



VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

Andrius STONYS

**ŽVILGSNIU VALDOMOS KLAVIATŪROS PRITRAUKIMO ALGORITMO
TYRIMAS**
**RESEARCH OF STICKING ALGORITHM IN GAZE-CONTROLLED
KEYBOARD**

Magistro baigiamasis darbas

Elektronikos inžinerijos studijų kryptis
Elektronikos studijų programa, valstybinis kodas 621H61003
Kompiuterizuotų elektroninių sistemų specializacija

Vilnius, 2013

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas

(parašas)
prof. habil. dr. R. Martavičius
2013 m. _____ mėn. ____ d.

Andrius STONYS

**ŽVILGSNIU VALDOMOS KLAVIATŪROS PRITRAUKIMO ALGORITMO
TYRIMAS**
**RESEARCH OF STICKING ALGORITHM IN GAZE-CONTROLLED
KEYBOARD**

Magistro baigiamasis darbas

Elektros inžinerijos studijų kryptis
Elektronikos studijų programa, valstybinis kodas 621H61003
Kompiuterizuotų elektroninių sistemų specializacija

Vadovas	doc. dr. D. Miniotas (mokslinis vardas ir laipsnis, vardas, pavardė)	_____ (parašas)	_____ (data)
Konsultantas	lekt. V. Buivydienė (mokslinis vardas ir laipsnis, vardas, pavardė)	_____ (parašas)	_____ (data)

Vilniaus Gedimino technikos universitetas
Elektronikos fakultetas
Elektroninių sistemų katedra

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

Antrosios pakopos studijų **Elektronikos inžinerijos** programos baigiamasis darbas

Pavadinimas **Žvilgsniu valdomos klaviatūros pritraukimo algoritmo tyrimas**

Autorius **Andrius Stonys**

Vadovas **doc. dr. Darius Miniotas**

Kalba: lietuvių

Anotacija

Žvilgsniu valdomos klaviatūros pritraukimo algoritmo tyrimas. Magistro baigiamasis darbas elektronikos inžinerijos laipsniui. Vilniaus Gedimino technikos universitetas. Vilnius, 2013, 52 p., 43 iliustr., 5 lent., 23 bibl., 3 priedai.

Šio darbo tikslas – ištirti pritraukimo algoritmo įtaką rašymo virtualia klaviatūra kokybei, greičiui ir efektyvumui. Tirti pasirinkti trys pritraukimo savybėmis pasižymintys algoritmai: pritraukimo, traukos lauko ir greičio mažinimo. Atskleidus kiekvieno jų privalumus ir trūkumus, pasiūlytas naujas algoritmas žvilgsnio taško stabilumui išlaikyti. Įgyvendinus šiuos algoritmus taikomojoje programoje EyeTyping ir atlikus eksperimentą, nustatyta, kad tirti algoritmai rašymo kokybei įtakos praktiškai neturi – ji kinta ne daugiau kaip 3 – 4 %. Pastebėta, kad pritraukimo (sutapdinimo su taikinio centru) algoritmas ryškios įtakos rašymui virtualiąja klaviatūra neturi, o didžiausias rašymo greitis – 13,3 žodžių per minutę – ir efektyvumas – 38,9 % – pasiekti taikant traukos lauko algoritmą, kai traukimo stiprio koeficientas 0,6.

Prasminiai žodžiai

žvilgsnio taškas, žiūros taškas, virtuali klaviatūra, žvilgsnio sekimas, greičio mažinimo algoritmas, traukos lauko algoritmas, pritraukimo algoritmas

Vilnius Gediminas Technical University
Faculty of Electronics
Department of Electronic Systems

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

Master Degree Studies **Electronics Engineering** study programme Final Work

Title **Research of Sticking Algorithm in Gaze-Controlled Keyboard**

Author **Andrius Stonys**

Academic supervisor **Assoc Prof Dr Darius Miniotas**

Thesis language:
Lithuanian

Annotation

Research of Sticking Algorithm in Gaze-Controlled Keyboard. Master thesis for Electronics Engineering master degree. Vilnius Gediminas technical university. Vilnius, 2013. Thesis consist of: 52p text without appendixes, 43 pictures, 5 tables, 23 bibliographical entries, 3 appendixes.

The main purpose of master thesis – research of sticking (a.k.a. warping to target center) algorithm's influence on typing with virtual gaze-controlled keyboard. Three gaze-point stabilization methods were analyzed: sticking, force field and speed reduction algorithms. Advantages and disadvantages were revealed during the process of the analyzation and new algorithm was proposed. All methods were realized in EyeTyping application. The experiment indicated, that sticking algorithm does not have strong influence on typing with gaze-controlled keyboard, whereas best results were obtained when using force field algorithm with force field coefficient $s = 0.6$.

Keywords

gaze mapping, gaze point, eyetyping, gaze-controlled keyboard, speed reduction, force field, warping to target center, sticking algorithm

TURINYS

Žymenys ir santrumpos	6
1. ĮVADAS	7
1.1. Darbo aktualumas ir tikslas.....	7
1.2. Darbo uždaviniai	8
1.3. Naudoti tyrimo ir analizės metodai.....	8
1.4. Darbo naujumas ir praktinė nauda	8
1.5. Darbo struktūra	9
2. ŽVILGSNIU PAREMTOS ŽMOGAUS-KOMPIUTERIO SĄSAJOS ANALITINĖ APŽVALGA	10
2.1. Aparatinė įranga.....	10
2.2. Programiniai akies judesių apdorojimo metodai.....	13
3. PRITRAUKIMO ALGORITMŲ ANALIZĖ	20
3.1. Pritraukimo algoritmas.....	20
3.2. Traukos lauko algoritmas.....	21
3.3. Greičio mažinimo algoritmas.....	25
3.4. Traukimas pusiau kraštinės kryptimi	29
4. ALGORITMO ĮGYVENDINIMAS TAIKOMOJOJE PROGRAMOJE EYETYPING	33
4.1. EyeTyping programos veikimas	33
4.2. Taško padėties nustatymas.....	36
4.3. Pritraukimo algoritmo įgyvendinimas	39
5. REALIZUOTO METODO TYRIMO REZULTATAI	41
5.1. Tyrimo eiga.....	41
5.2. Tyrimo rezultatai.....	45
6. APIBENDRINIMAS IR IŠVADOS	50
Literatūra	52
PRIEDAI	54
1 priedas. Algoritmų veikimo simuliacijos <i>Matlab</i> programiniu paketu skriptas.....	54
2 priedas. Paprogramės, paruoštos algoritmų įdiegimui į taikomąją programą EyeTyping	60
3 priedas. Pranešimo 16-oje Lietuvos jaunųjų mokslininkų konferencijoje pažymos kopija.....	67

Žymenys ir santrumpos

$\vec{n}$	naujo (perskaičiuoto) taško vektorius
$\vec{a}$	esamo (neperskaičiuoto) taško vektorius
$\vec{f}$	objekto centro vektorius
$\vec{p}$	prieš tai buvusio taško vektorius
s	traukimo stiprio koeficientas
r	greičio mažinimo koeficientas
n_x	naujo taško X koordinatė
n_y	naujo taško Y koordinatė

SR 3/6/9 (angl. *Speed reduction*)

Greičio mažinimo algoritmas, su greičio mažinimo koeficientu $r = 0,3; 0,6; 0,9$

FF 3/6 (angl. *Force field*)

Traukos laugo algoritmas, su traukimo stiprio koeficientu $s = 0,3; 0,6$

TPK 3/6/9 (Traukimas Pusiaukraštinės Kryptimi)

Traukimo pusiaukraštinės kryptimi algoritmas, su traukimo stiprio koeficientu $s = 0,3; 0,6; 0,9$

1. ĮVADAS

1.1. Darbo aktualumas ir tikslas

Elektronikos ir kompiuterinės technikos tobulėjimas šiandien yra labai spartus – didinama įtaisų greitimeika, mažinami matmenys, suvartojama galia. Taip pat keičiamas ir jų valdymo pobūdis. Atsisakoma kompiuterinių pelių, klaviatūros keičia lietimui jautriame ekrane pateikiami mygtukai, tobulinant vaizdų atpažinimo technologijas, pritaikomi ir kitokio pobūdžio rodomieji įtaisai ar objektai, pavyzdžiui – žmogaus akys.

Akies judesiai pradėti tyrinėti dar XIX a. pirmoje pusėje. Iš pradžių buvo galvota, kad akys juda tolygiai, tačiau 1879 m. atliktas Louiso Émile Javalio skaitymo tyrimas parodė, kad akies judėjimas susideda iš daugybės trumpų sustojimų, vadinamų fiksacijomis (angl. *Fixations*) [1, p. 43] ir greitų obuolio pasisukimų, vadinamų sakadomis (angl. *Saccades*) [1, p. 42]. Buvo siekiama išsiaiškinti, kur įvyksta fiksacijos, kiek jos trunka, ar grįžtama į prieš tai buvusią padėtį. Todėl buvo pradėti kurti akies judesių sekimo aparatai, dar vadinami žvilgsnio sekikliais (angl. *Eye trackers*).

Tačiau šie įrenginiai buvo skirti ne tik tyrimiesiems darbams. XX a. antroje pusėje buvo iškelta interaktyvios, žvilgsniu paremtos sąsajos tarp žmogaus ir kompiuterio idėja. Pradėjus sparčiai populiarėti kompiuteriams, ši idėja buvo įgyvendinta. Sąsaja netrukus rado ir tikslinę naudotojų grupę – žmones, turinčius bendravimą bei kasdienių darbų atlikimą apsunkinančias negalias.

Kad sąsaja būtų sėkmingai įgyvendinta, reikėjo išspręsti tiek technines, tiek su natūralia žmogaus fiziologija bei nevalinga elgsena susijusias problemas. Akies judesių tyrinėjimų metu buvo pastebėta, kad net esant fiksacijoms, akis šiek tiek virpa – vyksta mikrosakados [1, p. 43-44]. Nors žmogus to ir nejaučia, žvilgsnio sekimo įrangos formuojamiems signalams šis reiškinys daro nemažą įtaką. Jei tam tikru dažniu būtų fiksuojamas žmogaus žvilgsnis ir užfiksuoti žiūros taškai būtų atvaizduojami kompiuterio ekrane, išryškėtų šių taškų išsibarstymas. Ši problema ypač apsunkino naudojimąsi sąsaja ir kompiuteryje veikiančios taikomosios programos valdymą. Tuo tikslu buvo pradėti kurti ir tirti įvairūs žvilgsnio taškų išsibarstymui sumažinti skirti metodai.

Baigiamajame magistro darbe pasirinkta tyrinėti paprasčiausio, tačiau mažiausiai literatūros šaltiniuose aprašomo pritraukimo (angl. *Sticking*), dar įvardijamo kaip priskyrimo centrui (angl. *Warping to target centre*), algoritmą. Metodą pasirinkta įdiegti į taikomąją programą *Eyotyping*, kuri atlieka virtualios klaviatūros funkciją. Darbo tikslas – išsiaiškinti šio metodo įtaką rašymo žvilgsniu valdoma klaviatūra kokybei, greičiui bei efektyvumui, o tyrimo ir analizės rezultatus palyginti su kitais dviem pritraukimo savybėmis pasižyminčiais algoritmais – traukos lauko (angl. *Force field*) ir greičio mažinimo (angl. *Speed reduction*).

1.2. Darbo uždaviniai

Kad būtų pasiektas darbo tikslas, pirmiausia būtina atlikti analitinę apžvalgą ir analogiškų metodų analizę, išsiaiškinti pagrindines žvilgsnio sekimo technikas, apžvelgti jau esamus programinius žvilgsnio taškų apdorojimo ir klasifikavimo metodus. Analitinės apžvalgos metu būtina nustatyti šių techninių ir programinių sprendimų privalumus bei trūkumus. Tik atlikus analizę, bus galima įvertinti tyrimui pasirinktos įrangos tinkamumą interaktyviai žvilgsniu paremtai sąsajai tarp žmogaus ir kompiuterio realizuoti.

Antra, būtina išsiaiškinti tiriamajame darbe pasirinktos naudoti taikomosios programos *EyeTyping* veikimą. Išanalizavus programos kodą ir duomenų apdorojimo žingsnių eiliškumą, bei išsiaiškinus pritraukimo algoritmo specifiką, bus galima paruošti ir į taikomąją programą įdiegti tiriamą metodą įgyvendinantį kodą. Taip pat parašyti traukos lauko bei greičio mažinimo algoritmus įgyvendinančius programinius kodus.

Darbui parengus technines ir programines tyrimo priemones, toliau būtina paruošti tyrimo planą ir nustatyti kriterijus, kuriais remiantis bus įvertinti gautieji eksperimento duomenys. Svarbu atlikti detalią surinktos informacijos analizę, kuri atskleistų tiriamo algoritmo privalumus, trūkumus ir tobulinimo galimybes.

Darbo tikslui pasiekti būtina išsamiai pateikti eksperimento rezultatus, kurie leistų padaryti argumentuotą išvadą dėl tirtų algoritmų įtakos rašymo žvilgsniu valdoma klaviatūra kokybei, greičiui ir efektyvumui.

1.3. Naudoti tyrimo ir analizės metodai

Tyrimas buvo vykdomas dviem etapais. Pirmiausia, visi tiriami metodai buvo išanalizuoti teoriniu požiūriu, išaiškintos jų formulės, *Matlab* programiniu paketu ir specialiai tam paruoštu skriptu buvo simuliuojamas jų veikimas, keičiant formulėje esančius parametrus. Po algoritmų analizės, tiriami algoritmai buvo įdiegti į pasirinktą taikomąją programą ir jų veikimas išbandytas praktiškai. Buvo sudarytas tyrimų eigos planas, kuriuo remiantis buvo atliktas rašymo eksperimentas. Gauti rezultatai buvo perkelti į *MS Office Excel* aplinką, kur buvo susumuoti, apibendrinti ir darbe pateikti grafiškai.

1.4. Darbo naujumas ir praktinė nauda

Pritraukimo, arba prirskyrimo centrui, algoritmas yra vienas paprasčiausių žvilgsnio taškų apdorojimo metodų, tačiau, skirtingai nei traukos lauko ar greičio mažinimo metodai, literatūros šaltiniuose jis yra mažiausiai aprašomas. Išanalizavus tiriamų metodų specifiką ir atskleidus kiekvieno iš jų privalumus ir trūkumus, šiame magistro darbe taip pat buvo pasiūlytas naujas žvilgsnio taškų apdorojimo metodas, turintis traukimo savybių. Teoriniu požiūriu, juo

išsprendžiamos problemos, kurios atsiskleidžia naudojant pritraukimo, traukos lauko ir greičio mažinimo algoritmus.

Daugelyje literatūros šaltinių aprašomuose algoritmų, skirtų žvilgsnio taško stabilumui išlaikyti, tyrimuose naudojamos specialiai tam paruoštos taikomosios programos [9, 10, 13, 15]. Tiriami tokie specifiniai parametrai kaip, objekto ribų kirtimo skaičius, stebint vieną objektą, laikas kurį žvilgsnio taškas buvo objekte ir už jo ribų, bei panašūs [9, 10, 13]. Tyrimų naudojant virtualiosios klaviatūros programą *EyeTyping* nemažai buvo atlikta Temperės universitete, Suomijoje. Taikomąja programa galima realizuoti slankiąją klaviatūrą, kuria mygtukai pateikiami po vieną eilę ekrane. Šis funkcionalumas ir jo įtaka rašymui yra pagrindinis straipsniuose aprašomo tyrimo objektas [14, 17]. Literatūroje aprašomi tyrimai yra atliekami įvairiomis sąlygomis ir naudojant skirtingą įrangą, todėl rezultatų palyginimas ne visada gali privesti prie teisingų išvadų.

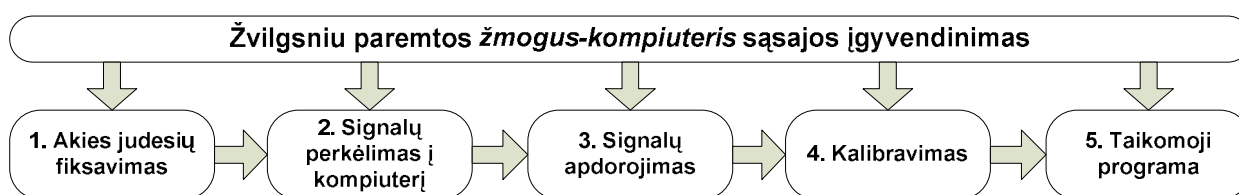
1.5. Darbo struktūra

Darbą sudaro šeši skyriai. Po tiriamojo darbo pristatymo, antrame skyriuje pateikta analitinė žvilgsniu paremtos sąsajos tarp žmogaus ir kompiuterio apžvalga, kuri apima aparatinės įrangos ir programinių akies judesių apdorojimo metodų analizę. Trečiajame darbo skyriuje pateikta detalesnė teorinė pasirinktų tirti algoritmų analizė, kurią sudaro formulių išaiškinimas bei simuliacijos *Matlab* programiniu paketu rezultatai. Šiame skyriuje taip pat pateikta siūlomo naujo traukimo savybių turinčio algoritmo teorinė analizė. Ketvirtame darbo skyriuje aprašytas taikomosios programos *EyeTyping* veikimas bei pateiktas algoritmų įgyvendinimo joje aprašymas. Penktąjį skyrių sudaro nuoseklios tyrimų eigos aprašas ir atliktų eksperimentų rezultatai, pateikti lentelėse bei grafiškai. Tiriamojo darbo rezultatai apibendrinti šeštajame skyriuje, pateikiamos išvados.

2. ŽVILGSNIU PAREMTOS ŽMOGAUS-KOMPIUTERIO SĄSAJOS ANALITINĖ APŽVALGA

Žvilgsniu paremtos interaktyviosios sąsajos tarp žmogaus ir kompiuterio įgyvendinimą galima išskaidyti į kelis uždavinius (2.1 pav.):

1. Akies judesių fiksavimas;
2. Signalų perkėlimas į kompiuterį;
3. Signalų apdorojimas;
4. Kalibravimas;
5. Taikomoji programa.



2.1 pav. Sąsajos įgyvendinimo žingsniai

Pirma, būtina rasti metodą, kuriuo būtų galima realiu laiku ir tiksliai nustatyti akies poziciją, išreikšti jos pokyčius elektriniais signalais ir juos perduoti į kompiuterinę įrangą. Priklausomai nuo to, kokiais signalais paversti akies judesiai, reikia parinkti programinius šių signalų apdorojimo metodus, kurie apimtų žvilgsnio taškų išskyrimą (pvz., X, Y koordinatėmis), jų atitikimą kompiuterio ekrane bei klasifikavimą į fiksacijas ir sakadas. Akivaizdu, kad kiekvieno iš šių uždavinių sprendinys turi būti sekančio uždavinio sprendimo atspirties taškas.

Šiame magistro darbo skyriuje pateikiama aparatinės akies judesių fiksavimo ir signalų perkėlimo į kompiuterį įrangos apžvalga. Aprašomi pagrindinių naudojamų technikų privalumai, trūkumai ir tinkamumas interaktyviai, žvilgsniu paremtai sąsajai tarp žmogaus ir kompiuterio realizuoti. Pasirinkus tinkamiausią žvilgsnio sekimo priemonę, toliau šiame skyriuje aprašoma programinė sąsajos dalis, apimanti signalų apdorojimo metodus, kalibravimo techniką bei žvilgsnio taškų klasifikavimo algoritmus.

2.1. Aparatinė įranga

Pagal akies judesių sekimo koncepciją yra skiriami keli okulografijos tipai: elektrookulografija, odenos kontaktinių lęšių/paieškos rite (angl. *Scleral Contact Lens/Search Coil*) paremtas metodas, foto/video okulografija bei vaizdo atpažinimu paremtas kombinuoto vyzdžio/ragenos atspindžio (angl. *Video-Based Combined Pupil/Corneal Reflection*) metodas [1, p. 51, 2].

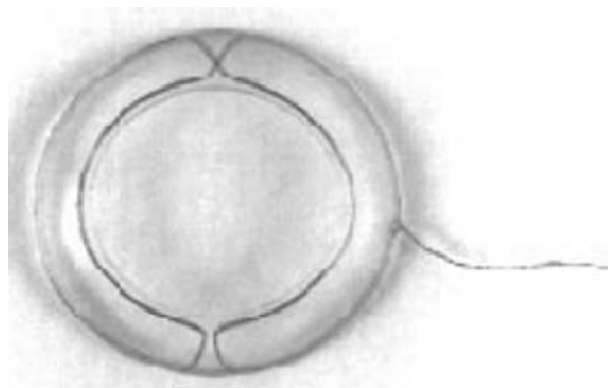
Elektrookulografijos (2.2 pav., a) pagrindas – įtampas, atsirandančios kintant akies pasisukimo kampui, matavimas. Ant odos, priešingose akių pusėse, pritvirtinus specialius elektrodus ir leidus

tekėti silpnai elektros srovei, buvo pastebėta, kad potencialų skirtumas tarp šių elektrodų, vartant akis, kinta. Šio pokyčio matavimas leido fiksuoti akies pasisukimus. Matavimų metu buvo registruojami 15-200 μV potencialai, su 20 $\mu\text{V}/^\circ$ akies judesiui nominaliu jautriu. Nors ši technika leidžia tiksliai fiksuoti akies pasisukimus, galvos judesiai ir jos pozicija yra neįvertinami. Todėl nustatyti objektą, į kurį sutelktas dėmesys, yra sudėtinga [1, p. 52].

Odenos kontaktinių lęšių/paieškos rite (2.2 pav. b) paremtas metodas yra tiksliausiai akies judesius fiksuojanti technika. Jos pagrindas – magnetinio lauko indukuotos elektrovaros paieškos ritėje matavimas. Vielinės ritės įmontuojamos į kontaktinius lęšius, kurie uždedami ant akies. Tyrimas vykdomas naudotojui sėdint specialioje kameroje, kurioje veikia elektromagnetinis laukas. Vartant akis, kartu judančiose ritėse indukuojama elektrovara, kurios dydis priklauso nuo akies pasisukimo kampo. Kontaktiniai lęšiai turi būti platūs, dengiantys rageną ir odeną, kad nepasireikštų lęšio dreifas akies obuolio plote. Ši akies judesių sekimo technika leido gauti net 5-10 sekundžių (angl. *Arcseconds*) į 5° paklaidą. Tačiau dirglumas akims ir diskomfortas yra pagrindiniai jos trūkumai. Taip pat, šiuo metodu fiksuojant akies pasisukimus, nėra įvertinami galvos judesiai. Todėl, kaip ir elektrookulografikos atveju, tyrimų metu reikalingos papildomos priemonės – specialūs stovai – galvos stabilumui išlaikyti [1, p. 53, 2].



a



b

2.2 pav. Elektrookulografijos metodas ir odenos kontaktinis lęšis/paieškos ritė [1, p. 52-53]

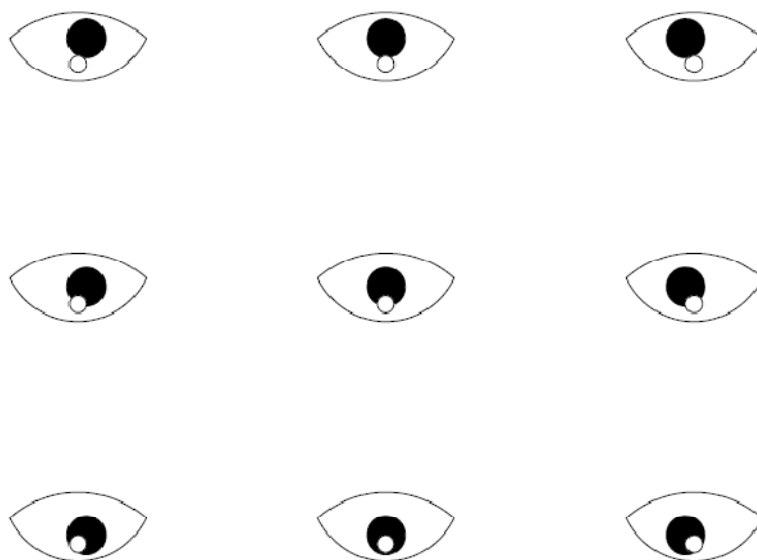
Foto arba video okulografija apima net kelis akių judesių fiksavimo metodus, paremtus tam tikrų akių savybių išskyrimu jų vartymo metu, pavyzdžiui, vyzdžio formos pokyčiu, rainelės žiedo (angl. *Limbus*) padėtimi akies plote ar atspindžiais nuo ragenos, kai į akis nukreipti arti esantys šviesos šaltiniai (dažniausiai infraraudonųjų spindulių). Nors kiekvienos šių technikų pagrindas yra skirtingas, jos priskiriamos vienai grupei, nes turi vieną bendrą trūkumą – neįvertina galvos judesių ir dėl to neleidžia vienareikšmiškai nustatyti dėmesio taško. Akių judesių analizė šiais metodais dažniausiai atliekama ne realiu laiku, o iš specialiai tam padaryto vaizdo įrašo [1, p. 54-55].

Kad būtų galima pakankamai tiksliai nustatyti dėmesio tašką, būtina išlaikyti stabilią galvos padėtį arba žvilgsnio sekimo metu matuoti kelių akies savybių pasikeitimus, pavyzdžiui fiksuoti ne tik šviesos šaltinio atspindį nuo ragenos, bet ir jo padėtį vyzdžio centro atžvilgiu. Tokie atspindžiai dar vadinami Purkinjė atspindžiais arba Purkinjė atvaizdais (angl. *Purkinje images*) (2.3 pav.).

Dėl akies sandaros yra išskiriami keturi Purkinjė vaizdų tipai:

- 1 – atspindys nuo priekinio ragenos paviršiaus;
- 2 – atspindys nuo galinio ragenos paviršiaus;
- 3 – atspindys nuo priekinio lęšiuko paviršiaus;
- 4 – atspindys nuo galinio lęšiuko paviršiaus.

Atspindys nuo priekinio ragenos paviršiaus yra aiškiausiai matomas, todėl vaizdo apdorojimo metu geriausiai išskiriamas. Dėl šios priežasties daugelyje žvilgsnio sekiklių yra naudojamas pirmasis Purkinjė atvaizdas [1, p. 55-59].



2.3 pav. Pirmasis Purkinjė vaizdas [1, p. 58]

Vėlesniuose, taip vadinamuose penktos kartos, žvilgsnio sekikliuose yra naudojami dvigubas Purkinjė atvaizdas (angl. *Dual-Purkinje image*), kurį sudaro atspindžiai nuo priekinio ragenos paviršiaus ir galinio lęšiuko paviršiaus [1, p. 56, 2, 3].

Vaizdo atpažinimu paremtas ragenos/vyždžio atspindžių metodas (2.4 pav., b) yra plačiausiai taikomas interaktyviajai žvilgsnio sąsajai įgyvendinti. Tai lėmė kelios priežastys:

- Naudojant šį metodą tiksliausiai galima nusakyti dėmesio tašką, net ir esant nedideliems galvos judesiams;
- Nereikia specializuotos kameros atspindžiams užfiksuoti. Apšvietus akis infraraudonąja spinduliuote, net ir naudojant internetinę vaizdo kamerą, galima išskirti atspindžių pokyčius;

Sąsaja gali būti įgyvendinta ir neinvaziniu būdu, t.y., ant žmogaus nereikia tvirtinti specialių elektrodų, kontaktinių lęšių ar kitų prietaisų [1, p. 55-59].



a



b

2.4 pav. Vaizdo atpažinimu paremtų sekiklių veikimas [1, p. 55]

2.2. Programiniai akies judesių apdorojimo metodai

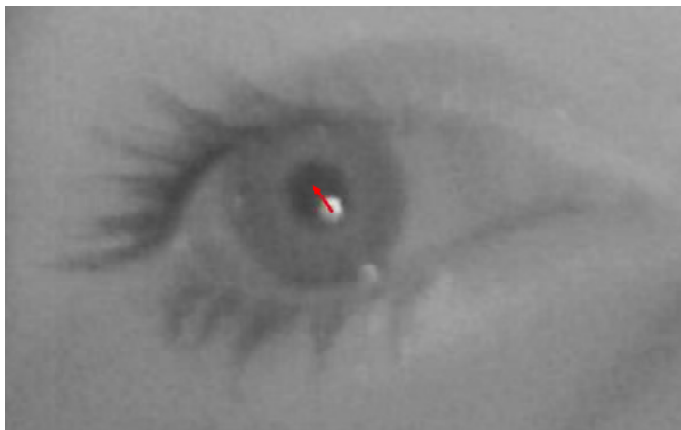
Šiame poskyryje daugiausia dėmesio bus skirta žvilgsnio taškų apdorojimo ir akies judesių klasifikavimo algoritmų apžvalgai, kai žvilgsnio taškai gaunami iš žvilgsnio sekiklio, kurio veikimas paremtas vaizdo atpažinimo metodu. Prieš tai trumpai bus aptarti signalų apdorojimo ir sistemos kalibravimo žingsniai.

Akies judesių fiksavimui naudojant vaizdo atpažinimu paremtą metodą, į kompiuterį patenka skaitmeniniai vaizdo signalai, kitaip tariant, vaizde esančių pikselių skaisčių vertės. Šiuo atveju, trečiasis išskirtas sąsajos įgyvendinimo žingsnis apima Purkinjė vaizdo išskyrimą ir žvilgsnio vektoriaus radimą. Žvilgsnio sekiklyje įmontuota kamera yra nukreipta į ekraną stebinčio žmogaus veidą. Iš gaunamo vaizdo yra iškerpama tik akių sritis. Tai galima atlikti remiantis įprasta proporcijos taisykle, sudalinant žmogaus veidą į keturias vienodo dydžio sritis. Kiekviena jų užima $\frac{1}{4}$ veido ploto, o akys, laikoma, kad yra antrame ketvirtyje. Taip pašalinama perteklinė informacija ir tolimesni skaičiavimai atliekami tik šioje srityje [4].

Toliau iš vaizdo išskiriamas Purkinjė atspindys. Pirmiausia randamas infraraudoniosios šviesos atspindys nuo ragenos. Panaudojant *Canny* briaunų aptikimą (*Canny* briaunų detektorius), gaunami balti apskritimai juodame lange, kurie reiškia ragenos ribas. Apskaičiuojamas šių apskritimų centras. Toliau ieškomas atspindys nuo vyzdžio. Priimama, kad jis yra ragenos aplinkoje, todėl ieškomi aptiktų taškų kaimyniniai pikseliai. Tai realizuojama naudojant histogramos atgalinę projekciją (angl. *histogram back projection*), kuri vėliau yra slenkstinama. Šių procesų metu tik slenkstintos histogramos pikseliai projektuojami į naują vaizdą. Taip į naują vaizdą perkeliame

tamsiausi pikseliai, o ragenos aplinkoje esantys apskritimai laikomi vyzdžiais. Randami šių apskritimų centrai [4].

Išskyrus pirmąjį Purkinjė vaizdą, tarp ragenos ir vyzdžio centrų apskaičiuojamas žvilgsnio vektorius (angl. *gaze vector*). Ragenos atspindžio centras yra laikomas pradžia, o vyzdžio apskritimo centras – vektoriaus pabaiga. Jei tariama, kad ragena yra idealios sferos formos, o naudotojo galva stabili, akims judant ragenos padėtis nesikeis. Tačiau keičiantis vyzdžio centro padėčiai, keisis ir jo atstumas iki ragenos centro, taigi ir žvilgsnio vektoriaus ilgis bei kryptis [4].



2.5 pav. Žvilgsnio vektorius [4]

Prieš žvilgsnio sekiklio naudojimą, įvykdoma kalibravimo procedūra. Procedūros esmė – išskirtų žvilgsnio vektorių atitikmenų ekrane radimas. Kalibravimo metu ekrane pateikiami taškai, į kuriuos turi žiūrėti sekiklio naudotojas. Sistema paprastai pateikia penkis arba devynis taškus, išdėstytus ekrano kraštinėse (devynių taškų atveju, vienas jų bus ekrano centre). Tokiu būdu žvilgsnio vektoriai susiejami su ekrano ribiniais taškais. Remiantis šiais ribiniais taškais, keičiantis žvilgsnio vektoriaus ilgiui ir kryptčiai, sistemoje apskaičiuojamas tą vektorių atitinkantis žvilgsnio taškas, kuris išreiškiamas koordinatėmis X ir Y [1, p. 54, 4]. Tolesni veiksmai, susiję su žvilgsnio taškų klasifikacija ir apdorojimu, atliekami priklausomai nuo sekiklio naudojimo paskirties.

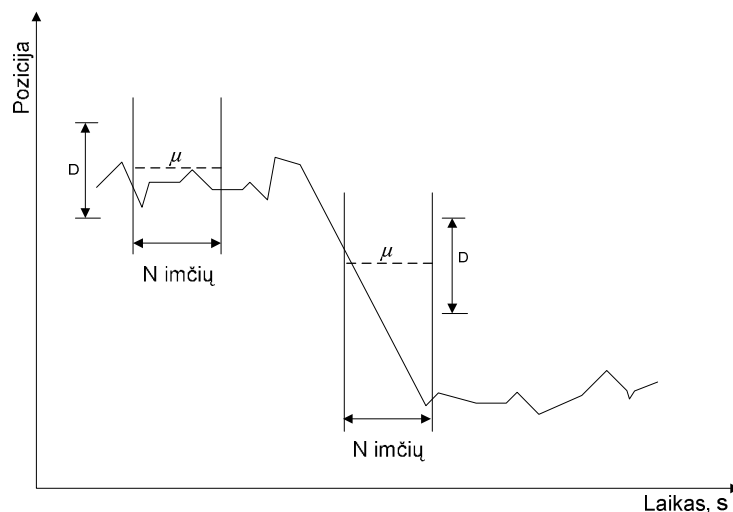
Kaip jau buvo minėta įvardiniame skyriuje, yra du pagrindiniai akies judesių tipai: fiksacijos – pasireiškia, kai dėmesys yra sukoncentruotas ties vienu objektu, ir sakados – staigūs akies obuolio pasisukimai nukreipiant dėmesį nuo vieno objekto prie kito [1, p. 42]. Galima būtų teigti, kad idealiu atveju, fiksacijų metu akis išlieka stabili, o sakadų metu juda. Tačiau realybėje taip nėra. Net ir sukoncentravus dėmesį ties dominančiu objektu, pasireiškia žvilgsnio taškų išsibarstymas, o tai reiškia, kad fiksacijų metu akis, nors ir nežymiai, juda. Tuo tarpu, sakadų metu gali būti užfiksuota viena ar kelios fiksacijos. Tai reikštų, kad nukreipiant žvilgsnį nuo vieno prie kito naudotoją dominančio objekto, akis labai trumpam laikui buvo sustojusi [5]. Jei po tam tikro sekiklio naudojimo laiko peržiūrėtume užfiksuotus ekrane taškus, pastebėtume šių taškų spiečius ties

objektais į kuriuos buvo sukoncentruotas dėmesys. Taip pat pamatytume atskirus taškus, išsibarsčiusius atokiau nuo objektų. Tuo tikslu žvilgsnio taškai, gauti iš sekiklio, turi būti papildomai apdorojami. Metodus, kuriais apdorojami žvilgsnio taškai, galima suskirstyti į dvi grupes:

1. Algoritmai, skirti apdoroti po tam tikro žvilgsnio sekiklio naudojimo laiko gautą taškų protokolą (išsklotinę);
2. Algoritmai, skirti apdoroti žvilgsnio taškus realiu laiku.

Pirmosios grupės algoritmai taikomi kai žvilgsnio sekiklis naudojamas skaitymo, patogumo naudoti (angl. *Useability*) ar panašiuose tyrimuose. Antrosios grupės algoritmai yra naudojami interaktyviose sąsajose, naudotojui žvilgsniu valdant taikomąją programą. Akivaizdu, kad šiuo atveju turi būti apdorojamas kiekvienas paeiliui iš sekiklio ateinantis žvilgsnio taškas.

Pirmosios grupės algoritmais sprendžiamas akies judesių klasifikavimo uždavinys – surinkti taškai suklasifikuojami į fiksacijas ir sakadas. Sakadų taškai neduoda jokios informacijos, todėl jie yra atmetami. Vieno iš tokių algoritmų pavyzdys yra mokslininko Anliker'io (1976) pasiūlytas metodas, kuriuo nustatoma, ar M iš N taškų yra nedidesniu nei D atstumu nutolę nuo signalo reikšmės μ [1, p. 139-141].

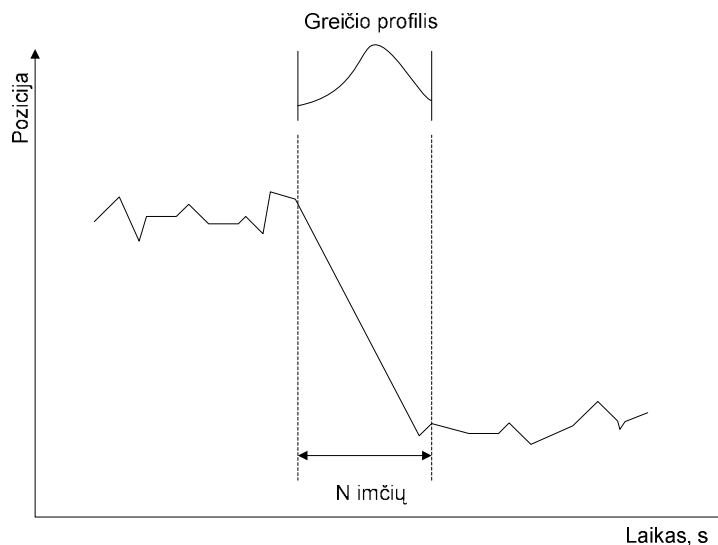


2.6 pav. Pozicijos nuokrypos metodas [1, p. 142]

Kai signalo kitimas viršija slenksčio atstumą D , reiškia, kad įvyko žymus koordinatės pasikeitimas, aptinkama sakada. M , N ir D reikšmės yra randamos empiriškai. Šis algoritmas yra vadinamas pozicijos nuokrypos algoritmu, nes yra paremtas tuo, kad fiksacija charakterizuojama kaip maža pozicijos nuokrypa (angl. *Low position variance*), o sakada – didelė pozicijos nuokrypa (angl. *High position variance*) [1, p. 141].

Kaip alternatyva šiam metodui, buvo pasiūlytas greičio aptikimo metodas. Šiuo atveju signalo greitis yra skaičiuojamas tam tikrame imties (angl. *Sample*) intervale ir lyginamas su nustatyta

atkirtos reikšmės. Jei imties signalo kitimo greitis yra mažesnis už atkirtos vertę, tariama, kad tai fiksacija, priešingu atveju signalas nurodomas kaip sakada [1, p. 141].



2.7 pav. Greičio kitimu paremtas metodas [1, p. 142]

Remiantis Yarus'o pastebėjimais, kad sakados greitis yra beveik simetrinio profilio, greičiu paremta spėjimo schema gali būti implementuota sakados pasibaigimo laikui aproksimuoti ir taip nuspėti sekančią fiksaciją. Kita fiksacijos vieta gali būti aproksimuota tuo pat, kai aptinkamas greičio kreivės (greičio profilio) maksimumas. Matuojant trunkantį laiką ir tuo laiku nukeliatą žvilgsnio taško atstumą, įvertinus sakados kryptį bei tai, kad dešinioji greičio kreivės šaka bus kairiosios veidrodinis atspindys, galima apskaičiuoti sakados pabaigos tašką [1, p. 141-144].

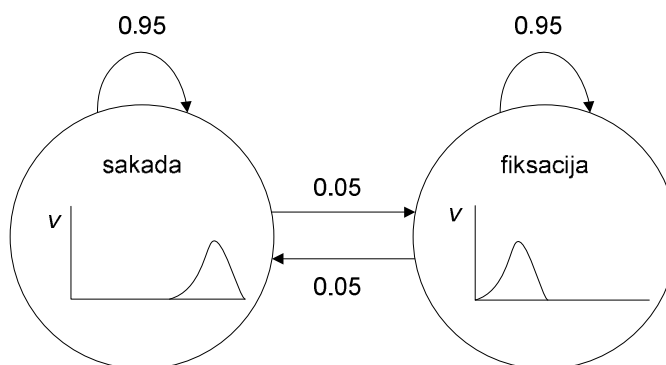
Tyrimai parodė, kad pozicijos nuokrypos ir greičiu paremti algoritmai duoda panašius rezultatus [1, p. 141].

Literatūroje galima rasti ir daugiau metodų fiksacijoms bei sakadoms identifikuoti. Smulkesnę taksonomiją pateikė mokslininkai Dario Salvuccis ir Josephas Goldbergas (2000). Autoriai išskyrė kelis pagrindinius algoritmus ir juos pagal charakteristikas suskirstė į erdvinčius (angl. *Spatial*) bei laikinius (angl. *Temporal*) [6].

Turintys erdvines charakteristikas algoritmai išskiriami trijų tipų: paremti greičiu, išsibarstymu (dispersija) ir sritimi. Pirmajam tipui priskiriamas fiksacijų identifikavimo algoritmas naudojant greičio atkirtos vertę. Algoritmas pradedamas apskaičiuojant taškas-taškas greičius kiekvienam protokolo taškui. Kiekvienas greitis yra skaičiuojamas naudojant atstumą tarp esamo ir ankstesnio taškų. Po to, naudojant greičio atkirtos vertę, kiekvienas taškas klasifikuojamas kaip sakada arba fiksacija. Jei taško greitis yra žemesnis už atkirtos reikšmę, tai šis taškas apibrėžiamas kaip fiksacija, jei didesnis – sakada. Sakados paprastai yra atmetamos, nes neteikia efektyvios informacijos apie dėmesio tašką, o fiksacijos surenkamos į fiksacijų grupes. Tai reiškia, kad esant

kelioms fiksacijoms, išsibarsčiusioms ties vienu objektu, jos surenkamos į vieną grupę, kuri nurodoma vienu žvilgsnio tašku. Jis apibrėžiamas šiais parametrais (x , y , t , d). X ir Y nusako fiksacijų grupės centrą, t – pirmojo taško laiką, d – visų taškų trukmę [6].

Fiksacijų ir sakadų identifikacijai gali būti naudojamas dviejų būsenų paslėptasis Markovo modelis (angl. *two-state Hidden Markov Model HMM*) (2.8 pav.). Būsenos atitinka greičio pasiskirstymą fiksacijų ir sakadų taškuose. Modelis susideda iš dviejų tikimybinių rinkinių: stebėjimo ir perėjimo tikimybių. Stebėjimo tikimybės kiekvienai būsenai atitinka tikimų greičių pasiskirstymą šioje būsenoje. Pirmoji būseną atitinka sakadų taškus ir apima didelius greičius. Antroji būseną atitinka fiksacijų taškus ir apima mažus greičius. Perėjimo tikimybės kiekvienai būsenai reiškia tikėtinumą, kad bus pereita iš vienos būsenos į kitą (0,05) arba pasilikta esamoje būsenoje (0,95). Tokiu būdu paslėptasis Markovo modelis iš anksto suteikia tikimybinę stebėjimo būsenų reikšmę, gautą sakadinių akies judesių metu.



2.8 pav. Paslėptąjo Markovo modelio metodas [6]

Salvucci ir Goldbergo (2000) pateiktas paslėptasis Markovo modelis kiekvienam protokolo taškui priskiria labiausiai tikėtiną identifikaciją. Toliau, kaip ir greičio atkirtos reikšmė paremtame algoritme, fiksacijos yra sugrupuojamos ir grąžinamas vienas centrinis visos grupės taškas [6].

Kitaip nei greičio atkirtos verte ar paslėptuoju Markovo modeliu paremtas algoritmas, išsibarstymu (dispersija) paremtuose algoritmuose panaudojamas reiškinys, kad fiksacijų taškai dėl savo nedidelio greičio linkę išsidėstyti arti vienas kito. Toks algoritmas identifikuoja fiksacijas kaip nuosekliai einančius taškus su tam tikru išsibarstymu ar maksimaliu galimu nuotoliu. Kadangi fiksacijos trunka bent jau 100 ms, dispersija paremtas metodas dažnai naudojamas su minimalia fiksacijos trukmės atkirtos verte, kuri gali būti 100–200 ms. Tai padeda sumažinti įrangos nepastovumą. Algoritme naudojamas slenkančio langas, kuris apima nuoseklius duomenų taškus, ieškant potencialių fiksacijų. Slinkimas prasideda nuo protokolo pradžios ir iš karto apima tam tikrą minimalų kiekį taškų (tai nustato trukmės atkirtos reikšmė ir kadravimo dažnis). Toliau yra tikrinamas lange esančių taškų išsibarstymas. Tai atliekama sumuojant skirtumus tarp taškų x ir y :

$$D = (x_{\max} - x_{\min}) + (y_{\max} - y_{\min})$$

čia D – dispersija. Jeigu dispersijos reikšmė yra didesnė nei nustatyta atkirtos riba, fiksacija neidentifikuojama ir langas paslenkamas per vieną tašką į dešinę. Jei dispersija yra žemiau atkirtos reikšmės, langas atitinka vieną fiksaciją. Tokiu atveju, langas yra plečiamas į dešinę tol kol dispersija viršiją atkirtą. Paskutinis užregistruotas langas pateikiamas kaip viena fiksacija (visų lange esančių taškų vidurkis, arba centras), su užfiksuotu pradiniu laiko momentu ir trukme. Langas juda tol kol pasiekama taškų protokolo pabaiga [6].

Susidomėjimo sritimi (angl. *Area-of-Interest*) paremtas fiksacijų identifikavimo metodas yra dar vienas algoritmas, skiriamas prie erdvinį. Nuo aptartųjų šis algoritmas skiriasi tuo, kad identifikuojamos tik tos fiksacijos, kurios yra tam tikroje taikinio vietoje. Taikiny – tai sritis, turinti tam tikrą vizualią informaciją. Šiame algoritme taip pat naudojama trukmės atkirta, kad būtų atskirtos taikinyje esančios fiksacijos, nuo pro taikinį pralekiančių sakadų. Algoritmas pradedamas susiejant duomenų taškus su taikiniais: taškai, esantys taikinyje, aprašomi kaip fiksacijos, ir taškai, esantys už taikinio ribų, aprašomi kaip sakados. Nuoseklios fiksacijos taikinyje yra sugrupuojamos, o sakados atmetamos. Fiksacijų grupės toliau yra tikrinamos panaudojant užduotą trukmės atkirtos reikšmę. Fiksacijos, kurių trukmė yra trumpesnė už atkirtą, atmetamos [6].

Šių fiksacijų ir sakadų identifikavimo algoritmų tyrimai atskleidė kiekvieno iš jų pranašumus ir trūkumus. Pastebėta, kad greičio slenksčio vertę naudojantis algoritmas yra lengvai įgyvendinamas, greitai veikia. Tačiau šis algoritmas nėra toks patikimas kaip kiti. Dėl triukšmų, kurie gali atkeliauti iš įrangos, greičio reikšmė svyruotų apie slenksčio vertę. Jei greičio vertė trumpam ją viršytų ten, kur neturėtų, rezultate būtų gautas trumpas pokytis – pyptelėjimas (angl. *Blip*), su vos keliais, nuosekliai einančiais taškais. Kad to būtų išvengta, į algoritmą įvedamas fiksacijų ir sakadų trukmių tikrinimas su nustatytomis slenkstinėmis trukmėmis [6].

Tyrimai parodė, kad patikimesnis fiksacijų identifikavimas gaunamas naudojant paslėptąjį Markovo modelį. Tikimybinis modelis patikimiau sugrupuoja fiksacijas ir triukšmų atveju jos neišskaidomos į atskiras, mažesnes grupes. Tačiau jo įgyvendinimas yra sudėtingesnis nei greičio atkirtos vertę paremto algoritmo. Aptartas taškų išsibastymu paremtas metodas taip pat pasižymi aukštu patikimumu, nors jame ir naudojamos nustatytos slenksčių vertės. Pyptelėjimų išvengiama panaudojant trukmės slenkstį. Tačiau šis algoritmas nėra toks greitas, kaip greičiu paremtas metodas. Tyrimų metu pastebėta, kad susidomėjimo srities metodai turi rimtą problemą – į taikinio fiksaciją yra įtraukiama ir daug sakadinių taškų. Ši problema rezultate duoda apgaulingai ilgas fiksacijų trukmes [6, 7].

Analizuojant žvilgsnio taškų protokolą, taškų apdorojimas vykdomas siekiant sugrupuoti fiksacijų taškus ir išskirti „karštašias“ žvilgsnio vietas (angl. *Hot spots*). Tuo tarpu nuosekliai iš

įrenginio ateinančių taškų apdorojimu siekiama šiuos žvilgsnio taškus realiu laiku priskirti vienam ar kitam objektui. Literatūros šaltiniuose teigiama, kad tiek paslėptuoju Markovo modeliu, tiek dispersija paremti metodai yra pakankami greiti, kad galėtų veikti realiu laiku [6], tačiau tyrimų, kuriuose būtų panaudoti šie metodai, aptikti nepavyko.

Kai siekiama pritaikyti žvilgsnio tašką kaip žymeklį (angl. *Cursor*), kuriuo naudotojas galėtų rinktis taikomosios programos objektus, susiduriama ne tik su žvilgsnio nestabilumu, tačiau objektų aktyvavimo (mygtukų paspaudimo) problema, kuri dar vadinama „Midas“ sąlyčiu (angl. *Midas touch*) problema. [1, p. 276]. Akis negali būti naudojama tiesiogiai kaip kompiuterio pelė, nes žvilgsniu negalima realizuoti spragtelėjimo. Ši problema sprendžiama dviem būdais: kombinuojant sąsają su tam tikru mechanizmu, kuris galėtų „aktyvuoti“ ir „deaktyvuoti“ objektą, arba įvedant išlaikymo laiką (angl. *Dwell time*), kuriam susikaupus iki tam tikros reikšmės, objektas, į kurį sutelktas dėmesys, būtų aktyvuotas. Tačiau kai sąsaja paremta vien žvilgsniu, išlaikymo laiko sprendimas yra vienintelis būdas aktyvuoti taikomosios programos mygtukus [4, 13, 19, 21]. Bendru atveju laiko skaičiavimas pradedamas taškui pataikius ant vieno ar kito objekto, ir baigiamas pasiekus nustatytą vertę, po kurios įvykdomas paspaudimas. Jei laiko skaičiavimo metu dėmesio taškas paliko objektą, laikas vėl nustatomas į 0. Dėl žvilgsnio taškų išsibarstymo mygtukas gali būti ir visai neaktyvuojamas.

Šiai problemai išspręsti yra pasiūlyti realiu laiku veikiantys algoritmai. Išanalizavus literatūros šaltinius, galima išskirti dvi šių algoritmų kategorijas. Vienai iš jų priklauso algoritmai, kuriais perskaičiuojamas išlaikymo laikas. Kaip pavyzdį galima pateikti Špakovo (2011) pasiūlytus varžymosi (angl. *Competing*) ir dinamiško varžymosi (angl. *Dynamic competing*) algoritmus [10]. Šių algoritmų esmė ta, kad vienu metu gali būti skaičiuojami kelių objektų išlaikymo laikai. Jeigu vienam iš taikomosios programos objektų buvo pradėtas skaičiuoti išlaikymo laikas, ir sekantis iš sekiklio atėjęs taškas į šį objektą nepateko, tai jo išlaikymo laikas ne iš karto nustatomas į pradinę padėtį, o nuosekliai mažinamas. Tuo metu, kai vieno objekto išlaikymo laikas mažėja, gali būti pradėtas skaičiuoti kito objekto išlaikymo laikas. Nuspaudžiamas tas objektas, kurio laiko vertė pirmoji pasiekė nustatytąją [10].

Kitai kategorijai galima priskirti algoritmus, kuriais perkciačiuojamos taško koordinatės. Šiuo atveju, taškui nepataikius į objektą, išlaikymo laikas bus nustatomas į pradinę padėtį. Kad nustatymo būtų išvengta, išsibarsčiusius žvilgsnio taškus siekiama pritraukti prie artimiausio objekto. Tokių pritraukimo savybių turinčių algoritmų išsami analizė pateikiama šio magistro darbo trečiajame skyriuje.

3. PRITRAUKIMO ALGORITMŲ ANALIZĖ

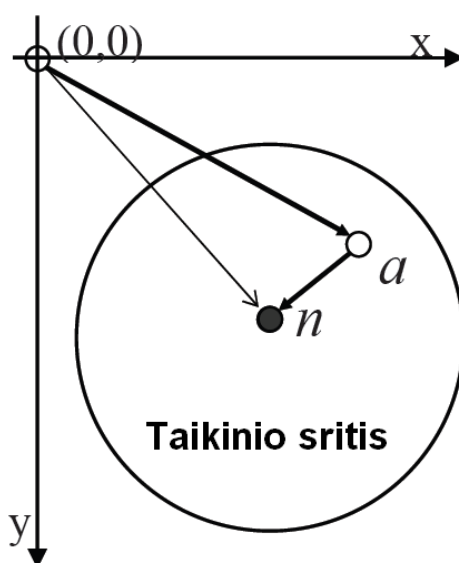
Šis magistro darbo skyriuje pateikiama detali pritraukimo, traukos lauko ir greičio mažinimo algoritmų analizė. Ji apima formulių išaiškinimą, bei traukos lauko ir greičio mažinimo algoritmų simuliaciją *Matlab* programiniu paketu. Po analizės, skyriaus pabaigoje pateikiamas naujas siūlomas traukimo savybėmis pasižymintis metodas – traukimo pusiaukraštinės kryptimi algoritmas. Taip pat pateikiami jo veikimo simuliacijos rezultatai.

3.1. Pritraukimo algoritmas

Pritraukimo (angl. *Sticking*) arba priskyrimo centrui (angl. *Warping to target center*) algoritmo esmė – pritraukti iš sekiklio atėjusį tašką prie objekto centro [9]. Šis algoritmas yra vienas paprasčiausių ir jokia formule neišreiškiamas, tačiau norint pritaikyti jį praktikoje, būtina nustatyti taško padėtį objekto, į kurį sutelktas dėmesys atžvilgiu. Tyrimai parodė, kad šis algoritmas nėra toks efektyvus kaip greičio mažinimo ar traukos lauko algoritmai [9].

Igyvendinat šį metodą susiduriama su tam tikromis neapibrėžtimis. Literatūros šaltiniuose pateikiama sutapdinimo su taikinio centru algoritmo sampratai pavaizduota 3.1 pav. Kyla klausimas, koks yra skirtumas tarp sutapdinimo su taikinio centru ir taikinio ribomis? Jei algoritmas gali būti taikomas ir nuo ribų nutolusiems taškams, nėra aišku, koku atstumu gali šis taškas nuo objekto būti nutolęs. Kitaip sakant, nėra žinoma riba, į kurią būtų atsižvelgiama priskiriant tašką vienam ar kitam objektui.

Lieka spėlioti, ar dėl šios neapibrėžties, ar dėl kitų priežasčių, algoritmo nauda nebuvo atskleista [9]. Priskyrimo centrui algoritmo veikimo principas pavaizduotas 3.1 pav.

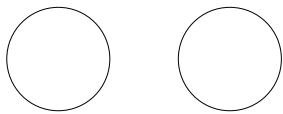
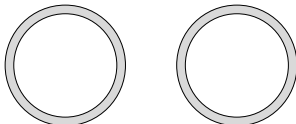
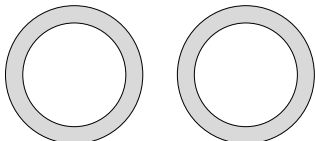
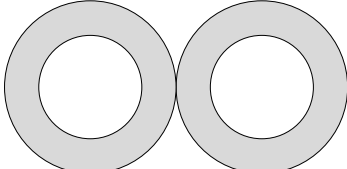


3.1 pav. Pritraukimo algoritmo veikimas [9]

Priskyrimo neapibrėžtį galima išspręsti nustatant tam tikrą atstumą tarp taško ir objekto centro, kuris turi būtų ne didesnis norint taikyti priskyrimą. Šį atstumą viršijus, priskyrimas nevykdomas. Tokiu būdu aplink objektą sudaroma tam tikra aktyvioji zona, tačiau vizualus objekto dydis nepakinta.

Darbo metu visus aprašomus algoritmus siekiama įgyvendinti virtualioje klaviatūroje. Klaviatūros išklotinėje visi mygtukai (objektai) ir juos skiriantys tarpai turi būti vienodo dydžio. Priskyrimo centrui algoritmo atveju aktyvioji zona kiekvienam mygtukui taip pat turi būti vienoda. Darbo metu buvo išskirti ir procentiškai išreikšti keturi aktyviosios zonos pločio atvejai. Jie pateikiami 1 lentelėje.

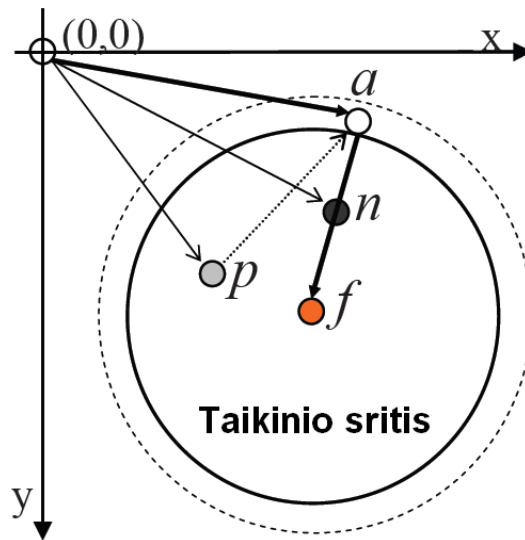
1 lentelė. Pritraukimo algoritmo konfigūracijos

0% pritraukimas – priskyrimo nėra;	
33% pritraukimas – priskyrimas vykdomas nuo $1/6 \times gap$ atstumo;	
66% pritraukimas – priskyrimas vykdomas nuo $1/3 \times gap$ atstumo;	
100% pritraukimas – priskyrimas vykdomas nuo $gap/2$ atstumo;	

Čia *gap* – tarpo tarp mygtukų ilgis. Detalesnis šio algoritmo veikimo aprašymas pateikiamas 4 ir 5 skyriuose.

3.2. Traukos lauko algoritmas

Ahlström'as (2005) pateikė traukos lauko (angl. *force field*) idėją pakopiniuose krentančiuose meniu, kad būtų pagerintas pelės žymeklio nuvedimas nuo šakninio į dukterinius meniu elementus. Užvedus kompiuterio pele ant šakninio elemento rodyklės, žymeklis nustumiamas į pirmąjį pošakninį elementą [8]. Šis metodas buvo pritaikytas ir žvilgsnio taškų apdorojimui [9, 10]. Traukos taškas laikomas objekto centru. Tokiu atveju įcentrinis žymeklio judėjimas sustiprinamas, o išcentrinis susilpninamas. Tokiu būdu žymeklį siekiama įtraukti į taikinio vidų [9].



3.2 pav. Traukos lauko algoritmo veikimas [9]

Čia apskritimas, kurio centras taške f , vaizduoja mygtuko plotą. Kadangi, šiuo atveju, mygtukai pateikiami kompiuterio ekrane, viršutinis kairys kampas atitinka koordinatinių pradžių. X ir Y koordinatinių didėjimo kryptis yra teigiamos. Raudonai pateikti taškai p ir n yra perskaičiuoti, kitaip tariant, gauti po algoritmo pritaikymo. Taškas p , su koordinatėmis p_x ir p_y , yra paskutinis prieš tai buvęs, o taškas n , su koordinatėmis n_x ir n_y , – naujas perskaičiuotas taškas. Mėlynai pavaizduotas taškas a , su koordinatėmis a_x ir a_y , yra iš žvilgsnio sekiklio atėjęs taškas, kuris ir bus koreguojamas.

Vektoriai $\vec{f}$, $\vec{p}$, $\vec{a}$ ir $\vec{n}$, yra formulėje figūruojantys vektoriai [8, 9].

$$\vec{n} = \vec{a} + s \cdot \frac{\vec{f} - \vec{a}}{\|\vec{f} - \vec{a}\|}$$

čia s – pritraukimo koeficientas, kuris gali būti $0 < s < 1$, nusako traukos lauko stiprį, $\vec{n}$ – naujo (perskaičiuoto) taško vektorius, $\vec{a}$ – iš sekiklio atėjusio taško vektorius, $\vec{p}$ – prieš tai buvusio taško vektorius, $\vec{f}$ – mygtuko centro taško vektorius. Šiuo atveju vektorių reiškmės yra ne kas kita, kaip jų galinių taškų (n , a , p ir f) X ir Y koordinatės. $\frac{\vec{f} - \vec{a}}{\|\vec{f} - \vec{a}\|}$ – dydis, nusakantis traukimo kryptį – į

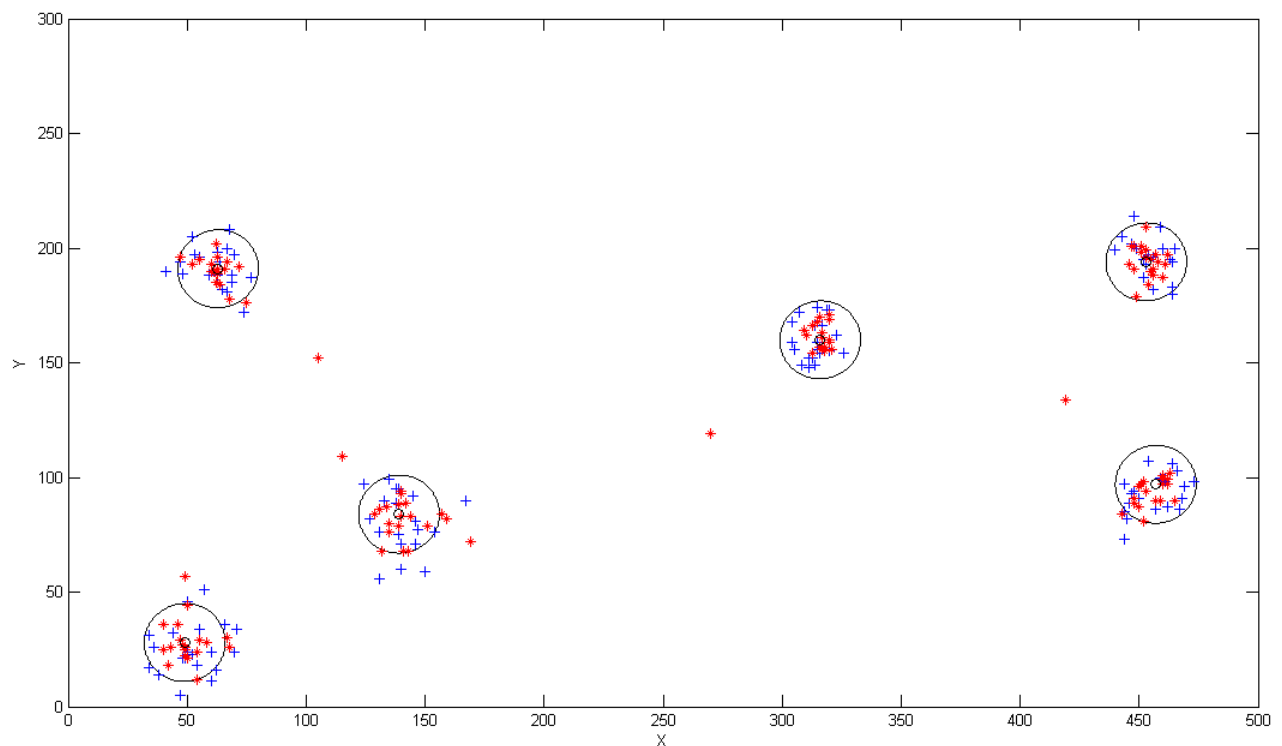
kairę/dešinę. Šis santykis gali būti 1 arba -1. Jei $\vec{f} > \vec{a}$, tuomet šis santykis bus 1, jei $\vec{f} < \vec{a}$,

tuomet jis bus -1. Euklido atstumas tarp vektorių $\left\| \vec{a} - \vec{p} \right\|$ yra, iš esmės, ilgis tarp taškų a ir p ir apskaičiuojamas formule

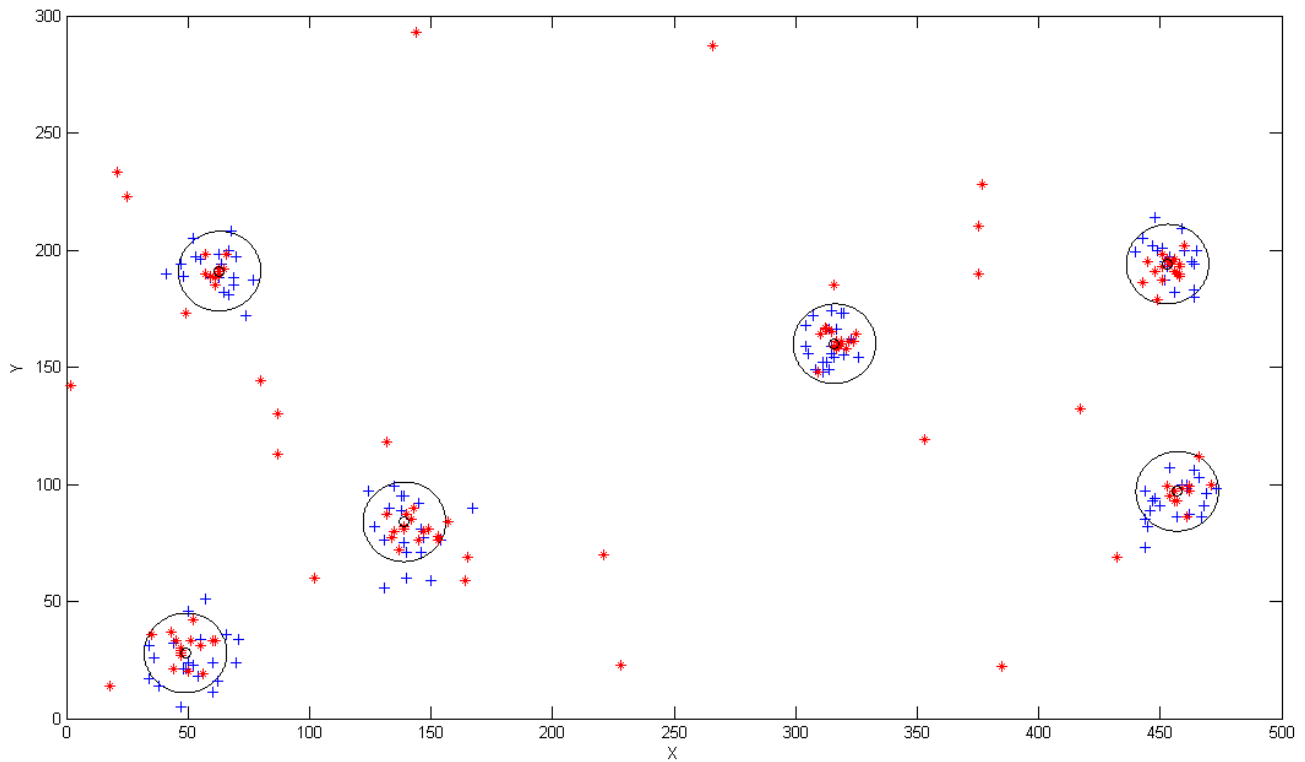
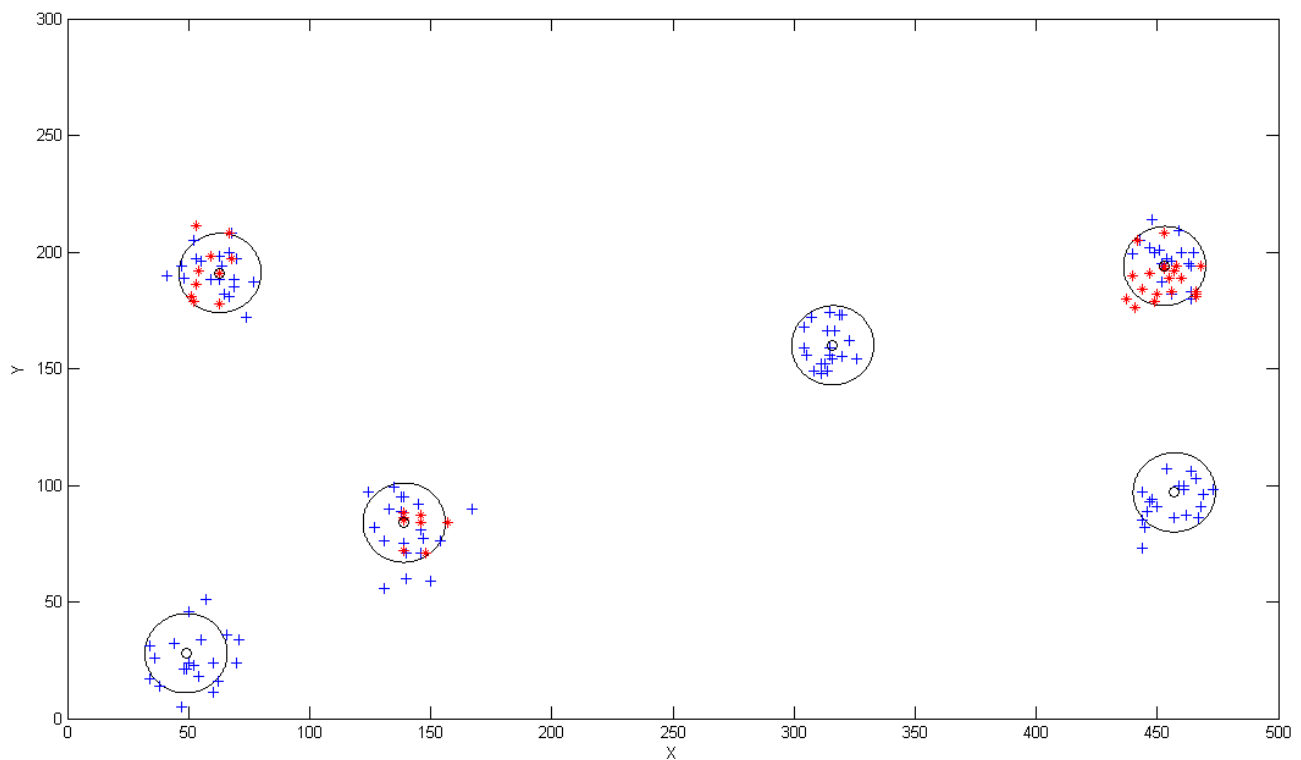
$$\left\| \vec{a} - \vec{p} \right\| = \sqrt{|a_x - p_x|^2 + |a_y - p_y|^2}$$

Kaip matyti iš formulės, naujo taško koordinatės apskaičiuojamos kaip dviejų dydžių suma – atėjusio taško koordinatė ir dydžio, nusakančio jų korekciją (traukimo atstumą). Šios korekcijos vertė priklauso nuo pritraukimo koeficiento s vertės, atėjusio taško a padėties mygtuko centro f atžvilgiu bei Euklido atstumo tarp vektorių $\vec{a}$ ir $\vec{p}$.

Matlab programiniu paketu buvo atliktas šio algoritmo veikimo simuliacija. Bandymo metu atsitiktine tvarka buvo sugeneruojami šeši taškų spiečiai, tariant, kad jie atitinka šešis objektus ekrane, į kuriuos laikui bėgant buvo sukoncentruotas žmogaus žvilgsnis. Šie objektai ir jų centrai apibraikti juodais apskritimais. Mėlynais „+“ ženklais yra pažymėti iš sekiklio atėję taškai, kurių koordinatės nebuvo perskaičiuotos. Raudonais „*“ ženklais pažymėti taškai su pagal traukos lauko algoritmą perskaičiuotomis koordinatėmis. Modeliavimo metu buvo keičiamas pritraukimo koeficientas ir stebėtas taškų išsidėstymas pritaikius algoritmą.



3.3 pav. Pritraukimo koeficientas $s = 0,3$


 3.4 pav. Pritraukimo koeficientas $s = 0,6$

 3.5 pav. Pritraukimo koeficientas $s = 0,9$

Iš paveikslėlių galima pamatyti, kad didinant pritraukimo koeficientą, perskaičiuoti taškai koncentruojami arčiau centro. Tačiau taip pat galima pastebėti, kad didėjant s , vis daugiau taškų išsibarsto plote tarp objektų.

Kol s nedidelis, dydis formulėje, nusakantis koordinatės korekciją, taip pat yra nedidelis, todėl perskaičiuoto taško koordinatės nedaug skiriasi nuo taško, atėjusio iš sekiklio, koordinatės.

Akivaizdu, kad didėjant s , koordinatės korekcija taip pat didėja, todėl perskaičiuoti taškai labiau nutolsta nuo sekiklio taškų ir artėja prie objekto centro. Kol žvilgsnis sukoncentruotas į vieną objektą, atstumas tarp speičiaus taškų yra nedidelis, ir perskaičiuoti taškai koncentruojasi arčiau objekto centro taško. Kai žvilgsnis nukreipiamas į kitą objektą, atstumas tarp naujo ir prieš tai buvusio taško, todėl ir Euklido atstumas, gaunamas didelis. Dėl šios priežasties išauga koordinatės korekcijos reikšmė ir perskaičiuotas taškas „pralekia“ dominantį objektą. Atstumai tarp sekančių taškų, sudarančių spiečių, yra maži, todėl kiekvienas naujas perskaičiuojamas taškas palaipsniui grįžta į objektą ir, kol dėmesys išlaikomas, koncentruojasi aplink to objekto centrą.

Kai pritraukimo koeficientas yra didelis (artimas vienetui), koordinatės korekcija dar labiau padidėja. Šiuo atveju, kai dėmesys nukreiptas į vieną iš objektų ir atstumas tarp speičiaus taškų yra nedidelis, visų perskaičiuotų taškų koordinatės praktiškai sutampa su objekto centro taško koordinatėmis. Tačiau nukreipiant dėmesį į sekantį objektą, dėl didelės koordinatės korekcijos reikšmės daugiau speičiaus taškų jį „pralekia“. Dėl to padidėja perskaičiuojamų taškų išsibarstymas plote tarp objektų.

3.3. Greičio mažinimo algoritmas

Kaip jau buvo minėta, idealiu atveju fiksacijos metu žvilgsnio taškų padėtis turi nekisti, tačiau realiu atveju taip nėra. Vienas iš programinių būdų šiam kitimui sumažinti yra greičio mažinimo algoritmas.

Sekiklyje įmontuota kamera dirba tam tikru dažniu. Todėl ir iš sekiklio koordinatės gaunamos tik kas tam tikrą laiko intervalą. Per šį laiko tarpą pakinta žvilgsnio taško padėtis. Darbo dažnis, todėl ir laikas, yra pastovus, o koordinatės pokytis kiekvienu atveju skirtingas, todėl skirtingas ir greitis, kurį taškas nukeliavo per vieną laiko intervalą. Kai pasireiškia fiksacija, gaunamų žvilgsnio taškų kitimo greitis yra mažas, tačiau daugeliu atvejų nepakankamai, kad taškai nekirstų objekto ribų (pastaba: priklauso nuo objekto dydžio). Todėl stabilumui padidinti ir naudojamas greičio mažinimo metodas [9, 10].

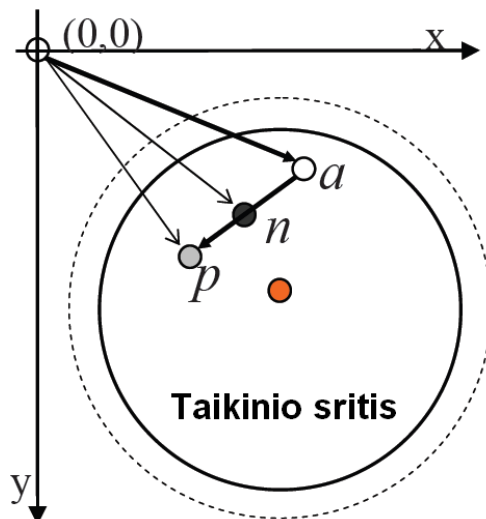
Algoritmas yra išreiškiamas formule

$$\vec{n} = \vec{a} \cdot (1 - r) + r \cdot \vec{p}$$

čia $\vec{n}$ – naujo (perskaičiuoto) taško vektorius, $\vec{a}$ – iš sekiklio atėjusio taško vektorius, $\vec{p}$ – prieš tai buvusio taško vektorius, r – greičio mažinimo koeficientas, kuris gali būti $0 < r < 1$, nusakantis greičio mažinimo stiprį.

Iš formulės matyti, kad naujo (perskaičiuojamo) taško koordinatės randamos kaip atėjusio taško koordinatė ir prieš tai buvusio taško koordinatė, padaugintų iš atitinkamų svorių, suma.

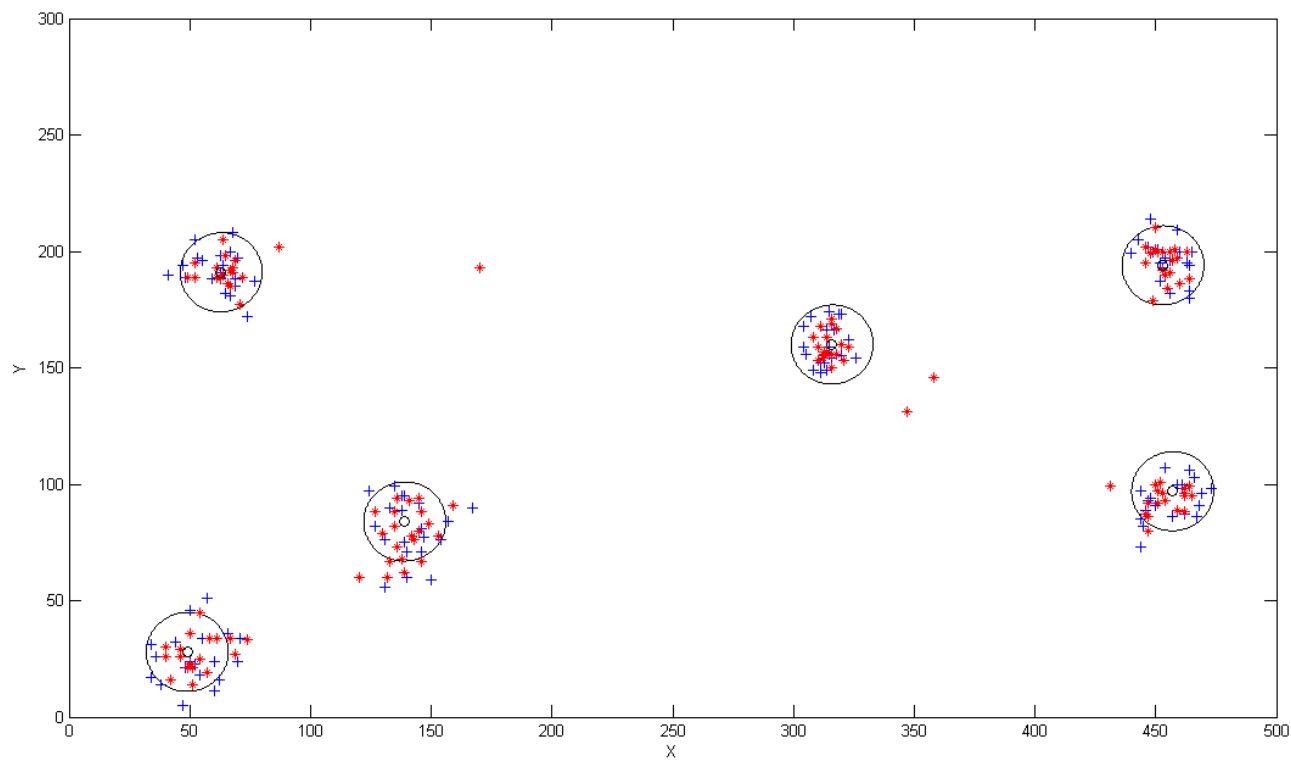
Grafiškai šio algoritmo veikimo principas yra pavaizduotas 3.6 pav.



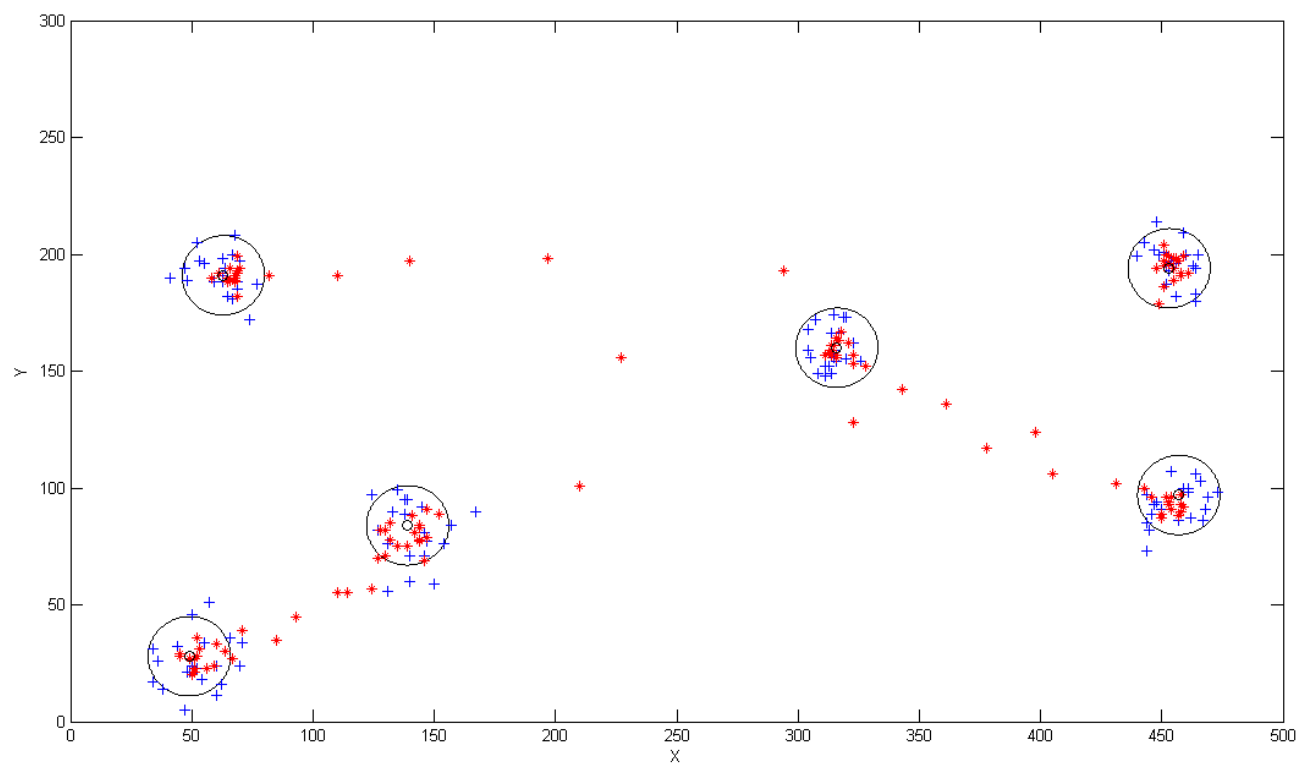
3.6 pav. Greičio mažinimo algoritmo veikimas [9]

Tyrimai parodė, kad greičio mažinimo metodas ryškiai sumažina tikimybę, kad dėl fliktuacinių akies judesių žymeklis paliks taikinio sritį, taip pertraukdamas nusistovėjimo laiką [9, 10].

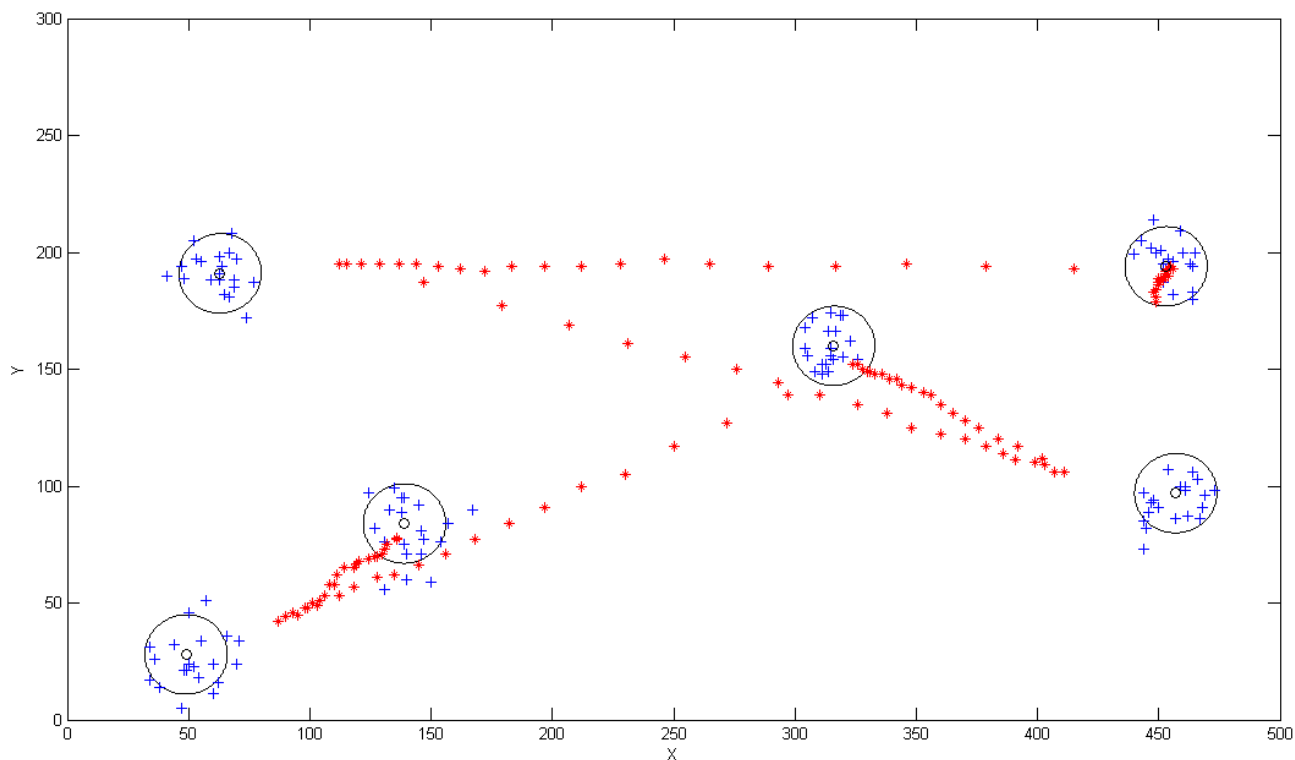
Greičio mažinimo algoritmo veikimas taip pat buvo simuliuojamas *Matlab* programiniu paketu, analogiškais traukos lauko algoritmo veikimo simuliacijos sąlygomis. Simuliacijos metu buvo keičiamas greičio mažinimo koeficientas, ir fiksuojamos perskaičiuotų taškų padėties.



3.7 pav. Greižio mažinimo koeficientas $r = 0,3$



3.8 pav. Greižio mažinimo koeficientas $r = 0,6$



3.9 pav. Greičio mažinimo koeficientas $r = 0,9$

Iš 3.7 pav. galima matyti, kad esant nedideliam greičio mažinimo koeficientui, perskaičiuotų taškų koordinatės praktiškai nesiskiria nuo iš sekiklio atėjusių taškų koordinatė. Didinant koeficientą, perskaičiuoti taškų spiečius gaunamas labiau koncentruotas, tačiau taip pat galima pastebėti šių taškų išsibarstymą plote tarp objektų. Kaip ir traukos lauko algoritmo atveju, išsibarstymas pasireiškia nukreipiant dėmesį nuo vieno objekto prie kito. Akivaizdu, kad kuo didesnis greičio mažinimo koeficientas, tuo labiau taškas traukiamas link prieš tai buvusio taško. Nukreipus žvilgsnį į kitą objektą, atstumas, tarp atėjusio iš sekiklio ir prieš tai buvusio taškų, yra didelis, todėl pirmieji perskaičiuoti taškai dar nėra pasiekę dominančio objekto. Koeficientą dar padidinus, taškų kitimo greitis dar labiau sumažėja ir žvilgsnio kreipimo metu, šie taškai išdėstomi tarpe tarp vieno objekto ir kito (3.9 pav.).

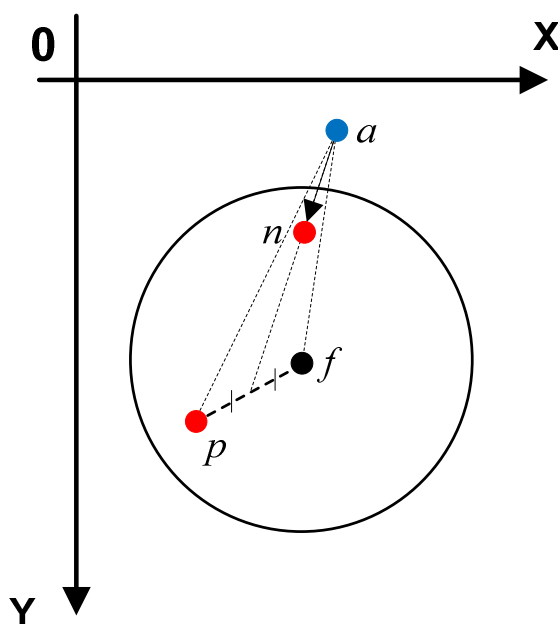
Greičio mažinimo algoritme yra neįvertinama objekto padėtis ekrane, formulėje neregistruoja objekto centro taškas ar kiti objekto padėtį nusakantys parametrai. Algoritmas veikia visoje aktyviojoje srityje, kur fiksuojami žvilgsnio taškai.

Taškų išsibarstymas nukreipiant žvilgsnį nuo vieno objekto prie kito pasireiškia dėl to, kad greičio mažinimo ir traukos lauko algoritmuose yra įvertinama prieš tai buvusio taško padėtis. Išsibarstymo galima išvengti, jei naujas taškas būtų perskaičiuojamas atsižvelgiant tik į dominančio objekto padėtį ekrane, t.y. įvertinant objekto centro koordinates, nustatant jo dydį (pvz., skersmenį), ar atsižvelgiant į kitus objektą ir jo padėtį ekrane nusakančius parametrus. Tokiu atveju, nukreipus žvilgsnį nuo vieno objekto prie kito, iš sekiklio atėję taškai bus vienareikšmiškai perskaičiuojami to objekto atžvilgiu.

3.4. Traukimas pusiauakraštinės kryptimi

Teoriškai išanalizavus visų trijų algoritmų veikimą galima išvelgti kiekvieno iš jų privalumus ir trūkumus. Greičio mažinimo algoritmas yra labiausiai dinamiškas ir visiškai nesusietas su objektų padėtimi ekrane. Traukos lauko algoritmas nors ir susiejamas su objekto centro koordinatėmis, vis dėlto yra pakankamai dinamiškas, nes taip pat įvertina koordinatės kitimo greitį – formulėje figūruoja Euklido atstumas (tai, praktiškai, nusako koordinatės kitimo greitį, nes laiko intervalas, kaip jau minėta, yra pastovus). Tačiau tiek greičio mažinimo, tiek traukos lauko algoritmų atveju pasireiškia taškų išsibarstymas plote tarp objektų. Esant didesniems r ir s koeficientams, sugaištama daugiau laiko kol taškai nusistovi dominančio objekto ribose. Tai nepasireiškia priskyrimo centro algoritmo atveju, kai taškai perskaičiuojami (arba, šiuo atveju, tiesiog priskyriami) atsižvelgiant tik į objekto padėtį nusakančius parametrus – jo centro koordinates ar skersmenį. Tačiau, nustačius atstumą, nuo kurio bus vykdomas priskyrimas, šis algoritmas pasirodo gana griežtai apibrėžtas ir statiškas. Įvertinus kiekvieno iš išvardintų algoritmų silpnąsias ir stipriąsias vietas, darbe pateikiamas dar vienas pritraukimo algoritmų grupės variantas – traukimas pusiauakraštinės kryptimi.

Sujungę turimus p , a ir f taškus, gauname trikampį (3.10 pav.).



3.10 pav. Traukimo pusiauakraštinės kryptimi algoritmo veikimo principas

Iš taško išveskime pusiauakraštinę, kuri padalintų atstumą tarp taškų p ir f . Taškas, dalinantis šį atstumą pusiau, yra išreiškiamas kaip $\frac{\vec{f} + \vec{p}}{2}$ (toliau – pusiauakraštinės taškas). Akivaizdu, kad šis taškas visada bus dvigubai arčiau objekto centro nei prieš tai buvęs taškas p . Tuomet, radę pokytį

tarp iš sekiklio atėjusio taško a ir pusiau kraštinės taško, padauginame jį iš koeficiento, nusakančio pritraukimo stiprį, ir gausime tokia traukimo pusiau kraštinės kryptimi algoritmą išreiškančią formulę

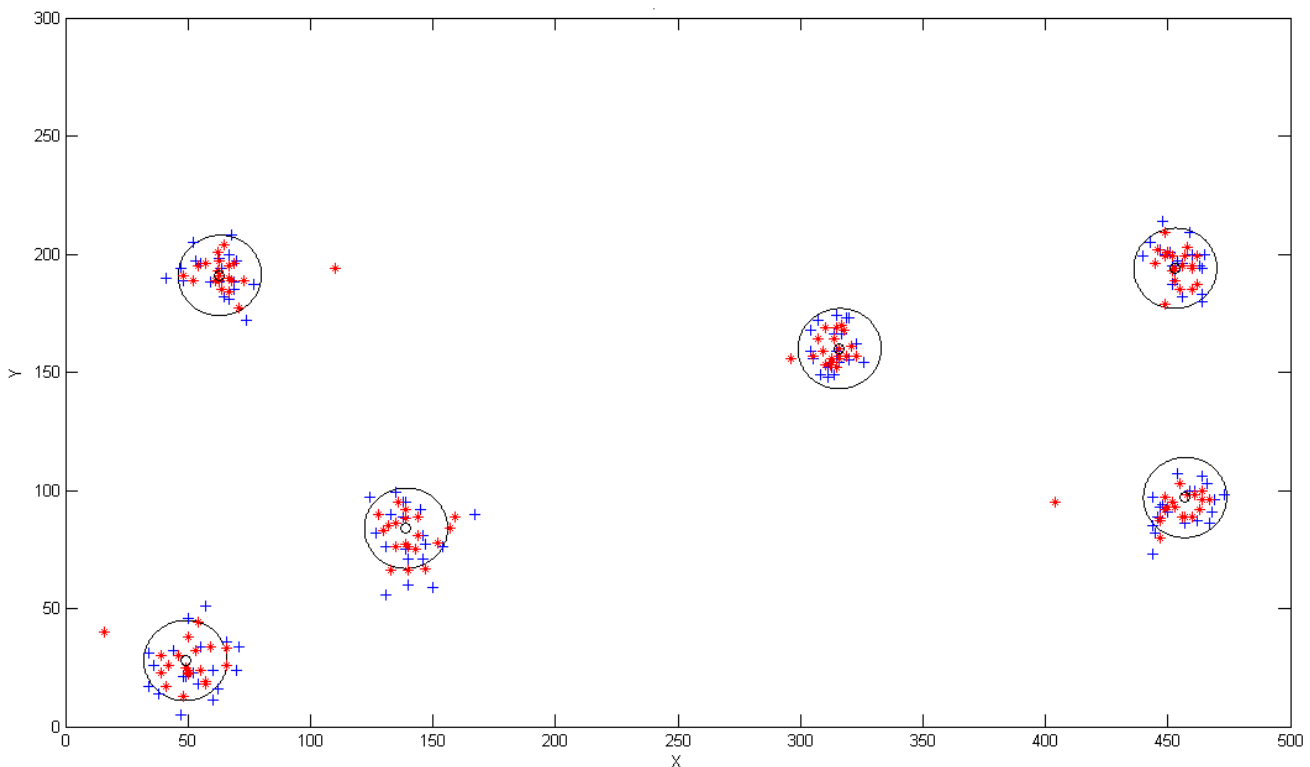
$$\vec{n} = \vec{a} + s \cdot \left| \vec{a} - \frac{\vec{f} + \vec{p}}{2} \right| \cdot \frac{\vec{f} - \vec{a}}{\left| \vec{f} - \vec{a} \right|}$$

čia dydis $\frac{\vec{f} - \vec{a}}{\left| \vec{f} - \vec{a} \right|}$ yra vis dar reikalingas, nes būtina įvertinti perskaičiuojamo taško padėtį objekto

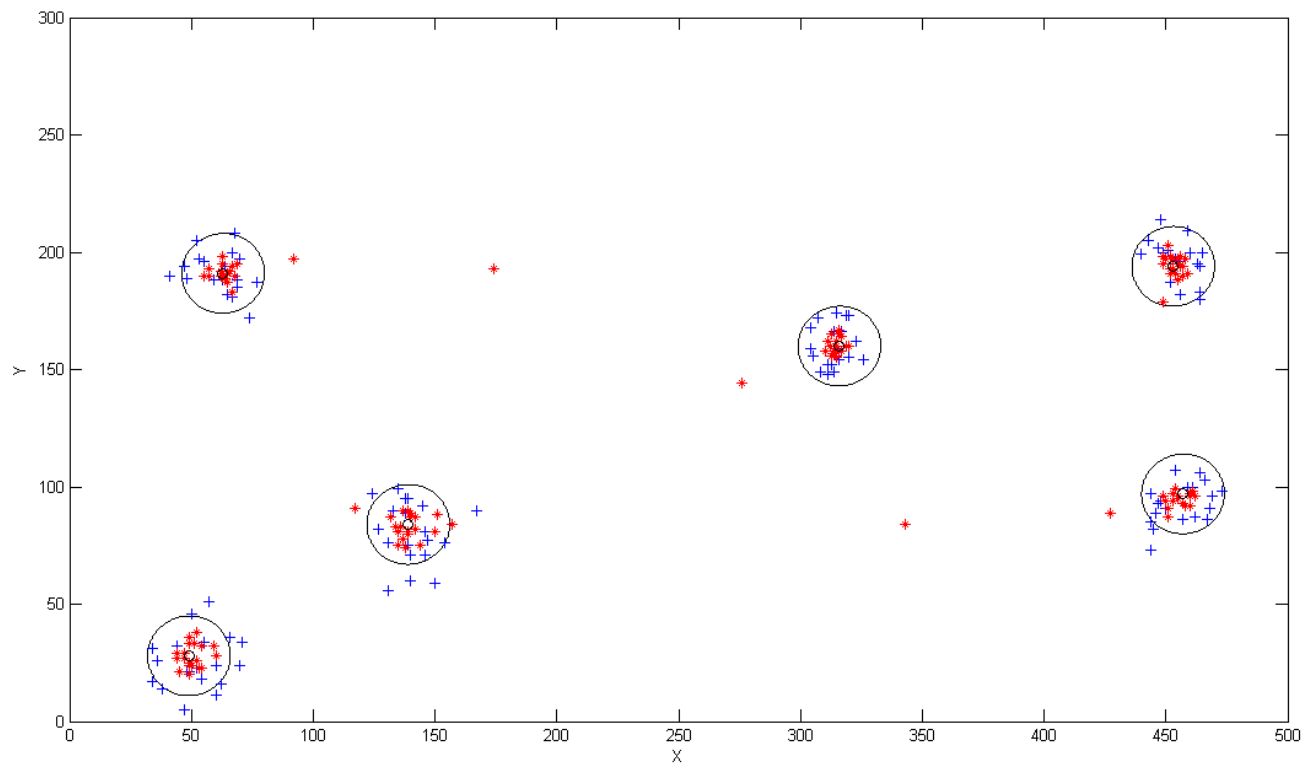
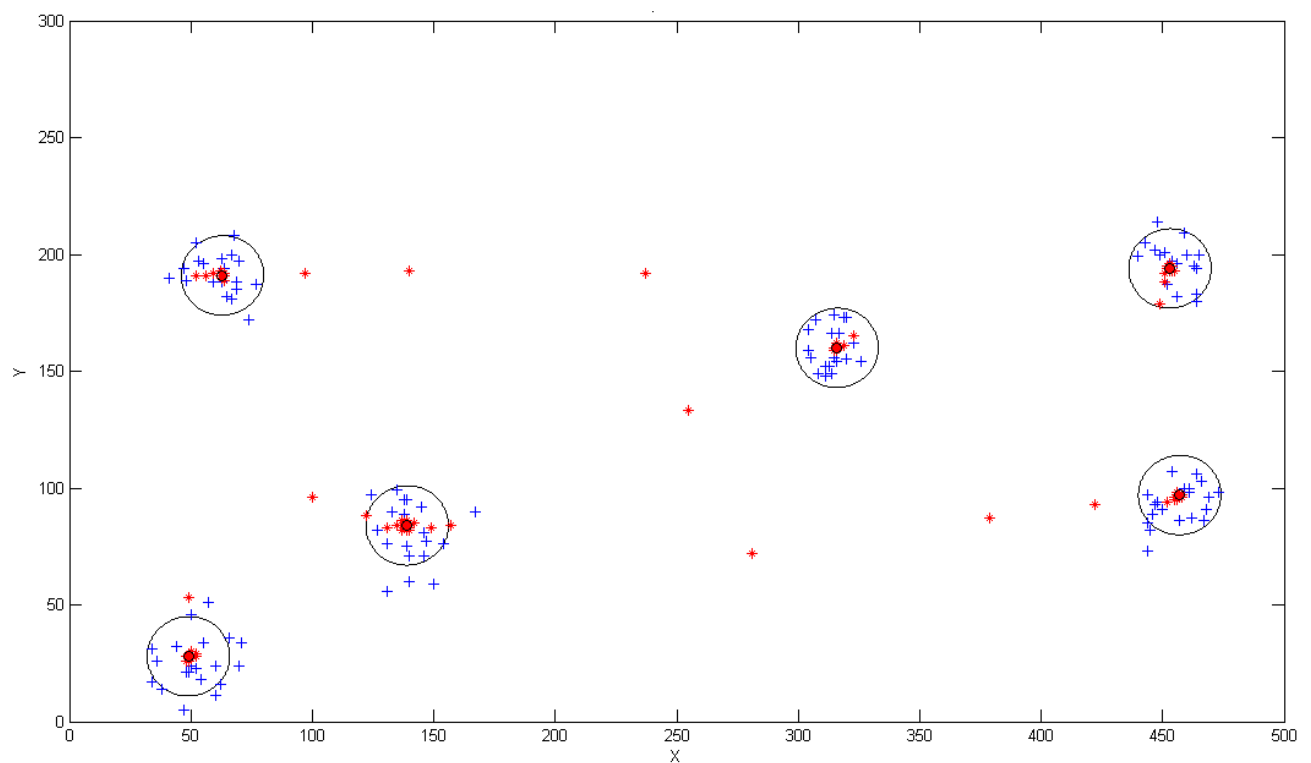
centro atžvilgiu. $\left| \vec{a} - \frac{\vec{f} + \vec{p}}{2} \right|$ – atstumas tarp taško a ir pusiau kraštinės taško. Skirtingai nei traukos

lauko įgyvendinimo metu, perskaičiuojant kiekvieną koordinatę atskirai imamoms atitinkamai X ir Y dedamosios. Tuo tarpu įgyvendinant traukos lauko algoritmą, perskaičiuojant koordinates būtina naudoti vieną ir tą patį Euklido atstumą abiejų naujo taško dedamųjų skaičiavimo metu. Priešingu atveju, pasikeis traukos lauko algoritmo veikimo principas.

Pateiktas algoritmo veikimas taip pat buvo simuliuojamas *Matlab* programiniu paketu, analogiškėmis sąlygomis greičio mažinimo ir trauko lauko algoritmų veikimo simuliacijai.



3.11 pav. Traukimo stipris $s = 0,3$

3.12 pav. Traukimo stipris $s = 0,6$ 3.13 pav. Traukimo stipris $s = 0,9$

Iš 3.11 pav. galima pastebėti, kad kol traukimo koeficientas yra nedidelis, perskaičiuotų taškų koordinatės nedaug skiriasi nuo iš sekiklio atėjusių taškų koordinatė. Didinant traukimo koeficientą perskaičiuoti taškai labiau koncentruojasi objekto centro srityje. Pasireiškia

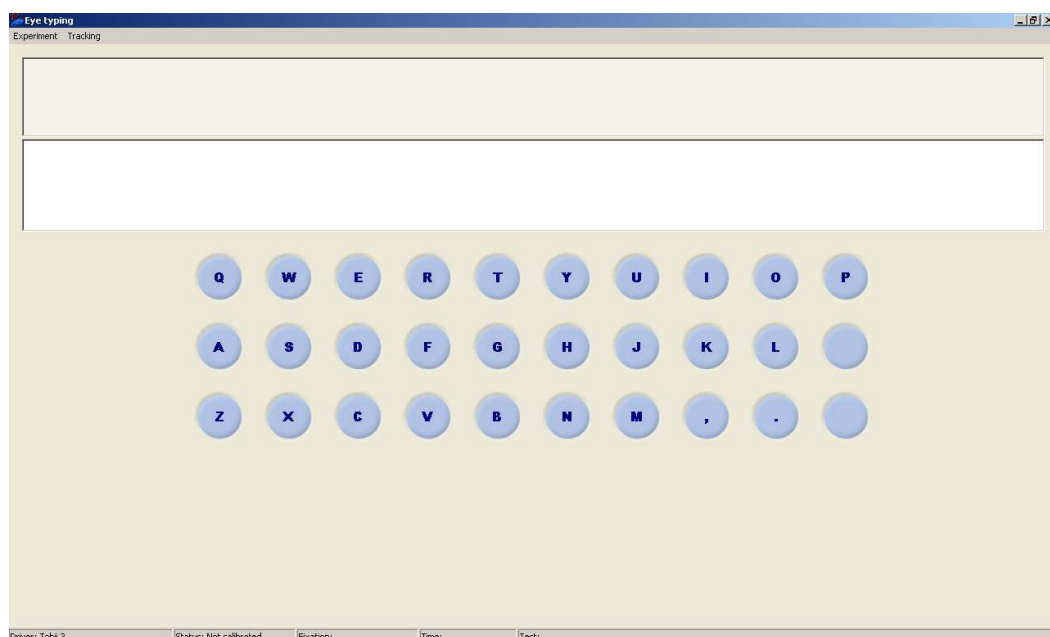
išsibarstymas, tačiau palyginus su greičio mažinimo ar traukos lauko algoritmais, jis yra nedidelis. Išsibarstymą mažina pusiauakraštinės taško panaudojimas. Nukreipiant žvilgsnį nuo vieno objekto prie kito, atstumas tarp a ir p taškų yra didelis, tačiau panaudojant pusiauakraštinės tašką, šis atstumas sumažėja dvigubai. Dėl šios priežasties pirmieji perskaičiuoti taškai greičiau pasiekia objektą ir jame nusistovi. Kai traukimo koeficientas yra artimas vienetui, perskaičiuotų taškų koordinatės beveik sutampa su objektų centrų koordinatėmis, o išsibarstymas išlieka ypač mažas (3.13 pav.).

Traukimo pusiauakraštinės kryptimi algoritme yra išlaikytas greičio mažinimo ir traukos lauko metodų dinamiškumas, bet kartu šis algoritmas labiau nei pastarieji įvertina objekto padėtį ekrane. Teoriniu požiūriu, traukimo pusiauakraštinės kryptimi metodas turėtų būti pranašesnis už greičio mažinimo, traukos lauko ir priskyrimo centrui algoritmus.

Visi skyriuje analizuoti pritraukimo algoritmai buvo įgyvendinti taikomojoje programoje *EyeTyping* ir praktiškai išbandyti *Tobii T120* žvilgsnio sekikliu. Algoritmų veikimų simuliacijos *Matlab* programiniu paketu skriptas yra pateiktas pirmajame šio darbo priede.

4. ALGORITMO ĮGYVENDINIMAS TAIKOMOJOJE PROGRAMOJE *EYETYPING*

*EyeTyping*¹ taikomosios programos (4.1 pav.) kūrėjai yra Temerės universiteto Regimosios Sąveikos Tyrimų Grupės (angl. *Visual Interaction Research Group VIRG*) nariai. Savo veikla ši grupė pelno nesiekia. *VIRG* yra tarptautinės asociacijos *COGAIN* bendravimo per žvilgsnio sąveiką (angl. *Communication by Gaze Interaction*) narė, taigi dalyvauja jos inicijuojamuose tyrimuose. Taikomoji programa yra licencijuojama GPL laisvosios programinės įrangos licencija. „*EyeTyping*“ įrankis yra skirtas naudoti su *Tobii* žvilgsnio sekimo sistemomis. Naudotojui leidžiama žvilgsniu išrinkti ekrane pateiktos klaviatūros mygtukus ir taip įvedinėti tekstą. Programa „*EyeTyping*“ yra orientuota į tiriamuosius darbus, naujų akies judesių apdorojimo algoritmų įdiegimą, jų analizavimą, palyginimą. Programa yra kurta *Borland C++ Builder 5* programiniu paketu ir skirta naudoti *Windows* aplinkoje.



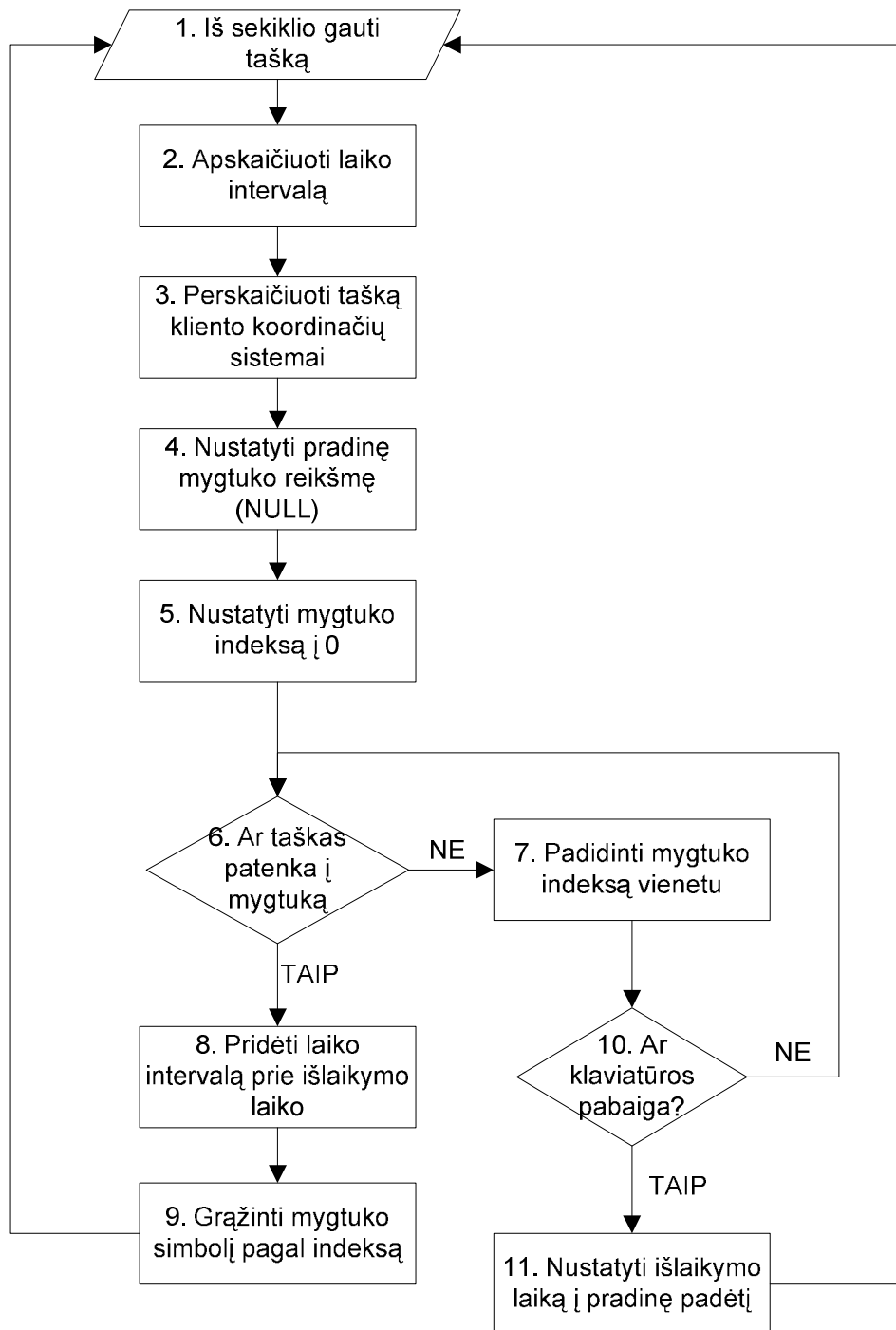
4.1 pav. Taikomosios programos EyeTyping pagrindinis langas

Programinė įranga atsisiunčiama ir įdiegiama kartu su *ETU-driver* – tvarkykle, kuri leidžia komunikuoti su *Tobii* žvilgsnio sekimo sistemomis.

4.1. EyeTyping programos veikimas

Išanalizavus taikomosios programos kodą, galima sudaryti apibendrintą programos veikimo algoritmą, kuris pateikiamas 4.2 pav.

¹ Taikomoji programa 2013-04-18 buvo atnaujinta ir dabar vadinama *AltTyping*. Priega per internetą: <<http://www.sis.uta.fi/~csolsp/downloads.php>>



4.2 pav. Taikomosios programos EyeTyping apibendrintas veikimo algoritmas

1. *EyeTyping* programos veikimo metu yra gaunamos taško X ir Y koordinatės bei laiko vertė iš žvilgsnio sekimo įrenginio. Programiškai taškas yra struktūra `TPoint`, kurioje saugomi sveiko tipo kintamieji `int x` ir `int y`.

```
TPoint pt = TPoint(smp->X[0]*X, smp->Y[0]*Y);
```

2. Turint prieš tai buvusio fiksacijos taško laiko vertę, galima apskaičiuoti laiko intervalą tarp vienas po kito atėjusių taškų. Ši trukmė kaupia išlaikymo laiko kintamąjį:

```

long lgpt = iLastGazePointTime;
iLastGazePointTime = smp->Time;
long interval = lgpt ? iLastGazePointTime - lgpt : 0;

```

3. Kadangi stebimo ir kompiuterio, kuriame veikia programa, ekranų matmenys gali nesutapti, gautas taškas perskaičiuojamas į klientinę (kompiuterio, kuriame vykdoma programa) sistemą:

```

TPoint pt = iParent->ScreenToClient(pt);

```

4. Perskaičiuotas taškas toliau siunčiamas į funkciją, kurios metu yra tikrinamas, ar jis patenka į mygtuką. Mygtukas programiškai yra aprašomas kaip struktūra `TButton`. Šiame žingsnyje prieš pradedant jo paiešką, nustatoma pradinė mygtuko kintamojo reikšmė (*NULL*).

```

btn = iKeyboard->findButtonFromPoint(pt);

```

5. Mygtukai klaviatūroje iš kairės į dešinę yra indeksuojami nuo 0 iki tiek, kiek yra objektų. Darbe naudojamos klaviatūros atveju 0 iki 29 (30 *QWERTY* klaviatūros mygtukų).

6. Objektas yra apibrežiamas kvadratine sritimi ir, jei taškas į ją patenka, funkcija grąžina tos srities kairiosios, dešinėsios, viršutinės ir apatinės kraštinių vidurio taškų koordinatas (*l, r, t, b*), pagal kurias vėliau nustatomas konkretus mygtukas.

```

TRect rc=TRect(btn->Viewport.GetLeft(),btn->Viewport.GetTop()...);
rc = TRect(int l, b, t, r);
if(::PtInRect(&rc, aPoint){
    result = btn;
    break;}

```

7. Jei taškas į mygtuką nepatenka, jo indeksas padidinamas vienetu.

8. Taško pataikymo į mygtuką atveju yra skaičiuojamas laikas. Kai ši vertė pasiekia nustatytą išlaikymo laiko vertę *iDwellTime*, mygtukas nuspaudžiamas.

```

btn->TimeAccumulated += aInterval;
...
If(aButton->TimeAccumulated >=iDwellTime){
    Press(aButton);}

```

9. Pagal indeksą nustatomas konkretus klaviatūros simbolis. Po šio žingsnio programoje atliekami tolimesni veiksmai, reikalingi atvaizduoti simboliui ekrane, patikrinti įvedamos frazės teisingumą bei suformuoti ataskaitos apie atliktą tyrimą bylą.

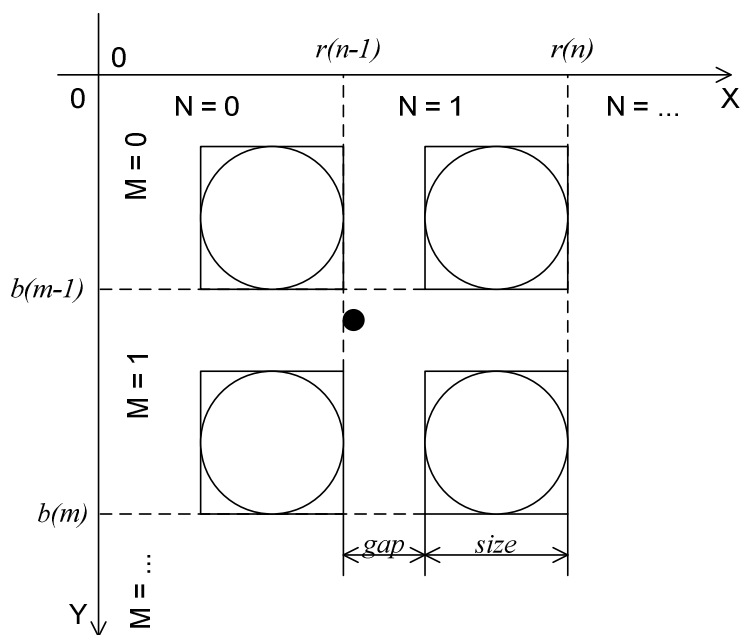
10. Jei taškas į nepatenka nei į vieną iš klaviatūros objektų, jokie tolimesni veiksmai su šiuo tašku neatliekami.

11. Išlaikymo laikas nustatomas į pradinę padėtį.

4.2. Taško padėties nustatymas

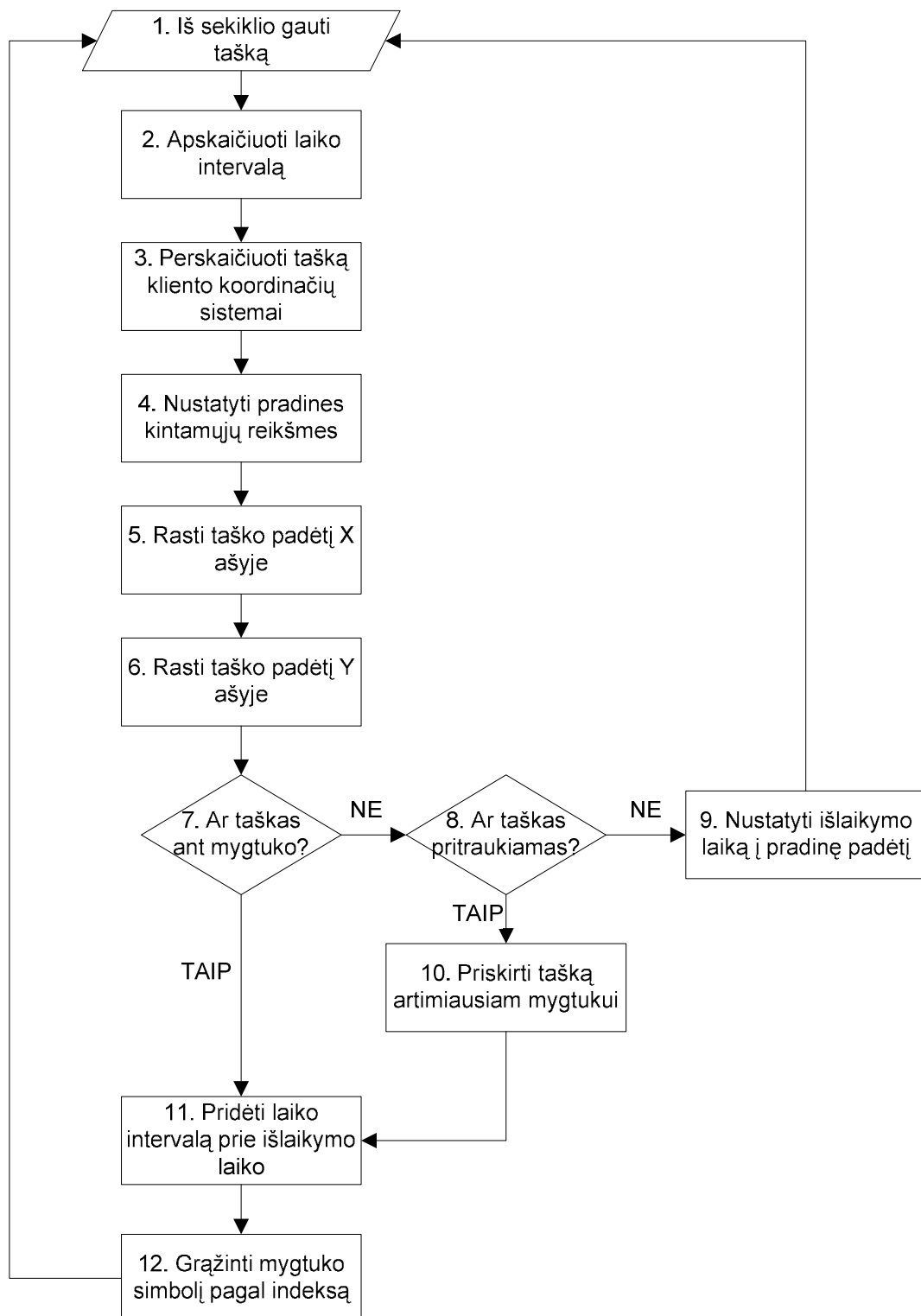
Atėjusio taško perskaičiavimą pagal greičio mažinimo, traukos lauko ar traukimo pusiaukraštinės kryptimi algoritmus galima įtraukti po trečiojo žingsnio, kai taškas yra perskaičiuotas kliento koordinačių sistemai, bet dar nepradėjus mygtuko paieškos. Norint niekur nepatekusiam mygtukui pritaikyti pritraukimo metodą, aptartas programos veikimo algoritmas nevisai tinka, nes jis apima tik mygtuko paiešką, o reiktų žinoti ir jo padėtį, kai mygtukas nerastas. Todėl šiame skyriuje bus aptartas kitas taško padėties nustatymo variantas.

Klaviatūrą galime išivaizduoti kaip dvimatį objektų – mygtukų – masyvą. Tačiau būtina įvertinti ir tai, kad klaviatūrą sudaro tarpai, kuriuose taškas taip pat gali užimti padėtį. Todėl paieška sudėtingėja. Kita vertus, mygtukų padėtis nekinta, todėl vietoj ciklo cikle naudojimo galima panaudoti du atskirus ciklus – vieno iš jų metu bus gauta informacija apie taško padėtį X ašies atžvilgiu, kito metu – Y ašies atžvilgiu. Kiekvienos iš ašių atveju, taškas gali užimti dvi padėtis – būti mygtuke arba būti tarpe tarp jų.

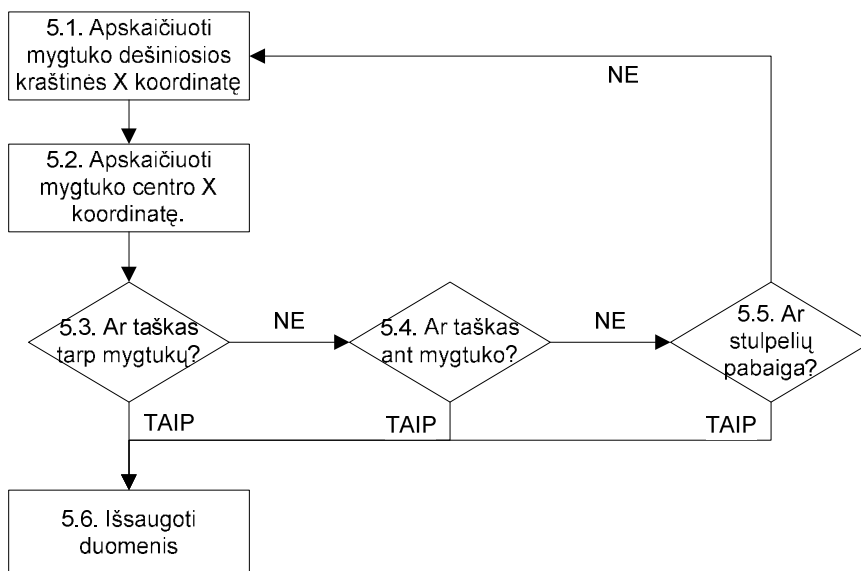


4.3 pav. Taško padėties nustatymo metodas naudojant du ciklus

4.2 pav. pateiktas apibendrintas programos veikimo algoritmas šiuo atveju turi būti papildytas 5 ir 6 žingsniais. Šie žingsniai detalizuojami atitinkamai 4.5 pav. ir 4.6 pav.



4.4 pav. Pakoreguotas programos veikimo algoritmas, skirtas taško padėčiai nustatyti

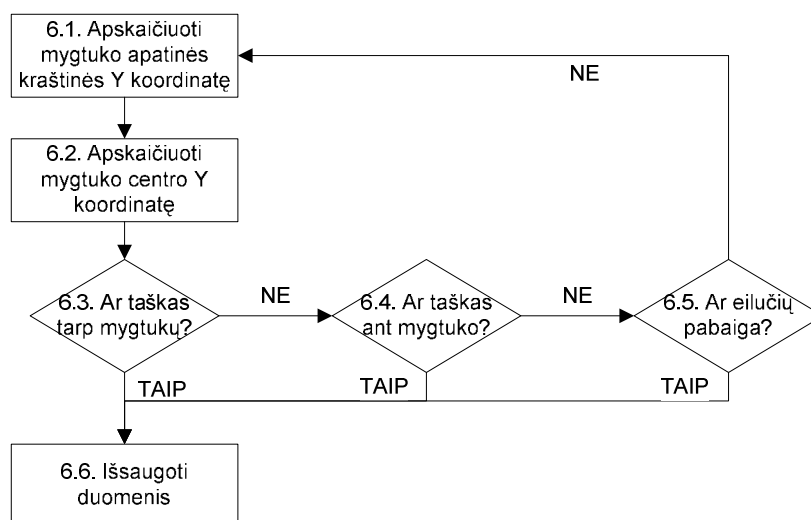


4.5 pav. Taško padėties X ašyje radimas

Kiekviena klaviatūros eilė yra suskaidyta į dešimt stulpelių (nes naudojama trisdešimties mygtuku klaviatūra), kuriuos sudaro tarpas ir mygtukas. Pirmojo ciklo (5 žingsnis) metu kiekvienoje iteracijoje perskaičiuojama dešinėsios objekto kraštinės koordinatė (r)

$$r(n) = \left(\frac{gap}{2} + size\right) + n \times (gap + size)$$

Tokiu būdu randama taško padėtis X ašyje paeiliui imant kiekvieną stulpelį. Nustatoma, ar tame stulpelyje esantis taškas užima padėtį tarpe (5.3) ar mygtuke (5.4). Radus taško padėtį arba pasiekus stulpelių galą, išsaugomu cikle surinkti duomenys – paskutinė dešinėsios objekto kraštinės koordinatė, ir stulpelio indeksas (n).



4.6 pav. Taško padėties Y ašyje radimas

Kiekvienas klaviatūros stulpelis yra suskaidytas į tris eilutes, kurias sudaro tarpas ir mygtukas. Antrojo ciklo (6 žingsnis) metu kiekvienoje iteracijoje perskaičiuojama apatinės objekto kraštinės koordinatė (b)

$$b(m) = \left(\frac{gap}{2} + size\right) + m \times (gap + size)$$

Tokiu būdu randama taško padėtis Y ašyje paeiliui imant kiekvieną eilutę. Nustatoma, ar tame stulpelyje esantis taškas užima padėtį tarpe (5.3) ar mygtuke (5.4). Radus taško padėtį arba pasiekus stulpelių galą, išsaugomu cikle surinkti duomenys – paskutinė apatinės objekto kraštinės koordinatė, ir stulpelio indeksas (m).

Galima pastebėti, kad paieškos ciklas X ašies atžvilgiu gali būti prasuktas dešimt kartų, o Y ašies atžvilgiu – tris kartus, taigi mygtuko padėties nustatymas neužtruks daugiau nei 13 iteracijų. Ciklo cikle atveju maksimalus iteracijų skaičius gali būti 30.

4.3. Pritraukimo algoritmo įgyvendinimas

Taško padėtį patogiu identifikuoti *bool* tipo kintamuoju, kuris bus „0“, kai taškas bus tarpe, ir „1“, kai taškas bus ant mygtuko. Po dviejų paieškų ciklų galimos keturios „0“ ir „1“ kombinacijos, kurios tiksliai nusakys mygtuko padėtį klaviatūroje:

11 – taškas ant vieno iš klaviatūros mygtukų;

01 – taškas tarpe tarp dviejų mygtukų X ašyje;

10 – taškas tarpe tarp mygtukų Y ašyje;

00 – taškas tarpų sankirtose tarp mygtukų.

Žinant taško padėtį ir objektą, prie kurio šis taškas yra, naudojant cikluose gautus indeksus tiesiogiai kreipiamasi į klaviatūros mygtuką. Gavus jo koordinates, atitinkamai modifikuojamos tos koordinatės, kurios ašies atžvilgiu mygtukas užima padėtį tarpe – tose pusėse sudaroma aktyvioji zona.

Traukos lauko, greičio mažinimo ir traukimo pusiaukraštinės kryptimi algoritmams įgyvendinti modifikuoti taško padėties nustatymą nebūtina. Tačiau įgyvendinimui pasirinktas aptartas trumpesnis paieškos metodas. Realizuojant šiuos algoritmus kiekviena taško koordinatė (X ir Y) perskaičiuojamos atskirai. Tačiau būtina paminėti, kad realizuojant traukos lauko algoritmą, Euklido atstumas (žr. 3.2 poskyrį.) apskaičiuojamas tarp $\vec{a}$ ir $\vec{p}$, ir naudojamas tas pats perskaičiuojant kiekvieną taško koordinatę

$$n_x = a_x + s \cdot \left\| \vec{a} - \vec{p} \right\| \cdot \frac{\vec{f} - \vec{a}}{\left| \vec{f} - \vec{a} \right|}, \text{ ir}$$

$$n_y = a_y + s \cdot \left\| \vec{a} - \vec{p} \right\| \cdot \frac{\vec{f} - \vec{a}}{\left| \vec{f} - \vec{a} \right|}$$

Tuo tarpu modifikuojant taškus traukimo pusiauakraštinės algoritmu, kiekvienai koordinatei perskaičiuoti naudojamos atitinkamai X ir Y dedamosios:

$$n_x = a_x + s \cdot \left| a_x - \frac{f_x + p_x}{2} \right| \cdot \frac{\vec{f} - \vec{a}}{\left| \vec{f} - \vec{a} \right|}, \text{ ir}$$

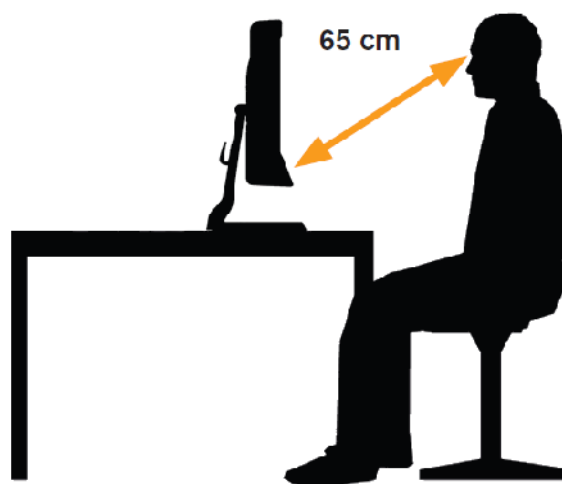
$$n_y = a_y + s \cdot \left| a_y - \frac{f_y + p_y}{2} \right| \cdot \frac{\vec{f} - \vec{a}}{\left| \vec{f} - \vec{a} \right|}$$

Visų darbo apimtyje tirtų algoritmų programų kodai yra pateikiami antrajame priede.

5. REALIZUOTO METODO TYRIMO REZULTATAI

5.1. Tyrimo eiga

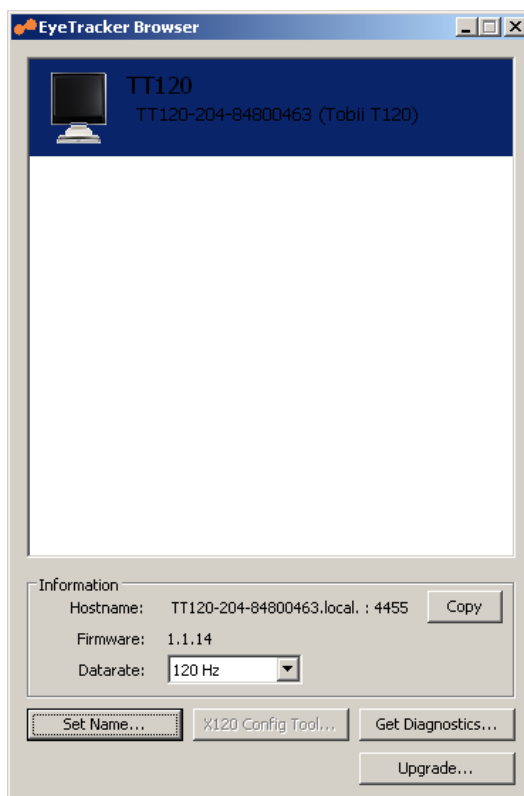
Tyrimas buvo atliktas naudojant *Tobii T120* žvilgsnio sekiklį ir *ASUS F5R* serijos nešiojamą kompiuterį, kuriame veikė paruošta programa. Sistema buvo prijungta ir sukonfigūruota pagal naudotojo vadove pateiktus nurodymus. Vaizdas iš kompiuterio į sekiklį buvo perduodamas *VGA*, o duomenys – *Ethernet* laidu. Iš gamintojo internetinės svetainės buvo atsisiųsta ir įdiegta *Tobii EyeTracker Tools* programa bei tvarkyklė *Tobii EyeTracker ETU driver*, kuria buvo sukonfigūruotas įrenginio prijungimas prie kompiuterio.



5.1 pav. Atstumo reikalavimai tyrimo dalyviui

Remiantis naudotojo vadove pateiktomis rekomendacijomis, atstumas tarp tiriamojo akių ir sekiklio apatinėje dalyje įmontuotų infraraudonųjų spindulių šviesos šaltinių turi būti apie 65 cm. Pabrėžiama, kad labai svarbu šį atstumą išlaikyti pastovų, nes kitu atveju, sekiklio kamera negali užfiksuoti visos akių aprėpiamos srities. Ekranas turėtų būti pastatytas taip, kad tiesė, vedanti nuo tiriamojo akių iki sekiklio šviesos šaltinių su horizontale sudarytų maždaug 42 laipsnių kampą.

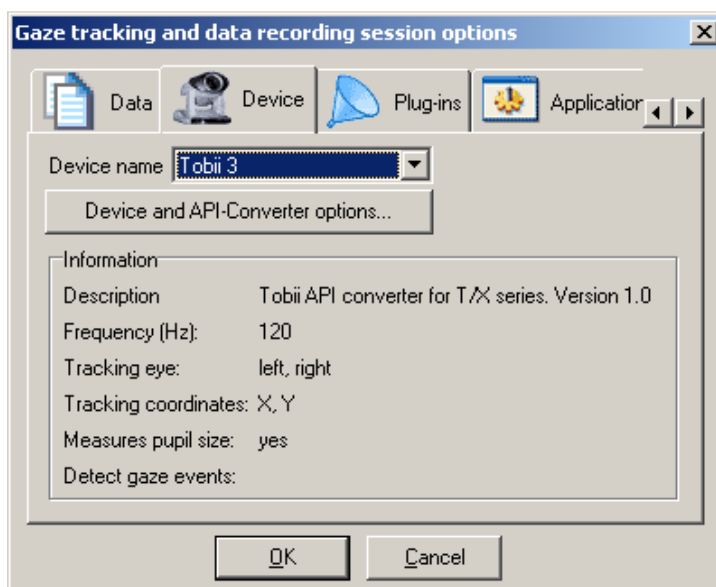
Prieš paleidžiant taikomąją programą pirmiausia būtina įjungti *EyeTracker Browser* programą, kuri pateikia visus prie kompiuterio prijungtus žvilgsnio sekimo įrenginius ir prisijungimo prie jų duomenis (5.2 pav.).



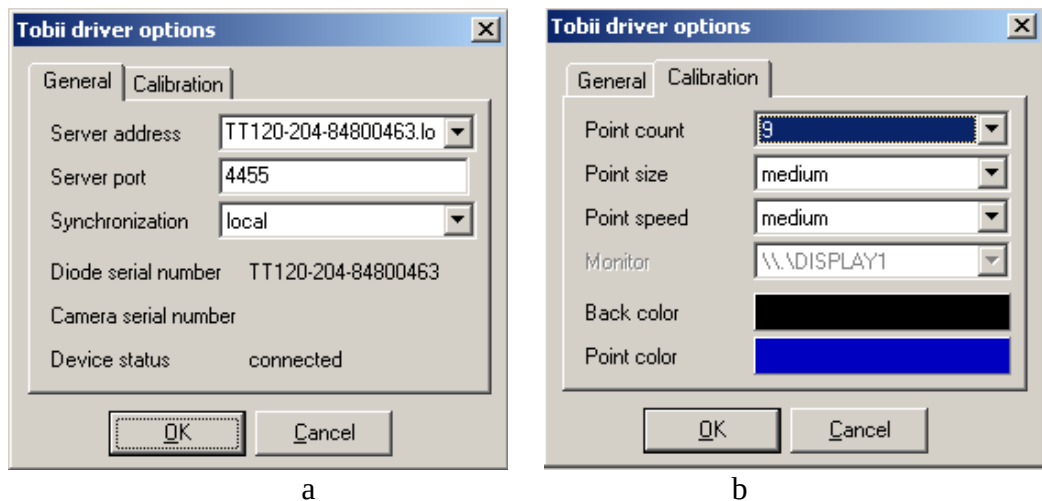
5.2 pav. Eyetracker Browser programos pagrindinis langas

Paleidus paruoštą *EyeTyping* taikomąją programą su implementuotu algoritmu, pagal *EyeTracker Browser* pateiktą informaciją pirmiausia buvo parinkti reikiami nustatymai:

1. Įrenginio vardas (angl. *Device name*) – Tobii 3 (5.3 pav.);
2. Prisijungimo duomenys: Serverio adresas, prievadas, sinchronizacija (5.4 pav. a);
3. *Tobii* tvarkyklės parinktys įrenginio kalibravimui (5.4 pav. b).



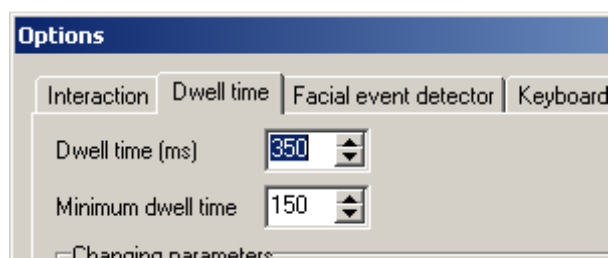
5.3 pav. Įrenginio nustatymų parinkimas EyeTyping programoje



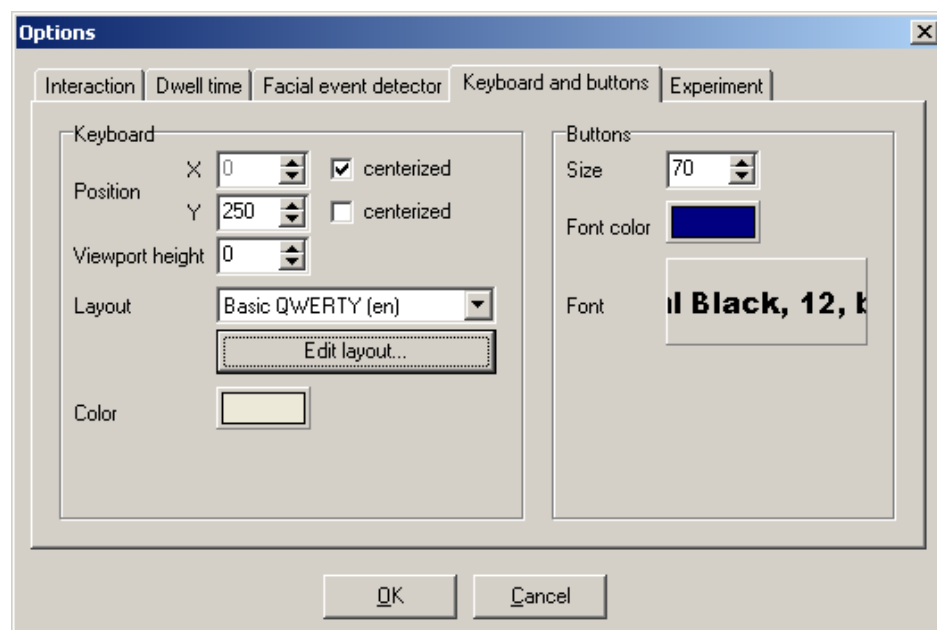
5.4 pav. Tobii tvarkyklės nustatymų langai EyeTyping programoje

Nustačiu įrangos parametrus, toliau nustatomi eksperimento parametrai:

4. Išlaikymo laikas (5.5 pav.);
5. Mygtukų dydis (5.6 pav.).

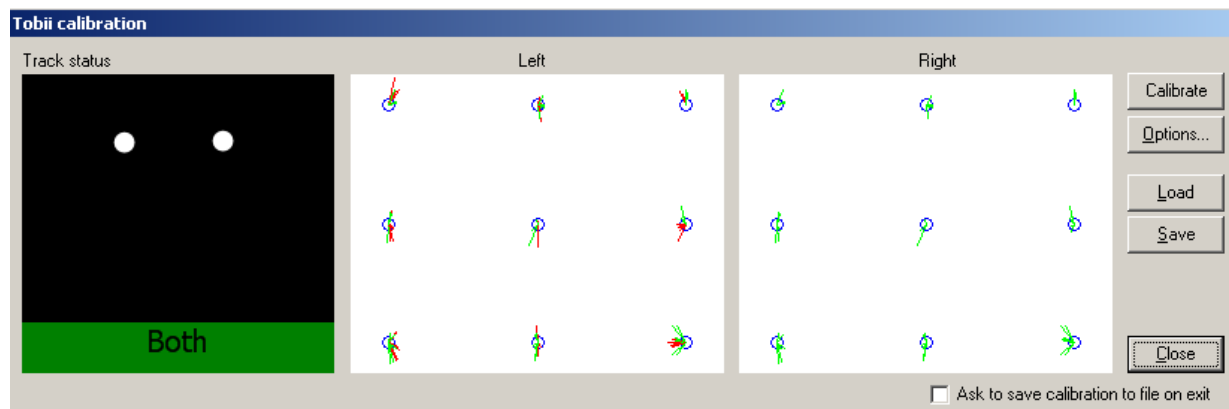


5.5 pav. Išlaikymo laiko nustatymų kortelė



5.6 pav. Klaviatūros ir jos mygtukų nustatymų langas

Po reikalingų parinkčių nustatymo sistema sukalibruojama (5.7 pav.) ir pradedamas eksperimentas.



5.7 pav. Tobii sekiklio kalibravimas

Eksperimento metu naudotojui ekrane buvo pateikiamos frazės, kurių simbolius jis turėjo rinktis vien žiūrint į juos. Frazės EyeTyping taikomosios programos pagalba buvo parenkamos atsitiktine tvarka iš paruoštos *.txt bylos. Bandymo metu buvo fiksuojamas sistemos laikas: bandymų pradžia, sekančios frazės pateikimas, bandymų pabaiga. Po kiekvieno bandymo pabaigos, programa suformuoja statistinę *.txt formato bylą, kuri apima, apskaičiuotus klaidos koeficientus, sugaištą laiką, pasirinktų simbolių skaičių per laiko vienetą. Sekiklis buvo perkalibruojamas kas 5 – 10 frazių;

Prioritetas buvo išbandyti priskyrimo centrui algoritmą. Kiekvienam procentiškai išreikštam pritraukimo lygiui buvo išmėginti trys išlaikymo laikai – iš viso, šio algoritmo tyrimui buvo sudaryta dvylika scenarijų. Tyrimas buvo atliekamas pagal šiuos bandymų scenarijus:

Išlaikymo laikas: 400, 350, 300 ms.

Pritraukimas: 0%, 33%, 66%, 100%.

Taip pat buvo išbandyti greičio mažinimo, traukos lauko bei traukimo pusiaukraštinės kryptimi algoritmai. Jų analizei buvo pasirinktas vienas pjūvis išlaikymo laiko atžvilgiu (350 ms).

Iš gautų rezultatų kiekvienas algoritmas buvo vertinamas pagal šiuos kriterijus:

- Rašymo kokybę – procentais išreikštas teisingų įvestų simbolių skaičius iš visų pateiktos frazės simbolių;
- Rašymo greitį – įvestų žodžių skaičius per minutę;
- Rašymo efektyvumą – procentiškai išreikštas laiko, išiekvoto mygtuko paspaudimui, ir viso laiko santykis;

Įvedus simbolių daugiau nei reikia, pavyzdžiui bandant pasirinkti simbolį kelis kartus nesėkmingai, klaida didėja. Todėl tyrimo dalyviams buvo nurodyta netaisyti padarytų klaidų ir tęsti

sekančių frazės simbolių rinkimą. Dėl akių nuovargio ar kitų nenumatytų nesklandumų, tyrimo dalyviui paprašius jo dalyvavimas bandyme buvo nutraukiamas.

5.2. Tyrimo rezultatai

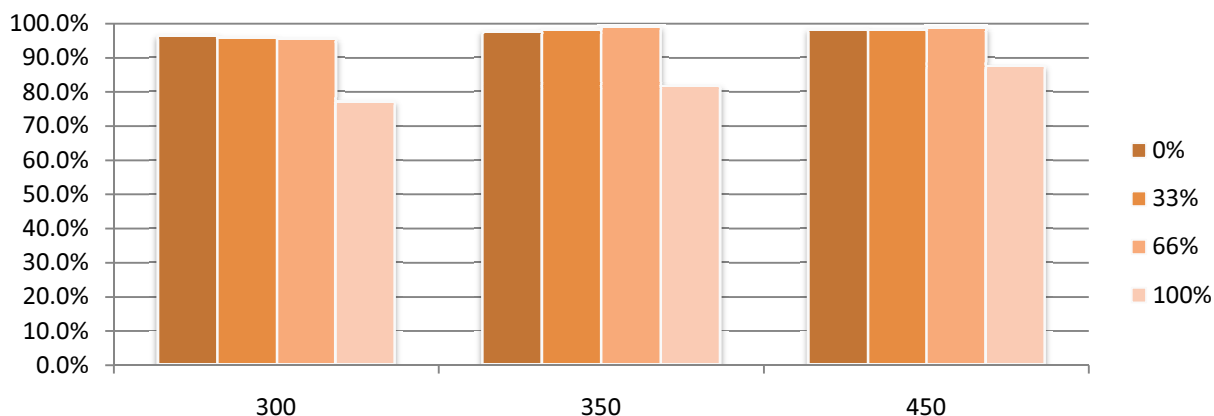
Tyrimė iš viso sudalyvavo 8 asmenys, kurių amžius 19 – 27 metų. Keturi iš jų – trumparegiai, trys iš kurių nešiojo akinius ir vienas – kontaktinius lęšius. Kiti tyrimo dalyviai regos problemų neturėjo. Prieš bandymus dalyviai buvo supažindinti su tyrimu, jo paskirtimi, specifika ir eiga, buvo skirta laiko priprasti prie sąsajos. Dalyviams buvo nurodyta pasistengti išlaikyti stabilią galvos padėtį.

Kiekvieno tyrimo scenarijaus metu buvo suvesta po 30 frazių, kuriose vidutiniškai buvo 28 - 30 simbolių. Surinkus reikiamą vieno scenarijaus frazių kiekį, duomenys buvo perkelti į *MS Office Excel* aplinką. Įvykdžius visus užsibrėžtus bandymų scenarijus, gauti rezultatai buvo susumuoti ir apibendrinti.

Pirmiausia buvo tiriamas pritraukimo (sutapdinimo su taikinio centru) algoritmas. Šis eksperimentas buvo vykdomas su trimis išlaikymo vertėmis: 300 ms, 350 ms ir 450 ms. 2 lentelėje pateikiami rašymo kokybės rezultatai, gauti įvykdžius visus pritraukimo algoritmo eksperimento scenarijus.

2 lentelė. Pritraukimo algoritmo rašymo kokybės rezultatai

Pritraukimas, % Dwell Time, ms	0	33%	66%	100%
300	96,4%	95,8%	95,5%	77,0%
350	97,6%	98,3%	99,0%	81,7%
450	98,3%	98,0%	98,6%	87,4%



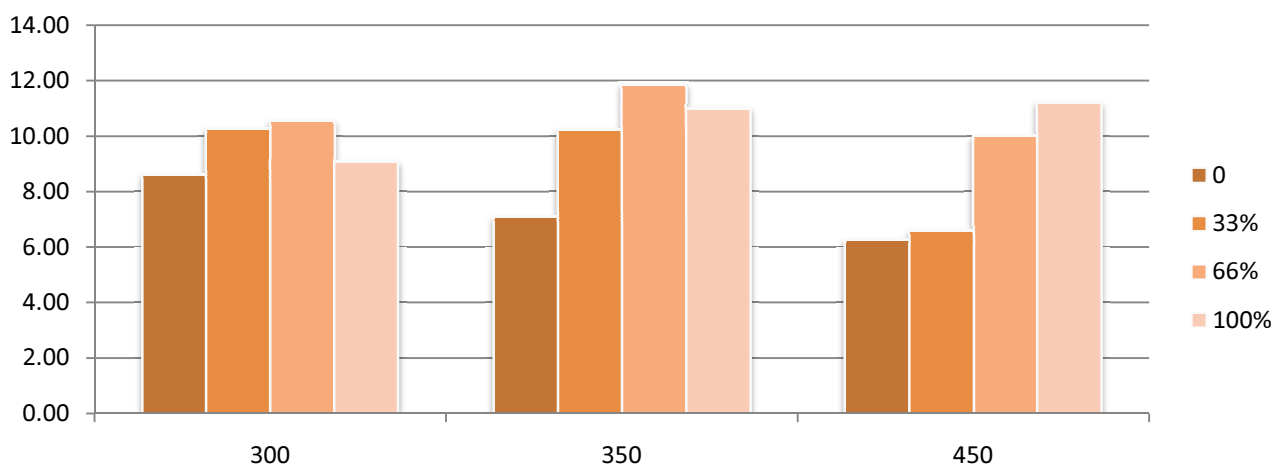
5.8 pav. Rašymo kokybė keičiantis pritraukimo algoritmo laipsniui ir išlaikymo laikui

Iš rezultatų galima pastebėti, kad rašymo kokybė visais atvejais praktiškai nekinta ir išlieka pakankamai aukšta. Ji krenta, kai naudojamas 100 % pritraukimas, t.y., kai taškas bet koku atveju priskiriamas vienam ar kitam klaviatūros objektui. Šiuo atveju algoritmas tampa ypač „lipnus“, dėl ko padidėja simbolių parinkimo klaidų skaičius. Galima pastebėti, kad net ir netaikant jokio algoritmo (netraukiant taško), rašymo kokybė yra aukšta. Akivaizdu, kad šiuo atveju naudotojas turi daugiau laiko nukreipti dėmesį nuo vieno objekto prie kito, o žvilgsnio nukreipimo metu kiti pašaliniai mygtukai neaktyvuojami. Tačiau algoritmo netaikymas daro įtaką kitiems dviems kriterijoms.

3 lentelėje pateikiami rašymo greičio rezultatai, gauti po visų eksperimento scenarijų.

3 lentelė. Pritraukimo algoritmo rašymo greičio rezultatai

Pritraukimas, % Dwell Time, ms	0 (netraukiant)	33%	66%	100%
300	8,6	10,3	10,5	9,1
350	7,1	10,2	11,9	11,0
450	6,3	6,6	10	11,2



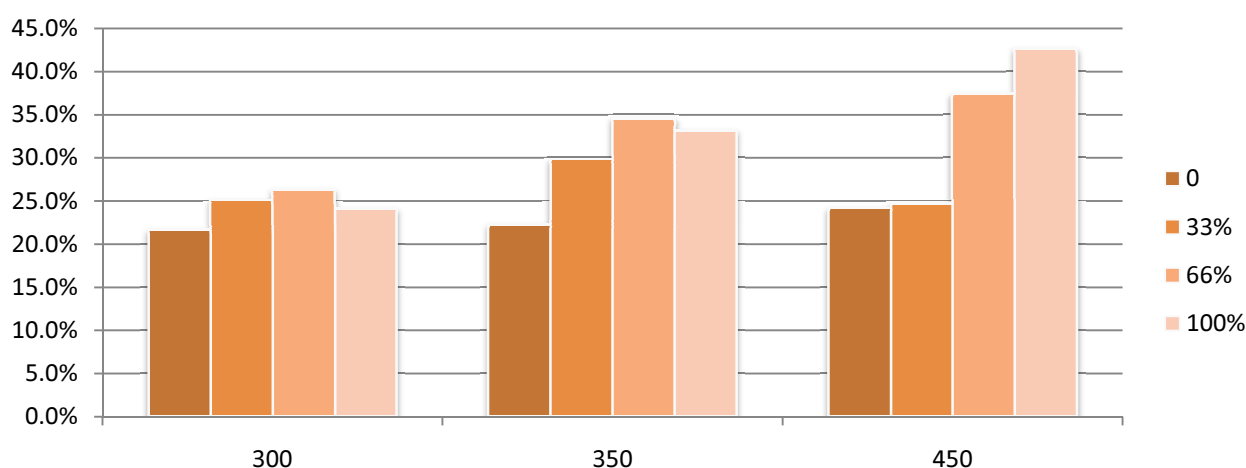
5.9 pav. Rašymo greičio rezultatai keičiant pritraukimo algoritmo laipsnį ir išlaikymo laiką

Iš rezultatų galima pastebėti, kad taikant algoritmą didėja simbolių parinkimo greitis. Geriausias rezultatas pasiekiamas taikant 66 % pritraukimą, o netraukiant taško, jis nesiekia 9 ž/min. 100 % traukimo atveju, greitis taip pat didelis, tačiau, kaip jau buvo aptarta, klaida naudojant šį pritraukimo algoritmo variantą yra didesnė nei kitų.

4 lentelėje pateikiami rašymo efektyvumo rezultatai.

4 lentelė. Pritraukimo algoritmo rašymo efektyvumo rezultatai

Pritraukimas, % Dwell Time, ms	0	33%	66%	100%
300	21,6%	25,1%	26,3%	24,1%
350	22,2%	29,8%	34,6%	33,2%
450	24,2%	24,7%	37,5%	42,6%

**5.10 pav.** Rašymo efektyvumo rezultatai keičiant pritraukimo algoritmo laipsnį ir išlaikymo laiką

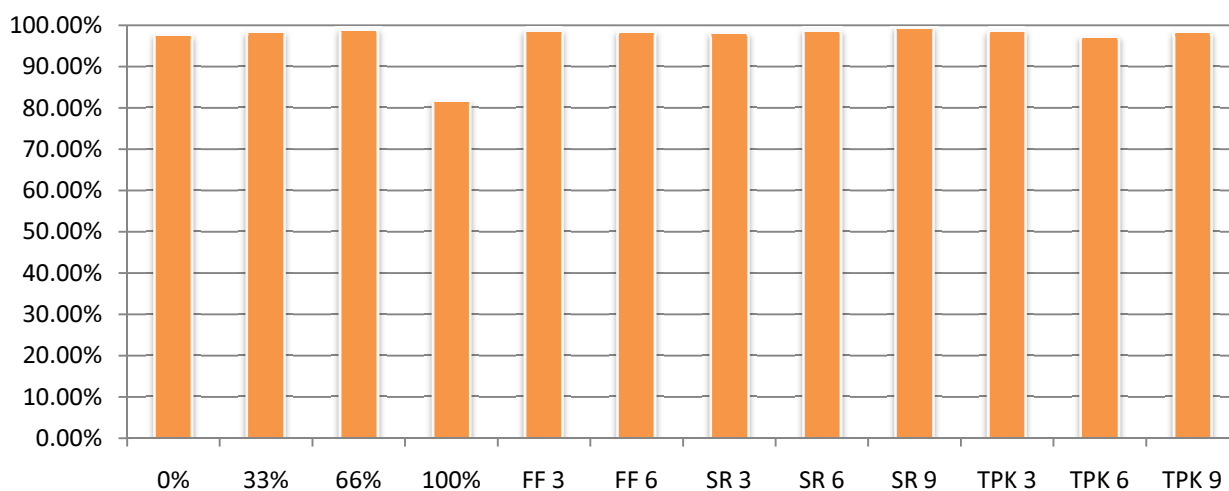
Verta atkreipti dėmesį, kad esant mažam išlaikymo laikui rašymo efektyvumas yra mažiausias visais pritraukimo algoritmo atvejais, o išlaikymo laikui didėjant, efektyvumas taip pat didėja. Teoriškai, per trumpesnę išlaikymo laiką turėtų įvykti mažiau žvilgsnio taško objekto ribų kirtimų, dėl ko, šis laikas mažiau kartų turėtų būti nustatomas į pradinę padėtį. Todėl, teoriniu požiūriu, esant mažesniai išlaikymo laikui efektyvumas turėtų būti didesnis, tačiau praktinis eksperimentas parodė visiškai priešingus rezultatus. Taip pat pažymėtina, kad teoriniu požiūriu panašią įtaką turi daryti ir pritraukimo laipsnio didinimas – kuo jis didesnis, tuo mažiau objekto ribų kirtimų įvyksta išlaikymo laiko skaičiavimo metu, ir rašymo laikas išnaudojamas efektyviau. Panašu, kad būtent tai atsispindi rezultatuose. Pastebėtina ir efektyvumo kitimo progresija, išryškėjanti rašymo efektyvumo rezultatuose, kaip jis didėja 100% pritraukimo, ir algoritmo netaikymo atvejais. Iš rezultatų matyti, kad kuo didesnis pritraukimo laipsnis, tuo labiau, kintant išlaikymo laikui, tai turi įtaką rašymo efektyvumui.

Iš gautų pritraukimo algoritmo eksperimento rezultatų galima pastebėti, kad geriausiai veikia variantas su 66 % pritraukimu. Visais išlaikymo laiko atvejais juo buvo gauti geriausi rezultatai.

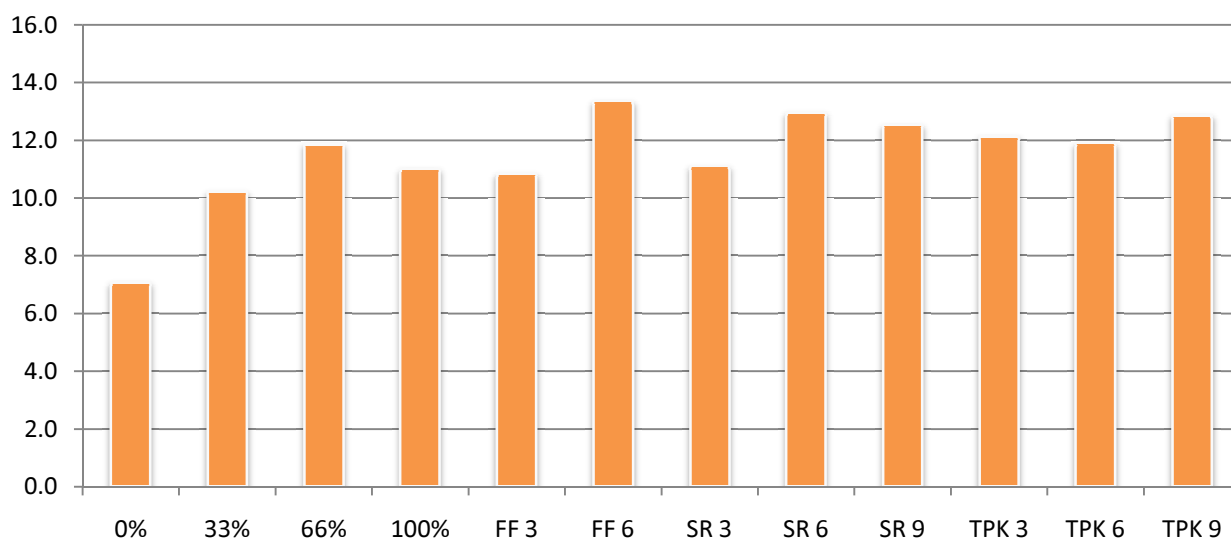
Antrajai tyrimo daliai buvo pasirinktas vienas išlaikymo laikas 350 ms, ir eksperimentų metu esant šiam laikui buvo išbandyti visi darbo apimtyje tirti algoritmai. Apibendrinti tyrimo rezultatai pateikiami 5 lentelėje ir 5.11, 5.12 bei 5.13 paveiksluose pavaizduotuose grafikuose.

5 lentelė. Traukimo algoritmų palyginimas, kai išlaikymo greitis 350 ms

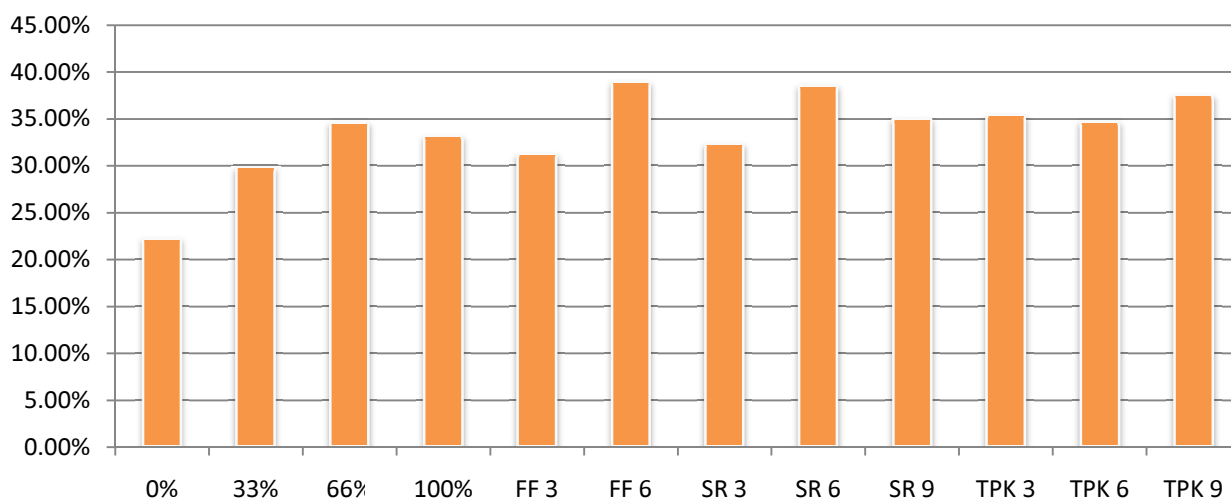
Metodas	Klaida, %	Rašymo greitis, ž/min	Efektyvumo koeficientas, %	Rašymo kokybė, %
0% (netraukiant)	2,3%	7,1	22,3%	97,7%
33%	1,7%	10,2	29,8%	98,3%
66%	1%	11,9	34,6%	99%
100%	18,3%	11	33,2%	81,7%
FF 3	1,4%	10,8	31,2%	98,6%
FF 6	1,6%	13,3	38,9%	98,4%
SR 3	1,9%	11,1	32,4%	98,1%
SR 6	1,2%	12,9	38,5%	98,8%
SR 9	0,6%	12,5	35%	99,4%
TPK 3	1,3%	12,1	35,4%	98,7%
TPK 6	2,8%	11,9	34,7%	97,2%
TPK 9	1,7%	12,9	37,5%	98,3%



5.11 pav. Traukimo algoritmų rašymo kokybės palyginimas



5.12 pav. Traukimo algoritmų rašymo greičio palyginimas



5.13 pav. Traukimo algoritmų rašymo efektyvumo palyginimas

Iš gautų rezultatų galima pastebėti, kad ir šiuo eksperimentu pasiekta rašymo kokybė gaunama aukšta ir praktiškai nekinta taikant kiekvieną iš algoritmų. Klaida, neskaitant 100 % pritraukimo algoritmo varianto, kinta vos 3 % intervale.

5.12 paveiklėse pateiktame grafike galima pastebėti, kad taikant traukos lauko, greičio mažinimo ir traukimo pusiau kraštinės kryptimi algoritmus rašymo greitis pasiekiamas didesnis nei taikant pritraukimo algoritmo variantus. Panaši tendencija pastebima ir rašymo efektyvumo rezultatus atvaizduojančiame grafike (5.13 pav.). Geriausias rašymo greitis ir efektyvumas pasiekiamas taikant traukos lauko algoritmą su traukimo stiprio koeficientu $s = 0,6$ (FF 6), greičio mažinimo algoritmą su greičio mažinimo koeficientu $r = 0,6$ (FF 6) bei traukimo pusiau kraštinės kryptimi su traukimo stiprio koeficientu $s = 0,9$ (TPK 9).

6. APIBENDRINIMAS IR IŠVADOS

Šio magistro darbo metu buvo ištirta pritraukimo (sutapdinimo su taikinio centru) algoritmo įtaka rašymo taikomąja programa *EyeTyping* kokybei, greičiui bei efektyvumui. Darbe pateikta akies judesių apdorojimo ir žvilgsnio taškų klasifikavimo metodų analitinė apžvalga, detaliai išanalizuoti trys traukimo savybėmis pasižymintys, žvilgsnio stabilumui išlaikyti skirti algoritmai: pritraukimo (sutapdinimo su taikinio centru), traukos lauko ir greičio mažinimo. Šių algoritmų simuliacijos pagalba buvo atskleisti kiekvieno iš jų privalumai ir trūkumai. Įvertinus juos, darbe pasiūlytas naujas traukimo savybėmis pasižymintis metodas – traukimo pusiaukraštinės kryptimi algoritmas. Visi darbo apimtyje aptarti algoritmai buvo įgyvendinti į taikomąją programą ir išbandyti praktiškai. Eksperimentuose iš viso sudalyvavo 8 asmenys.

1. Analizuojant literatūros šaltinius, kuriuose pateikiami akies judesių apdorojimo ir žvilgsnio taškų klasifikavimo metodai išryškėjo dvi algoritmų grupės: skirti taškų protokolo analizei ir skirti interaktyviajai žvilgsniu paremtai sąsajai tarp žmogaus ir kompiuterio realizuoti. Nors šaltiniuose buvo teigiama, kad kai kurie iš taškų protokolo analizės metodų gerai veikia ir realiu laiku, to įrodymų rasti nepavyko;
2. Algoritmus, taikomus žvilgsniu paremtai sąsajai realizuoti, taip pat galima suskirstyti į dvi kategorijas: vienai jų priklauso algoritmai, kuriais perskaičiuojamas išlaikymo laikas (pvz., varžymosi, ar dinamiško varžymosi algoritmai), kitai kategorijai priklauso algoritmai, kuriai perskaičiuojamos taško koordinatės. Šie algoritmai pasižymi traukimo prie objekto, į kurį sutelktas dėmesys, savybėmis (pvz., pritraukimo (sutapdinimo su taikinio centru), traukos lauko, greičio mažinimo metodai;
3. Išanalizavus literatūros šaltinius, kuriuose aprašomi traukimo algoritmų tyrimai, pastebėta, kad pritraukimo algoritmo nauda nėra atskleista, o geriausiai žvilgsnio tašką stabilizuoja greičio mažinimo algoritmas;
4. Išanalizavus darbo apimtyje tiriamus algoritmus ir atlikus traukos lauko ir greičio mažinimo metodų simuliaciją *Matlab* programiniu paketu, buvo nustatyta, kad:
 - a. tiek traukos lauko, tiek greičio mažinimo algoritmo atveju, kol s ir r yra nedideli, perskaičiuoto taško koordinatės nedaug skiriasi nuo taško, atėjusio iš sekiklio, koordinatė, todėl žvilgsnio taško padėtis praktiškai nestabilizuojama;
 - b. didinant koeficientus s ir r traukos lauko ir greičio mažinimo algoritmus išreiškiančiose formulėse, perskaičiuotų taškų spiečius gaunamas labiau koncentruotas, tačiau taip pat galima pastebėti šių taškų išsibarstymą plote tarp objektų, kuris pasireiškia nukreipiant dėmesį nuo vieno objekto prie kito;

- c. pritraukimo (sutapdinimo su taikinio centru) metodas, skirtingai nei traukos lauko ar greičio mažinimo algoritmai, yra statiškas, ir jo veikimas visiškai priklauso nuo taikomosios programos objektų ekrane padėties;
 - d. pasiūlytas traukimo pusiaukraštinės kryptimi algoritmas ypač sumažina taškų išsibarstymą didinant traukimo koeficientą (3.11 pav., 3.12 pav., 3.13 pav.);
5. Atlikus eksperimentą pastebėta, kad pritraukimo algoritmas neturi ryškos įtakos rašymo virtualia klaviatūra kokybei, greičiui ar efektyvumui. Nors algoritmas buvo įgyvendintas remiantis teorine analize, praktiškai algoritmo įgyvendinimas yra ekvivalentus artimiausio mygtuko ribų padidinimui.
 6. Naudojant 100 % pritraukimą pasireiškia „lipnumas“, dėl ko ypač išauga klaidingas simbolių parinkimas. Netaikant jokio algoritmo, rašymo kokybė yra gana aukšta tačiau rašymo greitis, dėl didelio išsibarstymo įtakos yra mažas. Iš gautų pritraukimo algoritmo eksperimento rezultatų pastebėta, kad geriausiai veikia variantas su 66 % pritraukimu. Visais išlaikymo laiko atvejais juo buvo gauti geriausi rezultatai.
 7. Atlikus eksperimentą buvo nustatyta, kad:
 - a. rašymo kokybė neišryškina vieno ar kito taškų stabilizavimo algoritmo pranašumo – beveik visais atvejais ji nesiskyrė daugiau nei 3 %;
 - b. geriausias rašymo greitis (13,3 ž/min) pasiektas taikant traukos lauko algoritmą kai $s = 0,6$. Taip pat aukšti rašymo greičio rezultatai (12,9 ž/min) pasiekiami taikant greičio mažinimo algoritmą, kai $r = 0,6$ ir traukimo pusiaukraštinės kryptimi algoritmą, kai $s = 0,9$. Taip pat, taikant šiuos algoritmus su tokiomis pačiomis konfigūracijoms, rašymo laikas išnaudojamas efektyviausiai – efektyvumas siekia apie 37 %. Remiantis šiais rezultatai galima teigti, kad pasiūlytas naujas traukimo pusiaukraštinės kryptimi algoritmas vienodai gerai stabilizuoja žvilgsnio taškus lyginant jį su literatūroje nagrinėjamais algoritmais;
 8. Atliekant eksperimentą buvo pastebėta, kad rezultatams didelės įtakos turi apšvietimas ir šviesos šaltinių išsidėstymas, taip pat ir fiziologinės žmogaus akių savybės, jų nuovargis ar gebėjimas išlaikyti koncentruotą dėmesį.

Literatūra

Knyga

1. DUCHOWSKI, A. T. 2007. *Eye Tracking Methodology: Theory and Practice*: vadovėlis. JAV: Springer. 360 p.

Nuorodos į interneto šaltinius

2. *Eye Controlled Media: Present and Future State, 2.1 Techniques Based on Reflected Light* [interaktyvus]. 1995. GLENSTRUP, A. J.; ENGEL-NIELSEN, T. [žiūrėta 2012 m. sausio 11 d.]. Prieiga per internetą: <<http://www.diku.dk/~panic/eyegaze/node9.html>>
3. *Measurement principle of SimpleGazeTracker* [interaktyvus]. Gaze Parser. [žiūrėta 2013 m. balandžio 2 d.] Prieiga per internetą: <<http://gazeparser.sourceforge.net/adv/principle.html>>

Straipsniai žurnaluose (periodiniuose leidiniuose)

4. KEIL, A.; ALBUQUERQUE, G.; BERGER, K.; MAGNOR, M. A. 2010. Real-time gaze tracking with a consumer-grade video camera, *WSCG'2010 Proceedings*: 129 – 134.
5. BLIGNAUT, P. 2009. Fixation identification: The optimum threshold for a dispersion algorithm, *Attention, Perception, & Psychophysics* 71(4): 881 – 895.
6. SALVUCCI, D. D.; GOLDBERG, J. H. 2000. Identifying fixations and saccades in eye-tracking protocols, *ACM ETRA '00 Proceedings of the 2000 symposium on Eye tracking research & applications*: 71 – 78.
7. SALVUCCI, D. D.; ANDERSON, R. J. 2000. Intelligent gaze-added interfaces, *CHI'00 Proceedings of the SIGCHI conference on Human Factors in Computing Systems*: 273 – 280.
8. AHLSTRÖM, D. 2005. Modeling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields, *CHI'05 Proceedings of* : 61 – 70.
9. XINYONG, Z.; XIANGSHI, R.; HONGBIN, Z. 2008. Improving eye cursor's stability for eye pointing tasks, *CHI'08 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*: 525 – 534.
10. ŠPAKOV, O. 2011. Comparison of gaze-to-object mapping algorithms, *NGCA'11 Proceedings of the 1st Conference on Novel Gaze-Controlled Applications*: Straipsnio nr.6 .
11. GIZATDINOVA, Y.; ŠPAKOV, O.; SURAKKA, V. 2012. Face typing: Vision-based perceptual interface for hands-free text entry with a scrollable virtual keyboard, *IEEE Workshop on Applications of Computer Vision (WACV)*: 81-87.
12. GIZATDINOVA, Y.; ŠPAKOV, O.; SURAKKA, V. 2012. Comparison of video-based pointing and selection techniques for hands-free text entry, *AVI'12 Proceedings of the International Working Conference on Advanced Visual Interfaces*: 132 – 139.
13. XINYONG, Z.; XIANGSHI, R.; HONGBIN, Z. 2010. Modeling dwell-based eye pointing target acquisition, *CHI'10 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*: 2083 – 2092.

14. MAJARANTA, P.; AHOLA, U. K.; ŠPAKOV, O. 2009. Fast gaze typing with an adjustable dwell time, *CHI'09 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*: 357 – 360.
15. ZHU, Z.; JI, Q. 2004. Eye and gaze tracking for interactive graphic display, *Machine Vision and Application*15(3): 139 – 148.
16. HANSEN, J. P.; JOHANSEN, A. S.; HANSEN, D. W.; ITOH, K. 2003. Command Without a Click: Dwell Time Typing by Mouse and Gaze Selections, *Proceedings of Human-Computer Interaction – INTERACT'03*: 121 – 128.
17. MAJARANTA, P.; ŠPAKOV, O. 2009. Scrollable keyboards for casual eye typing, *PsychNology Journal*7(2): 159 – 173.
18. ŠPAKOV, O.; MINIOTAS, D. 2005. Gaze-based selection of standard-size menu items, *ICMI '05 Proceedings of the 7th international conference on Multimodal interfaces*: 124 – 128.
19. ŠPAKOV, O.; MINIOTAS, D. 2004. On-line adjustment of dwell time for target selection by gaze, *NordiCHI'04 Proceedings of the third Nordic conference on Human-computer interaction*: 203 – 206.
20. HANSEN, J. P.; HANSEN, D. W.; JOHANSEN, A. S. 2001. Bringing gaze-based interaction back to basics, *Conference: Human-Computer Interaction – HCI*: 325 – 329.
21. JACOB, R. J. K. 1995. Eye tracking in advance interface design, *Virtual environments and advanced interface design*: 258 – 288.
22. DE SANTIS, A.; IACOVIELLO, D. 2009. Robust real time eye tracking for computer interface for disabled people, *Computer Methods and Programs in Biomedicine*96(1): 1 – 11.
23. SUNG, K.; RESCHKE, M. F. 1997. A model-based approach for the measurement of eye movements using image processing, *NASA technical paper no. 3680*.

PRIEDAI

1 priedas. Algoritmų veikimo simuliacinio *Matlab* programiniu paketu skriptas

Šiame priede pateikiamas skriptas, skirtas *Matlab* programiniu paketu susimuliuoti šešis taškų spiečius, atitinkančiu šešis atsitiktinius objektus, į kuriuos buvo sutelktas dėmesys. Objektai pateikiami apibrėžti juodu apskritimu. Funkcija centre yra sukurta Zhenhai Wang ir atsisiųsta naudojant priegai per internetą: <http://www.mathworks.com/matlabcentral/fileexchange/2876-draw-a-circle/content/circle.m>

```
clc
clear all;
close all;

%% Atsitiktinių taškų sugeneravimas
% Sugeneruojami atsitiktiniai taškai
p = [round(500*rand(10,1)) round(200*rand(10,1))];

% Nustatomi taškų spiečių centrai
f = [p(2,:); p(3,:); p(4,:); p(5,:); p(6,:); p(7,:)];

% Sugeneruojami šeši taškų spiečiai aplink centrų taškus
for i = 1:20
    s1(i,:) = p(2,:) + round(9*randn(1,2));
    if s1(i,1) < 0
        s1(i,1) = 0;
    end
    if s1(i,2) < 0
        s1(i,2) = 0;
    end
end

for i = 1:20
    s2(i,:) = p(3,:) + round(10*randn(1,2));
    if s2(i,1) < 0
        s2(i,1) = 0;
    end
    if s2(i,2) < 0
        s2(i,2) = 0;
    end
end

for i = 1:20
    s3(i,:) = p(4,:) + round(11*randn(1,2));
    if s3(i,1) < 0
        s3(i,1) = 0;
    end
    if s3(i,2) < 0
        s3(i,2) = 0;
    end
end

for i = 1:20
    s4(i,:) = p(5,:) + round(9*randn(1,2));
    if s4(i,1) < 0
```

```

        s4(i,1) = 0;
    end
    if s4(i,2) < 0
        s4(i,2) = 0;
    end
end

for i = 1:20
    s5(i,:) = p(6,:) + round(11*randn(1,2));
    if s5(i,1) < 0
        s5(i,1) = 0;
    end
    if s5(i,2) < 0
        s5(i,2) = 0;
    end
end

for i = 1:20
    s6(i,:) = p(7,:) + round(12*randn(1,2));
    if s6(i,1) < 0
        s6(i,1) = 0;
    end
    if s6(i,2) < 0
        s6(i,2) = 0;
    end
end

%% Taškų perskaičiavimas
% Keičiant koeficientu reikšmes (j = x:x:x) perskaičiuoti taškus pagal
% traukos lauko, greičio mažinimo ir traukimo pusiaukraštinės kryptimi
% algoritmus
for j = 1:1:9
    s = 0.1*j;
    r = 0.1*j;

    taskai = [s1; s2; s3; s4; s5; s6];
    taskai2 = taskai;
    taskai4 = taskai;
    taskai5 = taskai;

% Perskaičiuoti taškus pirmajam spiečiui
    for i = 2:length(s1)
        if f(1,1)==taskai(i,1) && f(1,2)~=taskai(i,2)
            taskai2(i,2) = round(taskai(i,2) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(1,2)-
taskai(i,2))/abs(f(1,2)-taskai(i,2))));
            taskai4(i,2) = round(taskai(i,2) + s*abs(taskai(i,2)-
((f(1,2)+taskai4(i-1,2))/2))*((f(1,2)-taskai(i,2))/abs(f(1,2)-taskai(i,2))));
        else if f(1,1)~=taskai(i,1) && f(1,2)==taskai(i,2)
            taskai2(i,1) = round(taskai(i,1) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(1,1)-
taskai(i,1))/abs(f(1,1)-taskai(i,1))));
            taskai4(i,1) = round(taskai(i,1) + s*abs(taskai(i,1)-
((f(1,1)+taskai4(i-1,1))./2)).*((f(1,1)-taskai(i,1))./abs(f(1,1)-taskai(i,1))));
        else if f(1,1)~=taskai(i,1) && f(1,2)~=taskai(i,2)
            taskai2(i,:) = round(taskai(i,:) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(1,:) -
taskai(i,:))./abs(f(1,:)-taskai(i,:))));
            taskai4(i,:) = round(taskai(i,:) + s*abs(taskai(i,:)-
((f(1,:)+taskai4(i-1,:))./2)).*((f(1,:)-taskai(i,:))./abs(f(1,:)-taskai(i,:))));
        end
    end
end
end

```

```

end

% Perskaičiuoti taškus antrajam spiečiui
for i = length(s1)+1:length(s1)+length(s2)
    if f(2,1)==taskai(i,1) && f(2,2)~=taskai(i,2)
        taskai2(i,2) = round(taskai(i,2) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(2,2)-
taskai(i,2))/abs(f(2,2)-taskai(i,2))));
        taskai4(i,2) = round(taskai(i,2) + s*abs(taskai(i,2)-
((f(2,2)+taskai4(i-1,2))/2))*((f(2,2)-taskai(i,2))/abs(f(2,2)-taskai(i,2))));
    else if f(2,1)~=taskai(i,1) && f(2,2)==taskai(i,2)
        taskai2(i,1) = round(taskai(i,1) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(2,1)-
taskai(i,1))/abs(f(2,1)-taskai(i,1))));
        taskai4(i,1) = round(taskai(i,1) + s*abs(taskai(i,1)-
((f(2,1)+taskai4(i-1,1))/2))*((f(2,1)-taskai(i,1))/abs(f(2,1)-taskai(i,1))));
    else if f(2,1)~=taskai(i,1) && f(2,2)~=taskai(i,2)
        taskai2(i,:) = round(taskai(i,:) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(2,:) -
taskai(i,:))./abs(f(2,:)-taskai(i,:))));
        taskai4(i,:) = round(taskai(i,:) + s*abs(taskai(i,:)-
((f(2,:)+taskai4(i-1,:))./2)).*((f(2,:)-taskai(i,:))./abs(f(2,:)-taskai(i,:))));
    end
end
end

% Perskaičiuoti taškus trečiajam spiečiui
for i = length(s1)+length(s2)+1:length(s1)+length(s2)+length(s3)
    if f(3,1)==taskai(i,1) && f(3,2)~=taskai(i,2)
        taskai2(i,2) = round(taskai(i,2) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(3,2)-
taskai(i,2))/abs(f(3,2)-taskai(i,2))));
        taskai4(i,2) = round(taskai(i,2) + s*abs(taskai(i,2)-
((f(3,2)+taskai4(i-1,2))/2))*((f(3,2)-taskai(i,2))/abs(f(3,2)-taskai(i,2))));
    else if f(3,1)~=taskai(i,1) && f(3,2)==taskai(i,2)
        taskai2(i,1) = round(taskai(i,1) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(3,1)-
taskai(i,1))/abs(f(3,1)-taskai(i,1))));
        taskai4(i,1) = round(taskai(i,1) + s*abs(taskai(i,1)-
((f(3,1)+taskai4(i-1,1))/2))*((f(3,1)-taskai(i,1))/abs(f(3,1)-taskai(i,1))));
    else if f(3,1)~=taskai(i,1) && f(3,2)~=taskai(i,2)
        taskai2(i,:) = round(taskai(i,:) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(3,:) -
taskai(i,:))./abs(f(3,:)-taskai(i,:))));
        taskai4(i,:) = round(taskai(i,:) + s*abs(taskai(i,:)-
((f(3,:)+taskai4(i-1,:))./2)).*((f(3,:)-taskai(i,:))./abs(f(3,:)-taskai(i,:))));
    end
end
end

% Perskaičiuoti taškus ketvirtajam spiečiui
for i =
length(s1)+length(s2)+length(s3)+1:length(s1)+length(s2)+length(s3)+length(s4)
    if f(4,1)==taskai(i,1) && f(4,2)~=taskai(i,2)
        taskai2(i,2) = round(taskai(i,2) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(4,2)-
taskai(i,2))/abs(f(4,2)-taskai(i,2))));
        taskai4(i,2) = round(taskai(i,2) + s*abs(taskai(i,2)-
((f(4,2)+taskai4(i-1,2))/2))*((f(4,2)-taskai(i,2))/abs(f(4,2)-taskai(i,2))));
    else if f(1,1)~=taskai(i,1) && f(1,2)==taskai(i,2)

```

```

        taskai2(i,1) = round(taskai(i,1) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(4,1)-
taskai(i,1))/abs(f(4,1)-taskai(i,1))));
        taskai4(i,1) = round(taskai(i,1) + s*abs(taskai(i,1)-
((f(4,1)+taskai4(i-1,1))/2))*((f(4,1)-taskai(i,1))/abs(f(4,1)-taskai(i,1))));
        else if f(4,1)~=taskai(i,1) && f(4,2)~=taskai(i,2)
            taskai2(i,:) = round(taskai(i,:) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(4,:)-
taskai(i,:))./abs(f(4,:)-taskai(i,:))));
            taskai4(i,:) = round(taskai(i,:) + s*abs(taskai(i,:)-
((f(4,:)+taskai4(i-1,:))./2))*((f(4,:)-taskai(i,:))./abs(f(4,:)-taskai(i,:))));
        end
    end
end
end

```

% Perskaičiuoti taškus penktajam spiečiui

```

    for i =
length(s1)+length(s2)+length(s3)+length(s4)+1:length(s1)+length(s2)+length(s3)+l
ength(s4)+length(s5)
        if f(5,1)==taskai(i,1) && f(5,2)~=taskai(i,2)
            taskai2(i,2) = round(taskai(i,2) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(5,2)-
taskai(i,2))/abs(f(5,2)-taskai(i,2))));
            taskai4(i,2) = round(taskai(i,2) + s*abs(taskai(i,2)-
((f(5,2)+taskai4(i-1,2))/2))*((f(5,2)-taskai(i,2))/abs(f(5,2)-taskai(i,2))));
            else if f(1,1)~=taskai(i,1) && f(1,2)==taskai(i,2)
                taskai2(i,1) = round(taskai(i,1) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(5,1)-
taskai(i,1))/abs(f(5,1)-taskai(i,1))));
                taskai4(i,1) = round(taskai(i,1) + s*abs(taskai(i,1)-
((f(5,1)+taskai4(i-1,1))/2))*((f(5,1)-taskai(i,1))/abs(f(5,1)-taskai(i,1))));
                else if f(5,1)~=taskai(i,1) && f(5,2)~=taskai(i,2)
                    taskai2(i,:) = round(taskai(i,:) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(5,:)-
taskai(i,:))./abs(f(5,:)-taskai(i,:))));
                    taskai4(i,:) = round(taskai(i,:) + s*abs(taskai(i,:)-
((f(5,:)+taskai4(i-1,:))./2))*((f(5,:)-taskai(i,:))./abs(f(5,:)-taskai(i,:))));
                end
            end
        end
    end
end

```

% Perskaičiuoti taškus šeštajam spiečiui

```

    for i =
length(s1)+length(s2)+length(s3)+length(s4)+length(s5)+1:length(taskai)
        if f(6,1)==taskai(i,1) && f(6,2)~=taskai(i,2)
            taskai2(i,2) = round(taskai(i,2) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(6,2)-
taskai(i,2))/abs(f(6,2)-taskai(i,2))));
            taskai4(i,2) = round(taskai(i,2) + s*abs(taskai(i,2)-
((f(6,2)+taskai4(i-1,2))/2))*((f(6,2)-taskai(i,2))/abs(f(6,2)-taskai(i,2))));
            else if f(1,1)~=taskai(i,1) && f(1,2)==taskai(i,2)
                taskai2(i,1) = round(taskai(i,1) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(6,1)-
taskai(i,1))/abs(f(6,1)-taskai(i,1))));
                taskai4(i,1) = round(taskai(i,1) + s*abs(taskai(i,1)-
((f(6,1)+taskai4(i-1,1))/2))*((f(6,1)-taskai(i,1))/abs(f(6,1)-taskai(i,1))));
                else if f(6,1)~=taskai(i,1) && f(6,2)~=taskai(i,2)
                    taskai2(i,:) = round(taskai(i,:) + s*sqrt((abs(taskai(i,1)-
taskai2(i-1,1)))^2+(abs(taskai(i,2)-taskai2(i-1,2)))^2)*((f(6,:)-
taskai(i,:))./abs(f(6,:)-taskai(i,:))));

```

```

        taskai4(i,:) = round(taskai(i,:) + s*abs(taskai(i,:)-
((f(6,:)+taskai4(i-1,:))./2)).*((f(6,:)-taskai(i,:))./abs(f(6,:)-taskai(i,:))));
    end
end
end

for i = 2:length(taskai5)
    taskai5(i,:) = round((1 - r).*taskai(i,:) + round(r*taskai5(i-1,:)));
end

```

%% Rezultatų pateikimas ekrane

R = 17;

% Traukos lauko algoritmu perskaičiuotų taškų pateikimas
% ekrane

```

figure(j+4*(j-1)-(j-1));
Center = [f(1,1), f(1,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(2,1), f(2,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(3,1), f(3,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(4,1), f(4,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(5,1), f(5,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(6,1), f(6,2)];
circle(Center, R, 1000, 'k-');
hold on
axis([0 500 0 300])
plot(taskai(:,1),taskai(:,2), '+b',
taskai2(:,1),taskai2(:,2), '*r',f(:,1),f(:,2), 'ok')
hold off
xlabel('X')
ylabel('Y')
title('Traukos lauko algoritmas')

```

% Greičio mažinimo algoritmu perskaičiuotų taškų pateikimas

% ekrane

```

figure(j+3*(j-1)+1);
Center = [f(1,1), f(1,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(2,1), f(2,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(3,1), f(3,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(4,1), f(4,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(5,1), f(5,2)];
circle(Center, R, 1000, 'k-');
hold on
Center = [f(6,1), f(6,2)];
circle(Center, R, 1000, 'k-');

```

```

        hold on
        axis([0 500 0 300])
        plot(taskai(:,1),taskai(:,2),
'+b',taskai5(:,1),taskai5(:,2),'*r',f(:,1),f(:,2),'ok')
        hold off
        xlabel('X')
        ylabel('Y')
        title('Greicio mazinimo algoritmas')

% Traukimo pusiaukraštines kryptimi algoritmu perskaičiuotų
% taškų pateikimas ekrane
        figure(j+3*(j-1)+2);
        Center = [f(1,1), f(1,2)];
        circle(Center, R, 1000, 'k-');
        hold on
        Center = [f(2,1), f(2,2)];
        circle(Center, R, 1000, 'k-');
        hold on
        Center = [f(3,1), f(3,2)];
        circle(Center, R, 1000, 'k-');
        hold on
        Center = [f(4,1), f(4,2)];
        circle(Center, R, 1000, 'k-');
        hold on
        Center = [f(5,1), f(5,2)];
        circle(Center, R, 1000, 'k-');
        hold on
        Center = [f(6,1), f(6,2)];
        circle(Center, R, 1000, 'k-');
        hold on
        axis([0 500 0 300])
        plot(taskai(:,1),taskai(:,2),
'+b',taskai4(:,1),taskai4(:,2),'*r',f(:,1),f(:,2),'ok')
        hold off
        xlabel('X')
        ylabel('Y')
        title('Traukimas pusiaukrastines kryptimi')

end

```

2 priedas. Paprogramės, paruoštos algoritmų įdiegimui į taikomąją programą EyeTyping

EyeTyping programos failuose buvo redaguojama *Keyboard.cpp* byloje esanti paprogramė *TiKeyboard::findButtonFromPoint()*. Visi algoritmai buvo išbandyti užprogramuojant juos šios paprogramės ribose.

Programos kodas, realizuojantis pritraukimo algoritmą:

```
SiButton* __fastcall TiKeyboard::findButtonFromPoint(TPoint aPoint)
{
    SiButton* result = NULL;
// Nustatomas pritraukimo laipsnis ir pradinės kintamųjų reikšmės
    float stickLevel = 0.66;
    int stickLeft = iInterButtonGap.cx*stickLevel;
    int stickRight = iInterButtonGap.cx*stickLevel;
    int stickTop = iInterButtonGap.cy*stickLevel;
    int stickBottom = iInterButtonGap.cy*stickLevel;
    int n = 0;
    int m = 0;
    int midX, midY;
    SiButton* btn = iButtons->get(0);
    int right = btn->Viewport.GetLeft()-iInterButtonGap.cx/2;
    int bottom = btn->Viewport.GetTop()-iInterButtonGap.cy/2;
    bool btnX = 0;
    bool btnY = 0;

// Perskaičiuoti tik tašką, esantį klaviatūros ribose
    if (aPoint.x>right && aPoint.y>bottom && aPoint.x<right+10*(btn-
>Size.cx+iInterButtonGap.cx)
        && aPoint.y<bottom+3*(btn->Size.cy+iInterButtonGap.cy)){

//-----
// --- PADETIES NUSTATYMAS ---
//-----

// Nustatoma taško padėtis X ašyje

        for (n = 0; n < 10; n++){
            right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
            midX = right - btn->Size.cx/2;
            if ((right - btn->Size.cx - aPoint.x)< iInterButtonGap.cx && (right - btn-
>Size.cx - aPoint.x) > 0){
                btnX=0;
                break;
            }
            else if ((right - aPoint.x)<= btn->Size.cx && (right - aPoint.x)>=0){
                btnX=1;
                break;
            }
            else{
                btnX=0;
            }
        }

// Jeigu taškas dešiniojoje klaviatūros dalyje
        if(n==10){
            right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
            midX = right - btn->Size.cx/2;
            n=9;
        }

// Nustatoma taško padėtis Y ašyje
```

```

        for (m = 0; m < 3; m++){
            bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
            midY = bottom - btn->Size.cy/2;
            if ((bottom - btn->Size.cy - aPoint.y) < iInterButtonGap.cy && (bottom - btn-
>Size.cy - aPoint.y) > 0){
                btnY=0;
                break;
            }
            else if ((bottom - aPoint.y) <= btn->Size.cy && (bottom - aPoint.y) >= 0){
                btnY=1;
                break;
            }
            else{
                btnY=0;
            }
        }

// Jeigu taškas apatinėje klaviatūros dalyje
if(m==3){
    bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
    midY = bottom - btn->Size.cy/2;
    m=2;
}

//-----
//Praplėsti aktyviąją zoną aplink nustatytą mygtuką

SiButton* btn = iButtons->get(10*m+n);
TRect rc = TRect(btn->Viewport.GetLeft()-stickLeft, btn->Viewport.GetTop()-
stickTop,
                btn->Viewport.GetRight()+stickRight, btn-
>Viewport.GetBottom()+stickBottom);
if(::PtInRect(&rc, aPoint)){
    result = btn;
}
}
return result;
}

```

Programos kodas, realizuojantis traukos lauko algoritmą:

```

SiButton* __fastcall TiKeyboard::findButtonFromPoint(TPoint aPoint)
{
    SiButton* result = NULL;
    int n = 0;
    int m = 0;
    int midX, midY;
    SiButton* btn = iButtons->get(0);
    int right = btn->Viewport.GetLeft()-iInterButtonGap.cx/2;
    int bottom = btn->Viewport.GetTop()-iInterButtonGap.cy/2;
    bool btnX = 0;
    bool btnY = 0;
    float EuDist = 0;

    // Perskaičiuoti tik tašką, esantį klaviatūros ribose
    if (aPoint.x>right && aPoint.y>bottom && aPoint.x<right+12*(btn-
>Size.cx+iInterButtonGap.cx)
        && aPoint.y<bottom+5*(btn->Size.cy+iInterButtonGap.cy)){

//-----
// --- PADETIES NUSTATYMAS ----
//-----

// Nustatoma taško padėtis X ašyje

        for (n = 0; n < 10; n++){
            right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);

```

```

        midX = right - btn->Size.cx/2;
        if ((right - btn->Size.cx - aPoint.x) < iInterButtonGap.cx && (right - btn-
>Size.cx - aPoint.x) > 0){
            btnX=0;
            break;
        }
        else if ((right - aPoint.x) <= btn->Size.cx && (right - aPoint.x) >= 0){
            btnX=1;
            break;
        }
        else{
            btnX=0;
        }
    }

// Jeigu taškas dešiniojoje klaviatūros dalyje
    if(n==10){
        right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
        midX = right - btn->Size.cx/2;
        n=9;
    }

// Nustatoma taško padėtis Y ašyje

    for (m = 0; m < 3; m++){
        bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
        midY = bottom - btn->Size.cy/2;
        if ((bottom - btn->Size.cy - aPoint.y) < iInterButtonGap.cy && (bottom - btn-
>Size.cy - aPoint.y) > 0){
            btnY=0;
            break;
        }
        else if ((bottom - aPoint.y) <= btn->Size.cy && (bottom - aPoint.y) >= 0){
            btnY=1;
            break;
        }
        else{
            btnY=0;
        }
    }

// Jeigu taškas apatinėje klaviatūros dalyje
    if(m==3){
        bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
        midY = bottom - btn->Size.cy/2;
        m=2;
    }

//-----

// Taško perskaičiavimas pagal traukos lauko algoritmą
    float s = 0.3; //Traukos lauko stiprio koeficientas
    EuDist = sqrt(abs(aPoint.x-prevX)^2+abs(aPoint.Y-prevY)^2);
    if (midX==aPoint.x && midY!=aPoint.y){
        aPoint.y = (int)(aPoint.y + s*EuDist*((midY-aPoint.y)/(abs(midY-aPoint.y))));
    }
    else if (midX!=aPoint.x && midY==aPoint.y){
        aPoint.x = (int)(aPoint.x + s*EuDist*((midX-aPoint.x)/(abs(midX-aPoint.x))));
    }
    else if (midX!=aPoint.x && midY!=aPoint.y){
        aPoint.y = (int)(aPoint.y + s*EuDist*((midY-aPoint.y)/(abs(midY-aPoint.y))));
        aPoint.x = (int)(aPoint.x + s*EuDist*((midX-aPoint.x)/(abs(midX-aPoint.x))));
    }
    prevX = aPoint.x;
    prevY = aPoint.y;

//-----
// Kreiptis į rastą mygtuką

```

```

        SiButton* btn = iButtons->get(10*m+n);
        TRect rc = TRect(btn->Viewport.GetLeft(), btn->Viewport.GetTop(),
            btn->Viewport.GetRight(), btn->Viewport.GetBottom());
        if (::PtInRect(&rc, aPoint)){
            result = btn;
        }
    }
    return result;
}

```

Programinis kodas, realizuojantis greičio mažinimo algoritmą:

```

SiButton* __fastcall TKeyboard::findButtonFromPoint(TPoint aPoint)
{
    SiButton* result = NULL;
    int n = 0;
    int m = 0;
    int midX, midY;
    SiButton* btn = iButtons->get(0);
    int right = btn->Viewport.GetLeft()-iInterButtonGap.cx/2;
    int bottom = btn->Viewport.GetTop()-iInterButtonGap.cy/2;
    bool btnX = 0;
    bool btnY = 0;
    float EuDist = 0;

    // Perskaičiuoti tik tašką, esantį klaviatūros ribose
    if (aPoint.x>right && aPoint.y>bottom && aPoint.x<right+12*(btn->
        Size.cx+iInterButtonGap.cx)
        && aPoint.y<bottom+5*(btn->Size.cy+iInterButtonGap.cy)){

//-----
// --- PADETIES NUSTATYMAS ----
//-----

// Nustatoma taško padėtis X ašyje

        for (n = 0; n < 10; n++){
            right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
            midX = right - btn->Size.cx/2;
            if ((right - btn->Size.cx - aPoint.x)< iInterButtonGap.cx && (right - btn->
Size.cx - aPoint.x) > 0){
                btnX=0;
                break;
            }
            else if ((right - aPoint.x)<= btn->Size.cx && (right - aPoint.x)>=0){
                btnX=1;
                break;
            }
            else{
                btnX=0;
            }
        }

// Jeigu taškas dešiniojoje klaviatūros dalyje
        if(n==10){
            right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
            midX = right - btn->Size.cx/2;
            n=9;
        }

// Nustatoma taško padėtis Y ašyje

        for (m = 0; m < 3; m++){
            bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
            midY = bottom - btn->Size.cy/2;

```

```

        if ((bottom - btn->Size.cy - aPoint.y) < iInterButtonGap.cy && (bottom - btn->Size.cy - aPoint.y) > 0){
            btnY=0;
            break;
        }
        else if ((bottom - aPoint.y) <= btn->Size.cy && (bottom - aPoint.y) >= 0){
            btnY=1;
            break;
        }
        else{
            btnY=0;
        }
    }

// Jeigu taškas apatinėje klaviatūros dalyje
    if(m==3){
        bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy + iInterButtonGap.cy);
        midY = bottom - btn->Size.cy/2;
        m=2;
    }

//-----
// Perskaiciuoti taška pagal greicio mažinimo algoritma

    float r = 0.9;
    aPoint.x = (int)((1 - r) * aPoint.x + r * prevX);
    aPoint.y = (int)((1 - r) * aPoint.y + r * prevY);
    prevX = aPoint.x;
    prevY = aPoint.y;

//-----
// Kreiptis i rastą mygtuką

    SiButton* btn = iButtons->get(10*m+n);
    TRect rc = TRect(btn->Viewport.GetLeft(), btn->Viewport.GetTop(),
        btn->Viewport.GetRight(), btn->Viewport.GetBottom());
    if (::PtInRect(&rc, aPoint)){
        result = btn;
    }
}
return result;
}

```

Programinis kodas, realizuojantis traukimo pusiaukraštinės kryptimi algoritmą:

```

SiButton* __fastcall TiKeyboard::findButtonFromPoint(TPoint aPoint)
{
    SiButton* result = NULL;
    int n = 0;
    int m = 0;
    int midX, midY;
    SiButton* btn = iButtons->get(0);
    int right = btn->Viewport.GetLeft()-iInterButtonGap.cx/2;
    int bottom = btn->Viewport.GetTop()-iInterButtonGap.cy/2;
    bool btnX = 0;
    bool btnY = 0;
    float EuDist = 0;

    // Perskaiciuoti tik tašką, esantį klaviatūros ribose
    if (aPoint.x>right && aPoint.y>bottom && aPoint.x<right+12*(btn->Size.cx+iInterButtonGap.cx)
        && aPoint.y<bottom+5*(btn->Size.cy+iInterButtonGap.cy)){

//-----
// --- PADETIES NUSTATYMAS ---
//-----

// Nustatoma taško padetis X ašyje

```

```

        for (n = 0; n < 10; n++){
            right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
            midX = right - btn->Size.cx/2;
            if ((right - btn->Size.cx - aPoint.x)< iInterButtonGap.cx && (right - btn-
>Size.cx - aPoint.x) > 0){
                btnX=0;
                break;
            }
            else if ((right - aPoint.x)<= btn->Size.cx && (right - aPoint.x)>=0){
                btnX=1;
                break;
            }
            else{
                btnX=0;
            }
        }

// Jeigu taškas dešiniojoje klaviatūros dalyje
if(n==10){
    right = btn->Viewport.GetLeft() + btn->Size.cx + n * (btn->Size.cx +
iInterButtonGap.cx);
    midX = right - btn->Size.cx/2;
    n=9;
}

// Nustatoma taško padėtis Y ašyje

        for (m = 0; m < 3; m++){
            bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
            midY = bottom - btn->Size.cy/2;
            if ((bottom - btn->Size.cy - aPoint.y)< iInterButtonGap.cy && (bottom - btn-
>Size.cy - aPoint.y) > 0){
                btnY=0;
                break;
            }
            else if ((bottom - aPoint.y) <= btn->Size.cy && (bottom - aPoint.y) >= 0){
                btnY=1;
                break;
            }
            else{
                btnY=0;
            }
        }

// Jeigu taškas apatinėje klaviatūros dalyje
if(m==3){
    bottom = btn->Viewport.GetTop() + btn->Size.cy + m * (btn->Size.cy +
iInterButtonGap.cy);
    midY = bottom - btn->Size.cy/2;
    m=2;
}

//-----

// Perskaiciuoti taška pagal traukimo pusiaukraštines kryptimi algoritma
float s = 0.3;
if (midX==aPoint.x && midY!=aPoint.y){
    aPoint.y = (int)(aPoint.y + s*abs(aPoint.y-((prevY+midY)/2))*((midY-
aPoint.y)/(abs(midY-aPoint.y))));
}
else if (midX!=aPoint.x && midY==aPoint.y){
    aPoint.x = (int)(aPoint.x + s*abs(aPoint.x-((prevX+midX)/2))*((midX-
aPoint.x)/(abs(midX-aPoint.x))));
}
else if (midX!=aPoint.x && midY!=aPoint.y){
    aPoint.y = (int)(aPoint.y + s*abs(aPoint.y-((prevY+midY)/2))*((midY-
aPoint.y)/(abs(midY-aPoint.y))));

```

```

        aPoint.x = (int)(aPoint.x + s*abs(aPoint.x-((prevX+midX)/2))*((midX-
aPoint.x)/(abs(midX-aPoint.x))));
    }
    prevX = aPoint.x;
    prevY = aPoint.y;

```

```

//-----

```

```

// Kreiptis i rastą mygtuką

```

```

        SiButton* btn = iButtons->get(10*m+n);
        TRect rc = TRect(btn->Viewport.GetLeft(), btn->Viewport.GetTop(),
            btn->Viewport.GetRight(), btn->Viewport.GetBottom());
        if(::PtInRect(&rc, aPoint)){
            result = btn;
        }
    }
    return result;
}

```

3 priedas. Pranešimo 16-oje Lietuvos jaunųjų mokslininkų konferencijoje pažymos kopija



VILNIAUS GEDIMINO
TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS

XVI-toji Lietuvos jaunųjų mokslininkų konferencija
„MOKSLAS – LIETUVOS ATEITIS“
ELEKTRONIKA IR ELEKTROTECHNIKA

PAŽYMA
Andrius Stonys

Vaizdų technologijos sekcijoje perskaitė pranešimą
Pritraukimo algoritmo tyrimas žvilgsniu valdomoje klaviatūroje

Mokslo komiteto pirmininkas

Sekcijos pirmininkas

prof. habil. dr. Romanas Martavičius

doc. dr. Darius Miniotas

2013 m. kovo 15 d.
Vilnius