Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Laumžirgio sparno plasnojimą atkartojančio mechanizmo tyrimas ir kūrimas

Baigiamasis magistro projektas

Laurynas Medzevičius
Projekto autorius

Prof. dr. Rimvydas Gaidys
Vadovas

Kaunas, 2026

Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Baigiamojo projekto pavadinimas

Baigiamasis magistro projektas

Mechanikos inžinerija (6211EX009)

Laurynas Medzevičius
Projekto autorius

Prof. dr. Rimvydas Gaidys
Vadovas

Lekt. dr. Birutė Narijauskaitė
Recenzentė

Kaunas, 2026

Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Laurynas Medzevičius

Baigiamojo projekto pavadinimas

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.

Laurynas Medzevičius

Patvirtinta elektroniniu būdu

Kauno technologijos universitetas

Mechanikos inžinerijos ir dizaino fakultetas

Baigiamojo magistro projekto užduotis

Studentui(-ei) – Laurynas Medzevičius

1. Projekto tema

Laumžirgio sparno plasnojimą atkartojančio mechanizmo tyrimas ir kūrimas

(Lietuvių kalba)

Research and Development of a Biomimetic Mechanism for Dragonfly Wing Flapping

(Anglų kalba)

2. Projekto tikslas ir uždaviniai

Tikslas: Sukurti plasnojimo mechanizmą skirtą atkartoti laumžirgio plasnojimą
Uždaviniai:
1. Išanalizuoti laumžirgio sparnų plasnojimo anatomiją
2. Sudaryti laumžirgio sparno judesio mechanizmo matematinį modelį
3. Sudaryti sparno judesio mechanizmo parametrų optimizavimo uždavinį
4. Atlikti sudaryto mechanizmo modeliavimą ir optimizavimą
5. Nustatyti faktorius lemiančius mechanizmo judėjimą

3. Pagrindiniai reikalavimai ir sąlygos

Plasnojimo mechanizmas turi atkartoti tris skirtingas sparno plasnojimo trajektorijas.

1. Papildomi reikalavimai projektui, ataskaitai ir jos priedams

Netaikoma

Projekto autorius	Laurynas Medzevičius		
	(Vardas, Pavardė)	*(Parašas)*	*(Data)*
Projekto vadovas	Rimvydas Gaidys		
	(Vardas, Pavardė)	*(Parašas)*	*(Data)*
Krypties studijų programų vadovas	Kęstutis Pilkauskas		
	(Vardas, Pavardė)	*(Parašas)*	*(Data)*

Laurynas Medzevičius. Laumžirgio sparno plasnojimą atkartojančio mechanizmo tyrimas ir kūrimas. Magistro baigiamasis projektas / vadovas Rimvydas Gaidys; Kauno technologijos universitetas, Mechanikos inžinerijos ir dizaino fakultetas.
Studijų kryptis ir sritis (studijų krypčių grupė): Mechanikos inžinerija, Inžinerijos mokslai.

Reikšminiai žodžiai: Plasnojimo mechanizmas, genetinis algoritmas, penkių grandžių mechanizmas, kinematika, „python“ programavimo kalba.

Kaunas, 2026. 57 p.

Santrauka

Darbo metu pirmiausia analizuojama laumžirgio anatomija ir sparnų judėjimo principai, išskiriant pagrindinius plasnojimo režimus bei jų įtaką skrydžio savybėms. Taip pat atliekama esamų plasnojimo mechanizmų apžvalga, įvertinant jų konstrukcinius sprendimus, privalumus ir trūkumus. Remiantis atlikta analize, sukurtas plasnojimo mechanizmas, paremtas penkių grandžių kinematine sistema, leidžiančia generuoti sudėtingas sparno trajektorijas. Siekiant pagerinti mechanizmo veikimą, pritaikytas genetinis algoritmas, skirtas optimizuoti geometrinius parametrus. Optimizavimo metu nustatyti parametrai, leidžiantys mechanizmui kuo tiksliau atkartoti pasirinktas trajektorijas. Atlikti eksperimentai parodė, kad algoritmas geba rasti tinkamus sprendinius, nors idealiai atkartoti trajektorijų nepavyksta dėl konstrukcinių apribojimų. Darbo rezultatai rodo, kad sukurtas mechanizmas gali būti pritaikytas bioįkvėptų skraidančių sistemų kūrime, o taikomi optimizavimo metodai yra efektyvūs sprendžiant sudėtingus inžinerinius uždavinius.

Laurynas Medzevičius. Research and Development of a Biomimetic Mechanism for Dragonfly Wing Flapping, Master's Thesis / Supervisor Rimvydas Gaidys; Kaunas University of Technology, Faculty of Mechanical Engineering and Design.

Field of study (group of fields): Mechanical Engineering, Engineering Sciences.

Keywords: Flapping mechanism, genetic algorithm, five-link mechanism, kinematics. python programing language.

Kaunas, 2026. 57 p.

Summary

During the work, the anatomy of a dragonfly and the principles of wing movement are first analyzed, identifying the main flapping modes and their influence on flight characteristics. An overview of existing flapping mechanisms is also performed, evaluating their design solutions, advantages, and disadvantages. Based on the analysis, a flapping mechanism was developed based on a five-bar kinematic system, which allows the generation of complex wing trajectories. To improve the performance of the mechanism, a genetic algorithm was applied to optimize the geometric parameters. During the optimization, parameters were determined that allow the mechanism to replicate the selected trajectories as accurately as possible. Experiments showed that the algorithm is able to find suitable solutions, although ideal replication of the trajectories is not achieved due to structural limitations. The results of the work show that the developed mechanism can be applied in the development of bio-inspired flying systems, and the applied optimization methods are effective in solving complex engineering problems.

Turinys

Paveikslų sąrašas

Įvadas

Pastaraisiais metais daugėja dėmesio bioinžineriniams sprendimams, kurie siekia atkartoti gamtoje egzistuojančius mechanizmus. Vienas iš šių pavyzdžių – laumžirgis. Laumžirgiai pasižymi išskirtiniu manevringumu, jie gali kyboti ore, staigiai keisti judėjimo kryptį, ar net skristi atbuli. Šie gebėjimai yra labai svarbūs kuriant mikro oro transporto priemones. Šių technologijų vystymas užtikrina efektyvesnį stebėjimo, gelbėjimo ir kitų panašių sričių tobulėjimą. Nepaisant sparčios technologijų pažangos, sukurti mechanizmą, gebantį atkartoti laumžirgio sparnų judėjimą, išlieka sudėtingu inžineriniu uždaviniu. Tam būtina ne tik išanalizuoti vabzdžio anatomiją ir skrydžio kinematiką, bet ir pritaikyti tinkamus mechaninius sprendimus bei optimizavimo metodus.

Darbo tikslas: sukurti plasnojimo mechanizmą, skirtą atkartoti laumžirgio plasnojimą.

Tikslui pasiekti buvo išsikelti penki uždaviniai:

1. Išanalizuoti laumžirgio sparnų plasnojimo anatomiją
2. Sudaryti laumžirgio sparno judesio mechanizmo matematinį modelį
3. Sudaryti sparno judesio mechanizmo parametrų optimizavimo uždavinį
4. Atlikti sudaryto mechanizmo modeliavimą ir optimizavimą
5. Nustatyti faktorius lemiančius mechanizmo judėjimą

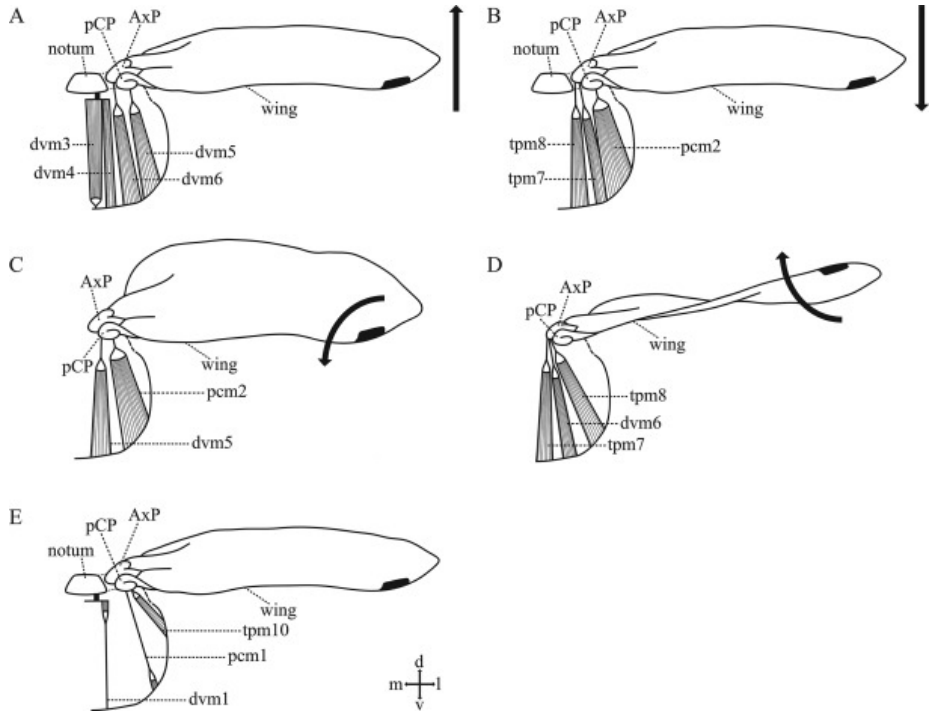
Pasiekti užsibrėžtam tikslui buvo kuriamas genetinis algoritmas bei kinematinis modelis naudojant „Python“ programavimo kalbą ir „Visual Studio“ programinę aplinką.

1. Laumžirgio anatomijos analizė

Laumžirgis – vabzdys, turintis išskirtinių skrydžio gebėjimų. Jis turi dvi poras sparnų – priekinius ir galinius, kuriuos gali judinti nepriklausomai vienas nuo kito (1). Ši savybė leidžia jam atlikti itin sudėtingus ir greitus manevrus ore: sustoti, kaboti, skristi atgal, kilti tiesiai aukštyn ar net apsisukti vietoje.

1.1. Sparną valdantys raumenys

Pagrindinis skirtumas tarp laumžirgio ir kitų vabzdžių, pavyzdžiui, bičių, yra tai, kad laumžirgiai sparnus judina naudodamiesi tiesioginiais raumenimis (2). Tai reiškia, kad raumenys yra tiesiogiai pritvirtinti prie sparnų pamatų, todėl kiekvienas sparnas gali būti valdomas atskirai. Toks mechanizmas suteikia didesnę judesių įvairovę ir leidžia itin tiksliai reguliuoti skrydžio kryptį ir greitį (žr. 1 pav).



1 pav. Laumžirgio sparno raumenų grupės [2]

Laumžirgio raumenis galima skirstyti į tris raumenų grupes (2). Pirmoji, didžiausia raumenų grupė, yra plasnojimo raumenys, kurie yra atsakingi už sparno judėjimą aukštyn ir žemyn (2). Šie raumenys yra patys stambiausi ir jungiasi prie vabzdžio krūtinės bei nugaros (3)(4). Kaip parodyta pirmame paveikslėlyje, atsipalaiduodami dvm3 ir dvm4 raumenys judina sparną aukštyn. Raumenys tpm7, tpm8, pcm2 susitraukdami judina sparną žemyn. Šių sparnų judėjimas aukštyn ir žemyn sukuria keliamąją jėgą.

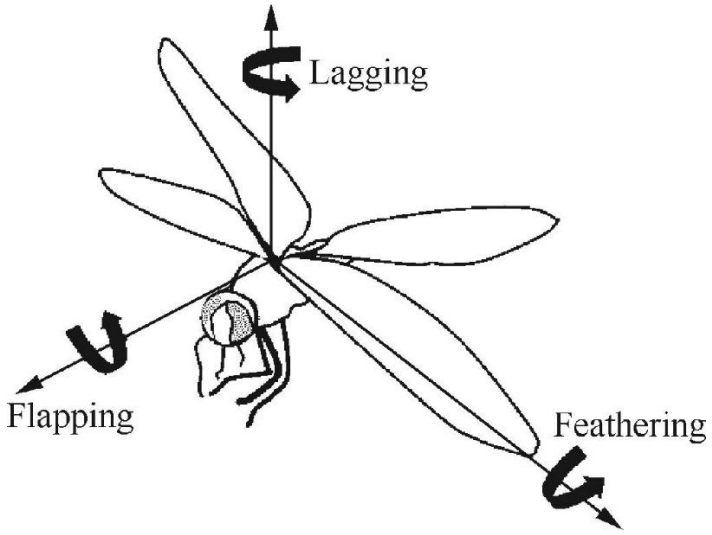
Antroji raumenų grupė pcm2, dvm5, dvm6, tpm7, tpm8 yra atsakinga už sparno atakos kampą. Verta paminėti, jog anksčiau minėti raumenys taip pat prisideda prie šio sparno judesio. Tuo tarpu raumenys pcm2 ir dvm5 susitraukdami netolygia jėga verčia sparną žemyn, o raumenys tpm7, dvm6, tpm8 traukdamiesi netolygiai verčia sparną i viršų.

Trečioji raumenų grupė yra atsakinga už sparnų ir jų judesių stabilumą. Raumuo dvm1 yra atsakingas už krūtinės stabilumą didelių apkrovų metu, kuomet krūtinę tempia ne vienas stambus raumuo. Raumenys tpm10 ir pcm1 padeda tiksliai sureguliuoti sparno atakos kampą (5).

1.2. Laumžirgio plasnojimo analizė

Skrydžio metu laumžirgis gali pasirinkti skristi sinchroniškai – kai abi sparnų poros juda vienu metu – arba asinchroniškai, kai priekiniai ir galiniai sparnai plasnoja priešingu ritmu (6). Laumžirgis kybo ore kuomet abi sparnų poros juda asinchroniškai, taip yra sukuriamas oro srautas, kuris leidžia laumžirgiui stabiliai laikytis ore (6). Taip pat laumžirgis gali skristi atgal – tai pasiekiama sinchroniškai judinant sparnus taip, kad sukurtas oro srautas nukreiptų vabzdį atgal (6). Toks manevras reikalauja labai tikslios raumenų kontrolės ir aerodinaminio suderinimo. Tokie skrydžio ypatumai yra itin naudingi grobio medžioklei ore arba vengiant plėšrūnų.

Laumžirgio sparnai juda trimis laisvės laipsniais – plasnoja aukštyn ir žemyn, juda pirmyn ir atgal, arba sukinėjasi aplink savo ašį (žr. 2 pav.).

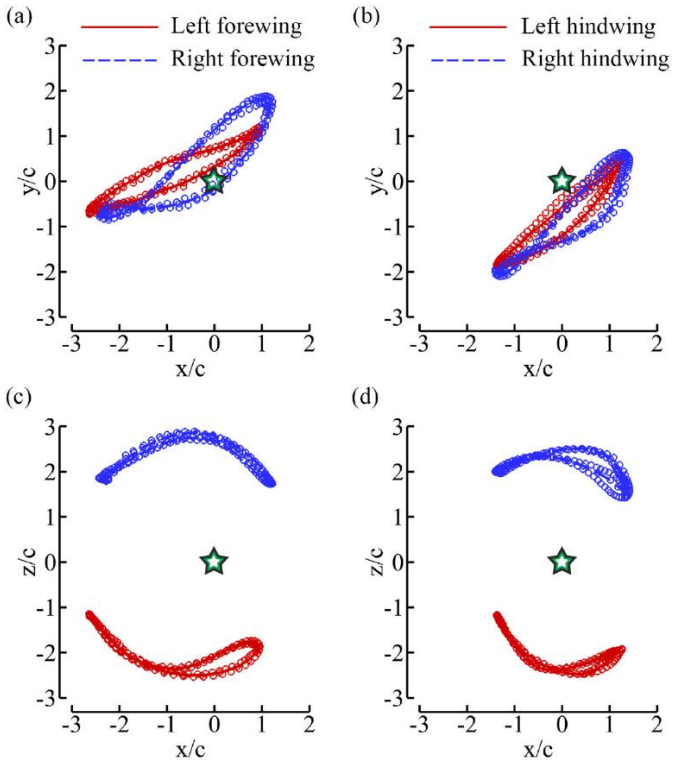


2 pav. Laumžirgio sparno judėjimas [6]

Be to, kadangi laumžirgiai turi stiprius krūtinės raumenis, jie leidžia ilgai išlaikyti aktyvų skrydį, net ir esant didelei apkrovai (7). Jų aerodinaminė forma, ilgi ir standūs sparnai bei puiki nervų sistema sudaro vieną sudėtingiausių natūralios inžinerijos pavyzdžių vabzdžių pasaulyje (7). Laumžirgio sparnų raumenys yra kruopščiai išsidėstę krūtinėje ir yra atsakingi už tikslius judesius, leidžiančius šiam vabzdžiui atlikti tiek tiesinį skridimą, tiek staigius posūkius ar net kabėti ore vietoje.

Vienas svarbiausių dalykų nagrinėjant laumžirgio plasnojimą yra išsiaiškinti, kokius judesius laumžirgis daro plasnodamas. Tai ilgą laiką buvo neįmanoma, nes laumžirgis plasnoja iki 70Hz dažniu ir žmogaus akiai tikslių judesių pastebėti neįmanoma (7). Tačiau turint dabartines technologijas, tokias kaip aukštos spartos kameros, plasnojimo judesius galima tiksliai atpažinti. Naudojant aukštos spartos kameras, galima pastebėti, kad laumžirgis turi tris plasnojimo režimus – aštuoneto, dvigubo aštuoneto ir kilpos (8). Šios formos yra nustatomos stebint kokią trajektoriją išbrėžia sparno galiukas (8). Kiekvienas režimas turi savo paskirtį ir yra labai svarbus laumžirgio skrydžiui. Laumžirgio staigiam pakilimui arba krovinio nešimo judesiams, kurie reikalauja didelės keliamosios jėgos, laumžirgis naudoja kilpos judesį (8). Kai laumžirgiui reikia greitai judėti į priekį,

jis naudoja aštuoneto plasnojimo trajektorijos judesį (8). Kuomet laumžirgis keičia savo skrydžio kryptį, prie aštuoneto judesio jis prideda dar vieną papildomą kilpą, taip gaudamas dvigubo aštuoneto mostą (9)(10). Visus šiuos judesius laumžirgis gali atlikti su skirtingais sparnais vienu metu, taip taisant ir staigiai keičiant skridimo kryptį (žr 3 pav.).



3 pav. Laumžirgio sparno galiuko trajektorija plasnojant [8]

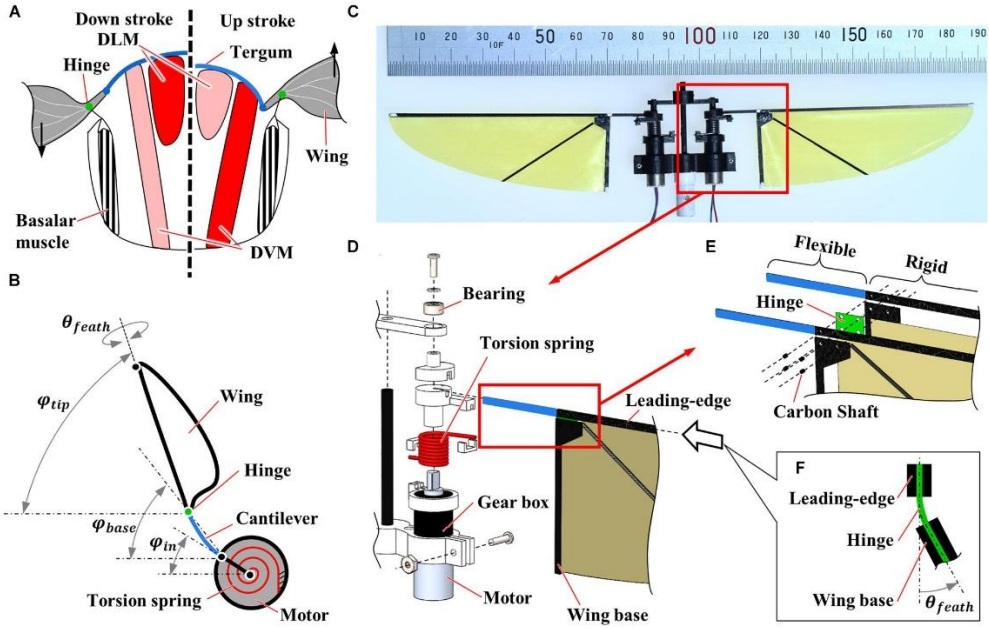
Šioje iliustracijoje pavaizduotos vidutinės laumžirgio sparnų galiukų trajektorijos, apskaičiuotos per tris sparnų plazdėjimo ciklus ir pateiktos kūno atskaitos sistemoje. Viršutinėse dalyse (a) ir (b) parodytos trajektorijos žiūrint iš šono, atitinkamai priekiniams ir užpakaliniams sparnams, kur raudona linija žymi kairįjį, o mėlyna – dešinįjį sparną. Apatinėse dalyse (c) ir (d) pateikti tie patys duomenys žiūrint iš viršaus, leidžiantys aiškiai matyti sparnų judėjimo asimetriją posūkio metu. Žvaigždutė kiekviename grafike žymi laumžirgio kūno masės centrą. Trajektorijų forma atskleidžia, kad vidiniai ir išoriniai sparnai juda skirtingomis erdvinėmis trajektorijomis, kas patvirtina asimetrišką sparnų kinematiką ir jos svarbą manevruojant skrydžio metu.

2. Esamų plasnojimo mechanizmų apžvalga

Ši darbo dalis skirta išanalizuoti jau esamus plasnojimo mechanizmus, išsiaiškinti jų sukuriamus plasnojimo judesius, valdymo variklių tipus ir veikimo principus.

2.1. Motorinis lankstus plasnojimo mechanizmas

Motorinis lankstus plasnojimo mechanizmas yra lengva, efektyvi sistema, skirta imituoti vabzdžių sparnų judesius (11). Jį sudaro trys pagrindinės dalys: tiesiogiai sparną varantis mikro variklis, torsinė spyruoklė ir lankstus sparnas su vyriu (žr. 4 pav.). Mechanizmas veikia rezonanso principu – sparno inercija įtempia spyruoklę, kuri grąžina energiją atgal į judesį, taip sumažindama variklio apkrovą ir energijos sąnaudas.



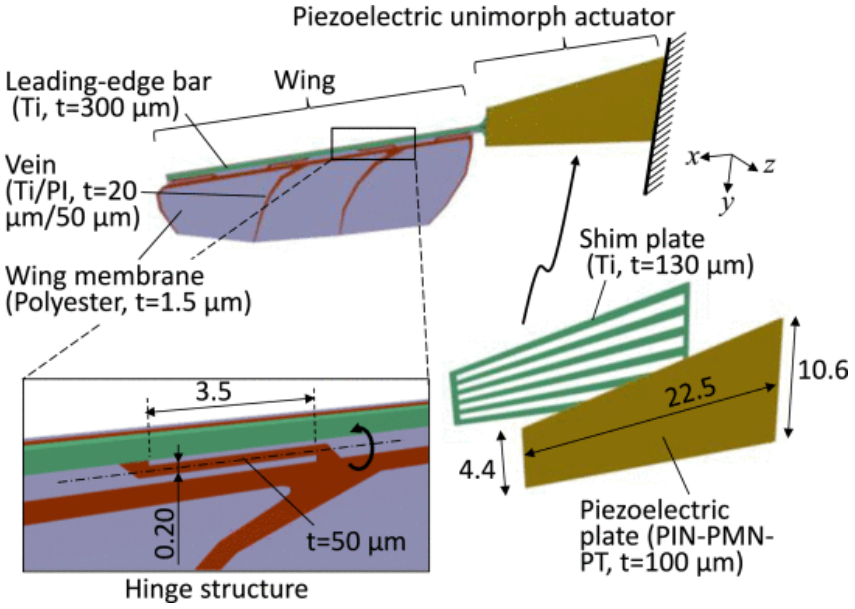
4 pav. plasnojimo konstrukcija su lanksčiomis jungtimis [11]

Mechanizmas turi tris elastingus komponentus: torsinę spyruoklę, lankstų kotą ir vyrius. Tinkamai parinkus jų standumą ir darbo dažnį, sistema gali veikti rezonanse ir efektyviai generuoti keliamąją jėgą, viršijančią savo svorį. Lankstūs komponentai pagerina ne tik efektyvumą, bet ir atsparumą išoriniams trikdžiams, pavyzdžiui, priekinėms ar šoninėms oro srovėms. Pasyvus sparno prisitaikymas (angl. „feathering“) leidžia sumažinti traukos ir sukamąsias jėgas, atsirandančias dėl vėjo. Tokia reakcija sumažina sparnų pasipriešinimą ir sukimo momentus, taip didindama skrydžio stabilumą. Pastebėta kad, tarp efektyvumo ir stabilumo – „kietesnės“ spyruoklės padidina atsparumą trikdžiams, bet sumažina energinį efektyvumą. Todėl mechanizmo standumas turėtų būti pasirenkamas atsižvelgiant į planuojamą darbo aplinką – ramų ar vėjuotą orą (11).

2.2. Robotas, naudojantis tiesiogiai prijungtą pjezo elementą

Šiame tyrime nagrinėjamas tiesiogiai varomas plasnojantis sparnas, sudarytas tik iš sparno ir trapecinės pjezo unimorfinės pavaros. Sparnas susideda iš titaninio priekinio krašto, lanksčios poliamidinės gyslos ir poliesterinės plėvelės. Netoli priekinio krašto įrengta lanksto struktūra leidžia sparnui atlikti pasyvų pakrypimo judesį, kuris svarbus keliamosios jėgos generavimui. Pjezo medžiagai pasirinkta PIN-PMN-PT dėl itin didelio pjezo efekto koeficiento ir tamprumo, leidžiančių

pasiekti dideles deformacijas. Unimorfinė gembė sukurta klijuojant pjezoelektrinę plokštelę prie titano lakšto. Kai įjungiama įtampa, piezo elementas išsilenkia, sukeldamas sparno plasnojimo judesį. Tuo pačiu metu, dėl oro slėgio ir inercijos, atsiranda pasyvus pakrypimo judesys (žr. 5 pav.). Sparno ilgis – 32,4 mm, o visa pavaros sistema sveria tik 217 mg, todėl mechanizmas pasižymi itin mažu svoriu ir paprasta struktūra (12).

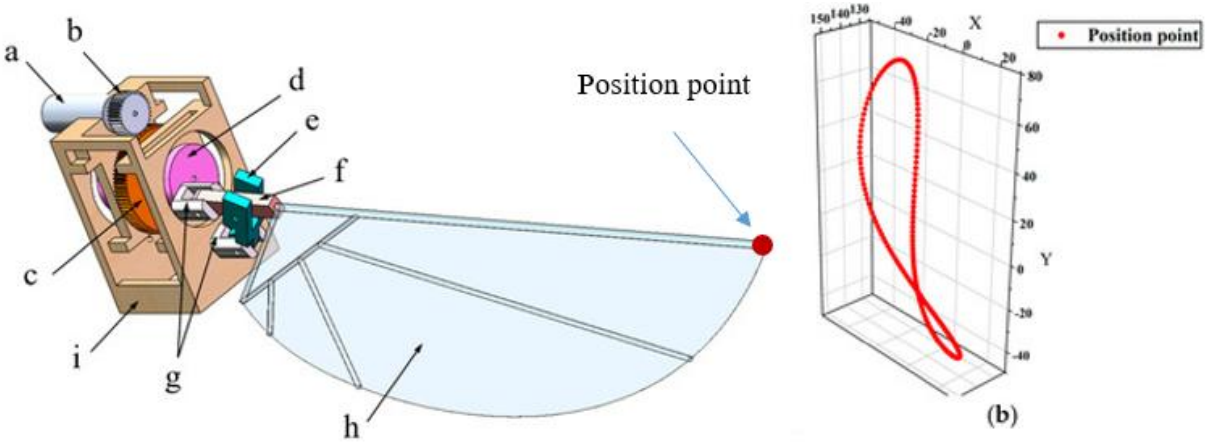


5 pav. Plasnojančio roboto sparnas, naudojant tiesiogiai prijungtą piezo elementą [12]

Šiame tyrime buvo pademonstruotas skraidančio mikro roboto pakilimas, naudojant tiesioginę pjezoelektrinę pavarą. Prototipas sukūrė 6,52 mN keliamąją jėgą ir svėrė 598 mg, o jo traukos ir svorio santykis siekė 1:1. Nors mechanizmas pasižymi paprastumu, kuris gali palengvinti praktinį roboto panaudojimą, išlieka keletas iššūkių. Tarp jų – brangi pjezoelektrinė medžiaga bei ilgalaikio patikimumo klausimai dėl didelių deformacijų. Siekiant nepriklausomos (be laido) skrydžio galimybės ir didesnio manevringumo, būtina padidinti keliamąją jėgą, sumažinti kūno virpesius ir integruoti stabilumo valdymo sistemą.

2.3. Laumžirgio plasnojimo mechanizmas su vienu varikliu

Šiame tyrime aprašytas sudėtingas plasnojimo mechanizmas, kuris imituoja laumžirgių, sparnų judėjimą. Mechanizmo veikimas pagrįstas sukamąja variklio jėga, kuri per krumpliaračių sistemą perduodama transmisijos velenui. Velenas yra sujungtas su dviem svirtimis (jungiamaisiais elementais), kurios leidžia sparnui atlikti dvi pagrindines judėjimo kryptis – vertikalų plasnojimą ir pasvyrimą, kuris keičia sparno atakos kampą. Tokiu būdu sukuriamas suderintas sparno judesys, būtinas efektyviam keliamosios jėgos generavimui. Ši konstrukcija leidžia efektyviai atkartoti natūralius vabzdžių skrydžio mechanizmus bei yra tinkama naudoti mikro oro transporto priemonėse (angl. micro air vehicle, toliau – MAV). (žr. 6 pav.).

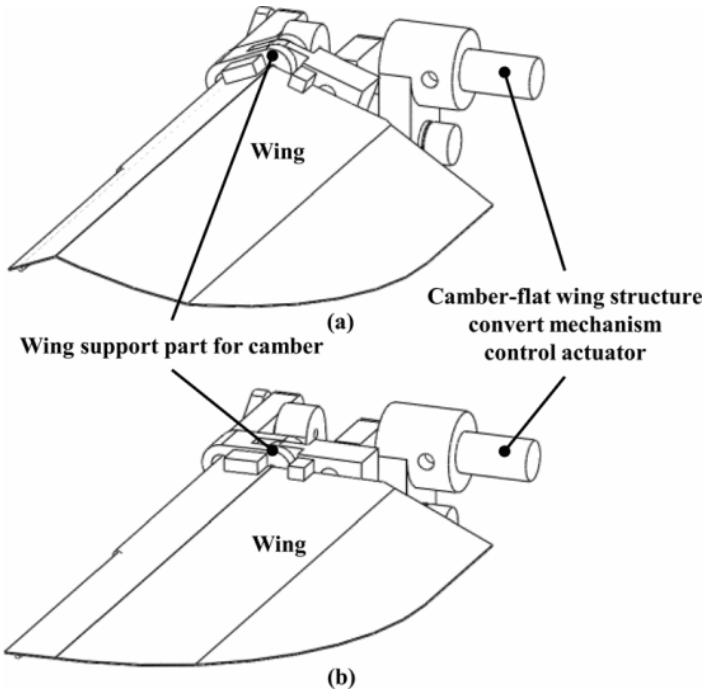


6 pav. Kairėje – laisvės laipsnių plasnojimo mechanizmas su vienu varikliu, dešinėje – sparno judėjimo trajektorija [13]

Remiantis laumžirgio skrydžio analizėmis, buvo sukurtas „smūgio-pasvyrimo“ judėjimo modelis, leidžiantis mechanizmui tiksliai imituoti daugumą vabzdžio skrydžio modelių. Norint įvertinti asimetrinių judesių poveikį, buvo įvesti du nauji parametrai: pasvyrimo nuokrypio kampas ir smūgio vidurio kampas. Skaitmeniniai modeliavimai parodė, kad, keičiant šiuos parametrus, galima reikšmingai keisti keliamąją ir stūmos jėgas – pavyzdžiui, sumažinus nuokrypio kampą 10°, keliamoji jėga padidėja daugiau nei keturis kartus (13). Taip pat nustatyta, kad smūgio vidurio kampo keitimas leidžia optimizuoti keliamąją arba stūmos jėgą pagal poreikį.

2.4. Laumžirgio plasnojimo mechanizmas, galintis keisti sparno formą

Šiame tyrime sukurtas MAV prototipas su mechanizmu, skirtu sparno išlinkimui keisti; jis leidžia sparnui aktyviai keisti formą skrydžio metu. Mechanizmas turi dvi laisvės laipsnių kryptis: viena skirta sparno plasnojimui, kita – sparno išlinkimo keitimui. Plasnojimo judesį generuoja EC6 variklis su 15:1 pavarų dėže, kuri suka alkūnę, o sparno išlinkimo valdymui naudojamas LCP06 variklis su 26:1 pavarų dėže, kuri judina rutulinį sraigtą. Sistema atspausdinta 3D spausdintuvu iš ABS plastiko. Sparno karkasas pagamintas iš anglies strypų: nuo šaknies standus 1,5 mm, o toliau – lankstesni 0,6 mm, kad būtų imituojamas tikrojo laumžirgio sparnų pasyvusis atakos kampas. Membrana sudaryta iš 25 mikrometrų PET plėvelės. Šis konstrukcinis sprendimas leidžia sparnui pasyviai suktis smūgio krypties keitimo metu, padidinant keliamąją jėgą ir sumažinant reakcinį sukimo momentą (žr. 7 pav.). Taigi, šis mechanizmas ne tik atlieka sparno plasnojimą, bet ir aktyviai valdo sparno formą skrydžio metu (14).



7 pav. Laumžirgio plasnojimo mechanizmas, galintis keisti sparno formą [14]

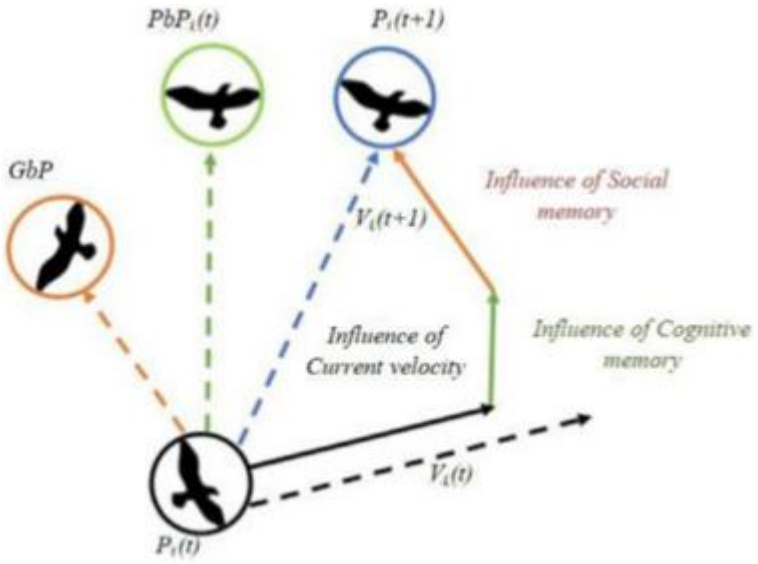
Šiame tyrime parodyta, kad MAV gali keisti sugeneruotos jėgos kryptį ir dydį vien tik modifikuodama sparno struktūrą, nekeisdama korpuso padėties ar smūgio kampo. Lyginant lenktą ir plokščią sparno struktūrą, nustatyta, kad lenktas sparnas yra stabilesnis kybojimo režime, o plokščias – efektyvesnis skrendant į priekį. Tai reiškia, kad MAV su plokščiu sparnu privalo keisti kūno padėtį keičiant skrydžio režimą.

3. Optimizavimo algoritmų analizė mechaninių sistemų projektavime

Žinant, kaip veikia laumžirgio plasnojimas ir kokių yra sukurtų plasnojimo mechanizmų, paskutinė dalis, kurią reikia apžvelgti, yra mechanizmo optimizavimo algoritmai. Šie algoritmai yra vienas iš efektyviausių būdų ieškant idealiausio užduoties sprendinio. Kadangi darbe optimizavimo algoritmas bus naudojamas nustatyti optimaliausius grandžių ilgius, algoritmų apžvalgoje bus analizuojami algoritmai, pritaikomi grandžių sintezės optimizavimui.

3.1. Dalelių spiečiaus optimizavimas

Dalelių spiečiaus optimizavimas (angl. particle swarm optimization, toliau – PSO) yra algoritmas, pagrįstas atsitiktinio optimizavimo metodu. Algoritmas veikia kaip supaprastintas socialinis modelis, kuriame dalelės juda kaip paukščių arba žuvų būriai. Šis algoritmas neturi klasikinių genetinio algoritmo operatorių (15), o jo sprendinių ieškojimas yra grindžiamas dviejų lygių informacija: individualios dalelės patirtimi ir viso būrio patirtimi (žr. 8 pav.). Dėl to šis metodas yra paprastas, lengvai įgyvendinamas ir turintis nedaug derinamų parametrų.



8 pav. Dalelių greičio keitimo atvaizdavimas [19]

PSO algoritmas yra iteracinis ir įgyvendinamas devyniais žingsniais(16)(17)(18):

1. Pirmiausia algoritmas sugeneruoja pradinę dalelių populiaciją. Populiacijos dydis priklauso nuo uždavinio sudėtingumo. Dalelėms atsitiktinai parenkami greičiai ir padėtys.
2. Tuomet, kiekviena dalelė yra įvertinama su tinkamumo funkcija. Ši funkcija įvertina, kiek kiekviena dalelė (sprendinys) yra tinkamas atsižvelgiant į duotą uždavinį. Šiame etape visos dalelės yra įvertinamos.
3. Įvertinus visas daleles, lyginama sprendinių kokybė su anksčiau įvertintais sprendiniais. Kiekviena dalelė saugo informaciją apie geriausią savo rastą sprendinį. Po kiekvienos iteracijos šis sprendinys yra palyginamas su naujausiu sprendiniu. Jeigu naujausias sprendinys yra geresnis už ankstesnį, jis yra įrašomas, kaip geriausias.
4. Sekantis palyginimas yra ieškant geriausio rezultato iš viso būrio. Algoritmas saugo geriausią sprendinį rastą visame būryje. Jeigu paskutinės iteracijos sprendinys yra geresnis už prieš tai buvusį, jis įrašomas kaip geriausias.

5. Sekantis algoritmo etapas – naujos judėjimo krypties ir greičio nustatymas. Greitis nustatomas atsižvelgiant į ankstesnį dalelės judėjimą, geriausią individualiai rastą sprendinį ir geriausią viso būrio rastą poziciją. Šis būdas leidžia ieškoti sprendinio derinant individualią patirtį su kolektyvine informacija.
6. Algoritmui radus naujus greičius, dalelių pozicijos yra atnaujinamos.
7. Atnaujinus pozicijas, tikrinama, ar dalelės neišeina už ribų. Jeigu taip atsitinka – dalelės yra apgręžiamos arba atstatomos į buvusią poziciją.
8. Atlikus visus šiuos veiksmus, algoritmas pradeda naują iteraciją. Iteracijos kartojamos tol, kol pasiekiamas norimas rezultatas, pasiekiamas maksimalus iteracijų skaičius arba sprendinys nebesikeičia.
9. Pasibaigus iteracijoms, algoritmas pateikia geriausią populiacijos sprendinį.

Šio algoritmo privalumai yra tie, kad jį yra lengva derinti, nes jis turi mažiau kintamųjų negu genetinis algoritmas. Visgi, jo neigiama savybė yra ta, kad algoritmas gali pastrigti vienoje srityje nebesugeneruodamas naujų pasiūlymų (19).

3.2. Diferencinė evoliucija

Diferencinės evoliucijos algoritmas priklauso evoliucinių algoritmų šeimai. Šis algoritmas yra paprastas, lengvai įgyvendinamas ir efektyvus sprendžiant nepertraukiamo optimizavimo uždavinius (20). Algoritmas remiasi trijų pagrindinių operatorių seka: mutacija, kryžminimas, atranka (21). Algoritmas įgyvendinamas penkiais etapais (22)(23):

1. Algoritmas startuoja sukurdamas pradinę populiaciją. Populiacija yra pasirenkama pagal uždavinio sudėtingumą. Populiaciją sudaro parametrų rinkinys. Populiacija sudaroma atsitiktinai renkantis parametrus nustatytos ribose.
2. Sekantis etapas yra mutacija; šis etapas yra svarbiausias algoritme, nes jo metu yra kuriami bandomieji individai. Jie kuriami pasirenkant atsitiktinius populiacijos individus. Iš pradžių yra pasirenkamas bazinis sprendinys iš populiacijos, kuris bus modifikuojamas. Tuomet pasirenkami dar du atsitiktiniai sprendiniai. Pasirinkus du atsitiktinius sprendinius, yra apskaičiuojamas jų skirtumas. Gautas skirtumas yra padauginamas iš algoritme nurodyto koeficiento. Šis skirtumas yra pridedamas prie modifikuojamo individo, taip gaunant naują individą (mutantą).
3. Tuomet algoritmas atlieka kryžminimo etapą. Šio etapo metu pradinis individas yra sukryžminamas su mutantu. Kiekvienam individo parametrui yra generuojamas atsitiktinis skaičius, kuris yra lyginamas su algoritme nustatyta kryžminimo tikimybe. Jeigu atsitiktinė reikšmė yra mažesnė arba lygi kryžminimo tikimybei, naujas individas paveldi mutanto parametrą. Kitu atveju individas pasilieka senąjį parametrą. Siekiant užtikrinti evoliuciją, bent vienas atsitiktinai parinktas parametras visada paimamas iš mutanto. Taip suformuojamas naujas individas.
4. Sekantis etapas yra atranka. Atrankos metu senasis individas yra lyginamas su naujuoju. Į kitą etapą patenka geresnis individas.
5. Paskutinis etapas yra iteracijos kartojimas. Po kiekvienos iteracijos sprendimas tampa vis geresnis. Algoritmas kuria naujas iteracijas tol, kol yra pasiekiamas norimas rezultatas, pagerėjimai tampa labai maži, arba pasiekiamas maksimalus iteracijų skaičius.

Diferencinės evoliucijos metodas vertinamas kaip patikimas optimizavimo būdas, ypač kai tenka ieškoti geriausio sprendimo sudėtinguose uždaviniuose. Didžiausias jo privalumas yra gebėjimas

savaime reguliuoti žingsnius – kai populiacija juda arčiau tikslo, mutacijų intensyvumas natūraliai silpnėja. Metodui nereikia žinoti funkcijos išvestinių, todėl jis puikiai veikia su optimizavimo problemomis, kurių vidinė funkcijos struktūra visiškai nežinoma. Visgi, metodas turi ir trūkumų – pavyzdžiui, algoritmas gali pareikalauti nemažai procesinės galios. Jei valdymo parametrai netinkamai nustatomi, procesas gali per anksti sustoti ties lokaliu maksimumu ar minimumu (24)(25).

3.3. Genetinis algoritmas

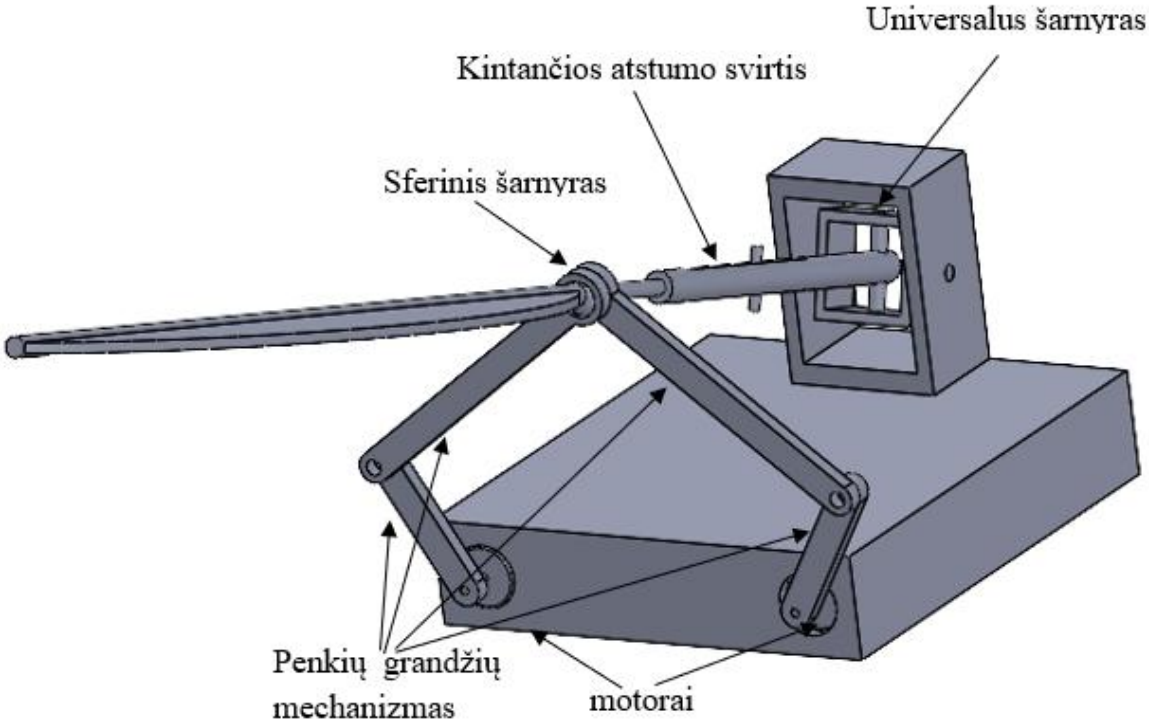
Genetinis algoritmas yra optimizavimo metodas, paremtas biologinės evoliucijos principu. Šis algoritmas yra tinkamas spręsti uždavinius, kurie turi labai didelius sprendinių kiekius. Kuomet tradiciniai matematiniai metodai tampa neefektyvūs, genetinis algoritmas yra vienas tinkamiausių būdų optimizuoti sistemą (26). Algoritmas veikia natūralios atrankos metodu ir susideda iš septynių pagrindinių etapų:

1. Sukuriama pradinė populiacija. Ji kuriama iš atsitiktinių parametrų, kitaip vadinamų chromosomomis. Populiacijos dydis turi įtakos sprendinių kokybei, tačiau kuo didesnė populiacija, tuo daugiau skaičiavimų resursų reikia.
2. Tuomet kiekvienas populiacijos individas yra vertinamas pagal tinkamumo funkciją. Ši funkcija nustato individo sprendinio kokybę. Ši funkcija yra viena svarbiausių algoritmo dalių, nes nuo jos priklauso sprendinio tikslumo įvertinimas.
3. Įvertinus individus, kitas etapas yra atranka. Šio etapo metu individai yra lyginami vienas su kitu pagal sprendinio kokybę. Geriausi individai yra atrenkami į kitą etapą.
4. Atrinkti individai yra kryžminami vienas su kitu. Šio proceso metu imami individai pasirinkti atsitiktiniu būdu. Jų sprendiniai yra padalijami ir iš jų sudaromi nauji du palikuonys.
5. Sekančiame etape yra atsitiktinai pasirenkami palikuonys, kurių genai yra modifikuojami. Šio proceso tikslas yra išlaikyti populiacijos įvairovę.
6. Atlikus mutaciją, yra sukuriama nauja populiacija. Prie naujos populiacijos papildomai pridedamas geriausias rastas sprendinys tam, kad algoritmas išlaikytų sprendinių kokybę.
7. Algoritmas yra stabdomas, kuomet sprendinys atitinka reikalavimus, yra įvykdomas užduotų iteracijų skaičius arba sprendinys daugiau nebetobulėja (27).

Apibendrinant, genetinis algoritmas leidžia efektyviai rasti sudėtingų uždavinių sprendimus, kurių kiti algoritmai nesugeba pasiekti. Nors šis procesas negarantuoja geriausio sprendinio radimo, tačiau jis gali surasti sprendinį, kuris yra pakankamai tikslus (28) (29) (30).

4. Plasnojimo mechanizmo kinematika ir matematinis modeliavimas

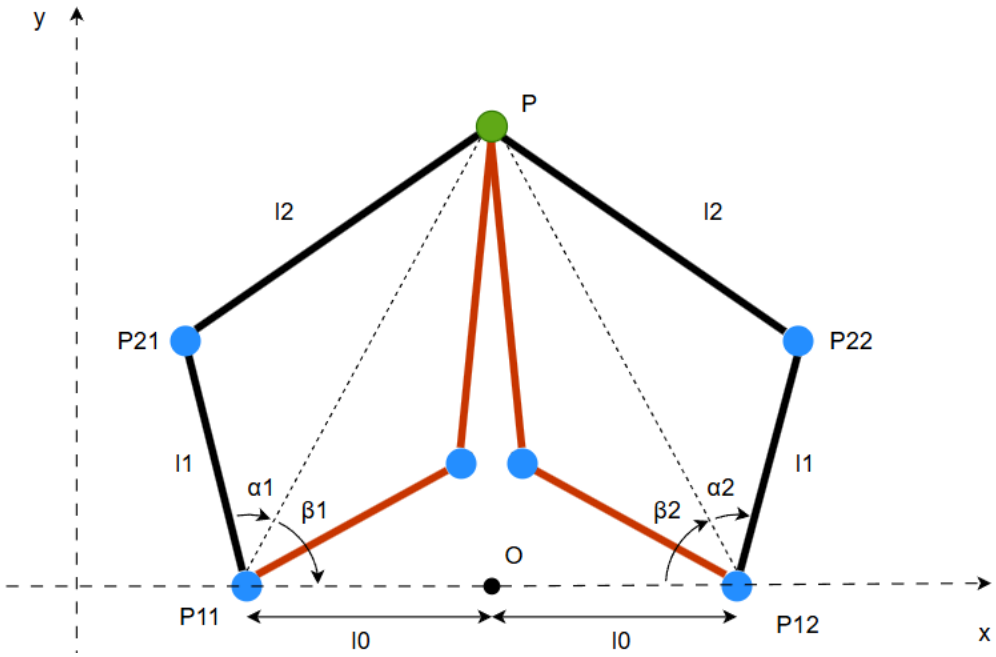
Atlikus literatūros analizę ir apžvelgus esamus mechanizmus, buvo nuspręsta mechanizmą kurti taip, kad jis atkartotų kuo panašesnius plasnojimo judesius į laumžirgio plasnojimą. Šiam tikslui pasiekti buvo sugalvotas mechanizmas, kurio valdančioji dalis yra penkių grandžių mechanizmas (žr. 9 pav.). Šį mechanizmą valdo du elektriniai varikliai, kurie, keičiant savo sukimosi greičius, reguliuoja sparno poziciją. Mechanizme sparnas yra įtvirtintas dviejose vietose, sparno šaknis yra įtvirtinta ant universalaus šarnyro. Sekantis tvirtinimo taškas yra ties penkių grandžių mechanizmu. Sparnas su penkių grandžių mechanizmu yra sujungiamas sferiniu šarnyru. Sparnas mechanizme sudarytas iš dviejų dalių: sparno plasnojimo dalies ir kintančio atstumo svirties. Kintančio atstumo svirtis reikalinga, nes judant penkių grandžių mechanizmui tarp sferinio šarnyro ir universalaus šarnyro atstumas kinta.



9 pav. Plasnojimo mechanizmas

4.1. Penkių grandžių mechanizmo kinematikos analizė

Viena iš pagrindinių plasnojimo mechanizmo dalių yra penkių grandžių mechanizmas. Iš pradžių yra nagrinėjamas atvirkštinės kinematikos modelis. Šio modelio tikslas – rasti valdymo kampus, žinant galutinio taško x, y koordinates (žr. 10 pav.).



10 pav. Penkių grandžių mechanizmas

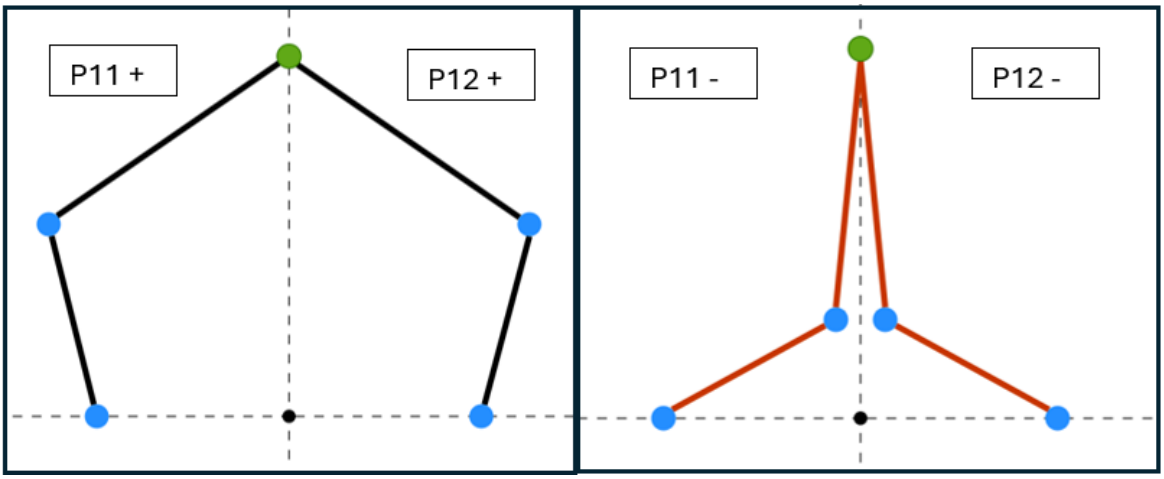
Šio mechanizmo pasukimo kampams rasti reikia keturių pagrindinių formulių:

$$\alpha_1 = \operatorname{acos}\left(\frac{l_1^2+((l_0+x)^2+y^2)-l_2^2}{2l_1\sqrt{(l_0-x)^2+y^2}}\right) \quad (1)$$

$$\alpha_2 = \operatorname{acos}\left(\frac{l_1^2+((l_0-x)^2+y^2)-l_2^2}{2l_1\sqrt{(l_0-x)^2+y^2}}\right) \quad (2)$$

$$\beta_1 = \operatorname{atan}\left(\frac{y}{l_0+x}\right) \quad (3)$$

$$\beta_2 = \operatorname{atan}\left(\frac{y}{l_0-x}\right) \quad (4)$$



11 pav. Penkių grandžių mechanizmo konfigūracijos

Apskaičiavus pasukimo kampus, kitas žingsnis – apskaičiuoti galimas pasukimo konfigūracijas. Pasukimo konfigūracijos (žr. 11 pav.) gali būti keturios skirtingos. Mechanizmo grandys gali būti

išdėstomos taip, kaip juodos grandys (11 pav. kairė pusė) arba taip, kaip raudonos (11 pav. dešinė pusė). Konfigūracijoms apskaičiuoti naudojamos keturios formulės:

(P11 + konfigūracija): $\theta_{a1} = \beta_1 + \alpha_1$ (5)

(P11 - konfigūracija): $\theta_{a1} = \beta_1 - \alpha_1$ (6)

(P12 + konfigūracija): $\theta_{a2} = \pi - \beta_2 + \alpha_2$ (7)

(P12 - konfigūracija): $\theta_{a2} = \pi - \beta_2 - \alpha_2$ (8)

Sekantis etapas, apskaičiavus konfigūracijas, yra rasti taškų P21 ir P22 koordinates. Norint rasti šias koordinates, naudojamos keturios formulės:

$P_{21}(x) = -l_0 + l_1 \cos(\theta_{a1})$ (9)

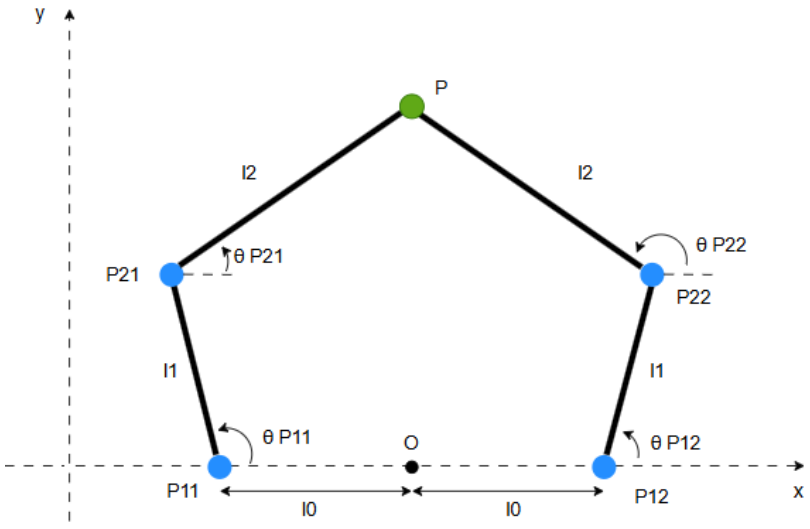
$P_{21}(y) = l_1 \sin(\theta_{a1})$ (10)

$P_{22}(x) = -l_0 + l_1 \cos(\theta_{a2})$ (11)

$P_{22}(y) = l_1 \sin(\theta_{a2})$ (12)

Žinant šias formules, galima nustatyti, kaip reikia susukti variklius, kad mechanizmas pasiektų užduotas koordinates. Tai padės vėliau norint rasti mechanizmo darbinį plotą, galimas konfigūracijas pasiekti tašką.

Sekantis modelis, kurį reikia sužinoti analizuojant penkių grandžių mechanizmą, yra tiesinė kinematika. Šis skaitinis modelis leidžia nustatyti, kokias koordinates mechanizmas pasieks, kuomet yra žinomi valdymo kampų dydžiai (pav. 12).



12 pav. Penkių grandžių mechanizmo schema tiesioginei kinematikai skaičiuoti

Formulės, skirtos rasti x koordinatę:

$x = l_0 + l_1 cos\theta_{P12} + l_2 cos\theta_{P22}$ (13)

$$x = -l_0 + l_1 cos\theta_{P11} + l_2 cos\theta_{P21} \quad (14)$$

Formulės, skirtos rasti y koordinatę:

$$y = l_1 sin\theta_{P12} + l_2 sin\theta_{P22} \quad (15)$$

$$y = l_1 sin\theta_{P11} + l_2 sin\theta_{P21} \quad (16)$$

Apskaičiuoti koordinatėms, reikia rasti θ_{P11}, θ_{P12}, θ_{P21}, θ_{P22}. Jas galima rasti apskaičiavus keturias papildomas formules:

$$k = 2l_0 + l_2(cos\theta_{P12} - cos\theta_{P11}) \quad (17)$$

$$j = l_1|sin\theta_{P12} - sin\theta_{P22}| \quad (18)$$

$$\varphi_1 = \varphi_3 = \text{acos}\left(\frac{\sqrt{k^2+j^2}}{2l_2}\right) \quad (19)$$

$$\varphi_2 = \text{atan}\left(\frac{j}{k}\right) \quad (20)$$

Turint šias formules, galima sužinoti, kokioje galinėje koordinatėje yra mechanizmas, kuomet žinome variklių pasisukimą.

4.2. Sparno galiuko koordinatės radimas

Norint gauti galutinį plasnojimo mechanizmo matematinį modelį, reikia rasti sparno galiuko koordinates. Kuomet žinome penkių grandžių mechanizmo galinę poziciją P ir įtvirtinto taško koordinatę P2 (žr. 13 pav.) sparno galiuko koordinatę P0 galima rasti šiomis formulėmis:

Pirmiausia randamas vektorius nuo P_2 taško iki P taško.

$$v = (P_x - P_{2x},\ P_y - P_{2y}, P_z - P_{2z}) \quad (21)$$

Tuomet randamas vektoriaus ilgis.

$$\|v\| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2} \quad (22)$$

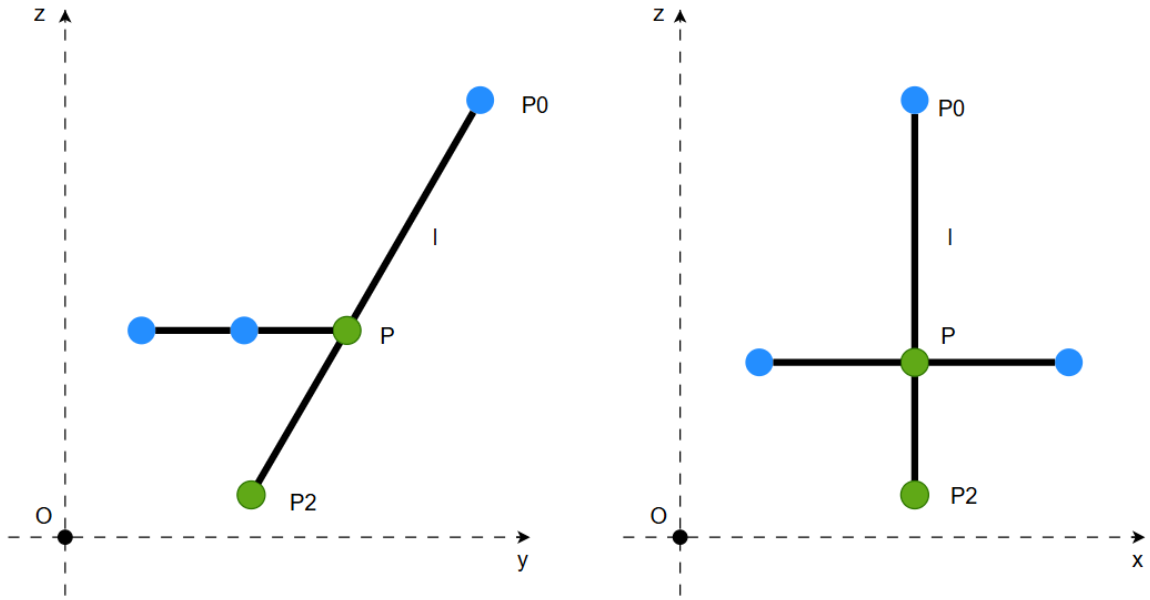
Vektorių paverčiamas vienetiniu. Vektoriaus ilgis tampa lygus vienetui, bet išlaikom vektoriaus kryptis.

$$n = \frac{\text{v}}{\|v\|} \quad (23)$$

Apskaičiuojama sparno galiuko koordinatę.

$$P_0 = P + l * n \quad (24)$$

Šie skaičiavimai naudojami, kai norima apskaičiuoti sparno galiuko poziciją (žr. 13 pav.) žinant motorų pasisukimo kampus.



13 pav. Plasnojimo mechanizmas atvaizduotas YZ ir XZ ašyse

Sekantys skaičiavimai, kuriuos reikia atlikti – penkių grandžių mechanizmo galinio taško radimas, kai žinoma sparno galiuko pozicija.

$$v = (P_{2x} - P_{0x},\ P_{2y} - P_{0y}\ , P_{2z} - P_{0z}) \tag{25}$$

$$\|v\| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2} \tag{26}$$

$$n = \frac{\mathrm{v}}{\|v\|} \tag{27}$$

$$P = P_0 + l * n \tag{28}$$

Šių skaičiavimų tikslas yra rasti variklių susisukimo kampus, kad sparno galiukas atsirastų norimoje koordinatėje.

5. Plasnojimo mechanizmo modeliavimo rezultatų analizė

Sudarius matematinį plasnojimo mechanizmo modelį, sekantis etapas yra padaryti mechanizmo analizę, sukuriant papildomus testavimo algoritmus naudojant „Python“ programavimo kalbą ir „Visual Studio“ programinę aplinką.

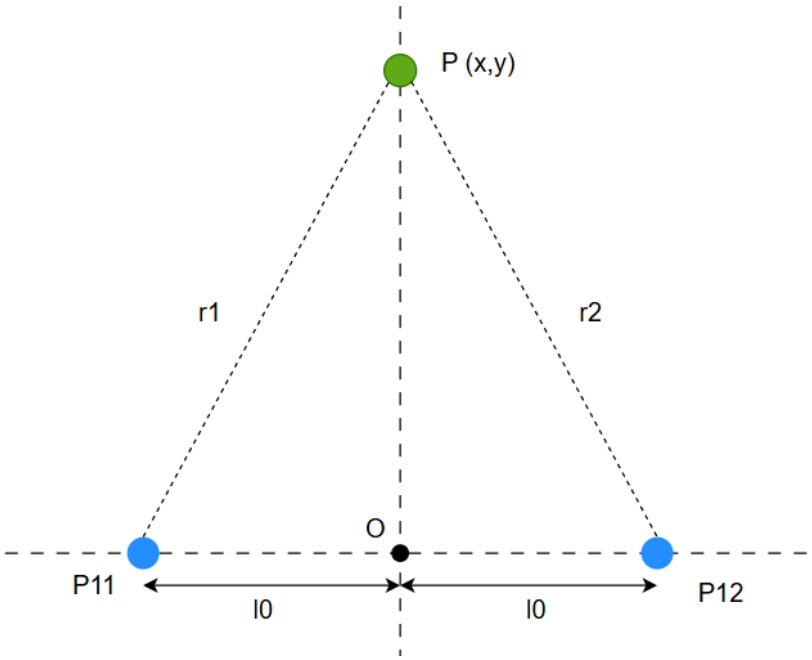
5.1. Darbinio ploto skaičiavimas penkių grandžių mechanizmui

Turint mechanizmo skaitmeninę versiją, reikia apskaičiuoti darbinį mechanizmo plotą. Norint rasti darbinį plotą, apskaičiuojamas aukščiausias ir žemiausias taškas, kurį gali pasiekti sistema.
Aukščiausias pasiekiamas taškas:

$$P_h = \sqrt{(l_1 + l_2)^2 - {l_0}^2} \tag{29}$$

Žemiausias pasiekiamas taškas:

$$P_l = \sqrt{(l_1 - l_2)^2 - {l_0}^2} \tag{30}$$



14 pav. Penkių grandžių pasiekiamo taško apskaičiavimas

Apskaičiavus aukščiausią tašką (žr. 14 pav.) sudaromas taškų masyvas. Masyvo taškai pradedami skaičiuoti nuo aukščiausio taško ir skaičiuojami tol, kol kažkuris iš taškų yra nepasiekiamas arba pasiekia apibrėžtas ribas iki kurių norima skaičiuoti. Pasiekus vieną iš šių sąlygų horizontaliai pajudama per vieną tašką ir vėl skaičiuojama pasiekiami taškai. Taip algoritmas tęsia skaičiavimus, kol visame vertikaliame masyve nebėra pasiekiamų taškų.
Pirmiausia yra apskaičiuojami r1 ir r2 ilgiai:

$$r_1 = \sqrt{(x + l_0)^2 + y^2} \tag{31}$$

$$r_2 = \sqrt{(l_0 - x)^2 + y^2} \quad (32)$$

Sužinojus r1 ir r2 ilgius yra patikrinama ar ilgiai yra tinkami:

$$|l_1 - l_2| \leq r_1 \leq l_1 + l_2 \quad (33)$$

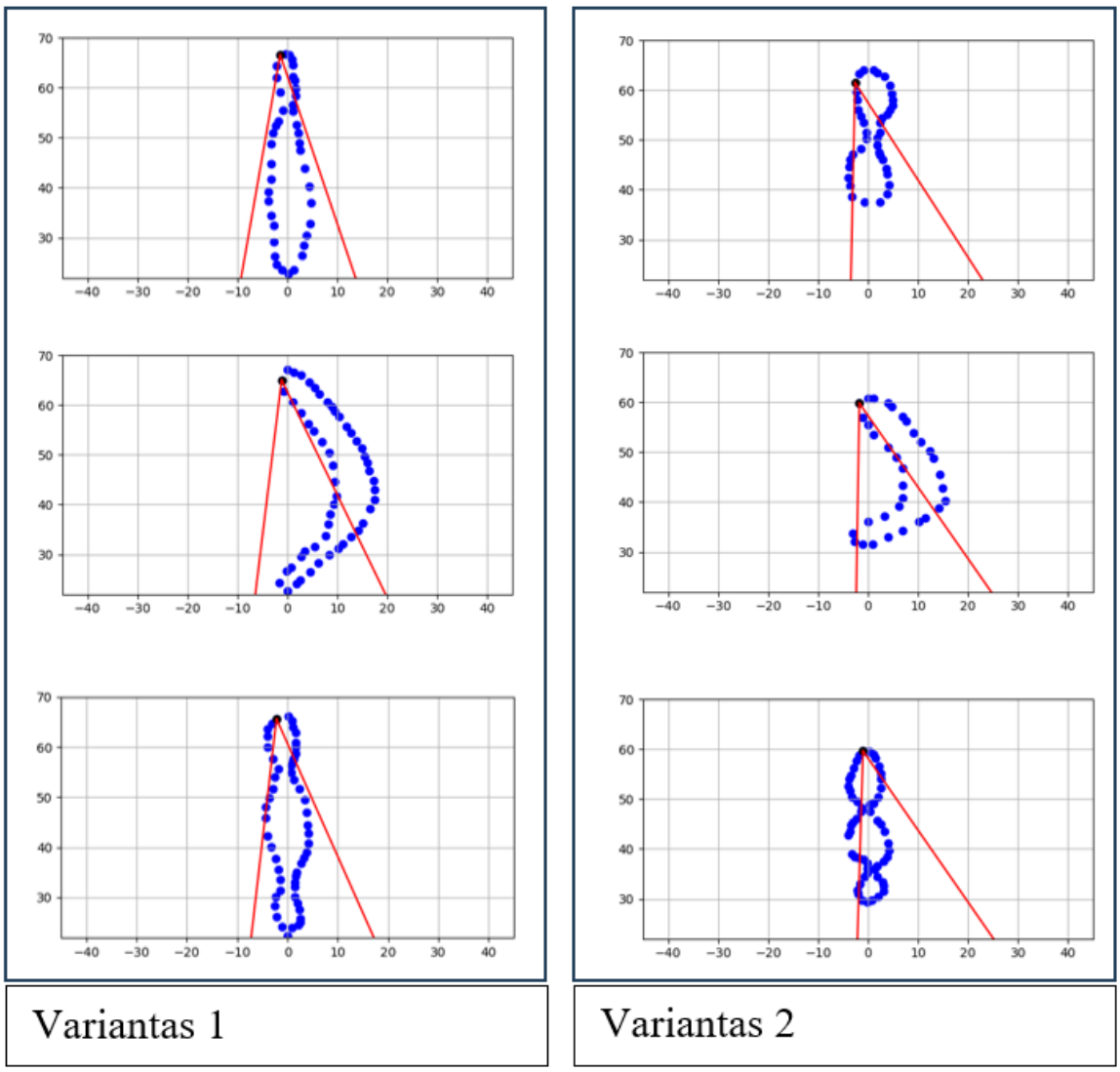
$$|l_1 - l_2| \leq r_2 \leq l_1 + l_2 \quad (34)$$

5.2. Penkių grandžių mechanizmo simuliacijos algoritmo kūrimas

Išvedus matematines išraiškas, kitas etapas – patikrinti, ar mechanizmas geba brėžti reikiamas trajektorijas, nekeisdamas variklių sukimosi krypties. Kadangi laumžirgio sparnų plakimo dažnis siekia iki 70 Hz (2), dažnas krypties keitimas sukeltų nepageidaujamas vibracijas ir dinaminį daužymąsi sistemoje. Siekiant įvertinti penkių grandžių mechanizmo tinkamumą – sukurtas algoritmas, patikrinantis, ar sistema geba tiksliai atkartoti reikiamas trajektorijas. Literatūros analizė parodė, kad laumžirgiams būdingiausios „aštuoneto“, „dvigubo aštuoneto“ ir „kilpos“ formų plasnojimo trajektorijos.

5.2.1. Algoritmas, skirtas nurodyti norimą judėjimo trajektoriją

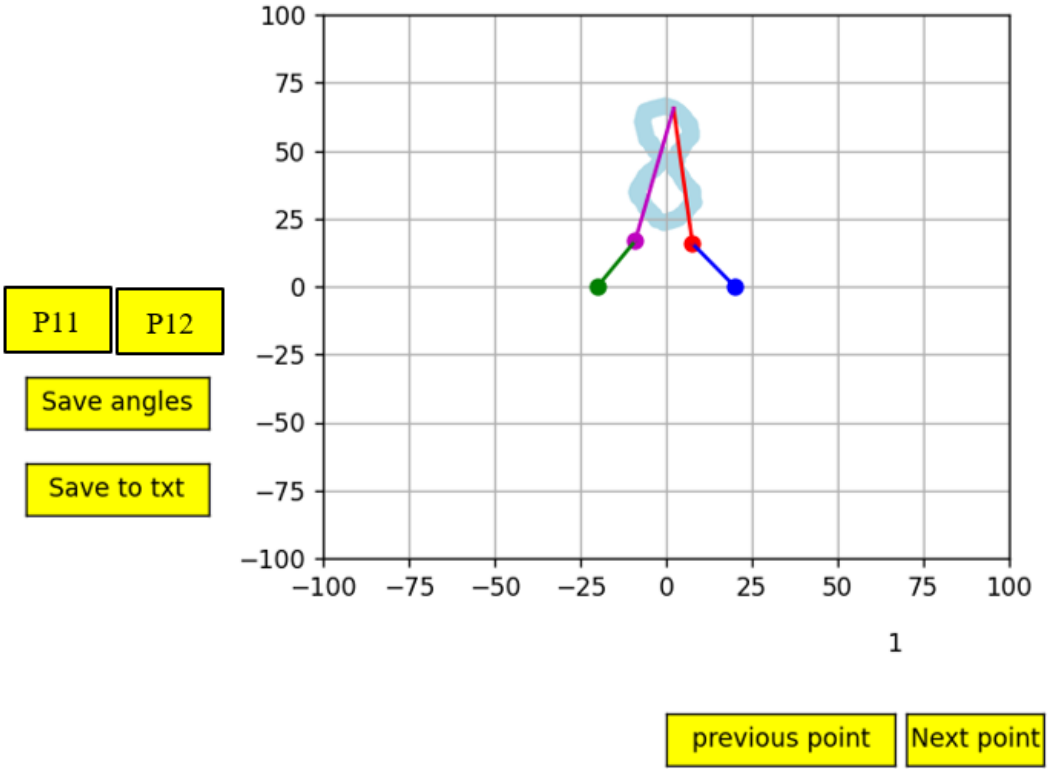
Šio algoritmo paskirtis – koordinačių sistemoje nurodyti taškus, sudarančius trajektoriją, kuria turi judėti mechanizmo galinė grandis. Taškai įvedami rankiniu būdu, o siekiant užtikrinti sistemos stabilumą, už darbinės zonos ribų esantys taškai automatiškai anuliuojami. Siekiant optimizuoti algoritmą, koordinačių ašys apribotos tik pagrindine mechanizmo pasiekiama sritimi. Programoje suformuoti trys baziniai trajektorijų tipai: „aštuonetas“, „kilpa“ ir „dvigubas aštuonetas“ (žr. 15 pav.). Kiekviena trajektorija turi po dvi variacijas: pirma variacija turi taškus aukščiausiame ir žemiausiame y ašies taškuose, antra variacija neturi taškų kraštinėse koordinatėse.



15 pav. Atvaizduotos judėjimo trajektorijos, naudojantis sukurtu algoritmu

5.2.2. Algoritmas, skirtas parinkti mechanizmo konfigūraciją

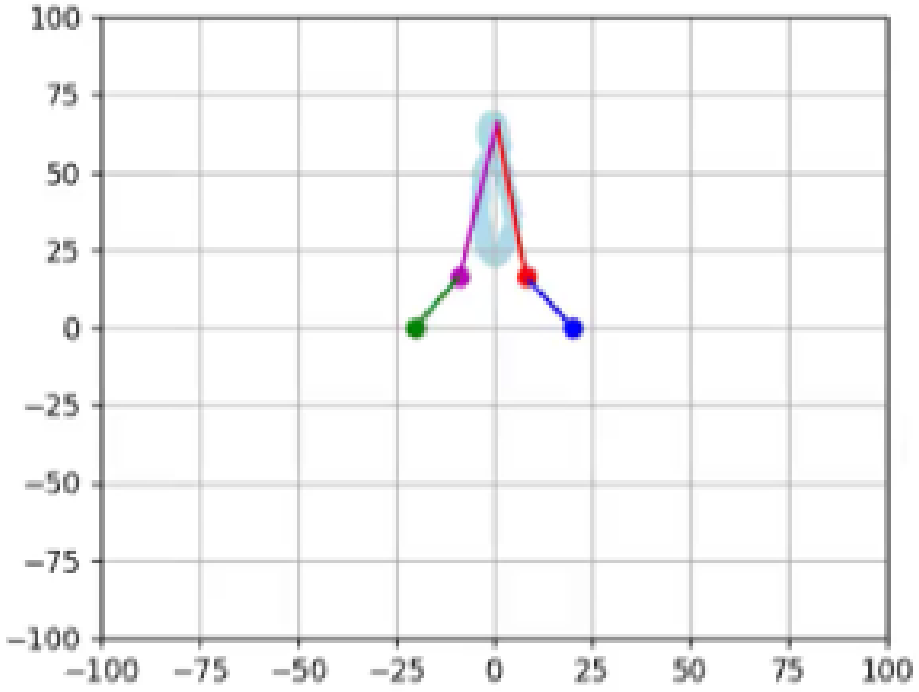
Sukūrus testuoti skirtas trajektorijas, kitas darbo etapas yra patikrinti, ar mechanizmo varikliai, sekdami šias trajektorijas, gali suktis viena kryptimi. Šiam tikslui buvo sukurtas algoritmas, leidžiantis judinti mechanizmą tarp apibrėžtų taškų. Rankiniu būdu judinant mechanizmą nurodytais taškais, galima pasirinkti norimą mechanizmo konfigūraciją (P11+, P11−, P12+, P12−). Keičiant ašių konfigūraciją, atitinkamai kinta ir variklių ašių pasisukimo kampai (žr. 16 pav.). Taip galima įvertinti, ar mechanizmas sugeba brėžti pasirinktą trajektoriją nekeičiant variklių sukimosi krypties. Kiekvienas sukonfigūruotas taškas yra išsaugomas kartu su atitinkamais variklių pasukimo kampais ir taško koordinatėmis, o visi duomenys sukaupiami masyve ir įrašomi į „.txt“ formato failą.



16 pav. Programos langas, skirtas sukonfigūruoti mechanizmą

5.2.3. Algoritmas, atvaizduojantis mechanizmo judėjimą

Šio algoritmo tikslas atvaizduoti judėjimą užduotomis trajektorijomis ir sukonfigūruotomis ašimis. Animacija yra sukuriama iš paskutinės aprašytos programos duomenų. Programa ima „.txt“ failą išsaugotą konfigūravimo programoje ir atvaizduoja mechanizmo judėjimą koordinačių sistemoj (pav. 17).



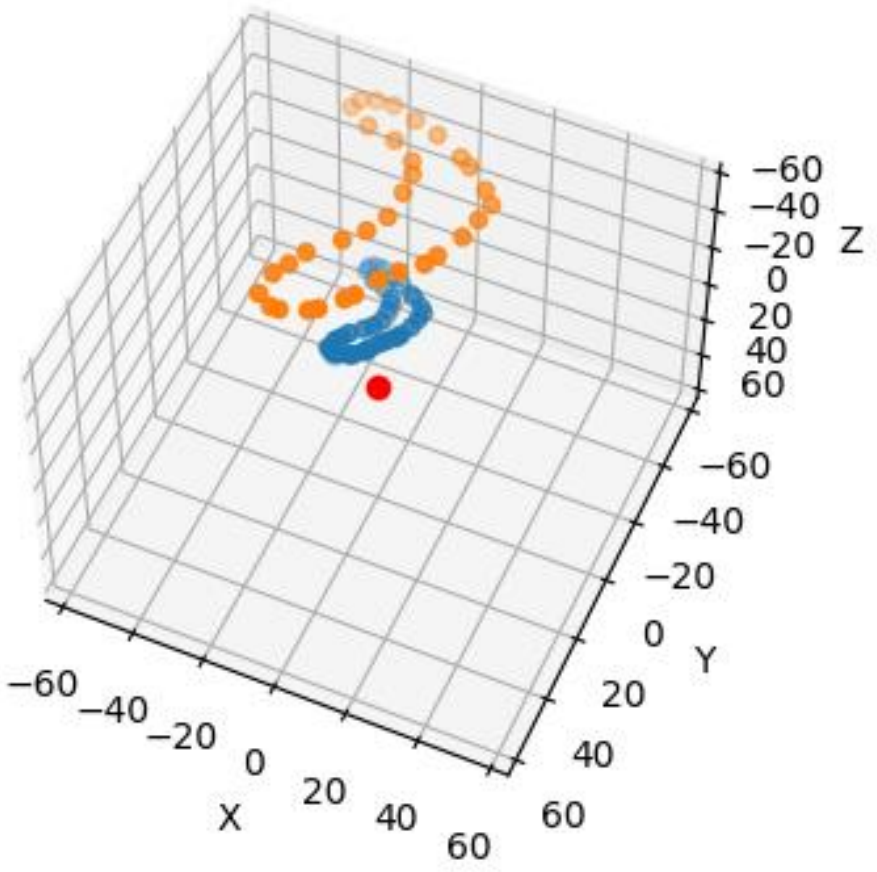
17 pav. Algoritmo langas, atvaizduojantis judėjimą

Susimuliavus penkių grandžių mechanizmą išsiaiškinta, kad varikliai gali suktis nekeičiant krypties. Tačiau sistema turi apribojimą – brėžiama trajektorija turi kirsti aukščiausią ir žemiausią y koordinatės tašką tam, kad nereiktų keisti variklių sukimosi krypties. Šis apribojimas lemia tai, kad plasnojimo mechanizmas turės nekintančią plasnojimo amplitudę. Taip pat buvo pastebėta, kad mechanizmo varikliai gali suktis nekeičiant krypties tik tada kai:

$$l_1 + l_0 \leq l_2 \tag{35}$$

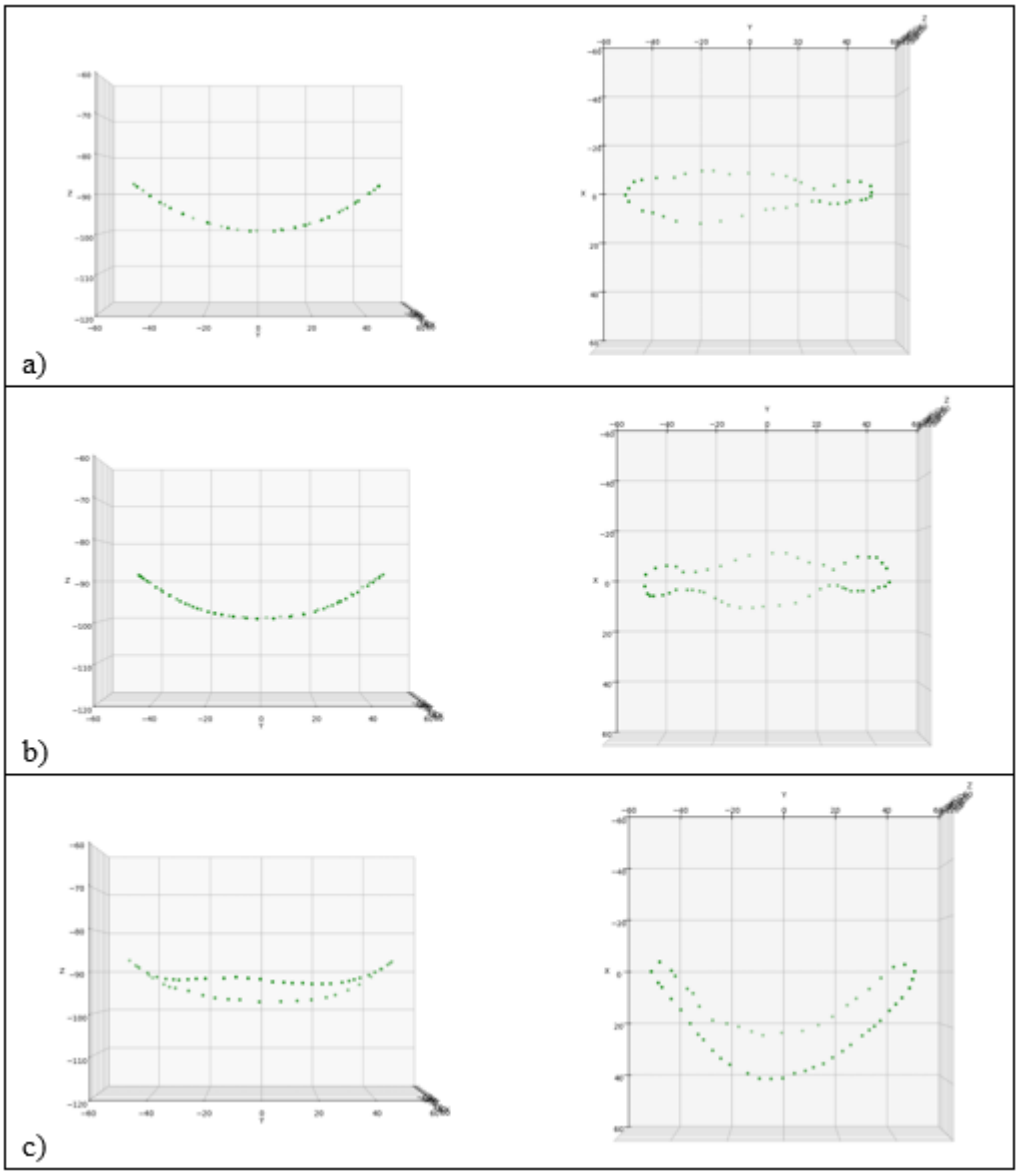
5.3. Plasnojimo trajektorijos atvaizdavimas 3D erdvėje

Kadangi šiame darbe yra analizuojamas laumžirgio plasnojimo mechanizmas, tam reikalinga turėti plasnojimo trajektoriją su tiksliomis taškų koordinatėmis, kurios atvaizduotų sparno galiuko judėjimą. Tikslių trajektorijų, kuriomis plasnoja laumžirgis, nepavyko rasti. Dėl to buvo sukurtos trys panašios trajektorijos, atitinkančios laumžirgio plasnojimą: kilpa, aštuonetas ir dvigubas aštuonetas. Atvaizduoti šioms trajektorijoms buvo parašytas papildomas algoritmas. Algoritmas konvertuoja turimą 2D trajektoriją, skirtą penkių grandžių mechanizmui, į trajektoriją, sekančią sparno galiuką (žr. 18 pav.).



18 pav. Konvertuota trajektorija iš 2D į 3D.

Šis algoritmas apskaičiuoja sparno galiuko padėtį (oranžiniai taškai), kuomet yra žinomas sparno ilgis (100mm), penkių grandžių mechanizmo galinė padėtis (mėlyni taškai) ir įtvirtinto taško koordinatės (raudonas taškas x=0, y=0 ,z=0). Algoritmas skaito, kad atstumas tarp mėlyno taško ir raudono taško yra kintantis, o atstumas tarp oranžinio ir mėlyno visą laiką lieka toks pat – 100mm. Atvaizduoti sparno galiuko koordinates, buvo naudojamos formulės iš 4.2 skyriaus. Sukurtas algoritmas atvaizdavo ir išsaugojo 3 plasnojimo trajektorijas: kilpos, aštuoneto, dvigubo aštuoneto (žr. 19 pav.).

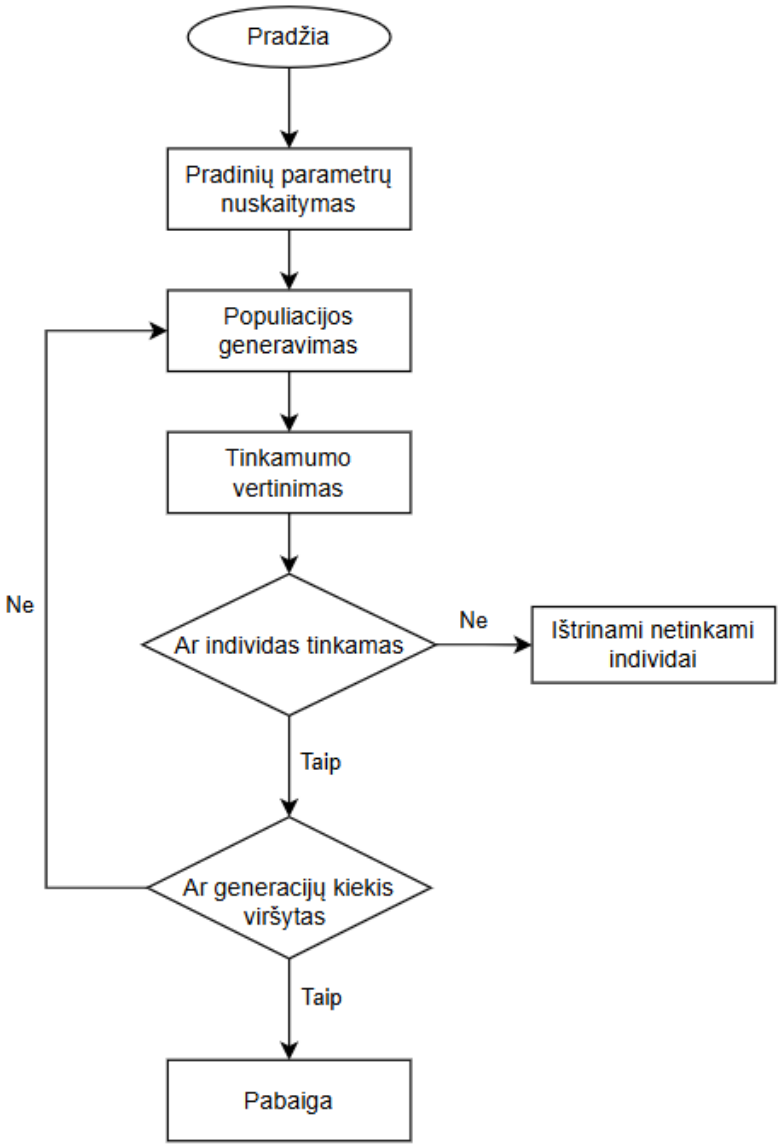


19 pav. Atvaizduotos 3D plasnojimo trajektorijos

Šiame paveikslėlyje galima matyti tris skirtingus plasnojimo trajektorijos variantus. Kiekvienas variantas yra atvaizduotas iš šono ir viršaus. Variantas a) – vaizduoja aštuoneto formos trajektoriją, b) – dvigubo aštuoneto forma, c) – kilpos forma. Šios formos svarbios tuo, nes pagal jas bus optimizuojamas mechanizmas.

6. Mechanizmo svirčių geometrinių parametrų optimizavimas, taikant genetinį algoritmą

Siekiant, kad penkių grandžių mechanizmas kuo tiksliau atkartotų laumžirgio skrydžiui būdingas trajektorijas, būtina parinkti optimalius grandžių ilgius ir įtvirtinimo taškų koordinates. Atsižvelgiant į tai, kad mechanizmo darbinė zona yra apribota netiesinių geometrinių sąlygų, o paieškos erdvė yra plati, šiame darbe optimizavimui pasirinktas genetinis algoritmas (žr. 20 pav.).



20 pav. Genetinio algoritmo diagrama

6.1. Optimizavimo uždavinio formulavimas

Optimizavimo proceso metu algoritmas ieško geriausio keturių parametrų rinkinio – individo. Algoritme individo parametrai apibrėžiami kaip: l_0 – atstumas tarp variklių ašių (bazė), l_1, l_2 – pirmosios ir antrosios mechanizmo grandžių ilgiai, Z_{fix} – sparno įtvirtinimo taško atstumas, nulemiantis trajektorijos erdvinę orientaciją. Paieškos erdvė apribota rėžiais: svirtys l_1, l_2 = (5, 70) mm, o fiksavimo taškas Z_{fix} = (5, 70) mm. Tai užtikrina, kad gautas sprendinys bus techniškai realizuojamas ir proporcingas bendram laumžirgio modelio masteliui. Turint šiuos parametrus, algoritmas gali kurti populiacijas atsitiktiniu būdu formuojant individus.

6.2. Tikslo funkcija

Tinkamumo funkcija šiame darbe veikia kaip pagrindinis atrankos filtras. Jos užduotis įvertinti parametrų rinkinį (l_0, l_1, l_2, Z_{fix}) ir grąžinti skaitinį įvertinimą, nusakantį, kiek gautas mechanizmas yra artimas idealiam sprendiniui. Funkcijos vykdymas suskirstytas į tris loginius blokus: geometrinį filtravimą, judesio amplitudės analizę ir tikslumo skaičiavimą.

6.2.1. Geometrinis filtravimas

Pirmasis blokas tikrina geometrinį filtravimą. Blokas skaičiuoja, ar svirčių ilgiai sudaro veikiantį mechanizmą. Iš pradžių yra tikrinama ar įmanomas tokios konfigūracijos mechanizmas. Kad mechanizmo svirtys susijungtų, jos turi atitikti šią sąlygą:

$$l_0 = |l_1 - l_2| \tag{36}$$

Jeigu sąlyga nėra pasiekiama, svirtys yra per trumpos, kad penkių grandžių mechanizmas susijungtų. Sekanti sąlyga, kurią turi atitikti mechanizmas yra ši:

$$l_1 > l_2 \tag{37}$$

Neįgyvendinus šios sąlygos, mechanizmą valdantys varikliai negali suktis nekeičiant krypties. Jeigu viena iš šių sąlygų yra neįgyvendinama, funkcija atiduoda -1000000 įvertinimą. Gavus tokį įvertinimą, algoritmas eliminuoja individą. Trečiasis filtras yra:

$$l_1 < \frac{l_2}{x_{leng}} \tag{38}$$

Šio filtro tikslas yra neleisti l_2 tapti neproporcingai ilgesniam už l_1. Tai taikoma norint apriboti mechanizmo dydį.

6.2.2. Judesio amplitudės filtravimas

Sekantis blokas tikrina judesio amplitudę. Šio bloko tikslas aptikti ir eliminuoti konfigūracijas, kurios generuoja per didelį arba nepakankamą vertikalų poslinkį, peržengiantį numatytus paklaidos rėžius. Ši funkcija apskaičiuoja aukščiausią penkių grandžių mechanizmo poziciją naudojant formulę:

$$y_{high} = \sqrt{(l_1 + l_2)^2 - {l_0}^2} \tag{39}$$

Žemiausia pozicija apskaičiuojama naudojant šią formulę:

$$y_{low} = \sqrt{(l_1 - l_2)^2 - {l_0}^2} \tag{40}$$

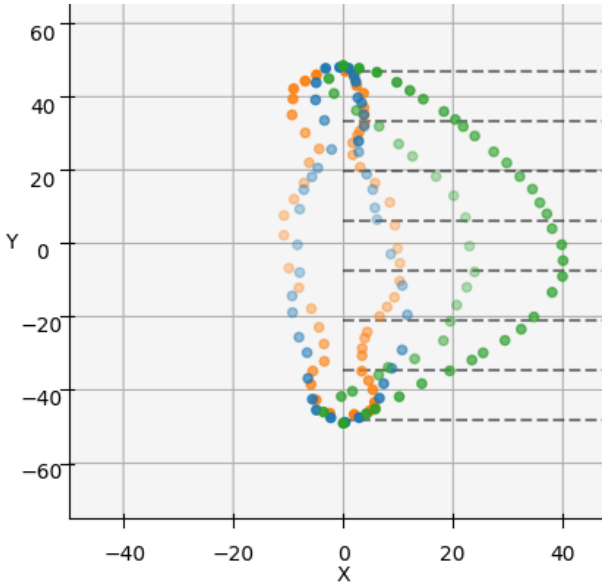
Turint šiuos taškus, apskaičiuojama sparno galiuko pozicija. Ji skaičiuojama naudojantis formulėmis iš 2.3 skyriaus. Apskaičiavus sparno galiuko poziciją, lyginama, ar pozicija neišeina iš duotų paklaidos rėžių. Jeigu išeina – funkcija duoda konfigūracijai -1000000 įvertinimą.

6.2.3. Sprendinio tikslumo skaičiavimo funkcija

Trečiasis tinkamumo funkcijos etapas yra skirtas mechanizmo trajektorijos sekimo tikslumui įvertinti. Tikslinės koordinatės nustatomos analizuojant tris bazines plasnojimo trajektorijas, aprašytas 1.3 skyriuje (kilpą, aštuonetą ir dvigubą aštuonetą). Remiantis šiomis kreivėmis, suformuojamas 8 kontrolinių taškų masyvas, apibrėžiantis reikalaujamą mechanizmo pasiekiamumą x ašies atžvilgiu (žr. 21 pav.). Sistemos generuojamos trajektorijos nuokrypis nuo tikslinių taškų vertinamas pagal formulę:

$$f(x) = -\sum_1^8 |X_i - X_{targ,i}|. \tag{41}$$

Gautas įvertinimas yra neigiamas skaičius, parodantis suminę taškų paklaidą. Tokia metodika leidžia algoritmui efektyviai identifikuoti sprendinius, kurių nuokrypis yra mažiausias.



21 pav. Judesio trajektorijų sulyginimas

6.3. Populiacijos generavimas

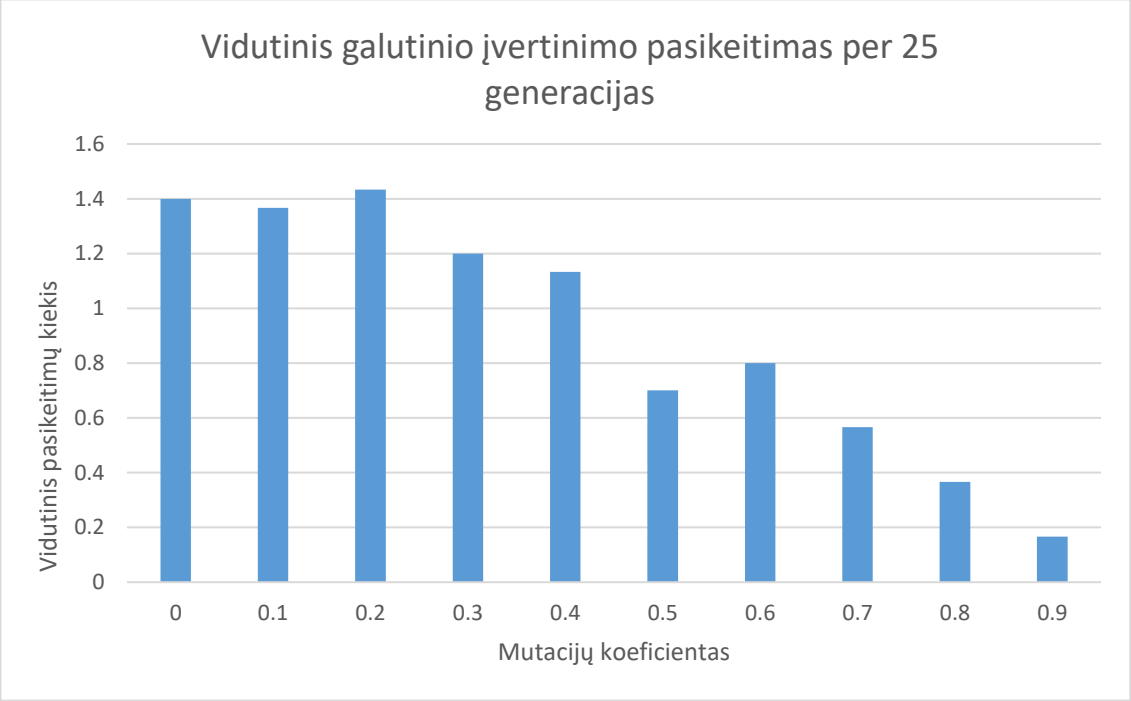
Ši funkcija yra skirta sukurti skirtingas konfigūracijas. Iš pradžių yra suformuojama pirma populiacija iš atsitiktinių parametrų. Šios populiacijos dydis yra nurodomas programos pradžioje. Parametrai yra parenkami pagal 6.1 skyriuje aprašytus rėžius; tai yra daroma tam, kad konfigūracijos neperžengtų grandžių matmenų limitų. Sukūrus populiaciją, visi individai praeina tinkamumo funkciją. Praėjus šia funkciją, kiekvieno individo rezultatas yra įrašomas. Surinkus šiuos duomenis, yra naudojama papildoma funkcija, kuri varžymosi būdu atrenka geriausią individą iš trijų atsitiktinai pasirinktų individų. Laimėję individai yra surašomi į naują masyvą, o visas ciklas kartojamas tol, kol sudaromas naujas masyvas su nurodytu populiacijos kiekiu. Iš kiekvienos poros yra paimama tam tikra parametrų dalis; tie parametrai yra sudedami į 2 naujus individus. Tuomet naudojame papildomą mutacijos funkciją, kuri daliai individų pakeičia parametrus, taip neleidžiant nusistovėti vienodiems parametrams. Galiausiai yra sukuriama nauja populiacija, kuri yra testuojama ir dauginama tokiu pačiu būdu.

7. Genetinio algoritmo parametrų derinimas

Kad genetinis algoritmas skaičiuotų galimas konfigūracijas efektyviai ir nestringant, tam reikia suderinti algoritmo parametrus. Algoritmui suderinti reikėjo nustatyti mutacijos, populiacijos ir generacijos parametrų optimalias vertes.

7.1. Mutacijos parametro derinimas

Pirmoji derinimo dalis prasideda testuojant ir derinant mutacijos funkciją. Šios funkcijos tikslas yra neleisti algoritmui kurti tokių pačių konfigūracijų. Funkcija keičia suporuotų naujų individų konfigūraciją. Šioje funkcijoje taip pat galima keisti individų mutacijos dažnį. Šis parametras apibrėžia individo parametrų keitimo ribas. Tinkamiausio parametro radimui geometrinio filtravimo funkcijos parametrų ribos ir judesio amplitudės funkcijos parametrų ribos buvo neribojamos. Tuomet buvo nustatyti parametrai: populiacijos dydis = 250 ir generacijų kiekis = 25. Tyrimo metu algoritmas buvo vykdomas cikliškai, sistemingai didinant mutacijos tikimybę nuo 0 iki 1, taikant 0,1 žingsnį, algoritmas buvo leidžiamas 30 kartų. Padarius šį testą, buvo išvestas 5 bandymų vidurkis, kuris parodė, kiek kartų per 25 generacijas pasikeičia galutinis įvertinimas (pav. 22).

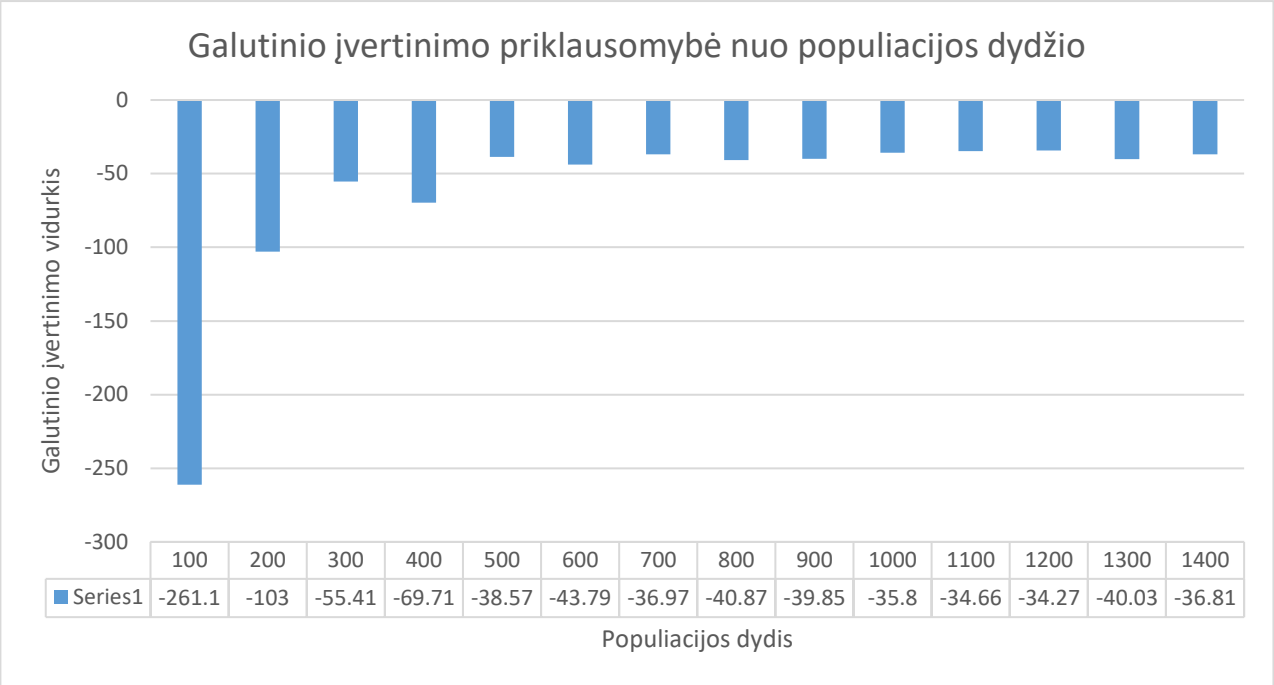


22 pav. Vidutinis galutinio įvertinimo pasikeitimas per 25 generacijas

Remiantis duomenimis, pastebėta, kad daugiausiai sprendiniai keičiasi kuomet mutacijos koeficientas yra tarp 0 ir 0,2. Nuo 0,3 mutacijos koeficiento vidutinis sprendinių gerėjimas pradeda blogėti. Atsižvelgiant į duomenis, darbe buvo pasirinkta naudoti 0,2 mutacijos koeficientą, nes tai duoda geriausius rezultatus.

7.2. Populiacijos parametro derinimas

Norint nustatyti tinkamiausią algoritmo populiaciją, buvo paimtas jau nustatytas mutacijos koeficientas 0.2 ir generacijų kiekis 25. Turint šiuos parametrus, buvo leidžiamas algoritmas keičiant populiacijos parametrą nuo 100 iki 1400 žingsniu 100 (žr. 23 pav.).

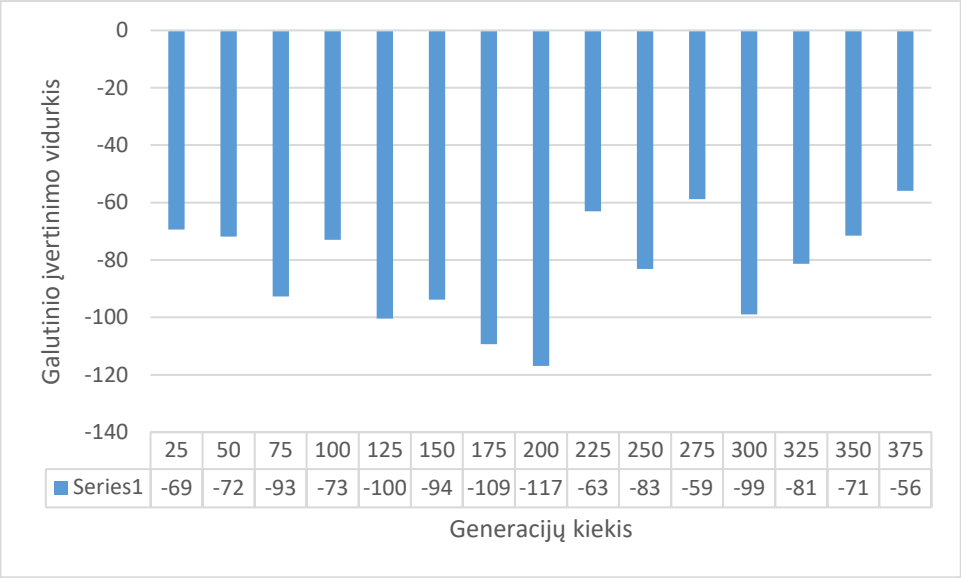


23 pav. Galutinis įvertinimo priklausomybė nuo populiacijos dydžio

Atlikus tyrimą, buvo pastebėta, kad prasčiausi rezultatai gaunami, kuomet populiacija yra maža – nuo 100 individų iki 400. Nuo 500 matomas rezultatų pagerėjimas. Nuo 700 iki 1400 individų matomas nusistovėjimas. Nors geriausias rezultatas yra 1200 individų populiacijos, tačiau, taupant skaičiavimų resursus, buvo pasirinkta naudoti 700 individų populiaciją.

7.3. Generacijų kieko parametro parinkimas

Paskutinis algoritmo derinimo etapas yra generacijų kiekio nustatymas. Eksperimento metu populiacijos dydis buvo 200, o mutacijų koeficientas 0,2. Siekiant rasti kuo tikslesnį generacijų kiekio parametrą, buvo atliekami 30 bandymų. Bandymai buvo atliekami leidžiant algoritmą ir matuojant galutinį įvertinimą, kai yra keičiamas generacijų kiekis nuo 25 iki 375 žingsniu po 25 (žr. 24 pav.).

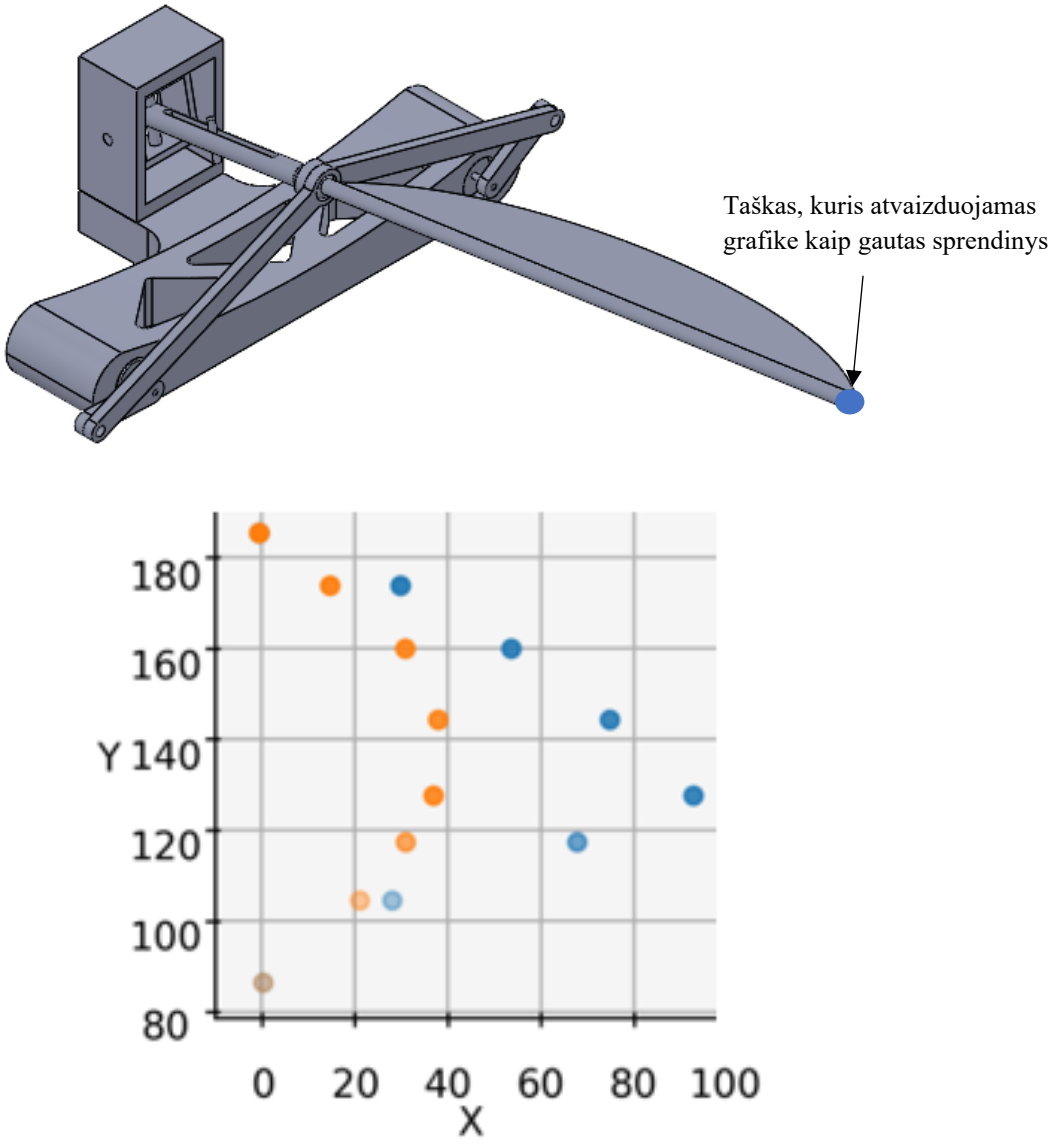


24 pav. Galutinio įvertinimo vidurkio priklausomybė nuo generacijų kiekio

Gauti rezultatai rodo, kad, keičiantis generacijų kiekiui, procesas nėra tolygus. Nors teoriškai didesnis generacijų kiekis turėtų leisti algoritmui rasti geresnį sprendinį, tačiau iš rezultatų matoma, kad ir mažesni generacijų kiekiai leidžia algoritmui rasti gerus rezultatus – 275 generacijos turi panašų rezultatą kaip 375 generacijos. Turint šiuos rezultatus, algoritmui naudoti buvo pasirinkta 275 generacijos.

8. Genetinio algoritmo optimizavimo rezultatų apibendrinimas

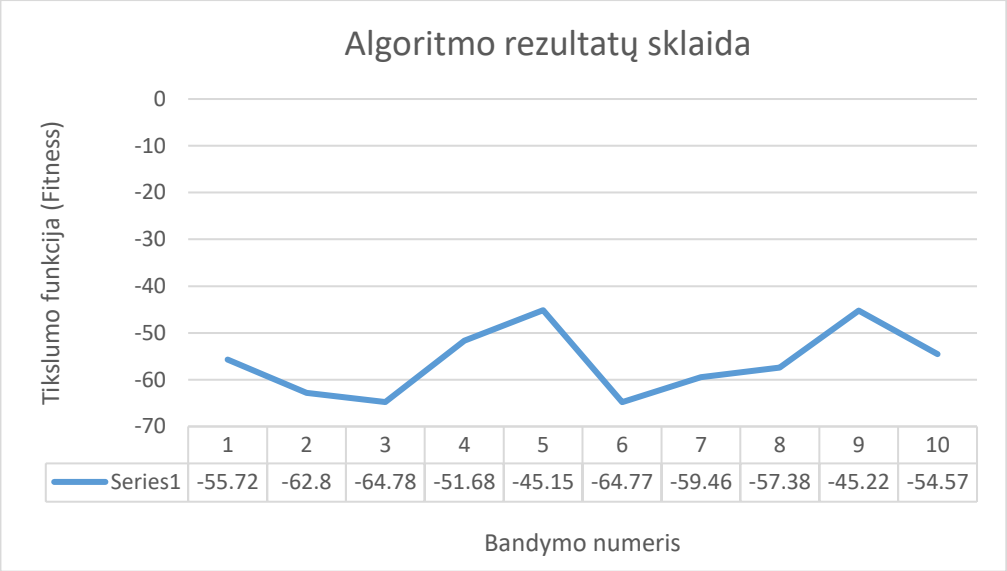
Atlikus algoritmo derinimą, sekantis etapas buvo algoritmo išbandymas sprendžiant optimizavimo uždavinį. Norint palyginti algoritmo rezultatus, pirmiausia buvo sukurtas pradinis mechanizmas be optimizavimo. Šio mechanizmo parametrai buvo: l0=47,57mm, l1=19.27mm, l2=96.69mm, fiksuoto taško z koordinatė = 43.15mm. Su šiais parametrais mechanizmas pasiekė -173 balų įvertinimą. Ši konfigūracija yra atskaitos taškas norint palyginti optimizavimo tikslumą.



25 pav. Atskaitos konfigūracijos taškų (mėlynos spalvos) išsidėstymo palyginimas su norimos sparnui braižyti trajektorijos taškais (oranžinės spalvos)

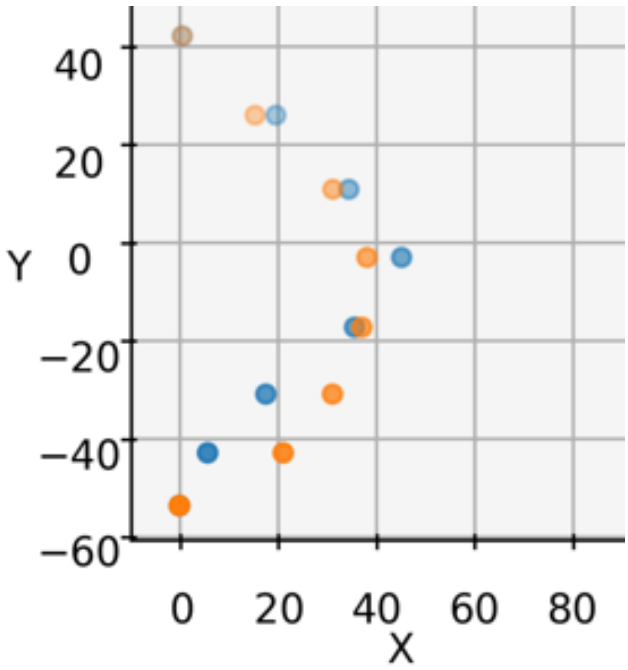
Gautame paveikslėlyje (žr. 25 pav.) matomos dvi taškų grupes. Pirmoji taškų grupė yra oranžinė, šie taškai vaizduoja norimą išbrėžti trajektoriją. Oranžiniai taškai yra nustatyti iš visų norimų atkartoti plasnojimo trajektorijų sulyginimo (žr. 21 pav.). Mėlyni taškai yra sprendinio pasiekiama trajektorija. Mėlyni taškai žymi kraštines pozicijas, kurias pasiekia mechanizmo sparno galiukas, kuris pavaizduotas kaip taškas P0 (žr. 13 pav.). Kaip matoma iš paveikslėlio (žr. 25 pav.), sprendinys yra netikslus. Turint konfigūraciją, nuo kurios galima lyginti optimizavimą, buvo ištestuotas suderintas

optimizavimo algoritmas. Eksperimentas buvo atliekamas dešimt kartų ir tikrinama kokie rezultatai gaunami. Atlikus eksperimentą, buvo pamatyta, kad algoritmas randa konfigūracijas, patenkančias tarp -45 ir -65 įvertinimo (žr. 26 pav.).



26 pav. Algoritmo rezultatų sklaida

Iš grafiko galima matyti, kad įvertinimai nėra vienodi. Tai yra todėl, nes genetinis algoritmas sprendžia uždavinį atsitiktinai ir kiekvienas sprendinys yra unikalus. Todėl, norint gauti kuo geresnį įvertinimą, algoritmas buvo leidžiamas 10 kartų. Atlikus šį eksperimentą, buvo išrinkta geriausią įvertinimą turinti konfigūracija. Geriausia konfigūracija yra kuomet l0 = 31,05mm; l2 = 11,86mm; l3 = 45,11; fiksuoto taško z koordinatė = 44,18. Turint šiuos parametrus, tikslumo funkcija yra lygi -45 balams (žr. 27 pav.).



27 pav. Gautos konfigūracijos taškų išsidėstymo palyginimas su kontroliniais taškais

Gautame paveikslėlyje matomos algoritmo sugeneruotos konfigūracijos trajektorijos lyginant su kontroliniais taškais. Iš šio grafiko matoma, kad idealiai trajektorijos atkartoti nepavyksta. Optimizavus mechanizmą, judesio trajektorija susiaurėja už norimų ribų trajektorijos apačioje. Šį trūkumą galima panaikinti keičiant mechanizmo konfigūraciją, tačiau tokiu atveju mechanizmas padidėtų. Kadangi norima, kad mechanizmas išliktų kuo mažesnis, konfigūracija yra paliekama tokia, kokią parinko algoritmas.

Atlikus optimizavimą, geriausias algoritmo sprendinys buvo 3,8 karto geresnis negu kontrolinė konfigūracija. Po optimizavimo buvo pastebėta, kad, keičiant mechanizmo parametrus, didžiausią įtaką trajektorijos formavimui turi penkių grandžių mechanizmo grandžių ilgiai. Didinant atstumą tarp įtvirtinto taško P2 ir penkių grandžių mechanizmo galinio taško P matoma, kad plasnojimo amplitudė didėja.

Išvados

1. Atlikus laumžirgio sparnų plasnojimo anatomijos analizę, nustatyta, kad skrydžio efektyvumą lemia sudėtinga sparnų kinematika ir tiesioginių raumenų sistema. Ši sistema leidžia nepriklausomai valdyti sparnų poras bei realizuoti skirtingus plasnojimo režimus. Surasta, kad pagrindiniai plasnojimo rėžimai yra kilpa, aštuonetas ir dvigubas aštuonetas.
2. Sudarytas laumžirgio sparno plasnojimo mechanizmo matematinis modelis. Modelis pagrįstas penkių grandžių mechanizmo tiesioginės ir atvirkštinės kinematikos analize. Sudarytas modelis leidžia apskaičiuoti valdymo kampus, žinant sparno galiuko koordinates. Modelis taip pat leido nustatyti mechanizmo darbinę sritį ir įvertinti galimas mechanizmo konfigūracijas, reikalingas nenutrūkstamam sparno plasnojimo judesiui užtikrinti.
3. Darbe buvo sudarytas parametrų optimizavimo uždavinys „Python“ programavimo kalba. Šis uždavinys yra svarbiausia genetinio algoritmo dalis. Funkcija išfiltruoja mechanizmo konfigūracijas, kurios yra netinkamos atlikti uždaviniui. Pagrindiniai uždavinio filtrai yra išmesti mechanizmo konfigūracijas, kurios negali išbraižyti plasnojimo trajektorijos nekeičiant variklių sukimosi krypties, bei išmesti sprendinius, kurie nesudaro susirenkančio mechanizmo. Atlikus filtravimo uždavinį, buvo apskaičiuota likusių sprendinių kokybė. Kokybės funkcija skaičiuojama randant atstumą tarp norimos trajektorijos taškų ir sprendinio trajektorijų taškų atstumų sumos.
4. Darbe buvo atliktas mechanizmo optimizavimas. Optimizuoti mechanizmui buvo pasitelktas genetinis algoritmas. Pirmiausia buvo sukurtas mechanizmas su atskaitos konfigūracija l0=47,57mm, l1=19.27mm, l2=96.69mm, fiksuoto taško z koordinatė = 43.15mm – mechanizmas buvo įvertintas -173 balams. Tuomet buvo atliktas optimizavimas su genetiniu algoritmu. Algoritmas apskaičiavo, kad geriausia konfigūracija yra l1 = 31,05mm; l2 = 11,86mm; l3 = 45,11; fiksuoto taško z koordinatė = 44,18. Su šia konfigūracija optimizavimo uždavinio rezultatas yra -45,146. Optimizuota konfigūracija gavosi 3,8 karto geresnė negu atskaitos konfigūracija.
5. Darbe buvo nustatyti faktoriai, lemiantys mechanizmo judėjimą. Vienas pagrindinių faktorių yra penkių grandžių mechanizmo grandžių ilgiai – jeigu yra ilginama l0 grandis, tuomet sparno galiuko trajektorija išplatėja x koordinate. Grandys l1 ir l2 reguliuoja sparno trajektorijos braižymą y ašimi – kuo didesnė l1 ir l2 grandžių suma, tuo didesne amplitude y koordinačių ašimi gali judėti mechanizmas. Didelę įtaką sparno judesiams turi įtvirtinto universalaus šarnyro koordinatės – judinant įtvirtintą šarnyrą xy koordinatėmis, reguliuojama sparno galiuko centrinė padėtis, o traukiant įtvirtintą šarnyrą toliau nuo penkių grandžių mechanizmo, didinamas pasiekiamas sparno galiuko plotas.

Literatūra

1. Vance, J. T., Altshuler, D. L., Dickson, W. B., Dickinson, M. H., & Roberts, S. P. (2014). Hovering Flight in the HoneybeeApis mellifera: Kinematic Mechanisms for Varying Aerodynamic Forces. Physiological and Biochemical Zoology, 87(6), 870–881. https://doi.org/10.1086/678955
2. Bäumler, F., Gorb, S. N., & Büsse, S. (2018). Comparative morphology of the thorax musculature of adult Anisoptera (Insecta: Odonata): Functional aspects of the flight apparatus. Arthropod Structure & Development, 47(4), 430–441. https://doi.org/10.1016/j.asd.2018.04.003
3. Dickinson, M. (2006). Insect flight. Current Biology, 16(9), R309–R314. https://doi.org/10.1016/j.cub.2006.03.087
4. Nachtigall, W., Wisser, A., & Eisinger, D. (1998). Flight of the honey bee. VIII. Functional elements and mechanics of the "flight motor" and the wing joint - one of the most complicated gear-mechanisms in the animal kingdom. Journal of Comparative Physiology B, 168(5), 323–344. https://doi.org/10.1007/s003600050152
5. Rajabi, H., & Gorb, S. N. (2020). How do dragonfly wings work? A brief guide to functional roles of wing structural components. International Journal of Odonatology, 23(1), 23–30. https://doi.org/10.1080/13887890.2019.1677515
6. Sivasankaran, P. N., Ward, T. A., Salami, E., Viyapuri, R., Fearday, C. J., & Johan, M. R. (2017). An experimental study of elastic properties of dragonfly-like flapping wings for use in biomimetic micro air vehicles (BMAVs). *Chinese Journal of Aeronautics*, *30*(2), 726–737. https://doi.org/10.1016/j.cja.2017.02.011
7. Hu, Y., Zhu, C., Liu, Q., Zhu, D., Xue, J., Li, Q., & Zhou, X. (2025). Research on the aerodynamic characteristics of dragonfly leading edge. *Microscopy Research and Technique*, *88*(1), 181–201. https://doi.org/10.1002/jemt.24693
8. Li, C., & Dong, H. (2017). Wing kinematics measurement and aerodynamics of a dragonfly in turning flight. *Bioinspiration & Biomimetics*, *12*(2), Article 026001. https://doi.org/10.1088/1748-3190/aa5761
9. Wang, L., Shi, Z., Zhang, W., Chen, S., Geng, X., Zhi, Q., & Liao, X. (2025). Kinematics and aerodynamic performance of active rotating wings inspired by dragonfly muscle-driven wing veins. Aerospace Science and Technology, 164, Article 110411. https://doi.org/10.1016/j.ast.2025.110411
10. Chen, Y. H., Skote, M., Zhao, Y., & Huang, W. M. (2013). Dragonfly (Sympetrum flaveolum) flight: Kinematic measurement and modelling. *Journal of Fluids and Structures*, *40*, 115–126. https://doi.org/10.1016/j.jfluidstructs.2013.04.003
11. Koizumi, S., Nakata, T., & Liu, H. (n.d.). Flexibility Effects of a Flapping Mechanism Inspired by Insect Musculoskeletal System on Flight Performance. *Frontiers in Bioengineering and Biotechnology*, *9*, 612183. https://doi.org/10.3389/fbioe.2021.612183
12. Ozaki, T., & Hamaguchi, K. (2018). Bioinspired Flapping-Wing Robot With Direct-Driven Piezoelectric Actuation and Its Takeoff Demonstration. *IEEE Robotics and Automation Letters*, *3*(4), 4217–4224. https://doi.org/10.1109/LRA.2018.2863104
13. Liang, J., Zheng, M., Pan, T., Su, G., Deng, Y., Cao, M., & Li, Q. (2025). Development of a Dragonfly-Inspired High Aerodynamic Force Flapping-Wing Mechanism Using Asymmetric Wing Flapping Motion. *Biomimetics (Basel, Switzerland)*, *10*(5), 309. https://doi.org/10.3390/biomimetics10050309

14. Jang, J., & Yang, G.-H. (2018). Development of Camber-Flat Wing Structure Convert Mechanism for Asymmetric Flapping Micro Air Vehicle. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1395–1400. https://doi.org/10.1109/IROS.2018.8594104
15. Abualigah, L. (2024). Particle swarm optimization algorithm: review and applications. In *Metaheuristic Optimization Algorithms*. Elsevier Science & Technology. https://doi.org/10.1016/B978-0-443-13925-3.00019-4
16. Nayak, J., Swapnarekha, H., Naik, B. et al. 25 Years of Particle Swarm Optimization: Flourishing Voyage of Two Decades. Arch Computat Methods Eng 30, 1663–1725 (2023). https://doi.org/10.1007/s11831-022-09849-x
17. Piotrowski, A. P., Napiorkowski, J. J., & Piotrowska, A. E. (2023). Particle Swarm Optimization or Differential Evolution—A comparison. *Engineering Applications of Artificial Intelligence*, *121*, Article 106008. https://doi.org/10.1016/j.engappai.2023.106008
18. Gong, C., Zhou, N., Xia, S., & Huang, S. (2024). Quantum particle swarm optimization algorithm based on diversity migration strategy. *Future Generation Computer Systems*, *157*, 445–458. https://doi.org/10.1016/j.future.2024.04.008
19. Tijjani, S., Ab Wahab, M. N., & Mohd Noor, M. H. (2024). An enhanced particle swarm optimization with position update for optimal feature selection. *Expert Systems with Applications*, *247*, Article 123337. https://doi.org/10.1016/j.eswa.2024.123337
20. Ye, X., Li, J., Wang, P., & Suganthan, P. N. (2025). A comprehensive survey of adaptive strategies in differential evolutionary algorithms. *Swarm and Evolutionary Computation*, *98*, Article 102081. https://doi.org/10.1016/j.swevo.2025.102081
21. Bajpai, P., Anicho, O., Nagar, A. K., & Bansal, J. C. (2024). Dynamic Mutation Strategy Selection in Differential Evolution Using Perturbed Adaptive Pursuit. *SN Computer Science*, *5*(6), Article 771. https://doi.org/10.1007/s42979-024-03062-2
22. Abdelkader, E. M., Moselhi, O., Marzouk, M., & Zayed, T. (2022). An exponential chaotic differential evolution algorithm for optimizing bridge maintenance plans. *Automation in Construction*, *134*, Article 104107. https://doi.org/10.1016/j.autcon.2021.104107
23. Chai, X., Zheng, Z., Xiao, J., Yan, L., Qu, B., Wen, P., Wang, H., Zhou, Y., & Sun, H. (2022). Multi-strategy fusion differential evolution algorithm for UAV path planning in complex environment. *Aerospace Science and Technology*, *121*, Article 107287. https://doi.org/10.1016/j.ast.2021.107287
24. Zeng, Z., Zhang, M., Zhang, H., & Hong, Z. (2022). Improved differential evolution algorithm based on the sawtooth-linear population size adaptive method. *Information Sciences*, *608*, 1045–1071. https://doi.org/10.1016/j.ins.2022.07.003
25. Ahmad, M. F., Isa, N. A. M., Lim, W. H., & Ang, K. M. (2022). Differential evolution: A recent review based on state-of-the-art works. *Alexandria Engineering Journal*, *61*(5), 3831–3872. https://doi.org/10.1016/j.aej.2021.09.013
26. Chaudhary, R., Verma, P., Salgotra, R., & Gandomi, A. H. (2026). Five Decades of Genetic Algorithms: A Systematic and Bibliometric Review (1975-2025). *Archives of Computational Methods in Engineering*. https://doi.org/10.1007/s11831-025-10487-2
27. Alhijawi, B., & Awajan, A. (2024). Genetic algorithms: theory, genetic operators, solutions, and applications. *Evolutionary Intelligence*, *17*(3), 1245–1256. https://doi.org/10.1007/s12065-023-00822-6
28. Kanyilmaz, A., Tichell, P. R. N., & Loiacono, D. (2022). A genetic algorithm tool for conceptual structural design with cost and embodied carbon optimization. *Engineering*

Applications of Artificial Intelligence, *112*, Article 104711. https://doi.org/10.1016/j.engappai.2022.104711
29. Zhang, T., Chen, L., & Wang, J. (2023). Multi-objective optimization of elliptical tube fin heat exchangers based on neural networks and genetic algorithm. *Energy (Oxford)*, *269*, Article 126729. https://doi.org/10.1016/j.energy.2023.126729
30. Gao, S., Zhang, Y., Ji, S., Wang, X., & Li, Y. (2025). Optimized design for crankshaft bearing of RV reducer based on genetic algorithm. *Journal of Mechanical Science and Technology*, *39*(4), 1915–1928. https://doi.org/10.1007/s12206-025-0318-y

Priedai

1 Programa skirta nurodyti norimą trajektoriją

```
import sys
import math
import matplotlib.pyplot as plt
import time
import numpy as np

l0 = 20
l1 = 20
l2 = 50
EndPoints = []
#loaded_array = np.loadtxt("Figure8.txt", delimiter=",")

def calc_angles(x, y):
    # Angle from left shoulder to end effector
    beta1 = math.atan2( y, (l0 + x) )

    # Angle from right shoulder to end effector
    beta2 = math.atan2( y, (l0 - x) )

    # Alpha angle pre-calculations
    alpha1_calc = (l1**2 + ( (l0 + x)**2 + y**2 ) - l2**2) / (2*l1*math.sqrt( (l0 +
x)**2 + y**2 ))
    alpha2_calc = (l1**2 + ( (l0 - x)**2 + y**2 ) - l2**2) / (2*l1*math.sqrt( (l0 -
x)**2 + y**2 ))
    # If calculations > 1, will fail acos function

    if alpha1_calc > 1 or alpha1_calc < -1 or alpha2_calc > 1 or alpha2_calc < -1:
        print("Unreachable coordinates")
        return(-999, -999)

    # Angle of left shoulder - beta1 and right shoulder - beta2
    alpha1 = math.acos(alpha1_calc)
    alpha2 = math.acos(alpha2_calc)
    # Angles of left and right shoulders
    shoulder1_1 = beta1 - alpha1
    shoulder2_1 = math.pi - beta2 - alpha2
    shoulder1 = shoulder1_1
    shoulder2 = shoulder2_1

    return(shoulder1, shoulder2)

def ShowOnPlot(shoulder1, shoulder2, x, y):

    plt.cla()

    #---------------------------------------------------------
    #draw points
    #---------------------------------------------------------
    A1 = (-l0,0)
    A2 = (l0,0)
    plt.plot(A1[0],A1[1],'bo')
    plt.plot(A2[0],A2[1],'bo')

    P1 = (-l0 + l1*math.cos(shoulder1), l1*math.sin(shoulder1))
    P2 = (l0 + l1*math.cos(shoulder2), l1*math.sin(shoulder2))
    plt.plot(P1[0],P1[1],'ro')
    plt.plot(P2[0],P2[1],'ro')

    PEndPoints = np.array(EndPoints)
    plt.scatter(PEndPoints[:, 0], PEndPoints[:, 1], color='blue', marker='o')
    plt.plot(x,y,'ko')
```

```
    #plt.scatter(loaded_array[:, 0], loaded_array[:, 1], color='blue', marker='o')
    #------------------------------------------------------------

    #------------------------------------------------------------
    #draw lines
    #------------------------------------------------------------
    plt.plot([A1[0],P1[0]],[A1[1],P1[1]],'b')
    plt.plot([P1[0],x],[P1[1],y],'r')
    plt.plot([A2[0],P2[0]],[A2[1],P2[1]],'b')
    plt.plot([P2[0],x],[P2[1],y],'r')
    #------------------------------------------------------------

    plt.xlim(-45, 45)
    plt.ylim(22, 70)
    # plt.xlim(-100, 100)
    # plt.ylim(-100, 100)
    plt.plot()
    plt.grid(True)
    plt.show()

if __name__ == "__main__":

    #--------------------------------------------------------------------
    #code to save draw and save path to .txt
    #--------------------------------------------------------------------

     # Create the figure
    fig, ax = plt.subplots()
    ax.set_title("Click to add points")
    ax.set_xlim(-45, 45)
    ax.set_ylim(22, 70)
    ax.set_aspect('equal')
    ax.grid(True)

    # This function runs every time you click
    def on_click(event):
        # Only accept clicks inside the axes
        if event.inaxes != ax:
            return

        # Get clicked coordinates
        x = event.xdata
        y = event.ydata

        # Print them in terminal
        print(f"Clicked at: ({x:.2f}, {y:.2f})")
        shoulder1, shoulder2 = calc_angles(x, y)

        if shoulder1 == -999 and shoulder2 == -999:
            return
        else:
            #add point to the array
            EndPoints.append([x,y])
            #Save to .txt file
            np.savetxt("coordinates.txt", EndPoints, delimiter=",", fmt="%.2f")

        ShowOnPlot(shoulder1, shoulder2, x, y)

        print("captured points")
        print(EndPoints)


    # Connect mouse click to event handler
    cid = fig.canvas.mpl_connect('button_press_event', on_click)
    plt.show()
```

2 Programa skirta parinkti mechanizmo konfigūraciją

```
from ctypes.wintypes import LPCOLESTR
from pickle import NONE
import sys
import math
from tkinter import Checkbutton
from typing import Text
import matplotlib.pyplot as plt
from matplotlib.widgets import Button, CheckButtons
import time
import numpy as np
from sklearn.setup import configuration

l0 = 20
l1 = 20
l2 = 50
EndPoints = []
TurningAngles = []
LeftAngle = RightAngle = 0.0
Indexnumber = None
configurationL = 1
configurationR = 1
#LPC, LMC, RPC, RMC = TYR, False, False, False
# load data, ensure it's 2D (N x 2)
loaded_array = np.loadtxt("kilpa2.txt", delimiter=",")
loaded_array = np.atleast_2d(loaded_array)



def calc_angles(x, y):
    global shoulderLM, shoulderLP, shoulderRM, shoulderRP
    # Angle from left shoulder to end effector
    beta1 = math.atan2(y, (l0 + x))

    # Angle from right shoulder to end effector
    beta2 = math.atan2(y, (l0 - x))

    # Alpha angle pre-calculations
    alpha1_calc = (l1**2 + (((l0 + x)**2 + y**2)) - l2**2) / (2 * l1 * math.sqrt((l0 +
x)**2 + y**2))
    alpha2_calc = (l1**2 + (((l0 - x)**2 + y**2)) - l2**2) / (2 * l1 * math.sqrt((l0 -
x)**2 + y**2))

    if alpha1_calc > 1 or alpha1_calc < -1 or alpha2_calc > 1 or alpha2_calc < -1:
        print("Unreachable coordinates")
        return (-999, -999)

    alpha1 = math.acos(alpha1_calc)
    alpha2 = math.acos(alpha2_calc)

    shoulderLM = beta1 - alpha1
    shoulderLP = beta1 + alpha1
    shoulderRM = math.pi - beta2 - alpha2
    shoulderRP = math.pi - beta2 + alpha2


def draw_ALLarms(x, y, shoulderLM, shoulderLP, shoulderRM, shoulderRP):

    global a1, a2, p1, p2, l_a1, l_a2, l_b1, l_b2, a12, a22, p12, p22, l_a12, l_a22,
l_b12, l_b22, configurationL, configurationR

    ax.cla()
    ax.set_xlim(-100, 100)
    ax.set_ylim(-100, 100)
```

```
    ax.grid(True)
    ax.scatter(loaded_array[:, 0], loaded_array[:, 1], color='lightblue', marker='o')

    #-----------------------------------------------------------
    #draw points
    #-----------------------------------------------------------
    A1 = (-l0,0)
    A2 = (l0,0)

    #P1 left,P2 right joints with Plus configuration
    #P12 left,P22 right joints with Minuse configuration

    P1 = (-l0 + l1*math.cos(shoulderLP), l1*math.sin(shoulderLP))
    P2 = (l0 + l1*math.cos(shoulderRP), l1*math.sin(shoulderRP))
    P12 = (-l0 + l1*math.cos(shoulderLM), l1*math.sin(shoulderLM))
    P22 = (l0 + l1*math.cos(shoulderRM), l1*math.sin(shoulderRM))
    if configurationR == 1:
        p2, = ax.plot(P2[0], P2[1], 'ro')
        a2, = ax.plot(A2[0], A2[1], 'bo')
        l_b1, = ax.plot([A2[0], P2[0]], [A2[1], P2[1]], 'b')
        l_b2, = ax.plot([P2[0], x],     [P2[1], y],     'r')
    else:
        a22, = ax.plot(A2[0], A2[1], 'go')
        p22, = ax.plot(P22[0], P22[1], 'mo')
        l_b12, = ax.plot([A2[0], P22[0]], [A2[1], P22[1]], 'g')
        l_b22, = ax.plot([P22[0], x],     [P22[1], y],     'm')

    if configurationL == 1:
        p1, = ax.plot(P1[0], P1[1], 'ro')
        a1, = ax.plot(A1[0], A1[1], 'bo')
        l_a1, = ax.plot([A1[0], P1[0]], [A1[1], P1[1]], 'b')
        l_a2, = ax.plot([P1[0], x],     [P1[1], y],     'r')
    else:
        p12, = ax.plot(P12[0], P12[1], 'mo')
        a12, = ax.plot(A1[0], A1[1], 'go')
        l_a12, = ax.plot([A1[0], P12[0]], [A1[1], P12[1]], 'g')
        l_a22, = ax.plot([P12[0], x],     [P12[1], y],     'm')

    fig.canvas.draw_idle()

def PoinPlotForward(event=None):
    IndexCalculator(1)
    calc_angles(loaded_array[current_index, 0], loaded_array[current_index, 1])
    draw_ALLarms(loaded_array[current_index, 0], loaded_array[current_index, 1],
shoulderLM, shoulderLP, shoulderRM, shoulderRP)

def PoinPlotBackward(event=None):
    IndexCalculator(-1)
    calc_angles(loaded_array[current_index, 0], loaded_array[current_index, 1])
    draw_ALLarms(loaded_array[current_index, 0], loaded_array[current_index, 1],
shoulderLM, shoulderLP, shoulderRM, shoulderRP)

def ResetPlot(event=None):
    IndexCalculator(0)
    calc_angles(loaded_array[current_index, 0], loaded_array[current_index, 1])
    draw_ALLarms(loaded_array[current_index, 0], loaded_array[current_index, 1],
shoulderLM, shoulderLP, shoulderRM, shoulderRP)

def LeftConfig(event=None):
    global configurationL,LeftAngle
    try:
        if configurationL == 1:
            configurationL = 2
        else:
            configurationL = 1;
```

```
    except:
        configurationL = 1
    #LeftAngle = shoulder1P

    calc_angles(loaded_array[current_index, 0], loaded_array[current_index, 1])
    draw_ALLarms(loaded_array[current_index, 0], loaded_array[current_index, 1],
shoulderLM, shoulderLP, shoulderRM, shoulderRP)

def RightConfig(event=None):
    global configurationR,RightAngle
    try:
        if configurationR == 1:
            configurationR = 2
        else:
            configurationR = 1;
    except:
        configurationR = 1
    #RightAngle = shoulder2P

    calc_angles(loaded_array[current_index, 0], loaded_array[current_index, 1])
    draw_ALLarms(loaded_array[current_index, 0], loaded_array[current_index, 1],
shoulderLM, shoulderLP, shoulderRM, shoulderRP)

def LeftMinuseConfig(event=None):
    global a1, a2, p1, p2, l_a1, l_a2, l_b1, l_b2, a12, a22, p12, p22, l_a12, l_a22,
l_b12, l_b22, LeftAngle
    a12.remove()
    p12.remove()
    l_a12.remove()
    l_a22.remove()
    fig.canvas.draw_idle()
    #LeftAngle = shoulder1M


    draw_ALLarms(loaded_array[current_index, 0], loaded_array[current_index, 1],
shoulderLM, shoulderLP, shoulderRM, shoulderRP)

def RightMinuseConfig(event=None):
    global a1, a2, p1, p2, l_a1, l_a2, l_b1, l_b2, a12, a22, p12, p22, l_a12, l_a22,
l_b12, l_b22, RightAngle
    a22.remove()
    p22.remove()
    l_b12.remove()
    l_b22.remove()
    fig.canvas.draw_idle()
    #RightAngle = shoulder2M

def IndexCalculator(step):
    global current_index, loaded_array, Indexnumber
    # Indexnumber = plt.text(0.5, 0.5, current_index, horizontalalignment='center',
verticalalignment='center', transform=ax.transAxes)
    if Indexnumber is not None:
        try:
            Indexnumber.remove()
        except Exception:
            pass
        Indexnumber = None
    current_index += step
    if current_index >= len(loaded_array):
        current_index = 1
    elif current_index < 0:
        current_index = len(loaded_array)-1

    Indexnumber = plt.text(0.4, 0.15, current_index, horizontalalignment='center',
verticalalignment='center', transform=fig.transFigure)
```

```
    return current_index

def save_angles(event=None):
    global x, y , configurationL, configurationR
    TurningAngles.append([loaded_array[current_index, 0], loaded_array[current_index,
1] , configurationL, configurationR])
    print(current_index)

def save_to_TXT(event=None):
    np.savetxt("Angles.txt", TurningAngles, delimiter=",", fmt="%.2f")

if __name__ == "__main__":
    current_index = 0

    #create figure / axes and an empty scatter to be updated
    fig = plt.figure()
    ax = fig.subplots()
    plt.subplots_adjust(left=0.3, bottom=0.25)

    ax.set_xlim(-100, 100)
    ax.set_ylim(-100, 100)
    ax.grid(True)

    # create the button and wire the callback
    axes = plt.axes([0.81, 0.01, 0.12, 0.06])  # [left, bottom, width, height] in
figure coords
    bnext = Button(axes, "Next point", color="yellow")
    bnext.on_clicked(PoinPlotForward)

    axes2 = plt.axes([0.60, 0.01, 0.20, 0.06])  # [left, bottom, width, height] in
figure coords
    bprev = Button(axes2, "previous point", color="yellow")
    bprev.on_clicked(PoinPlotBackward)

    #axes4 = plt.axes([0.05, 0.6, 0.06, 0.06])  # [left, bottom, width, height] in
figure coords
    axes4 = plt.axes([0.05, 0.5, 0.06, 0.06]) # [left, bottom, width, height] in
figure coords
    LP = Button(axes4, "A1", color="yellow")
    LP.on_clicked(LeftConfig)

    axes6 = plt.axes([0.12, 0.5, 0.06, 0.06]) # [left, bottom, width, height] in
figure coords
    RP = Button(axes6, "A2", color="yellow")
    RP.on_clicked(RightConfig)


    axes8 = plt.axes([0.81, 0.07, 0.16, 0.06]) # [left, bottom, width, height] in
figure coords
    Save = Button(axes8, "Save angles", color="yellow")
    Save.on_clicked(save_angles)

    axes9 = plt.axes([0.04, 0.3, 0.16, 0.06]) # [left, bottom, width, height] in
figure coords
    SaveToTxt = Button(axes9, "Save to txt", color="yellow")
    SaveToTxt.on_clicked(save_to_TXT)



    plt.show()
```

3 Programa atvaizduojanti mechanizmo judėjimą

```
from pickle import NONE
import sys
```

```
import math
from tkinter import Checkbutton
from typing import Text
import matplotlib.pyplot as plt
from matplotlib.widgets import Button, CheckButtons
import time
import numpy as np
from sklearn.setup import configuration

l0 = 20
l1 = 20
l2 = 50
EndPoints = []
TurningAngles = []
LeftAngle = RightAngle = 0.0
Indexnumber = None
configurationL = 1
configurationR = 1
#LPC, LMC, RPC, RMC = TYR, False, False, False
# load data, ensure it's 2D (N x 2)
loaded_array = np.loadtxt("Angles_astuonetas1.txt", delimiter=",")
loaded_array = np.atleast_2d(loaded_array)

def calc_angles(x, y):
    global shoulderLM, shoulderLP, shoulderRM, shoulderRP
    # Angle from left shoulder to end effector
    beta1 = math.atan2(y, (l0 + x))

    # Angle from right shoulder to end effector
    beta2 = math.atan2(y, (l0 - x))

    # Alpha angle pre-calculations
    alpha1_calc = (l1**2 + (((l0 + x)**2 + y**2)) - l2**2) / (2 * l1 * math.sqrt((l0 +
x)**2 + y**2))
    alpha2_calc = (l1**2 + (((l0 - x)**2 + y**2)) - l2**2) / (2 * l1 * math.sqrt((l0 -
x)**2 + y**2))

    if alpha1_calc > 1 or alpha1_calc < -1 or alpha2_calc > 1 or alpha2_calc < -1:
        print("Unreachable coordinates")
        return (-999, -999)

    alpha1 = math.acos(alpha1_calc)
    alpha2 = math.acos(alpha2_calc)

    shoulderLM = beta1 - alpha1
    shoulderLP = beta1 + alpha1
    shoulderRM = math.pi - beta2 - alpha2
    shoulderRP = math.pi - beta2 + alpha2


def draw_ALLarms(x, y, shoulderLM, shoulderLP, shoulderRM, shoulderRP):

    global a1, a2, p1, p2, l_a1, l_a2, l_b1, l_b2, a12, a22, p12, p22, l_a12, l_a22,
l_b12, l_b22, configurationL, configurationR

    ax.cla()
    ax.set_xlim(-100, 100)
    ax.set_ylim(-100, 100)
    ax.grid(True)
    ax.scatter(loaded_array[:, 0], loaded_array[:, 1], color='lightblue', marker='o')

    #-------------------------------------------------------------
    #draw points
    #-------------------------------------------------------------
    A1 = (-l0,0)
```

```
    A2 = (l0,0)

    #P1 left,P2 right joints with Plus configuration
    #P12 left,P22 right joints with Minuse configuration

    P1 = (-l0 + l1*math.cos(shoulderLP), l1*math.sin(shoulderLP))
    P2 = (l0 + l1*math.cos(shoulderRP), l1*math.sin(shoulderRP))
    P12 = (-l0 + l1*math.cos(shoulderLM), l1*math.sin(shoulderLM))
    P22 = (l0 + l1*math.cos(shoulderRM), l1*math.sin(shoulderRM))
    if configurationR == 1:
        p2, = ax.plot(P2[0], P2[1], 'ro')
        a2, = ax.plot(A2[0], A2[1], 'bo')
        l_b1, = ax.plot([A2[0], P2[0]], [A2[1], P2[1]], 'b')
        l_b2, = ax.plot([P2[0], x],     [P2[1], y],     'r')
    else:
        a22, = ax.plot(A2[0], A2[1], 'go')
        p22, = ax.plot(P22[0], P22[1], 'mo')
        l_b12, = ax.plot([A2[0], P22[0]], [A2[1], P22[1]], 'g')
        l_b22, = ax.plot([P22[0], x],     [P22[1], y],     'm')

    if configurationL == 1:
        p1, = ax.plot(P1[0], P1[1], 'ro')
        a1, = ax.plot(A1[0], A1[1], 'bo')
        l_a1, = ax.plot([A1[0], P1[0]], [A1[1], P1[1]], 'b')
        l_a2, = ax.plot([P1[0], x],     [P1[1], y],     'r')
    else:
        p12, = ax.plot(P12[0], P12[1], 'mo')
        a12, = ax.plot(A1[0], A1[1], 'go')
        l_a12, = ax.plot([A1[0], P12[0]], [A1[1], P12[1]], 'g')
        l_a22, = ax.plot([P12[0], x],     [P12[1], y],     'm')

    # draw to canvas (non-blocking)
    fig.canvas.draw_idle()


current_index = 0

#create figure / axes and an empty scatter to be updated
fig = plt.figure()
ax = fig.subplots()
plt.subplots_adjust(left=0.3, bottom=0.25)

ax.set_xlim(-100, 100)
ax.set_ylim(-100, 100)
ax.grid(True)

print("Starting animation...")
print("loaded_array.shape =", loaded_array.shape)

# enable interactive mode so the plot updates without blocking
plt.ion()

for i in range(len(loaded_array)):

    row = loaded_array[i]
    # ensure there are enough columns in the file; if not adjust accordingly
    if row.size < 2:
        raise ValueError("Expected at least 2 columns (x,y) in Angles.txt")
    x = float(row[0])
    y = float(row[1])
    # if file contains config columns, read them; otherwise use defaults
    if row.size >= 4:
        configurationL = int(row[2])
        configurationR = int(row[3])
    calc_angles(x, y)
```

```
    draw_ALLarms(x, y, shoulderLM, shoulderLP, shoulderRM, shoulderRP)
    print(i)
    # use plt.pause so GUI events are processed and the figure updates
    plt.pause(0.2)


# optional: keep window open at the end
plt.ioff()
plt.show()
```

4 Genetinis algoritmas

```
import random
from unittest import result
import matplotlib.pyplot as plt
import numpy as np
import psutil
import os
from datetime import datetime
import time
import pandas as pd

def eksportuoti_i_excel(filename, data_dict):
    # Tikriname, ar failas jau egzistuoja
    if os.path.exists(filename):
        # Jei egzistuoja, nuskaitome esamus duomenis
        df_existing = pd.read_excel(filename)
        # Sukuriame naują eilutę iš gautų duomenų
        df_new = pd.DataFrame([data_dict])
        # Sujungiame senus duomenis su naujais
        df_final = pd.concat([df_existing, df_new], ignore_index=True)
    else:
        # Jei failo nėra, sukuriam naują DataFrame
        df_final = pd.DataFrame([data_dict])

    # Įrašome į Excel
    df_final.to_excel(filename, index=False)
    print(f"Duomenys sėkmingai eksportuoti į {filename}")

def fitness_function(params, l1l2distdiff, Yboundarys, XTargetArray):

    l0, l1, l2, FixedPointZ = params

    YTargetSum = 100
    FixedPointX = 0
    FixedPointY = 50
    WingLenght = 100
    x_lim = l0+l1+l2
    StepSize = 1
    AddPoints = []
    ReachPoints = []
    WingEndPointsArray = []
    DistFromX0ToXEndPointArray = []
    X_lenght_array = []

    if abs(l1-l2) < l0 or l0+l1>l2 or l1 < l2 / l1l2distdiff: #2*l1>l2
        YResult = -1000000
        XResult = -1000000
        #return [], YResult, XResult
        return XResult, []

    #calculate highest and lowest reachable points
    #highest point
    Y_highPoint = np.sqrt((l1 + l2)**2 - l0**2)
```

```
    #lowest point
    Y_lowPoint = np.sqrt((l1 - l2)**2 - l0**2)


    FiveBarEndPoint = np.array([0, Y_lowPoint, 0])
    WinglineVector = FiveBarEndPoint - (FixedPointX, FixedPointY, FixedPointZ)
    WinglineVectorNormalized = WinglineVector/np.linalg.norm(WinglineVector)
    WingEndPointCoordinates = FiveBarEndPoint +
(WingLenght*(WinglineVectorNormalized))
    YWing_highPoint = WingEndPointCoordinates[1]

    FiveBarEndPoint = np.array([0, Y_highPoint, 0])
    WinglineVector = FiveBarEndPoint - (FixedPointX, FixedPointY, FixedPointZ)
    WinglineVectorNormalized = WinglineVector/np.linalg.norm(WinglineVector)
    WingEndPointCoordinates = FiveBarEndPoint +
(WingLenght*(WinglineVectorNormalized))
    YWing_lowPoint = WingEndPointCoordinates[1]

    #calulate distance between Y highest point and Y lowest point, which is the
maximum possible Y lenght of the wing end point trajectory
    Y_lenght = abs(YWing_highPoint - YWing_lowPoint)
    #print("Y_highPoint: ", YWing_highPoint, "YWing_lowPoint: ", YWing_lowPoint, " =
", Y_lenght)
    if abs(YTargetSum-Y_lenght) > Yboundarys:
        XResult = -1000000
        return XResult, []


    #calculate step size to increase searching reachable points in x direction
    Y_Value = np.linspace(Y_lowPoint, Y_highPoint, 8)

    for i in range(0, 8):
        y = Y_Value[i]
        for x in range(0, int(x_lim), int(StepSize)):
            r1 = np.sqrt((x + l0)**2 + y**2)
            r2 = np.sqrt((l0 - x)**2 + y**2)
            cond1 = (r1 >= abs(l1 - l2)) and (r1 <= (l1 + l2))
            cond2 = (r2 >= abs(l1 - l2)) and (r2 <= (l1 + l2))
            if cond1 and cond2:
               AddPoints.append((x,y))
            else:
                if len(AddPoints) > 0:
                    ReachPoints.append(AddPoints[-1])
                else:
                     ReachPoints.append((0,y))
                break

    for i in range(len(ReachPoints)):
        x = ReachPoints[i][0]
        y = ReachPoints[i][1]
        FiveBarEndPoint = np.array([x, y, 0])
        WinglineVector = FiveBarEndPoint - (FixedPointX, FixedPointY, FixedPointZ)
        WinglineVectorNormalized = WinglineVector/np.linalg.norm(WinglineVector)
        WingEndPointCoordinates = FiveBarEndPoint +
(WingLenght*(WinglineVectorNormalized))
        WingEndPointsArray.append(WingEndPointCoordinates)

    WingEndPointsArray = np.array(WingEndPointsArray)

    for i in range(len(WingEndPointsArray)):
        X_lenght_array.append(WingEndPointsArray[i,0])

    XLenghtRes = []
    for i in range(len(X_lenght_array)):
        XLenghtRes.append(-abs(X_lenght_array[i]-XTargetArray[i]))
```

```
    XResult = np.sum(XLenghtRes)

    return XResult, WingEndPointsArray
def create_initial_population(size, lever_lower_bound, lever_upper_bound,
fixPoint_lower_bound, fixPoint_upper_bound):
    population = []
    for _ in range(size):
        individual = (random.uniform(lever_lower_bound, lever_upper_bound),
                      random.uniform(lever_lower_bound, lever_upper_bound),
                      random.uniform(lever_lower_bound, lever_upper_bound),
                      random.uniform(fixPoint_lower_bound, fixPoint_upper_bound))
        population.append(individual)
    return population

def selection(population, fitnesses, tournament_size=3):
    selected = []
    for _ in range(len(population)):
        tournament = random.sample(list(zip(population, fitnesses)), tournament_size)
        winner = max(tournament, key=lambda x: x[1])[0]
        selected.append(winner)
    #print("selected: ", len(selected))
    return selected

def crossover(parent1, parent2):
    alpha = random.random()
    child1 = tuple(alpha * p1 + (1 - alpha) * p2 for p1, p2 in zip(parent1, parent2))
    child2 = tuple(alpha * p2 + (1 - alpha) * p1 for p1, p2 in zip(parent1, parent2))
    return child1, child2

def mutation(individual, mutation_rate, lever_lower_bound, lever_upper_bound,
fixPoint_lower_bound, fixPoint_upper_bound):
    individual = list(individual)
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            mutation_amount = random.uniform(-1, 1)
            individual[i] += mutation_amount
            # Ensure the individual stays within bounds
            if i <= 3:
                individual[i] = max(min(individual[i], lever_upper_bound),
lever_lower_bound)
            else:
                individual[i] = max(min(individual[i], fixPoint_upper_bound),
fixPoint_lower_bound)
    return tuple(individual)


def genetic_algorithm(population_size, lever_lower_bound, lever_upper_bound,
fixPoint_lower_bound, fixPoint_upper_bound, l1l2distdiff, Yboundarys, XTargetArray,
generations, mutation_rate):
    population = create_initial_population(population_size, lever_lower_bound,
lever_upper_bound, fixPoint_lower_bound, fixPoint_upper_bound)

    best_performers = []
    #all_populations = []
    BestReachArr = []

    for generation in range(generations):
        fitnesses = [fitness_function(ind, l1l2distdiff, Yboundarys, XTargetArray)[0]
for ind in population]

        # Store the best performer of the current generation
        best_fitness = max(fitnesses)
        best_individual = population[fitnesses.index(best_fitness)]
        best_performers.append((best_individual, best_fitness))
        population = selection(population, fitnesses)
```

```
        next_population = []
        #print('population size: ', len(population))
        for i in range(0, len(population), 2):
            parent1 = population[i]
            parent2 = population[i + 1]
            child1, child2 = crossover(parent1, parent2)
            next_population.append(mutation(child1, mutation_rate, lever_lower_bound,
lever_upper_bound, fixPoint_lower_bound, fixPoint_upper_bound))
            next_population.append(mutation(child2, mutation_rate, lever_lower_bound,
lever_upper_bound, fixPoint_lower_bound, fixPoint_upper_bound))

        # Replace the old population with the new one, preserving the best individual
        next_population[0] = best_individual
        population = next_population
        #next_population = []
        print(f"Generation {generation + 1}: Best Individual = {best_individual},
Fitness = {best_fitness}")

    return best_performers

def Genetic_algorithm_run():
    start_time = time.time()
    Best_solution = genetic_algorithm(population_size ,lever_lower_bound,
lever_upper_bound, fixPoint_lower_bound, fixPoint_upper_bound, l1l2distdiff,
Yboundarys, XTargetArray, generations, mutation_rate)
    #print("Best solution: ", Best_solution[-1])
    Best_solutionarr = Best_solution[-1][0]


    end_time = time.time()

    ProgDur = end_time - start_time

    result = fitness_function(Best_solutionarr, l1l2distdiff, Yboundarys,
XTargetArray)[1]
    #print("result",result)
    if len(result) > 0:
        Y_lenght = abs(result[-1][1] - result[0][1])
        targetPath =  result.copy()
        targetPath[:, 0] = XTargetArray
    else:
        Y_lenght = 0
        targetPath = []


    #print("Ylenght: ", Y_lenght)
    fitnes_value_arr = [round(gen[1], 4) for gen in Best_solution]
    #print ("Best_solution: ", Best_solution)
    #print("fitnes_value_arr: ", fitnes_value_arr)

    excel_data = {
        "Data ir laikas": datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
        "Populiacijos dydis": population_size,
        "Generacijos": generations,
        "Mutacija": mutation_rate,
        "Svirčių rėžiai": f"{lever_lower_bound}-{lever_upper_bound}",
        "Z rėžiai": f"{fixPoint_lower_bound}-{fixPoint_upper_bound}",
        "l1/l2 riba": l1l2distdiff,
        "Y paklaida": Yboundarys,
        "l0": Best_solution[-1][0][0],
        "l1": Best_solution[-1][0][1],
        "l2": Best_solution[-1][0][2],
        "Z": Best_solution[-1][0][3],
        "Ivertinimas (Fitness)": Best_solution[-1][1],
        "Geriausiu sprendimu masyvas": fitnes_value_arr,
```

```
        "Gautas Y ilgis": Y_lenght,
        "Programos trukmė (s)": ProgDur
    }

    eksportuoti_i_excel("optimizavimo_rezultatai.xlsx", excel_data)

    fig = plt.figure()
    ax = plt.axes(projection='3d')
    plt.subplots_adjust(left=0.3, bottom=0.25)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    axislim = 80
    #ax.set_ylim(result[0][1]-10, result[-1][1]+10)
    ax.set_xlim(-10, 100)
    ax.set_zlim(-axislim, axislim)
    ax.view_init(elev=90, azim=-90)

    if len(result) > 0:
        resultPath = ax.scatter(result[:, 0], result[:, 1], result[:, 2], 'g')
        targetPath = ax.scatter(targetPath[:, 0], targetPath[:, 1], targetPath[:, 2],
'b')
        plt.savefig(os.path.join("C:/Users/laury/OneDrive/Documents/mokslai/MA
Mechanikos inžinerija/magistrinis darbas/Genetinis algoritmas plasnojimo
mechanizmui/plots", f"galutinis_mechanizmas{datetime.now().strftime('%Y-%m-%d
%Hh%Mm%Ss')}.png"), dpi=300)
        #plt.savefig(os.path.join("C:/Users/laury/OneDrive/Documents/mokslai/MA
Mechanikos inžinerija/magistrinis darbas/Genetinis algoritmas plasnojimo
mechanizmui/plots", f"galutinis_mechanizmas{datetime.now().strftime('%Y-%m-%d
%Hh%Mm%Ss')}.png"), dpi=300)
        print(f"Diagrama išsaugota kaip
galutinis_mechanizmas{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}.png")
        #plt.show()

#-- Main program ------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------

population_size = 2000
lever_lower_bound = 5
lever_upper_bound = 50
fixPoint_lower_bound = 5
fixPoint_upper_bound = 50
XTargetArray = [0, 21, 31, 37, 38, 31, 15, 0]
l1l2distdiff = 10
Yboundarys = 5
generations = 300
mutation_rate = 0.2



for j in range (1, 11, 1):
    Genetic_algorithm_run()
```