KAUNO TECHNOLOGIJOS UNIVERSITETAS
ELEKTROS IR ELEKTRONIKOS FAKULTETAS

Karolis Šakurskis

MOBILAUS AUTONOMINIO ROBOTO PROGRAMOS TIKSLO TAŠKUI PASIEKTISUKŪRIMAS

Baigiamasis magistro projektas

Vadovas
Lekt. Gintautas Narvydas

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS
ELEKTROS IR ELEKTRONIKOS FAKULTETAS
AUTOMATIKOSKATEDRA

MOBILAUS AUTONOMINIO ROBOTO PROGRAMOS TIKSLO TAŠKUI PASIEKTI SUKŪRIMAS

Baigiamasis magistro projektas
VALDYMO TECHNOLOGIJOS (621H66001)

Vadovas
Lekt. Gintautas Narvydas
2015.05.25

Recenzentas
(parašas)
(data)

Projektą atliko
Karolis Šakurskis
2015.05.25

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Elektros ir Elektronikos Fakultetas
(Fakultetas)
Karolis Šakurskis
(Studento vardas,pavardė)
Valdymo technologijos621H66001
(Studijų programos pavadinimas, kodas)

„Mobilaus autonominio roboto programos tikslo taškui pasiekti sukūrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 15 m. Gegužės 25 d.
Kaunas

Patvirtinu, kad mano **Karolio Šakurskio** baigiamasis projektas tema „Mobilaus autonominio roboto programos tikslo taškui pasiekti sukūrimas“ yra parašytas visiškai savarankiškai, o visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai.Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Šakurskis, K. Mobilaus autonominio roboto programos tikslo taškui pasiekti sukūrimas. Magistrobaigiamasis projektas / vadovas lekt. Gintautas Narvydas; Kauno technologijos universitetas, Elektros ir Elektronikos fakultetas, Automatikos katedra.

Kaunas, 2015. 54 psl.

SANTRAUKA

Mobilūs autonominiai robotai šiais laikais gali pasitarnauti daugelyje sričių. Jie naudojami gamyboje, sandėliuose, o sudėtingesni ir kitų planetų paviršiams tirti bei duomenims surinkti. Mobilūs autonominiai robotai judėjimui ir orientavimuisi tarp kliūčių bei tikslo taškui pasiekti naudoja algoritmus. Kuriami nauji, konkrečioms užduotims pritaikomi algoritmai dažniausiai naudoja klasikinius algoritmus, kaip pagrindą, kuris vėliau modifikuojamas.

Dėl šios priežasties magistro baigiamojo projekto tikslas yra sukurti programą, kuri padėtų mobiliam autonominiam robotui išvengti kliūčių ir pasiekti užduotą tikslo tašką statiškoje aplinkoje. Atlikus algoritmų, naudojamų trajektorijos planavimui, literatūros analizę,pasirenkamidu metodai, kurie galėtų geriausiai rasti reikiamą trajektoriją. Pasirinkti algoritmai suprogramuojami ir palyginamasjų sugebėjimas rasti trajektoriją, vidutinis rastos trajektorijos ilgis bei rastos trajektorijos posūkių skaičius. Atlikus palyginimą,pateikiama rekomendacija algoritmo naudojimui. Geresnis algoritmas pasirenkamas roboto programavimui.

Sukurta programa mobiliam autonominiam Arduino robotui valdyti. Robotui 10 kartų važiuojant ta pačia trajektorija atliekamas eksperimentas, kurio metu matuojamas trajektorijos įvykdymo laikas, bei veiksmų skaičius.Išmatuojamas duomenų perdavimo dažnis, bei aprašomi faktoriai, lėmę eksperimento rezultatus.

Reikšminiai žodžiai: Mobilus autonominis robotas, A, potencialų laukai, algoritmas, optimali trajektorija.*

Šakurskis, Karolis.Development of a Program for mobile autonomous robot to reach a target point. Masters final project / supervisorlekt. Gintautas Narvydas; Kaunas University ofTechnology, FacultyofElectricaland Electronics Engineering, departmentof Automatics.

Kaunas, 2015. 54 p.

SUMMARY

Nowadays mobile autonomous robots can be employed in many different fields. Most of them are used in production or warehouses, and those more sophisticated can even be used for research of terrain on other planets. For motion, orientation among obstacles and reaching a target point, mobile autonomous robots use algorithms. New algorithms that are being created for a various specific tasks usually use classical algorithms as a base, which is later modified.

Because of these reasons, aim of the masters final project is to create a program for a mobile autonomous robot, to avoid obstacles and reach a target point in static environment. After analysis of the algorithms used for optimal path planning was conducted, two best algorithms were chosen for a further analysis. Two chosen algorithms were programmed, and compared against each other for the ability to find path, average length of the path, and the number of turns in the path. After this test, recommendation for the use of better algorithm is provided. Best algorithm is chosen for a robot to be programmed.

Program for the control of the mobile autonomous Arduino robot was created. Experiment for the robot, driving the same path 10 times was conducted. Robot was measured against time and steps needed to complete the path. Frequency of communication was calculated. Factors that had influence on the eksperiment results were presented.

Keywords: mobile autonomous robot, A, potential fields, algorithm, optimal path.*

TURINYS

ĮVADAS

Šiais laikais, tobulėjant technologijoms, daugelį procesų stengiamasi automatizuoti. Dažnai tai daroma dėl to, kad robotai padeda procesus patobulinti - sudėtinga gamyba atliekama greičiau, kokybiškiau - efektyviau. Kita procesų automatizavimo priežastis - žmogaus nesugebėjimas to darbo atlikti dėl vienokių ar kitokių priežasčių. Vienas procesų automatizavimo įrankių - mobilieji autonominiai robotai. Jie gali pasitarnauti dėl abiejų priežasčių - padidina darbo efektyvumą, taip pat gali atlikti darbus terpėse, pavojingose žmogaus sveikatai.

Mobilieji autonominiai robotai jau dabar yra naudojami įvairiose srityse. Paprastesni - sandėliuojant bei markiruojant detales sandėliuose, dažniausiai dirbantys statiškoje aplinkoje, sudėtingesni - gamyklos detalių logistikos sistemose, dirbantys iš dalies dinamiškoje terpėje. Itin sudėtingi robotai naudojami kitų planetų paviršiaus tyrimams, dirba ant nelygaus nežinomo reljefo. Tačiau pramonėje naudojamų autonominių mobilių robotų galimybių ribos dar tikrai nėra pasiektos. Tai padaryti bandoma kuriant skirtingus optimalios trajektorijos skaičiavimo algoritmus, kurie leistų robotui laisvai važiuotistatiškose ir dinamiškose aplinkose, bei judėti optimalia trajektorija.

Dėl šių priežasčių magistro baigiamojo projekto tikslas - sukurti programą mobiliam autonominiam robotui, kuri padėtų robotuipasiekti užduotą tikslo taškąstatiškoje aplinkoje ir išvengtikliūčių. Tikslui pasiekti išsikeliami uždaviniai:

1. Išanalizuoti egzistuojančius algoritmus, naudojamus optimalios trajektorijos planavimui;
2. Pasirinkti du algoritmus trumpiausio kelio paieškai;
3. Ištirti kuris algoritmas yra tinkamesnis roboto programavimui;
4. Mobiliam autonominiam robotui sukurti programą tikslo taškui pasiekti pagal parinktą algoritmą.

1 PROBLEMOS ANALIZĖ

1.1 Roboto navigacija

Kiekvienam įrenginiui, kurio paskirtis yra nusigauti nuo taško A į tašką B, svarbu naviguoti erdvėje, kurioje judama. Optimalios trajektorijos planavimas apima ne tik fizinių kliūčių išvengimą, bet ir nesaugių terpių, ar nepageidaujamų sąlygų apėjimą. Kad tai padarytų, robotas turi visuomet žinoti savo padėtį erdvėje naudojamos atskaitos sistemos atžvilgiu.

Atliekant užduotį, svarbu žinoti, kokiu tikslumu robotas turi naviguoti erdvėje. Žinoma, šie reikalavimai labiausiai priklauso nuo paties roboto pritaikymo ar konkrečios atliekamos užduoties, tačiau pirminis įvertinimas gali būti toks: norėdamas nustatyti savo padėtį erdvėje, robotas turėtų naviguoti bent jau savo paties matmenų tikslumu. Pagal mastelį ir atskaitos sistemą, navigacija gali būti skirstoma naudojant šiuos tris terminus:

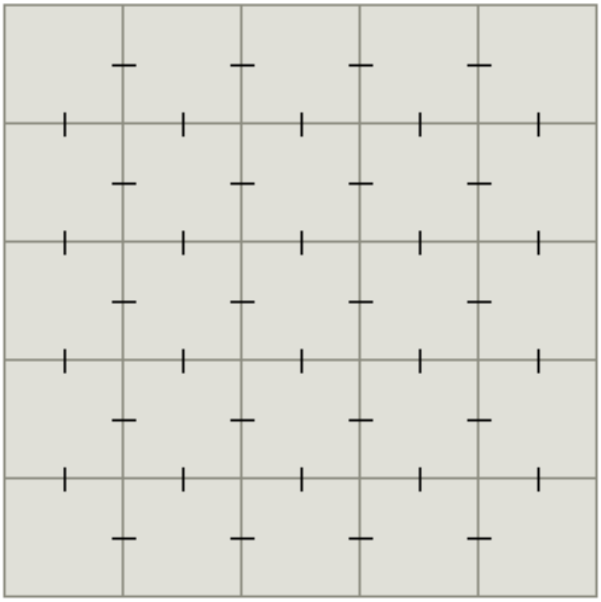
a) *Globali navigacija* - tai gebėjimas nustatyti padėtį absoliutinėje (žemėlapio) atskaitos sistemoje, ir pasiekti pageidautiną tikslą joje;
b) *Lokali navigacija* - tai gėbėjimas nustatyti savo padėtį supančių objektų atžvilgiu ir su jais sąveikauti;
c) *Personalinė navigacija* - tai žinojimas kur ir kokioje padėtyje viena kitos atžvilgiu yra dalys, sudarančios patį robotą [1].

Šių skirtingų mastelių tikslai taip pat yra skirtingi. Globali navigacija skirta judėjimui tarp galinių lokacijų, kai tuo metu lokali navigacija koncentruojasi į užduočių atlikimą tose lokacijose. Personalinė navigacija naudojama roboto ir su juo kontaktuojančių objektų monitoringui [1].

1.2 A algoritmas*

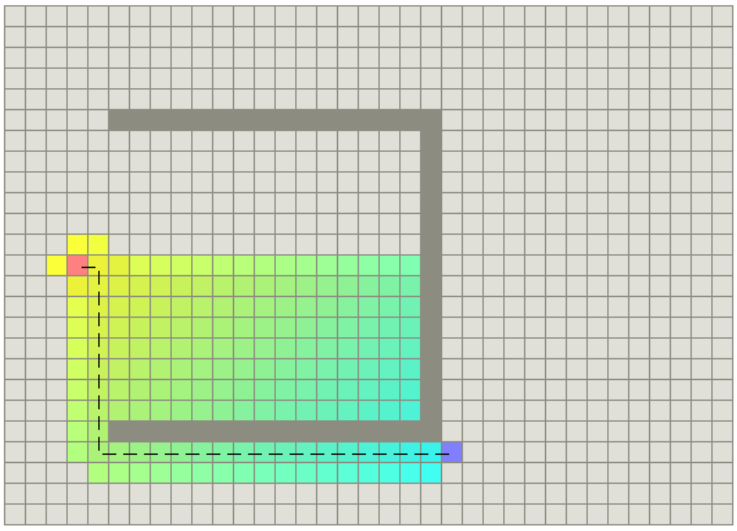
A* algoritmas yra vienas populiariausių optimalios trajektorijos radimui. Jis yra ganėtinai lankstus ir pritaikomas daugelyje sričių. Šis algoritmas dažnai naudojamas žaidimų programuotojų ir nesunkiai paaiškinamas šiame kontekste.

Sukuriamas žaidimo žemėlapis, kuris veikia kaip grafikas, kuriame plotas pažymėtas kvadratėliais, o juos tarpusavyje jungia kraštinės (pav. 1.1). Žinoma, gali būti naudojami ir kitokie metodai žemėlapio įvertinimui, penkiakampiai, trikampiai mazgai, ir t.t. [2]



1.1 pav. Žemėlapio grafikas. [2]

A* algoritmas, įvertina kiekvienos trajektorijos tikslią vertę (*exactcost*) nuo pradinio langelio, iki bet kurio langelio n. Tokiu atveju turime funkciją *g(n)*, kuri žymi visas šias reikšmes. Tuo tarpu funkcija *h(n)* žymi euristines apskaičiuotas vertes (*estimated cost*) nuo bet kurio langelio *n* iki galutinio tikslo. 1.2 paveiksle langeliai pažymėti geltonai vaizduoja toli nuo tikslo esančius langelius, o žydri - artimus. Taigi algoritmas A* kombinuoja abi vertes objektui judant tikslo link. Kiekvieno ciklo metu algoritmas tikrina lauką *n*, ir randama bei pasirenkama tokia tolesnė trajektorija, kurios $f(n) = g(n) + h(n)$ yra mažiausias [2].



1.2pav. Trajektorijos radimas A* algoritmu [2].

Panagrinėkime šį algoritmą nuodugniau. Pradedant paiešką, pirmiausia, kaip jau buvo galima pastebėti aukščiau pateiktuose paveiksluose, paieškos laukas supaprastinamas jį suskaidžius į kvadratinius laukelius, vadinamus mazgais. Mazgas iš esmės gali būti ir penkiakampis, šešiakampis ar bet kokia kita figūra, tačiau tuomet trajektorijos paieška tampa sudėtingesnė. Paieška pradedama atliekant sekančius žingsnius:

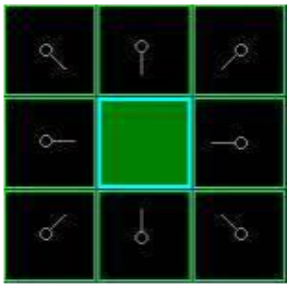
a) Pradedame mazgu A ir įtraukiame jį į atvirą sąrašą (*open list*) mazgų. Šis sąrašas sudarytas iš mazgų, kuriuos potencialiai gali tekti aplankyti, keliaujant optimalia trajektorija, todėl juos reikia patikrinti.

b) Patikrinami visi pasiekiami mazgai, kurie yra šalia pradinės pozicijos (mazgas A). ignoruojami langeliai su kliūtimis. Visi langeliai kurie gali būti pasiekiami, taip pat įrašomi į atvirą sąrašą, o mazgas A išsaugomas kaip visų šių langelių pirminis mazgas (*parent node*). Ši hierarchija yra svarbi, norint atsekti būsimą trajektoriją.

c) Pradinis mazgas A išmetamas iš atvirojo sąrašo ir įrašomas į uždarąjį sąrašą (*closed list*), tai sąrašas mazgų, kurių tikrinti nebereikia.

Tokiu būdu, gauname tokį žemėlapio vaizdą, kaip pateikta 1.3 paveiksle. Centrinis mazgas yra pradinis - A. Jis yra įrašytas į uždarą sąrašą. Visi gretimi, galimi aplankyti, langeliai yra įrašyti į atvirą sąrašą ir rodyklėmis sujungti su savo pirminiu mazgu.

Toliau pasirenkamas vienas iš gretimų langelių ir kartojamas tas pats procesas, kaip ką tik aprašytas. Iš visų langelių atvirame sąraše, pasirenkamas tas, kurio mažiausia $f(n)=g(n)+h(n)$ funkcijos vertė.



1.3 pav. A* algoritmo taikymas [3].

Jeigu kiekvienam perėjimui langeliu horizontaliai ar vertikaliai suteiksime vertę 10, tai pajudėjimas langeliu įstrižai turėtų turėti vertę 14 (dauginama iš $\sqrt{2}$). Taigi, skaičiuojant *g(n)* vertę iki specifinio langelio, reikia paimti priminio mazgo*g(n)* vertę, ir prie jos pirdėti 10 arba 14, priklausomai nuo to, ar judesys bus įstrižas ar ortogonalus nuo pirminio mazgo [3].

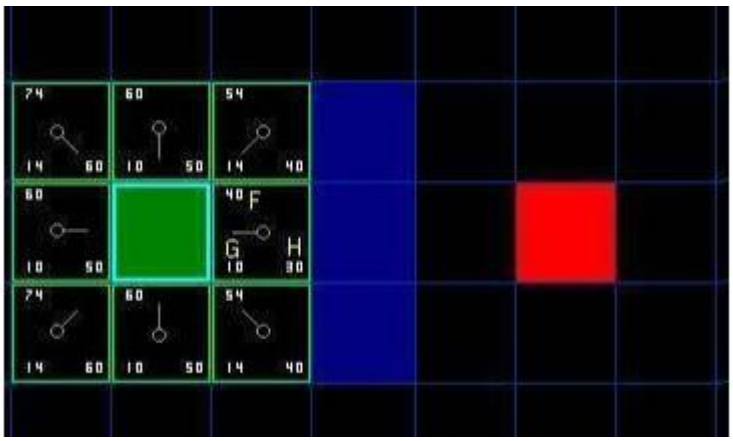
Kita funkcija - *h(n)*- gali būti įvertinta daugeliu metodų, vienas iš jų - Manhatano (*Manhattan)* metodas. Naudojant šį metodą yra suskaičiuojamas bendras langelių skaičius iki kliūties horizontaliai ir vertikaliai, ignoruojant visas kliūtisbei galimą judėjimą įstrižai. Tuomet gautas skaičius yra padauginamas iš 10. H įverčio skaičiavimui naudojama formulė (1). Toks skaičiavimas nevisuomet užtikrina mažiausią *h(n)* įvertinimą, tačiau pagreitina algoritmą. Kai h(n) reikšmė negarantuoja trumpiausio kelio, ji vadinama nepriimtina euristika (*inadmissible heuristics*) [3].

$$H = 10*(|dabartinisX-tiksloX| + |dabartinisY-tiksloY|) \qquad (1)$$

Taip pat *h(n)* įvertinimui naudojamas Kampo kirtimo (*Diagonal shortcut*) metodas. Naudojant šį metodą, įskaičiuojamas judėjimas įstrižai, todėl atstumas įvertinamas tiksliau. Šio metodo minusas tas, kad jis veikia lėčiau lyginant su Manhatano metodu. Šiam metodui naudojamas pseudokodas:

xAtstumas = |dabartinisX-tiksloX|
yAtstumas = |dabartinisY-tiksloY|
if xAtstumas> yAtstumas
*H = 14*yAtstumas + 10*(xAtstumas-yAtstumas)*
else
*H = 14*xAtstumas + 10*(yAtstumas-xAtstumas)*
end if [3]

Taigi, *f(n)* yra suskaičiuojamas sumuojant *g(n)* ir *h(n)* funkcijas. 1.4 paveiksle galima pamatyti pirmojo žingsnio skaičiavimą, funkcijų vertės yra parašytos kiekviename langelyje.



1.4 pav. A* algoritmo pirmojo žingsnio skaičiavimas [3].

Tęsiant paiešką, pasirenkamas mazgas, su mažiausiu F įvertinimu iš atviro sąrašo ir atliekami veiksmai sekančia tvarka:

a) Pasirinktas mazgas iš atvirojo sąrašo perkeliamas į uždarąjį sąrašą;

b) Patikrinami visi greta esantys mazgai, išskyrus tuos, kurie yra uždarame sąraše, arba kliūtys, į kurias negalima patekti. Jei kažkurie iš greta esančių, galimų užimti langelių nėra atvirame sąraše, juos reikia ten perkelti. Dabartinis pasirinktas mazgas turi tapti pirminiu visiems kitiems galimiems užimti langeliams.

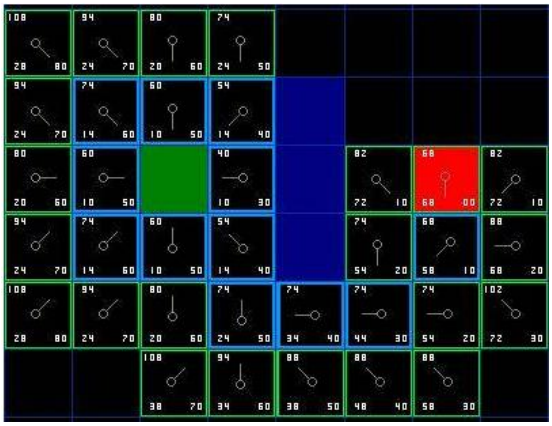
c) Jeigu greta esantis langelis jau yra atvirame sąraše, reikia patikrinti, ar kelias iki jo yra geresnis (ar G mažesnis, naudojant dabartinį langelį). Jei kelias prastesnis, nereikia imtis jokių veiksmų. Jei kelias yra geresnis, tai dabartinis užimtas langelis, padaromas tikrinamo langelio pirminiu mazgu. Tai padarius, reikia perskaičiuoti to langelio *f(n)* ir *g(n)* reikšmes[3].

Vėl tikrinami visi mazgai, esantys atvirame sąraše, kuriame dabar yra septyni langeliai. Iš šių septynių išsirenkamas vienas su mažiausia *f(n)* verte. Šiuo atveju yra du langeliai kurių vertė yra mažiausia. Nėra esminio skirtumo kuris bus pasirinktas, dažniausiai tokiu atveju pasirenkamas langelis, kuris buvo paskutinis pridėtas prie atviro sąrašo. Taip teikiama primenybė langeliams, kurie yra randami vėlesnėje ieškojimo stadijoje, kai artėjama prie taikinio. Iš tikrųjų skirtumo nėra, todėl pasirenkant kitą langelį su tokiu pačiu funkcijos *f(n)* įverčiu, A* algoritmu galima surasti kelias skirtingas trajektorijas su tokiu pačiu atstumo įvertinimu. Šiuo atveju pasirinktas langelis, ir tolesnė aprašyta skaičiavimo eiga matoma 1.5 paveiksle[3].

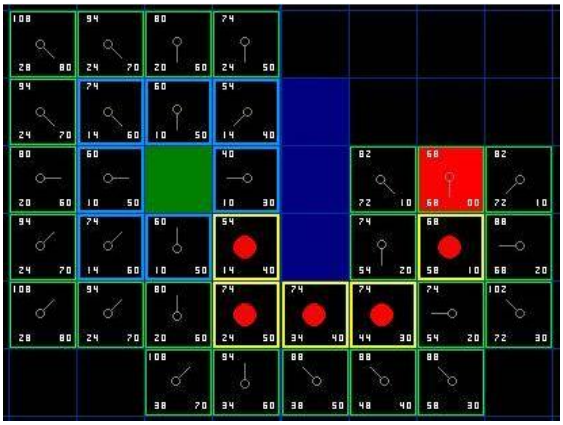
1.5 pav. Tolimesnis skaičiavimas A* algoritmu [3].

Šį kartą, tikrinant naujus, gretimus mazgus, ignoruojami ne tik kliūtį vaizduojantys mazgai, tačiau ir langelis einantis įstrižai, iškart po kliūtimi - norint jį pereiti įstrižai, dalis judančio objekto visvien kirstų kliūtį, todėl šią reikia apeiti pilnai.

Taigi, šis procesas kartojamastol, kol tikslo mazgas yra pridedamas prie uždarojo sąrašo (pav. 1.6). Trajektorija nustatoma pradedant tikslo mazgu. Nuo tikslo mazgo, judama link jo pirminio mazgo, o nuo šio langelio, iki jo pirminio mazgo, sekant pavaizduotomis rodyklėmis. Galiausiai atsiduriama pradiniame langelyje ir taip gaunama visa trajektorija (pav 1.7).



1.6 pav. Uždarojo sąrašo mazgai [3].



1.7 pav. Optimali trajektorija[3].

Naudojant A* algoritmą, nemažai pasiekta robotui naviguojant iš anksto žinomoje aplinkoje. Aplinkoje atsiradus judančioms arba iš anksto nežinomų matmenų kliūtims, šis algoritmas tampa mažiau efektyvus. Tokiu atveju dažniau naudojami dirbtinio potencialų lauko, arba neraiškios logikos metodai [4].

A* algoritmas yra naudojamas daugelyje sričių, dažniausiai kaip bazinis algoritmas, kuris yra redaguojamas taip, kad galėtų atlikti konkrečią užduotį. Viena iš tokių sričių yra GPS navigacijos sistemos. Modifikuotas A* arba Dijkstra algoritmas padeda ieškoti kelių, taikant skirtingus paieškos kriterijus, kaip greičiausias kelias, trumpiausias kelias, trumpiausias kelias važiuojant asfaltuotais keliais ir t.t. Skaičiavimams dažnai naudojami ir kitokie euristiniai

įverčiai. Pvz.: GPS sistemose galima keisti euristinių įverčių vertes įvertinant maksimalų greitį kuriuo galima važiuot šiame kelyje, arba tik pačio kelio ilgį. Neasfaltuotą kelią galima traktuoti kaip kliūtį ir jų išvengti.

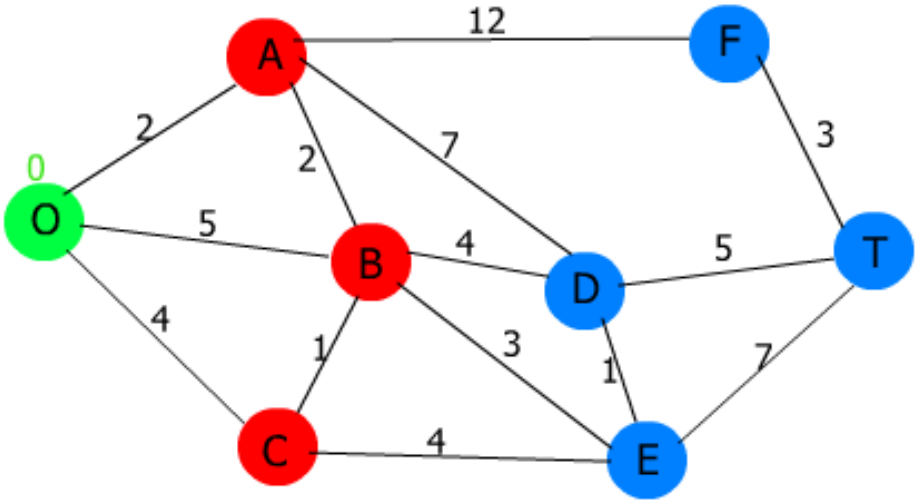
Labai panašus algoritmas yra naudojamas TEMPEST navigacijos sistemoje, kuri sukurta NASA. Tai ilgo nuotolio navigacijos sistema naudojama kitų planetų paviršiams tirti. Viena iš šios sistemos dalių - ISE (*incremental search engine*), kuri naudojama globaliai ir lokaliai navigacijai. ISE sistemos pradinės paieškos rezultatai būna identiški, kaip naudojant A* algoritmą, kur neįvertinami roboto holonominiai apribojimai. Tik vėliau, perskaičiuojant ir tikslinant duomenis ISE tampa efektyvesnis už A* [5].

1.3 Dijkstra algoritmas

Dijkstra algoritmas, panašiai kaip ir algoritmas A*, naudoja žemėlapį, kuris sudarytas iš laukelių,vadinamų mazgais, tarp kurių egzistuoja tam tikri atstumai. Dijkstra algortmas, tikrina visus mazgus, susietus su pradiniu mazgu, ir suranda mazgą, kuris yra arčiausiai. Toks mazgas tampa išspręstu, ir toliau tikrinami visi mazgai kurie yra šaliaišspręstų mazgų. Taip algoritmas veikia besiplėsdamas tol, kol išsprendžiamas galutinio tikslo mazgas[6].

Algoritmo veiksmų seka yra tokia:

1. Nustatomas pradinis mazgas, kuris pažymimas kaip išspręstas;
2. Identifikuojami visi mazgai, kurie yra sujungti su pradiniu mazgu, bei suskaičiuojamas jų egzaminuojamas atstumas EA. EA = Atstumas iki išspręsto mazgo + atstumas iki egzaminuojamo mazgo (pav. 1.8);
3. Išsirenkamas mažiausias iš egzaminuojamų atstumų, ir mazgas pažymimas kaip išspręstas. Gauta trajektorija pridedama prie trajektorijų aibės;
4. Toliau mazgai, tiesiogiai sujungti su išspręstais mazgais egzaminuojami tuo pačiu metodu. Esant keliems skirtingiems mazgams su tuo pačiu atstumu, išsirenkamas bet kuris vienas ir pažymimas kaip išspręstas. Trajektorija pridedama prie trajektorijų aibės;
5. Algoritmas baigiasi, kai galutinio tikslo mazgas tampa išspręstas [6].



1.8 pav. Mazgų identifikavimas. O - pradinis išspręstas mazgas, T - tikslas [6].

Galima pastebėti, kad Dijkstra algoritmas labai panašus į A*, todėl natūraliai kyla klausimas, kuo jis pranašesnis arba prastesnis už pastarąjį. Skirtumas tas, kad šis algoritmas neturi euristinio įvertinimo (h(n) funkcijos) ir skirtingai negu A*, plečiasi į visas puses tolygiai, o ne kryptingai link tikslo. Dijkstra algoritmas dažniausiai išanalizuoja daug didesnį paieškos plotą prieš rasdamas tikslą, todėl ir šio algoritmo veikimas yra lėtesnis. Pasitaiko atvejų, kai neaišku, kur ar kaip toli yra tikslas, arba netgi keletas tikslų, o reikia pasiekti bent vieną iš jų. Taigi Dijkstra algoritmas padeda rasti bet kurį iš tikslų, kuris yra arčiausiai. Norint tą patį pasiekti su A* algoritmu, reikėtų suskaičiuoti trajektorijas iki vieno ir iki kito tikslo atskirai bei įvertinti atstumus ir tuomet pasirinkti arčiau esantį tikslą.

1.4 D algoritmas*

D* vardas šiam algoritmui buvo parinktas dėl to, kad pats algoritmas savo struktūra panašus į A*, bet kadangi trajektorijos skaičiavimo metu, jis gali būti *dinamiškai* perskaičiuojamas, pasirinkta raidė D.

Kaip A* ar Dijkstra algoritmai, D* algoritmas turi sąrašą būsenų, kurios gali būti priskirtos tam tikriems mazgams. Taip pat kaip ir minėtuose algoritmuose, paieškos laukas yra suskaidomas į mazgus. Viso D* algoritmo mazgai, gali turėti tokias būsenas:

- NAUJAS (NEW) reiškia, kad šis mazgas dar niekomet nebuvo Atvirąjame (OPEN) sąraše.
- ATVIRAS (OPEN) mazgas šiuo metu yra Atvirąjame sąraše (kaip A* algoritme).
- UŽDARAS (CLOSED) reiškia, kad mazgas nebėra atvirąjame sąraše.
- PADIDINTAS (RAISE) pažymi, kad kelio vertė padidėjo, lyginant su paskutiniu kartu, kai šis mazgas buvo atvirąjame sąraše.

- PAMAŽINTAS (LOWER) reiškia, kad kelio vertė sumažėjo, lyginant su paskutiniu kartu, kai mazgas buvo atvirąjame sąraše[7].

Judėjimo erdvė yra apibrėžiama kaip aibė būsenų, žyminčių roboto vietas, sujungtas krypties arkomis (*directional arcs*), kur kiekviena iš jų turi priskirtą vertę (kaip kelio vertę A* algoritme). Robotas pradeda savo kelionę konkrečioje būsenoje ir juda kryptiniais lankais pagal jų vertę, kol pasiekia TIKSLO (GOAL) būseną. Kiekviena būsena X (išskyrus tikslą) turi rodyklę (*backpointer*) į sekančią būseną Y, pažymimą funkcija b(X) = Y. D* algoritmas naudoja rodykles tam, kad nurodytų kelią iki tikslo. Kelionės vertė(*cost*), nuo būsenos Y iki būsenos X yra teigiamas skaičius, kuriam reikšmę suteikia vertės funkcija c(X,Y). Jei nuo Y iki X nėra lanko, tai reiškia, kad funkcija c(X,Y) yra neapibrėžta[7].

Būsenos, esančios atvirąjame sąraše, yra surikiuotos pagal savo pagrindinės funkcijos vertę (*key function value*). Parametras k_{min} yra apibrėžtas kaip *min(k(X))* visiems X, kurie yra atvirąjame sąraše t(X)=OPEN. Parametras k_{min} žymi labai svarbią ribą D* algoritme. Kai trajektorijos vertė yra lygi arba mažesnė negu k_{min}, ji yra optimali. Reikšmės didesnės už k_{min} negali būti optimalios trajektorijos dalis. Parametras k_{old}, yra lygus k_{min}, prieš pašalinant šią reikšmę iš atvirojo sąrašo (t.y. prieš tai buvęs k_{min})[7].

Būsenų išrikiavimas į eilę apibrėžia trajektoriją, kuri pažymima rodyklėmis (*backpointers*). Šis būsenų išdėstymas yra seka, jei $b(X_{i+1})=X_i$, visiems *i*,yra$1 \le i < N$ ir $X_i \neq X_j$, kur i,j, kurie $1 \le i < j \le N$. Taip ši seka apibrėžia kelią nuo X_N iki X_1. Seka $\{X_1,X_N\}$ apibrėžiama kaip *monotoniška*, jei "$t(X_i)$=CLOSED, ir $h(G,X_i)<h(G,X_{i+1})$", arba "$t(X_i)$=OPEN ir $k(G,X_i)<h(G,X_{i+1})$" visiems i, kurie $1 \le i < N$. D* algoritmas kuria ir stengiasi išlaikyti monotonišką seką {G,X}, mažejančią dabartinę arbą mažesnę suminę trajektorijos vertę kiekvienai X būsenai, kuri yra arba buvo atvirąjame sąraše. Turint būsenų seką $\{X_1,X_N\}$, būsena X_i yra protėvis (*ancestor*) X_j, jeigu $1 \le i < j \le N$, ir palikuonis X_j, jeigu $1 \le j < i \le N$[7].

D* algoritmas susideda iš dviejų pradinių funkcijų: peržiūrėti būsenas (*Process-state*) ir modifikuoti vertę (*Modify-cost*). *Process-state* funkcija yra naudojama skaičiuoti optimalios trajektorijos vertes iki tikslo, o *modify-cost* funkcija yra naudojama keisti lankų verčių funkcijas c(°)ir įvesti susietus mazgus į atvirąjį sąrašą. Pradedant naviguoti, t(°)=NEW yra nustatomas visoms būsenoms, h(G) prilyginamas nuliui, o G yra patalpinamas į atvirąjį sąrašą. Pirmoji funkcija process-state yra iššaukiama ir kartojama tol, kol roboto būsena X yra pašalinama iš atvirojo sąrašo. Tai reiškia, kad buvo suskaičiuota būsenų seka iki tikslo, arba, kad tokia seka neegzistuoja. Tada robotas pradeda sekti rodykles sekoje {X}, kol pasiekia tikslą arba aptinka klaidą lankų c(°) funkcijoje (pvz.: dėl aptiktos kliūties). Tuomet iškviečiama *modify-cost* funkcija, kuri pakoreguoja c(°), ir susietas būsenas patalpina į atvirąjį sąrašą. Tuomet sugeneruojama nauja seka ir robotas gali toliau sekti rodyklėmis iki tikslo arba kitos klaidos [7].

Paveiksluose 1.9 ir 1.10 pateikiami funkcijų *process-state* ir *modify-cost* pseudokodai.

```
L1   X = MIN – STATE( )
L2   if X = NULL then return –1
L3   k_old = GET – KMIN( ); DELETE(X)
L4   if k_old < h(X) then
L5     for each neighbor Y of X:
L6       if h(Y) ≤ k_old and h(X) > h(Y) + c(Y, X) then
L7         b(X) = Y; h(X) = h(Y) + c(Y, X)
L8   if k_old = h(X) then
L9     for each neighbor Y of X:
L10      if t(Y) = NEW or
L11        (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) or
L12        (b(Y) ≠ X and h(Y) > h(X) + c(X, Y)) then
L13        b(Y) = X; INSERT(Y, h(X) + c(X, Y))
L14  else
L15    for each neighbor Y of X:
L16      if t(Y) = NEW or
L17        (b(Y) = X and h(Y) ≠ h(X) + c(X, Y)) then
L18        b(Y) = X; INSERT(Y, h(X) + c(X, Y))
L19      else
L20        if b(Y) ≠ X and h(Y) > h(X) + c(X, Y) then
L21          INSERT(X, h(X))
L22        else
L23          if b(Y) ≠ X and h(X) > h(Y) + c(Y, X) and
L24            t(Y) = CLOSED and h(Y) > k_old then
L25            INSERT(Y, h(Y))
L26  return GET – KMIN( )
```

1.9 pav. Funkcija *process-state*.

```
L1   c(X, Y) = cval
L2   if t(X) = CLOSED then INSERT(X, h(X))
L3   return GET – KMIN( )
```

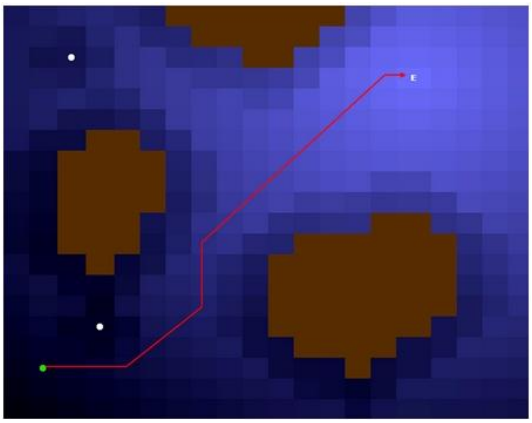
1.10 pav. Funkcija *modify-cost*.

1.5 Potencialų laukų algoritmas

Potencialų laukų metodo autonominio roboto navigacijai esmė yra traukiančių ir atstumiančių potencialų priskyrimas. Traukiantis potencialas priskiriamas naviguojančio roboto tikslui, o stumiantis potencialas - kiekvienai iš kliūčių naviguojamojoje aplinkoje.[7]

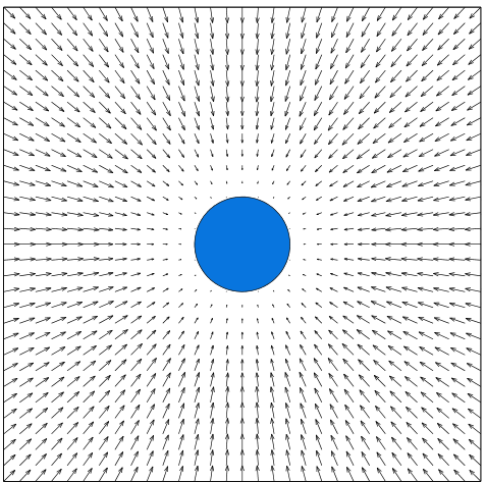
Potencialai - traukiantis ir atstumiantis - sukuria apie save lauką, kuris tolstant nuo to taško, silpsta, kol galiausiai pranyksta. Naviguojančio objekto atžvilgiu veikia dvi jėgos -

traukianti ir atstumianti, tokiu atveju kiekvienoje pozicijoje galima rasti atstojamąją jėgą, kurios kryptimi objektas turėtų toliau judėti. 1.11 paveiksle vaizduojamas optimalaus kelio potencialų laukų metodu radimas; šviesesni laukeliai - traukiantis potencialas, tamsesni - atstumiantis.

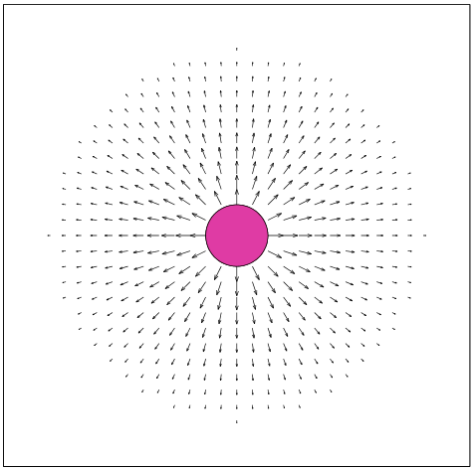


1.11 pav. Trajektorijos radimas potencialų laukų metodu [8].

Pagrindinis statybos vienetas, kuriant potencialų lauką, yra veiksmo vektorius (*action vector*), kuris turėtų atitikti roboto judėjimo kryptį ir greitį. Kiekviena skirtinga elgsena iššaukia tam reikalingą vektorių. Viena iš tokių elgsenų yra tikslo siekimas. Šiai elgsenai yra priskiriamas tikslas robotui judėti link identifikuoto taikinio. Tokios elgsenos išvestis yra vektorius, kuris nukreipia robotą link tikslo. Tarkime, kad turime dviejų dimensijų erdvę, o jos viduryje - tikslą. Iš kiekvieno taško toje erdvėje, stumdami robotą tikslo link, užfiksuotume roboto judesio vektorių ir visą vektorių aibę pavaizduotume erdvėje (pav. 1.12). Tokia erdvę būtų galima vadinti potencialų lauku, nes joje atvaizduojami dirbtiniai energijos potencialai, kuriais seks robotas. Potencialų laukas susietas su tiklo siekimo elgsena, yra traukiančio potencialų lauko pavyzdys, nes laukas verčia robotą judėti link tikslo (vektoriai nukreipti tikslo link). Tačiau iš tikrųjų visas toks laukas nėra skaičiuojamas roboto - suskaičiuojama tik ta lauko dalis, kuria robotas juda[9].



1.12 pav. Tikslo siekimo PL [9].



1.13. Kliūties vengimo PL [9].

Potencialų laukas gali būti suskaičiuojamas sekančiu būdu:

1. Tarkim, kad tikslo pozicija yra (x_T, y_T). Tarkime, kad r yra tikslo spindulys, o vektorius $v=[x,y]^T$ žymi objekto poziciją (x,y) Dekarto plokštumoje.
2. Surandamas atstumas tarp objekto ir tikslo:

$$d = \sqrt{(x_T - x)^2 + (y - y_T)^2} \quad (2)$$

3. Randamas objekto pasisukimo kampas (orientacija):

$$\theta = \tan^{-1}\left(\frac{y_T - y}{x_T - x}\right) \quad (3)$$

4. Pagal pateiktas taisykles nustatomi Δx ir Δy:

jei $d<r$, tai $\Delta x = \Delta y = 0$

jei $r \leq d \leq s + r$, tai $\Delta x = \alpha(d - r)\cos(\theta)$ ir $\Delta y = \alpha(d - r)\sin(\theta)$

jei $d > s + r$, tai $\Delta x = \alpha s \cos(\theta)$ ir $\Delta y = \alpha s \sin(\theta)$ (4)

Šios formulės nustatomos tikslui su spinduliu r. Kai objektas pasiekia tikslą, jokios jėgos jo nebeveikia, kaip sąlygoje $d < r$, kur Δx ir Δy tampa lygūs nuliui. Laukas yra išplitęs nuotoliu s, ir objektas pasiekia lauko ribą, kai $d = s + r$. Už šio lauko ribų, vektoriaus dydis yra nustatomas kaip maksimalus. Kai objektas yra tarp lauko r ir $r+s$ atstumų, vektoriaus dydis kinta proporcingai atstumui nuo tikslo - kuo atstumas didesnis, tuo vektoriaus reikšmė didesnė. Konstanta α naudojama tam, kad būtų galima patogiai keisti lauko stiprumą[9].

Plokštumoje, kuri pavaizduota 1.12 paveiksle, robotas lengvai pasiektų tikslą, be jokių problemų. Tačiau šioje plokštumoje pavaizduotas tik vienas elgsenos tipas. Be šios elgsenos, toje pačioje plokštumoje, galėtų atsirasti ir kliūties vengimo elgsena (pav. 1.13). Kliūties vengimo elgsenos diagrama gali būti sudaroma taikant sekančias taisykles:

a) Pažymėkime (x_K, y_K), kaip kliūties poziciją. Pažymėkime $r$kaip kliūties spindulį. Vektorius $v=[x,y]^T$ žymi objekto poziciją erdvėje.

b) Randamas atstumas tarp objekto ir kliūties:

$$d = \sqrt{(x_K - x)^2 + (y - y_K)^2} \quad (5)$$

c) Randamas kampas, tarp objekto ir kliūties:

$$\theta = \tan^{-1}\left(\frac{y_K - y}{x_K - x}\right) \quad (6)$$

d) Pagal pateiktas taisykles nustatomi Δx ir Δy:

jei $d<r$, tai $\Delta x = -sign(\cos(\theta))\infty$ ir $\Delta y = -sign(\sin(\theta))\infty$.

jei $r \leq d \leq s + r$, tai $\Delta x = -\beta(s + r - d)\cos(\theta)$ ir $\Delta y = -\beta(s + r - d)\sin(\theta)$.

jei $d > s + r$, tai $\Delta x = \Delta y = 0$. (7)

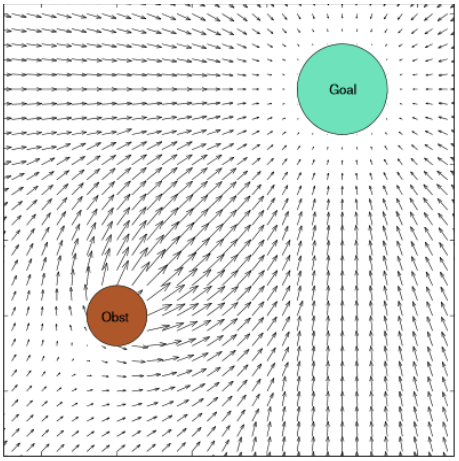
Kliūties viduje atstumiantis potencialų laukas yra begalinai didelis ir nukreiptas nuo centro. Už įtakos lauko ($r+s$), atstumiantis potencialų laukas yra lygus nuliui. Įtakos lauke, už

kliūties apskritimo, vektoriaus reikšmė didėja nuo nulio (kai $d=s+r$) iki β*s (kai $d=r$). β konstanta yra naudojama kaip koeficientas, kuriuo būtų galima keisti lauko stiprį. Vektorius visuomet nukreiptas nuo kliūties centro, tai padaroma padėjus neigiamą ženklą, prieš apibrėžiant Δx ir Δy.

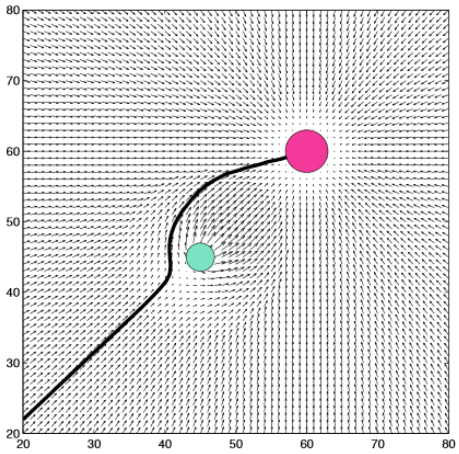
Kadangi realybėje reikalinga, kad robotas pasiektų tikslą ir išvengtų kliūčių, naudinga yra tik šių dviejų skirtingų elgsenų potencialų laukų kombinacija. Tai atliekama šiuos laukus tiesiog sudedant. Tai padarius, robotui, kuris turi dvi elgsenas, plokštumoje su tikslu ir kliūtimi turi būti sugeneruojamas naujas potencialų laukas, pavaizduotas 1.14 paveiksle. Šis laukas sugeneruojamas pirmiausia randant $\Delta_T x$ ir $\Delta_T y$ - vektorių, sugeneruotą traukiančio tikslo, po to randant vektorių $\Delta_K x$ ir $\Delta_K y$ - sugeneruotą atstumiančiosios kliūties. Galiausiai šie vektoriai sudedami kartu ir gaunamas atstojantysis vektorius:

$$\Delta x = \Delta_K x + \Delta_T x \quad (8)$$

$$\Delta y = \Delta_K y + \Delta_T y \quad (9)$$



1.14. pav. Potencialų laukas, kai egzistuoja abi elgsenos - traukianti ir atstumianti [9].



1.15. Galutinė roboto trajektorija judant potencialų lauku [9].

Tarkime, robotas atsiduria tokioje dviejų elgsenų potencialų plokštumoje kaip 1.15 paveiksle, netoli koordinačių pradžios. Pirmiausia, robotas nustato Δx pagal (8) formulę, kuris yra sugeneruojamas potencialų laukų pagal dvi elgsenas. Tokiu pat būdu, pagal (9) formulę nustatomas Δy. Tuomet, roboto greitis nustatomas pagal formulę (10). Pasisukimo kampas nustatomas pagal formulę (11). Judant plokštuma, robotas stebi aplinką ir nustatomi nauji veiksmo vektoriai, pasirenkamos kitos kryptys bei greičiai. Galutinė trajektorija atrodo kaip 1.5 paveiksle[8].

$$v = \sqrt{\Delta x^2 + \Delta y^2} \quad (10)$$

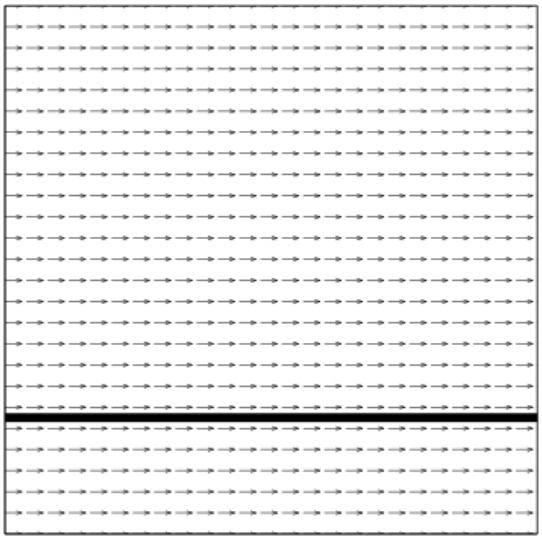
$$\theta = \tan^{-1}\frac{\Delta y}{\Delta x} \qquad (11)$$

1.5.1 Kiti potencialų laukų tipai

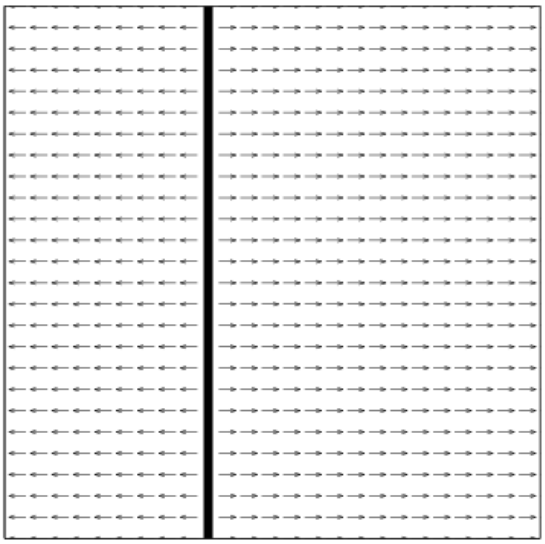
Potencialų laukų metodo principas buvo paaiškintas naudojant dviejų tipų elgsenas arba laukus - kai tikslas arba kliūtis yra apskritimo formos, o jėgos veikia nuo centro iki objekto, arba atvirkščiai - nuo objekto iki centro (traukimo ir atstūmimo efektai). Tačiau yra ir kiti laukų tipai, kurie gali būti naudingi taikant šį algoritmą roboto navigacijai ar kompiuterinių žaidimų programavimui.

Vienas iš tokių laukų yra tolygus (*uniform*) potencialų laukas (pav. 1.16). Tokio lauko vektorų reikšmėms priskiriami tam tikri pastovūs dydžiai, tarkim*Δx* = c, *Δy*= 0 (arba kitos pastovios reikšmės). Toks laukas gali būti naudojamas tokioms elgsenoms, kaip sekimas pagal sieną, arba sugrįžimas į žinomą teritoriją ir t.t.

Dar vienas toks laukas - statmenasis (*perpendicular*). Šio lauko vektorių reikšmės, pvz.: *Δx* = ±c, *Δy*= 0 (arba kitos norimą efektą suteikiančios reikšmės). Tokio tipo laukas gali išreikšti elgsenas kaip: išvengti sienos, vengti tam tikros teritorijos ir t.t. Laukas pavaizduotas 1.17 paveiksle.



1.16 pav. Tolygus potencialų laukas. [9]

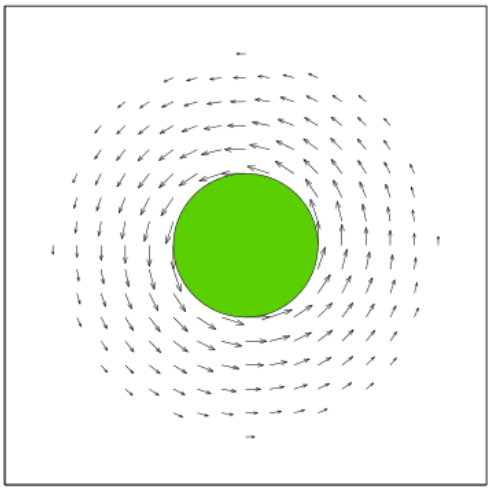


1.17 pav. Statmenas potencialų laukas. [9]

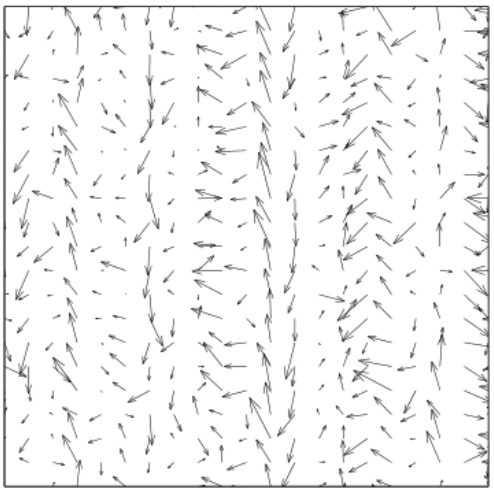
Vienas iš žaidimuose dažniau naudojamų laukų tipų yra liestinės (*tangential*) potencialų laukas (pav. 1.18). Šio tipo lauko vektorių parametrai *Δx* ir *Δy*randami taip pat, kaip ir atstumiančios kliūties lauko, tačiau kampas θ yra modifikuojamas pagal formulę θ = θ ± 90°.

Taip vektorius perstumiamas ir jo kryptis yra nebe nuo kliūties centro iki objekto, tačiau liestinės kryptimi. Ženklu keičiamas judesio tipas prieš arba pagal laikrodžio rodyklę. Toks laukas gali būti naudingas elgsenoms, kaip apsupti objektą, saugoti, apvažiuoti ratu ir t.t.[13].

Prie šių laukų dar reikėtų pridėti atsitiktinį (*random*) potencialų lauką (pav. 1.19). Šis laukas naudingas tuo, kad gali padėti atsitiktiniais judesiais išsivaduoti robotui iš lokalaus minimumo (*local minima*). Šiuo metodu d ir θ parametrai pasirenkami atsitiktinai su vienoda tikimybe iš [0, λ] ir [0, 2π] intervalų.



1.18 pav. Liestinės potencialų laukas [9].



1.19 pav. Atsitiktinis potencialų laukas [9].

Potencialų laukų metodo privalumai yra tie, kad optimali trajektorija yra skaičiuojama ir įvertinama naviguojančiam objektui judant. Šiuo metodu iškart įvertinamas visas kelias ir trajektorija keičiama tik pasikeitus aplinkai. Taip lengviau susidorojama su dinamika, jeigu yra judančių kliūčių, ir sutaupoma laiko skaičiuojant, lyginant su kitais algoritmais[8].

Šis metodas jau buvo pritaikytas keletu skirtingų atvejų roboto navigacijai. Paprasčiausias iš atvejų - kai robotas naviguoja žinomoje aplinkoje, kur tikslui ir kliūtims jau priskirti potencialai. Kai kliūtys nėra iš anksto žinomos, potencialai turi būti priskiriami robotui judant ir aptinkant naujas kliūtis. Tokiu atveju, kai yra vienas traukiantis potencialas priskirtas tikslui, ir visi kiti atstumiantys potencialai - kliūtims, potencialų laukų metodas turi vieną svarbų apribojimą - kai kurioms kliūčių konfigūracijoms gali būti neįmanoma sukurti tinkamos atstojamosios jėgos, kad kliūčių būtų išvengta.

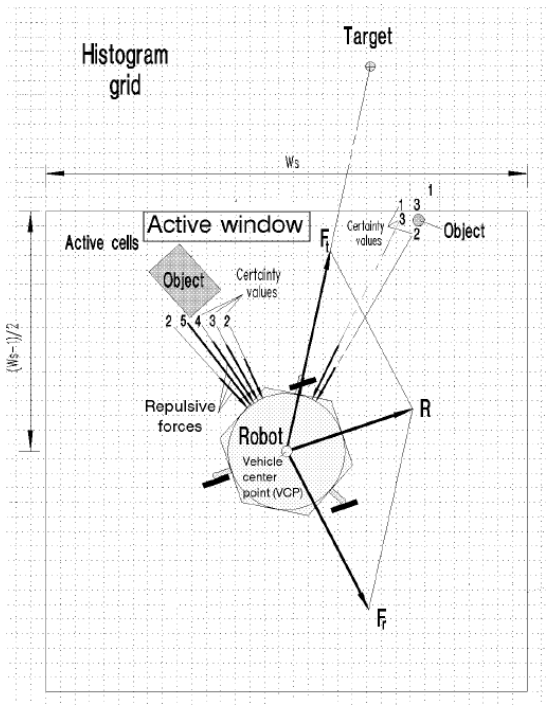
Kitas atvejis - kai naudojami keli traukiantys potencialai, bei laipsniškas atstumiančių potencialų įterpimas, robotui aptinkant naujas kliūtis, buvo įgyvendintas siekant išvengti didelių kliūčių nežinomoje aplinkoje [10].

*The virtual force field*arba virtualių jėgų laukas, - potencialų laukų metodu paremtas metodas, specialiai pritaikytas realaus laiko kliūčių išvengimui greitaeigiams robotams. VJL metodas suteikia greitą, nenutrūkstamą, sklandų judėjimą tarp netikėtų kliūčių.

VJL metodas naudoja dviejų dimensijų Dekarto sistemą (tinklelį) kliūtims atvaizduoti (pav. 1.20). Kiekvienas laukelis (i,j) šioje sistemoje turi užtikrintumo reikšmę (*certainty value*) $C_{i,j}$, kuri parodo algoritmo užtikrintumą dėl kliūties egzistavimo šiame laukelyje. Užtikrintumo reikšmės laukeliuose didėja, kai ultragarsinis jutiklis aptinka kliūtį tame laukelyje.

Tuo pačiu metu yra pritaikomas ir potencialų laukų principas. Kiekvienas laukelis, kuriame yra kliūtis sukuria atstumiančią jėgą roboto link, kuri yra atvirkščiai proporcinga atstumui tarp roboto centro ir objekto. Taip pat taikinys (tikslas) sukuria jėgą, kuri veikia nuo roboto, tikslo link - jėga yra priklausoma nuo tikslo koordinačių bei atstumo iki roboto.

Sudėjus visas veikiančias jėgas, gauname atstojamąją jėgą - vektorių, kurio dydis ir kryptis parodo robotui kuria kryptimi ir kokiu greičiu važiuoti optimalia trajektorija. [10]



1.20 pav. Virtualių jėgų metodu planuojama trajektorija [10].

1.6 Trajektorijos radimas naudojant dirbtinius neuroninius tinklus

Dirbtiniai neuroniniai tinklai, taip pat naudojami roboto navigacijai ir kelio planavimui. Lyginant su kitais algoritmais, naudojamais šiai problemai spręsti, DNT yra naujesnis metodas, ir šio metodo potencialas dar nėra iki galo atskleistas.

Vienas iš pavyzdžių, kai ieškant optimalios trajektorijos naudojami dirbtiniai neuroniniai tinklai, neuroninis tinklas yra išskaidomas į du potinklius: tikslo siekimui ir kliūties išvengimui. Dirbtinis neuroninis artumo (*proximity*) potinklis bando pasiekti užduotą galutinį tikslą, kai tuo tarpu kitas - spindulinis (*ray*) potinklis - bando išvengti kliūčių. Žinoma, tarp šių tinklų vyksta konfliktas, kai bandant išvengti kliūties, tuo pačiu reikia nukrypti nuo trajektorijos, vedančios į tikslą. Šie konfliktai gali priversti robotą judėti ta pačia trajektorija arba pasiklysti ir prarasti užduotą tikslą.

Dar vienas būdas - naudoti tikimybinį neuroninį tinklą (*probabilistic neural network*). Šio tipo neuroninis tinklas palengvina mokymosi procesą, sumažiną reakcijos laiką bei greičiau atlieka skaičiavimus. Šio tinklo išvesties duomenys - vairavimo kryptis, kuri gali būti trijų tipų: kairėn, dešinėn arba tiesiai.

Neuroninių tinklų privalumas yra tas, kad *mapping'o* funkcija tarp jutiklių duomenų ir valdymo veiksmų nereikalauja analitinio modelio. Kita vertus, neuroninio tinklo mokymuisi reikia apdoroti didžiulį duomenų kiekį [12].

1.7 RNA (Robot Navigation Ant) algoritmas

Autonominio roboto navigacija iš anksto nežinomoje aplinkoje, reiškia, kad jis neturi jokios išankstinės informacijos apie aplinką, kurioje bus naviguojama, ir visą informaciją turi gauti iš jutiklių. Sritis, kurią robotas gali skanuoti savo jutikliais, yra vadinama matymo sritimi, MS (*Visual Domain, VD*). Robotas gali skaičiuoti optimalią trajektoriją matymo srityje, tačiau negali suplanuoti optimalaus kelio globalioje aplinkoje pirmu bandymu, nes informacija gaunama tik iš matymo srities. Nesunku nuspėti, kad didėjant plotui kuriame naviguojama ir kliūčių skaičiui, užduotis surasti optimalią trajektoriją sunkėja.

Situacijose, kuriose robotas naviguoja plote, didesniame už savo matymo sritį, ir kuriame egzistuoja nežinomos statiškos ir dinamiškos kliūtys, robotas turi sugebėti spręsti tris uždavinius:

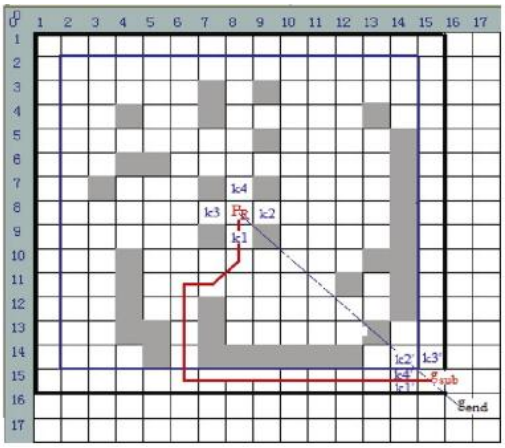
- Pakankamai greitai planuoti trajektoriją lokaliame plote, kad globalios trajektorijos planavimas vyktų realiu laiku;
- Garantuoti, kad trajektorija yra trumpiausia globalioje aplinkoje;
- Kaip išvengti dinamiškų kliūčių ir išspręsti *aklavietės* (*deadlock*) ir švytavimo uždavinius.

Šios trys užduotys iki galo vis dar nėra išspręstos.

Sprendžiant šiuos uždavinius, buvo pasitelkiamas RNA algoritmas. Šis algoritmas paremtas skrudžių-žvalgių elgesiu maisto paieškos metu. Naudojant šį algoritmą pirmiausia galutinio tikslo taškas g_e yra pažymimas arčiausiame taške g_{e*} už roboto matymo srities, kuris yra apytiksliai žinomas. Šis taškas yra imamas kaip lokalios navigacijos tarpinis tikslas (pav. 1.21).

Nepaisant to, kad robotas negali matyti galutinio tikslo taško, jo koordinatės gali būti žinomos, nes yra tiksliai įvestos. Taip pat ir tarpinio tikslo taškas, robotui nėra matomas, tačiau koordinatės yra suskaičiuojamos. Tada robotas naudodamasis jutikliais gali aptikti kliūtis ir sudaryti aplinkos modelį savo matymo srityje. Tuomet MSAC (*multi-scout ants cooperation*) algoritmas yra naudojamas suplanuoti maršrutą roboto matymo srityje, kurioje iš pradžių yra ieškomos ir įvertinamos tik statiškos kliūtys. Po to robotas tikrina, ar kuri nors iš kliūčių juda, ir bando numatyti jų trajektorijas. Jeigu pastebima, kad judančios kliūties ir roboto judėjimo trajektorijos kertasi, numatomas susikirtimo taškas yra laikinai įvertinamas kaip statiška kliūtis, ir vėl naudojamas MSAC algoritmas perskaičiuoti maršrutui. Robotas pajuda vieną ėjimą į priekį ir visas procesas yra kartojamas iš naujo. Visas procesas yra kartojamas tol, kol galutinis tikslas atsiduria roboto matymo srityje. Šiuo atveju RNA algoritmas yra globalaus kelio algoritmas,

kuris panaudoja MSAC algoritmą lokaliai navigacijai statiškoje aplinkoje, ir susidūrimo su dinamiškomis kliūtimis išvengimui [13].



1.21 pav. Tarpinio tikslo schema [13].

MSAC algoritmas naudoja dvi skruzdžių šeimas dviejų krypčių paieškoje. Skruzdžių šeima *m* pažymi dabartinę roboto *($P_R(t)$)* poziciją kaip lizdą, o tarpinį tikslą g_{sub}, - kaip maisto šalitinį. Kita skruzdžių šeima n pažymi tarpinį tikslą kaip lizdą, o dabartinę roboto poziciją - kaip maisto šaltinį. Šios dvi šeimos ieško kelio dviem priešingomis kryptimis. Kiekviena skruzdė ieško kelio, aplankydama arčiausiai tikslo esantį langelį, iš ją supančių langelių - bet tik jei jis dar nebuvo aplankytas kitos skruzdės. Aibė langelių, kuriuos skruzdės jau aplankė yra išsaugomos globaliame *TABU* sąraše. Skruzdės dalijasi šiuo sąrašu, kad išvengtų pakartotinio ieškojimo tuose pačiuose langeliuose, ir tik tada, kai nėra gretimo neaplankyto langelio, galima pasirinkti jau aplankytą [13].

1.8 Literatūros analizės rezultatai ir išvados

Šioje darbo dalyje atlikta egzistuojančių algoritmų optimalios trajektorijos planavimui apžvalga. Plačiausiai buvo apžvelgti A*, Dijkstra, D*, potencialų laukų metodo algoritmai bei keletas jų atmainų. Trumpai aprašyti rečiau naudojami RNA, MSAC algoritmai bei dirbtinių neuroninių tinklų naudojimas.

Iš surinktos informacijos apie algoritmus, pastebėta, kad A* ir Dijikstra algoritmai yra labai panašūs. Pagrindinis skirtumas tarp jų yra tas, kad A* algoritmas iškart atlieka paiešką visame lauke viena kryptimi - tikslo link. Tuo tarpu Dijkstra algoritmas plečiasi tolygiai visomis kryptimis, todėl jis tinkamas naudoti esant keliams tikslo taškams - taip pirmiausia surandamas arčiau esantis. D* algoritmas yra A* modifikacija dinamiškai aplinkai, jis turi gerokai daugiau mazgo būsenų ir yra sudėtingesnis. Potencialų laukų algoritmas yra visiškai kitokio pobūdžio - jo stiprybė lyginant su kitais algoritmais yra ta, kad trajektorija yra skaičiuojama robotui judant, taigi geresnė paties algoritmo greitaveika. Kiti algoritmai buvo apžvelgti trumpiau, ir stipriosios bei silpnosios savybės nenustatytos.

A* ir potencialų laukų algoritmai yra vieni iš plačiausiai taikomų, tačiau ganėtintai skirtingi. Abu šie algoritmai naudojami statiškoje, iš anksto žinomoje aplinkoje, turi skirtingas stipriąsias bei silpnąsias puses. Dėl šių priežasčių nuspręsta juos pasirinkti testavimui.Surinkta informacija bus panaudota programuojant robotą optimalios trajektorijos planavimui, palyginamas skirtingų algoritmų pritaikymo efektyvumas.

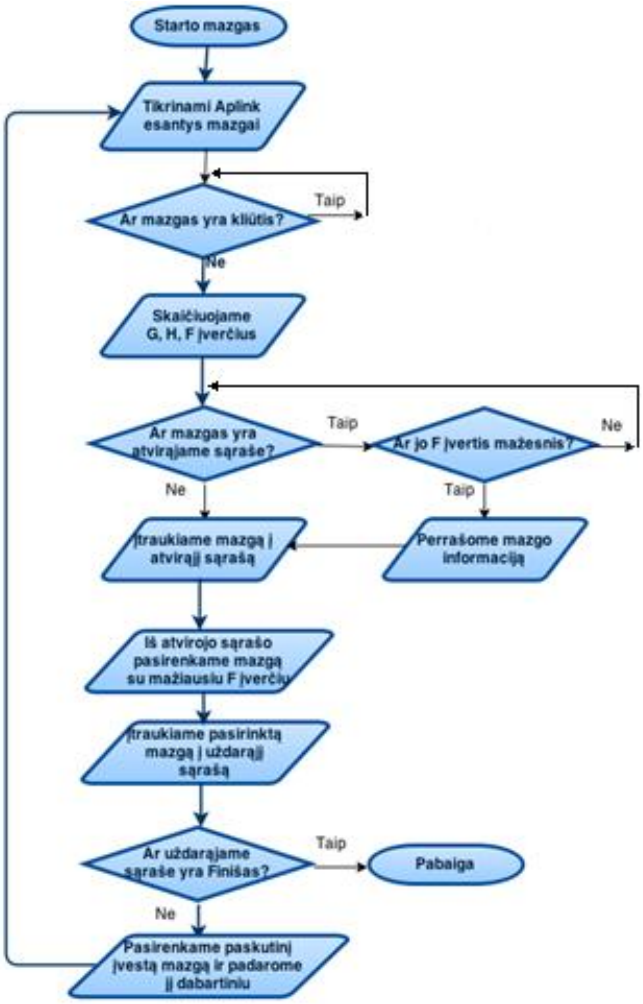
2 TYRIMŲ DALIS

Tiriamojoje baigiamojo projekto dalyje C# programavimo kalba suprogramuojami A* ir potencialų laukų metodoalgoritmai optimaliai trajektorijai rasti. Algoritmai programuojami turint omenyje, kad vienas iš jų bus naudojamas galutiniam roboto programavimui. Judėjimo laukas algoritmams kuriamas žinant tikruosius lauko matmenis, roboto gabaritinius matmenis bei apytikslį veikimo principą: kliūčių atpažinimui ir navigacijai naudojama tinklo kamera, sujungta su kompiuteriu, o duomenys Bluetooth ryšiu perduodami į Arduino robotą.

Algoritmams palyginti naudojamas vienodas judėjimo laukas, kurio matmenys 10x7 langelių (mazgų). Vieno mazgo dydis yra roboto gabaritinių matmenų (19x19 cm).Kiekvienam iš metodų sugeneruojama 10 000 atstitiktinių laukų su tikimybe nuo 0% iki 25%, kad laukelyje bus kliūtis. Kiekvienai imčiai suskaičiuojama, kiek procentų iš šių sugeneruotų laukų trajektorija tikslui pasiekti buvo surasta. Taip pat suskaičiuojamas vidutinis surastos trajektorijos ilgis ir reikalingas pasisukimų skaičius.Algoritmai tarpusavyje palyginami pagal rastos trajektorijos ilgį, pasisukimų skaičių ir surastų trajektorijų procentą. Trajektorijos ilgis bei pasisukimų skaičius labiausiai įtakoja roboto greitaveiką, judant link užduoto tikslo taško, tačiau svarbiausias algoritmo įvertinimui naudojamas parametras yra sugebėjimas surasti trajektoriją.

2.1 A algoritmo programa*

A* algoritmui realizuoti C# programavimo kalba buvo sukurta programa (pav. 2.1). Programa susideda iš keleto svarbių skirtingų komponentų. Visų pirma sudaroma matrica judėjimo laukui atpažinti. Matrica sudaryta iš skaičių - kiekviena vieta matricoje atitinka erdvės mazgą, kuriame judės robotas. Skaičius tame mazge nurodo jo būseną. 0 matricoje reprezentuoja laisvą taką, 1 - kliūtį, 2 - starto tašką, 3 - užduotą tikslą (pav. 2.2).



2. 1 pav. A* algoritmo įgyvendinimo programoje struktūrinė schema.

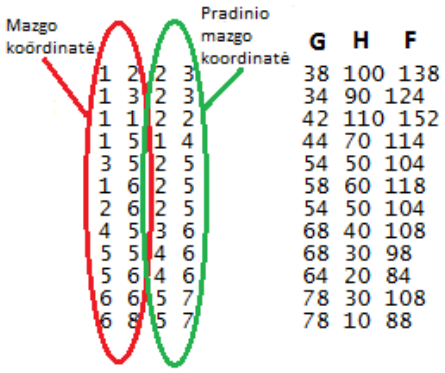
```
1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 1 0 1 0 0 0 0 1
1 0 2 0 1 0 0 1 0 0 0 1
1 0 0 0 1 0 0 1 0 0 0 1
1 0 0 0 1 0 0 0 0 0 0 1
1 0 0 0 1 0 0 0 3 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1
```

2.2 pav. Užduoties ir skaičiavimų matrica.

Kitas svarbus algoritmo komponentas - atvirasis sąrašas (*Open List*). Jame kaupiami visi mazgai, kuriuos robotui gali tekti aplankyti, tačiau nebūtinai - tik dalis iš jų priklausys optimaliai trajektorijai. Visiems šiame sąraše esantiems mazgams buvo atliekami skaičiavimai dėl jų tinkamumo būti optimalios trajektorijos dalimi. Atvirąjame sąraše išsaugomi mazgai su savimi turi papildomą informaciją - pradinio mazgo (*parent node*), iš kurio jie atėjo, koordinates. Taip pat ir euristinius įverčius G, H ir F kurie šioje programoje skaičiuojami Manhatano metodu. G H F įverčiai nepaisant to, kad yra išsaugomi atskirame .txt faile, yra susieti su atvirojo sąrašo eilutėmis, todėl kiekviena šiame sąraše esanti eilutė tikrai turi G, H, F įverčius. Kiekvieno žingsnio (pav. 2.1) metu įtraukiant naujus mazgus į atvirąjį sąrašą, jiems yra suskaičiuojami G,

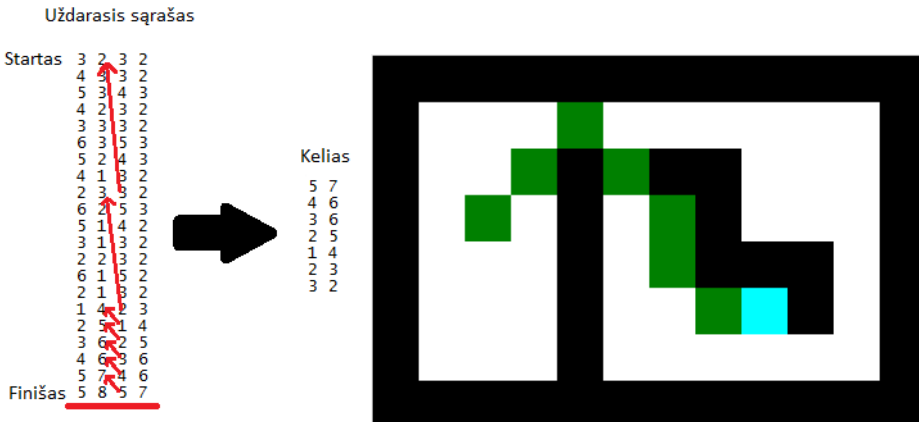
H, F įverčiai, o mažiausią įvertį turintis mazgas pasirenkamas kaip *dabartinis* ir kitame žingsnyje visi skaičiavimai atliekami su juo. Įverčiai skaičiuojami Manhatano metodu, pagal (1) formulę.

Alternatyviai galima naudoti kitus metodus, tokius kaip kampo kirtimo (*diagonal shortcut*), kurie su Manhatano metodu yra plačiausiai naudojami. Šiuo metodu įverčiai suskaičiuojami pagal antrame skyriuje pateiktą pseudokodą.



2.3 pav. Atvirasis sąrašas (kairėje) ir G, H, F įverčių matrica (dešinėje).

Pasirinktas dabartinis langelis yra įtraukiamas į uždarąjį sąrašą. Toks mazgas yra įtraukiamas kas žingsnį, kol surandamas užduotas finišas. Uždarąjame sąraše talpinami mazgai taip pat turi pradinio langelio (*parent*) koordinatę, todėl, kai į šį sąrašą patenka finišo mazgas, galima atsekti, iš kokio mazgo buvo patekta į šį. Iš prieš tai buvusio galima surasti dar ankstesnį ir t.t. Kaip tai atliekama pavaizduota 2.4 paveiksle. Toliau programoje surenkamos trumpiausio kelio koordinatės ir atvaizduojamos paveikslo srityje. Šios koordinatės ir yra optimali trajektorija.



2.4 pav. Trajektorijos radimas naudojant uždarąjį sąrašą ir atvaizduota trajektorija.

2.2 Potencialų laukų programa

Kaip ir A* algoritmui, taip pat ir potencialų laukų metodui, judėjimo laukui sudaryti naudojama skaičių matrica. Skaičių matricoje 1 žymi kliūtį, 2 - tikslą, o 3 - starto tašką. Naudojantis anksčiau(apžvelgiant potencialų laukų algoritmą) paminėtomis

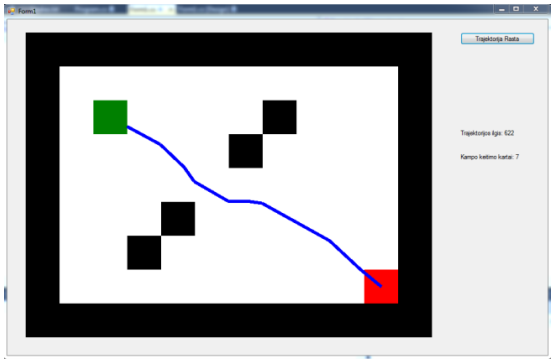
formulėmis,programoje sukuriamos dvi elgesio formos robotui. Pirma - siekti tikslo, antra - vengti kliūties. Kadangi judėjimo lauke gali būti daug kliūčių, visi jų poveikiai susumuojami, ir suminis vektorius naudojamas kaip galutinis *vengti kliūties* elgesio poveikis.

Skirtingai negu A* algoritme, kur keliaujantis objektas juda nuo mazgo link mazgo (nuo mazgo centro iki mazgo centro), šiame algoritme judantis objektas juda nuo vieno potencialo vektoriaus veikimo zonos, prie kito. Šios veikimo zonos yra tokio paties dydžio, kaip ir A* algoritme naudojami mazgai, tačiau iškart pasiekus naują veikimo zoną, pradeda veikti naujas vektorius, kurio kryptis yra "siekti tikslo" bei "vengti kliūties" vektorių suma. Patekus į naują zoną gaunamas veikiantis potencialas yra konvertuojamas į kampą ir atstumą. Kampas ir atstumas nurodo vietą, ties kuria judantis objektas pateks į kito potencialo zoną.

C# programavimo kalba sukurta programa potencialų laukų metodo algoritmui (Pav. 2.5). Jeigu A* algoritme naudojami metodai turi nedaug įtakos surandamai trajektorijai, potencialų laukų metodo algoritme nuo parametrų priklauso labai daug. Pagrindiniai parametrai yra:

- traukiančiojo lauko sklaida;
- stumiančiojo lauko sklaida;
- traukiančiojo lauko stipris;
- stumiančiojo lauko stipris.

Lauko sklaidos, nurodo kokiu atstumu nuo objekto (kliūties ar tikslo) pradeda keistis jo poveikis keliaujančiam objektui (robotui). Parinktu atstumu nuo kliūties atsiranda stumiantis potencialas, kuris stiprėja artėjant link kliūties, ir pasiekia maksimalų stiprį, kai yra visiškai šalia. Esant visiškai šalia tikslo - atvirkščiai; potencialas tampa nulinis ir stiprėja tolstant nuo jo. Pasiekus numatytą sklaidos ribą, traukiantis potencialas tampa maksimalus.

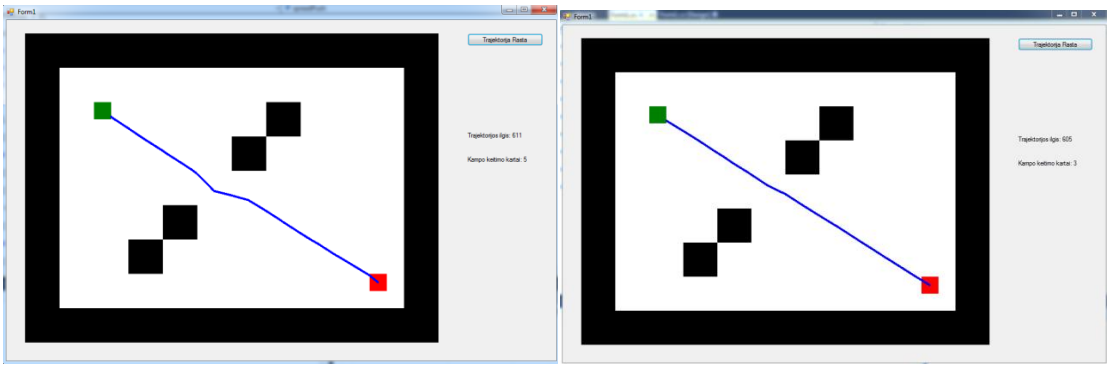


2.5 pav. Potencialų laukų programos langas.

Kadangi objektai (tikslai ir kliūtys) traktuojami kaip taškai, norint keisti jų dydį jiems suteikiamas spindulys, taigi objektas visuomet tampa apskritimo formos. Tai reiškia, kad norint bet kokiu atveju išvengti kliūties, vienetinio kvadrato formos kliūtis tampa $\sqrt{2}$ diametro apskritimo formos kliūtimi. Esant tokiam mažam judėjimo laukui, kaip pavaizduota (10x7),viena

priešais kitą esančios kliūtys bei jų sklaidos laukai labai stipriai įtakoja vienas kitą ir veikia neefektyviai - ne tik nustumia judantį objektą nuo kliūties, tačiau ir nuo trumpiausio galimo kelio, arba sudaro lokalųjį minimumą, kuriame robotas gali užstrigti. Taip pat tokios kliūtys gali sudaryti tokią šalia esančių potencialų kombinaciją, kad robotas judės ta pačia trajektorija ir niekaip neištrūks iš kilpos.

Paveiksle 2.6 galima pastebėti, kaip susmulkinus potencialų tinklelį pasikeičia trajektorija. Sutrumpėja trajektorijos atstumas, sumažinamas pasisukimo kartų skaičius ir, svarbiausia, kliūtis apvažiuojama tiksliau. Pakeitus lauko paplitimo parametrus, gautus rezultatus galima dar pagerinti. Visa tai rodo, kad potencialų lauko metodas labai priklausomas nuo lauko dydžio (potencialų zonų skaičiaus). Kuo smulkesnis tinklelis ir kuo daugiau zonų, tuo geriau veikia algoritmas. Taip yra dėl to, kad robotui artėjant prie kliūties jos įtakos paplitimas gali būti tokių pačių matmenų, tačiau juose telpa daugiau potencialų, taigi laukas turi daugiau vietų su skirtingais potencialais kurios leidžia atlikti daugiau pasisukimų tame pačiame plote. Taip galutinė trajektorija tampa sklandesnė, kliūtys mažiau įtakoja viena kitą, arba galima sumažinti kliūties poveikio zoną.



2.6 pav. Trajektorijos kitimas keičiantis parametrams.

2.3 *Potencialinių laukų algoritmo parametrų nustatymas*

Kadangi potencialų laukų algoritmas labai priklausomas nuo parametrų, kad būtų galima palyginti algoritmus, visų pirma reikia parinkti geriausius parametrus, kurie padėtų rasti trajektoriją. Tam keičiant parametrus 10 000 kartų generuojami atsitiktiniai judėjimo laukai su 5% tikimybe, kad kiekviename iš laukelių gali būti kliūtis (vidutiniškai 3.5 kliūties viename sugeneruotame lauke).

jei $d<r$, tai	$\Delta x = \Delta y = 0$	
jei $r \leq d \leq s + r$, tai	$\Delta x = \alpha(d - r)\cos(\theta)$ ir $\Delta y = \alpha(d - r)\sin(\theta)$	
jei $d > s + r$, tai	$\Delta x = \alpha s \cos(\theta)$ ir $\Delta y = \alpha s \sin(\theta)$	(4)

jei $d<r$, tai $\Delta x = -sign(\cos(\theta))\infty$ ir $\Delta y = -sign(\sin(\theta))\infty$.

jei $r \leq d \leq s + r$, tai $\Delta x = -\beta(s + r - d)\cos(\theta)$ ir $\Delta y = -\beta(s + r - d)\sin(\theta)$.

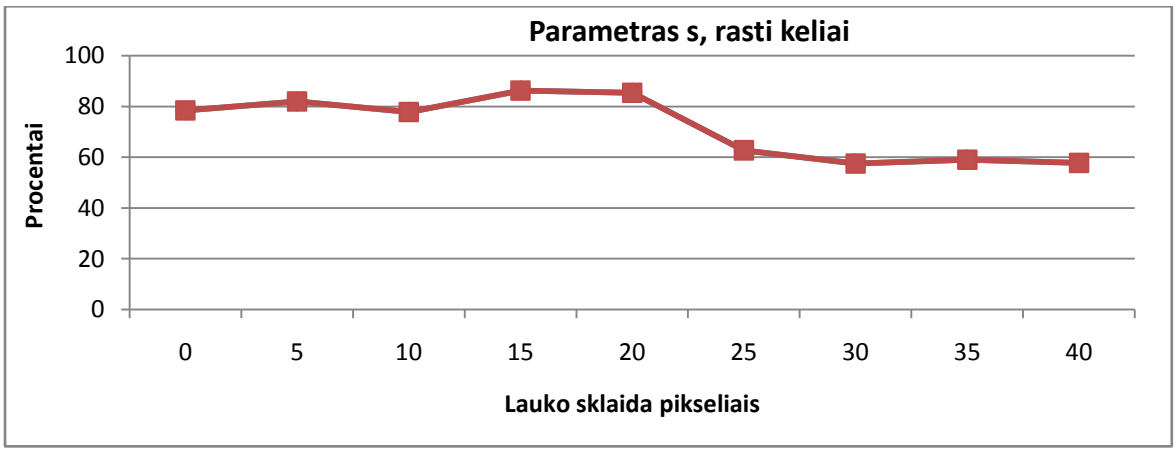
jei $d > s + r$, tai $\Delta x = \Delta y = 0$. (7)

Geriausiems rezultatams pasiekti, keičiami parametrai s ir β. Parametras s - tai kliūties arba tikslo įtakos sklaida (*spread*). Jei tai kliūtis - sklaida parodo, kokiu atstumu nuo roboto esant kliūčiai ji pradeda keisti potencialą, ir paveikia judėjimo trajektoriją. Jei tai tikslo taškas, sklaida parodo, kokiu atstumu nutolus nuo tikslo, pradeda veikti maksimali traukos jėga. Visa tai atsispindi formulėse (4) ir (7).

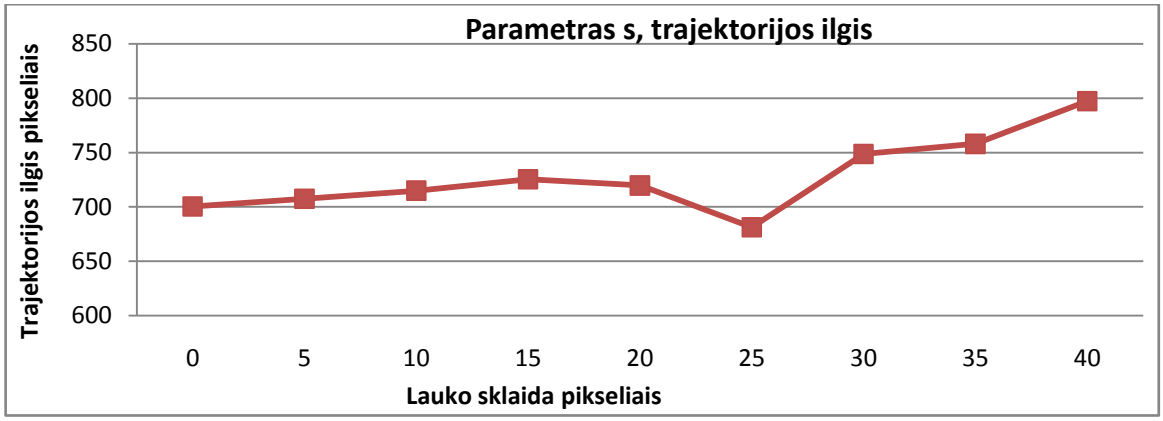
Kitas keičiamas parametras β. Kaip matoma formulėje (7) jis nurodo kokio dydžio bus potencialas, veikiantis robotą, jam patekus į kliūties veikimo zoną. Matant, kad potencialas yra nepakankamas robotui nustumti nuo kliūties, šį parametrą reikia didinti. Jei robotas nustumiamas per stipriai - mažinti. Praktikoje svarbus ne tik pats β parametras, tačiau ir jo bei α parametro (parodančio link tikslo traukiančio lauko stiprį) santykis. Tačiau keičiant β, kartu keičiamas ir santykis, todėl pradžioje nustatomos lygios šių parametrų reikšmes $\alpha = \beta = 2$ ir keičiamas tik stumiančiojo lauko stipris.

Keičiant parametrus gaunami ir lyginami trys skirtingi rezultatai - rastų trajektorijų procentas, vidutinis rastų trajektorijų ilgis bei vidutinis posūkių skaičius trajektorijoje. Kadangi algoritmo svarbiausias tikslas yra rasti optimalią trajektoriją, įvertinant pakeisto parametro efektyvumą pirmiausia atsižvelgiama į tai, koks yra rastų trajektorijų procentas nuo viso skaičiaus sugeneruotų skirtingų laukų. Antras parametras, labiausiai įtakojantis greitą tikslo taško pasiekimą, rastas vidutinis trajektorijų ilgis. Trečiasis - posūkių skaičius.

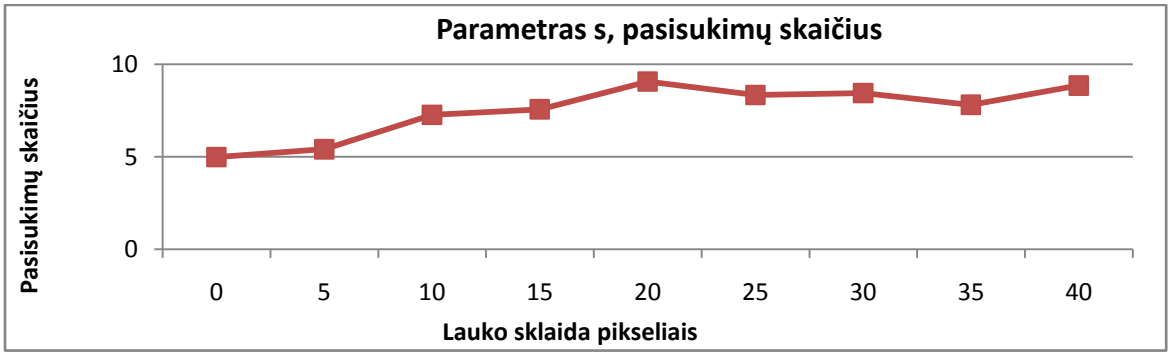
Pateiktuose grafikuose (pav 2.7 - 2.9) galima pastebėti, kaip keičiasi ieškomi dydžiai, keičiant parametrą s - lauko sklaidą. Geriausias rastų kelių procentas buvo ties 5, 15 ir 20 lauko sklaidos pikselių (pav. 3.8); visos kitos prametro reikšmės buvo atmestos, o su šiomis parametro s reikšmėmis gautų trajektorijų ilgiai taip pat buvo palyginami. 3.7 paveiksle matoma, kad geriausi trajektorijos ilgio rezultatai iš likusių verčių yra ties 5 ir 20, tačiau kai s = 5, trajektorija gaunama gerokai tiesesnė - mažiau pasisukimų, taigi sklaidos parametro parinkta vertė yra lygi 5 pikseliams.



2.7 pav. Potencialinių laukų parametrų parinkimo diagrama, rastų kelių procentas.

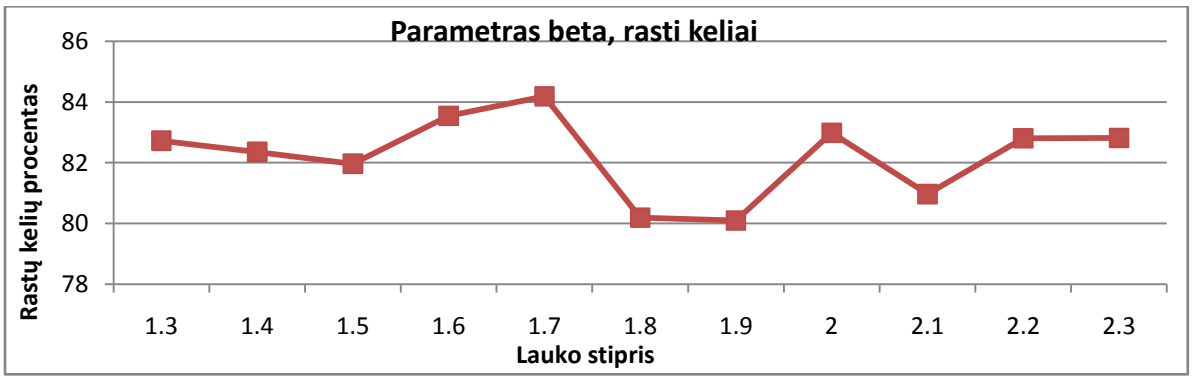


2.8 pav. Potencialinių laukų parametrų parinkimo diagrama, trajektorijos ilgis.

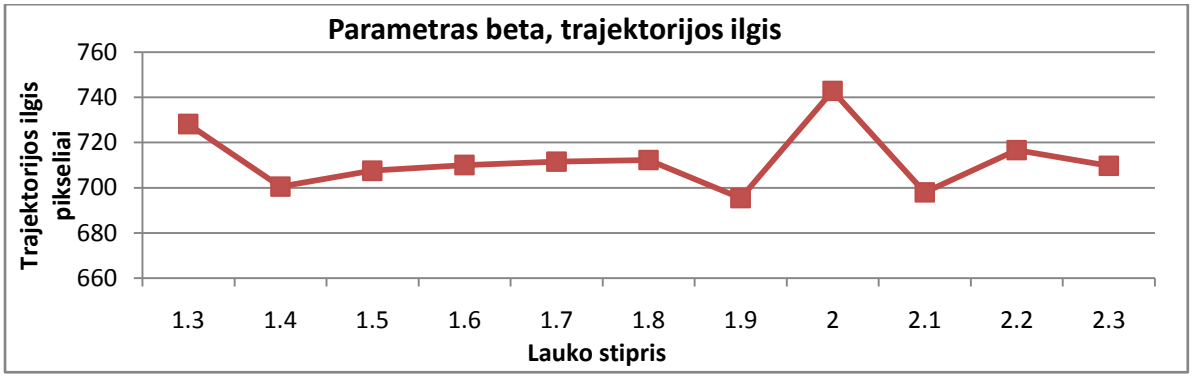


2.9 pav. Potencialinių laukų parametrų parinkimo diagrama, pasisukimų skaičius.

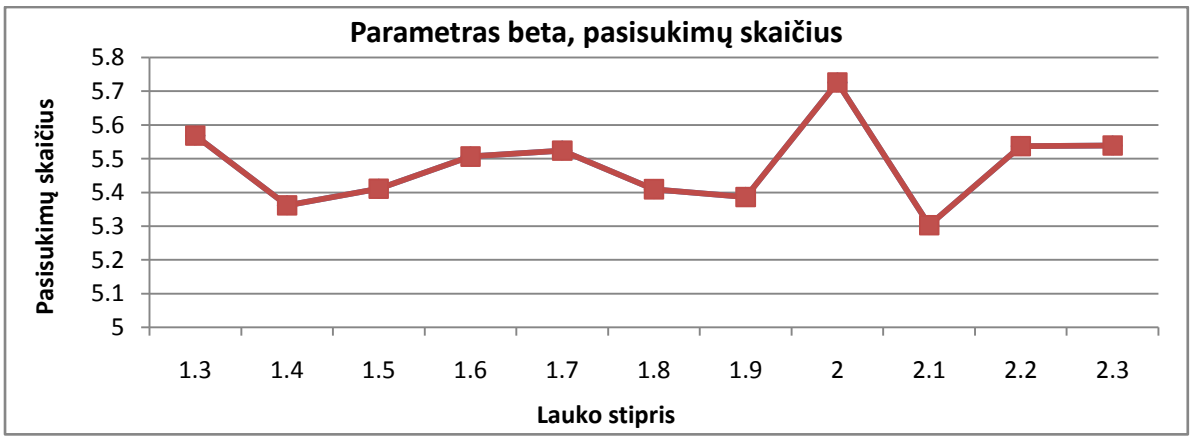
Parinkus ir nustačius parametrą *s*, toliau tokie pat skaičiavimai atliekami keičiant stumiančiojo lauko stiprį - prametrą β. Iš rastų kelių grafiko (pav. 2.10) atmetamos visos parametro reikšmės žemesnės už 82%, ir pagal vidutinės trajektorijos ilgio (pav. 2.11), bei pasisukimo skaičiaus (pav. 2.12) grafikus, matome, kad geriausia parametro reikšmė yra 1.4 (parametras $\alpha = 2.0$).



2.10 pav. Potencialinių laukų parametrų parinkimo diagrama, lauko stipris, rasti keliai.



2.11 pav. Potencialinių laukų parametrų parinkimo diagrama, lauko stipris, trajektorijos ilgis.

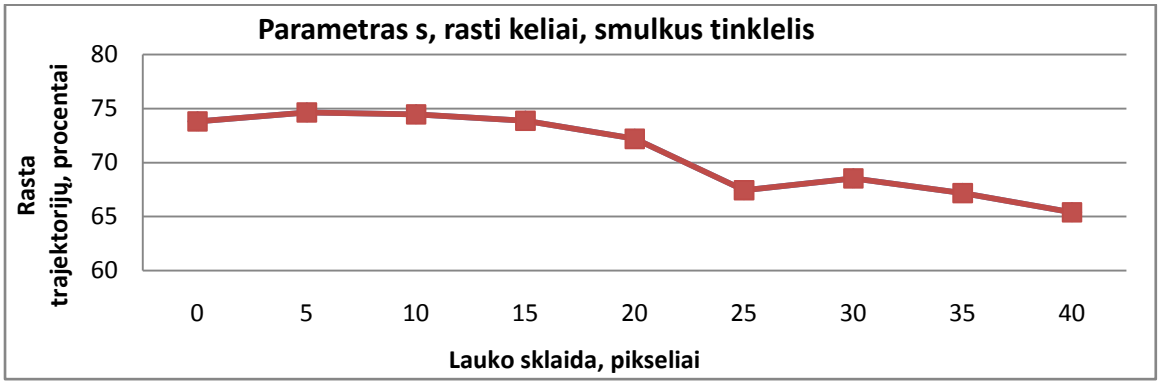


2.12 pav. Potencialinių laukų parametrų parinkimo diagrama, lauko stipris, pasisukimų skaičius.

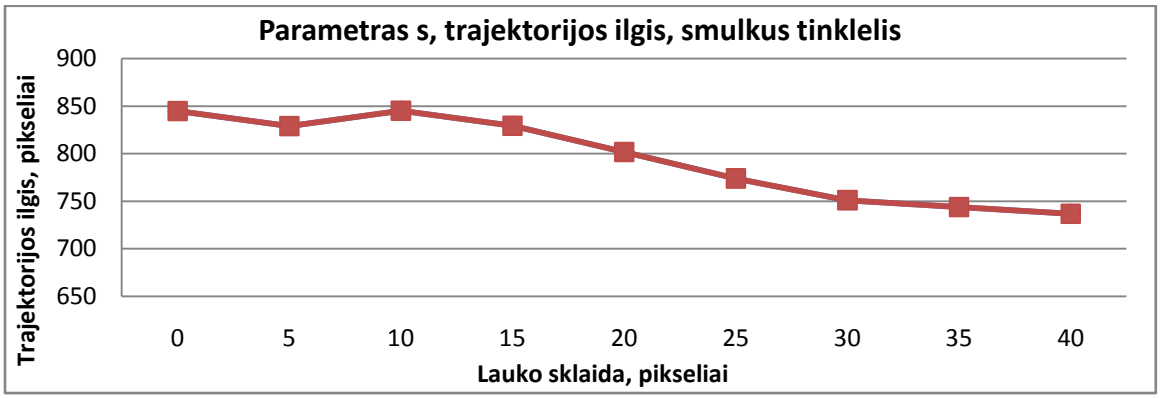
Mažas parinktas sklaidos laukas rodo, kad lauko tinklelis gali būti šiek tiek per mažas ir laukų zonos per didelės, tai verčia mažinti laukų sklaidos lauką, kad kliūtys, esančios arti, leistų dažniau praeiti judančiam objektui. Tai taip pat paaiškina reiškinį, kad kai kuriais atvejais, padidėjus rastų kelių procentui, pailgėja vidutinis trajektorijos ilgis bei posūkių skaičius. Taip atsitinka dėl to, kad kliūtims esant arti viena kitos, robotas turi daug kartų judėt nuo vienos kliūties link kitos, tik po truputį artėdamas tikslo link. Sumažinus sklaidos lauką, tokia žemėlapio

modifikacija tampa nepraeinama ir trajektorijos - nesurastos, tačiau vidutinis rastų trajektorijų ilgis - trumpesnis. To galima išvengti, judėjimo lauko tinklelį susmulkinus.

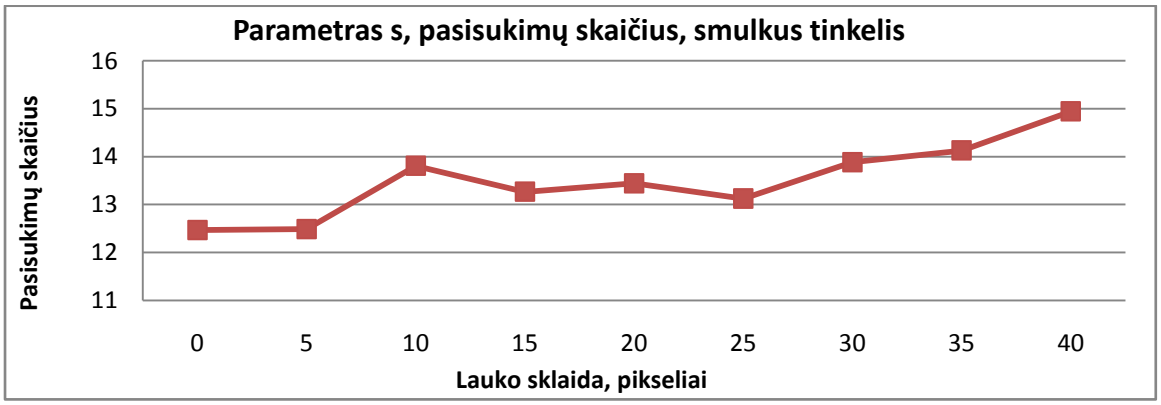
Judėjimo lauko tinklelį padarome smulkesnį keturis kartus - nuo 10x7 iki 20x14 laukelių (70 ir 280 mazgų atitinkamai). Atliekamos tos pačios procedūros potencialų laukų metodui parametrams s ir β nustatyti. Gauti grafikai pateikiami 3.13, 3.14 ir 3.15 paveiksluose. Gauti parametrai skiriasi nuo ankstesnių: s =20 , o β = 1.6.



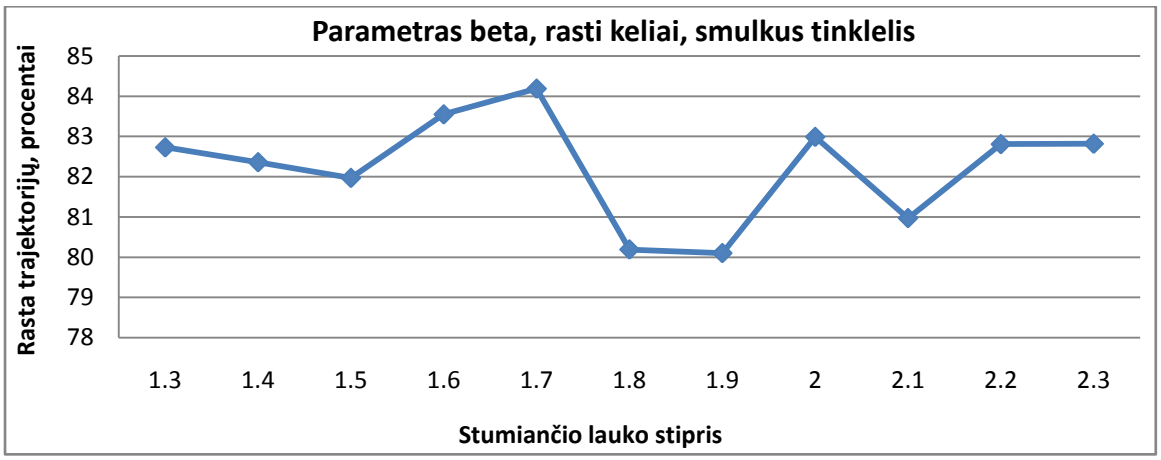
2.13 pav. Potencialinių laukų parametrų parinkimo diagrama esant smulkesniam tinkeliui, rastų kelių procento kitimas keičiant lauko sklaidą.



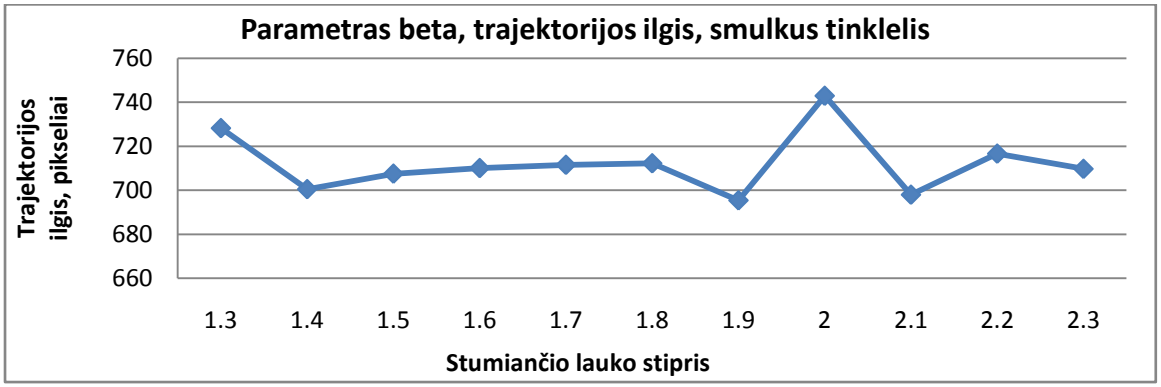
2.14 pav. Potencialinių laukų parametrų parinkimo diagrama, trajektorijos ilgio kitimaskeičiant lauko sklaidą.



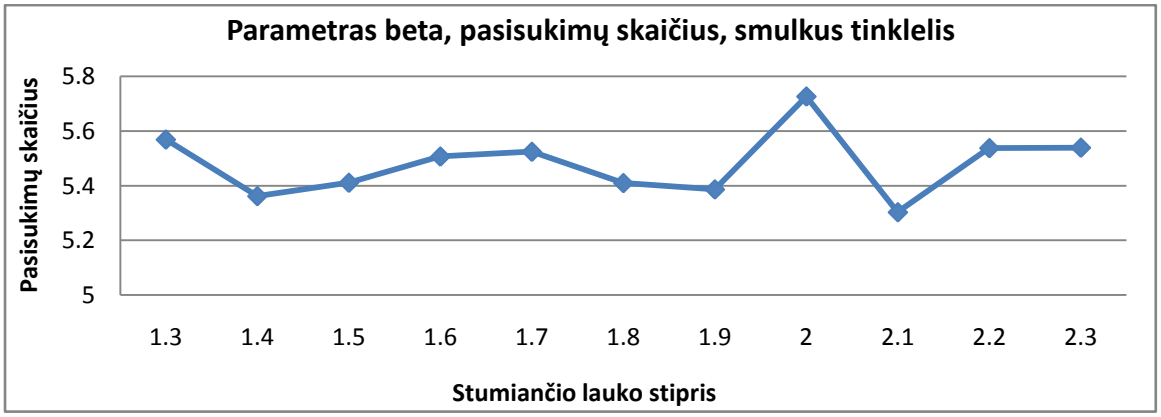
2.15 pav. Potencialinių laukų parametrų parinkimo diagrama, pasisukimų skaičiaus kitimas, keičiant lauko sklaidą.



2.16 pav. Potencialinių laukų parametrų parinkimo diagrama esant smulkesniam tinkeliui, rastų kelių procento kitimas keičiant stumiančio lauko stiprį.



2.17 pav. Potencialinių laukų parametrų parinkimo diagrama, trajektorijos ilgio kitimaskeičiant stumiančio lauko stiprį.



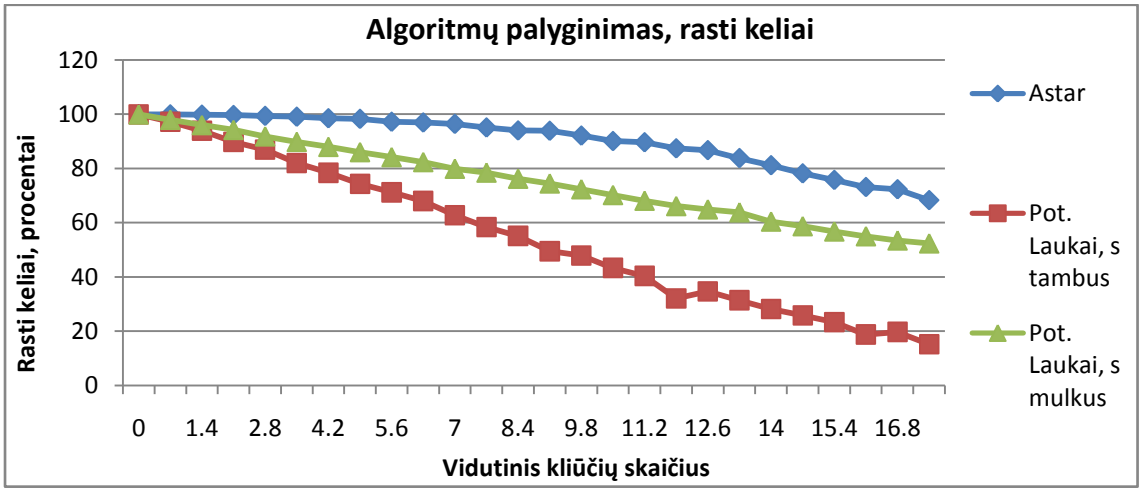
2.18 pav. Potencialinių laukų parametrų parinkimo diagrama, pasisukimų skaičiaus kitimas, keičiant stumiančio lauko stiprį.

2.4 *A* ir potencialinių laukų algoritmų palyginimas*

Parinkus parametrus potencialų laukų algoritmui su stambesniu ir smulkesniu tinkleliu, panašiu metodu šis algoritmas palyginamas su A* algoritmu. Atstitiktiniu būdu sugeneruojama 10 000 skirtingų žemėlapio variantų su vienoda tikimybe žemėlapyje turėti kliūtį,t.y. jei žemėlapį sudaro 70 langelių, tai, generuojant langelį, tikimybė, kad jame bus kliūtis yra keičiama nuo 0 iki 25% (vidutiniškai nuo 0 iki 17.5 kliūties). Atitinkamai smulkiąjame potencialų laukų žemėlapyje iš 280 langelių, taikoma tikimybė nuo 0 iki 6.25% (vidutiniškai nuo 0 iki 17.5 kliūties). Taip užtikrinama, kad kliūčių skaičius visuose žemėlapiuose vienodas. Taip pat naudojant parametrą r potencialų laukų algoritme nurodomas kliūties spindulys yra tokio paties dydžio kaip ir A* algoritmo žemėlapyje.

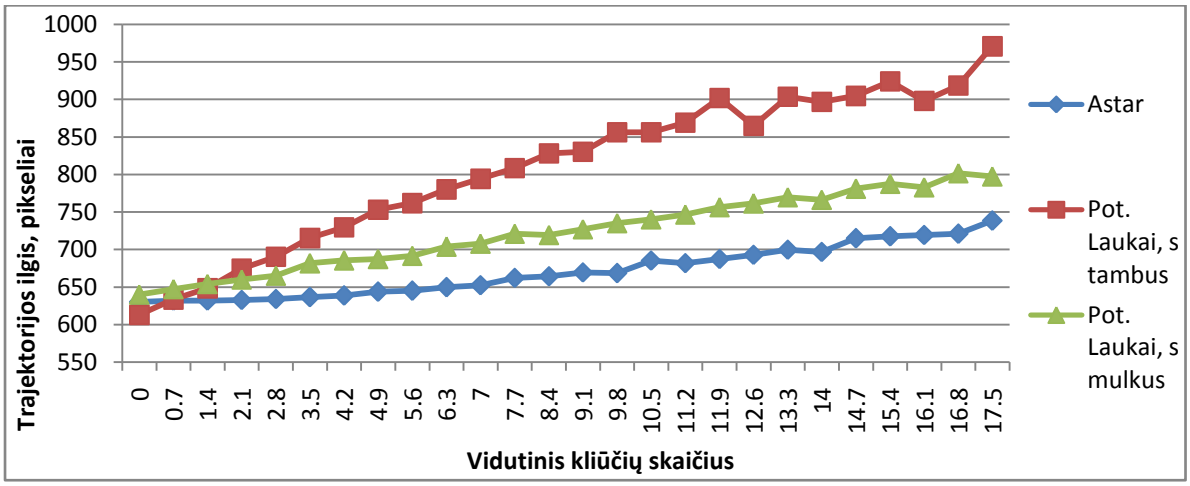
Palyginus algoritmų surastų trajektorijų procentus (pav. 2.19) galima teigti, kad A* algoritmas geriausiai atliko užduotį. Taip pat, galima pastebėti, kad esant mažam kliūčių skaičiui (0 - 7, iki 10%), rastų kelių procentas mažėja nežymiai, toliau pradeda mažėti didesniais intervalais. Tačiau A* algortimo struktūra užtikrina, kad jeigu kelias iki tikslo yra, tai jis bus rastas, taigi iš esmės potencialų laukų metodo rezultatai galėtų būti tik nežymiai geresni dėl dėkingiau sugeneruotų skirtingų žemėlapių. Tokiu atveju tikslinga būtų lyginti rastų trajektorijų ilgius.

Kalbant apie potencialų laukų algoritmą, daugiau trajektorijų buvo rasta sugeneravus smulkesnį žemėlapio tinklelį, tačiau lyginant su A*, rezultatai gerokai prastesni. Galima daryti prielaidą, kad smulkinant tinklelį dar keletą kartų, būtų galima tikėtis panašių rezultatų į A* algoritmo rezultatus, tačiau šiuo momentu tikslinga patikrinti rastų trajektorijų ilgius (2.20 pav.). Taip pat iš grafiko matome, kad rezultatai potencialų laukų metodu labiau tiesiški, kinta tolygesniais intervalais.



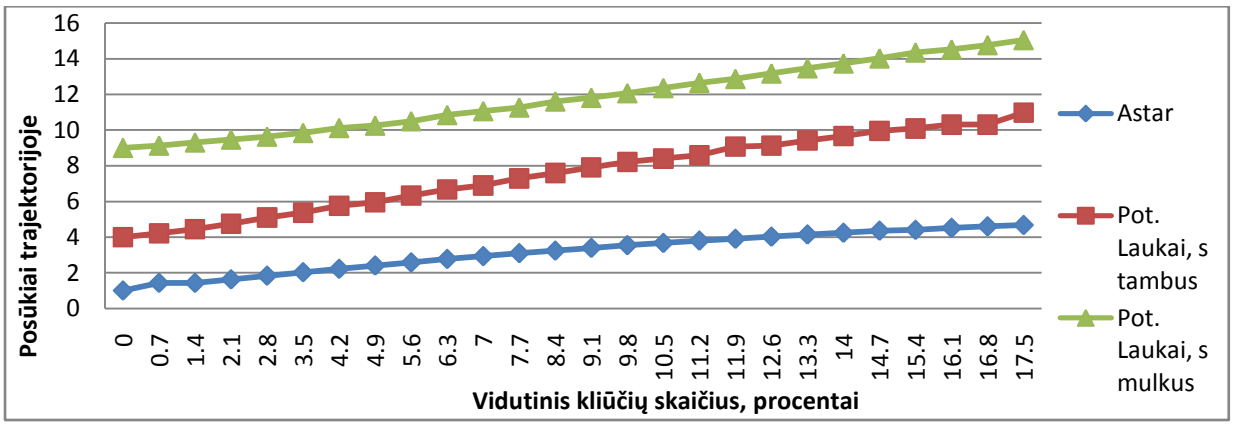
2.19 pav. Algoritmų rastų kelių, kintant vidutiniam kliūčių skaičiui palyginimas.

Grafike, kuris pateiktas 2.20 paveiksle, galima stebėti, kaip kinta rastos trajektorijos vidutinis ilgis, keičiantis kliūčių skaičiui, tarp skirtingų algoritmų. Matome, kad A* algoritmo surasti vidutiniai trajektorijų ilgiai yra trumpiausi. Lyginant potencialinių laukų surastus trajektorijų ilgius ir šiuo atveju smulkaus tinklelio rezultatai geresni - surastos trajektorijos trumpesnės. Taip yra dėl to, kad į kliūtį pradedama reaguoti anksčiau (β parametras didesnis), be to, kliūtims esant arti viena kitos, surandamas kelias aplenkiant jas abi, o ne einant per tarpą tarp jų, kur prasideda švytavimai. Tačiau matome, kad nesant kliūčių arba kai jų labai mažai, potencialų laukų metodu randamas netgi trumpesnis kelias, negu naudojant A* algoritmą. Galima daryti išvadą, kad potencialų laukų algoritmas veikia geriausiai tada, kai yra mažas kliūčių skaičius, arba reikia išvengti tik vienos kliūties.



2.20 pav. Algoritmų vidutinių trajektorijų ilgio, kintant vidutiniam kliūčių skaičiui palyginimas

Paskutiniąjame grafike, 2.21 paveiksle, pavaizduota, kiek kartų keičiamas kampas surastose trajektorijose. Tai nusako trajektorijos sudėtingumą. Mažiausiai kampą keičia A* algoritmo trajektorijos, taigi galima daryti išvadą, kad jos yra paprasčiausios. Tai lengvai paaiškinama, kadangi judėjimas gali vykti tik aštuoniomis kryptimis. Tuo tarpu potencialų laukų atveju - krypties vektoriaus kampas gali įgyti 1800 skirtingų pozicijų, nes tikslumo dėlei potencialų laukų algoritmuose buvo atfiltruoti kampai, kurių pokyčiai neviršijo 0,2°. Galima neabejoti, kad dar labiau smulkinant potencialų laukų tinklelį trajektorijos gautųsi dar sudėtingesnės. Kita vertus, bendras pasisukimų skaičius galėtų ir sumažėti (arba didėti mažesniais intervalais), nes vis dažniau būtų išvengiama švytavimo problemos. Taip pat galima daryt išvadą, kad roboto valdymas būtų sudėtingesnis esant tokiai sudėtingai trajektorijai.



2.21 pav. Algoritmų vidutinių trajektorijų ilgio, kintant vidutiniam kliūčių skaičiui palyginimas

2.5 *Tyrimų dalies rezultatai ir išvados*

Šioje darbo dalyje buvo ištirti A* ir potencialų laukų algoritmai. Potencialų laukų algoritmo atveju, buvo ištirtos dvi skirtingos žemėlapio modifikacijos su stambesniu ir smulkesniu tinkleliu. A* algoritmui buvo naudojamas žemėlapis, su stambesniu tinkleliu. Buvo išmatuoti ir palyginti jų surastų trajektorijų procentai, vidutiniai rastų trajektorijų ilgiai, bei trajektorijos sudėtingumas, matuojant pasisukimų skaičių trajektorijoje. Tam, kad būtų galima palyginti algoritmus teko, nustatyti parametrus potencialų laukų algoritmui.

Pagal gautus rezultatus, didžiausią trajektorijų procentą pavyko rasti naudojant A* algoritmą, o smulkesnio tinklelio žemėlapyje naudotas potencialų laukų algoritmas surado daugiau trajektorijų, negu tas pats algoritmas stambesnio tinklelio žemėlapyje.

A* algoritmu taip pat buvo gautos trumpiausios vidutinės trajektorijos, išskyrus žemėlapius su labai nedideliu kliūčių skaičiumi, arba žemėlapius, kur nėra kliūčių - šiuo atveju geresni rezultatai gauti naudojant potencialų laukų algoritmą ir stambaus tinklelio žemėlapį.

Paprasčiausios trajektorijos su mažiausiu pasisukimų skaičiumi buvo rastos naudojant A* algoritmą. Potencialų laukų metodu, naudojant stambų tinklelį, vidutiniškai trajektorijoje reikėjo mažiau pasisukimų, negu judant žemėlapyje su smulkesniu tinkleliu.

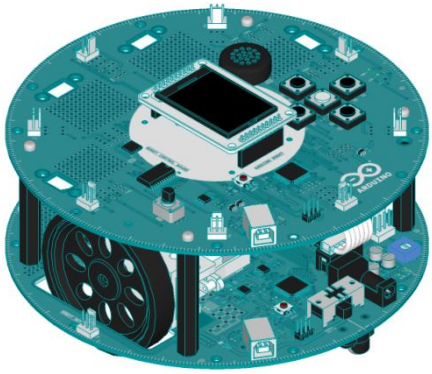
Pagal atlikto tyrimo duomenis galima daryti išvadas, kad potencialų laukų algoritmo geriausios savybės atsiskleidžia jį naudojant žemėlapiuose su smulkesniu tinkleliu ir tokiomis sąlygomis, kai reikia apvažiuoti nedidelį kliūčių skaičių. Kaip jau minėta literatūros analizėje, pastebimas šio metodo trūkumas - švytavimų ir aklavietės problemos, ypač, esant didesniam kliūčių skaičiui. Tačiau gali būti, kad žemėlapio tinklelį smulkinant dar labiau, galima parinkti tokius parametrus, kurie leistų rasti trajektorijas, trumpesnes negu naudojant A* algoritmą, netgi esant sąlyginai didesniam kliūčių skaičiui. Visgi mažai tikėtina, kad trajektorijų radimo atžvilgiu, šis algoritmas būtų toks patikimas kaip A*. Taipogi, norint pasiekti gerų rezultatų, atitinkamam

kliūčių skaičiui žemėlapyje reikėtų iš naujo ir labai tiksliai nustatinėti parametrus, kad algoritmas būtų efektyvus. Net jei ir pavyktų gauti gerus trajektorijos ilgio rezultatus, iš surinktų duomenų tyrimo metu galime tvirtai teigti, kad trajektorija būtų gerokai sudėtingesnė, ir programuojant robotą reikėtų įdėti daugiau pastangų. Dėl šių priežasčių mobilaus autonominio roboto programavimui, atlikus tyrimą pasirenkamas A* algoritmas.

3 PROJEKTINĖ DALIS

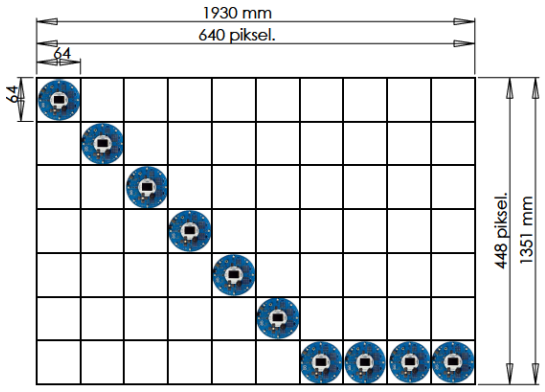
Roboto programavimui tikslo taškui pasiekti buvo pasirinktas Arduino robotas (*Official Arduino Robot*). Šioje dalyje aprašomi projektui naudojami kiti komponentai, navigacijos, duomenų perdavimo ir programavimo principai.

Optimalaus kelio paieškai buvo pasirinktas Arduino robotas (Pav. 3.1).Robotui valdyti C# programavimo kalba, naudojant Microsoft Visual C# 2010 Express, sukuriama programa, kurios dalis - A* algoritmas - suranda trajektoriją tikslo taškui pasiekti. Duomenims apie žemėlapį ir roboto padėtį gauti, naudojama papildoma įranga.



3.1 pav. *Arduino* robotas, naudojamas tikslo taškui pasiekti [15].

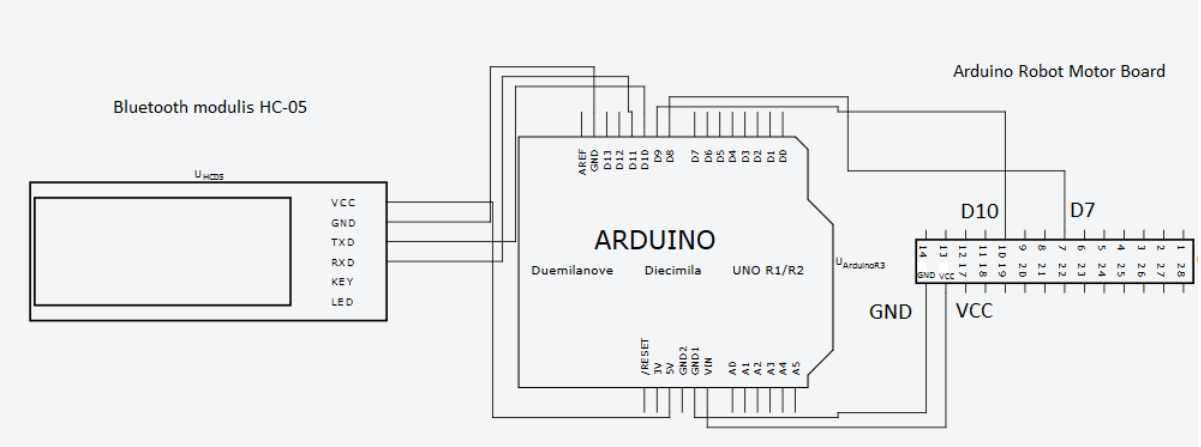
Roboto navigacijai - padėties nustatymui ir orientavimuisi tarp kliūčių - naudojama tinklo kamera[15], kuri pakabinama 245 cm aukštyje. Šia kamera taip pat atpažįstamos kliūtys, išsidėsčiusios žemėlapyje. Tokiame aukštyje pakabinta kamera, kurios raiška 640x480 taškų, aprėpia vaizdą, kurio matmenys ant grindų (ploto, kuriame bus važinėjama) yra 1930x1450 mm. Kadangi roboto diametras 190 mm - jam važinėti sudaromas tinklelis 10x7, vieno langelio matmuo pikseliais 64x64, arba 19.3x19.3 cm. Sudarytas tinklelis pavaizduotas 3.2 paveiksle.



3.2 pav. Žemėlapio laukas, matmenys ir tinkelis.

Kamera sujungta su kompiuteriu, kur programa skaičiuoja roboto buvimo vietą, pasisukimo kampą, nustato kliūtis ir saugo informaciją apie trajektoriją, kuria reikia sekti. Iš kameros gavus informaciją, duomenys perduodami *bluetooth* ryšiu į Arduino Uno valdiklį, kurio

išėjimai, sujungti su įėjimais Arduino robote. Taip yra todėl, kad Arduino robotas turi dvi skirtingas plokštes su skirtingais procesoriais ir nuosekliąja jungtimi (*serial port*) dalijasi duomenimis tarp procesorių[14]. Kadangi reikalinga laisva nuoseklioji jungtis duomenims iš kompiuterio priimti, naudojamas papildomas Arduino Uno R3 valdiklis. Taip pat naudojamas *bluetooth* modulis HC-05.*Bluetooth* modulio HC-05 ir valdiklio Arduino Uno sujungimo su roboto variklių valdymo plokšte schema parodyta 3.3 paveiksle.

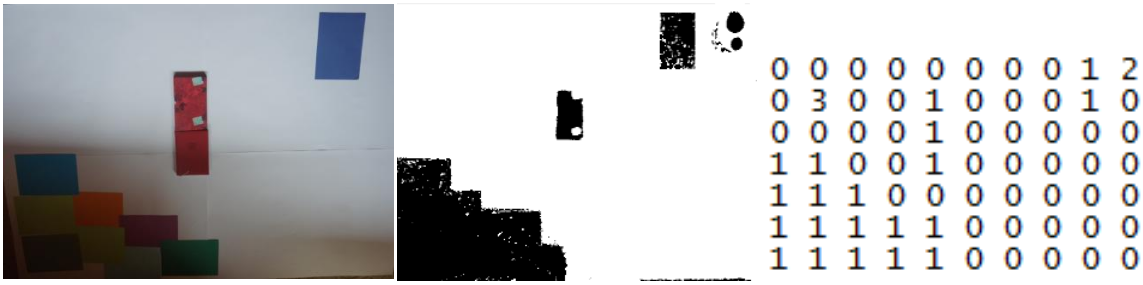


3.3 pav. Bluetooth modulio, Arduino valdiklio ir Arduino roboto sujungimo schema.

3.1 Kliūčių aptikimas

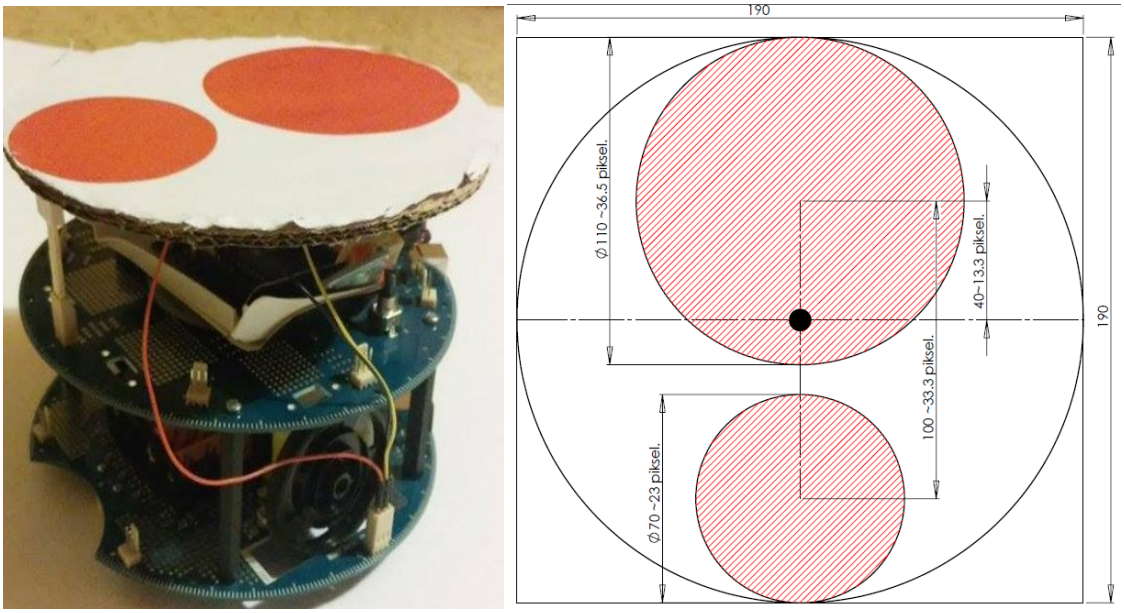
Kadangi judėjimo laukas statiškas, kliūčių atpažinimas ir trajektorijos sudarymas vyksta vieną kartą, prieš robotui pradedant važiuoti. Vaizdų atpažinimui ir apdorojimui naudojama Emgu CV biblioteka, pritaikyta C# programavimo kalbai.

Programos veikimo metu, iš tinklo kameros rodomo vaizdo padaroma nuotrauka, kuri bus panaudojama kliūčių atpažinimui ir žemėlapio sudarymui. Kadangi plotas, kuriame važiuoja robotas, yra baltas, kliūtims sudaryti naudojami spalvoto popieriaus lapai arba kitokie ne baltos spalvos objektai. Tuomet, panaudojant HSV (*hue, saturation, value*) spektrą, atfiltruojama spalva, kuri kameroje atpažįstama kaip balta, o visos kitos(nebaltos) spalvos tampa juodomis. Sekančiame žingsnyje filtruotas vaizdo paveikslėlio formatas pakeičiamas į juodai-baltą ir paeiliui dalinamas į 64x64 pikselių laukelius. Pamatuojama kiekvieno langelio vidutinė spalvos vertė nuo 0 iki 255, kur 0 - visiškai balta spalva, o 255 - visiškai juoda. Jeigu gauname, kad langelyje yra 10% arba daugiau juodos spalvos - visas langelis pažymimas kaip kliūtis. Tokiu būdu gaunama skaičių matrica, tokia pati, kaip naudojama tyrimų dalyje, A* algoritmų žemėlapiui sudaryti. Spalvų filtravimas ir žemėlapio sudarymas pavaizduotas 3.4 paveiksle.



3.4 pav. Žemėlapio sudarymas.

Kad būtų galima nustatyti roboto buvimo vietą, jis pažymimas specialiu žymekliu. Žymeklis sudarytas iš dviejų raudonos spalvos skritulių. Mažesnis skritulys žymi roboto priekį, o didesnis - galą. Vaizdo kamerai atpažinus apskritimus, kad būtų apsisaugota nuo neteisingos informacijos ar pasitaikančių panašių formų kliūčių, patikrinama, ar apskritimai patenka į tolerancijos ribas pagal savo dydį ir atstumą tarp jų. Jeigu gauti duomenys neatitinka pavaizduotų 3.5 paveiksle, vaizdo atpažinimas vykdomas iš naujo. Roboto centro koordinatė yra išskaičiuojama, atpažinus šiuos apskritimus. Kaip atrodo žymekliai vaizduojama 3.5 paveiksle.

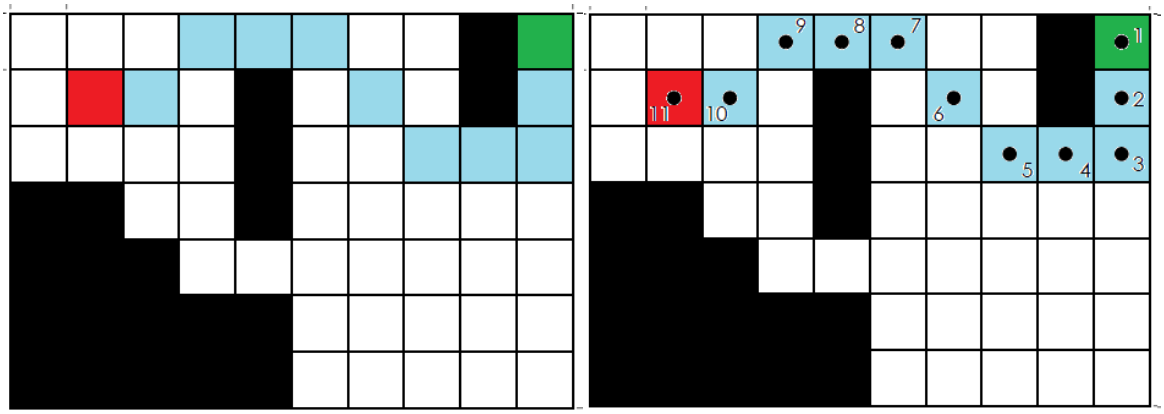


3.5 pav. Arduino robotas ir žymekliai

3.2 Trajektorijos sudarymas ir roboto navigacija

Atpažinus kliūtis, įvykdomas A* algoritmas ir surandama trajektorija iki tikslo taško. Trajektorija surandama išsaugant eilutės ir stulpelio numerį sąraše, nuo tikslo taško atgal. Žinoma, kad tokiu sąrašu vadovaudamasis robotas negali rasti kelio iki tikslo, taigi, šis sąrašas

turi būti perdaromas. Dėl šios priežasties eilutės ir stulpelio numeris perskaičiuojamas ir pakeičiamas į koordinates X ir Y ašims, išreikštas pikseliais. Pikseliais tai daryti patogiausia, nes iš tinklo gauta informacija apie roboto buvimo tašką iškart gaunama šiuo dydžiu, todėl nebereikia papildomai šių koordinačių konvertuoti į milimetrus. Perskaičiavus visą sąrašą, sudaromas naujas sąrašas su koordinatėmis, kurias robotui reikia aplankyti - tai tarsi tarpiniai tikslai, kurie veda iki galutinio tikslo. Vienas tarpinis tikslas nurodo vieno langelio iš sudarytos trajektorijos centro koordinates pikseliais. Žemėlapis su tarpiniais tikslais nuo 1 iki 11 pavaizduotas 3.6 paveiksle.

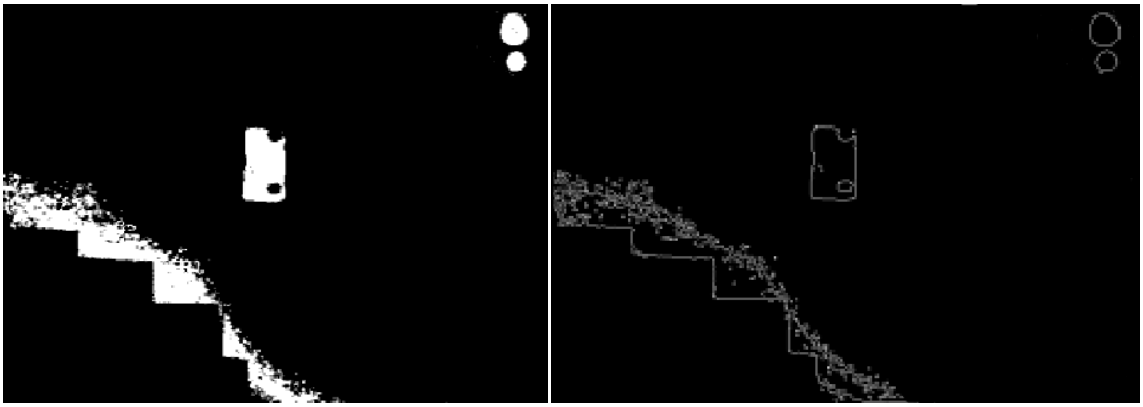


3.6 pav. Iš eilės sunumeruoti tarpiniai tikslai nuo starto iki galutinio tikslo.

Kad robotas galėtų taisyklingai važiuoti užduota trajektorija ir pasiekti tarpinius tikslus, programa turi žinoti jo padėtį judėjimo erdvėje. Kadangi roboto ratai gali prasisukti, paviršiuje gali pasitaikyti nelygumų ar panašių problemų, naudojant enkoderius roboto poziciją nustatyti tiksliai būtų sudėtinga. Jeigu momentinės paklaidos ir nebūtų didelės, jos kauptųsi ir esant ilgesnei trajektorijai didėtų tikimybė robotui nepasiekti užduoto tikslo arba kelyje sutikti kliūtį, kuri jau yra žinoma ir taip. Naudoti GPS sistemos taip pat nėra galimybių, kadangi jos sunkiai veikia uždarose patalpose, o reikalingo tikslumo nepavyktų pasiekti. Tokia sistema tiktų tyrimą darant atviroje erdvėje ir dideliame plote, kur GPS paklaida būtų sąlyginai mažesnė.

Šiame baigiamajame projekte nuspręsta roboto navigacijai naudoti vaizdo kamerą ir vaizdų atpažinimą. Roboto pozicijai ir pasisukimui nustatyti naudojamas jau anksčiau paminėtas žymeklis su skrituliais (pav. 3.5). Raudonos spalva filtruojama naudojant HSV spektrą, parinkus atitinkamas H, S ir V reikšmes. Taip gaunamas juodai baltas vaizdas, kur raudona spalva pažymėta baltai, o neraudonos spalvos - juodai. Tuomet, panaudojus *Canny* funkciją, skirtą kontūrams aptikti, surandami apskritimai. Kadangi atstumai tarp apskritimų bei apskritimų matmenys žinomi ir gali būti išskaičiuojami pikseliais, tai leidžia atmesti neteisingus surastus duomenis, net jeigu ir pasitaikytų kitų raudonų objektų ir panašaus dydžio apskritimo kontūrų. Taip užtikrinama, kad duomenys gaunami iš kameros apie roboto padėtį yra gauti užfiksavus

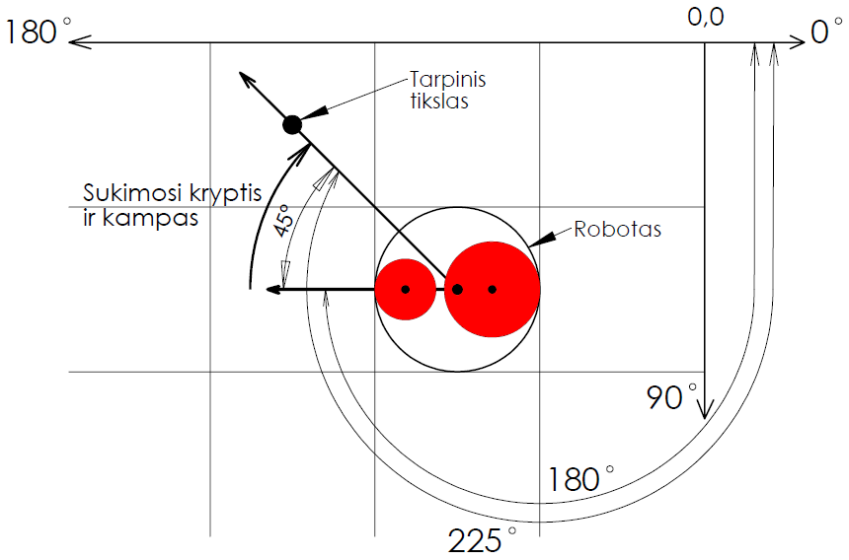
žymeklį pritvirtintą ant roboto. Vaizdai, gauti atfiltravus raudoną žymeklio spalvą, bei objektų kontūrai gauti naudojant *Canny* funkciją, pavaizduoti 3.7 paveiksle. Eksperimento atlikimo metu nustatytos HSV spektro minimalios reikšmės raudonai spalvai filtruoti buvo: (0, 105, 35), o maksimalios reikšmės: (180, 255, 250). Atitinkamai minimalios reikšmės baltai spalvai: (0, 0, 0), o maksimalios: (180, 140, 255). Maksimali vienos dedamosios vertė 255. Tokie dideli reikšmių diapazonai gaunami dėl prasto apšvietimo, ir žemos kameros spalvų atpažinimo kokybės.



3.7 pav. Raudonos spalvos filtravimas ir kontūrų suradimas *Canny* metodu.

Kadangi robotas nėra holonomiškai apribotas, jam valdyti pakanka trijų skirtingų komandų: važiuoti tiesiai, sukti į kairę ir sukti į dešinę. Programoje tai išsprendžiama nusiunčiant atitinkamos reikšmės simbolį: 8, 4 arba 6.

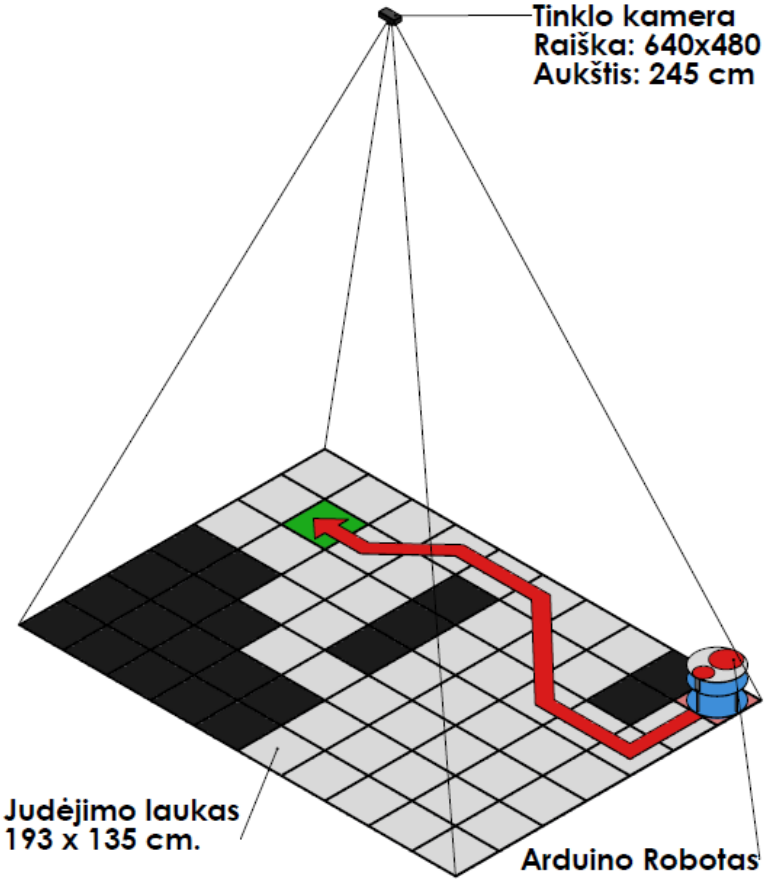
Kai iš kameros gaunama roboto buvimo padėtis, pradedama iš eilės judėti link pirmo neįvykdyto tarpinio tikslo. Programoje pirmiausia patikrinama, ar robotas jau yra tarpiniame tiksle. Jei ne, patikrinama roboto orientacija edvėje, ar jis yra atsisukęs į tarpinio tikslo tašką. Jei robotas nukrypęs nuo taikinio, tikrinama, kokiu kampu nuo roboto taškas yra orientuotas. Tam nubrėžiama menama atkarpa tarp roboto centro bei tikslo taško ir pamatuojamas kampas nuo 0° ašies. Taip pat pamatuojama roboto orientacija edvėje, naudojant menamą atkarpą, einančią per du jo žymeklyje esančius skritulių centrus. Skirtumas tarp šių kampų ir yra ieškomas dydis, kuriuo robotas turi pasisukti, kad būtų orientuotas tarpinio tikslo kryptimi, kampo radimas pavaizduotas schemoje 3.8 paveiksle.



3.8 pav. Roboto orientavimas tarpinio tikslo link.

Testuojant roboto važiavimą, buvo pastebėta, kad tik dalį trajektorijos robotas įveikia taisyklingai, toliau vykdydamas komandas, kurios neveda trajektorijos taškais. Taip nutiko dėl to, kad robotas komandas gauna per vėlai - važiuojant dideliu greičiu kamera sunkiau užfiksuoja buvimo koordinates ir orientaciją, todėl informacija vėluoja. Mažinant roboto važiavimo greitį, didesnė trajektorijos dalis buvo įvykdoma teisingai, tačiau robotas ne visuomet išvengdavo kliūčių, o tarpiniai tikslai taip pat būdavo pravažiuojami. Kadangi pakankamai sumažintį greitį neleido roboto pavara, buvo priimtas sprendimas, kad robotas važiuotų trumpais impulsais. Nustačius reikiamą judėjimo kryptį, į roboto motorus perduodamas signalas trumpą laiko tarpą (25 ms) važiuoti tiesiai, suktis į kairę arba dešinę. Po kiekvieno tokio ėjimo, iš naujo tikrinamos roboto koordinatės ir veiksmai atliekami tol, kol surandami visi tarpiniai tikslai ir pasiekiamas galutinis.

Atliekant bandymą robotas 10 kartų važiuoja ta pačia trajektorija link nustatyto tikslo. Roboto starto pozicija ir pasisukimo kampas vienodas. Robotui važiuojant, skaičiuojama, kiek kartų programoje suveikė važiavimo ciklas, t.y. kiek išvis ėjimų - pasisukimų ir pavažiavimų tiesiai robotas padarė. Taip pat matuojamas laikas, kiek robotas užtruko pasiekti galutinį tikslo tašką. Surinkti duomenys pateikiami 3.1 lentelėje. Roboto trajektorijos tikslumo paklaidos buvo nustatytos remiantis vieno žingsnio pasisukimo ar pavažiavimo dydžiu. Svarbu, kad pasisukimo kampas nebūtų didesnis už leistiną paklaidą taikantis važiuoti tiesiai, nes kitaip robotas galėtų niekaip nepasisukti tinkamu kampu. Programoje buvo nustatyta pasisukimo paklaida 5° (roboto pasisukimo žingsnis ~3.5°), o skaitoma, kad į tarpinį tikslą robotas atvyko, jei jo centro taškas yra nutolęs nuo tarpinio tikslo taško 9 mm (3 pikselių) ar mažesniu spinduliu. Eksperimento atlikimo schema vaizduojama 3.8 paveiksle.



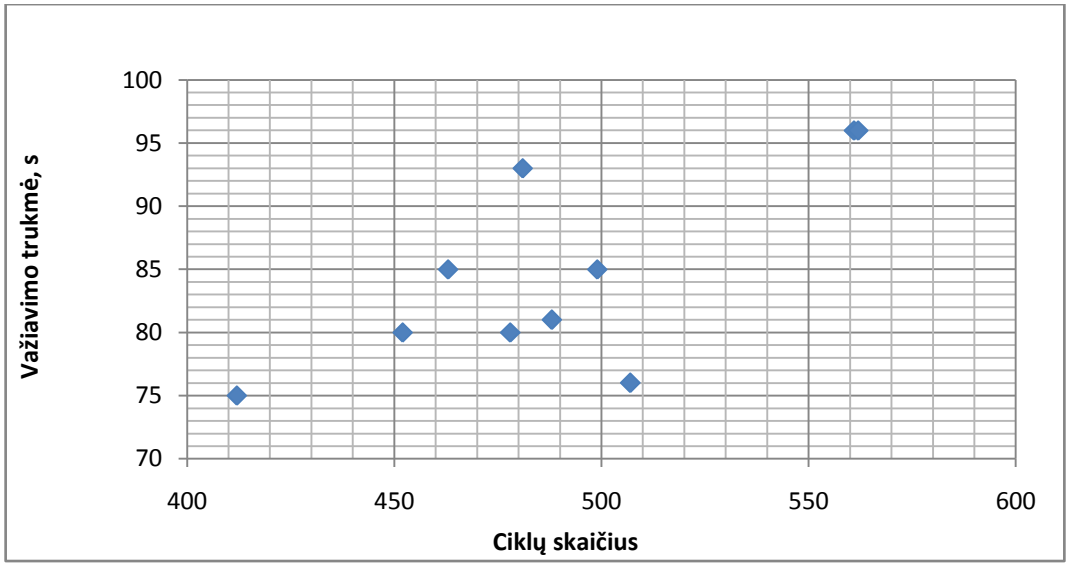
3.8 pav. Eksperimento atlikimo schema.

3.1 lentelė. Eksperimento rezultatai

Važiavimo nr.	Ciklų sk.	Važiavimo trukmė, sek.	Ciklų sk./važ. trukmė
1	561	96	5.84
2	488	81	6.02
3	507	76	6.67
4	478	80	5.98
5	499	85	5.87
6	412	75	5.49
7	562	96	5.85
8	481	93	5.17
9	463	85	5.45
10	452	80	5.65
Vidurkis:	490.3	84.7	5.80

Iš lentelėje pateiktų duomenų, matome, kad nors robotas ir įveikinėjo tą pačią trajektoriją, važiavimo trukmė buvo skirtinga. Taip galėjo nutikti dėl to, kad spalvos filtruotos buvo tik vieną kartą, prieš patį pirmą važiavimą. Kadangi važiavimo metu buvo naudojamas tik natūralus apšvietimas, jo kitimas įtakojo rezultatus. Ilgiau užtrukdavo ne tik programos veikimas, kol tiksliai surandami roboto žymeklio skritulių kontūrai, tačiau ir jų padėtis nustatoma su skirtinga

paklaida. Kadangi apskritimai arti vienas kito, robotui dažnai reikėdavo keletą kartų pataisyti pasisukimo kampą, kad būtų orientuotas tarpinio tikslo link. Diagramoje, kuri pavaizduota 3.9 paveiksle, matome, kad važiavimo trukmė ir ciklų skaičius nėra priklausomi tiesiškai. Galima daryti išvadą, kad nuokrypius sukėlė aplinkos įtakoti veiksniai (apšvietimas), įrangos(prastas spalvų atpažinimas, maža raiška) bei programos netobulumas. Roboto pozicionavimo tikslumui įtakos turėjo jo aukštis, todėl kameroje vaizdas matomas su perspektyva, o ir projektuojant roboto žymeklio plokštumą ant judėjimo plokštumos, gaunama nevisai reali roboto pozicija. Neskaitant šios paklaidos, pikseliais suskaičiuojant trajektorijos ilgį, iš vaizdo kameros, gaunama, kad robotas idealiai įveikęs trajektorją nukeliautų 2160 mm ilgio atstumą.



3.9 pav. Važavimo trukmė ir ciklų skaičius.

4 REZULTATAI IR IŠVADOS

Šiame baigiamajame projekte buvo programuojamas Arduino robotas, kuris sėkmingai apvažiuoja kliūtis ir suranda tikslo tašką. Kad tikslas būtų sėkmingai pasiektas, buvo įgyvendintos tokios užduotys:

1. Iš viso buvo apžvelgti ir aprašyti aštuoni algoritmų tipai. Daugiausia dėmesio skirta A*, Dijikstra, D*, ir potencialų laukų algoritmams. Buvo nustatytos šių algoritmų pritaikymo galimybės, stipriosios ir silpnosios pusės.Taip pat buvo aprašyti virtualaus jėgų lauko, dirbtinių neuroninių tinklų metodai, RNA (*robot navigation ants*) ir MSAC (*multi-scout ants cooperation*) algoritmai.
2. Atlikus literatūros analizę,iš surinktos informacijos pastebėta, kad A* ir Dijkstra algoritmai yra panašūs. Pagrindinis skirtumas tarp jų yra tas, kad A* algoritmas paieškos metu plečiasi viena kryptimi tikslo link, tuo tarpu Dijkstra algoritmas - visomis kryptimis. Šiuo atveju Dijkstra algoritmas būtų naudingesnis kai žemėlapyje yra keletas tikslų, ir reikia rasti arčiau esantį. Esant vienam tikslui trumpiau trunka A* algoritmo skaičiavimai. Taip pat nustatyta, kad potencialų laukų algoritmas skaičiuoja trajektoriją judėjimo metu, todėl neatlieka jokių nereikalingų skaičiavimų, ir paties algoritmo greitaveika yra dar geresnė. D* algoritmas yra A* algoritmo atmaina, judėjimui dinamiškoje aplinkoje.

   Nustačius stipriąsias ir silpnąsias algoritmų savybes buvo pasirinkti du algoritmai, kurie galėtų būti pritaikomi autonominio mobilaus roboto programavimui statiškoje aplinkoje - potencialų laukų ir A* algoritmai.
3. A* ir potencialų laukų algoritmai buvo suprogramuoti C# programavimo kalba naudojant *Microsoft Visual C# 2010 Express* programinę įrangą. Buvo pastebėta, kad potencialų laukų algoritmas labai priklausomas nuo žemėlapio tinklelio smulkumo, todėl šis algoritmas buvo tiriamas naudojant smulkaus ir stambaus tinklelio žemėlapius. Suprogramavus algoritmus buvo tiriama kaip esant atitinkamam kliūčių skaičiui į 10000 sugeneruotų skirtingų žemėlapių vidutiniškai sėkmingai trajektoriją suranda programa, naudodama šiuos skirtingus metodus. Taip pat buvo lyginamas vidutinis surastos trajektorijos ilgis, bei trajektorijos sudėtingumas, įvertinant pasisukimų skaičių. Nustatyta, kad trumpiausią trajektoriją esant skirtingam kliūčių skaičiui žemėlapyje suranda A* algoritmas, išskyrus atvejus kai žemėlapyje nėra kliūčių, arba tik 1 kliūtis. Tokiu atveju potencialų laukų metodas gali būti efektyvesnis. Paprastesnę trajektoriją randa, bei efektyviau tikslą pasiekia A*

algoritmas. Pastebėta, kad potencialų laukų algoritmas yra labai priklausomas nuo parametrų, kuriuos sunku nustatyti optimaliai.

Įvertinus tyrimo rezultatus, bei išvardytus algoritmų ypatumus, esant tokioms pačioms sąlygoms (žemėlapio dydis, kliūčių skaičius, naudojamo roboto gabaritiniai matmenys, statiška aplinka) roboto programavimui tikslo taškui pasiekti rekomenduojama naudoti A* algoritmą.

4. Projektinėje darbo dalyje buvo sėkmingai suprogramuotas mobilui autonominis Arduino robotas tikslo taškui pasiekti. Roboto navigacijai buvo naudojama tinklo kamera ir pasitelkiamas vaizdų atpažinimas, algoritmas ir roboto valdymas įgyvendinti naudojant *Microsoft Visual C# 2010 Express* programinį paketą, o duomenys iš programos į robotą perduodami bluetooth ryšiu. Buvo pasiektas vidutiniškai 5.8 kartų per sekundę komunikavimo dažnis.

   Buvo pastebėta, kad roboto trajektorijos įveikimo greitį įtakojo apšvietimas, vaizdų atpažinimo programos netobulumas, tinklo kameros raiška. Taip pat roboto aukštis kameros užfiksuotame vaizde buvo su perspektyva, todėl tikslo tašką robotas pasiekdavo su paklaida, priklausančia nuo tikslo taško buvimo vietos žemėlapyje. Neskaitant šios paklaidos, bei kameros iškreipiamo vaizdo įtakos, robotas tikslo tašką pasiekia su 5pikselių (~15 mm) paklaida.

5 LITERATŪROS SĄRAŠAS

1. DIXON, J; HENLICH, O. - Mobile Robot Navigation [interaktyvus]. Imperial College [London, UK]: Surprise, June 1997 [žiūrėta 2014 m. Kovo 14 d.]. Prieiga per internetą: http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/jmd/

2. Amit's A* Pages. Introduction to A* [interaktyvus]. [žiūrėta 2014 m. Kovo 14 d.] Prieiga per internetą: http://theory.stanford.edu/~amitp/GameProgramming/ /AStarComparison.html

3. LESTER, P. A* Pathfinding for beginners [interaktyvus]. July 2005 [žiūrėta 2014 m. Gegužės 27 d.]. Prieiga per internetą: http://www.policyalmanac.org/games/aStarTutorial.htm

4. DAS, A;MOHAPTRA, P; MISHRA, P; DAS, P; MADHATA, S. Improved A* algorithm for Path Planning of Mobile Robot in Quadrant Based Environment [interaktyvus]. Ravenshaw University [Odisha, India]: International Journal of Advanced Computer Theory And Engineering, Issue - 1 2012 [žiūrėta 2014 m. Kovo 22 d.]. Prieiga per internetą:http://www.irdindia.in/Journal_IJACTE/PDF/Vol1_Iss1/2.pdf

5. TOMPKINS, P; STENTZ, A; WETTERGREEN, D. Global Path Planning for Mars Rover Exploration [interaktyvus].The Robotics Institute, Carnegie Mellon University [Pittsburgh, USA]: [žiūrėta 2015 m. Balandžio 22 d.]. Prieiga per interintą: https://www.ri.cmu.edu/pub_files/pub4/tompkins_paul_2004_3/tompkins_paul_2004_3.pdf

6. KINAHAN, K; PRYOR, J. Example Networks. Dijkstra's Algorithm for Shortest Route Problem [interaktyvus]. [žiūrėta 2014 m. Kovo 25 d.] Prieiga per internetą: http://optlab-server.sce.carleton.ca/POAnimations2007/DijkstrasAlgo.html

7. STENTZ, A. Optimal and Efficient Path Planning for Partially Known Environments [interkatyvus]. The Robotics Institute, Carnegie Mellon University [Pittsburgh, USA]:[žiūrėta 2014 m. Birželio 13 d.]. Prieiga per internetą: http://www.frc.ri.cmu.edu/~axs/doc/icra94.pdf

8. CASTANEDA, M; SAVAGE, J; HERNANDEZ, A; COSIO, F. Local Autonomous Robot Navigation Using Potential Fields [interaktyvus]. University Country, Intech, June 2008 [žiūrėta 2014 m. Kovo 22 d.].Prieiga per internetą: http://www.intechopen.com/download/get/type/pdfs/id/5350

9. HAGELBACK, J. Using Potential Fields In RTS Game Scenario [interaktyvus]. January 2009 [žiūrėta 2014 m. Kovo 27 d.]. Prieiga per internetą: http://aigamedev.com/open/tutorials/potential-fields/

10. KOREN, Y; BORENSTEIN, J. Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation [interkatyvus]. Proceedings of the IEEE conference on Robotics and Automation [Sacramento, USA]: April 1991 [žiūrėta 2014 m. Balandžio 15 d.]. Prieiga per internetą: http://www-cgi.cs.cmu.edu/afs/cs.cmu.edu/Web/ /People/biorobotics/papers/sbp_papers/integrated1/borenstein_potential_field_limitations.pdf

11. GOODRICH, M. Potential Fields Tutorial [interkatyvus]. Prieiga per internetą:http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.3259&rep=rep1&type=pdf

12. AL-SAGBAN, M; DHAOUADI, R. Reactive Navigation Algorithm for Wheeled Mobile Robots Under Holonomic Constraints [interaktyvus]. American University of Sharaj [Sharaj, UAE]: April 2011 [žiūrėta 2014 m. Kovo 27 d.]. Prieiga per internetą: http://www.academia.edu/757737/Reactive_Navigation_Algorithm_for_Wheeled_Mobile_Robots_under_Non-Holonomic_Constraints

13. QUINGBAO, Z; HU, J; WENBIN, C; HENSCHEN, L. Robot Navigation Algorithm for Dynamic Unknown Environments Based on Dynamic Path Recalculation and Improved Scout Ant Algorithm [interaktyvus].*Applied Soft Computing.*December 2011, vol. 11, no. 8, p. 4667-4676 [žiūrėta 2014 m. Balandžio 5 d.]. Prieiga per internetą:http://www.sciencedirect.com/science/article/pii/S1568494611002778

14. LESTER, P. Heuristics and A* pathfinding. http://www.policyalmanac.org/games/heuristics.htm

15. Arduino Roboto Charakteristikos - Arduino Robot.http://arduino.cc/en/Guide/Robot

16. Tinklo kameros charakteristikos. http://www.acme.eu/en-us/node/292/pdf

17. HC-05 Product Data Sheet. Guanzhou HC Information Technology Ltd. http://www.seeedstudio.com/wiki/images/4/48/HC-05_datasheet.pdf