VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

Dirbtinio intelekto metodų panaudojimas autonominiam besimokančiam robotui

(Autonomous Self-Learning Robot)

Atliko: 4 kurso, 1 grupės studentas
Mindaugas Čiulada (parašas)
Darbo vadovas:
Prof., Dr. Aistis Raudys (parašas)
Recenzentas:
Prof., Dr. Linas Laibinis (parašas)

Vilnius
2020

TURINYS

Įvadas

Žmonijai besivystant, sparčiai kinta ir tobulėja naujosios technologijos, leidžiančios atlikti kasdienius veiksmus greičiau, paprasčiau ir patogiau. Galima sakyti gyvename technologijų amžiuje, kuriame gyvenimo tempas su kiekviena diena vis greitėja, norimų atlikti darbų kiekis vis didėja, todėl siekiame kiekvieną žmogaus žingsnį automatizuoti įvairiose sferose. Viena iš labiausiai vystomų technologijų – dirbtinis intelektas. Dirbtinį intelektą bandoma pritaikyti daugelyje sričių: nuo automatizuotos programinės įrangos iki robotų.

Robotai naudojami restoranuose, kovojant su nusikalstamumu, medicinoje, švietime, apsaugai, namuose ir net kavos gaminimo procese. Taip pat yra keletas robotų rūšių, pavyzdžiui, pramoniniai robotai, socialiniai robotai, žaislai ir t.t. Robotai gali atlikti daugybę užduočių, kurių nesugeba žmonės. Jie užtikrina produktyvų ir sklandų užduočių vykdymą ir išvengia dažnai pasikartojančių „žmogiškų klaidų“ atliekant pasikartojantį, monotonišką darbą.

Minėtos sferos atveria dar daugiau naujų galimybių. Tačiau daugelis robotų nenaudoja dirbtinio intelekto. Visai neseniai pramoniniai robotai galėjo būti programuojami tik atliekant pasikartojančius judesių ciklus, neturint jokio dirbtinio intelekto. Tačiau tokių robotų funkcionalumas yra gana ribotas todėl dirbtinio intelekto (toliau DI) algoritmai yra būtini, kad robotas gebėtų išmokti atlikti sudėtingesnes užduotis ir taip paskatintų intelektualaus automatizavimo vystymąsi.

DI privalo atlikti pagrindinį vaidmenį robotikoje, kad ryšys tarp aplinkos ir roboto atliekamų veiksmų būtų intelektualus. DI nagrinėja esminius klausimus: kaip teisingai interpretuoti išorinius duomenis, kaip iš jų mokytis, ir kaip tos žinios turėtų būti panaudotos, kad būtų įgyvendinti konkretūs tikslai ir uždaviniai. Robotikai sujungus variklius, jutiklius ir kompiuterius yra metamas iššūkis dirbtiniam intelektui, kurio pagalba robotas būtų priverstas atlikti veiksmus su realiais objektais, realiame pasaulyje.

Aktualumas – kadangi žmonių gyvenimo tempas greitėja ir žmonės ieško būdų, kaip palengvinti kasdienybę buityje, todėl pasirenka naudoti robotus atliekančius darbus už juos. Dėl šios priežasties robotikos sritis plečiasi itin sparčiai. Tačiau robotų programavimas gali būti ilgas ir daug laiko reikalaujantis procesas. Programuotojui reikia aprašyti daug sudėtingų veiksmų, kuriuos robotas

turėtų atlikti įvairiose situacijose, tačiau visų situacijų aprašyti neįmanoma. Todėl idėja, kad robotas išmoktų užduotį atlikti pats yra itin aktuali. Programuotojui yra daug lengviau ir intuityviau nurodyti, ką robotas turėtų atlikti ir leisti jam išmokti, kaip tai atlikti.

Darbo tikslas – Sukurti autonominį besimokantį robotą, kurio tikslas važinėjant padengti kuo didesnį aplinkos plotą.

Siekiami rezultatai:

1. Išnagrinėti literatūrą ir esančius metodus leidžiančius įgyvendinti autonomino roboto mokymasį.
2. Atlikti besimokančio roboto eksperimentines simuliacijas virtualioje aplinkoje.
3. Atlikti skatinamojo mokymosi algoritmų palyginimą autonominiam besimokančiam robotui.

1. Situacijos apžvalga

1.1. Dirbtinio intelekto sąvoka

DI yra plati kompiuterių mokslo šaka, susijusi su intelektualių mašinų kūrimu, galinčių atlikti užduotis, kurioms paprastai reikalingas žmogaus intelektas. Išsamesnis apibrėžimas apibūdina dirbtinį intelektą kaip sistemos sugebėjimą teisingai interpretuoti išorinius duomenis, mokytis iš tokių duomenų ir iš šių mokymų lanksčiai prisitaikant įgyvendinti konkrečius tikslus ir uždavinius [PMG98].

DI tyrimo tikslas yra sukurti technologiją, leidžiančią kompiuteriams ir mašinoms veikti intelektualiai. Bendroji intelekto imitavimo (arba kūrimo) problema buvo suskirstyta į kategorijas. Tai susideda iš tam tikrų bruožų ar galimybių, kuriuos tyrėjai tikisi, kad atvaizduos intelektuali sistema. Pagrindinės DI tyrimo problemos ir tikslai:

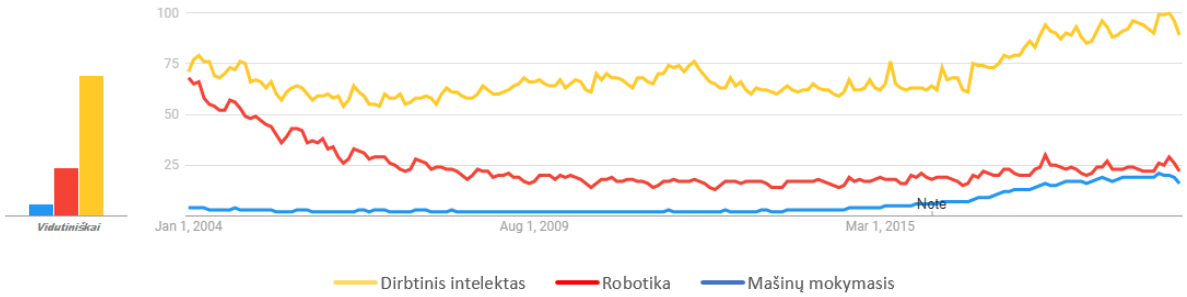
- samprotavimas, problemų sprendimas;
- žinių atvaizdavimas;
- planavimas;
- mokymas;
- natūralios kalbos apdorojimas;
- suvokimas;
- judesys ir manipuliavimas;
- socialinis intelektas;
- bendras intelektas.

Bendras intelektas yra vienas iš ilgalaikių šios srities tikslų. Požiūriai apima statistinius metodus, skaičiavimo intelektą ir tradicinį simbolinį DI. DI naudojama daug įrankių, įskaitant paiešką ir matematinį optimizavimą, dirbtinius neuroninius tinklus ir statistiką, tikimybes bei ekonomikos metodais pagrįstus metodus. DI sritis remiasi kompiuterių, informacijos inžinerijos, matematikos, psichologijos, kalbotyros, filosofijos ir daugeliu kitų sričių [Kur99].

DI pripažintas, kaip akademinė disciplina 1955 m. Per šiuos metus ši sritis sulaukė keletą optimizmo bangų lėmusių spartesnius tyrimus šioje srityje, taip pat ir nusivylimų po kurių buvo prarastas finansavimas. DI yra suskirstytas į daug pogrupių. Šiame darbe bus nagrinėjami du pogrupiai: mašininis mokymas ir robotika.

1.2. Mašininis mokymas

Didėjantis turimų duomenų kiekis ir įvairovė, pigesnė ir galingesnė kompiuterių skaičiavimo galia lėmė tikslesnę duomenų analizę. Organizacija analizuodama turimus duomenis turi daugiau galimybių priimti teisingus sprendimus ir išvengti tam tikros rizikos. Pastaraisiais metais mašinų mokymas yra vis didesnį populiarumą įgaunanti studijų sritis, tačiau vis tiek nusileidžia dirbtiniam intelektui ir robotikai (1 diagrama).



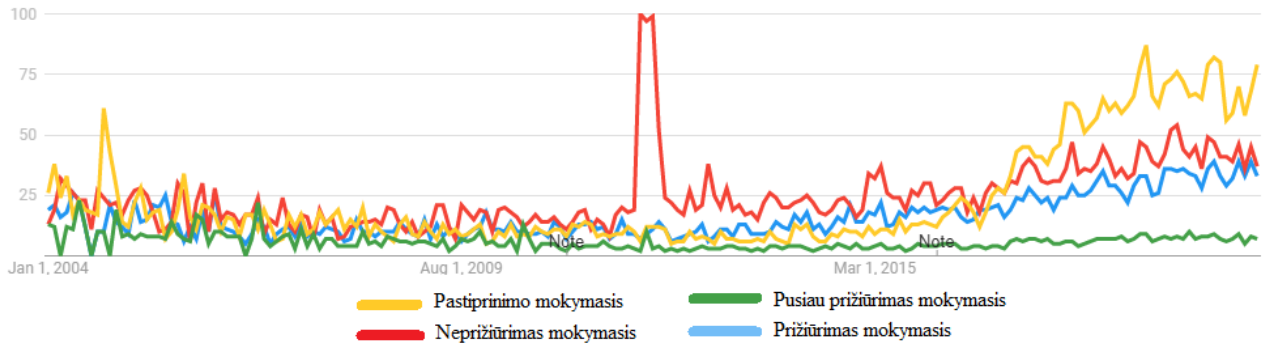
1 diagrama dirbtinis intelektas, robotika, mašinų mokymasis temų populiarumo palyginimas 2004-2020m. Šalt. https://trends.google.com/trends

Mašininis mokymas (angl. *machine learning*) - tai yra algoritmo kategorija, leidžianti programinei įrangai tiksliau numatyti rezultatus, net nebūnant aiškiai užprogramuotai. Pagrindinė mašininio mokymosi užduotis yra sudaryti algoritmus, galinčius priimti įvestus duomenis ir naudoti statistinę analizę numatyti išvesčiai, tuo pačiu metu atnaujinant išvestis, kai tik atsiranda naujų duomenų [Bis06].

Mašininis mokymas yra mokslinis algoritmų ir statistinių modelių, kuriuos kompiuterinės sistemos naudoja atlikdamos konkrečią užduotį, nenaudojant aiškių nurodymų, o remiantis modeliais ir išvadomis, tyrimas. Tai yra vienas iš dirbtinio intelekto pogrupių. Mašininio mokymosi algoritmai sukuria matematinį modelį, pagrįstą pavyzdžių duomenimis, vadinamais apmokymo duomenimis, kad būtų galima daryti prognozes ar priimti sprendimus, kurie nėra aiškiai užprogramuoti atlikti užduotį. Mašininio mokymosi algoritmai naudojami daugybės programų, tokių kaip el. pašto filtravimas ir kompiuterinis matymas, kur sunku arba neįmanoma sukurti įprasto algoritmo, kuris veiksmingai atliktų užduotį [Bis06].

Mašininio mokymo užduotys skirstomos į kelias plačias kategorijas. Algoritmų tipai skiriasi pagal požiūrį, įvestų ir išvedamų duomenų tipą, užduoties ar problemos, kurią jie skirti išspręsti, tipą. Šios kategorijos yra:

- prižiūrimas mokymas (angl. *Supervised learning*);
- neprižiūrimas mokymas (angl. *Unsupervised learning*);
- pusiau prižiūrimas mokymas (angl. *Semi-supervised learning*);
- skatinamasis mokymas (angl. *Reinforcement learning*).



2 diagrama Mašininio mokymosi tipai. Temų populiarumo palyginimas 2004-2020m. Šalt. https://trends.google.com/trends

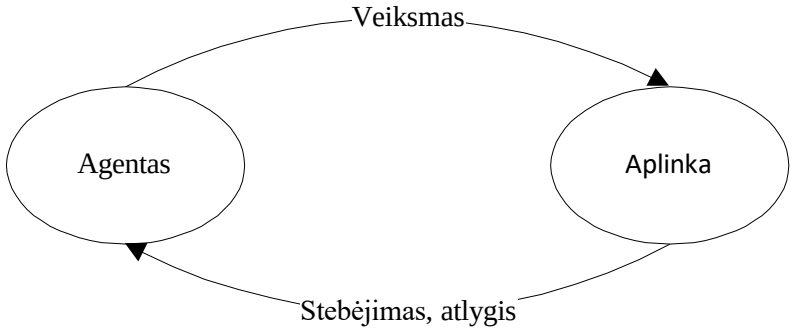
Remiantis diagrama (2 diagrama) galime teigti, jog skatinamasis mokymas per pastaruosius tris metus buvo pats populiariausias iš visų išvardintų kategorijų. Šis pakilimas yra susijęs su *OpenAI* dirbtinio intelekto tyrimų laboratorijos įkūrimu. Taip pat galime pastebėti didelį neprižiūrimo mokymosi susidomėjimą nuo 2011 m. iki 2012 m. kuris buvo dėl reikšmingų pasiekimų objektų atpažinime [Sch14].

Prižiūrimas mokymas yra mokymas iš daugybės paženklintų pavyzdžių, kuriuos pateikia prižiūrėtojas (angl. *supervisor*). Mokymosi metu funkcija sieja įvestį ir išvestį remiantis pateiktais pavyzdžiais. Kiekvienas pavyzdys yra pora, susidedanti iš įvesties objekto ir norimos išvesties vertės, kurią nurodo prižiūrėtojas. Algoritmas analizuoja mokymo duomenis ir sukuria numanomą funkciją, kurią galima panaudoti naujų pavyzdžių analizei. Tokio tipo mokymosi tikslas - turimų duomenų analizė, atsakymų apibendrinimas, tam kad sistema veiktų tinkamai situacijose su kuriomis ji dar nesusidūrė. Tai yra svarbus mokymo tipas, tačiau vien tik šio tipo naudojimas mokytis iš sąveikos nėra tinkamas. Interaktyviose problemose dažnai nepraktiška gauti norimo elgesio pavyzdžių, kurie būtų teisingi ir atspindintys visas situacijas, kuriose agentas turi veikti. Neaprašytose situacijose, kur galima tikėtis, kad mokymas bus pats naudingiausias, agentas turi sugebėti mokytis iš savo patirties.

Neprižiūrimas mokymas yra mokymas, kurio pagrindinis tikslas yra surasti paslėptą struktūrą nesužymėtuose duomenų rinkiniuose. Nors atrodo, kad skatinamasis mokymas yra lygiai toks pats kaip neprižiūrimas mokymas, nes jis nesiremia teisingo elgesio pavyzdžiais, tačiau skatinamojo mokymosi metu siekiama maksimaliai padidinti agento gaunamą atlygį, užuot ieškant paslėptos struktūros. Agento patirties struktūra tikrai gali būti naudinga mokymosi eigoje, tačiau tai neišsprendžia skatinamojo mokymosi problemos, susijusios su atlygio maksimizavimu.

Pusiau prižiūrimas mokymas yra mašininio mokymosi metodas, kai mokymo metu naudojamas nedidelis kiekis sužymėtų duomenų su dideliu kiekiu nepaženklintų duomenų. Pasibaigus mokymosi procesui gaunamas modelis pagal kurį yra analizuojami panašūs nepažymėti duomenys. Pusiau prižiūrimas mokymas priklauso nuo neprižiūrimo mokymo (be pažymėtų mokymo duomenų) ir prižiūrimojo mokymosi (turinčio tik paženklintus mokymo duomenis). Šiuo metodu siekiama pagerinti mokymosi tikslumą ir sumažinti išlaidas, kadangi sužymėti dideli duomenų rinkiniai yra brangūs palyginus su nepaženklintais duomenimis.

Skatinamasis mokymas yra mašininio mokymosi modelių mokymas priimti sprendimų seką. Agentas išmoksta pasiekti tikslą neapibrėžtoje, potencialiai sudėtingoje aplinkoje. Kad mašina atliktų tai, ko nori programuotojas, DI gauna atlygį arba baudą už veiksmus, kuriuos jis atlieka. Tikslas - maksimaliai padidinti bendrą atlygį. Dažniausiai veiksmai gali paveikti ne tik tiesioginį atlygį, sekančią situaciją, bet ir visų būsimų situacijų atlygį. Veikimo principas pavaizduotas (Pav. 1). Pagrindinės šio mokymosi savybės – paieška bandymų-klaidų būdu, bei ateities atlygis.



1 pav. Skatinamojo mokymosi veikimo principas. Agentas atlieka veiksmą aplinkoje, už kurį gaunamas atlygis ir dabartinė būsena.

Skatinamasis mokymas yra pritaikomas daugelyje pramonės sričių, įskaitant internetinę reklamą, elektroninę prekybą, finansus, gamybą ir robotiką. Šis mokymas suteikia robotui galimybę

savarankiškai atrasti optimalų elgesį, atliekant veiksmus ir sąveikaujant su aplinka (angl. *enviroment*), todėl šiame darbe buvo pasirinktas būtent šis mokymosi tipas.

Nors programuotojas nustato atlygio strategiją, tačiau jis modeliui neteikia jokių užuominų ar pasiūlymų, kaip išspręsti problemą. Agentas turi išsiaiškinti, kaip atlikti užduotį, siekiant maksimaliai padidinti atlygį, pradedant nuo visiškai atsitiktinių bandymų ir baigiant sudėtinga taktika. [KBP13] Pasinaudodamas bandymų-klaidų būdu, skatinimasis mokymas yra pats efektyviausias būdas leisti suprasti apie mašinos kūrybingumą. Priešingai nei žmonės, DI gali kaupti patirtį iš tūkstančių lygiagrečių situacijų, jei skatinamojo mokymosi algoritmas vykdomas pakankamai galingame kompiuteryje [KLM96].

Be agento ir aplinkos, galima išskirti keturis pagrindinius skatinamojo mokymosi sistemos elementus: strategiją (angl. *policy*), atlygį (angl. *reward*), vertės funkciją (angl. *value function*) ir aplinkos modelį (angl. *model*). Strategija tai sąryšių matrica tarp aplinkos būsenų ir atliekamų veiksmų, ji nurodo agento elgesį tam tikru metu. Kai kuriais atvejais strategija gali būti funkcija arba paprasta paieškos lentelė, o kitais atvejais tai gali būti daug resursų reikalaujantis paieškos procesas. Strategija yra skatinamojo mokymosi pagrindinė dalis, kurios visiškai užtenka apibūdinti agento elgesiui.

Atlygis nusako skatinamojo mokymosi problemos tikslą. Kiekviename etape aplinka siunčia skatinamojo mokymosi agentui reikšmę, vadinamą atlygiu. Vienintelis agento tikslas yra maksimaliai padidinti visą jo gaunamą atlygį per ilgą laiką. Atlygis taip pat nusako, kokie yra geri ir blogi įvykiai. Atlygis yra pagrindinis politikos pakeitimo pagrindas; jei po politikos pasirinkto veiksmo gaunamas mažas atlygis, tada strategija gali būti keičiama, kad ateityje agentas pasirinktų kitą veiksmą toje situacijoje. Skiriamas atlygis gali būti stochastinės aplinkos būsenos ir atliktų veiksmų funkcijos. Atlygis parodo, kad dabar atliktas veiksmas yra vertinamas teigiamai, o vertės funkcijos tikslas yra kuo didesnis agento atlygis ateityje (angl. *future reward*), kitaip tariant, būsenos vertė yra bendra atlygio suma, kurios agentas gali tikėtis ateityje. Pozityvus atlygis lemia pirmaeilį aplinkos būsenos siekį, o vertės funkcija rodo ilgalaikį būsenų pageidavimą, atsižvelgiant į tikėtinas būsenas ir jose esantį atlygį. Pavyzdžiui būsena visada gauna nedidelį atlygį, tačiau vis tiek turi didelę vertę, nes po jos seka kitos būsenos, įvertintos dideliu atlygiu. Be atlygio negalime nustatyti vertės, o vienintelis vertės nustatymo tikslas yra gauti kuo didesnį atlygį. Nepaisant to, priimant ir vertinant veiksmus svarbiausias rodiklis yra vertė. Veiksmai pasirenkami remiantis vertės vertinimais. Todėl yra siekiami veiksmai, kurie leistų gauti maksimalią vertę, o ne aukščiausią atlygį, nes šie veiksmai

ilgainiui suteiks didžiausią atlygį. Tačiau vertę nustatyti yra daug sunkiau, nei nustatyti atlygį. Atlygį teikia aplinka, tačiau vertes reikia įvertinti atsižvelgiant į visas simuliacijas, kurias agentas darė nuo pat pradžių. Vertės nustatymo vaidmuo yra pats svarbiausias dalykas, kuris buvo nagrinėjamas per pastaruosius šešis dešimtmečius.

Aplinkos modelį galima prilyginti kaip aplinkos simuliaciją, kuri leidžia daryti išvadas apie tai, kaip aplinka elgsis. Pavyzdžiui atsižvelgiant į būseną ir veiksmą, modelis gali numatyti kitą būsimą būseną ir sekantį atlygį. Modeliai yra naudojami planavimui, kuriais yra remiamasi prieš priimant tam tikrą veiksmą, apsvarstant galimas situacijas ateityje.

Žemiau pateikiami skatinamojo mokymosi algoritmo įgyvendinimo tipai:

1. Remiantis verte (angl. *Value-based*) – Taikydami vertės mokymosi būdą, yra stengiamasi maksimaliai padidinti vertės funkciją. Pagal šį metodą agentas tikisi ilgalaikės grąžos iš dabartinių būsenų.
2. Remiantis strategija (angl. *Policy-based*) – Šiuo būdu yra stengiamasi suformuoti tokią strategiją, kad kiekvienoje būsenoje atliktas veiksmas padėtų ateityje gauti maksimalų atlygį. Yra išskiriami du strategija pagrįsti metodai. Deterministiniu metodu bet kuriai būsenai tą patį veiksmą sukuria strategija. Stochastiniu metodu kiekvienas veiksmas turi tam tikrą tikimybę, kurią nustato funkcija.
3. Remiantis modeliu (angl. *Model-based*) - Taikydami šį skatinamojo mokymosi būdą yra sukuriamas virtualus aplinkos modelis. Agentas mokosi ir planuoja atlikti veiksmus sukurtoje aplinkoje.

Skatinamojo mokymosi ypatybės:

- nuoseklus sprendimų priėmimas;
- laikas yra svarbus rodiklis;
- agentų veiksmai lemia sekančius duomenis;
- nėra prižiūrėtojo (angl. *supervisor*);
- agentas stebi tik gaunamus duomenis ir jam skiriamą atlygį už atliktus veiksmus.

Skatinamojo mokymosi tipai:

- Teigiamas - apibrėžiamas kaip įvykis, atsirandantis dėl tam tikro agento elgesio. Agentui skirtas pozityvus atlygis už veiksmą padidina šio veiksmo dažnį bei daro teigiamą poveikį

agento veiksmams. Šis skatinamojo mokymosi tipas padeda maksimaliai padidinti agento našumą ir išlaikyti pokyčius ilgesnį laiką.

- Neigiamas – agento veiksmus lemia neigiamai vertinamas elgesys, kuris atsiranda dėl nurodytų sąlygų, kurios turėjo būti išvengtos. Tai apibrėžia minimalų agento darbo atlikimo lygį tam tikroje aplinkoje.

1.3. Robotika

Robotika yra tarpdisciplininė inžinerijos ir mokslo šaka, apimanti mechanikos inžineriją, elektronikos inžineriją, informatiką ir kt. Robotika susijusi su robotų projektavimu, konstravimu, valdymu ir naudojimu, taip pat kompiuterinėmis sistemomis, skirtomis jų valdymui, jutiminiam grįžtamajam ryšiui ir informacijos apdorojimui.

Yra daugybė robotų tipų, kurie naudojami daugelyje skirtingų aplinkų ir įvairiems tikslams, nors yra labai skirtingo taikymo ir formos, tačiau juos visus sudaro trys pagrindiniai panašumai, susiję su jų konstrukcija:

- Visi robotai turi tam tikrą mechaninę konstrukciją, rėmą, formą, skirtą tam tikrai užduočiai pasiekti. Mechaninis aspektas dažniausiai yra kūrėjo sprendimas baigti paskirtą užduotį ir spręsti aplink ją esančios aplinkos fiziką. Forma atitinka funkciją [Ark98].
- Robotai turi elektrinius komponentus, kuriais varomas ir valdomas mechanizmas. Pavyzdžiui elektra naudojama roboto judėjimui, jutimui (kai elektriniai signalai naudojami tokiems dalykams kaip šiluma, garsas, padėtis ir energijos būsenai matuoti) ir veikimui (robotams reikia tam tikro lygio elektros energijos, tiekiamos jų varikliams ir jutikliams, norint atlikti pagrindines operacijas)
- Visuose robotuose yra tam tikro lygio kompiuterio programavimo kodas. Programa yra tai, kaip robotas nusprendžia, kada ir kaip ką nors padaryti. Ji yra pagrindinė roboto esmė, jis gali turėti puikią mechaninę ir elektrinę konstrukciją, tačiau jei jo programa yra blogai parašyta, jo našumas bus labai prastas arba jis nesubės įvykdyti jam paskirtos užduoties. Robotas su nuotolinio valdymo programavimu turi iš anksto egzistuojančių komandų rinkinį, kurį jis vykdys tik tada, kai gaus signalą iš valdymo šaltinio, paprastai žmogaus su nuotolinio valdymo pultu. Galbūt labiau tikslinga į prietaisus, valdomus visų pirma žmonių komandomis, žiūrėti kaip į automatikos, o ne į robotikos discipliną. Robotai, kurie naudoja dirbtinį intelektą, patys sąveikauja su aplinka be valdymo šaltinio ir gali nustatyti reakcijas į objektus bei problemas, su kuriomis susiduria.

2. Naudojamos technologijos

2.1. *OpenAI gym*

OpenAI Inc kompanija vykdo mokslinius tyrimus dirbtinio intelekto srityje, siekdama jį skatinti ir plėtoti. San Franciske įsikūrusi organizacija, įkurta 2015 m., siekia bendradarbiauti su kitomis institucijomis ir tyrėjais, palikdama savo mokslinius tyrimus atvirus visuomenei. 2016 m. *OpenAI* išleido viešą *OpenAI Gym* platformą skirtą skatinamojo mokymosi tyrimams. Pagrindiniai platformos uždaviniai:

- Suteikti daug skirtingų ir įvairių aplinkų, nes egzistuojančiose atvirojo kodo aplinkų rinkiniuose nepakanka įvairovės, sunku jas įdiegti ir naudoti.
- Standartizuoti publikacijose naudojamas aplinkas. Net ir nedideli pakeitimai atlygio funkcijoje ar agento atliekamuose veiksmuose, gali drastiškai pakeisti užduoties sudėtingumą. Dėl šių problemų sunku atkartoti paskelbtus tyrimus ir palyginti skirtingų publikacijų rezultatus.

Platformos pagrindinis tikslas yra kuo greičiau maksimizuoti epizodinį atlygį ir lengvai iš naujo panaudoti esamus sprendimus. *OpenAI Gym* suskaldo agento patirtį į keletą epizodų. Kiekvienas epizodas susideda iš pradinės agento būsenos, agento sąveikos su aplinka ir pasiektos galutinės būsenos. Epizodinio skatinamojo mokymosi tikslas yra gauti maksimalų simuliacijos taškų skaičių ir pasiekti norimą rezultatą su kuo mažiau epizodų. Platforma naudojama įvairiems projektams - DARPA *Robotics Challenge*, NASA *Space Robotics Challenge, Toyota Prius Challange* ir t.t [BCP+16]. *OpenAI gym* biblioteka yra bandomųjų problemų ir aplinkų rinkinys, kurį galima naudoti norint kurti skatinamojo mokymosi algoritmus. Šios aplinkos turi bendrą sąsają, leidžiančią rašyti bendruosius algoritmus.

2.2. *Gazebo*

Gazebo yra atvirojo kodo 3D robotikos simuliatorius skirtas išbandyti algoritmus, projektuoti robotus, išmokyti DI sistemą naudojant realius scenarijus. *Gazebo* gali simuliuoti sąlyginai sudėtingą aplinką pavyzdžiui: kambarį su interjero detalėmis ir lauke esančius objektus - tai suteikia tikrovišką aplinkos (įskaitant apšvietimą, šešėlius) perteikimą. Remiantis lazeriniais atstumo sensoriais, kameromis robotas gali modeliuoti simuliuojamą aplinką. *Gazebo* yra integruotas *ODE* fizikos variklis, *OpenGL* apdorojimas, taip pat palaikoma programinė įranga skirta įvairiems įrenginiams. 2012 m. *Gazebo* projekto valdytoja tapo *Open Robotics*. Naudodami *Gazebo* simuliatorių galime

įvertinti ir išbandyti robotą esant sudėtingiems ar pavojingiems scenarijams, nepadarant žalos robotui. Taip pat tai leidžia sutaupyti laiko, užuot paleidus visą scenarijų realioje aplinkoje, tai galima atlikti simuliacijoje.

2.3. ROS

ROS (*robot operating system*) tai yra atviro kodo tarpinė programinė įranga teikianti bibliotekas ir įrankius, kurie padeda programinės įrangos kūrėjams kurti robotikos programas. ROS suteikia techninės įrangos simuliaciją, įrenginių tvarkykles, bibliotekas, vizualizatorius, pranešimų siuntimą, paketų valdymą ir dar daugiau. ROS licencijuojama pagal atvirojo kodo, BSD licenciją [Tul18]. ROS buvo sukurta siekiant skatinti bendrą robotikos programinės įrangos kūrimą.

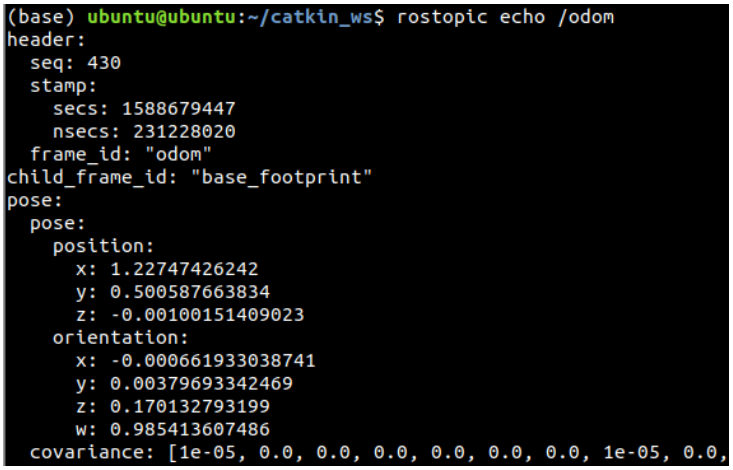
Open Robotics ne pelno siekianti organizacija yra viena iš pagrindinių šalių prisidedančių prie ROS kūrimo. Kiekvienais metais išleidžiama nauja versija ir susidomėjimas ROS toliau auga. *ROSCons* konferencijos vyksta kasmet nuo 2012 m., Kartu su ICRA (*International Conference on Robotics and Automation*) ir IROS (*IEEE Robotics and Automation Society*) - įvairiose šalyse buvo organizuojami *ROS* kūrėjų susitikimai, inicijuojamos švietimo programos bei leidžiamos knygos. 2014 m. NASA pranešė apie pirmąjį robotą kosmose, naudojantį ROS *Robotnaut 2*.

ROS susideda iš kelių dalių: mazgų (angl. *nodes*), temų (angl. *topics*), paslaugų (angl. *services*) ir parametrų serverio (angl. *parameter server*). ROS procesai yra pavaizduoti kaip mazgai grafo struktūroje, sujungti briaunomis, vadinamomis temomis. ROS mazgai gali perduoti pranešimus vienas kitam per temas, pasinaudoti kitų mazgų teikiamomis paslaugomis, taip pat gali teikti paslaugas kitiems mazgams arba nuskaityti duomenis iš duomenų bazės vadinamos parametrų serveriu. ROS naudojama decentralizuota architektūra leidžia tinklo įrenginiams naudojant temas efektyviai apsikeisti informacija, todėl sudėtingus skaičiavimus ar komandas galima atlikti nutolusiame kompiuteryje. [Tul18]

Mazgas atitinka vieną procesą ROS grafe. Kiekvienas mazgas turi pavadinimą, kuris yra užregistruojamas ROS pagrindiniam mazge, kad galėtų atlikti veiksmus. Mazgai yra pagrindinė ROS dalis, kadangi visi priimti veiksmai remiasi iš kitų mazgų gauta ir išsiųsta informacija. [Tul18]

Temos tai magistralės per kuriuos mazgai siunčia ir gauna žinutes. Temų pavadinimai taip pat turi būti unikalūs jų vardų srityje. Norėdami išsiųsti pranešimus į temą, mazgas turi paskelbti (angl. *publish*) ta tema, o gauti pranešimus – užsiprenumeruoti (angl. *subscribe*). Užsiprenumeruoti/paskelbti modelis yra anoniminis, tik pats mazgas žino, kad siunčia arba priima ta

tema, tačiau kurie kiti mazgai siunčia ar gauna temą nežino. Pranešimų, perduodamų šia tema, tipai skiriasi ir juos gali apibrėžti vartotojas. Šių pranešimų turinį gali sudaryti bet kas pavyzdžiui: jutiklio duomenys, variklio valdymo komandos, būsenos informacija ir t.t. Paveiksliuke (2 pav.) pateikiamas temos */odom* prenumeratos pavyzdys. Matome *base_footprint* buvimo koordinates tam tikru laiko momentu.



2 Pav. Išvedami odometro duomenys į ekraną naudojant /odom temą.

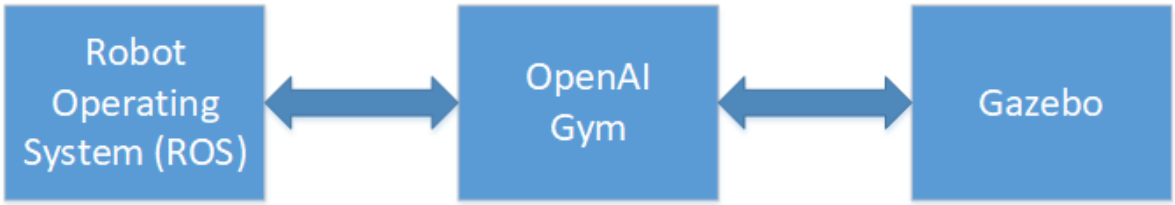
Paslauga reiškia veiksmą, kurį mazgas gali atlikti. Paprastai paslaugos yra naudojamos veiksmams, kurie turi apibrėžtą pradžią ir pabaigą, pavyzdžiui, fiksuoti vieno kadro vaizdą, o ne apdoroti variklio ar odometro duomenis. Mazgai gali naudoti paslaugas, o paslaugos kviesti kitas paslaugas. [Tul18]

Parametrų serveris yra bendra duomenų bazė skirta dalintis statine arba pusiau statine informacija tarp mazgų. Tai būtų duomenys, kurie dažnai nesikeičia ir yra retai nuskaitomi, pavyzdžiui, atstumas tarp dviejų fiksuotų aplinkos taškų arba roboto svoris. [Tul18]

3. Projektinė dalis

3.1. Projekto aplinka

Projektas susideda iš trijų programinių blokų: ROS, *OpenAI Gym* ir *Gazebo* (3 pav.). Aplinka sukurta *OpenAI Gym* keičiasi duomenimis su ROS, kurioje yra aprašyti visi robotui reikalingi įtaisai (davikliai, ratai, kamera ir t.t.). Grafinę sąsają ir fizikos variklį suteikia įrankis *Gazebo*. Jis atvaizduoja roboto sąveiką su aplinka [ZLV+16].

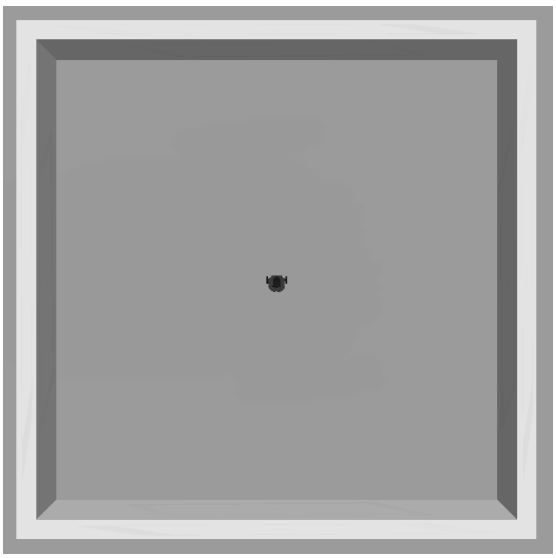


3 pav. Projekto modelis. Visos dalys tarpusavyje keičiasi duomenimis.

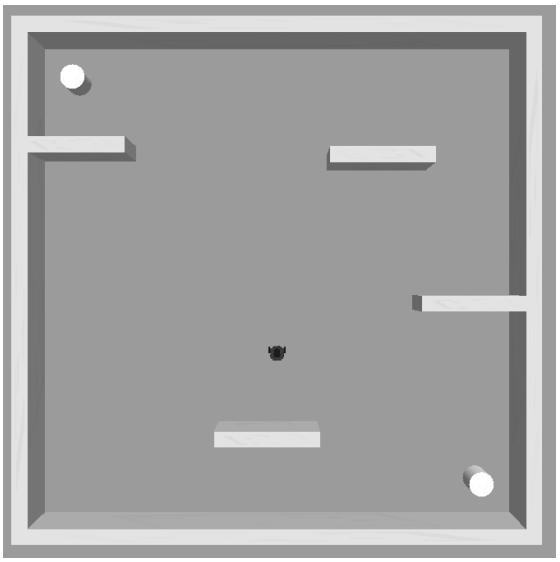
Projektas vykdomas ROS *catkin* darbinėje aplinkoje. Simuliacijoje buvo pasirinktas *Turtlebot3* (4 pav) roboto modelis, kuris yra sukuriamas pasinaudojant openai_ros paketu. *TurtleBot3* yra nedidelis, palyginus pigus, ROS platformai sukurtas robotas, skirtas naudoti švietimui, tyrimams ar laisvalaikiui, jo pagrindinis tikslas yra sumažinti platformos dydį ir sumažinti kainą, nedarant įtakos jo funkcionalumui ir kokybei. Robotas gali būti pritaikytas įvairioms situacijoms. Jis sukurtas naudojant nedidelę ir visus reikiamus komponentus talpinančią kompiuterinę plokštę integruotoms sistemoms (angl. *embedded system*). Kuriant *TurtleBot3* ypatingas dėmesys buvo skirtas SLAM *(simultaneous localization and mapping)* nuolatinės lokalizacijos ir navigacijos funkcijoms atlikti, tam kad robotas būtų efektyvus buities darbų sferoje. *TurtleBot3* naudodamasis lokalizacijos ir žemėlapių sudarymo algoritmais gali sudaryti kambario žemėlapį, važiuoti aplink namus, atvaizduoti aplinką 3D modeliu ir kitas suprogramuotas užduotis. Robotą galima valdyti nuotoliniu būdu iš nešiojamojo kompiuterio, *joypad* ar *Android* sistemos naudojančio išmaniojo telefono. *TurtleBot3* taip pat gali sekti nurodytus objektus, pavyzdžiui vaikštančio žmogaus kojas ir važiuoti iš paskos.

4 pav. Robotas Turtlebot3

Projekte bus naudojamos dvi aplinkos. Pirmoje aplinkoje yra suformuotas kvadratas, kuriame vienintelė kliūtis yra kvadrato briaunos (pav. 5). Antroje aplinkoje yra taip pat suformuotas kvadratas, kuriame yra pridėtos papildomos kliūtys neskaitant kvadrato briaunų – keturios nejudančios nedidelės sienos ir du atsitiktine tvarka judantys cilindrai (pav. 6). Robotas naudodamas *LIDAR* daviklius turės atlikti jam paskirtą užduotį - padengti kuo didesnį plotą kvadratų esančių šiame plote neatsitrenkiant į kliūtis.

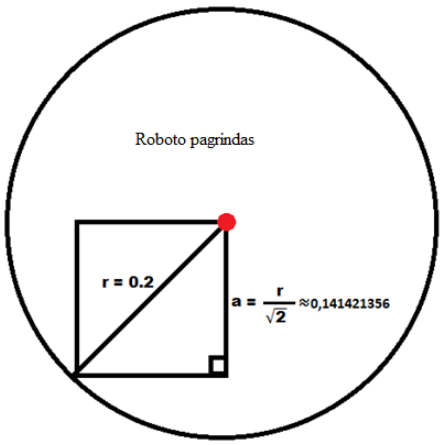


5 pav. Aplinka Nr. 1



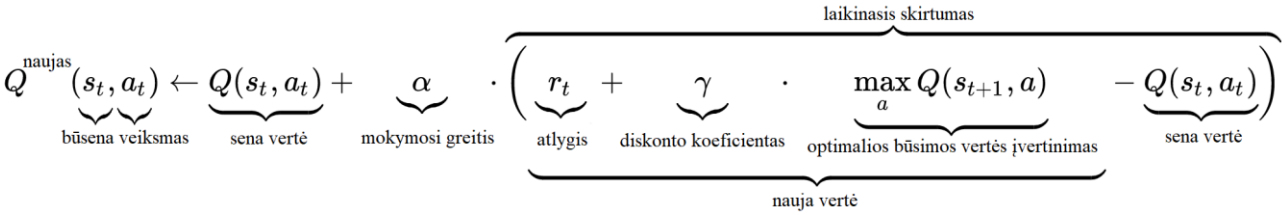
6 pav. Aplinka Nr. 2

Naudojamo roboto pagrindo spindulys yra 0.2 todėl pasirinkto kvadrato kraštinės dydis yra $a = \frac{r}{\sqrt{2}}$ tam, kad kvadrato plotas būtų visiškai padengtas (*pav. 7*).



7 pav. Roboto pagrindas visiškai padengia kvadratą, jeigu roboto pagrindo centras paliečia kvadrato viršūnę.

Problemos sprendimui buvo pasirinkti du skatinamojo mokymosi algoritmai - *Q–learning* ir *SARSA (State–action–reward–state–action*). *Q-learning* - skatinamojo mokymosi algoritmas, kurio pagrindinis tikslas yra išmokti strategiją, kai agentui nurodoma, kokių veiksmų imtis tam tikromis aplinkybėmis. Algoritmui nereikia pateikti aplinkos modelio, jam pakanka skiriamo atlygio už tam tikrus veiksmus, kad galėtų spręsti nurodytą problemą. Bet kokiam baigtiniam *Markovo* sprendimo procesui, *Q-learning* programa yra optimali ta prasme, kad ji maksimaliai padidina tikėtiną viso atlygio vertę per bet kuriuos iš eilės einančius žingsnius, pradedant nuo dabartinės būsenos. *Q-learning* gali nustatyti optimalią bet kurio tokio tipo procesui veiksmų atrankos strategiją, turint begalinį tyrimo laiką ir iš dalies atsitiktinę strategiją [Mat15].



8 pav. Q-learning algoritmo formulė

Q-learning mokymosi metu yra sukuriama matrica, kuri atitinka [būsena, veiksmas] formą, iš pradžių jų reikšmė yra 0. Po epizodo šios Q reikšmės yra atnaujinamos ir išsaugomos. Ši lentelė tampa orientacine ir agentas pasirenka geriausią veiksmą pagal Q vertę. Agentas sąveikauja su aplinka dviem būdais:

Pirmas - naudojama Q orientacinė lentelė, peržiūrint visus galimus veiksmus tam tikroje būsenoje ir tuomet agentas pasirenka veiksmą pagal maksimalią tų veiksmų vertę. Tai vadinama eksplotavimu (angl. *exploiting*), nes sprendimo priėmimui naudojame turimą informaciją.

Antras - veiksmai atliekami atsitiktinai. Tai vadinama tyrinėjimu (angl. *exploring*). Užuot pasirinkę veiksmus pagal maksimalų būsimą atlygį, pasirenkame veiksmą atsitiktinai. Veikti atsitiktine tvarka yra svarbu, nes tai leidžia agentui tyrinėti ir atrasti naujas būsenas, kurių priešingu atveju agentas gali neaptikti naudojant veiksmus pagal Q lentelę. Nepaisant šių būdų pasirinkimo Q lentelės reikšmės yra nuolat atnaujinamos. Taip pat galima subalansuoti aplinkos tyrinėjimą ir orientacinės Q lentelės naudojimą pasitelkiant *epsilon* (ε) kintamąjį. Nustatydami ε nurodome agentui ant kiek dažnai norime priimti atsitiktinį veiksmą ar naudotis orientacine Q lentele priimant sprendimą.

Q lentelės atnaujinimai atliekami po kiekvieno veiksmo ir baigiasi, kai baigiamas epizodas. Atlikta (*done*) reiškia, kad agentas pasiekia tam tikrą galutinį tašką, šiuo atveju tai būtų, kai robotas

atsitrenkia į kliūtį arba pasiekia nurodyta aplankytų kvadratų skaičių. Agentas daug ko neišmoks po vieno epizodo, bet galiausiai pakankamai ištyręs žingsnius ir epizodus išmoks optimalias Q reikšmes.

Mokymosi greitis dažnai vadinamas alfa arba α, jis apibrėžia ant kiek dažnai sena vertė pakeičiama naująja. Jeigu α lygi tuomet 0 agentas nieko nesimoko (naudojasi ankstesnėmis žiniomis), jei α lygi 1 tuomet agentas atsižvelgia tik į naujausią informaciją (ignoruodamas anksčiau surinktą informaciją). Formulėje vaizduojamas naujos ir senos reikšmių skirtumas bei daugyba su mokymosi greičio reikšme. Tuomet gauta reikšmė pridedama prie ankstesnės Q vertės, kuri tampa naujausiu reikšmių atnaujinimu. Mokymosi greičio dydis buvo pasirinktas $\alpha = 0.2$.

Būsena tai aplinkos stebėjimas todėl jai priskiriamos LIDAR lazerių reikšmės ir aplankytų kvadratų skaičius. Veiksmas tai yra ką agentas gali daryti kiekvienoje būsenoje: važiuoti pirmyn, pasukti į dešinę arba pasukti į kairę.

Diskonto koeficientas lemia ateities atlygio svarbą. Jeigu γ koeficientas lygus 0, agentas atsižvelgs tik į paskutinius atlygius, o jei koeficientas yra arti 1, privers agentą siekti didelio atlygio ateityje. Jei diskonto koeficientas lygus arba viršija 1 neturint galinės būsenos, arba jei agentas jos niekada nepasiekia, visa agento patirtis ir atlygis tampa neapibrėžtai begalinė. Net jei diskonto koeficientas yra tik šiek tiek mažesnis nei 1, tai lemia klaidų ir nestabilumo atsiradimą. Todėl rekomenduojama, pradėto nuo mažesnio diskonto koeficiento ir palaipsniui didinti iki galutinės vertės. Šiame darbe γ dydžio pasirinkta pradinė reikšmė lygi 0.85 ir bus didinama iki maksimalios reikšmės 0.9 [Pit19].

Atlygis yra vertė, skiriama agentui atlikus veiksmą tam tikroje būsenoje. Atlygis gali būti suteiktas bet kuriuo laiko momentu.

Maksimaliai reikšmei naudojama numpy biblioteka, kuri remiasi maksimalia būsimojo atlygio dalimi ir pritaiko tai esamai būsenai. Galimas atlygis ateityje daro įtaką dabartiniams veiksmams. Todėl skiriame būsimą atlygį dabartiniams veiksmams, kad padėtume agentui pasirinkti geriausią veiksmą kiekvienoje būsenoje [RN18].

Naudojami trys pagrindiniai žingsniai:

- agentas pradeda būsenoje (s1) imasi veiksmo (a1) ir gauna atlygį (r1);
- agentas pasirenka veiksmą, naudodamas Q lentelę su didžiausia verte arba atsitiktine tvarka;
- atnaujinamos Q reikšmės.

SARSA (9 pav.) yra Markovo sprendimų priėmimo proceso mokymosi algoritmas, taip pat naudojamas mašininio mokymosi stiprinimo srityje. Pagrindinis skirtumas tarp jo ir *Q-Learning*, yra

tai, kad maksimalus atlygis už kitą būseną nebūtinai naudojamas atnaujinant Q reikšmes. Vietoj to, naujas veiksmas pasirenkamas naudojant tą pačią strategiją [RN94].

$$\overset{\text{naujas}}{Q(\underbrace{s_t}_{\text{būsena}}, \underbrace{a_t}_{\text{veiksmas}})} \leftarrow \underbrace{Q(s_t, a_t)}_{\text{sena vertė}} + \underbrace{\alpha}_{\text{mokymosi greitis}} \cdot [\underbrace{r_t}_{\text{atlygis}} + \underbrace{\gamma}_{\text{diskonto koeficientas}} \cdot Q(s_{t+1}, a_{t+1}) - \underbrace{Q(s_t, a_t)}_{\text{sena vertė}}]$$

9 pav. SARSA algoritmo formulė

Simuliacijos metu iš pradžių robotas pastatomas pradiniame taške. Už kiekvieną atliktą veiksmą jam skiriami atitinkami taškai. Jei robotas:

- aplankė dar neaplankytą kvadratą skiriamas 1 taškas [JG12];
- aplankė prieš tai aplankytą kvadratą atimamas 1 taškas [JG12];
- robotas dar nespėjo išvažiuoti iš kvadratėlio už kurį buvo skirtas atitinkamas atlygis, tuomet jam skiriama 0 taškų [JG12];
- aplankė 100 kvadratų skiriami 100 taškų;
- aplankė 200 kvadratų skiriami 200 taškų;
- aplankė 300 kvadratų skiriami 300 taškų;
- aplankė 400 kvadratų skiriami 400 taškų;
- aplankė 500 kvadratų skiriami 500 taškų;
- aplankė 600 kvadratų skiriami 600 taškų;
- aplankė 700 kvadratų skiriami 700 taškų;
- aplankė 80% aplinkoje esančių kvadratų skiriama 1000 taškų, baigiamas epizodas, robotas grįžta į pradinę poziciją ir taškai pradedami skaičiuoti iš naujo;
- atsitrenkia į kliūtį atimami 200 taškų, baigiamas epizodas, robotas grįžta į pradinę poziciją ir taškai pradedami skaičiuoti iš naujo.

Projekto aplinka apibrėžia veiksmus, kuriuos agentas gali atlikti ir nurodo kaip skirti atlygį. Naudojant *OpenAI* standartizuotą aplinkos kūrimo būdą, sukurtoje aplinkos klasėje privalu įgyvendinti šias funkcijas:

- *_seed* funkcija yra skirta atsitiktinių skaičių generavimui, kurie naudojami mokymosi algoritmui atliekant atsitiktinius veiksmus.
- *_reset* funkcija atstato visą aplinką iki pradinės būsenos, tam kad simuliacijos galėtų prasidėti tomis pačiomis sąlygomis (atstatomos temų reikšmės).

- _step funkcija yra naudojama kiekviename žingsnyje. Algoritmo pasirinktas veiksmas yra nurodomas, kaip parametras. Funkcija grąžina keturias reikšmes: stebėjimą, atlygį, simuliacijos pabaigos būseną ir papildomą informaciją. Funkcijoje *_step* kviečiamas metodas *getNewAreaValue*, kuris nustato, kokį atlygį skirti agentui.

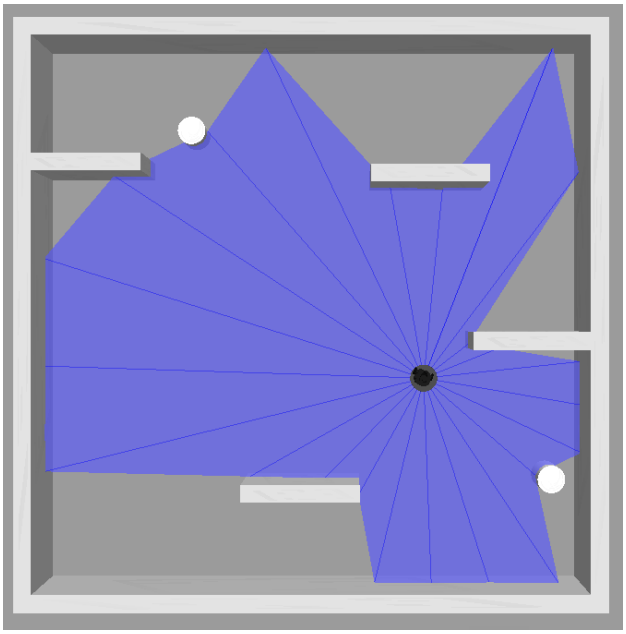
```
    def getNewAreaValue(self):
        if (len(self.currentAreas) > 0):
            self.lastArea = self.currentAreas[len(self.currentAreas)-1]

        newAreaCoordinatesString = str(
            math.floor(self.newSquareCoordinates["x"] / SQUARE_SIDE_SIZE) * SQUARE_
SIDE_SIZE) + "," + str(
            math.floor(self.newSquareCoordinates["y"] / SQUARE_SIDE_SIZE) * SQUARE_
SIDE_SIZE)

        hasVisitedAreaBefore = newAreaCoordinatesString in self.visitedAreas
        self.currentAreas.append(newAreaCoordinatesString)

        if self.lastArea == newAreaCoordinatesString:
            return NOT_EXITED_LAST_AREA
        if hasVisitedAreaBefore:
            return EXITED_LAST_AREA_TO_OLD
        else:
            self.visitedAreas.append(newAreaCoordinatesString)
            return EXITED_LAST_AREA_TO_NEW
```

Kodo fragmentas skirtas apdoroti esamą roboto poziciją ir atlygio metode skirti už tai atlygį. Kiekviename žingsnyje reikšmė *self.lastArea* yra priskiriama aplankytam kvadratui už kurį buvo agentui skirtas atlygis. Taip pat tikrinama ar robotas jau buvo šiame kvadratėlyje su kintamuoju *hasVisitedAreaBefore*. Jeigu kvadratas nebuvo aplankytas anksčiau, tuomet jis yra pridedamas prie aplankytų kvadratų sąrašo *visitedAreas*.



10 pav. Robotas turintis LIDAR daviklius (mėlynos linijos) mokosi virtualioje aplinkoje

Programos išvestyje (11 Pav.) yra matoma:

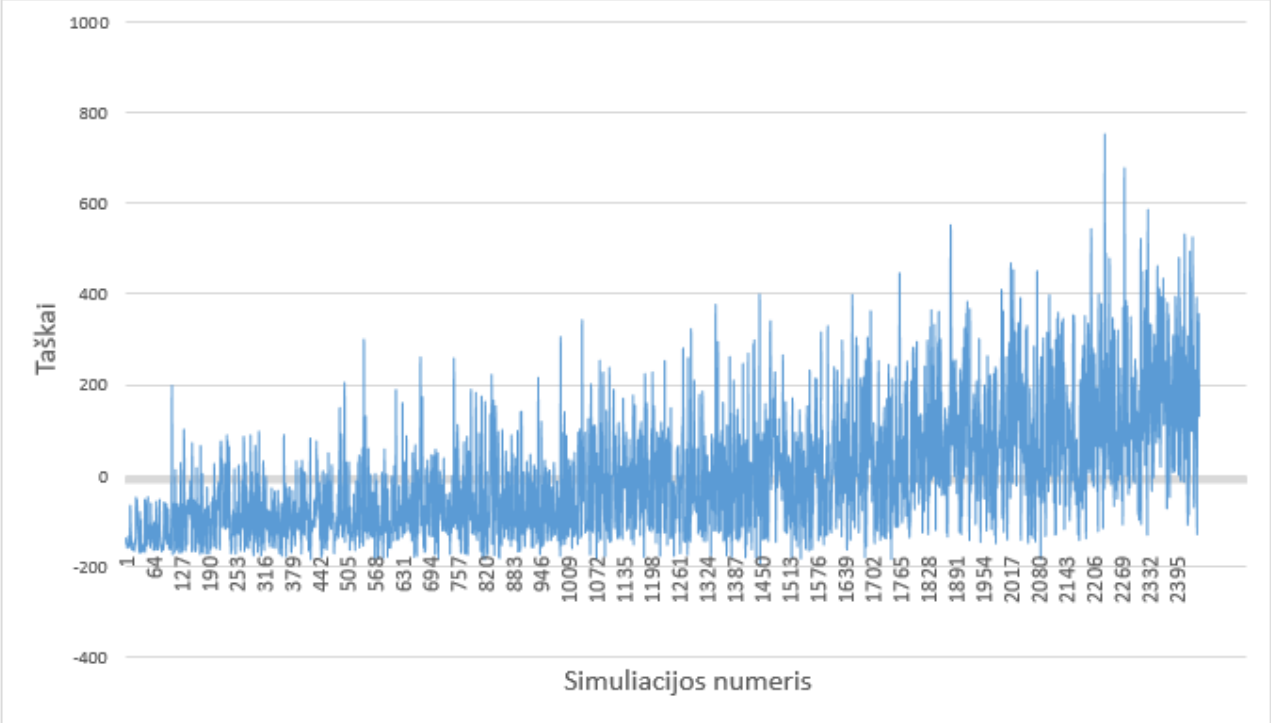
- daviklio *LIDAR* fiksuojami rodmenys (10 pav.);
- aplankyti koordinačių plokštumoje esantys kvadratai;
- roboto atliekami veiksmai;
- atlygis už atliktą veiksmą;
- bendras simuliacijoje surinktas atlygis;
- sekantis roboto veiksmas.

```
[WARN] [1578482397.735105]: NOT done Validation >>> item=1.04820072651
[WARN] [1578482397.735696]: NOT done Validation >>> item=0.749586343765
[WARN] [1578482397.736237]: NOT done Validation >>> item=0.630516231106
[WARN] [1578482397.736775]: NOT done Validation >>> item=0.671691477299
[WARN] [1578482397.737235]: NOT done Validation >>> item=1.07765638828
[WARN] [1578482397.737442]: NOT done Validation >>> item=5.07441329956
[ERROR] [1578482397.737710]: TurtleBot is Ok ==>
[WARN] [1578482397.737882]: Visited areas ==> ['0.0,-0.141421356', '0.141421356,-0.141421356', '0.2
842712,-0.141421356', '0.424264068,-0.141421356', '0.565685424,-0.141421356', '0.70710678,-0.141421
6', '0.848528136,-0.141421356', '0.848528136,0.0', '0.989949492,0.0', '1.131370848,0.0', '1.2727922
,0.0', '1.41421356,0.0', '1.555634916,0.0', '1.697056272,0.141421356']
[WARN] [1578482397.738428]: # action that we took=>1
[WARN] [1578482397.738566]: # reward that action gave=>5
[WARN] [1578482397.738717]: # cumulated_reward=>104
[WARN] [1578482397.739036]: NOT DONE
[WARN] [1578482397.739551]: Next action is:2
```

11 pav. Programos generuojama išvestis

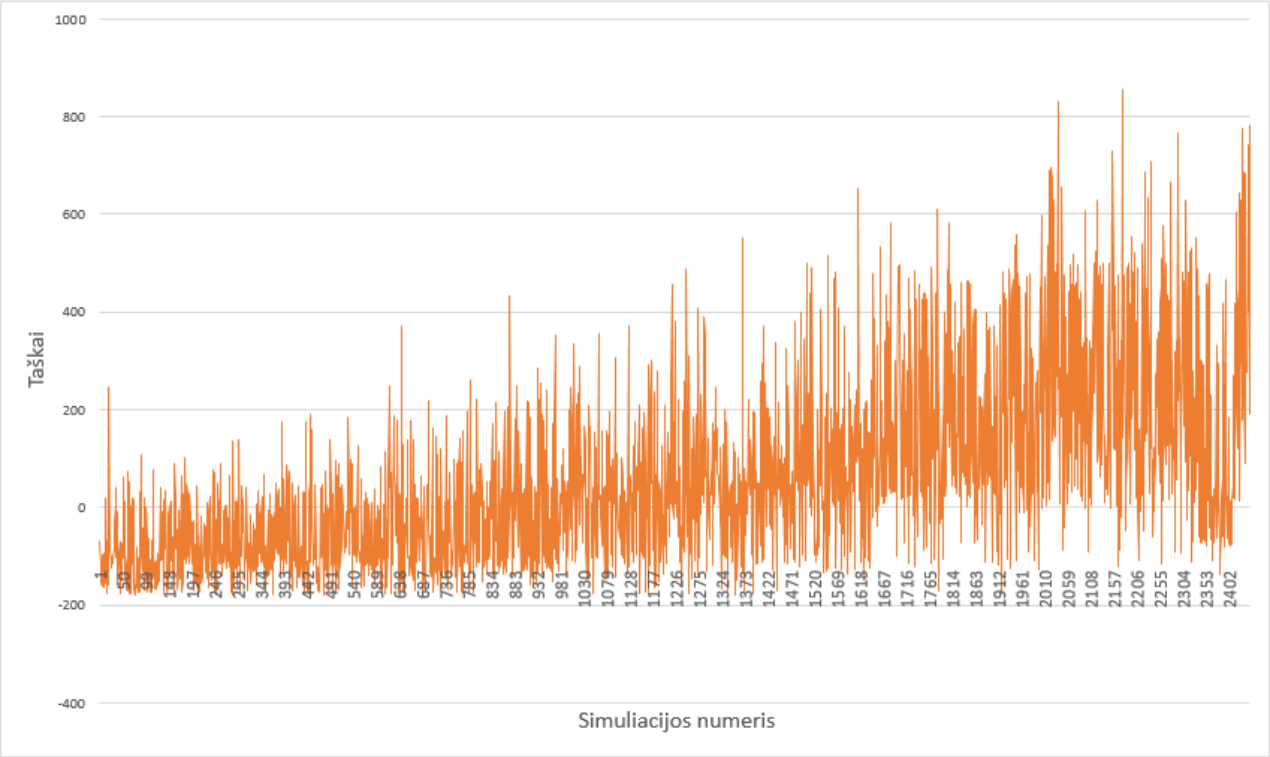
3.2. Simuliacijų rezultatai

Siekdami palyginti *Q-learning* ir *SARSA* algoritmus naudosime prieš tai aprašytą užduotį ir stebėsime programų generuojamą atlygį keičiantis simuliacijų skaičiui.



3 diagrama Q-learning algoritmo taškai mokantis aplinkoje Nr. 1

Diagramoje (3 diagrama) galime matyti kaip kinta algoritmo taškų skaičius didėjant simuliacijų skaičiui aplinkoje Nr. 1 (pav. 5). Besimokantis robotas renka vis daugiau taškų. 400 taškų ribą robotas peržengia ties 1444 simuliacija. Maksimalus surinktas taškų skaičius - 754.

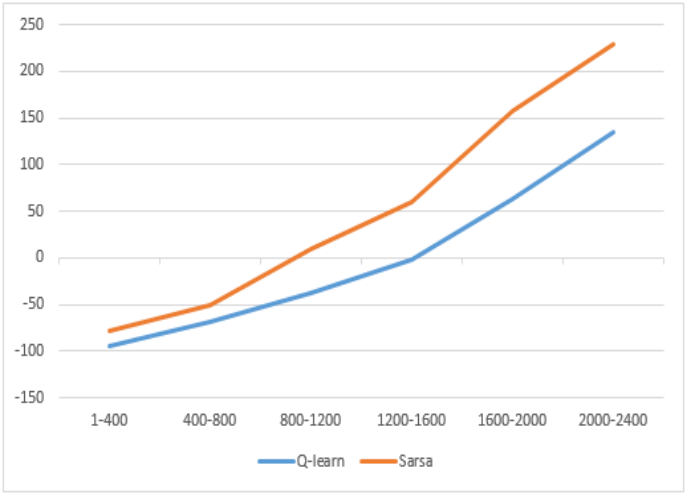


4 diagrama SARSA algoritmo taškai robotui mokantis aplinkoje Nr. 1.

Diagramoje (4 diagrama) galime matyti kaip kinta taškų skaičius naudojant *SARSA* algoritmą keičiantis simuliacijų skaičiui aplinkoje Nr. 1 (pav. 5). 400 taškų ribą robotas peržengia ties 872 simuliacija. Maksimalus surinktų taškų skaičius - 856.

Simuliacijų nr.	Q-learn	Sarsa
1-400	-94.22	-77.64
400-800	-68.60	-49.88
800-1200	-38.10	8.83
1200-1600	-2.05	59.82
1600-2000	62.84	158.04
2000-2400	134.53	229.25

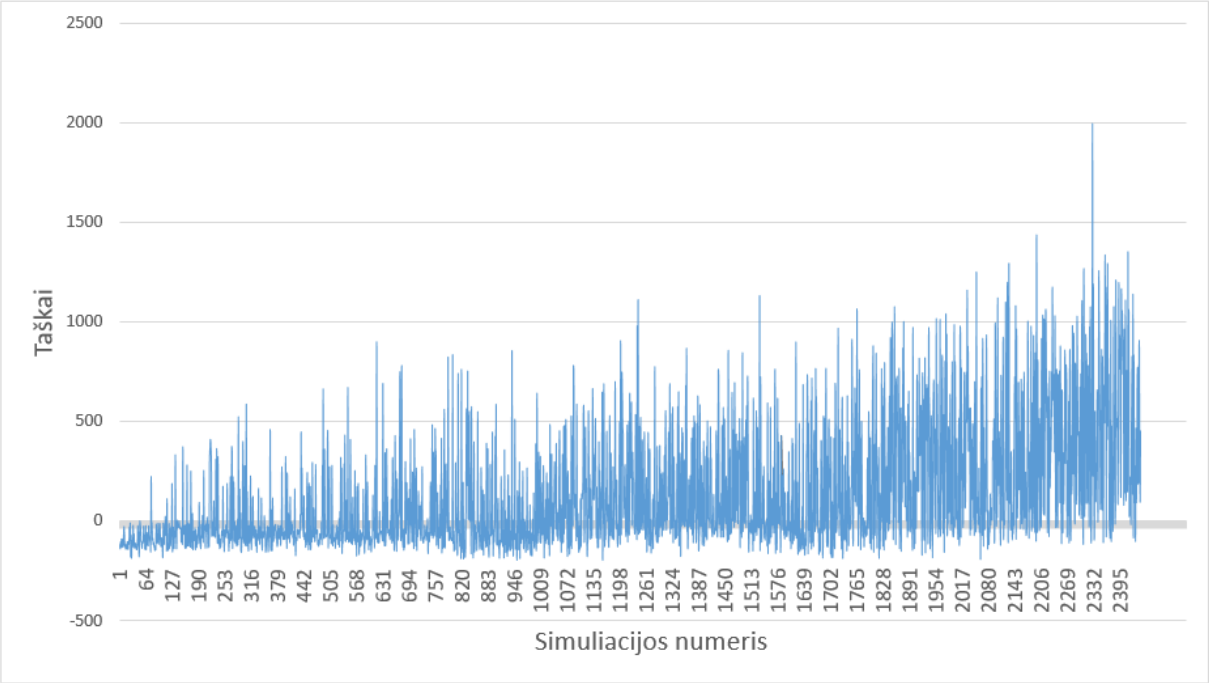
1 lentelė Vidutinis algoritmų taškų kiekis nurodytuose rėžiuose aplinkoje Nr. 1



5 diagrama Q-learning ir SARSA algoritmų vidutinių taškų grafikas aplinkoje Nr. 1

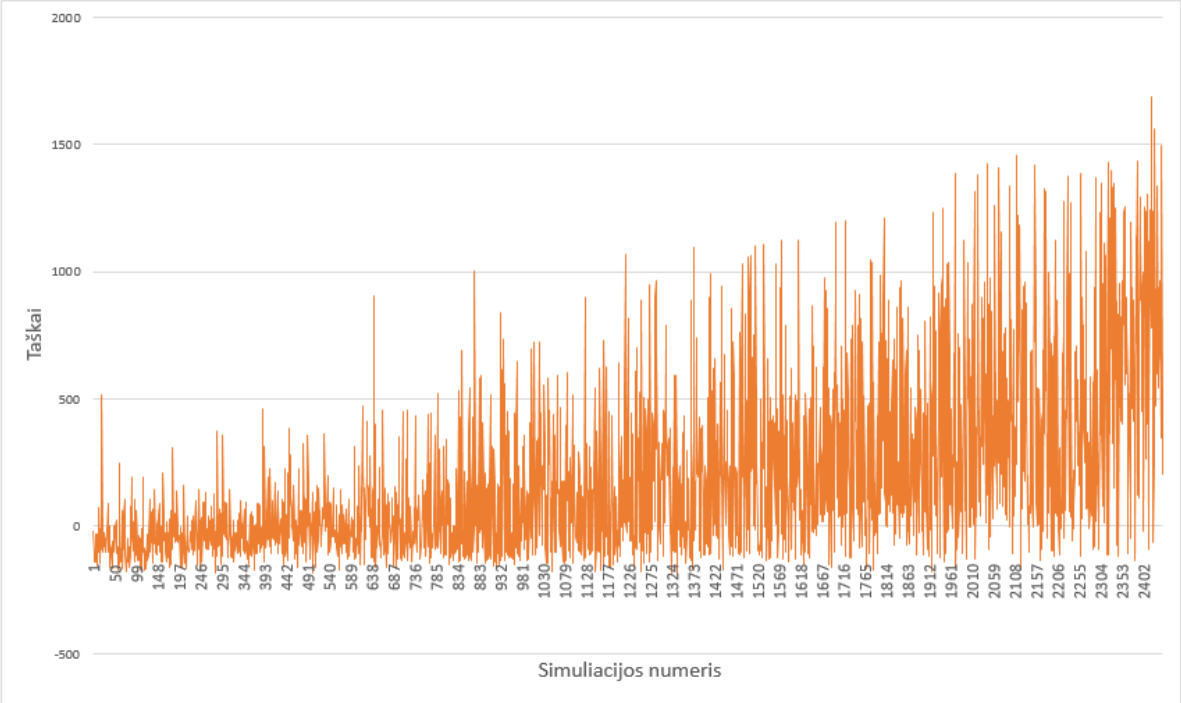
Remiantis (1 lentelė) galime teigti, jog *SARSA* algoritmas visais simuliacijos etapais surinko daugiau taškų nei *Q-learning* algoritmas.

Iš diagramos (5 diagrama) duomenų galime teigti, jog *SARSA* algoritmo mokymosi kreivė yra spartesnė palyginus su *Q-learning* tokiais pačiais simuliacijos etapais.



6 diagrama Q-learning algoritmo taškai mokantis aplinkoje Nr. 2

Diagramoje (6 diagrama) matome kaip kinta algoritmo taškų skaičius didėjant simuliacijų skaičiui aplinkoje Nr. 2 (pav. 6). Besimokantis robotas renka vis daugiau taškų. Tūkstančio taškų ribą robotas peržengia ties 1242 simuliacija. Maksimalus surinktų taškų skaičius - 1990.

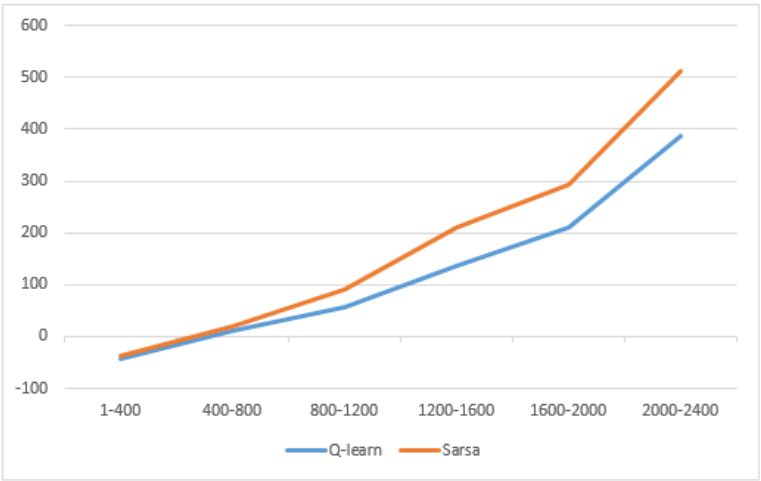


7 diagrama SARSA algoritmo taškai robotui mokantis aplinkoje Nr. 2

Diagramoje (7 diagrama) galime matyti kaip kinta taškų skaičius naudojant *SARSA* algoritmą keičiantis simuliacijų skaičiui aplinkoje Nr. 2 (pav. 6). Tūkstančio taškų ribą robotas peržengia ties 872 simuliacija. Maksimalus surinktų taškų skaičius mažesnis lyginant su *Q-learning* algoritmu – 1688.

Simuliacijų nr.	Q-learn	Sarsa
1-400	-44.23	-38.27
400-800	10.04	20.38
800-1200	57.61	91.31
1200-1600	135.93	210.83
1600-2000	211.03	293.89
2000-2400	386.45	513.22

2 lentelė Vidutinis algoritmų taškų kiekis nurodytuose rėžiuose aplinkoje Nr. 2



8 diagrama Q-learning ir SARSA algoritmų vidutinių taškų grafikas aplinkoje Nr. 2

Remiantis lentele (2 lentelė) galime teigti, jog *SARSA* algoritmas visais atvejais turėjo rinko taškų nei *Q-learning* algoritmas. Iš diagramos duomenų (8 diagrama) galime teigti, jog *SARSA* algoritmo mokymosi kreivė yra spartesnė palyginus su *Q-learning* tokiais pačiais simuliacijos etapais.

Išvados

Atlikta kelių mokslo šakų – dirbtinio intelekto, mašininio mokymosi, robotikos teorinė apžvalga ir palygintas šių temų populiarumas pasaulyje. Apžvelgti įrankiai leidžiantys apjungti šias mokslo šakas ir įgyvendinti dirbtinio intelekto metodų panaudojimą autonominiam roboto mokymuisi.

Sukurtas autonominis besimokantis robotas, kurio tikslas padengti kuo didesnį aplinkos plotą. Simuliacijos metu buvo matomas akivaizdus roboto tobulėjimas lyginant simuliacijos pradžią ir pabaigą.

Atliktas dviejų skatinamųjų mokymosi algoritmų *Q-learning* ir *SARSA* palyginimas dviejose aplinkose. Gauti rezultatai parodė, jog *SARSA* algoritmas buvo efektyvesnis, surinko didesnį maksimalų ir vidutinių taškų skaičių pirmoje ir antroje aplinkoje. Mokymosi kreivė taip pat parodė aiškų *SARSA* algoritmo pranašumą, agentas mokėsi greičiau naudodamas šį algoritmą. Galime teigti, jog nurodytai užduočiai atlikti rekomenduojama rinktis *SARSA* algoritmą.

Literatūros sąrašas

[Ark98] Ronald C. Arkin, BEHAVIOR-BASED ROBOTICS, 1998. Prieigą per internetą: http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.208.4463

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, OpenAI Gym, 2016. Prieiga per internetą: https://arxiv.org/abs/1606.01540. Žiūrėta 2019-11-12

[Bis06] Christopher M. Bishop, Pattern Recognition and Machine Learning, 2006. Prieiga per internetą: http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf. Žiūrėta 2019-12-15

[JG12] Christian P. Janssen, Wayne D. Gray When, What, and How Much to Reward in Reinforcement Learning-Based Models of Cognition, prieiga per internetą: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1551-6709.2011.01222.x. Žiūrėta 2020-02-20

[KBP13] J. Kober, J. Andrew Bagnell, J. Peters, "Reinforcement Learning in Robotics: A Survey," International Journal of Robotics Research, 2013. Prieiga per internetą: https://www.ias.informatik.tu-armstadt.de/uploads/Publications/Kober_IJRR_2013.pdf. Žiūrėta 2020-04-11

[KLM96] Kaelbling, L.P., Littman, M.L., Moore, A.W. (1996) Reinforcement Learning: A Survey, Volume 4, puslapiai 237-285. Prieiga per internetą: https://arxiv.org/abs/cs/9605103. Žiūrėta 2020-01-04

[Kur99] Ray Kurzweil. The Age of Spiritual Machines: When Computers Exceed Human Intelligence, 1999. Prieiga per internetą http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf. Žiūrėta 2020-01-03

[Mat15] Tambet Matiisen, Demystifying Deep Reinforcement Learning, 2015. Priegą per internetą http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning. Žiūrėta 2019-12-13

[Pit19] Silviu Pitis, „Rethinking the Discount Factor in Reinforcement Learning: A Decision Theoretic Approach", 2019. Prieiga per internetą: https://arxiv.org/abs/1902.02893. Žiūrėta 2020-02-11

[PMG98] David Poole, Alan Mackworth, Randy Goebel. Computational Intelligence: A Logical Approach, Oxford University Press; 1 edition, 1998, UK. Prieiga per internetą http://people.cs.ubc.ca/~poole/ci/ch1.pdf. Žiūrėta 2019-12-12

[RN18] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, second edition, 2018. Prieiga per internetą: http://incompleteideas.net/book/RLbook2020.pdf. Žiūrėta 2020-02-14

[RB14] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, 6.4 skyrius, 2014. Prieiga per internetą: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf. Žiūrėta 2019-12-02

[Sch14] J. Schmidhuber, Deep Learning in Neural Networks: An Overview, 2014. Prieiga per internetą: https://arxiv.org/abs/1404.7828. Žiūrėta 2020-05-01

[Tul18] ROS, dokumentacija, prieiga per internetą: http://wiki.ros.org/melodic/Installation/Ubuntu, http://wiki.ros.org/ROS/Tutorials. Žiūrėta 2019-10-20

[ZLV+16] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, Alejandro Hernandez Cordero, Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo, 2016. Prieiga per internetą: https://arxiv.org/abs/1608.05742. Žiūrėta 2019-11-30