kadangi agentai stumdytūsi ir nepasidalintų vieta. Vengrų algoritmas yra kombinatorinis optimizavimo algoritmas, skirtas priskyrimo problemai spręsti. Vengrų algoritmas priskyrimo problemą sprendžia $O(n^3)$ laiko sudėtingumu, kur n yra dvišalio grafo vienos dalies dydis. [37] Algoritmo pagrindą sudaro principas, kad pridėjus ar atėmus konstantą iš bet kurios efektyvumo matricos eilutės ar stulpelio, optimalus priskyrimo sprendinys išlieka nepakitęs. [38] Šių principų dėka yra apskaičiuojamas mažiausiai kainos reikalaujantis paskirstymas. Pagrindinė algoritmo eiga matoma žemiau pateiktoje loginėje sekoje. (Žr 12 pav.)



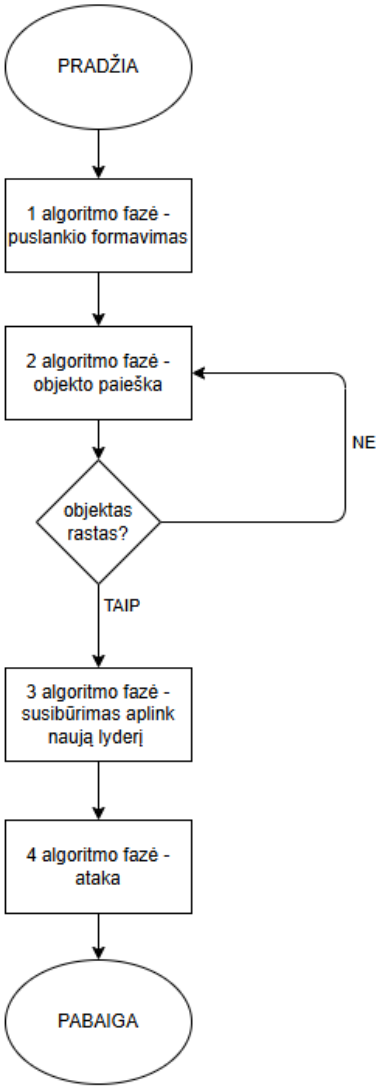
12 pav. Klasikinė Vengrų algoritmo loginė seka [38]

Šis algoritmas gali būti naudojamas ir atsiradus kintamoms sąlygoms, pvz. kai darbų ir darbuotojų pasiskirstymo problemoje atsiranda daugiau darbų ar darbuotojų arba kai bepiločių orlaivių spiečiuje padaugėja agentų arba pozicijų, kuriose reikia atsidurti. Tai gali reikalauti dinaminio Vengrų algoritmo, tačiau kuriamame algoritme kintamų salygų 1-oje fazėje nėra, todėl užtenka ir klasikinės versijos.

3. Eksperimentinė dalis

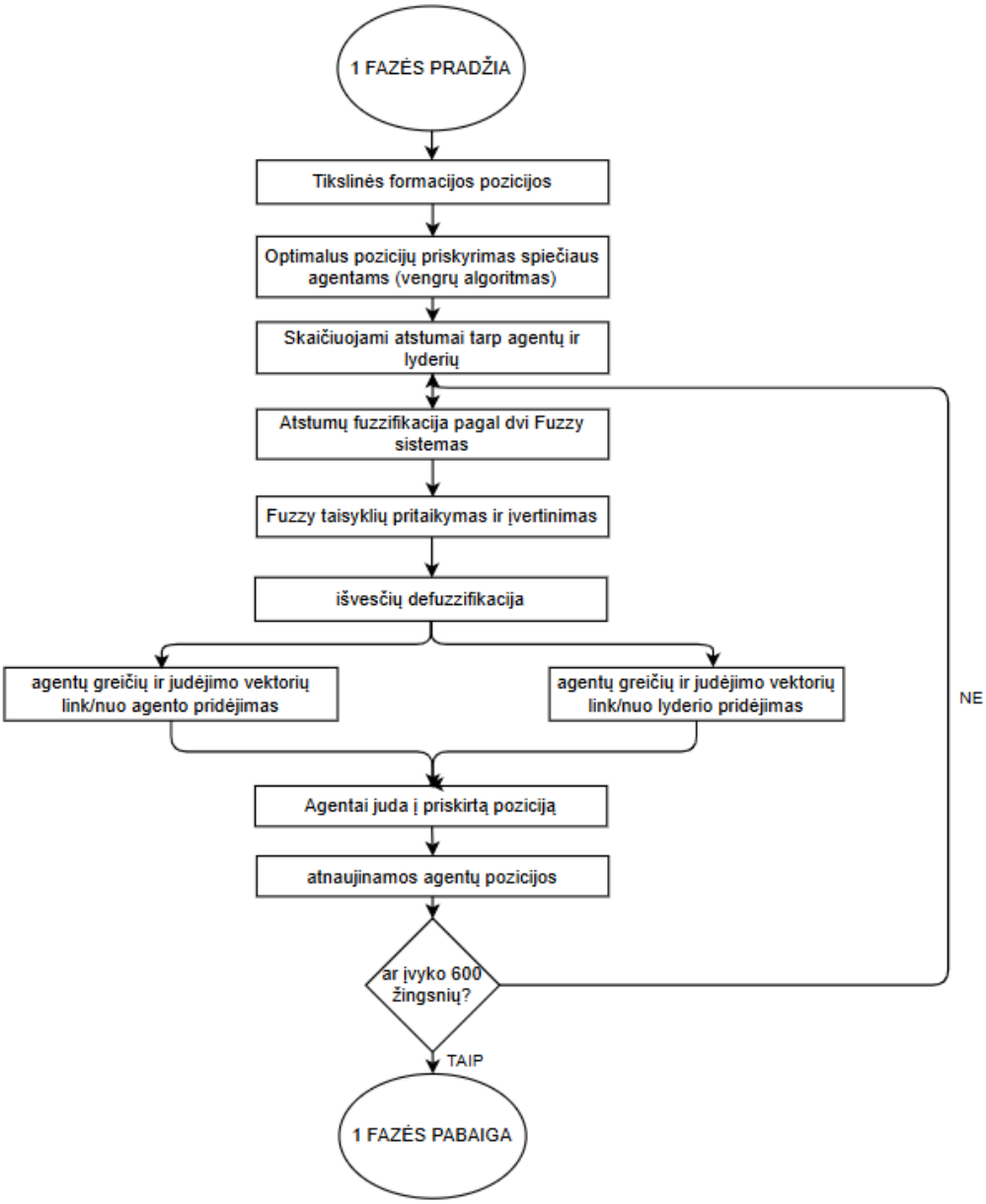
3.1. Bepiločių orlaivių spiečiaus algoritmo loginė schema

Kuriamas bepiločių orlaivių spiečiaus algoritmas yra apibrėžtas logine algoritmo schema. Žemiau galima matyti bendrą algoritmo schemą bei atskirų algoritmo fazių schemas.



13 pav. Bepiločių orlaivių spiečiaus algoritmo loginė seka

13 paveiksle galima matomos keturios algoritmo fazės. Tarp antrosios ir trečiosios fazės yra įterpiama objekto radimo sąlyga, dėl kurios gali keistis standartinė algoritmo eiga. Kiekviena fazė yra atitinkamai apibrėžta likusiomis algoritmo schemomis.

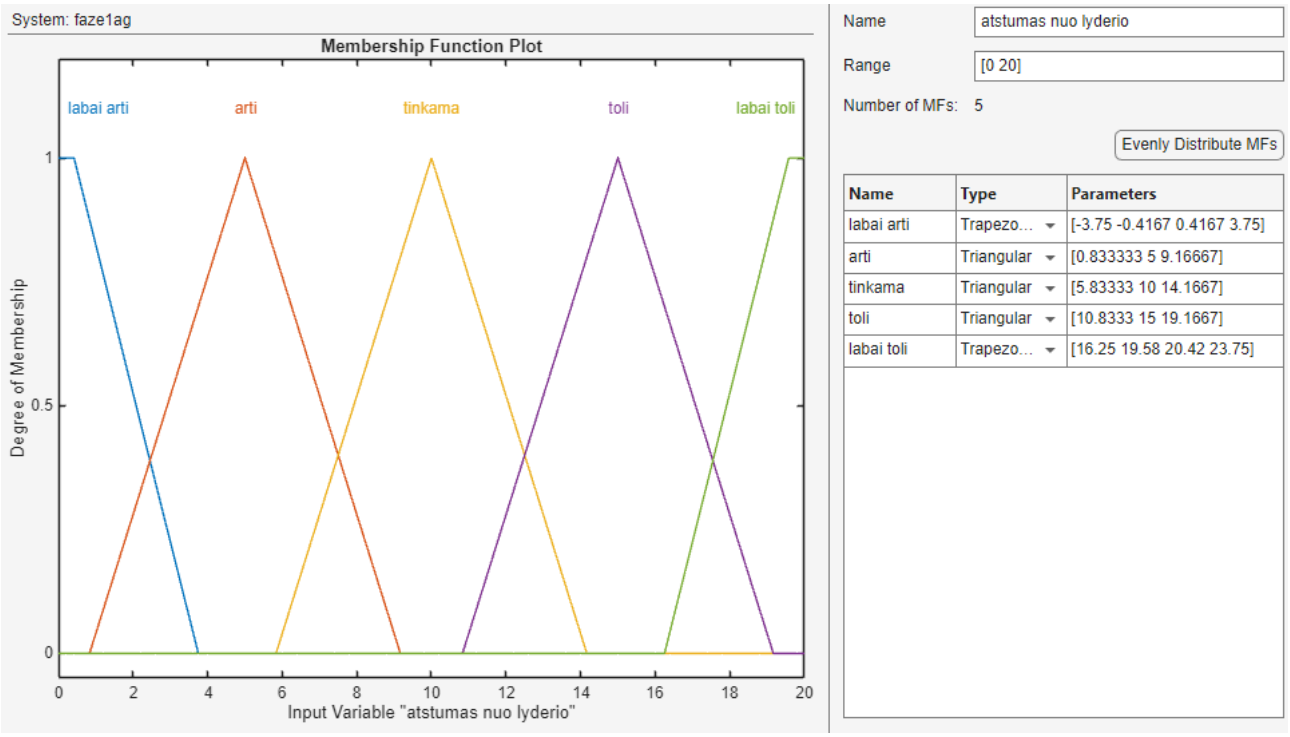


14 pav. Pirmosios algoritmo fazės loginė seka

14 paveiksle matomas pradinis algoritmo etapas – spiečiaus susiformavimas puslankiu aplink lyderį. Šiai fazei yra skiriama 600 žingsnių, kurių atlikimo greitis 0,4s. Tokiu atveju ši fazė įvyksta per 240 sekundžių. Puslankio formacija yra puikiai priderinta tolimesniai ieškojimo fazei dėl tolygaus išsiskristymo į priekį koordinačių sistemoje, tačiau vienas puslankis paskirstytas po 9° laipsnius aplink lyderį sugeneruoja per mažus atstumus tarp agentų, kad per norimą laiko tarpą agentai tiksliai pasiektų savo poziciją. Šiuo atveju yra generuojama du puslankiai, kurie išdėsto agentų pozicijas tais pačiais kampais nuo spiečiaus lyderio, bet jie sustoja šachmatų formacija. Tai padidina atstumus tarp atskirų agentų iki atstumo, kuris netrukdo agentams per norimą laiko tarpą pasiekti optimalią poziciją. Matoma, kad pirmi algoritmo žingsniai yra sugeneruoti norimų agentų pozicijų nustatymui koordinačių sistemoje.

Antrame loginiame žingsnyje pagal Vengrų optimizavimo algoritmą yra priskiriamos optimalios pozicijos kiekvienam iš 20 spiečiuje skrendančių agentų. Nuo trečiojo etapo pradedamas algoritmo ciklas, kuriame skaičiuojamos atstumų tarp agentų reikšmės.

Po atstumų apskaičiavimo yra integruojama pirmosios fazės *„Fuzzy“* matematinio modelio įvestys ir jų galimos reikšmės, rezultatų skaičiavimo taisyklės bei rezultatų išvestys su galimomis vertėmis. Pirmajai algoritmo fazei yra naudojama dvi *„Fuzzy“* sistemos. Pirmoji reguliuoja greitį link/nuo artimiausio agento, o antroji - greitį link/nuo lyderio. Abejose sistemose naudojamos dvi tokios pačios įvestys – atstumas nuo lyderio ir atstumas nuo artimiausio agento. Žemiau paveikslėliuose matoma abiejų įvesčių narystės funkcijas.



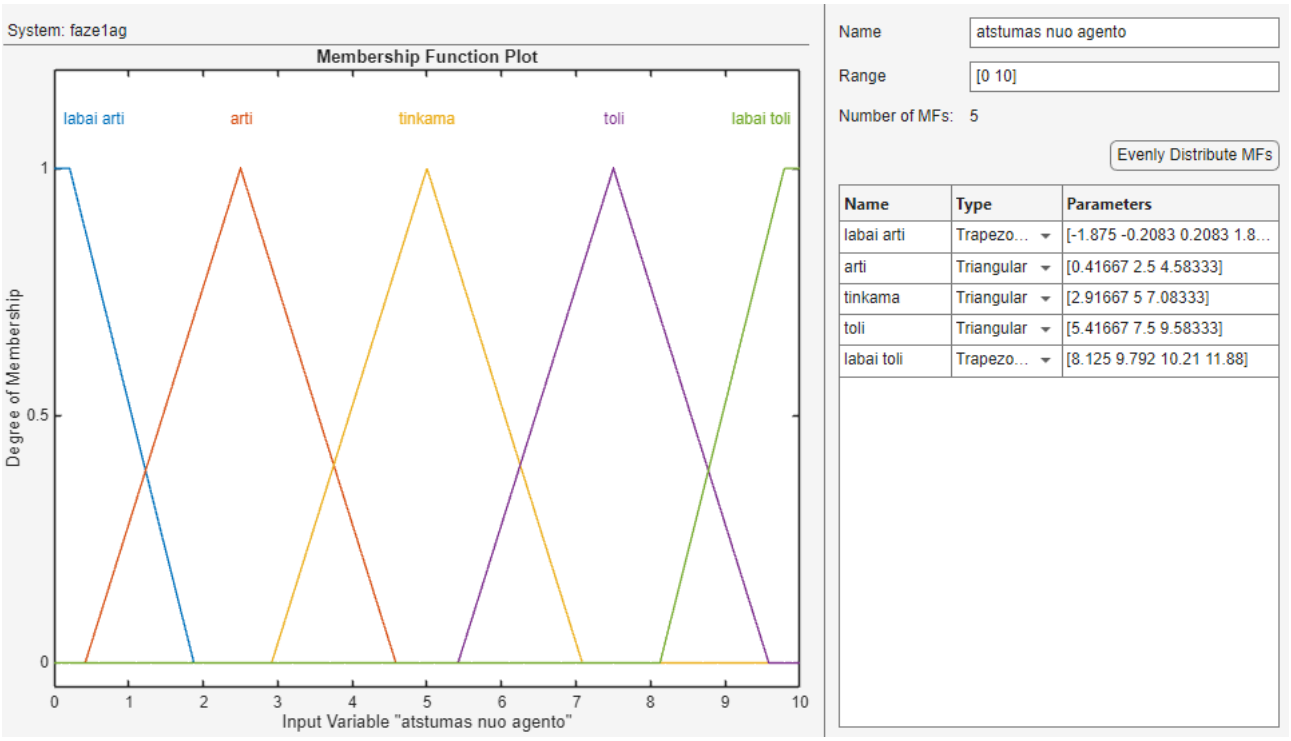
15 pav. Pirmosios fazės atstumo nuo lyderio įvesties narystės funkcijų grafikas (greičio link/nuo lyderio ir greičio link/nuo artimiausio agento *„Fuzzy“* sistemos)

15 paveiksle matoma, kad atstumo nuo lyderio funkcija turi 2 trapezoidines ir 3 trikampines narystes, kurios yra tolygiai išdalinamos per atstumą [0, 20] m. Naryščių pikai yra pateikiami 2 lentelėje.

2 lentelė. Pirmosios fazės atstumo nuo lyderio naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Labai arti	Iki 0.42 m
Arti	5 m
Tinkamas	10 m
Toli	15 m
Labai toli	Nuo 19,6 m

16 paveiksle matomas atstumo nuo artimiausio agento naryščių funkcijų grafiką.

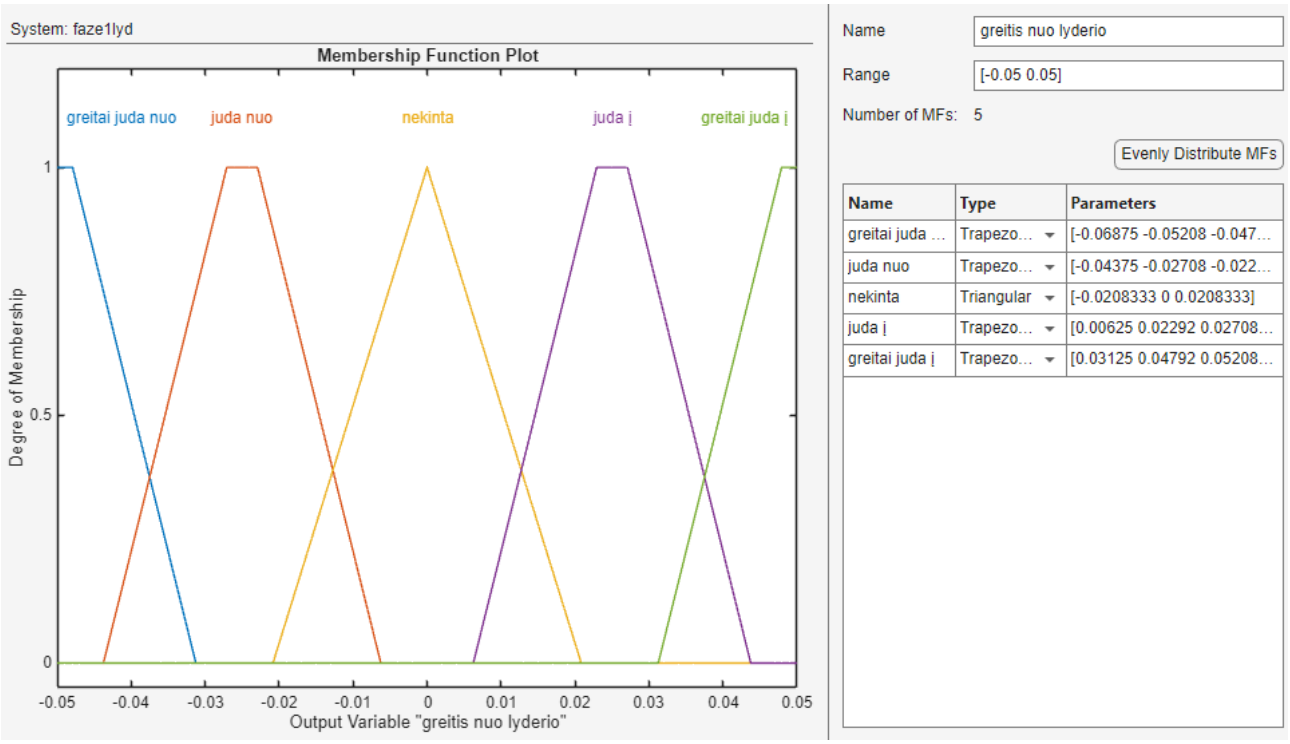


16 pav. Pirmosios fazės atstumo nuo artimiausio agento įvesties narystės funkcijų grafikas (greičio link/nuo lyderio ir greičio link/nuo artimiausio agento *„Fuzzy“* sistemos)

Grafikas turi 5 tokias pat narystes, kaip ir atstumo nuo lyderio narystės funkcija, tačiau jos ribos yra [0, 10] m. (Žr. 3 lentelę)

3 lentelė. Pirmosios fazės atstumo nuo artimiausio lyderio naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Labai arti	Iki 0,21 m
Arti	2,5 m
Tinkamas	5 m
Toli	7,5 m
Labai toli	Nuo 10,21 m



17 pav. Pirmosios fazės greičio nuo lyderio fuzzy sistemos išvesties narystės funkcijos grafikas

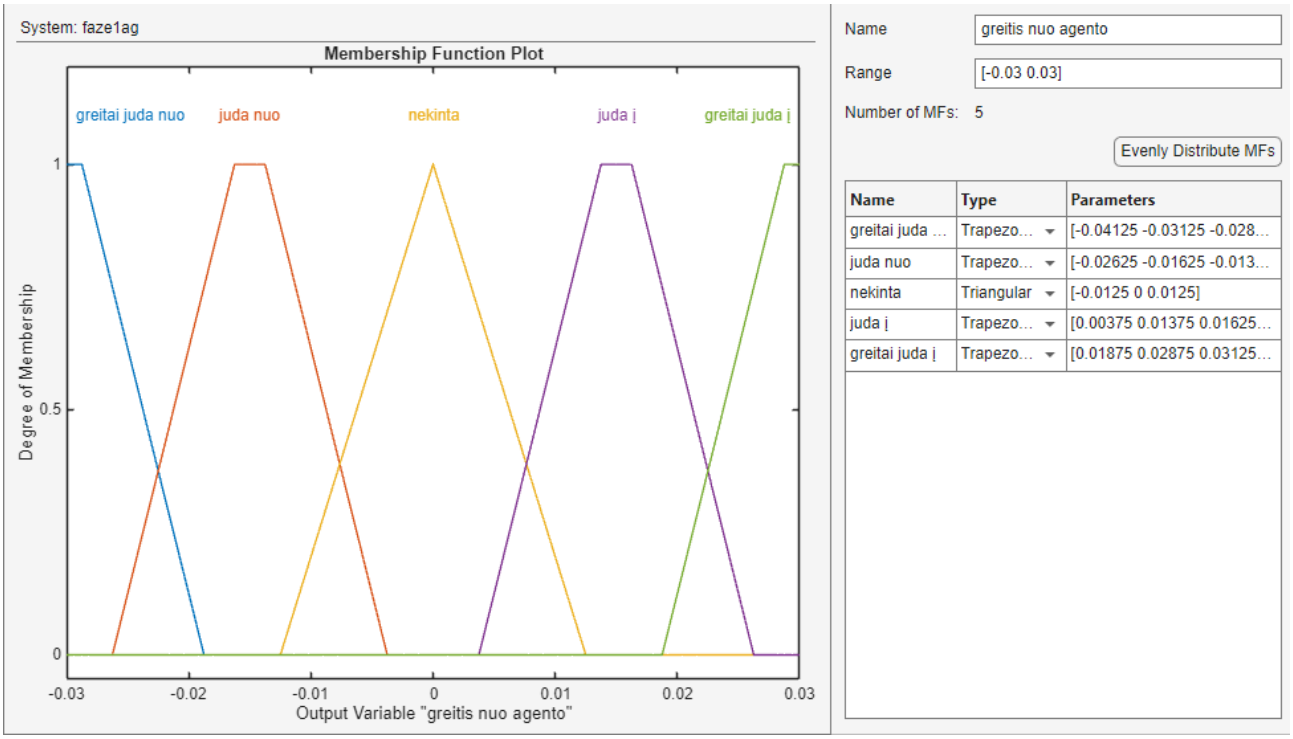
Šioje išvesties funkcijoje, matomoje 17 paveiksle, yra naudojamos 4 trapezoidinės narystės ir viena trikampinė. Greičiui nuo lyderio yra priskiriama riba [-0,05 0,05] m/žingsnis. Jų pikų paskirstymas matomas 4 lentelėje.

4 lentelė. Pirmos fazės greičio nuo lyderio naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Greitai juda nuo	Iki -0,05 m/žingsnis
Juda nuo	Nuo -0,03 iki -0,02 m/žingsnis
Nekinta	0 m/žingsnis
Juda į	Nuo 0,02 iki 0,03 m/žingsnis
Greitai juda į	Iki 0,05 m/žingsnis

Šioje narystės funkcijoje neigiamas greitis reiškia judėjimą nuo lyderio, o tinkama pozicija nuo lyderio neleidžia agentams judėti toliau arba arčiau lyderio.

Greičio nuo artimiausio agento narystės funkcijų grafikas yra taip pat panašus į praeitos išvesties grafiką. Vieninteliai pakeitimai yra siauresnės greičio ribos, kurios gali kisti [-0,03 0,03] diapazone, bei naryščių pikai. (žr. 18 pav.) (Žr. 5 lentelę)



18 pav. Pirmosios fazės greičio nuo artimiausio agento *„Fuzzy“* sistemos išvesties narystės funkcijos grafikas

5 lentelė. Pirmos fazės greičio nuo artimiausio agento naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Greitai juda nuo	Iki -0,03 m/žingsnis
Juda nuo	Nuo -0,016 iki -0,013 m/žingsnis
Nekinta	0 m/žingsnis
Juda į	Nuo 0,013 iki 0,016 m/žingsnis
Greitai juda į	Iki 0,03 m/žingsnis

Sugeneravus norimas *„Fuzzy“* sistemų įvestis bei jų išvestis yra atitikamai sugeneruojamos abiejų sistemų veikimo taisyklės. 6 Lentelėje yra pateikiama abiejų sistemų taisyklių kombinacijų matricos.

6 lentelė. Pirmosios fazės greičio link/nuo lyderio *„Fuzzy“* sistemos taisyklių kombinacijų matrica

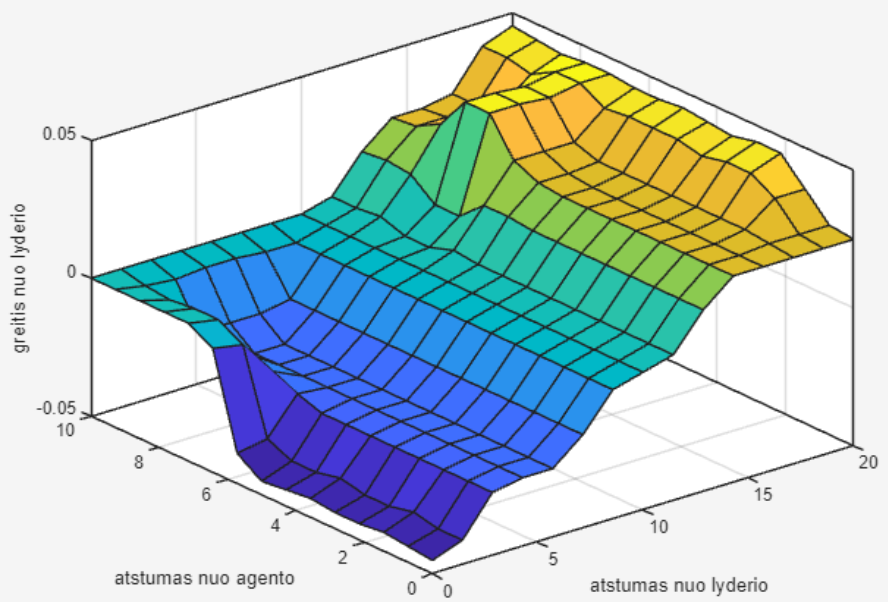
		Atstumas nuo lyderio				
		Labai arti	Arti	Tinkamas	Toli	Labai toli
Atstumas nuo artimiausio agento	Labai arti	Greitai juda nuo	Juda nuo	Nekinta	Juda į	Juda į
	Arti	Greitai juda nuo	Juda nuo	Nekinta	Juda į	Greitai juda į
	Tinkamas	Greitai juda nuo	Juda nuo	Nekinta	Juda į	Greitai juda į
	Toli	Greitai juda nuo	Nekinta	Nekinta	Greitai juda į	Greitai juda į
	Labai toli	Greitai juda nuo	Juda nuo	Nekinta	Greitai juda į	Greitai juda į

Analizuojant greičio link/nuo lyderio *„Fuzzy“* sistemos taisykles matoma, kad greičio reikšmė yra labiau priklausoma nuo atstumo nuo lyderio įvesties reikšmių, kadangi norima išvengti susidūrimų su lyderiu arba greičiau artėti link jo, jeigu atstumas yra per didelis (žr. 6 lentelę). Matricoje išsiskiria keli langeliai, kuomet atstumas nuo lyderio yra „arti“, „toli“ bei „labai toli“, o atstumas nuo agento atitinkamai yra „labai toli“, „toli“, bei „labai arti“. Kuomet agentas yra toli nuo lyderio bei toli arba labai toli nuo artimiausio kaimyno, greitis link lyderio gali būti didesnis, kadangi yra mažiau rizikos susidurti su kaimynu. Kitu atveju, jeigu atstumas nuo lyderio yra mažas, o nuo artimiausio agento yra didelis, judėjimas sustoja dėl rizikos susidurti su lyderiu.

7 lentelė. Pirmosios fazės greičio link/nuo artimiausio agento *„Fuzzy“* sistemos taisyklių kombinacijų matrica

		Atstumas nuo lyderio				
		Labai arti	Arti	Tinkamas	Toli	Labai toli
Atstumas nuo artimiausio agento	Labai arti	Greitai juda nuo	Greitai juda nuo	Greitai juda nuo	Greitai juda nuo	Greitai juda nuo
	Arti	Juda nuo	Juda nuo	Juda nuo	Juda nuo	Greitai juda nuo
	Tinkamas	Nekinta	Nekinta	Nekinta	Nekinta	Nekinta
	Toli	Nekinta	Juda į	Juda į	Juda į	Juda į
	Labai toli	Nekinta	Nekinta	Juda į	Greitai juda į	Greitai juda į

Taip pat yra sudėliojama ir greičio link/nuo artimiausio agento *„Fuzzy“* sistemos taisyklių kombinacijos (žr. 7 lentelę). Atvirkščiai negu greičio link/nuo lyderio matricoje, greitis daugiausiai kinta nuo atstumo nuo artimiausio agento, bet taip pat yra išimčių. Kuomet atstumas nuo lyderio yra labai arti, o nuo kaimyno – toli, greitis nekinta, kadangi norima išvengti susidūrimo su lyderiu. Kuomet atstumas nuo kaimyno yra labai toli o nuo lyderio – labai arti arba arti, greičio išvestis taip pat yra „Nekinta“, nes norima išvengti susidūrimo. Po taisyklių įvertinimo yra generuojamas 3D grafikas, kuriame galima matyti, kaip greitis priklauso nuo dviejų įvesčių reikšmių.

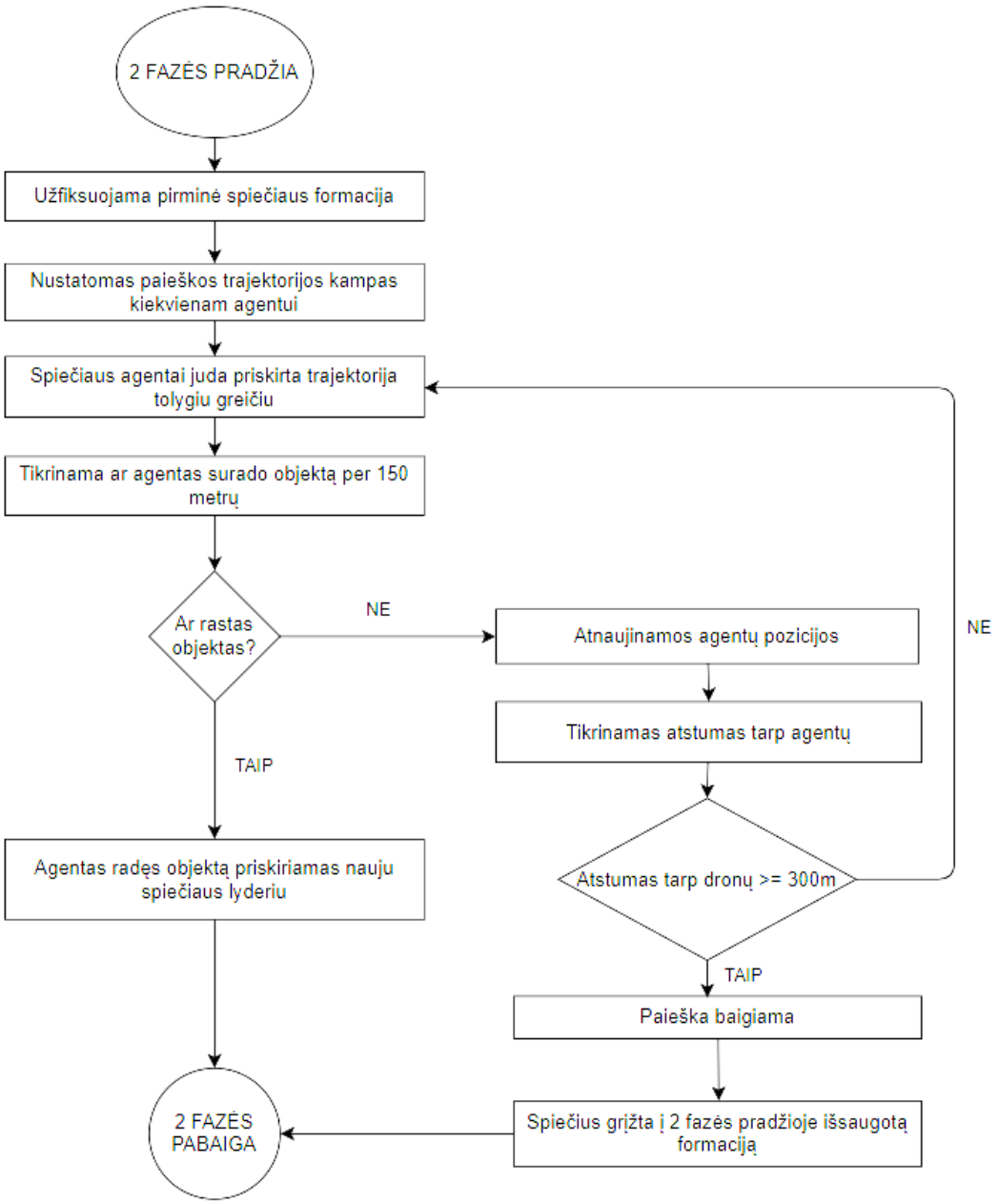


19 pav. Pirmosios fazės *„Fuzzy“* sistemos reikšmių 3D grafikas susidarantis iš atstumo nuo artimiausio agento bei lyderio įvesčių ir greičio nuo lyderio išvesties.

19 paveiksle matoma, kad šiuo atveju greičio didėjimas labiau priklauso nuo atstumo iki lyderio, bet prie didelių atstumo nuo agento reikšmių greitis taip pat išauga. Greičio link/nuo artimiausio agento atveju 3D grafikas labiau priklauso nuo atstumo iki kaimyno, bet tolimuose nuotoliuose nuo lyderio išauga.

Turint dvi pilnas *„Fuzzy“* sistemas ir apskaičiavus atstumus tarp artimiausio agento bei lyderio, vykdoma šių duomenų „fuzzifikacija“ abejoms sistemoms, t.y realios atstumų reikšmės yra verčiamos į žodines reikšmes, kurios dalyvauja narystės funkcijose. Kitas ciklo žingsnis pritaiko aukščiau minėtas sistemų taisykles, per kurias generuojamos abiejų sistemų išvestys. Po Išvesčių sugeneravimo, žodinė reikšmė yra „defuzzifikuojama“ ir yra gaunama reali greičio dedamoji. Taip pat yra apskaičiuojami atstūmimo, lygiavimo bei kohentiškumo komponentės, kurių suma skirta keisti agentų poziciją kitam ciklo žingsniui.

Ciklo pabaigoje yra vykdomas agentų pozicijos judėjimas koordinačių sistemoje ir yra išsaugojamos naujos agentų pozicijos. Po šių žingsnių ciklas kartojasi 600 kartų, po kurių yra gaunama norima pradinė agentų formacija.

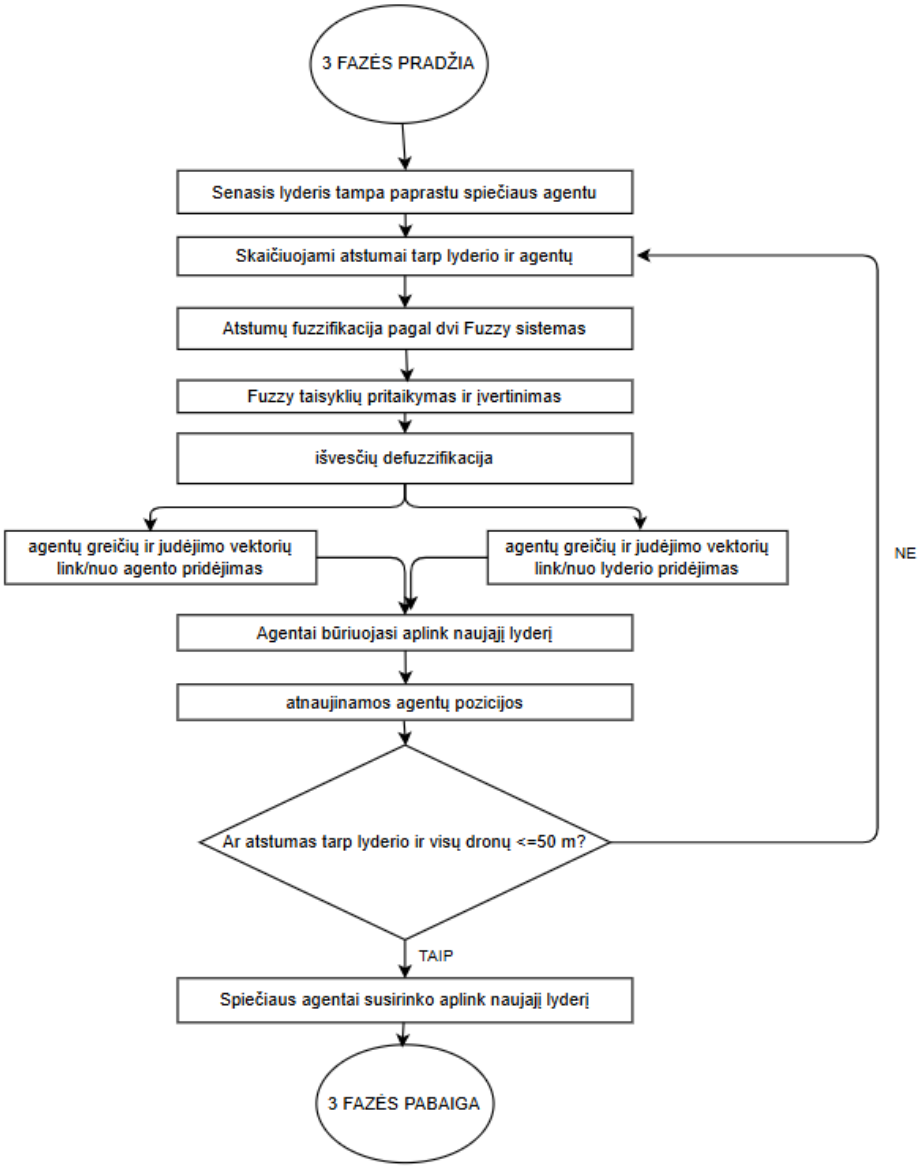


20 pav. Antrosios algoritmo fazės loginė seka

Turint 1-oje fazėje gautą spiečiaus formaciją yra pradedama antroji algoritmo fazė – objekto paieška. 20 paveiksle matoma, kad pirmasis antros fazės žingsnis yra užfiksuoti, kokiose koordinatėse buvo spiečiaus agentai bei lyderis. Šios pozicijos yra spiečiaus grįžimo koordinatės, jeigu paieška pasiektų maksimalų atstumą ir objektas nebūtų surastas. Toliau yra apskaičiuojamas tikslus kampas tarp lyderio ir kiekvieno agento. Šis kampas yra naudojamas trajektorijos nustatymui kiekvienam agentui, kad būtų padengtas maksimalus paieškos plotas.

Nustačius agentų trajektorijas, spiečiaus agentui yra užduodamas pastovus greitis $v_{ieš}$, kuriuo agentai išsisklaido objekto paieškai. Kiekviename ciklo žingsnyje yra tikrinama, ar agentas aptiko objektą per 150 metrų spindulį nuo agento pozicijos. Jeigu objektas yra surandamas, objektą suradęs agentas tampa spiečiaus lyderiu ateinančiai algoritmo fazei. Jeigu ne, yra atnaujinamos tame žingsnyje agentų užimamos pozicijos ir yra tikranama, koks atstumas yra tarp šalia skrendančių agentų. Jeigu atstumas

yra didesnis arba lygus 300 metrų, skelbiama paieškos pabaiga. Tai nutraukia paiešką, kadangi agento paieškos spindulys yra 150 metrų, todėl atstumui viršijant 300 metrų, pradeda atsirasti paieškos tarpai, per kuriuos galima neaptikti objekto. Tokiu atveju spiečiui duodama komanda tuo pačiu greičiu grįžti į savo buvusias pozicijas, kad būtų galima vėl susiburti ir iš naujo rengti misiją. Šioje fazėje žingsnių kiekis nėra ribojamas, o fazės pabaiga yra tapatinama su naujo lyderio išsirinkimu, jeigu agentas suranda objektą,, arba spiečiaus grįžimu į pradinę algoritmo formaciją.

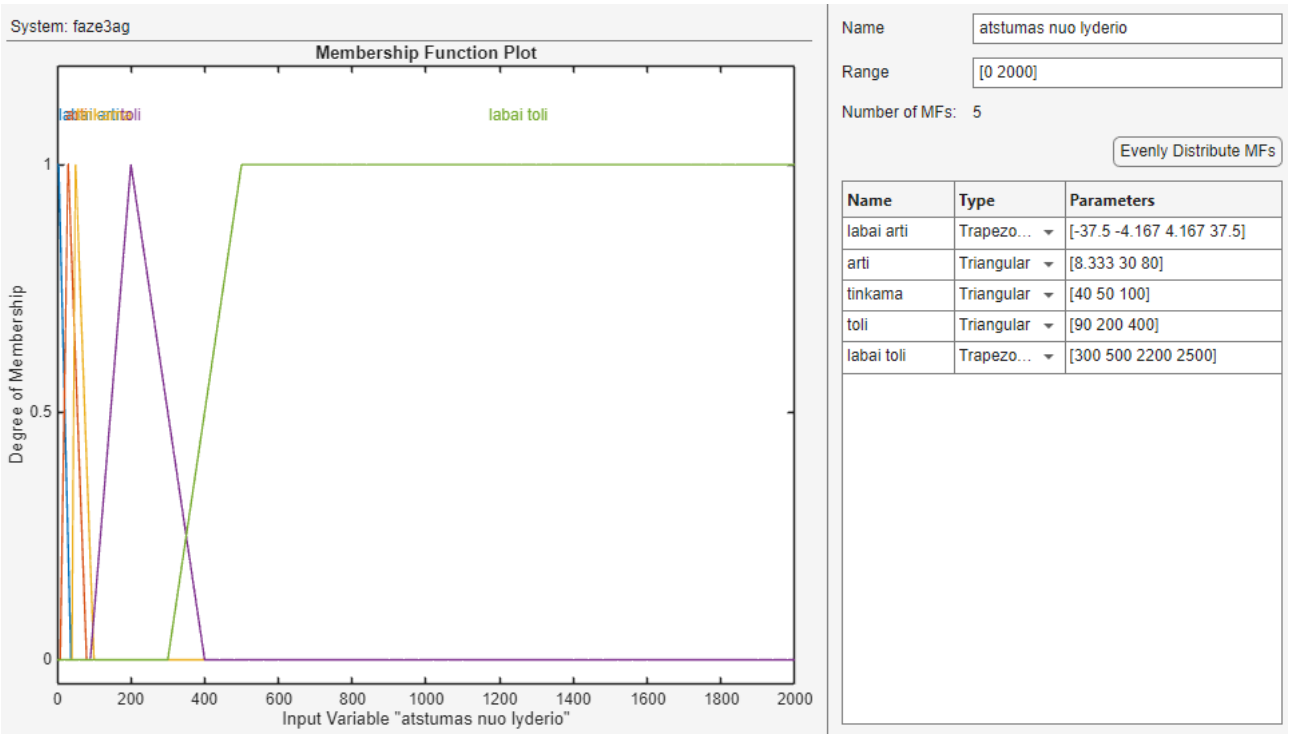


21 pav. Trečiosios algoritmo fazės loginė seka

Trečioji algoritmo fazė yra pradedama tik tada, jeigu antrosios fazės metu yra surandamas objektas. Tokiu atveju fazė prasideda nuo spiečiaus lyderių apkeitimo. Antrosios fazės pabaigoje yra išrenkamas naujasis spiečiaus lyderis, aplink kurį susibūriuos išsisklaidę spiečiaus agentai, tačiau senasis lyderis praranda prasmę spiečiaus valdyme. Dėl to jis yra prijungiamas prie likusių agentų, kad galėtų judėti link naujojo lyderio ir susigrupuoti. (Žr. 20 pav.) Toliau yra pradedami reikiami skaičiavimai spiečiui judėti link lyderio, t.y. apskaičiuojami atstumai tarp lyderio ir artimiausio agento. Nuo šių žingsnių algoritmas tampa panašus į pirmosios fazės judėjimą. Didžiausi pakeitimai

yra pirminiai atstumai tarp spiečiaus agentų bei konkrečios formacijos nebuvimas susirinkus prie lyderio. Tolimesniuose žingsniuose vėl naudojamos dvi *„Fuzzy"* sistemos, kurios reguliuoja greitį link/nuo lyderio ir greitį link/nuo artimiausio agento.

Nors trečiosios fazės *„Fuzzy"* sistemų pavadinimai ir funkcijos yra tokios pačios, kaip ir pirmosios fazės. Jų įvesčių narystės funkcijos yra visiškai kitokios, bet tinkančios abiejoms sistemoms (žr. 23 pav.)



22 pav. Trečiosios fazės atstumo nuo lyderio įvesties narystės funkcijų grafikas (greičio link/nuo lyderio ir greičio link/nuo artimiausio agento *„Fuzzy"* sistemos)

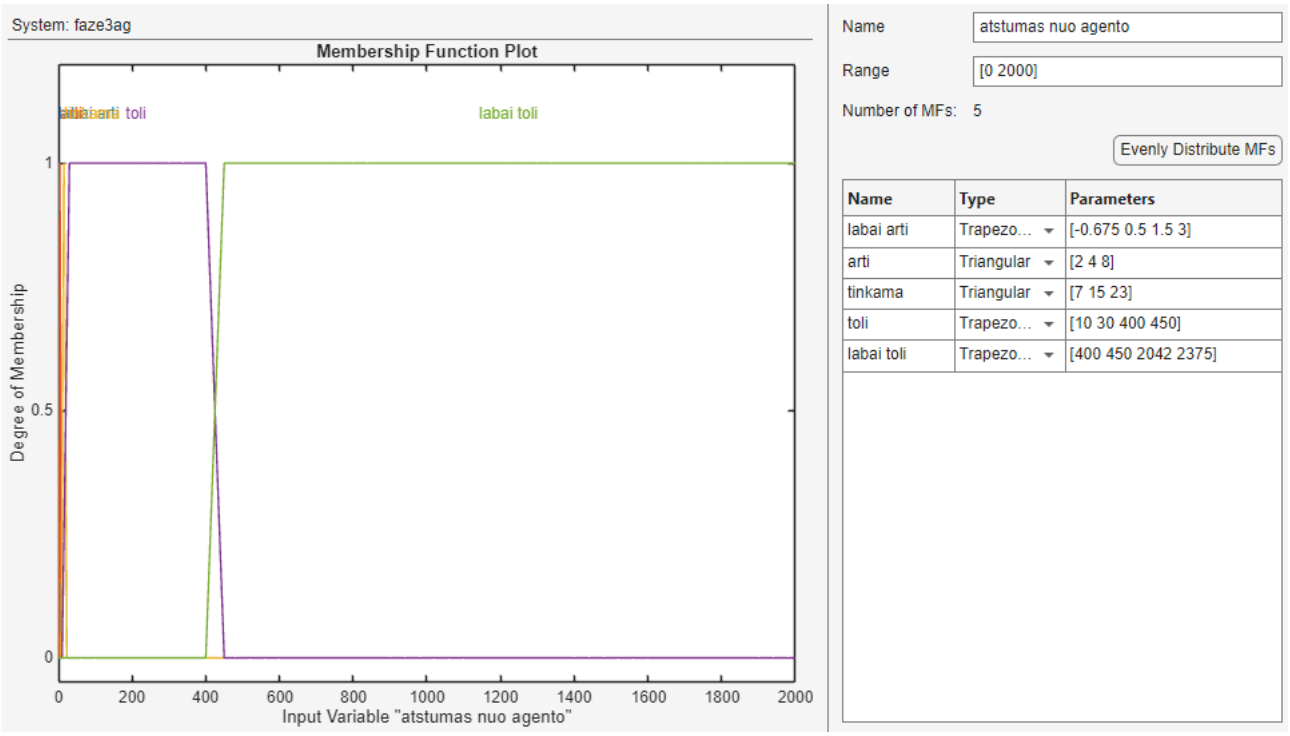
22 paveiksle yra matoma pagrindinis pakeitimas nuo pirmos fazės *„Fuzzy"* sistemų – ši narystės funkcija turi [0, 2000] metrų ribą dėl itin plačiai išsidėsčiusio spiečiaus. Nors atstumai fazės pradžioje gali būti ir labai dideli, fazės pabaigoje spiečius susibūriuoja gana arti. Kad sistemos įvestis teisingai atsižvelgtų į atstumą artimuose bei tolimuose atstumuose yra naudojami dviejų trapezoidinių ir trijų trikampinių naryščių pikai (žr. 8 lentelę).

8 lentelė. Trečiosios fazės atstumo nuo lyderio naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Labai arti	Iki 4.2 m
Arti	30 m
Tinkamas	50 m
Toli	200 m
Labai toli	Nuo 500 m

Tokiu atveju agentai fazės pradžioje gali judėti maksimaliu greičiu link lyderio, o atskridus artimesniu atstumu pradeda lėtėti. Šioje fazėje spiečius neturi būti labai susispaudęs prie lyderio, todėl „labai arti" narystė veikia tik kaip paskutinė apsauga nuo avarijų.

Atstumo nuo agento įvestis yra ganėtinai panaši į prieš tai nagrinėtą dydį (žr. 23 pav.).



23 pav. Trečiosios fazės atstumo nuo artimiausio agento įvesties narystės funkcijų grafikas (greičio link/nuo lyderio ir greičio link/nuo artimiausio agento *„Fuzzy"* sistemos)

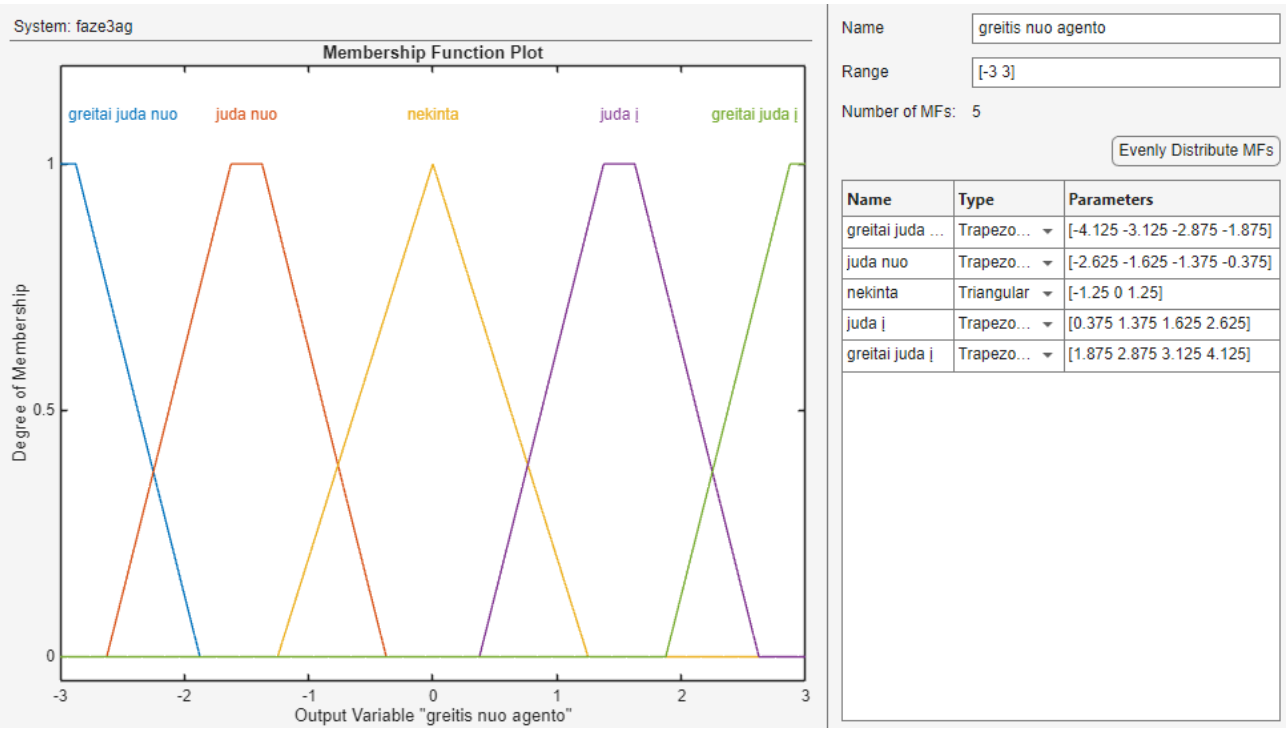
23 paveiksle matoma, kad šios naryščių funkcijos ribos yra tokios pat, kaip ir atstumo nuo lyderio naryščių, tačiau yra naudojami glaudesni rėžiai artimų atstumų narystėms bei didesni rėžiai tolimų atstumų narystėms (žr. 9 lentelę).

9 lentelė. Trečiosios fazės atstumo nuo artimiausio agento naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Labai arti	Iki 0,5 m
Arti	4 m
Tinkamas	15 m
Toli	Nuo 30 iki 400 m
Labai toli	Nuo 450 m

Naryščių ribos artimiems atstumams yra siauresnės, nes spiečiaus agentai dėl didelio jų kiekio spiečiuje turi didesnę tikimybę atsidurti mažesniu atstumu nuo kito agento, nei nuo lyderio. Didelės tolimų atstumų naryščių ribos taip pat leidžia spiečiui ilgesnį laiką skristi maksimaliu leistinu greičiu.

Trečiosios algoritmo fazės *„Fuzzy"* sistemų išvestys taip pat yra greičiai link/nuo agento arba lyderio, tačiau greičiai yra žymiai didesni, nei pirmoje fazėje dėl didesnio spiečiaus išsisklaidymo. (žr. 25 pav.)



24 pav. Trečiosios fazės greičio nuo artimiausio agento fuzzy sistemos išvesties narystės funkcijos grafikas

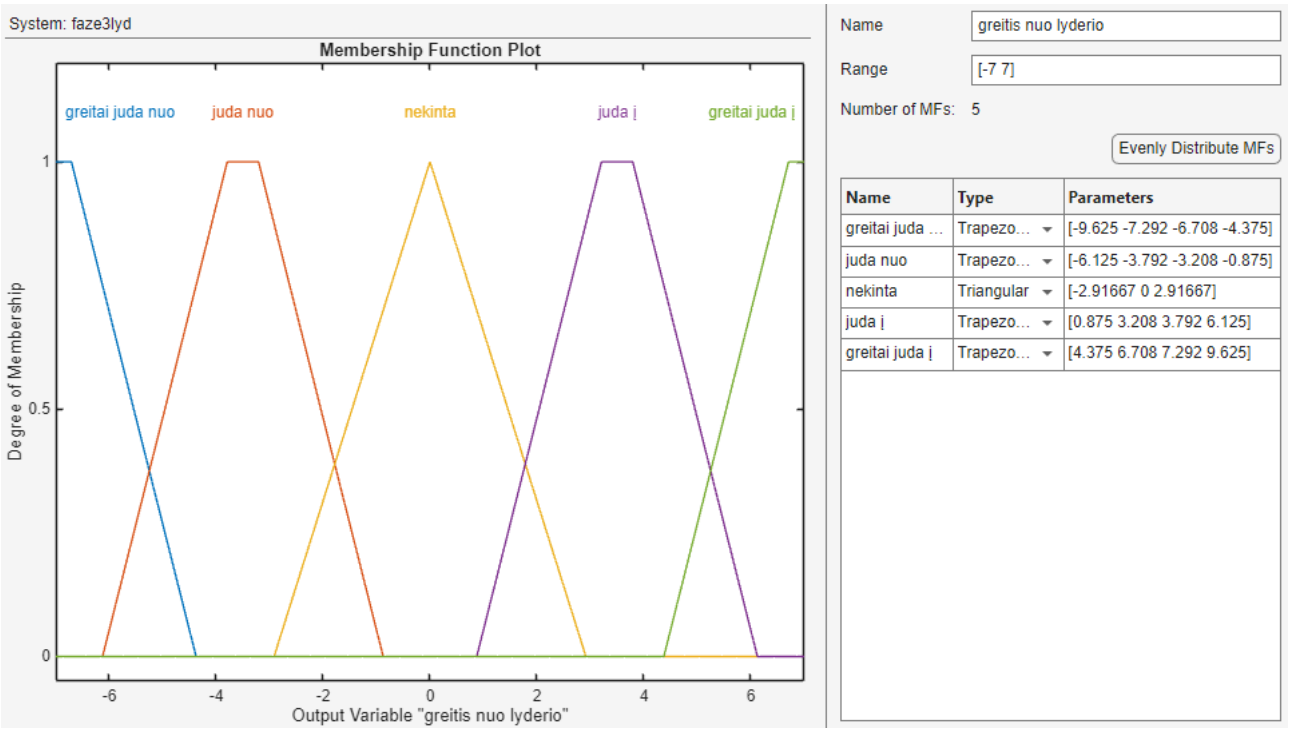
24 paveiksle matoma, kad greičio link/nuo artimiausio agento naryščių ribos yra [-3 3] m/žingsnis. Greičių paskirstymas per narystes yra gana tolygus naudojant 4 trapezoidines ir 1 trikampinę narystę. 10 lentelėje galima matyti greičio naryščių pikus.

10 lentelė. Trečiosios fazės greičio link/nuo artimiausio agento naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Greitai juda nuo	Iki -2,88 m/žingsnis
Juda nuo	Nuo -1,63 iki -1,38 m/žingsnis
Nekinta	0 m/žingsnis
Juda į	Nuo 1,38 iki 1,63 m/žingsnis
Greitai juda į	Nuo 2,88 m/žingsnis

Vienintelė tikampinė narystė yra „nekinta“, kadangi norimas jos rezultatas yra nekeisti pozicijos, o tai pasiekiama kuomet greitis yra 0 m/žingsnis. Visos kitos narystės turi platesnes pikų reikšmes, tačiau jos niekada neviršija užbrėžtų [-3 3] metrų/žingsnį ribos.

Greičio link/nuo lyderio narystės funkcija yra labai panaši į prieš tai išnagrinėtą narystės funkciją. Vieninteliai pakeitimai yra greičio ribos bei naryščių pikai.



25 pav. Trečiosios fazės greičio nuo lyderio *„Fuzzy"* sistemos išvesties narystės funkcijos grafikas

25 paveiksle matoma, kad greičio riba šiai narystės funkcijai yra [-7 7] m/žingsniui. Naryščių išsidėstymas ir jų forma yra tokia pati, kaip ir greičio link/nuo artimiausio agento.

11 lentelė. Trečiosios fazės greičio link/nuo artimiausio agento naryščių pikai

Funkcijos narystės	**Naryščių pikai**
Greitai juda nuo	Iki -6,7 m/žingsnis
Juda nuo	Nuo -3,8 iki -3,2 m/žingsnis
Nekinta	0 m/žingsnis
Juda į	Nuo 3,2 iki 3,8 m/žingsnis
Greitai juda į	Nuo 6,7 m/žingsnis

11 lentelėje yra matomi naryščių pikai. Naryščių pikai yra sudaryti su pakankamai nemažais tarpais tarp jų, tačiau nepikinės reikšmės dažnais atvejais persidengia su šalia esančiomis narystėmis. Tokiu būdu agento greitis gali būti ne tik pikinės vertės. Taip pat matoma, kad kraštiniai naryščių reikšmių pikai niekada neperžengia nurodytos funkcijos ribos.

Trečiosios fazės funkcijoms sujungti yra generuojamos dvi taisyklių matricos skirtos kiekvienai išvesčiai. (Žr. 12 ir 13 lentelę)

12 lentelė. Trečiosios fazės greičio link/nuo artimiausio agento *„Fuzzy”* sistemos taisyklių kombinacijų matrica

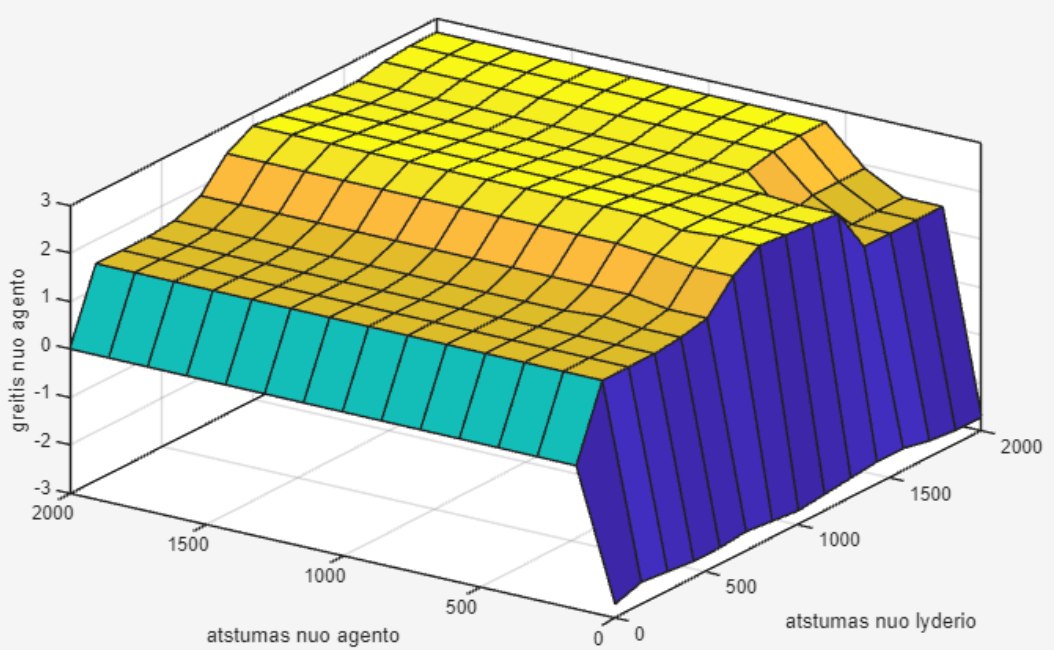
		Atstumas nuo lyderio				
		Labai arti	**Arti**	**Tinkamas**	**Toli**	**Labai toli**
Atstumas nuo artimiausio agento	**Labai arti**	Greitai juda nuo	Greitai juda nuo	Greitai juda nuo	Greitai juda nuo	Greitai juda nuo
	Arti	Juda nuo	Juda nuo	Juda nuo	Juda nuo	Greitai juda nuo
	Tinkamas	Nekinta	Nekinta	Nekinta	Nekinta	Nekinta
	Toli	Nekinta	Juda į	Juda į	Juda į	Juda į
	Labai toli	Nekinta	Nekinta	Juda į	Greitai juda į	Greitai juda į

13 lentelė. Trečiosios fazės greičio link/nuo lyderio *„Fuzzy”* sistemos taisyklių kombinacijų matrica

		Atstumas nuo lyderio				
		Labai arti	Arti	Tinkamas	Toli	Labai toli
Atstumas nuo artimiausio agento	Labai arti	Greitai juda nuo	Juda nuo	Nekinta	Juda į	Juda į
	Arti	Greitai juda nuo	Juda nuo	Nekinta	Juda į	Greitai juda į
	Tinkamas	Greitai juda nuo	Juda nuo	Nekinta	Juda į	Greitai juda į
	Toli	Greitai juda nuo	Nekinta	Nekinta	Greitai juda į	Greitai juda į
	Labai toli	Greitai juda nuo	Juda nuo	Nekinta	Greitai juda į	Greitai juda į

Trečiosios fazės taisyklės yra tokios pačios kaip ir pirmoje fazėje, kurios užtikrina greitesnį judėjimą kai agentai yra arba toli, arba arti nuo šalia skrendančio kaimyno ir nuo lyderio, o lėtesnis judėjimas pritaikoma, kai tarpusavio atstumai sumažėja.

Suvedus visas reikiamas *„Fuzzy”* sistemos dedamasias yra gaunamas 3D greičio grafikas, kuris priklauso nuo atstumų iki lyderio bei artimiausio kaimyno. Žemiau pateiktas grafikas yra greičio link/nuo artimiausio agento priklausomybė nuo atstumo nuo artimiausio agento bei lyderio (žr. 26 pav.**).**

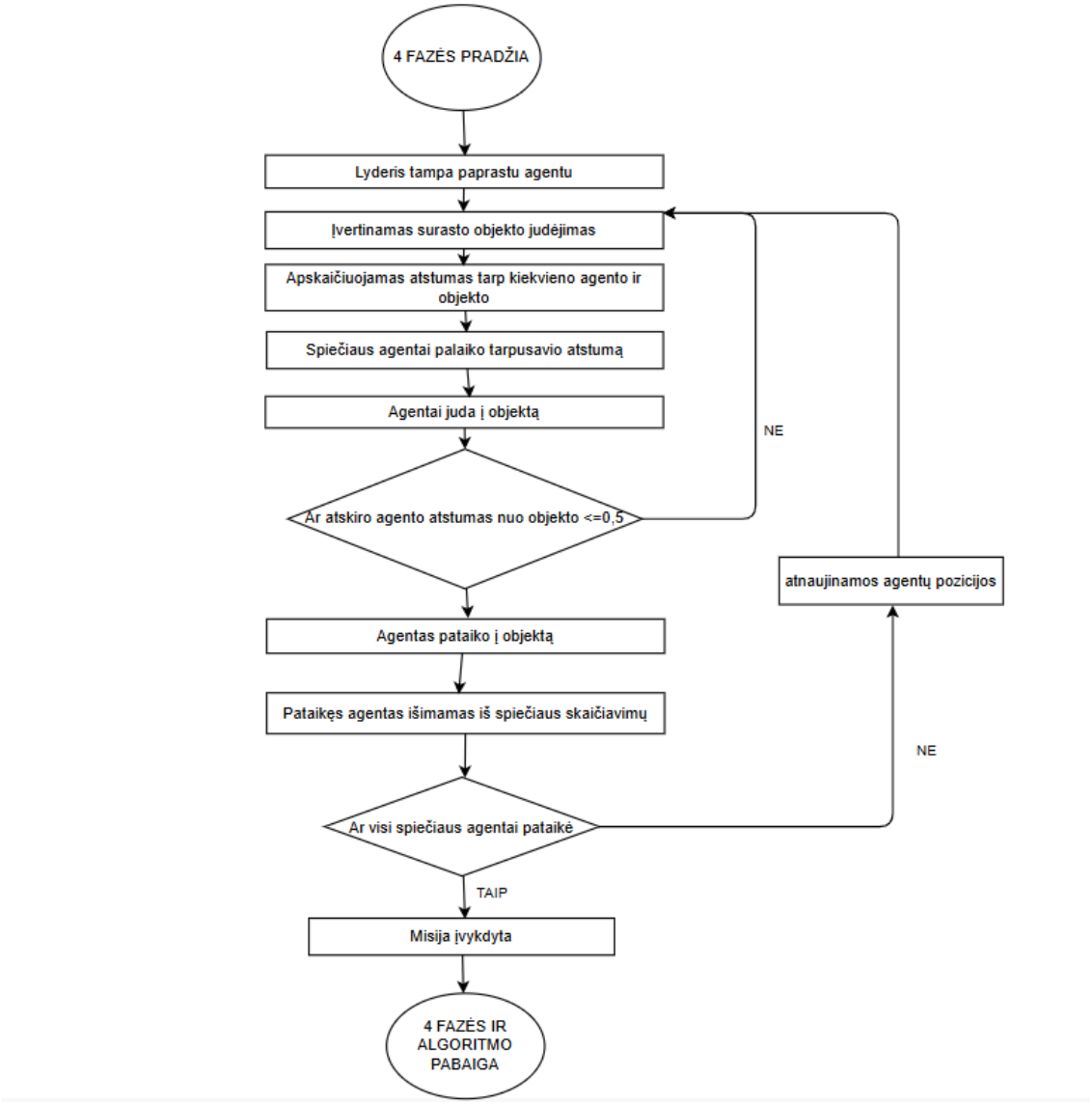


26 pav. Pirmosios fazės *„Fuzzy”* sistemos reikšmių 3D grafikas susidarantis iš atstumo nuo artimiausio agento bei lyderio įvesčių ir greičio link/nuo agento išvesties.

Grafike matoma, kad atstumas nuo lyderio turi žymiai mažesnę įtaką greičio nuo agento reikšmei, kadangi šios sistemos užduotis yra palaikyti atstumą nuo agento. Tik kai yra itin mažas atstumas nuo lyderio, greitis yra mažinamas iki minimalaus, tačiau jeigu atstumas nuo agento yra mažesnis negu norimas, sistema priverčia agentą judėti nuo kaimyno.

Suformavus dvi naujas *„Fuzzy”* sistemas, kurios yra tinkamos naudoti trečiąjai fazei, yra naudojamas ciklas, per kurį kiekviename žingsnyje „fuzzifikuojami“ agentų renkami atstumo duomenys. „Fuzzifikuotiems“ duomenims yra pritaikomos sistemos taisyklės, pagal kurias išvedamos dvi skirtingų *„Fuzzy”* sistemų išvestys. Jų žodinės vertės yra „defuzzifikuojamos“ į greičio dedamąją agento judėjimo komandai. Po *„Fuzzy”* sistemos veikimo yra apskaičiuojamos lygiavimo, atskyrimo bei kohentiškumo komponentės, o jų suma sudaro visą agento judėjimo žingsnį.

Pridėjus judėjimo žingsnį yra apskaičiuojamos ir išsaugojamos naujos spiečiaus agentų koordinatės, pagal kurias yra tikrinama, ar visi agentai yra susibūrę prie naujojo lyderio per 50 metrų atstumą. Jeigu tai nėra įvykę, ciklas kartojasi, kol paskutinis spiečiaus agentas susibūriuoja minėtu atstumu.



27 pav. Ketvirtosios algoritmo fazės loginė seka

Ketvirtoje ir paskutinėje algoritmo fazėje spiečius skrenda į rastą objektą su tikslu susidurti su juo. (Žr 27 pav.) Šios fazės metu yra nustatoma susidūrimo sąlyga, kad susidūrimas įvyko, jei atstumas tarp objekto koordinačių ir agento koordinačių yra mažesnis arba lygus 0,5 metro. Taip pat, ketvirtoje algoritmo fazėje yra nustatoma, kad objektas juda tolygiu greičiu. Fazės pradžioje spiečiaus lyderis yra priskiriamas į spiečiaus agentų aibę, nes jo tikslas tampa toks pat, kaip ir visų likusių spiečiaus agentų. Nors spiečiaus agentų tikslas yra susidurti su judančiu objektu, yra būtina įvertinti ir sustabdyti agentų susidūrimą tarpusavyje. Ši algoritmo dalis yra itin svarbi spiečiui sekant objektą ir norint su juo susidurti, nes atstumai tarp agentų gali tapti itin maži. Dėl šios priežasties agentui nėra priskiriamas maksimalus galimas greitis, kadangi algoritmas gali nesuspėti reaguoti į greitus pozicijos pakitimus. Ketvirtoje fazėje *„Fuzzy”* sistemos nėra naudojamos keisti greičio komponentę, kadangi atstūmimo ir lygiavimo komponentės pilnai prižiuri atstumus tarp agentų, o agentų greitis palaikomas kuo tolygesnis, kad susidūrimas su objektų įvyktų kuo greičiau.

Toliau algoritmas pradeda naudoti ciklą, kurio pradžia yra objekto judėjimo įvertinimas ir jo pozicijos atnaujinimas kievieno ciklo pradžioje. Nustačius objekto poziciją yra skaičiuojami atstumai tarp

atskirų agentų ir objekto, į kuriuos yra atsižvelgiama, kai agentai palaiko tarpusavio atstumą ir juda link objekto tuo pačiu metu. Tada yra tikrinama, ar atskiras agentas yra nustatytu atstumu nuo objekto. Jeigu dronas patenka į atstumo ribą, jis užskaitomas kaip pataikęs į objektą ir yra išimamas iš tolimesnių skaičiavimų. Ciklas toliau tikrina, ar vis dar yra agentų, kurie nepataikė, ir atnaujina agentų poziciją kitam ciklo žingsniui. Ciklas pasibaigia, kuomet visi agentai susiduria su objektu. Tokiu atveju ketvirtoji fazė, spiečiaus misija bei algoritmas pasibaigia.

3.2. Bepiločių orlaivių spiečiaus algoritmo simuliacijos rezultatų parametrai ir jų vertinimas

Bepiločių orlaivių spiečiaus algoritmo simuliacija yra vykdoma *„MATLAB“* programoje. Visas *„MATLAB“* programos kodas yra rašomas pagal algoritmo logiką ir yra testuojamas keliais skirtingais scenarijais. Simuliacijoje yra testuojami du pagrindiniai scenarijai. Pirmąjame scenarijuje spiečius suranda objektą ir pilnai įgyvendina savo misiją, o antrąjame – spiečius savo paieškos spindulyje nesuranda objekto ir misija yra nutraukiama. Taip pat scenarijai gali būti varijuojami, kuomet yra keičiama objekto pozicija, judėjimo kryptis ar judėjimo greitis.

Simuliacijų rezultatai yra renkami pagal skirtingas algoritmo fazes. Pirmoje fazėje yra renkami tokie duomenys:
1. atliktų žingsnių kiekis ir realus fazės įvykdymo laikas;
2. mažesnių nei 1m atstumų tarp agentų ar lyderio kiekis ir žingsnis, kada tai įvyko;
3. vidutinis agento nuokrypis nuo jam nustatytos pozicijos;
4. minimalus atstumas tarp agentų ar lyderio laiko momentu;
5. maksimalus, minimalus bei vidutinis agentų greitis laiko momentu.

Antroje fazėje yra renkami tokie duomenys:
1. atliktų žingsnių skaičius ir realus fazės įvykdymo laikas;
2. spiečiaus apieškotas plotas;
3. maksimalus paieškos plotas;
4. apieškoto ploto santykis su maksimaliu plotu.

Trečioje fazėje renkami šie duomenys:
1. atliktų žingsnių kiekis ir realus fazės įvykdymo laikas;
2. mažesnių nei 1m atstumų tarp agentų ar lyderio kiekis ir žingsnis kada tai įvyko;
3. vidutinis agentų atstumas nuo lyderio pasibaigus fazei;
4. galutinis agentų atstumas nuo lyderio;
5. minimalus atstumas tarp agentų ar lyderio laiko momentu.

Ketvirtoje fazėje renkami šie duomenys:
1. pirmojo ir paskutinio agento susidūrimo žingsnis;
2. visų agentų susidūrimo laikas;
3. minimalus atstumas tarp agentų laiko momentu.

Jeigu yra atliekamas scenarijus, kuriame objektas nėra surandamas, duomenys renkami tik iš pirmųjų dviejų fazių. Šiose simuliacijose nėra įtraukiami Gauso trikdžiai ir nėra analizuojama jų poveikis spiečiui, tačiau per visas fazes yra renkamas spiečiaus susibūrimo vertinimo kriterijus E(t).

3.3. Bepiločių orlaivių spiečiaus algoritmo simuliacijos eiga ir rezultatai

Norint sužinoti, ar algoritmas yra veiklus ir efektyvus, yra leidžiama simuliacija *„MATLAB“* programoje. Simuliacijos pradžioje yra įrašomi pradiniai spiečiaus duomenys, t.y kokios yra pradinės spiečiaus agentų koordinatės, nustatomas *„MATLAB“* naudojamo žingsnio laiko tarpas, nustatomi pirmosios fazės formacijos puslankiai ir priskiriamos agentų pozicijos formacijoje pagal Vengrų algoritmą. Pradinės spiečiaus agentų koordinatės yra matomos 14 lentelėje.

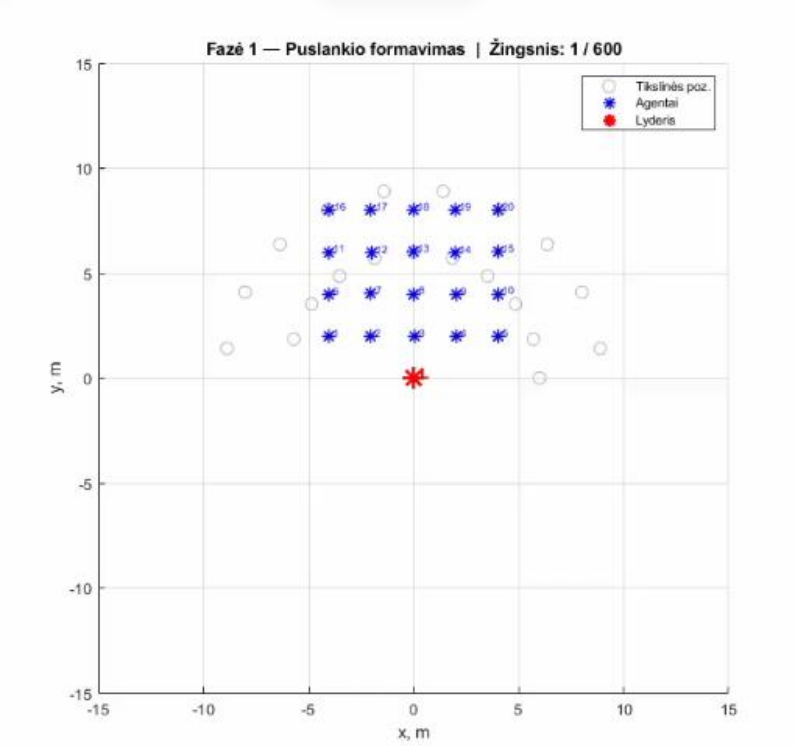
14 lentelė. Pradinės spiečiaus agentų koordinatės plokštumoje

Agento Nr.	**Agento koordinatės (X, Y)**	**Agento Nr.**	**Agento koordinatės (X, Y)**
1.	-4, 2	11.	-4, 6
2.	-2, 2	12.	-2, 6
3.	0, 2	13.	0, 6
4.	2, 2	14.	2, 6
5.	4, 2	15.	4, 6
6.	-4, 4	16.	-4, 8
7.	-2, 4	17.	-2, 8
8.	0, 4	18.	0, 8
9.	2, 4	19.	2, 8
10.	4, 4	20.	4, 8

14 lentelėje matoma, kad spiečius pradžioje yra išsidėstęs 4 eilėmis ir 5 stulpeliais, nes realybėje taip išdėstyti agentus yra pakankamai paprasta ir tarp jų yra saugus pradinis 2 metrų atstumas. Pradinė lyderio pozicija spiečiuje yra [0,0]. *„MATLAB“* programoje naudojamas žingsnis lygus 0,4 s.

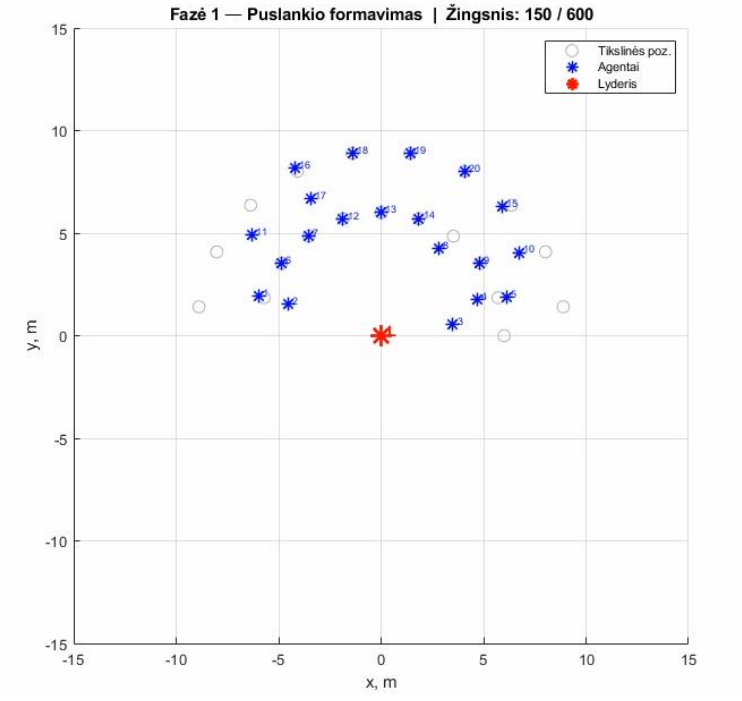
3.3.1. Spiečiaus algoritmo rezultatai, kuomet objektas nėra randamas

Pirmojo algoritmo scenarijuje yra nagrinėjama situacija, kuomet spiečius antroje fazėje neranda objekto ir po pilno išsisklaidymo grįžta į pirmosios fazės gale užfiksuotą formaciją. 29 paveiksle yra rodomas pradinis paleistas spiečius ir jo algoritmas.



28 pav. Spiečiaus algoritmo vaizdas pradiniame žingsnyje pirmoje fazėje

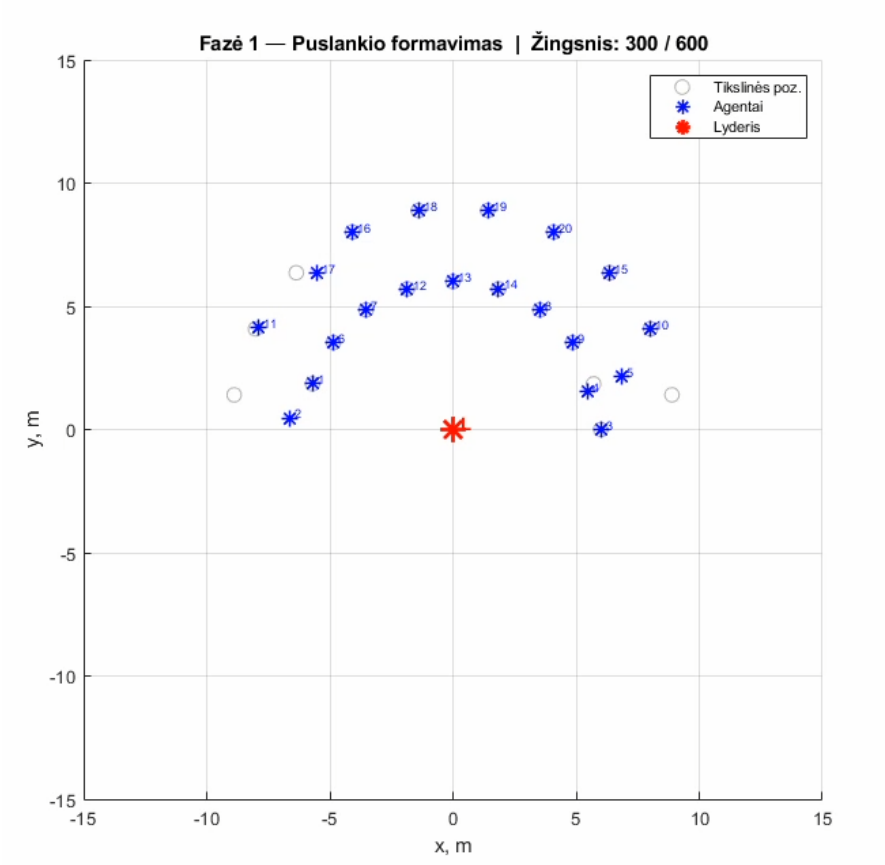
28 paveiksle matomi 20 agentų (mėlynų žvaigždučių) ir 1 spiečiaus lyderis pažymėtas raudona žvaigždute. Baltais tuščiaviduriais apskritimais yra pažymėtos dviejų puslankių vietos, į kurias turi sustoti spiečiaus agentai. Spiečiaus agentai išsidėstę pilnu keturkampiu kiekvieno žingsnio metu juda į Vengrų algoritmo priskirtą poziciją. 30 paveiksle pateikiamas vaizdas po 150 žingsnių.



29 pav. Spiečiaus algoritmo vaizdas po 150 žingsnių pirmoje fazėje

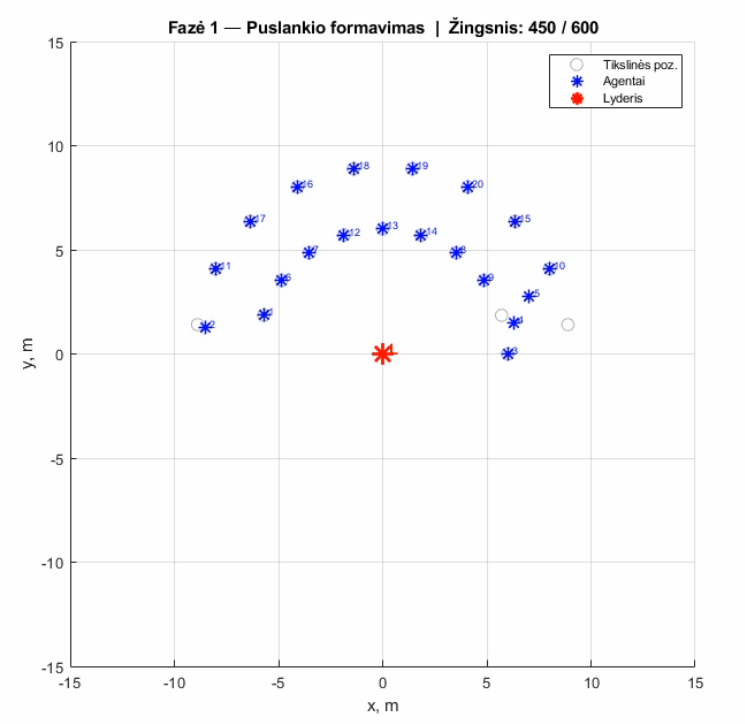
29 paveiksle galim matyti, jog po 150 žingsnių 11 iš 20 agentų jau yra užėmę priskirtas pozicijas. 7 iš agentų yra išsidėstę nedideliu atstumu nuo savo pozicijos, o likusiems 2 agentams, kurių numeriai yra 2 ir 4, iškilo problemų judėti į savąją poziciją dėl priešais esančių agentų (5 ir 1 agentai). Priešais esantys agentai nori užimti savo poziciją ir tokiu atveju stumia šalia esančius dronus nuo savęs, taip yra trukdoma 2 ir 4 agentams, norintiems užimti už 5 ir 1 agento esančias pozicijas.

Po 300 žingsnių yra matomas tolimesnis spiečiaus išsidėstymas (žr. 30 pav.).



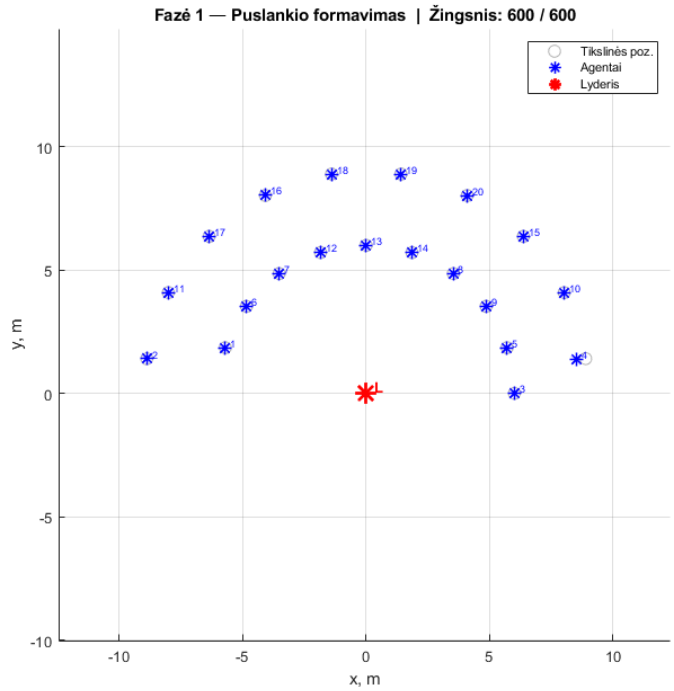
30 pav. Spiečiaus algoritmo vaizdas po 300 žingsnių pirmoje fazėje

30 paveiksle galima matyti, kad 16 iš 20 agentų jau yra užėmę tinkamas pozicijas. Taip pat matoma, kad 2 agentas saugiu atstumu apskrido 1 agentą per apačią, kad galėtų pasiekti savo poziciją, o tai pirmąjam agentui leido užimti savo poziciją. Dešinėje spiečiaus pusėje problema vis dar neišspręsta, bet 5 agentas yra nuskridęs gana toli nuo savo pozicijos, stengdamasis praleisti 4 agentą į jo vietą.



31 pav. Spiečiaus algoritmo vaizdas po 450 žingsnių pirmoje fazėje

31 paveiksle matomas vaizdas po 450 žingsnių. 17 iš 20 agentų yra pilnai užėmę savo vietas. 2 agentas yra labai arti savos pozicijos, o 4 agentas turi tiesų kelią link norimos vietos. 5 Agentas pilnai atvėrė kelią laikydamasis saugaus atstumo ir nuo 10 agento, bei praleidęs 4 agentą grįš atgal į savo vietą.



32 pav. Spiečiaus algoritmo vaizdas po 600 žingsnių pirmoje fazėje

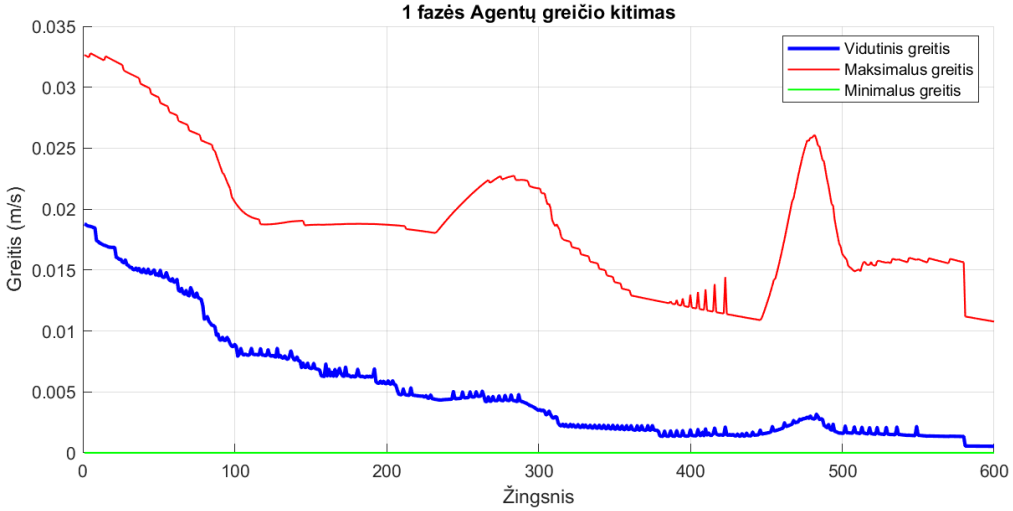
32 paveiksle matoma, kokią formaciją sudaro bepiločių orlaivių spiečius pasibaigus pirmąjai algoritmo fazei. 19 iš 20 agentų spėjo tiksliai užimti savo pozicijas, tačiau 4 agentas nespėjo tiksliai užimti savo pozicijos iki ciklo pabaigos. 4 agento atstumas nuo jo pozicijos yra 0.4 m. Tai didelės įtakos kitai algoritmo fazei neturi, kadangi kampas nuo lyderio yra tinkamas. Vidutinis agentų nuokrypis nuo jų pozicijos yra 0,02 m, nes visi likę agentai tiksliai užėmė savo vietas. Šis spiečiaus manevras buvo atliktas per 240 sekundžių, o mažiausi atstumai tarp agentų pirmos fazės metu yra matomi 15 lentelėje.

15 lentelė. Visų agentų mažiausi atstumai nuo kito agento laiko momente

Agento Nr.	Mažiausias atstumas nuo agento, m	Žingsnio Nr.	Agento Nr.	Mažiausias atstumas nuo agento, m	Žingsnio Nr.
1.	1,49	118	11.	1,86	102
2.	1,49	118	12.	1,67	27
3.	1,54	458	13.	1,87	47
4.	1,44	125	14.	1,62	72
5.	1,44	125	15.	1,98	25
6.	1,73	271	16.	1,61	53
7.	1,67	27	17.	1,61	53
8.	1,62	72	18.	2	1
9.	1,75	514	19.	2	1
10.	1,66	454	20.	1,98	25

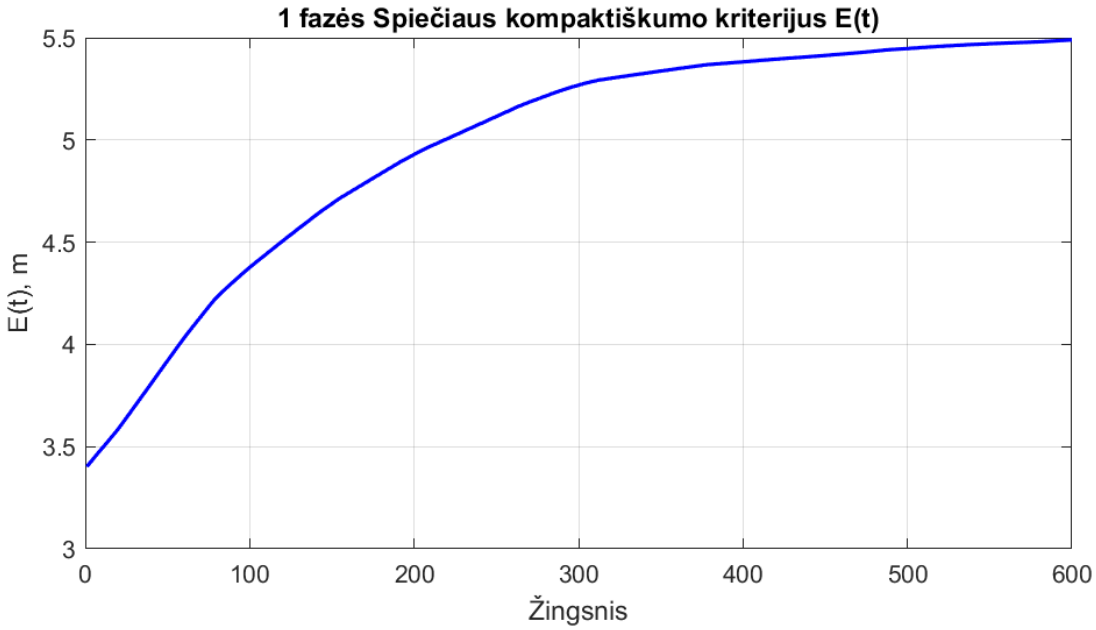
15 lentelėje matoma, kad mažiausias atstumas tarp agentų buvo 1,44 m. Jis buvo užfiksuotas tarp 4 ir 5 spiečiaus agentų 125 algoritmo žingsnyje. Šis rezultatas yra tinkamas bepiločių orlaivių spiečiaus formavimui, kadangi nei vienas agentas nepasiekė pavojingo mažesnio nei 1 metro atstumo tarp agentų.

Pirmoje algoritmo fazėje taip pat yra braižomas spiečiaus agentų maksimalaus, vidutinio bei minimalaus greičio grafikas, kuris parodo kaip agentų greitis kito per įvykusius 600 žingsnių (žr 33 pav.)



33 pav. Pirmosios algoritmo fazės agentų greičio kitimo per fazės žingsnius grafikas

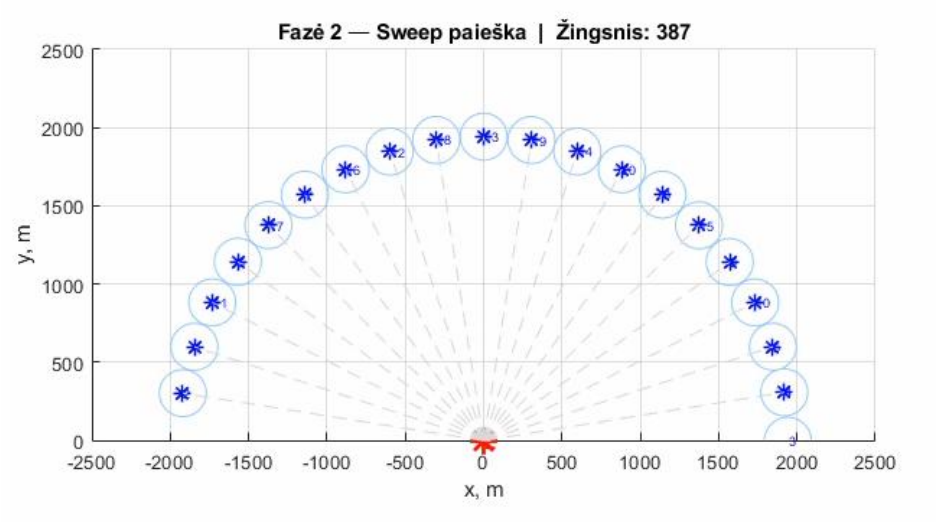
33 paveiksle nurodoma kiekvieno žingsnio maksimali greičio vertė (raudona linija), visų spiečiaus agentų vidutinis greitis (mėlyna linija) bei minimali greičio vertė (žalia linija). Iš grafiko galima teigti, kad maksimalus agento greitis laiko momente buvo 0,032 m/s, o visos fazės metu bent vienas spiečiaus agentas visiškai nejudėjo. Pirmosios fazės eigoje yra matomas pastovus vidutinio greičio mažėjimas, kadangi vis daugiau agentų atskrenda į jam nustatytą poziciją ir jis likusiuose fazės žingsniuose nebejuda. Nuo 440 žingsnio matomas staigus maksimalaus greičio šuolis dėl į paskutinę poziciją prasiveržusio agento greičio, tačiau beveik visi dronai jau buvo pasiekę savo vietas ir šis greičio šuolis vidutiniam spiečiaus greičiui didelės įtakos neturėjo.



34 pav. Pirmosios algoritmo fazės spiečiaus kompaktiškumo kriterijaus per fazės žingsnius grafikas

34 paveiksle pateiktame grafike matoma, kad spiečiaus kompaktiškumas laikui bėgant mažėja nuo 3,4 iki 5,5 m. Kadangi pradinis spiečiaus judėjimas yra naudojamas užimti platesnes pozicijas ruošiantis kitos fazės paieškai, kompatiškumo kriterijaus augimas yra priimtinas.

Antroje algoritmo fazėje agentai pajuda iš prieš tai pasiektų pozicijų. Agento greitis yra 5 m/žingsnis, o realus agento greitis lygus 12,5 m/s. Agentų kampai nuo lyderio kinta kas 9° laipsnius [0°, 171°] rėžiuose. Kadangi šiame scenarijuje spiečius neaptinka objekto, maksimalus paieškos atstumas yra pasiekiamas per 387 žingsnius arba 155 sekundes. 35 paveiksle matomas maksimalios paieškos spiečiaus išsidėstymas.



35 pav. Spiečiaus algoritmo vaizdas po 387 žingsnių antroje fazėje

35 paveiksle matomi simuliacijos rezultatai rodo, jog spiečiaus apieškotas plotas yra lygus 5936288 m² arba 5,94 km². Maksimalus paieškos plotas turint omenyje, kad maksimalus atstumas tarp agentų lygus 300 m, yra 5170944 m² arba 5,17 km². Pagal gautus rezultatus paieškos efektyvumas yra 114,8%. Šis nesutapimas gaunamas, kadangi algoritmas neįvertina ir neatima ploto, kurį vienu metu padengia keli agentai. Iš vienos pusės gaunami paieškos ploto rezultatai nėra pilnai tikslūs, tačiau spiečiaus paieška vyksta su redundantiškumu. Šiuo atveju plotų persidengimas yra algoritmo ypatumas, kuris užtikrina paieškos patikimumą naudojant mažiau apibrėžtas matematines sistemas, tokias kaip *„Fuzzy“* modelis. Vykdoma paieška tampa žymiai agresyvesnė, nepalikdama aklųjų zonų. Didžioji paklaidos dalis yra sukaupiama fazės pradžioje, kuomet agentai vis dar yra sąlyginai arti vienas kito.

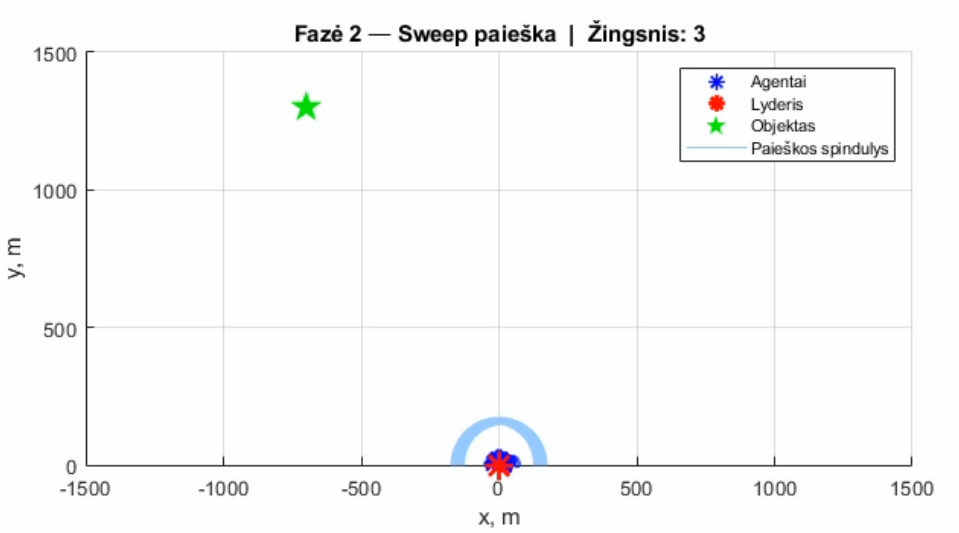
Likusioje algoritmo dalyje spiečius per tą patį laiką ta pačia trajektorija grįžta į pirmoje fazėje užfiksuotą formaciją ir misija yra baigiama.

Pagal įgyvendinto spiečiaus algoritmo scenarijų galima teigti, kad naudojant šį algoritmą spiečius susiformuoja į norimą formaciją ir atlieka 5,17 km² paiešką per 395 sekundes. Per šį laikotarpį, spiečiaus agentai nei karto nesusiduria su šalia esančiais kaimynais, o mažiausias atstumas tarp jų yra 1,44 metro.

3.3.2. Spiečiaus algoritmo rezultatai, kuomet objektas randamas

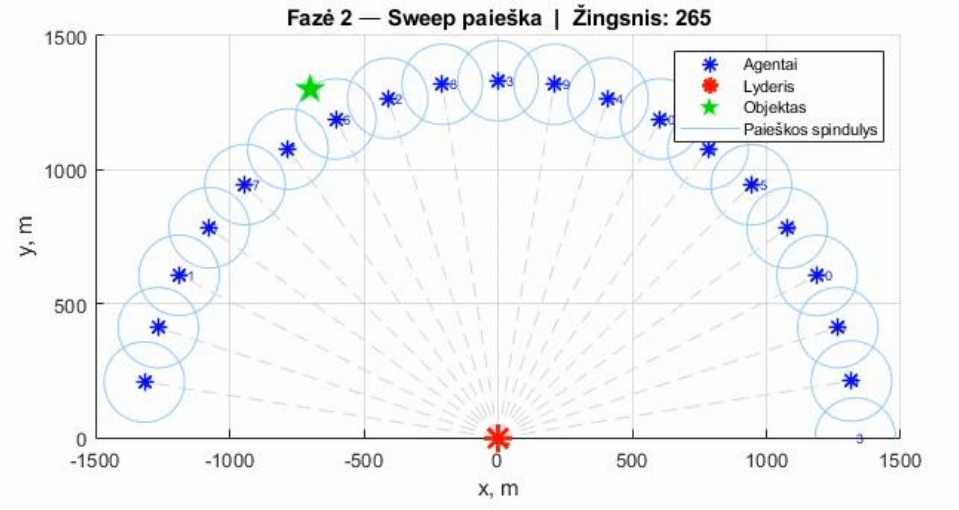
Antrasis scenarijus, kuris yra tikrinamas su spiečiaus algoritmu, yra kuomet antroje fazėje algoritmas randa objektą ir toliau tęsia trečią ir ketvirtą fazes. Šiuo atvėju *„MATLAB“* programoje yra įrašomos objekto koordinatės [-700, 1300]. Pirmoji algoritmo fazė išlieka tokia pati, kaip ir scenarijuje, kai objektas nerandamas. Spiečius juda taip pat, kas nesudaro pavojingų atstumų tarp agentų bei lyderio. Vidutinis nuokrypis nuo pozicijos išlieka 0,02 m dėl vieno agento netikslios pozicijos.

Antroje fazėje judėjimo greitis ir agentų trajektorijos nuo lyderio yra tokios pačios, kaip ir pirmame scenarijuje. Tačiau atsiradus objektui, spiečius turi jį aptikti apieškant mažesnį plotą, nei maksimalus. Spiečiaus pirmieji žingsniai su agento atsiradimu koordinačių sistemoje pavaizduoti 36 paveiksle, kuomet žalia žvaigžde yra žymima objekto, kurį turi aptikti spiečius, pozicija.



36 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 3 žingsnių antroje fazėje

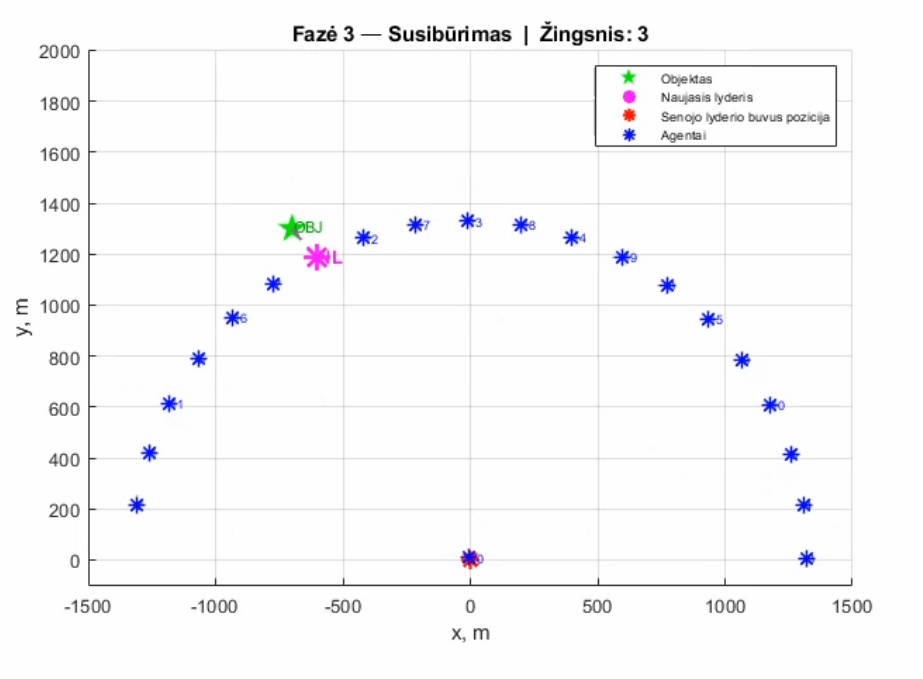
Antroji fazė yra baigiama, kai vienas agentas savo 150 metrų aptikimo spindulyje sutinka objektą. Šiuo atveju spiečius atliko 265 žingsnius arba užtruko 105 sekundes. Objektą surado 16 agentas, kuris nuo pradinio taško nuskrido apie 1325 metrus. Šiuo atveju realus paieškos plotas yra 2,8 km² , o paieškos ploto santykis su maksimalia paieška – 54,1%. Antrosios fazės rezultatai rodo, jog paieška yra sėkminga ir realiomis salygomis yra atliekama gana greitai. Taip pat yra įvertinamas E koeficientas, kuris šioje fazėje lygus 966. Tai parodo, jog spiečius yra labai plačiai išsidėstęs, bet šioje fazėje toks rezultatas ir norimas. Toliau tęsiant algoritmą 16 agentas tampa naujuoju spiečiaus lyderiu, kuris suburs likusius agentus trečioje fazėje. Spiečiaus vaizdas galutinėje antros fazės pozicijoje matomas 37 paveiksle.



37 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 265 žingsnių antroje fazėje

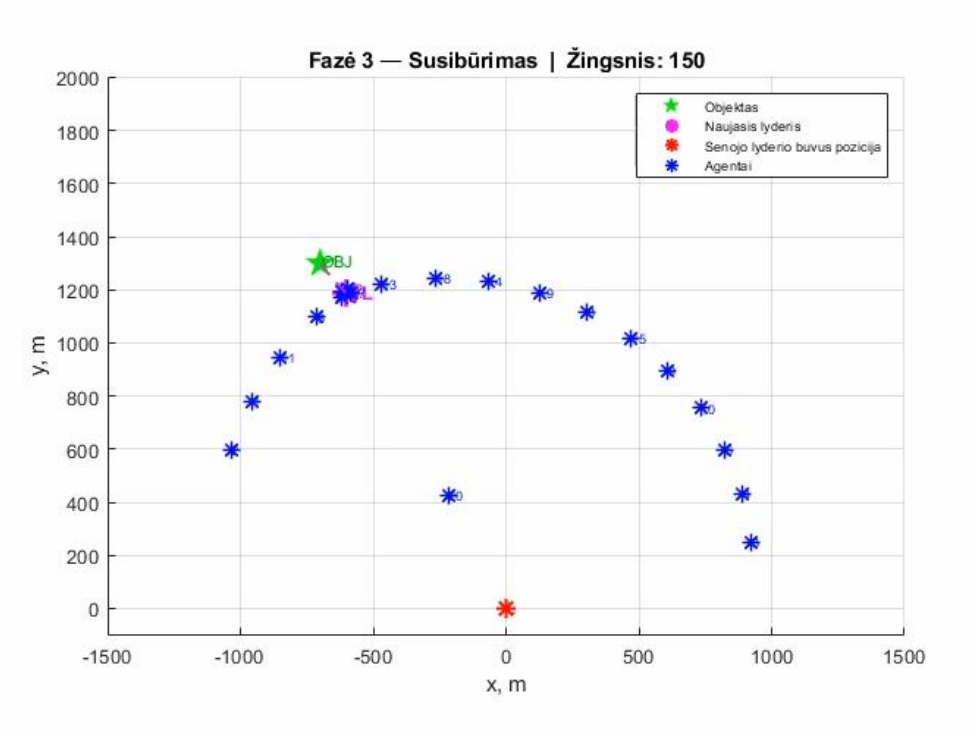
Trečiosios fazės pradžioje, naujajam spiečiaus lyderiui yra užduodama misija išlaikyti objektą savo paieškos spindulyje. Ši funkcija yra duodama tam, kad lyderis nepamestų surasto objekto, jeigu

objektas būtų judantis. Taip pat, prie spiečiaus susibūrimo prisijungia ir senasis spiečiaus lyderis, nes jis irgi yra naudojamas ketvirtoje atakos fazėje. Kadangi spiečius šioje fazėje nesibūriuoja itin glaudžiai, agentams yra nustatomas minimalus greitis = 2 m/žingsnis, o maksimalus – 8 m/žinsgnis arba atitinkamai 5 m/s bei 20 m/s. Šie greičiai paspartina agentų susibūrimo greitį, net jei *„Fuzzy“* sistema priskirtų mažesnį greitį. Pradinis fazės vaizdas matomas 38 paveiksle.



38 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 3 žingsnių trečioje fazėje

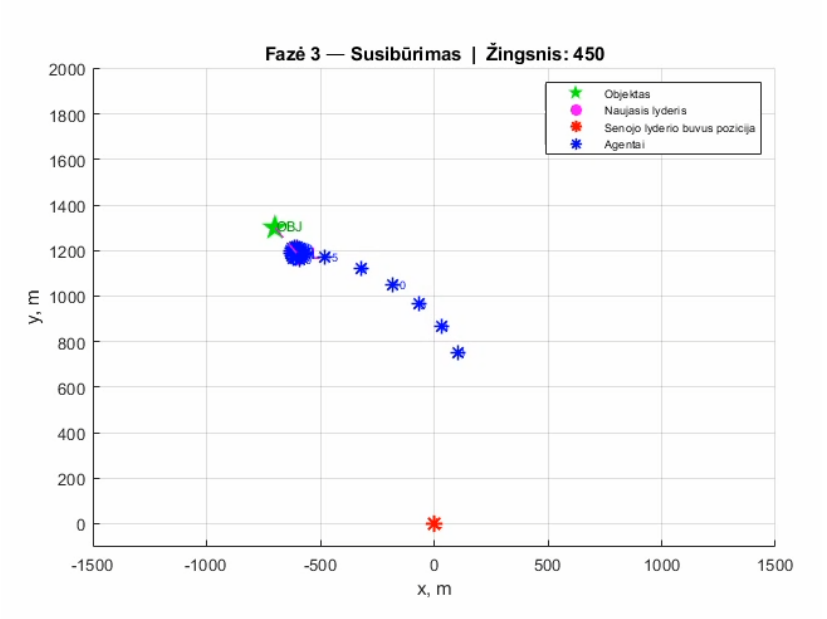
Bėgant algoritmo žingsniams, galima matyti spiečiaus judėjimą link naujojo lyderio. Po 150 žingsnių (žr. 39 pav.**)** matoma, jog 16 spiečiaus agentų nėra pasiekę norimo 50 metrų atstumo iki lyderio, tačiau stabiliu greičiu juda link jo.



39 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 150 žingsnių trečioje fazėje

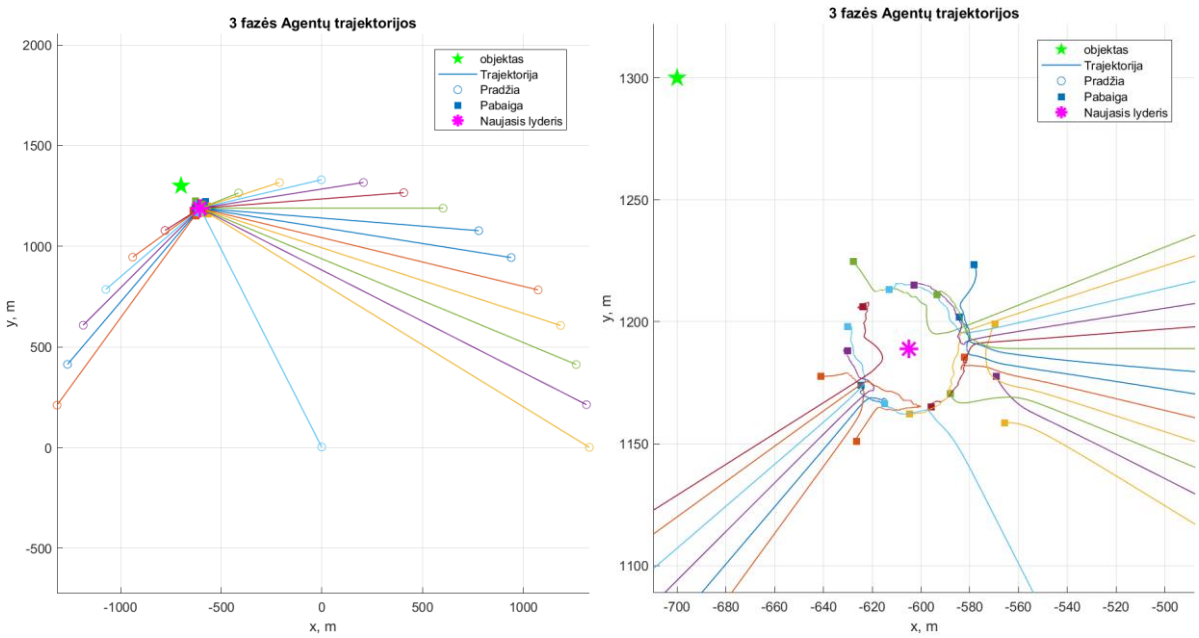
Taip pat galima matyti, jog buvęs spiečiaus lyderis be jokių trikdžių skrenda link naujo lyderio pozicijos. Per pirmus 150 žingsnių nebuvo užfiksuotas nei vienas pavojingas atstumas tarp spiečiaus agentų bei lyderio.

Po 450 žingsnių matoma, jog 14 iš 20 agentų jau yra susispietę aplink lyderį, o tolimiausiuose atstumuose buvę 6 agentai vis dar skrenda link lyderio. Per šį laikotarpį taip pat nebuvo užfiksuotas mažesnis nei metro atstumas tarp agentų. (Žr. 40 pav.)



40 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 450 žingsnių trečioje fazėje

Trečiojoje fazėje algoritmas yra užbaigiamas su 722 žingsniu, kuomet aplink lyderį visi agentai susirenka bent 50 metrų atstumu. Galutinis trečios fazės vaizdas matomas 41 paveiksle.



41 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 722 žingsnių trečioje fazėje

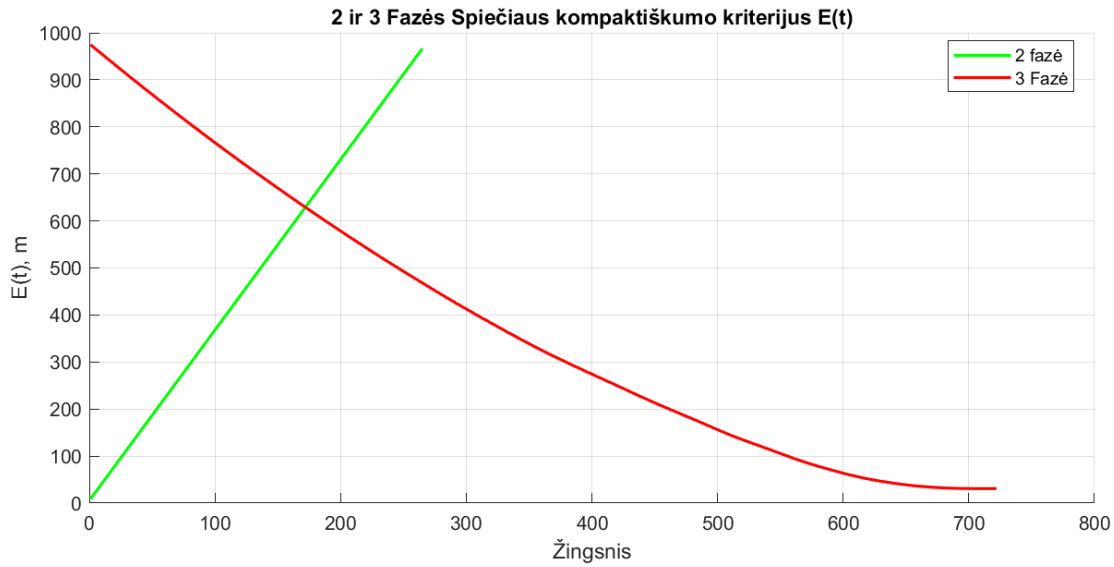
41 paveiksle matoma, jog spiečiaus formacija nėra tvarkinga, t.y. agentai yra susispietę aplink lyderį taip, kad jie būtų reikiamame atstume nuo jo. Kadangi galutinėje algoritmo fazėje nėra reikiamybės pradėti ją iš tam tikros pozicijos, spiečiaus trajektorijos didžiąją fazės dalį yra tiesės iki lyderio, tačiau priartėjus prie lyderio yra matomas netvarkingas judėjimas. Tai nulemia galutinės agento pozicijos nenustatymas. Netvarkingas judėjimas taip pat parodo, kad agentai stabiliai reguliuoja tarpusavio atstumą ir leidžia susibūriuoti vėliau prie lyderio atskridusius dronus. Trečioje fazėje spiečius susirinko per 289 sekundės, neturėdamas nei vienos susidūrimo situacijos. Galutinis vidutinis atstumas nuo lyderio yra 30,96 m 16 lentelėje yra pateikiami kiekvieno agento galutiniai atstumai nuo lyderio, artimiausi atstumai nuo kito agento bei žingsnis, kada tai ivyko.

16 lentelė. Visų agentų galutiniai atstumai nuo lyderio bei mažiausi atstumai nuo kito agento laiko momente

Agento Nr.	Galutinis atstumas nuo lyderio, m	Mažiausias atstumas nuo agento, m	Žingsnio Nr.	Agento Nr.	Galutinis atstumas nuo lyderio, m	Mažiausias atstumas nuo agento, m	Žingsnio Nr.
1.	24,8	6,9	330	11.	25,4	6,9	330
2.	37,6	9,4	674	12.	42,3	9,6	667
3.	49,9	19	714	13.	25,4	9,2	685
4.	37,6	15,2	722	14.	25,6	6,8	329
5.	25,2	8,8	715	15.	24,2	9,4	659
6.	26,8	9,2	697	16.	43,7	9,3	660
7.	25,4	9,3	686	17.	26,7	9,2	680
8.	43,3	9,4	659	18.	26	6,8	329
9.	23	10,1	651	19.	25	9,1	675
10.	36,6	10,1	651	20.	24,9	9,1	680

16 lentelėje matoma, kad po trečiosios fazės artimiausias atstumas iki lyderio buvo pasiektas 9 agento, kuris yra per 23 metrus nuo lyderio. Tai yra tinkamas susibūriavimo atstumas. Mažiausias atstumas nuo kito agento buvo užfiksuotas 329 žingsnyje, kuriame 14 ir 18 agentai buvo per 6,8 m atstumą. Šis atstumas taip pat yra visiškai saugus.

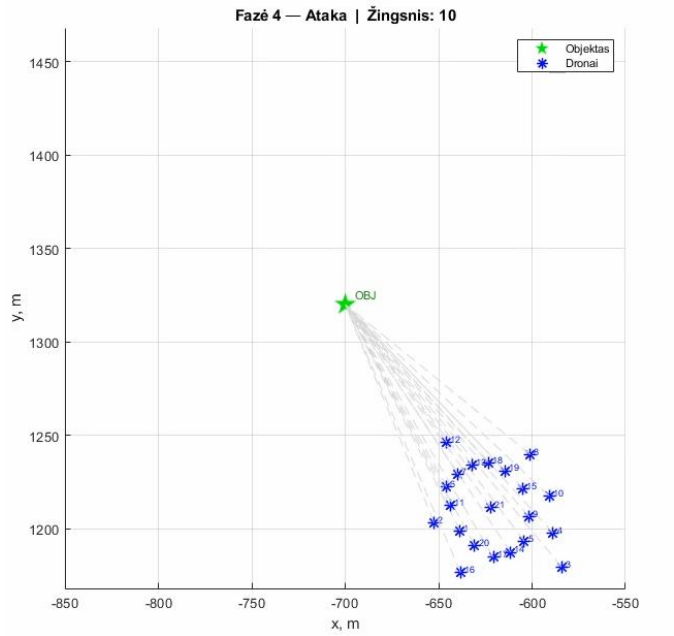
Po antrosios ir trečiosios fazės yra brėžiamas E(t) kompaktiškumo grafikas abiejoms fazėms.



42 pav. Antrosios bei trečiosio algoritmo fazės spiečiaus kompaktiškumo kriterijaus per fazės žingsnius grafikas

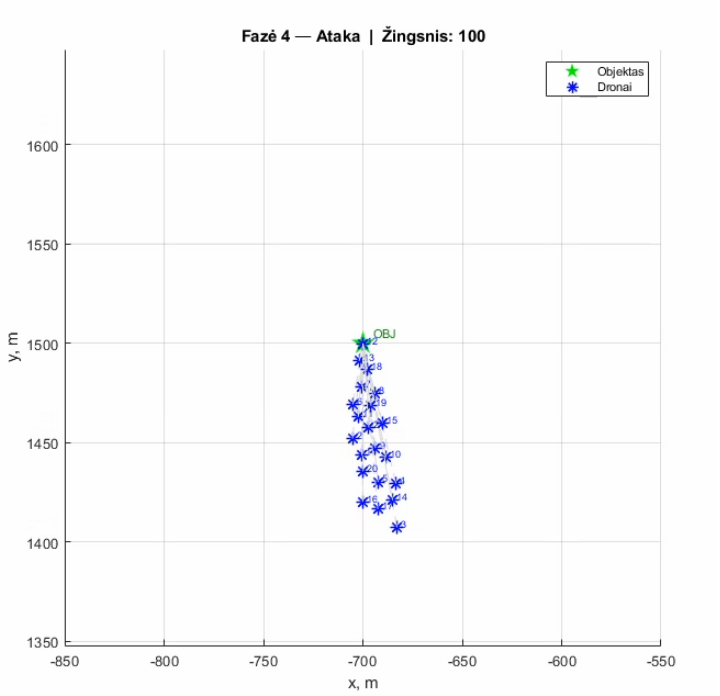
42 paveiksle pavaizduotame grafike matoma, kad antroje fazėje spiečiaus kompaktiškumas tolygiai mažėja per visos fazės laikotarpį. Nuo pirmoje fazėje turėtos 5,4 m reikšmės per 265 žingsnius, kompaktiškumas išauga iki 966,5 m Tai yra įprasta, kadangi spiečius atlieka paieškos misiją, kurios tikslas yra plačiai apieškoti erdvę. Prasidėjus trečiąjai fazei galima matyti, kad kompaktiškumo kriterijaus reikšmė tolygiai mažėja iki 500 žingsnio. Likusius fazės žingsnius kompaktiškumo reikšmės mažėjimas pradeda lėtėti, nes dauguma spiečiaus agentų jau būna susibūriavę prie naujojo lyderio ir toliau nesispaudžia. Per 722 žingsnius kompaktiškumas didėja nuo 970 m iki 31 m Šie grafikai parodo, kad norimomis algoritmo fazėmis spiečius efektyviai atlieka išsisklaidymo bei susibūriavimo funkcijas.

Ketvirtoje ir paskutinėje algoritmo fazėje spiečius juda link objekto su susidūrimo tikslu. Pagrindiniai fazės parametrai yra agentų greitis, lygus 7 m/žingsnis arba 17,5 m/s, objekto judėjimo greitis yra 5 m/s, o jo judėjimo kryptis koordinačių sistemoje yra vertikali į viršų. Taip pat, spiečiaus lyderis yra priskiriamas prie likusių agentų spiečiaus, kadangi jis taip pat atakuos objektą. Šioje algoritmo fazėje pavojingas atstumas tarp agentų laikomas 0,4 m, kadangi spiečius konverguoja į vieną judantį tašką. Tai nulemia labai glaudų spiečiaus skrydį, bet 0,4 metro riba vis dar laikoma saugia.



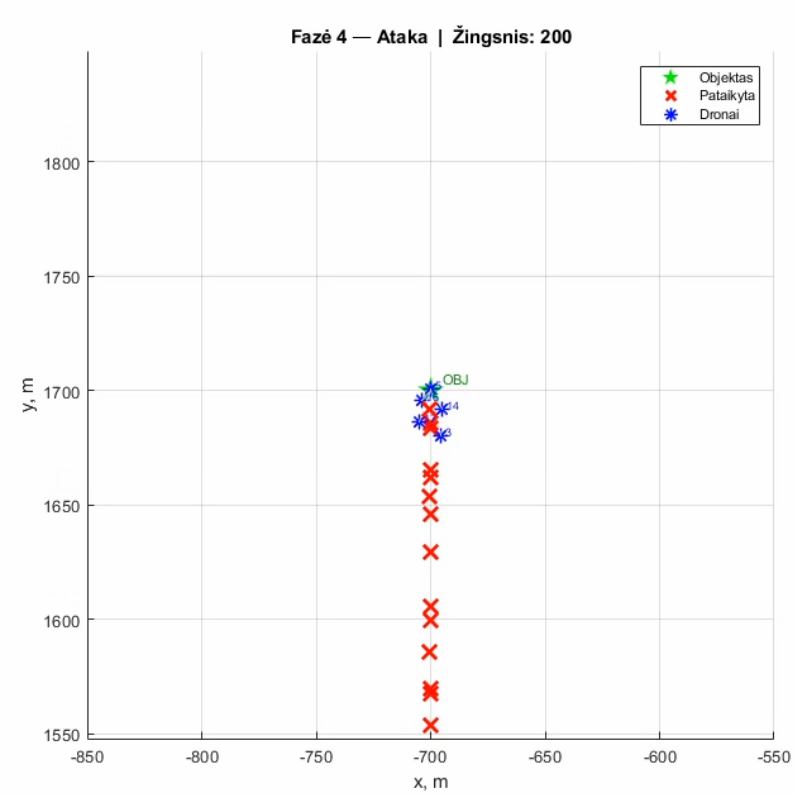
43 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 10 žingsnių ketvirtoje fazėje

43 paveiksle matoma, kad dronų spiečius po 10 žingsnių vis dar yra panašioje formacijoje, kurioje užbaigė trečiąją fazę, tačiau tarpusavio atstumai yra sumažėję. Po 100 žingsnių spiečiaus agentai yra žymiai arčiau objekto, tačiau dar nėra nei vieno agento kuris susidūrė su objektu (žr. 44 pav.).



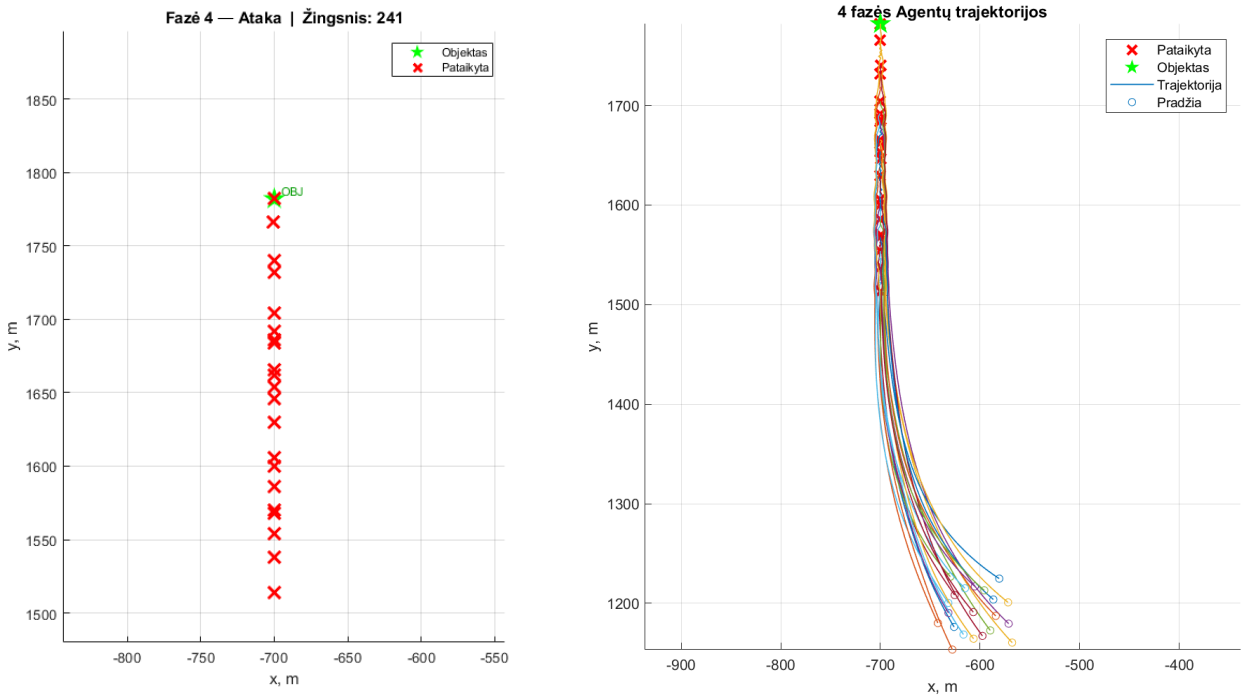
44 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 100 žingsnių ketvirtoje fazėje

44 paveiksle matoma, kad agentai laikosi saugaus tarpusavio atstumo sekant objektą trijomis eilėmis. Tolimesniuose žingsniuose agentai po truputį lengviau spaudžiasi ir juda į objektą dėl mažesnio agentų kiekio.



45 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 200 žingsnių ketvirtoje fazėje

Po 200 ketvirtos fazės žingsnių matoma, jog į objektą dar nepataikė 5 iš 21 agento tačiau agentai labai arti seka judantį objektą (žr. 45 pav.). Paskutinis spiečiaus agentas į objektą pataiko 241 žingsnyje. Tai reiškia, kad ketvirta fazė įvyko per 96,4 sekundes.



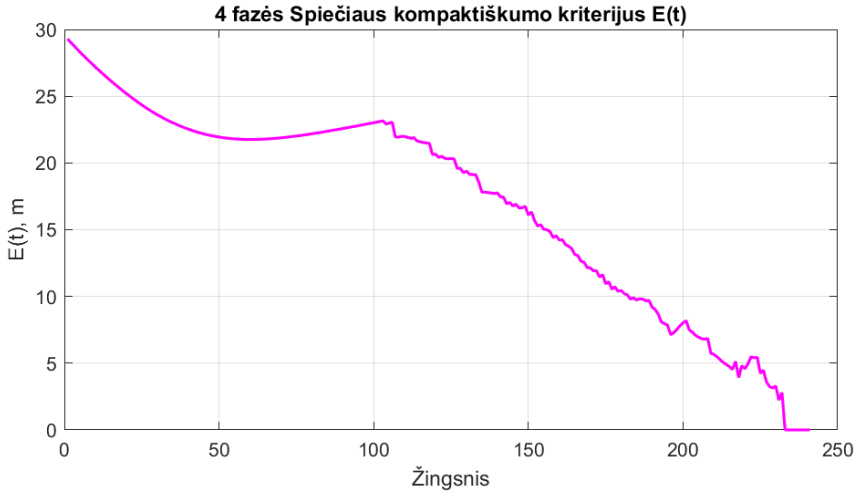
46 pav. Antrojo scenarijaus spiečiaus algoritmo vaizdas po 241 žingsnių ketvirtoje fazėje

Galutiniai fazės rezultatai parodo, kad pirmasis pataikymas įvyko po 107 žingsnių arba 42,8 sekundės, o objektas per šį laiką pajudėjo 482 metrus (žr. 46 pav.). Iš spiečiaus trajektorijų galima matyti, kad spiečius stabiliai konverguojasi link tuometinės objekto pozicijos ir su kiekvienu agento pataikymu spiečius vis glaudžiau seka objektą. Spiečius išvengė tarpusavio susidūrimų, tačiau artimiausias atstumas tarp agentų buvo 180 žingsnyje, kuomet 2 ir 10 agentas buvo per 0,4 metro. Šis atstumas yra ant pavojingumo ribos, kurioje pradeda didėti agentų susidūrimo tikimybė. Visų agentų susidūrimo su objektu laiko momentus ir artimiausius atstumus galima matyti 17 lentelėje.

17 lentelė. Visų agentų susidūrimo su objektu žingsniai bei mažiausi atstumai nuo kito agento laiko momente

Agento Nr.	Susidūrimo su objektu, žingsnis	Mažiausias atstumas nuo agento, m	Žingsnio Nr.	Agento Nr.	Susidūrimo su objektu, žingsnis	Mažiausias atstumas nuo agento, m	Žingsnio Nr.
1.	192	0,82	191	12.	107	2,94	104
2.	181	0,4	180	13.	127	0,67	126
3.	241	4,47	231	14.	220	0,73	215
4.	193	0,82	191	15.	177	0,47	173
5.	202	4,81	177	16.	216	0,73	215
6.	153	0,58	150	17.	233	3,6	218
7.	135	0,43	133	18.	119	0,78	114
8.	134	0,43	133	19.	143	1,64	142
9.	173	0,49	172	20.	196	1,73	190
10.	183	0,4	180	21.	165	0,75	162
11	150	0,8	148	-			

Iš surinktų duomenų matoma, jog susidūrimo momentų išsidėstymas yra gana tolygiai išsisklaidęs laike, kadangi agentai iki pat susidūrimo su objektu stengiasi išlaikyti saugų tarpusavio atstumą. Susidūręs dronas yra išimamas iš skaičiavimų, o tai užleidžia vietą kitam spiečiaus agentui lengviau atakuoti objektą. Mažesniais nei 0,5 metro atstumais spiečiaus agentai skrido 6 kartus. Tai parodo, kad spiečius prioritizuoja tarpusavio atstumų palaikymą, o ne greitesnį susidūrimą su objektu.



47 pav. Ketvirtosios algoritmo fazės spiečiaus kompaktiškumo kriterijaus per fazės žingsnius grafikas

Ketvirtojoje fazėje yra matomas spiečiaus kompaktiškumo augimas (žr. 47 pav.). Fazės pradžioje spiečiaus kompaktiškumas padidėja, tačiau augimas stabilizuojasi apie 22 metrus iki 100 žingsnio. Kuomet su objektu pradeda susidūrinėti pirmieji agentai, spiečiaus kompaktiškus pradeda tolygiai augti, nes susidūrę agentai nebeužima erdvės ir leidžia likusiems agentams spaustis ir artėti link objekto. Atlikus 200 žingsnių, kuomet 16 iš 21 agento jau yra pataikę, kompaktiškumas yra 8 m. Fazės pabaigoje spiečiuje beveik nebelieka agentų, o tai nulemia, kad kompaktiškumo kriterijus išauga iki 2,8 m. Šioje fazėje kompaktiškumo kriterijus parodo, kad norint pataikyti į objektą, spiečius efektyviai glaudžiasi tarpusavyje ir stabiliai mažina atstumus, kuomet erdvėje atsiranda daugiau vietos.

Apibendrinus pilnus antrojo scenarijaus simuliacijos rezultatus galima teigti, kad spiečius efektyviai įvykdo savo misiją. Visos misijos įvykdymas užtruko 730 sekundžių arba šiek tiek daugiau nei 12 minučių. Spiečiaus agentams, kurie skraido arti savo maksimalaus greičio, šis laikas yra pakankamas atsižvelgiant į standartinių bepiločių orlaivių akumuliatorius. Spiečiaus judėjimo koordinavimui pasirinktas *„Fuzzy“* matematinis modelis sėkmingai suvaldė misijos neapibrėžtumus, kurie gali kilti dėl agentų tarpusavio sąveikos bei ribotos komunikacijos tarp jų. Algoritmui veikiant ir su dideliu kiekiu agentų, sistema išlaikė stabilumą be tikslaus aplinkos modelio. Spiečiaus valdymo sistema, kuri naudoja atskyrimo, lygiavimo bei kohentiškumo komponentes, užtikrino efektyvų bei stabilų spiečiaus judėjimą per visas misijos fazes, kurių metu buvo eliminuota susidūrimų rizika. Per visą simuliaciją nebuvo užfiksuotas nei vienas agentų susidūrimas, o mažiausi atstumai per neapibrėžto judėjimo fazes buvo 1,44 m pirmoje fazėje, 6,8 m trečioje fazėje bei 0,4 m ketvirtoje fazėje. Pirmoje fazėje naudotas Vengrų algoritmas optimaliai priskyrė spiečiaus agentus į norimos formacijos pozicijas, kadangi neįvyko pozicijų nepasidalinimas ir per nustatytą fazės laiką spiečius labai tiksliai įgyvendino norimą dviejų puslankių formaciją. Be to, optimalus priskyrimas taupo spiečiaus energiją ir mažina jos sąnaudas. Vertinant pirmosios fazės spiečiaus greičio grafiką, matoma, jog *„Fuzzy“* sistemoje nustatyti maksimalūs greičiai nėra peržengiami, o laikui bėgant bei agentams atskrendant į jiems priskirtas vietas, vidutinis spiečiaus greitis stabiliai mažėja, kadangi daugumai agentų nebereikia didelių pozicijos pakeitimų.

Svarbus algoritmo efektyvumo įrodymas yra, jog šio algoritmo valdomas spiečius per 155 sekundes gali išieškoti 5,2 km² plotą su 15% persidengimu, o suradęs objektą per mažiau nei 289 sekundes, visi spiečiaus agentai susibūriuoja prie naujojo lyderio vidutiniu 31 metro atstumu nuo jo. Tai parodo, kad algoritmas sugeba efektyviai sukoordinuoti 20 agentų net ir tada,, kai agentai yra itin išsisklaidę. Taip pat, decentralizuota algoritmo struktūra išlaikė pilną algoritmo funkcionalumą kuomet pasikeitė spiečiaus lyderis. Automatiškai pasikeitus lyderiui, visa sistema be centrinio valdymo persiorientavo į tolimesnę misijos fazę, o tai parodo visos sistemos atsparumą trikdžiams bei įvairiems gedimams. Sistemos efektyvumą taip pat parodo ir kievienoje fazėje kintantis kompaktiškumo kriterijus E(t), kadangi pagal jį yra matomas norimas būriavimasis bei spiečiaus išsisklaidymas, kurių reikalauja skirtingos misijos fazės.

Nors sukurtas algoritmas sėkmingai įgyvendina visas keturias misijos fazes, jis turi keletą trūkumų. Pirmasis trūkumas yra tai, kad algoritmas yra projektuotas dvimatei koordinačių sistemai, todėl jo naudojimas realiose 3D aplinkose reikalautų papildomo pritaikymo. Antra, algoritmas neįvertina didelių komunikacijos trikdžių, kurie realiose sąlygose gali turėti daug įtakos spiečiaus veikimui. Simuliacijoje daroma prielaida, kad visi agentai iš karto gauna tikslią informaciją apie kitų dronų pozicijas, tačiau praktikoje radijo ryšio trukdžiai, specialus sabotažas, ribotas informacijos perdavimo greitis ar signalo praradimas gali sutrikdyti koordinuotą spiečiaus veikimą. Trečiasis trūkumas yra

„Fuzzy“ logikos trūkumai. Šios logikos taisyklių bazės yra statinės, kurios nesikeičia prisitaikydamos prie aplinkos pokyčių, todėl algoritmas gali veikti neoptimaliai, kuomet kinta sąlygos.

Išvados

1. Darbe yra išnagrinėtos dabartinės saviorganizuojančio bepiločių orlaivių spiečiaus technologijos bei jų algoritmų charakteristikos. Nustatyta, jog spiečiaus valdymui yra naudojami „*Boids“*, „*Vicsek*“, potencialių laukų matematiniai modeliai bei išnagrinėta jų privalumai bei trūkumai. Išanalizuota, kokie optimizaciniai metodai yra naudojami sudėtingesnėms algoritmų užduotims. Nustatyta, jog lyderio-sekėjų spiečiaus modelis yra tinkamas misijoms įvykdyti naudojamiems spiečiams, nes tai nustato tikslią spiečiaus hierarchiją ir palaiko decentralizuotą spiečiaus valdymą ir jo privalumus.
2. Saviorganizuojančio bepiločių orlaivių spiečiaus kontrolės algoritmui yra parenkamas naudoti *„Fuzzy“* matematinis modelis. Pasirinkime yra atsižvelgiama į *„Fuzzy“* logikos galimybes valdyti pagrindines spiečiaus sistemas, kuomet nėra tikslios matematinės aplinkos bei efektyviai naudoti neapibrėžtus duomenis, lokaliai gaunamus iš spiečiaus agentų jutiklių.
3. *„Fuzzy“* matematinio modelio analizė parodė, kad sistema susideda iš trijų pagrindinių dalių. Pirmoji dalis yra sistemos įvesčių „fuzzifikacija“, taisyklių apibrėžimo, jų įvertinimo bei išvesčių duomenų „defuzzifikacijos“. Gaunami neapibrėžti įvesčių duomenys yra vertinami pagal naryščių funkcijas, o jų narystės yra naudojamos iš anksto sugeneruotose taisyklėse, kurios pateikia žodinę išvesties reikšmę. Pagal išvesčių narystės funkcijas yra grąžinama reali išvesties reikšmė, kurią spiečius naudoja kaip judėjimo vektoriaus dedamąją.
4. Remiantis išnagrinėtu *„Fuzzy"* matematiniu modeliu, sukurtas keturių fazių bepiločių orlaivių spiečiaus saviorganizacijos algoritmas. Spiečiaus algoritmo tikslas yra nesusidūrus tarpusavyje surasti norimą objektą ir susidurti su juo. Šiam algoritmui sukurtos 4 *„Fuzzy“* sistemos, kurios naudojamos pirmoje bei trečioje algoritmo fazėse. Jos leidžia tiksliai reguliuoti agento greičio link/nuo spiečiaus lyderio bei artimiausio agento judėjimo dedamąją. Algoritmas taip pat naudoja lygiavimo, atskyrimo bei kohentiškumo komponenčių valdymo sistemą, kurios užtikrina saugų stabilų bei efektyvų spiečiaus judėjimą skirtingose misijos fazėse.
5. Bepiločių orlaivių spiečiaus algoritmo veikimas ištestuotas atliekant spiečiaus simuliaciją *„MATLAB“* programoje. Pirmoje fazėje spiečius formuojasi į dviejų puslankių formaciją aplink lyderį. Optimaliam puslankių pozicijos priskyrimui naudojamas Vengrų algoritmas. Antroje algoritmo fazėje iš nustatytos formacijos spiečiaus agentai skrenda nuo lyderio su tikslu surasti objektą. Jeigu objektas nėra randamas, spiečius grįžta į pirminę poziciją ir algoritmas baigiamas. Jeigu objektas surandamas, objektą suradęs agentas yra priskiriamas naujuoju spiečiaus lyderio ir trečioje fazėje likę agentai būriuojasi aplink jį. Ketvirtoje fazėje vyksta objekto ataka, kurios metu agentai turi išlaikyti saugų tarpusavio atstumą. Simuliacijoje yra renkami norimi duomenys, tokie kaip agentų tarpusavio atstumai, paieškos plotas, susibūrimo laikas, agentų trajektorijos, agentų skrydžio greitis bei spiečiaus kompaktiškumo kriterijus *E(t)*.
6. Simuliacijos rezultatų įvertinimas parodė, jog sukurtas algoritmas sėkmingai ir efektyviai įgyvendina visas misijos fazes. Pirmoje fazėje spiečius per 240 sekundžių suformavo norimą dviejų puslankių formaciją. Vidutinis nuokrypis nuo agentui nustatytos pozicijos buvo 0,02 m. Maksimalus užfiksuotas agento greitis pirmoje fazėje yra 0,032 m/s. Antroje fazėje maksimalus paieškos plotas lygus 5,2 km², kuris apieškomas su 15% ploto persidengimo saugos koeficientu. Maksimali paieška atliekama per 155 sekundes. Trečioje fazėje visi spiečiaus agentai aplink spiečiaus lyderį susibūrė per 289 sekundes. Iš agentų trajektorijų yra matoma, jog fazės gale agentai priima vėliau atskridusius agentus šalia lyderio ir dėl to vis glaudžiau būriuojasi. Ketvirtoje fazėje visi agentai sėkmingai sąusidūrė su judančiu objektu. Pirmasis agentas savo tikslą pasiekė per 42,8 sekundės, o paskutinis pataikymas įvyko po 96,4 sekundės. Iš spiečiaus

trajektorijos galima teigti, jog spiečiaus agentai efektyviai konvergavosi į tuometinę objekto poziciją, ypač tuomet, kai erdvės padidėjimas įvykdavo dėl agentų susidūrimo su objektu. Per visas misijos fazes neįvyko nei vienas tarpusavio susidūrimas, kas įrodo efektyvų spiečiaus judėjimą ir atstumo taisyklių laikymąsi. Vertinant spiečiaus kompaktiškumo kriterijų per visas misijos fazes galima teigti, jog *E(t)* kitimas patvirtina algoritmo veikimo logiką. Pirmoje fazėje kriterijus išaugo iki 5,5 m, nes spiečius išsiskleidė į puslankių formaciją. Antroje fazėje kriterijus toliau augo iki 966 metrų, kas pagrindžia plačią spiečiaus paieškos zoną. Trečioje fazėje kritejus mažėja iki 31 m dėl efektyvaus spiečiaus susibūriavimo prie naujojo lyderio. Ketvirtoje fazėje kompaktiškumas mažėja iki 2,8 m, dėl susidūrusių su objektu agentų išėmimo iš tolimesnių spiečiaus skaičiavimų.

Literatūros sąrašas

1. Alireza Gholami: Exploring drone classifications and applications: a review [interaktyvus] [žiūrėta 2026-03-15] Prieiga per: https://dergipark.org.tr/en/pub/ijeg/article/1587042?issue_id=88162
2. Luca Crupi, Luca Butera, Alberto Ferrante, Alessandro Giusti ir Daniele Palossi: An Efficient Ground-aerial Transportation System for Pest Control Enabled by AI-based Autonomous Nano-UAVs [interaktyvus] [žiūrėta 2026-03-15] Prieiga per: //dl.acm.org/doi/full/10.1145/3719210
3. Patricia Ventura Diaz ir Seokkwan Yoon: High-Fidelity Computational Aerodynamics of Multi-Rotor Unmanned Aerial Vehicles [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://arc.aiaa.org/doi/abs/10.2514/6.2018-1266
4. Yajun Bu, Ye Yan ir Yueneng Yang: Advancement Challenges in UAV Swarm Formation Control: A Comprehensive Review [interaktyvus] [žiūrėta 2026-03-16] Prieiga per: h--ttps://www.mdpi.com/2504-446X/8/7/320
5. Hashim A. Hashim: Advances in UAV avionics systems architecture, classfication and integration: A comprehensive review and future perspectives [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://www.sciencedirect.com/science/article/pii/S2590123024020292
6. Jean-Marc Moschetta: Introduction to UAV Systems [interaktyvus] [žiūrėta 2026-04-23] prieiga per: https://www.academia.edu/112350264/Introduction_to_UAV_Systems
7. Ghozali S. Hadi, Rivaldy Varianto, Bambang Riyanto T. , Agus Budiyono: Autonomous UAV System Development for Payload Dropping Mission [interaktyvus] [žiūrėta 2026-03-16] Prieiga per: https://www.researchgate.net/profile/Ghozali-Hadi/publication/273442804_Autonomous_UAV_System_Development_for_Payload_Dropping_Mission/links/55013a0b0cf2de950a71d9a5/Autonomous-UAV-System-Development-for-Payload-Dropping-Mission.pdf
8. Mitch Campion, Prakash Ranganathan, and Saleh Faruque: A Review and Future Directions of UAV Swarm Communication Architectures [interaktyvus] [žiūrėta 2026-03-16] Prieiga per: https://und.edu/research/rias/_files/docs/swarm_ieee.pdf
9. Qiangwei Pang, Yongyong Zhu, Ye Chen, Deshi Wang, Wenkai Suo: UAV formation control based on distributed Kalman model predictive control algorithm [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://pubs.aip.org/aip/adv/article/12/8/085304/2820098
10. Algoritmo termino paaiškinimas ir struktūra. [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://www.geeksforgeeks.org/introduction-to-algorithms/
11. Laith Abualigah, Ali Diabat „Advances in Sine Cosine Algorithm: A comprehensive survey" [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://link.springer.com/content/pdf/10.1007/s10462-020-09909-3.pdf
12. Rita Parada Ramos, Sancho Moura Oliveira, Susana Margarida Vieira1 , Anders Lyhne Christensen: Evolving flocking in embodied agents based on local and global application of Reynolds' rules [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0224376
13. Christopher Hartman ir Bedřich Beneš: Autonomous boids [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.123

14. Xiaocheng Wang, Hui Zhao, Li Li: An Improved Vicsek Model of Swarms Based on a New Neighbor Strategy Considering View and Distance [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://www.mdpi.com/2076-3417/13/20/11513
15. Zongchun Liu, Yantao Tian ir Mao Yang: Behavior analysis in free space and obstacle environment of swarm robot systems based on Vicsek model [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://ieeexplore.ieee.org/abstract/document/5487023/authors#authors
16. Yingxue Zhang, Jinbao Chen, Meng Chen, Chuanzhi Chen, Zeyu Zhang, Xiaokang Deng: Integrated the Artificial Potential Field with the Leader–Follower Approach for Unmanned Aerial Vehicles Cooperative Obstacle Avoidance [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://www.mdpi.com/2227-7390/12/7/954
17. Guoqiang Hao, Qiang Lv, Zhen Ghuang, Huanlong Zhao ir Wei Chen: UAV Path Planning Based on Improved Artificial Potential Field Method [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://www.mdpi.com/2226-4310/10/6/562
18. Zain Anwar Ali, Zhangang Han 1, Rana Javed Masood: Collective Motion and Self-Organization of a Swarm of UAVs: A Cluster-Based Architecture [interaktyvus] [žiūrėta 2026-04-25] Prieiga per: https://www.mdpi.com/1424-8220/21/11/3820
19. Batool Abdulsatar Abdulghani, Mohammed Abdulsattar Abdulghani: A Comprehensive Review of Ant Colony Optimization in Swarm Intelligence for Complex Problem Solving [interaktyvus] [žiūrėta 2026-04-25] Prieiga per: https://library.acadlore.com/ATAIML/2024/3/4/ATAIML_03.04_03.pdf
20. Quoc Bao Diep, Thanh-Cong Truong, Ivan Zelinka: Planning trajectory for UAVs using the self-organizing migrating algorithm [interaktyvus] [žiūrėta 2026-04-25] Prieiga per: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0327016
21. Tarek Sheltami, Gamil Ahmed, Mustafa Ghaleb ir Ashraf Mahmoud: UAV Path Planning and Trajectory Optimization: A Comprehensive Survey [interaktyvus] [žiūrėta 2026-03-17] Prieiga per: https://link.springer.com/article/10.1007/s13369-025-10971-8
22. Wilfried Yves Hamilton Adoni, Junaidh Shaik Fareedh, Sandra Lorenz, Richard Gloaguen, Yuleika Madriz, Aastha Singh ir Thomas D. Kühne: Intelligent Swarm: Concept, Design and Validation of Self-Organized UAVs Based on Leader–Followers Paradigm for Autonomous Mission Planning [interaktyvus] [žiūrėta 2026-04-08] Prieiga per: https://ouci.dntb.gov.ua/en/works/ldeb53Y9/
23. Yongnan Jia, ir Shanqiao Qiu: Distributed Leader-Follower Flight Control for Large-Scale Clusters of Small Unmanned Aerial Vehicles [interaktyvus] [žiūrėta 2026-04-08] Prieiga per: https://ieeexplore.ieee.org/abstract/document/8360424
24. Yunming Wang, Yilin Zhong, Yuhang Zhang, Yanhong Shi, Hongrui Chen: Unmanned aerial vehicle formation control method based on improved artificial potential field and consensus [interaktyvus] [žiūrėta 2026-04-16] Prieiga per: https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/cth2.12772
25. Eulalia Balestrieri, Pasquale Daponte, Luca De Vito, Francesco Picariello, Ioan Tudosa: Sensors and Measurements for UAV Safety: An Overview [interaktyvus] [žiūrėta 2026-04-16] Prieiga per: https://www.mdpi.com/1424-8220/21/24/8253
26. Carles Sastre , Jamie Wubben, Carlos T. Calafate , Juan-Carlos Cano, Pietro Manzoni: Safe and Efficient Take-Off of VTOL UAV Swarms [interaktyvus] [žiūrėta 2026-04-16] Prieiga per: https://www.mdpi.com/2079-9292/11/7/1128

27. Dinh Quang Nguyen, Giuseppe Loianno, and Van Anh Ho: Towards Design of a Deformable Propeller for Drone Safety [interaktyvus] [žiūrėta 2026-04-16] Prieiga per: https://ieeexplore.ieee.org/document/8764020
28. Alejandro Puente-Castro, Daniel Rivero, Alejandro Pazos, Enrique Fernandez-Blanco: A review of Artificial Intelligence applied to Path Planning in UAV swarms [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://link.springer.com/article/10.1007/s00521-021-06569-4
29. Archit Semwal, Sadik Shikalgar, Dr. Ramesh Solanki: The Use of Artificial Intelligence in Swarm Drones [interaktyvus] [žiūrėta 2026-04-17] Prieiga per: https://www.ijraset.com/research-paper/the-use-of-artificial-intelligence-in-swarm-drones
30. Yunes Alqudsi, Murat Makaraci: UAV swarms: research, challenges, and future directions [interaktyvus] [žiūrėta 2026-04-17] Prieiga per: https://link.springer.com/article/10.1186/s44147-025-00582-3
31. UAV swarm exploring a desert environment in a decentralized search and rescue task without wireless communication among the robots [interaktyvus] [žiūrėta 2026-04-17] Prieiga per: https://www.researchgate.net/figure/UAV-swarm-exploring-a-desert-environment-in-a-decentralized-search-and-rescue-task_fig1_364615908
32. Zhu Xiaoning: Analysis of military application of UAV swarm technology [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://ieeexplore.ieee.org/abstract/document/9274974
33. Saifullah Khan , Muhammad Zahid Khan, Pervez Khan, Gulzar Mehmood, Ajab Khan, Muhammad Fayaz: An Ant-Hocnet Routing Protocol Based on Optimized Fuzzy Logic for Swarm of UAVs in FANET [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://onlinelibrary.wiley.com/doi/full/10.1155/2022/6783777
34. Omar Rodríguez-Abreo, Juvenal Rodríguez-Reséndiz, A. García-Cerezo, José R. García-Martínez: Fuzzy logic controller for UAV with gains optimized via genetic algorithm [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://www.cell.com/heliyon/fulltext/S2405-8440(24)02394-6
35. Hooi Hung Tang, Nur Syazreen Ahmad: Fuzzy logic approach for controlling uncertain and nonlinear systems: acomprehensive review of applications and advances [interaktyvus] [žiūrėta 2026-04-29] Prieiga per: https://www.tandfonline.com/doi/full/10.1080/21642583.2024.2394429
36. Fangcheng Zhu, Yunfan Ren, Longji Yin, Fanze Kong, Qingbo Liu, Ruize Xue: Swarm-LIO2: Decentralized Efficient LiDAR-Inertial Odometry for Aerial Swarm Systems [interaktyvus] [žiūrėta 2026-04-16] prieiga per: https://ieeexplore.ieee.org/abstract/document/10816004
37. G. Ayorkor Mills-Tettey, Anthony Stentz, M. Bernardine Dias: The Dynamic Hungarian Algorithm for the Assignment Problem with Changing Costs [interaktyvus] [žiūrėta 2026-04-16] Prieiga per: https://kilthub.cmu.edu/articles/journal_contribution/The_Dynamic_Hungarian_Algorithm_for_the_Assignment_Problem_with_Changing_Costs/6561212/files/12043517.pdf
38. Tingpeng Li, Yue Li, Yanling Qian: Improved Hungarian algorithm for assignment problems of serial-parallel systems [interaktyvus] [žiūrėta 2026-04-17] Prieiga per: https://ieeexplore.ieee.org/abstract/document/7669704

Priedai

1 Priedas. Algoritmo simuliacijos kodas MATLAB programoje

```
clc; clear; close all;

%% ========================================================
%  Fazė 1: Puslankio formavimas
%  Fazė 2: Paieška
%  Fazė 3: Susibūrimas prie lyderio
%  Fazė 4: Ataka
% ========================================================

%% ========================================================
%  STATISTIKOS KINTAMIEJI
% ========================================================
% Fazė 1
stat_f1_zingsniai    = 0;
stat_f1_pavojingi    = 0;        % pavojingų atstumų skaičius
stat_f1_pavojingi_z = zeros(1, 10000);
stat_f1_pavojingi_z_idx = 0;
stat_f1_min_atstumas   = inf(20, 1);
stat_f1_min_atstumas_z = zeros(20, 1);

% Fazė 2
stat_f2_zingsniai    = 0;
stat_f2_plotas       = 0;
stat_f2_agentas      = -1;

% Fazė 3
stat_f3_zingsniai    = 0;
stat_f3_pavojingi    = 0;
stat_f3_pavojingi_z     = zeros(1, 10000);
stat_f3_pavojingi_z_idx = 0;
stat_f3_vid_atstumas = 0;
stat_f3_galutiniai_ats = zeros(20, 1);

% Fazė 4
stat_f4_pataikymai   = zeros(21, 1);  % kiekvieno drono pataikymo zingsnis
stat_f4_min_atstumas   = inf(21, 1);   %
stat_f4_min_atstumas_z = zeros(21, 1); %

%% --- vaizdo įrašas ---
writerObj = VideoWriter('simuliacija.mp4', 'MPEG-4');
writerObj.FrameRate = 10;
open(writerObj);

%% ========================================================
%  FUZZY SISTEMU KURIMAS – FAZĖ 1
% ========================================================

%% faze1ag – greitis nuo/į agentą
faze1ag = mamfis(Name="faze1ag", DefuzzificationMethod="bisector");
% Input 1: atstumas nuo lyderio [0-20m]
faze1ag = addInput(faze1ag,[0 20],Name="atstumas nuo lyderio");
faze1ag = addMF(faze1ag,"atstumas nuo lyderio","trapmf",[-3.75 -0.4167 0.4167
3.75],Name="labai arti",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo lyderio","trimf",[0.833333 5
9.16667],Name="arti",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo lyderio","trimf",[5.83333 10
14.1667],Name="tinkama",VariableType="input");
```

```
faze1ag = addMF(faze1ag,"atstumas nuo lyderio","trimf",[10.8333 15
19.1667],Name="toli",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo lyderio","trapmf",[16.25 19.58 20.42
23.75],Name="labai toli",VariableType="input");
% Input 2: atstumas nuo agento [0-10m]
faze1ag = addInput(faze1ag,[0 10],Name="atstumas nuo agento");
faze1ag = addMF(faze1ag,"atstumas nuo agento","trapmf",[-1.875 -0.2083 0.2083
1.875],Name="labai arti",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo agento","trimf",[0.41667 2.5
4.58333],Name="arti",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo agento","trimf",[2.91667 5
7.08333],Name="tinkama",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo agento","trimf",[5.41667 7.5
9.58333],Name="toli",VariableType="input");
faze1ag = addMF(faze1ag,"atstumas nuo agento","trapmf",[8.125 9.792 10.21
11.88],Name="labai toli",VariableType="input");
% Output: greitis nuo agento [-0.03, 0.03]
faze1ag = addOutput(faze1ag,[-0.03 0.03],Name="greitis nuo agento");
faze1ag = addMF(faze1ag,"greitis nuo agento","trapmf",[-0.04125 -0.03125 -0.02875 -
0.01875],Name="greitai juda nuo",VariableType="output");
faze1ag = addMF(faze1ag,"greitis nuo agento","trapmf",[-0.02625 -0.01625 -0.01375 -
0.00375],Name="juda nuo",VariableType="output");
faze1ag = addMF(faze1ag,"greitis nuo agento","trimf",[-0.0125 0
0.0125],Name="nekinta",VariableType="output");
faze1ag = addMF(faze1ag,"greitis nuo agento","trapmf",[0.00375 0.01375 0.01625
0.02625],Name="juda į",VariableType="output");
faze1ag = addMF(faze1ag,"greitis nuo agento","trapmf",[0.01875 0.02875 0.03125
0.04125],Name="greitai juda į",VariableType="output");
% Taisyklės
faze1ag = addRule(faze1ag,[
    1 1 1 1 1; 2 1 1 1 1; 3 1 1 0.8 1; 4 1 1 0.9 1; 5 1 1 1 1;
    1 2 2 0.8 1; 2 2 2 1 1; 3 2 2 1 1; 4 2 2 1 1; 5 2 1 0.9 1;
    1 3 3 0.8 1; 2 3 3 1 1; 3 3 3 1 1; 4 3 3 1 1; 5 3 3 1 1;
    1 4 3 1 1; 2 4 3 1 1; 3 4 4 1 1; 4 4 5 0.7 1; 5 4 4 1 1;
    1 5 3 1 1; 2 5 3 1 1; 3 5 4 1 1; 4 5 5 1 1; 5 5 5 1 1]);

%% faze1lyd – greitis nuo/į lyderį
faze1lyd = mamfis(Name="faze1lyd", DefuzzificationMethod="bisector");
% Input 1: atstumas nuo lyderio [0-20m]
faze1lyd = addInput(faze1lyd,[0 20],Name="atstumas nuo lyderio");
faze1lyd = addMF(faze1lyd,"atstumas nuo lyderio","trapmf",[-3.75 -0.4167 0.4167
3.75],Name="labai arti",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo lyderio","trimf",[0.833333 5
9.16667],Name="arti",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo lyderio","trimf",[5.83333 10
14.1667],Name="tinkama",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo lyderio","trimf",[10.8333 15
19.1667],Name="toli",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo lyderio","trapmf",[16.25 19.58 20.42
23.75],Name="labai toli",VariableType="input");
% Input 2: atstumas nuo agento [0-10m]
faze1lyd = addInput(faze1lyd,[0 10],Name="atstumas nuo agento");
faze1lyd = addMF(faze1lyd,"atstumas nuo agento","trapmf",[-1.875 -0.2083 0.2083
1.875],Name="labai arti",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo agento","trimf",[0.41667 2.5
4.58333],Name="arti",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo agento","trimf",[2.91667 5
7.08333],Name="tinkama",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo agento","trimf",[5.41667 7.5
9.58333],Name="toli",VariableType="input");
faze1lyd = addMF(faze1lyd,"atstumas nuo agento","trapmf",[8.125 9.792 10.21
11.88],Name="labai toli",VariableType="input");
```

```
% Output: greitis nuo lyderio [-0.05, 0.05]
faze1lyd = addOutput(faze1lyd,[-0.05 0.05],Name="greitis nuo lyderio");
faze1lyd = addMF(faze1lyd,"greitis nuo lyderio","trapmf",[-0.06875 -0.05208 -0.04792 -
0.03125],Name="greitai juda nuo",VariableType="output");
faze1lyd = addMF(faze1lyd,"greitis nuo lyderio","trapmf",[-0.04375 -0.02708 -0.02292 -
0.00625],Name="juda nuo",VariableType="output");
faze1lyd = addMF(faze1lyd,"greitis nuo lyderio","trimf",[-0.0208333 0
0.0208333],Name="nekinta",VariableType="output");
faze1lyd = addMF(faze1lyd,"greitis nuo lyderio","trapmf",[0.00625 0.02292 0.02708
0.04375],Name="juda į",VariableType="output");
faze1lyd = addMF(faze1lyd,"greitis nuo lyderio","trapmf",[0.03125 0.04792 0.05208
0.06875],Name="greitai juda į",VariableType="output");
% Taisyklės
faze1lyd = addRule(faze1lyd,[
    1 1 1 1 1; 2 1 2 1 1; 3 1 3 1 1;
    1 2 1 1 1; 2 2 2 1 1; 3 2 3 1 1;
    1 3 1 1 1; 2 3 2 1 1; 3 3 3 1 1;
    1 4 3 1 1; 1 5 3 1 1;
    2 4 2 1 1; 2 5 3 1 1;
    3 4 3 1 1; 3 5 3 1 1;
    4 1 4 1 1; 4 2 4 1 1; 4 3 4 1 1; 4 4 5 0.7 1; 4 5 4 1 1;
    5 1 4 0.8 1; 5 2 5 0.8 1; 5 3 5 0.8 1; 5 4 5 1 1; 5 5 5 1 1]);

%% ============================================================
%  FUZZY SISTEMU KURIMAS — FAZE 3
% ============================================================

%% faze3lyd — greitis link naujojo lyderio
faze3lyd = mamfis(Name="faze3lyd", DefuzzificationMethod="bisector");
% Input 1: atstumas nuo lyderio [0-2000m]
faze3lyd = addInput(faze3lyd,[0 2000],Name="atstumas nuo lyderio");
faze3lyd = addMF(faze3lyd,"atstumas nuo lyderio","trapmf",[-37.5 -4.167 4.167
37.5],Name="labai arti",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo lyderio","trimf",[8.333 50
91.667],Name="arti",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo lyderio","trimf",[58.333 500
941.667],Name="tinkama",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo lyderio","trimf",[458.333 1000
1541.667],Name="toli",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo lyderio","trapmf",[1500 1800 2200
2500],Name="labai toli",VariableType="input");
% Input 2: atstumas nuo agento [0-2000m]
faze3lyd = addInput(faze3lyd,[0 2000],Name="atstumas nuo agento");
faze3lyd = addMF(faze3lyd,"atstumas nuo agento","trapmf",[-0.675 -0.075 0.075
0.675],Name="labai arti",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo agento","trimf",[0.15 0.9
1.65],Name="arti",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo agento","trimf",[1.05 3.6
4.5],Name="tinkama",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo agento","trapmf",[4.5 10 400
450],Name="toli",VariableType="input");
faze3lyd = addMF(faze3lyd,"atstumas nuo agento","trapmf",[400 450 2042
2375],Name="labai toli",VariableType="input");
% Output: greitis nuo lyderio [-7, 7] m/s
faze3lyd = addOutput(faze3lyd,[-7 7],Name="greitis nuo lyderio");
faze3lyd = addMF(faze3lyd,"greitis nuo lyderio","trapmf",[-9.625 -7.292 -6.708 -4.375],
...
    Name="greitai juda nuo",VariableType="output");
faze3lyd = addMF(faze3lyd,"greitis nuo lyderio","trapmf",[-6.125 -3.792 -3.208 -0.875],
...
    Name="juda nuo",VariableType="output");
faze3lyd = addMF(faze3lyd,"greitis nuo lyderio","trimf",[-2.91667 0 2.91667], ...
```

```
    Name="nekinta",VariableType="output");
faze3lyd = addMF(faze3lyd,"greitis nuo lyderio","trapmf",[0.875 3.208 3.792 6.125], ...
    Name="juda į",VariableType="output");
faze3lyd = addMF(faze3lyd,"greitis nuo lyderio","trapmf",[4.375 6.708 7.292 9.625], ...
    Name="greitai juda į",VariableType="output");
% Taisyklės
faze3lyd = addRule(faze3lyd,[
    1 1 1 1 1; 2 1 2 1 1; 3 1 3 1 1;
    1 2 1 1 1; 2 2 2 1 1; 3 2 3 1 1;
    1 3 1 1 1; 2 3 2 1 1; 3 3 3 1 1;
    1 4 3 1 1; 1 5 3 1 1;
    2 4 2 1 1; 2 5 3 1 1;
    3 4 4 1 1; 3 5 4 1 1;
    4 1 4 1 1; 4 2 4 1 1; 4 3 4 1 1; 4 4 5 0.7 1; 4 5 4 1 1;
    5 1 4 0.8 1; 5 2 5 0.8 1; 5 3 5 0.8 1; 5 4 5 1 1; 5 5 5 1 1]);

%% faze3ag – atsitraukimas nuo kaimynų
faze3ag = mamfis(Name="faze3ag", DefuzzificationMethod="bisector");
% Input 1: atstumas nuo lyderio [0-2000m]
faze3ag = addInput(faze3ag,[0 2000],Name="atstumas nuo lyderio");
faze3ag = addMF(faze3ag,"atstumas nuo lyderio","trapmf",[-37.5 -4.167 4.167
37.5],Name="labai arti",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo lyderio","trimf",[8.333 50
91.667],Name="arti",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo lyderio","trimf",[58.333 500
941.667],Name="tinkama",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo lyderio","trimf",[458.333 1000
1541.667],Name="toli",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo lyderio","trapmf",[1500 1800 2200
2500],Name="labai toli",VariableType="input");
% Input 2: atstumas nuo agento [0-2000m]
faze3ag = addInput(faze3ag,[0 2000],Name="atstumas nuo agento");
faze3ag = addMF(faze3ag,"atstumas nuo agento","trapmf",[-0.675 -0.075 0.075
0.675],Name="labai arti",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo agento","trimf",[0.15 0.9
1.65],Name="arti",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo agento","trimf",[1.05 3.6
4.5],Name="tinkama",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo agento","trapmf",[4.5 10 400
450],Name="toli",VariableType="input");
faze3ag = addMF(faze3ag,"atstumas nuo agento","trapmf",[400 450 2042 2375],Name="labai
toli",VariableType="input");
% Output: greitis nuo agento [-3, 3]
faze3ag = addOutput(faze3ag,[-3 3],Name="greitis nuo agento");
faze3ag = addMF(faze3ag,"greitis nuo agento","trapmf",[-4.125 -3.125 -2.875 -1.875],
...
    Name="greitai juda nuo",VariableType="output");
faze3ag = addMF(faze3ag,"greitis nuo agento","trapmf",[-2.625 -1.625 -1.375 -0.375],
...
    Name="juda nuo",VariableType="output");
faze3ag = addMF(faze3ag,"greitis nuo agento","trimf",[-1.25 0 1.25], ...
    Name="nekinta",VariableType="output");
faze3ag = addMF(faze3ag,"greitis nuo agento","trapmf",[0.375 1.375 1.625 2.625], ...
    Name="juda į",VariableType="output");
faze3ag = addMF(faze3ag,"greitis nuo agento","trapmf",[1.875 2.875 3.125 4.125], ...
    Name="greitai juda į",VariableType="output");
% Taisyklės
faze3ag = addRule(faze3ag,[
    1 1 1 1 1; 2 1 1 1 1; 3 1 1 0.8 1; 4 1 1 0.9 1; 5 1 1 1 1;
    1 2 2 0.8 1; 2 2 2 1 1; 3 2 2 1 1; 4 2 2 1 1; 5 2 1 0.9 1;
    1 3 3 0.8 1; 2 3 3 1 1; 3 3 3 1 1; 4 3 3 1 1; 5 3 3 1 1;
    1 4 3 1 1; 2 4 3 1 1; 3 4 4 1 1; 4 4 5 0.7 1; 5 4 4 1 1;
```

```
    1 5 3 1 1; 2 5 3 1 1; 3 5 4 1 1; 4 5 5 1 1; 5 5 5 1 1]);

%% ============================================================
%  BENDRI PARAMETRAI
% ============================================================
dt = 0.4;  % laiko žingsnis (s) — 1 žingsnis = 0.4 sekundės

%% ============================================================
%  FAZĖ 1 — Puslankio formavimas
% ============================================================
disp('=== FAZĖ 1: Puslankio formavimas ===');

% Parametrai
N        = 20;
N_row    = N / 2;   % 10 agentų eilej
R1       = 6;       % vidinės eiles spindulys
R2       = 9;       % išorinės eiles spindulys
R        = 8;       % vidutinis spindulys
zingsniai1 = 600;
Et_f1 = zeros(zingsniai1, 1);
greitis_f1 = zeros(N, zingsniai1);

% Lyderio pozicija
lyd = [0, 0];

% Pradinės agentų pozicijos
A = [ -4 2; -2 2; 0 2; 2 2; 4 2; ...
      -4 4; -2 4; 0 4; 2 4; 4 4; ...
      -4 6; -2 6; 0 6; 2 6; 4 6; ...
      -4 8; -2 8; 0 8; 2 8; 4 8 ];

% Tikslinės pozicijos — dvi eilės šachmatais
kampas1 = linspace(deg2rad(0),   deg2rad(162), N_row);
kampas2 = linspace(deg2rad(9),   deg2rad(171), N_row);
ag_vieta = zeros(N, 2);
for i = 1:N_row
    ag_vieta(i,1)       = lyd(1) + R1 * cos(kampas1(i));
    ag_vieta(i,2)       = lyd(2) + R1 * sin(kampas1(i));
    ag_vieta(N_row+i,1) = lyd(1) + R2 * cos(kampas2(i));
    ag_vieta(N_row+i,2) = lyd(2) + R2 * sin(kampas2(i));
end

% vengru algoritmas
assigned_targets = assign_targets(A, ag_vieta);

figure('Color','w','Position',[100 100 700 700]);

for t = 1:zingsniai1
    clf; hold on; grid on;
    axis equal; axis([-15 15 -15 15]);
    xlabel('x, m'); ylabel('y, m');
    title(sprintf('Fazė 1 — Puslankio formavimas  |  Žingsnis: %d / %d', t,
zingsniai1));

    % Atstumai
    dxL = lyd(1) - A(:,1);
    dyL = lyd(2) - A(:,2);
    atstumas_lyd = hypot(dxL, dyL);
    D = pdist2(A, A);
    D(1:N+1:end) = inf;
    minAg = min(D, [], 2);
```

```
    % FIS ivertinimas
    u_lyd = zeros(N,1);
    u_ag  = zeros(N,1);
    for i = 1:N
        in1 = [min(atstumas_lyd(i), 20), min(minAg(i), 10)];
        u_lyd(i) = evalfis(faze1lyd, in1);
        u_ag(i)  = evalfis(faze1ag,  in1);
    end

    % Poziciju atnaujinimas
    for i = 1:N
        ti = assigned_targets(i);
        dx_t = ag_vieta(ti,1) - A(i,1);
        dy_t = ag_vieta(ti,2) - A(i,2);
        dist_t = hypot(dx_t, dy_t);
        if dist_t > 0.01
            nx = dx_t / dist_t; ny = dy_t / dist_t;
        else
            nx = 0; ny = 0;
        end



        % Atsitraukimas nuo lyderio
        dx_lyd = A(i,1) - lyd(1);
        dy_lyd = A(i,2) - lyd(2);
        ats_lyd_rep = hypot(dx_lyd, dy_lyd);
        lyd_rep_rad = 1.5;
        if ats_lyd_rep < lyd_rep_rad && ats_lyd_rep > 0.01
            lx = dx_lyd/ats_lyd_rep; ly = dy_lyd/ats_lyd_rep;
            lyd_rep_force = 1.5 * (1 - ats_lyd_rep/lyd_rep_rad);
        else
            lx = 0; ly = 0; lyd_rep_force = 0;
        end

        % Atsitraukimas nuo kaimynų
        sep_x = 0; sep_y = 0;
        sep_rad = 1.8;
        for j = 1:N
            if j == i, continue; end
            dx_ij = A(i,1)-A(j,1); dy_ij = A(i,2)-A(j,2);
            ats_ij = hypot(dx_ij, dy_ij);
            if ats_ij < sep_rad && ats_ij > 0.01
                ats_jega = 0.3 * (1 - ats_ij/sep_rad);
                sep_x = sep_x + ats_jega*dx_ij/ats_ij;
                sep_y = sep_y + ats_jega*dy_ij/ats_ij;
            end
        end

        v1 = abs(u_lyd(i)) + abs(u_ag(i));
        v1 = max(v1, 0.025) + 0.04 * min(dist_t/R, 1.0);

        vx_total = v1*nx + sep_x + lyd_rep_force*lx;
vy_total = v1*ny + sep_y + lyd_rep_force*ly;
greitis_f1(i, t) = hypot(vx_total, vy_total) * dt;
A(i,1) = A(i,1) + vx_total * dt;
A(i,2) = A(i,2) + vy_total * dt;
    end
% Statistika – pavojingi atstumai
min_dist = min(D(:));
if min_dist < 1.0
    stat_f1_pavojingi = stat_f1_pavojingi + 1;
```

```
    stat_f1_pavojingi_z_idx = stat_f1_pavojingi_z_idx + 1;
    stat_f1_pavojingi_z(stat_f1_pavojingi_z_idx) = t;
end
min_per_ag = min(D, [], 2);  % minimalus atstumas kiekvienam agentui
pakeistas = min_per_ag < stat_f1_min_atstumas;
stat_f1_min_atstumas_z(pakeistas) = t;
stat_f1_min_atstumas(pakeistas)   = min_per_ag(pakeistas);
stat_f1_zingsniai = t;
xc = mean(A(:,1)); yc = mean(A(:,2));
Et_f1(t) = mean(hypot(A(:,1)-xc, A(:,2)-yc));
    % Piešimas
    plot(ag_vieta(:,1), ag_vieta(:,2), 'o', 'Color',[0.7 0.7 0.7], 'MarkerSize',8);
    plot(A(:,1), A(:,2), 'b*', 'MarkerSize',9, 'LineWidth',1.2);
    for i = 1:N
        text(A(i,1)+0.2, A(i,2)+0.2, num2str(i), 'FontSize',7, 'Color','b');
    end
    plot(lyd(1), lyd(2), 'r*', 'MarkerSize',14, 'LineWidth',2);
    text(lyd(1)+0.3, lyd(2)+0.3, 'L', 'FontSize',10, 'Color','r', 'FontWeight','bold');
    legend({'Tikslinės poz.','Agentai','Lyderis'}, 'Location','northeast',
'FontSize',8);
    drawnow;
    frame = getframe(gcf);
    writeVideo(writerObj, frame);
end

% Vidutinis nuokrypis nuo tikslinės pozicijos
nuokrypiai = zeros(N, 1);
for i = 1:N
    ti = assigned_targets(i);
    nuokrypiai(i) = hypot(A(i,1)-ag_vieta(ti,1), A(i,2)-ag_vieta(ti,2));
end
stat_f1_vid_nuokrypis = mean(nuokrypiai);

disp('✓ Fazė 1 baigta.');

%% ============================================================
%  FAZĖ 2 — Sweep paieška
% ============================================================
disp('=== FAZĖ 2: Sweep paieška ===');

% Parametrai
stop_ats  = 300;   % sustojimo atstumas tarp agentu (m)
v_ies        = 5;     % agentų greitis (m/žingsnis)
detect_rad = 150; % aptikimo spindulys (m)
zingsniai2      = 500;
Et_f2 = zeros(265, 1);
% Objekto pozicija — keisti pagal poreikį
obj_koord  = [-700, 1300];
obj_rast   = false;
surades_ag = -1;

% Kiekvieno agento judėjimo kampas
ag_kamp = zeros(N, 1);
for i = 1:N
    ag_kamp(i) = atan2(A(i,2)-lyd(2), A(i,1)-lyd(1));
end

% Fazės 1 pozicijos išsaugomos grįžimui
A_faz1 = A;

figure('Color','w','Position',[100 100 700 700]);
```

```
for t = 1:zingsniai2
    clf; hold on; grid on;
    axis equal; axis([-1500 1500 0 1500]);
    xlabel('x, m'); ylabel('y, m');
    title(sprintf('Fazė 2 – Sweep paieška  |  Žingsnis: %d', t));

    % Agentų judėjimas
    for i = 1:N
        A(i,1) = A(i,1) + v_ies * cos(ag_kamp(i));
        A(i,2) = A(i,2) + v_ies * sin(ag_kamp(i));
    end

% Maksimalus paieškos plotas
% Puslankio plotas
max_r = max(hypot(A(:,1)-lyd(1), A(:,2)-lyd(2)));
stat_f2_plotas = max(stat_f2_plotas, 0.5 * pi * max_r^2);
stat_f2_zingsniai = t;

    % Objekto aptikimas
    for i = 1:N
        if hypot(A(i,1)-obj_koord(1), A(i,2)-obj_koord(2)) < detect_rad
            obj_rast    = true;
            surades_ag = i;
            break;
        end
    end
stat_f2_agentas = surades_ag;

    % Sustojimo sąlyga
    [~, sorted_idx] = sort(ag_kamp);
min_kaimynas_ats = inf;
for k = 1:N-1
    i1 = sorted_idx(k);
    i2 = sorted_idx(k+1);
    d = hypot(A(i1,1)-A(i2,1), A(i1,2)-A(i2,2));
    min_kaimynas_ats = min(min_kaimynas_ats, d);
end

% Paskutinio žingsnio realiai apieškotas plotas
last_r = max(hypot(A(:,1)-lyd(1), A(:,2)-lyd(2)));
stat_f2_plotas_last = 0.5 * pi * last_r^2;

% Maksimalus galimas plotas
R_max = (N-1) * stop_ats / pi;
stat_f2_plotas_max = 0.5 * pi * R_max^2;

    % Piešimas
    theta_c = linspace(0, 2*pi, 50);
    for i = 1:N
        plot(A(i,1)+detect_rad*cos(theta_c), A(i,2)+detect_rad*sin(theta_c), ...
            'b-', 'LineWidth',0.5, 'Color',[0.6 0.8 1.0]);
    end
    plot(A(:,1), A(:,2), 'b*', 'MarkerSize',9, 'LineWidth',1.2);
    for i = 1:N
        text(A(i,1)+2, A(i,2)+2, num2str(i), 'FontSize',7, 'Color','b');
    end
    plot(lyd(1), lyd(2), 'r*', 'MarkerSize',14, 'LineWidth',2);
    text(lyd(1)+5, lyd(2)+5, 'L', 'FontSize',10, 'Color','r', 'FontWeight','bold');
    for i = 1:N
        plot([lyd(1),A(i,1)],[lyd(2),A(i,2)],'--','Color',[0.85 0.85
0.85],'LineWidth',0.5);
    end
```

```
    h1 = plot(A(:,1), A(:,2), 'b*', 'MarkerSize',9, 'LineWidth',1.2);
h2 = plot(lyd(1), lyd(2), 'r*', 'MarkerSize',14, 'LineWidth',2);
h3 = plot(obj_koord(1), obj_koord(2), 'gp', 'MarkerSize',14, 'MarkerFaceColor','g');
h4 = plot(A(1,1)+detect_rad*cos(linspace(0,2*pi,50)), ...
          A(1,2)+detect_rad*sin(linspace(0,2*pi,50)), ...
          'b-', 'Color',[0.6 0.8 1.0]);

legend([h1(1), h2, h3, h4], ...
    {'Agentai','Lyderis','Objektas','Paieškos spindulys'}, ...
    'Location','northeast','FontSize',8);
    drawnow;
    frame = getframe(gcf);
    writeVideo(writerObj, frame);
xc = mean(A(:,1)); yc = mean(A(:,2));
Et_f2(t) = mean(hypot(A(:,1)-xc, A(:,2)-yc));
    % Pabaigos sąlygos
    if obj_rast
        disp(['✓ Objektas rastas! Agentas nr. ' num2str(surades_ag)]);
        disp('→ Tęsiama į fazę 3.');
        break;
    end
    if min_kaimynas_ats >= stop_ats
        disp('✗ Objektas nerastas. Agentai grįžta į fazės 1 pozicijas.');
        A = returnphase1(A, A_faz1, lyd, writerObj, v_ies);
        break;
    end

end
disp('✓ Fazė 2 baigta.');

%% ============================================================
%  FAZĖ 3 — Susibūrimas aplink naująjį lyderį
% ============================================================
if obj_rast
    disp('=== FAZĖ 3: Susibūrimas aplink naująjį lyderį ===');

    % Naujasis lyderis = agentas kuris rado objektą
    naujas_lyd = A(surades_ag, :);
    A(surades_ag, :) = [];
    A = [A; lyd];           % senasis lyderis tampa agentu
    N3 = size(A, 1);        % 20 agentų
    stat_f3_min_atstumas   = inf(N3, 1);
    stat_f3_min_atstumas_z = zeros(N3, 1);
    % Parametrai
    lyd_follow_dist = detect_rad;  % lyderio atstumas nuo objekto
    v_new_lyd       = 0;                % lyderis nejuda
    zingsniai3       = 900;
    Et_f3 = zeros(723, 1);
    figure('Color','w','Position',[100 100 700 700]);
traj_f3 = zeros(N3, 2, zingsniai3);  % x,y kiekvienam agentui
    for t = 1:zingsniai3
        clf; hold on; grid on;
        axis equal; axis([-1500 1500 -100 2000]);
        xlabel('x, m'); ylabel('y, m');
        title(sprintf('Fazė 3 — Susibūrimas  |  Žingsnis: %d', t));

        % Atstumai
        dxL = naujas_lyd(1) - A(:,1);
        dyL = naujas_lyd(2) - A(:,2);
        atstumas_lyd = hypot(dxL, dyL);
        D = pdist2(A, A);
```

```
        D(1:N3+1:end) = inf;
        minAg = min(D, [], 2);

        % FIS įvertinimas
        u_lyd = zeros(N3,1);
        u_ag  = zeros(N3,1);
        for i = 1:N3
            in1 = [min(max(atstumas_lyd(i),0),2000), min(max(minAg(i),0),2000)];
            u_lyd(i) = evalfis(faze3lyd, in1);
            u_ag(i)  = evalfis(faze3ag,  in1);
        end

        % Pozicijų atnaujinimas
        for i = 1:N3
            if atstumas_lyd(i) > 0.01
                nx = dxL(i)/atstumas_lyd(i); ny = dyL(i)/atstumas_lyd(i);
            else
                nx = 0; ny = 0;
            end

             % Atsitraukimas nuo kaimynų
            sep_x = 0; sep_y = 0;
            sep_rad = 100;
            for j = 1:N3
                if j == i, continue; end
                dx_ij = A(i,1)-A(j,1); dy_ij = A(i,2)-A(j,2);
                ats_ij = hypot(dx_ij, dy_ij);
                if ats_ij < sep_rad && ats_ij > 0.01
                    prox = max(0.1, 1 - atstumas_lyd(i)/1000);
                    ats_jega = 2.0 * (1 - ats_ij/sep_rad)^2 * prox;
                    sep_x = sep_x + ats_jega*dx_ij/ats_ij;
                    sep_y = sep_y + ats_jega*dy_ij/ats_ij;
                end
            end

            % Atsitraukimas nuo lyderio
            dx_lyd = A(i,1)-naujas_lyd(1); dy_lyd = A(i,2)-naujas_lyd(2);
            ats_lyd_rep = hypot(dx_lyd, dy_lyd);
            lyd_rep_rad = 40;
            if ats_lyd_rep < lyd_rep_rad && ats_lyd_rep > 0.01
                lx = dx_lyd/ats_lyd_rep; ly = dy_lyd/ats_lyd_rep;
                lyd_rep_force = 1.5 * (1 - ats_lyd_rep/lyd_rep_rad);
            else
                lx = 0; ly = 0; lyd_rep_force = 0;
            end

            v1 = abs(u_lyd(i));
            v1 = max(v1, 2.0);
            v1 = min(v1, 8.0);

            A(i,1) = A(i,1) + (v1*nx + sep_x + lyd_rep_force*lx) * dt;
            A(i,2) = A(i,2) + (v1*ny + sep_y + lyd_rep_force*ly) * dt;
        end

    traj_f3(:,:,t) = A;
% Pavojingi atstumai
min_dist_f3 = min(D(:));
if min_dist_f3 < 1.0
    stat_f3_pavojingi = stat_f3_pavojingi + 1;
    stat_f3_pavojingi_z_idx = stat_f3_pavojingi_z_idx + 1;
    stat_f3_pavojingi_z(stat_f3_pavojingi_z_idx) = t;
end
```

```
min_per_ag = min(D, [], 2);
pakeistas = min_per_ag < stat_f3_min_atstumas;
stat_f3_min_atstumas_z(pakeistas) = t;
stat_f3_min_atstumas(pakeistas)   = min_per_ag(pakeistas);
stat_f3_zingsniai = t;

        % Piešimas
        plot(obj_koord(1), obj_koord(2), 'gp', 'MarkerSize',14, 'MarkerFaceColor','g');
        text(obj_koord(1)+5, obj_koord(2)+5, 'OBJ', 'FontSize',8, 'Color',[0 0.6 0]);
        theta_c = linspace(0, 2*pi, 60);

        plot(naujas_lyd(1), naujas_lyd(2), 'm*', 'MarkerSize',14, 'LineWidth',2);
        text(naujas_lyd(1)+5, naujas_lyd(2)+5, 'NL', 'FontSize',10, 'Color','m',
'FontWeight','bold');
        plot(lyd(1), lyd(2), 'r*', 'MarkerSize',10, 'LineWidth',1.5);
        text(lyd(1)+5, lyd(2)+5, 'L', 'FontSize',8, 'Color','r');
        plot(A(:,1), A(:,2), 'b*', 'MarkerSize',9, 'LineWidth',1.2);
        for i = 1:N3
            text(A(i,1)+2, A(i,2)+2, num2str(i), 'FontSize',7, 'Color','b');
        end
        plot([naujas_lyd(1),obj_koord(1)],[naujas_lyd(2),obj_koord(2)],'m--
','LineWidth',1);
        legend({'Objektas','Naujasis lyderis','Senojo lyderio buvus
pozicija','Agentai'}, ...
            'Location','northeast','FontSize',7);
        drawnow;
        frame = getframe(gcf);
        writeVideo(writerObj, frame);
    xc = mean(A(:,1)); yc = mean(A(:,2));
    Et_f3(t) = mean(hypot(A(:,1)-xc, A(:,2)-yc));
        % Pabaigos sąlyga
        if all(atstumas_lyd < 50.0)
            stat_f3_vid_atstumas = mean(atstumas_lyd);
            stat_f3_vid_atstumas = mean(atstumas_lyd);
            stat_f3_galutiniai_ats = atstumas_lyd;
            disp('✓ Visi agentai susitelkė aplink naująjį lyderį.');
            break;
        end
        if abs(obj_koord(1)) > 5000 || abs(obj_koord(2)) > 5000
            disp('⚠ Objektas išėjo iš zonos.');
            break;
        end

    end

    disp('✓ Fazė 3 baigta.');

    %% ============================================================
    %  FAZĖ 4 — Ataka
    % ============================================================
    disp('=== FAZĖ 4: Ataka ===');

    % Parametrai
    v4   = 7;            % dronų greitis (m/žingsnis)
    obj_v4 = 2;            % objekto greitis (0=stovi, >0=juda)
    obj_kryptis4   = deg2rad(90); % objekto judėjimo kryptis
    zingsniai4  = 500;
    % Visi dronai + naujasis lyderis
    A4 = [A; naujas_lyd];
    N4 = size(A4, 1);  % 21
    traj_f4 = zeros(N4, 2, zingsniai4);
```

```
    hit     = false(N4, 1);
    hit_poz = zeros(N4, 2);
    obj_koord_pradzios = obj_koord;
    Et_f4 = zeros(zingsniai4, 1);
    figure('Color','w','Position',[100 100 700 700]);

    for t = 1:zingsniai4
        clf; hold on; grid on;
        axis equal;
        axis([obj_koord(1)-150 obj_koord(1)+150 obj_koord(2)-150 obj_koord(2)+150]);
        xlabel('x, m'); ylabel('y, m');
        title(sprintf('Fazė 4 – Ataka  |  Žingsnis: %d', t));

        % Objekto judėjimas
        obj_koord(1) = obj_koord(1) + obj_v4 * cos(obj_kryptis4);
        obj_koord(2) = obj_koord(2) + obj_v4 * sin(obj_kryptis4);

        % Dronų judėjimas
        for i = 1:N4
            dx_obj = obj_koord(1) - A4(i,1);
            dy_obj = obj_koord(2) - A4(i,2);
            ats_obj = hypot(dx_obj, dy_obj);

            % Susidūrimo tikrinimas
            if ats_obj < 0.5 && ~hit(i)
                hit(i)     = true;
                hit_poz(i,:) = A4(i,:);
                stat_f4_pataikymai(i) = t;
            end
            if hit(i), continue; end

            nx = dx_obj / ats_obj;
            ny = dy_obj / ats_obj;

            % Atsitraukimas nuo kaimynų
            sep_x = 0; sep_y = 0;
            sep_rad4 = 15;
            for j = 1:N4
                if j == i, continue; end
                dx_ij = A4(i,1)-A4(j,1); dy_ij = A4(i,2)-A4(j,2);
                ats_ij = hypot(dx_ij, dy_ij);
                if ats_ij < sep_rad4 && ats_ij > 0.01
                    obj_prox = max(0, 1 - ats_obj/100);
                    ats_jega = 3.0 * (1 - ats_ij/sep_rad4)^2;
                    sep_x = sep_x + ats_jega*dx_ij/ats_ij;
                    sep_y = sep_y + ats_jega*dy_ij/ats_ij;
                end
            end

            A4(i,1) = A4(i,1) + (v4*nx + sep_x) * dt;
            A4(i,2) = A4(i,2) + (v4*ny + sep_y) * dt;
        end
    traj_f4(:,:,t) = A4;
        aktyvus = find(~hit);
if length(aktyvus) > 1
    A4_aktyvus = A4(aktyvus, :);
    D4 = pdist2(A4_aktyvus, A4_aktyvus);
    D4(1:length(aktyvus)+1:end) = inf;
    min_per_ag4 = min(D4, [], 2);
    for k = 1:length(aktyvus)
        i = aktyvus(k);
        if min_per_ag4(k) < stat_f4_min_atstumas(i)
```

```
            stat_f4_min_atstumas(i)   = min_per_ag4(k);
            stat_f4_min_atstumas_z(i) = t;
        end
    end
end
% E(t) fazė 4
aktyvus_et = find(~hit);
if length(aktyvus_et) > 1
    xc = mean(A4(aktyvus_et,1)); yc = mean(A4(aktyvus_et,2));
    Et_f4(t) = mean(hypot(A4(aktyvus_et,1)-xc, A4(aktyvus_et,2)-yc));
else
    Et_f4(t) = 0;
end

        % Piešimas
        h1 = plot(obj_koord(1), obj_koord(2), 'gp', 'MarkerSize',14,
'MarkerFaceColor','g');
text(obj_koord(1)+5, obj_koord(2)+5, 'OBJ', 'FontSize',8, 'Color',[0 0.6 0]);

h2 = []; h3 = [];
for i = 1:N4
    if hit(i)
        h2 = plot(hit_poz(i,1), hit_poz(i,2), 'rx', 'MarkerSize',12, 'LineWidth',2);
    else
        h3 = plot(A4(i,1), A4(i,2), 'b*', 'MarkerSize',9, 'LineWidth',1.2);
        text(A4(i,1)+2, A4(i,2)+2, num2str(i), 'FontSize',7, 'Color','b');
        plot([A4(i,1),obj_koord(1)],[A4(i,2),obj_koord(2)], ...
            '--','Color',[0.85 0.85 0.85],'LineWidth',0.5,'HandleVisibility','off');
    end
end
handles = [h1];
labels = {'Objektas'};
if ~isempty(h2), handles(end+1) = h2; labels{end+1} = 'Pataikyta'; end
if ~isempty(h3), handles(end+1) = h3; labels{end+1} = 'Dronai'; end
legend(handles, labels, 'Location','northeast','FontSize',8);
 drawnow;
        frame = getframe(gcf);
        writeVideo(writerObj, frame);
if all(hit)
            disp('✓ Visi dronai pasiekė objektą. Fazė 4 baigta.');
break;
end
    end
    obj_nujudejes = hypot(obj_koord(1) - obj_koord_pradzios(1), ...
                          obj_koord(2) - obj_koord_pradzios(2));
    disp('✓ Fazė 4 baigta.');
end
stat_f4_zingsniai = t;
close(writerObj);
disp('=== SIMULIACIJA BAIGTA. Vaizdo įrašas: simuliacija.mp4 ===');

%% ============================================================
%  STATISTIKOS IŠVEDIMAS
% ============================================================

fprintf('\n========================================\n');
fprintf('       SIMULIACIJOS STATISTIKA\n');
fprintf('========================================\n');

fprintf('\n--- FAZĖ 1: Puslankio formavimas ---\n');
fprintf('Žingsnių skaičius:       %d\n', stat_f1_zingsniai);
```

```
fprintf('Realus laikas:            %.1f s\n', stat_f1_zingsniai * dt);
fprintf('Pavojingų atstumų (<1m):  %d\n', stat_f1_pavojingi);
if ~isempty(stat_f1_pavojingi_z)
    fprintf('Žingsniai:                %s\n', num2str(unique(stat_f1_pavojingi_z)));
end
fprintf('Vid. nuokrypis nuo poz.:  %.2f m\n', stat_f1_vid_nuokrypis);
fprintf('Min. atstumai tarp agentų:\n');
for i = 1:N
    fprintf('  Agentas %2d: %.2f m (žingsnis %d)\n', i, stat_f1_min_atstumas(i),
stat_f1_min_atstumas_z(i));
end

fprintf('\n--- FAZĖ 2: paieška ---\n');
fprintf('Žingsnių skaičius:        %d\n', stat_f2_zingsniai);
fprintf('Realus laikas:            %.1f s\n', stat_f2_zingsniai * dt);
fprintf('Realiai apieškotas plotas:  %.0f m²\n', stat_f2_plotas_last);
fprintf('Maks. galimas plotas:       %.0f m²\n', stat_f2_plotas_max);
fprintf('Paieškos efektyvumas:       %.1f %%\n', (stat_f2_plotas_last /
stat_f2_plotas_max) * 100);
if obj_rast
    fprintf('Objektą rado agentas nr.: %d\n', stat_f2_agentas);
else
    fprintf('Objektas nerastas.\n');
end

fprintf('\n--- FAZĖ 3: Susibūrimas ---\n');
fprintf('Žingsnių skaičius:        %d\n', stat_f3_zingsniai);
fprintf('Realus laikas:            %.1f s\n', stat_f3_zingsniai * dt);
fprintf('Pavojingų atstumų (<1m):  %d\n', stat_f3_pavojingi);
if ~isempty(stat_f3_pavojingi_z)
    fprintf('Žingsniai:                %s\n', num2str(unique(stat_f3_pavojingi_z)));
end
fprintf('Vid. atstumas nuo lyd.:   %.2f m\n', stat_f3_vid_atstumas);
fprintf('Galutiniai atstumai nuo lyderio:\n');
for i = 1:N3
    fprintf('  Agentas %2d: %.2f m\n', i, stat_f3_galutiniai_ats(i));
end
fprintf('Min. atstumai tarp agentų:\n');
for i = 1:N3
    fprintf('  Agentas %2d: %.2f m (žingsnis %d)\n', i, stat_f3_min_atstumas(i),
stat_f3_min_atstumas_z(i));
end

fprintf('\n--- FAZĖ 4: Ataka ---\n');
valid = stat_f4_pataikymai(stat_f4_pataikymai > 0);
fprintf('Pirmas pataikymas:        %d žingsnis (%.1f s)\n', min(valid), min(valid)*dt);
fprintf('Paskutinis pataikymas:    %d žingsnis (%.1f s)\n', max(valid), max(valid)*dt);
fprintf('Visi pataikymai:\n');
for i = 1:length(stat_f4_pataikymai)
    if stat_f4_pataikymai(i) > 0
        fprintf('  Dronas %2d: %d žingsnis (%.1f s)\n', i, stat_f4_pataikymai(i),
stat_f4_pataikymai(i)*dt);
    end
end
fprintf('Min. atstumai tarp dronų:\n');
for i = 1:N4
    fprintf('  Dronas %2d: %.2f m (žingsnis %d)\n', i, stat_f4_min_atstumas(i),
stat_f4_min_atstumas_z(i));
end
fprintf('Objekto nujudėtas atstumas: %.1f m\n', obj_nujudejes);
fprintf('========================================\n\n');
```

```
fprintf('--- E(t) VERTINIMO KRITERIJUS ---\n');
fprintf('Fazė 1: %.2f m\n', Et_f1);
fprintf('Fazė 2: %.2f m\n', Et_f2);
fprintf('Fazė 3: %.2f m\n', Et_f3);

%% .txt failas
fid = fopen('simuliacijos_statistika.txt', 'w');
fprintf(fid, '========================================\n');
fprintf(fid, '       SIMULIACIJOS STATISTIKA\n');
fprintf(fid, '========================================\n');
fprintf(fid, '\n--- FAZĖ 1: Puslankio formavimas ---\n');
fprintf(fid, 'Žingsnių skaičius:        %d\n', stat_f1_zingsniai);
fprintf(fid, 'Realus laikas:            %.1f s\n', stat_f1_zingsniai * dt);
fprintf(fid, 'Pavojingų atstumų (<1m):  %d\n', stat_f1_pavojingi);
if ~isempty(stat_f1_pavojingi_z)
    fprintf(fid, 'Žingsniai:                %s\n',
num2str(unique(stat_f1_pavojingi_z)));
end
fprintf(fid, 'Vid. nuokrypis nuo poz.:  %.2f m\n', stat_f1_vid_nuokrypis);
for i = 1:N
    fprintf(fid, '  Agentas %2d: %.2f m (žingsnis %d)\n', i, stat_f1_min_atstumas(i),
stat_f1_min_atstumas_z(i));
end
fprintf(fid, '\n--- FAZĖ 2: Sweep paieška ---\n');
fprintf(fid, 'Žingsnių skaičius:        %d\n', stat_f2_zingsniai);
fprintf(fid, 'Realus laikas:            %.1f s\n', stat_f2_zingsniai * dt);
fprintf(fid, 'Realiai apieškotas plotas:  %.0f m²\n', stat_f2_plotas_last);
fprintf(fid, 'Maks. galimas plotas:       %.0f m²\n', stat_f2_plotas_max);
fprintf(fid, 'Paieškos efektyvumas:       %.1f %%\n', (stat_f2_plotas_last /
stat_f2_plotas_max) * 100);
if obj_rast
    fprintf(fid, 'Objektą rado agentas nr.: %d\n', stat_f2_agentas);
else
    fprintf(fid, 'Objektas nerastas.\n');
end
fprintf(fid, '\n--- FAZĖ 3: Susibūrimas ---\n');
fprintf(fid, 'Žingsnių skaičius:        %d\n', stat_f3_zingsniai);
fprintf(fid, 'Realus laikas:            %.1f s\n', stat_f3_zingsniai * dt);
fprintf(fid, 'Pavojingų atstumų (<1m):  %d\n', stat_f3_pavojingi);
if ~isempty(stat_f3_pavojingi_z)
    fprintf(fid, 'Žingsniai:                %s\n',
num2str(unique(stat_f3_pavojingi_z)));
end
fprintf(fid, 'Vid. atstumas nuo lyd.:   %.2f m\n', stat_f3_vid_atstumas);
fprintf(fid, 'Galutiniai atstumai nuo lyderio:\n');
for i = 1:N3
    fprintf(fid, '  Agentas %2d: %.2f m\n', i, stat_f3_galutiniai_ats(i));
end
fprintf(fid, 'Min. atstumai tarp agentų:\n');
for i = 1:N3
    fprintf(fid, '  Agentas %2d: %.2f m (žingsnis %d)\n', i, stat_f3_min_atstumas(i),
stat_f3_min_atstumas_z(i));
end

fprintf(fid, '\n--- FAZĖ 4: Ataka ---\n');
fprintf(fid, 'Pirmas pataikymas:        %d žingsnis (%.1f s)\n', min(valid),
min(valid)*dt);
fprintf(fid, 'Paskutinis pataikymas:    %d žingsnis (%.1f s)\n', max(valid),
max(valid)*dt);
fprintf(fid, 'Visi pataikymai:\n');
for i = 1:length(stat_f4_pataikymai)
    if stat_f4_pataikymai(i) > 0
```

```
        fprintf(fid, '  Dronas %2d: %d žingsnis (%.1f s)\n', i, stat_f4_pataikymai(i),
stat_f4_pataikymai(i)*dt);
    end
end
fprintf(fid, 'Min. atstumai tarp dronų:\n');
for i = 1:N4
    fprintf(fid, '  Dronas %2d: %.2f m (žingsnis %d)\n', i, stat_f4_min_atstumas(i),
stat_f4_min_atstumas_z(i));
end
fprintf(fid, 'Objekto nujudėtas atstumas: %.1f m\n', obj_nujudejes);
fprintf(fid, '--- E(t) VERTINIMO KRITERIJUS ---\n');
fprintf(fid, 'Fazė 1: %.2f m\n', Et_f1);
fprintf(fid, 'Fazė 2: %.2f m\n', Et_f2);
fprintf(fid, 'Fazė 3: %.2f m\n', Et_f3);
fprintf(fid, '=======================================\n');
fclose(fid);
disp('✓ Statistika išsaugota: simuliacijos_statistika.txt');

%% ── FAZĖS 1 GREIČIO GRAFIKAS ──
figure('Color','w','Position',[100 100 900 400]);
hold on; grid on;
plot(1:stat_f1_zingsniai, mean(greitis_f1(:,1:stat_f1_zingsniai)), 'b-', 'LineWidth',
2);
plot(1:stat_f1_zingsniai, max(greitis_f1(:,1:stat_f1_zingsniai)), 'r-', 'LineWidth',
1);
plot(1:stat_f1_zingsniai, min(greitis_f1(:,1:stat_f1_zingsniai)), 'g-', 'LineWidth',
1);
xlabel('Žingsnis'); ylabel('Greitis (m/s)');
title('1 fazės Agentų greičio kitimas');
legend({'Vidutinis greitis','Maksimalus greitis','Minimalus greitis'},
'Location','best');
saveas(gcf, 'faze1_greitis.png');

%% ── E(t) GRAFIKAS – dvi atskiros figūros ──

% 1 pav.
figure('Color','w','Position',[100 100 700 350]);
plot(1:stat_f1_zingsniai, Et_f1(1:stat_f1_zingsniai), 'b-', 'LineWidth', 1.5);
xlabel('Žingsnis'); ylabel('E(t), m');
title('1 fazės Spiečiaus kompaktiškumo kriterijus E(t)');
grid on;
saveas(gcf, 'Et_faze1.png');

% 2 pav.
figure('Color','w','Position',[100 100 900 400]);
hold on; grid on;
plot(1:stat_f2_zingsniai, Et_f2(1:stat_f2_zingsniai), 'g-', 'LineWidth', 1.5);
plot(1:stat_f3_zingsniai, Et_f3(1:stat_f3_zingsniai), 'r-', 'LineWidth', 1.5);
xlabel('Žingsnis'); ylabel('E(t), m');
title('2 ir 3 Fazės Spiečiaus kompaktiškumo kriterijus E(t)');
legend({'2 fazė','3 Fazė'}, 'Location','best');
saveas(gcf, 'Et_faze2_3.png');

%% ── FAZĖS 4 E(t) GRAFIKAS ──
figure('Color','w','Position',[100 100 700 350]);
plot(1:stat_f4_zingsniai, Et_f4(1:stat_f4_zingsniai), 'm-', 'LineWidth', 1.5);
xlabel('Žingsnis'); ylabel('E(t), m');
title('4 fazės Spiečiaus kompaktiškumo kriterijus E(t)');
grid on;
saveas(gcf, 'Et_faze4.png');
```

```
%% ── FAZĖS 3 TRAJEKTORIJOS ──
figure('Color','w','Position',[100 100 700 700]);
hold on; grid on; axis equal;
xlabel('x, m'); ylabel('y, m');
title('3 fazės Agentų trajektorijos');
colors = lines(N3);
for i = 1:N3
    x_traj = squeeze(traj_f3(i,1,1:stat_f3_zingsniai));
    y_traj = squeeze(traj_f3(i,2,1:stat_f3_zingsniai));
    plot(obj_koord_pradzios(1), obj_koord_pradzios(2), 'gp', 'MarkerSize',14,
'MarkerFaceColor','g');
    plot(x_traj, y_traj, '-', 'Color', colors(i,:), 'LineWidth', 0.8);
    plot(x_traj(1), y_traj(1), 'o', 'Color', colors(i,:), 'MarkerSize', 6);
    plot(x_traj(end), y_traj(end), 's', 'Color', colors(i,:), 'MarkerSize', 6,
'MarkerFaceColor', colors(i,:));
    plot(naujas_lyd(1), naujas_lyd(2), 'm*', 'MarkerSize',14, 'LineWidth',2);
end
plot(naujas_lyd(1), naujas_lyd(2), 'm*', 'MarkerSize', 14, 'LineWidth', 2);
legend({'objektas','Trajektorija','Pradžia','Pabaiga','Naujasis lyderis'},
'Location','best');
saveas(gcf, 'faze3_trajektorijos.png');

%% ── FAZĖS 4 TRAJEKTORIJOS ──
figure('Color','w','Position',[100 100 700 700]);
hold on; grid on; axis equal;
xlabel('x, m'); ylabel('y, m');
title('4 fazės Agentų trajektorijos');
colors = lines(N4);
plot(hit_poz(:,1), hit_poz(:,2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);
plot(obj_koord(1), obj_koord(2), 'gp', 'MarkerSize', 14, 'MarkerFaceColor', 'g');
for i = 1:N4
    x_traj = squeeze(traj_f4(i,1,1:max(stat_f4_pataikymai(i),1)));
    y_traj = squeeze(traj_f4(i,2,1:max(stat_f4_pataikymai(i),1)));
    plot(x_traj, y_traj, '-', 'Color', colors(i,:), 'LineWidth', 0.8);
    plot(x_traj(1), y_traj(1), 'o', 'Color', colors(i,:), 'MarkerSize', 5);
end
legend({'Pataikyta','Objektas','Trajektorija','Pradžia'}, 'Location','best');
saveas(gcf, 'faze4_trajektorijos.png');


%% ============================================================
%  FUNKCIJOS
% ============================================================

function A = returnphase1(A, A_phase1, lyd, writerObj, v_sweep)
    N = size(A, 1);
    for t = 1:2000
        clf; hold on; grid on;
        axis equal; axis([1500 1500 0 1500]);
        xlabel('x, m'); ylabel('y, m');
        title(sprintf('Fazė 2 – Grįžimas į formaciją  |  Žingsnis: %d', t));
        all_home = true;
        for i = 1:N
            dx = A_phase1(i,1)-A(i,1); dy = A_phase1(i,2)-A(i,2);
            dist = hypot(dx, dy);
            if dist > 0.1
                all_home = false;
                A(i,1) = A(i,1) + v_sweep * dx/dist;
                A(i,2) = A(i,2) + v_sweep * dy/dist;
            end
        end
        plot(A(:,1), A(:,2), 'b*', 'MarkerSize',9, 'LineWidth',1.2);
```

```
        plot(lyd(1), lyd(2), 'r*', 'MarkerSize',14, 'LineWidth',2);
        plot(A_phase1(:,1), A_phase1(:,2), 'o', 'Color',[0.7 0.7 0.7], 'MarkerSize',8);
        drawnow;
        frame = getframe(gcf);
        writeVideo(writerObj, frame);
        if all_home
            disp('✓ Visi agentai grįžo į fazės 1 pozicijas.');
            break;
        end
    end
end

function skyrimas = assign_targets(A, targets)
    costs = pdist2(A, targets);
    M = matchpairs(costs, 1000);
    skyrimas = zeros(size(A,1), 1);
    for k = 1:size(M,1)
        skyrimas(M(k,1)) = M(k,2);
    end
end
```