VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMATIKOS KATEDRA

Neuroninių tinklų panaudojimas roboto rankos judėjimo kelio optimizavimui
Neural Networks Usage for Optimization of Robot Arm Path

Magistro baigiamasis darbas

Atliko: Informatikos 2 kurso studentas

Mykolas Banevičius

Darbo vadovas: prof. Dr. Aistis Raudys

Vilnius – 2021

Turinys.

1. Įžanga. 5
2. Temos aktualumas. 6
3. Tikslai ir uždaviniai. 7
4. Trumpa roboto rankos apžvalga, 6 laisvės laipsniai. 8
5. Trumpa roboto rankos kliūčių apėjimo problema. 9
6. Roboto rankos veikimo principas. 11
   - 6.1. Išvirkštinės kinematikos problema. 11
   - 6.2. Dviejų laisvės mazgų skaičiavimo pavyzdys dvimatėje erdvėje (2D). Paprastoji kinematika. 11
   - 6.3. Dviejų laisvės mazgų skaičiavimo pavyzdys dvimatėje erdvėje (2D). Išvirkštinė kinematika. 13
   - 6.4. Kinematika roboto rankoje. (3D) 15
   - 6.5. Alkūnės padėtis ir orientacijos nustatymas. 17
   - 6.6. Atvirkštinės kinematikos sprendimas. 19
7. Dirbtiniai neuroniniai tinklai. 20
   - 7.1. Dirbtinis neuronas. 21
   - 7.2. Dirbtinių neuronų junginiai 23
   - 7.3. Dirbtinių neuroninių tinklų mokymas. 25
   - 7.4. Daugiasluoksnis perceptronas. 25
8. Dirbtiniai neuroniniai tinklai 6 mazgų roboto rankoje. 26
   - 8.1. Šešių laisvės mazgų „Denso“ robotas. 26
     - 8.1.1. Roboto specifikacija. 26
     - 8.1.2. Tradiciniai neuroniniai tinklai. 27

8.1.3. Siūlomas dirbtinio neuroninio tinklo dizainas. 28
8.1.4. Nagrinėjamas uždavinys. 29
8.1.5. Rezultatai. 30
8.2. Neuroninių tinklų, šešių mazgų roboto rankos judėjimas su kompensacija. 31
8.2.1. Dirbtinio neuroninio tinklo atvirkštinio sprendimo (backpropogation) struktūra. 31
8.2.2. Rezultatai. 33
9. Simuliacinės aplinkos. 34
9.1. OpenAI Gym 34
8.1.1 OpenAI Gym aplinkos. 34
9.2. MATLAB simuliacinė aplinka. 35
9.2.1. MATLAB simuliacinės aplinkos kodas. 36
10. Sustiprinimo mokymasis (reinforcement learning). 37
10.1.1. Pagrindiniai RL aspektai. 37
10.1.2. RL ir (supervised) mokymo skirtumai bei RL tipai. 37
11. Optimizacija. 38
11.1. PA-10 manipuliatoriaus optimizacijos uždavinys. 38
11.2. PA-10 siūlomas optimizavimo algoritmas. 40
12. Praktinis aštuonių laisvės mazgų roboto rankos įgyvendinimas. 41
12.1. Simuliacinė aplinka. 41
12.1.1. Unity ML Įrankis. 42
12.1.2. Agento smegenys. 43
12.2. Aštuonių laisvės mazgų roboto ranka. 44
12.2.1. Rankos laisvės mazgų tipai. 44
12.2.2. Sukurtos 8 laisvės mazgų rankos modelis. 45
12.3. Aštuonių laisvė mazgų roboto rankos mokymas bei AI specifikacija. 47

12.3.1. Roboto rankos pagrindinis tikslas, AI išvestis, duomenų atvaizdavimo būdas. ... 47
12.3.2. Markov sprendimų procesas (MDP). ... 48
12.3.3. Atlygio schema. ... 49
12.3.4. Mokymo modelio konfigūracija. ... 50
12.3.5. Agentas (kodas). ... 50
12.4. Mokymosi rezultatai bei išvados. ... 52
12.4.1. Aštuonių laisvės mazgų mokymosi rezultatai be kliūčių (5 mln.). ... 52
12.4.2. Aštuonių laisvės mazgų mokymosi rezultatai su kliūtimis. (10 mln.). ... 52
12.5. Mokymosi rezultatų palyginimas. ... 54
13. Aštuonių laisvės mazgų mokymosi rezultatai su kliūtimis – pasisekęs bandymas. ... 54
13.1. Ištaisytos praeities bandymų klaidos. ... 54
13.2. Rezultatai. ... 55
14. Praktinis šešių laisvės mazgų roboto rankos įgyvendinimas. ... 56
14.1. Simuliacinė aplinka bei roboto ranka. ... 56
14.2. Rezultatai. ... 57
15. Dalinis Markov sprendimų procesas. (*Semi-MDP*). ... 58
16. Dalinio Markov sprendimų proceso įgyvendinimas. ... 59
16.1. Duoto laiko skaičiavimas iteracijai. ... 59
16.2. Laiko sekimas iteracijoje. ... 59
16.3. Rezultatai. ... 59
17. Išvados ... 60
18. Santrauka (*Summary*). ... 61
19. Literatūros sąrašas. ... 62
20. Priedai. ... 65

1. Įžanga.

Sparčiai modernizuojamame 21 - o amžiaus pasaulyje, robotai pradeda užimti reikšmingą vaidmenį. Žvelgiant į istoriją, robotizuoti manipuliatoriai bei rankos buvo patys pirmi robotai naudoti industrijoje. Jie buvo naudoti nuo 1960 -ųjų metų [WMH+12]. Šie, ganėtinai nauji, kūriniai padeda žmonėms automatizuoti didžiąją darbų dalį gamyklose. Pavyzdžiui, „BMW grupė“ (BMW group), robotų pagalba, per diena gali pagaminti iki 1000 automobilių [Cab12], kas būtų beveik neįmanoma, arba reikalautų labai daug pajėgų bei resursų, jei gamybai nebūtų naudojamasi mechanizmų pagalba.

Viena iš populiariausių šių laikų technologijų yra neuroniniai tinklai. Ši technologija yra algoritmų visuma, priklausomai nuo srities, kuri „apmoko“ įrenginį, jog šis galėtų atlikti tam tikrus darbus. Keli realūs pavyzdžiai būtų, objektų atpažinimas iš nuotraukos, filmuotos medžiagos. Šis pavyzdys yra labai plačiai naudojamas pradedant medicina, pavyzdžiui, vėžinių ląstelių atpažinimas nuotraukoje, baigiant neleistinų vaizdų atpažinimu Internete. Taip pat neuroniniai tinklai, atsižvelgiant į praeities duomenis, gali prognozuoti akcijų kainas ateityje. Prognozuoti gali ne tik skaitmeninius duomenis, bet ir nuotraukas. Pastaruoju laikotarpiu yra tobulinami algoritmai, kurie išryškina nuotraukas, skaitmeninę medžiagą. Šie metodai padeda apsaugos tarnyboms atpažinti kamerose nufilmuotus įstatymo pažeidėjus. Neuroninių tinklų panaudojimo sričių yra begalės. Juos galima panaudoti visur, kur tik įmanoma kompiuterį apmokyti daryti tam tikras užduotis. Šio magistrinio darbo vienas iš uždavinių bus apmokyti kompiuterį apskaičiuoti roboto rankos judėjimo optimizuotą kelią. „Inžinerinio projekto optimizavimas yra sistemingas procesas, naudojant projektavimo apribojimus ir kriterijus, leidžiančius dizaineriui surasti optimaliausią sprendimą“, - teigia Tod. R. Kelley [Kel10].

2. Temos aktualumas.

Žmonės robotų rankų panaudojimą mato ne tik gamybos bei konstravimo srityse, bet ir kasdieniniame gyvenime. Nuo 2011 – ųjų metų lapkričio mėnesio iki 2012 – ųjų metų sausio mėnesio buvo atliekama apklausa apie žmonių įsivaizdavimą, kur būtų galima panaudoti roboto ranką. Apklausoje dalyvavo 96 dalyviai, iš kurių 58 buvo tailandiečiai ir 38 japonai. Visi žmonės, dalyvavę apklausoje, neturėjo jokios patirties su robotizuotomis rankomis. Rezultatai buvo tokie, jog net 46 % atsakymų vyravo roboto rankos panaudojimas namų ūkio darbuose, 21 % darbo vietoje bei pagalboje „prie stalo“. Taip pat tarp dažniausiai paminėtų sričių atsakymuose pateko medicinos, judėjimo negalią turinčių žmonių pagalbos bei karo pramonės sritys [1].

Dar viena sritis, kuri taip pat buvo paminėta ir aukščiau aptartoje apklausoje, kur galima efektyviai panaudoti robotizuotas rankas, yra medicinos. Šioje srityje, atliekant sudėtingas ir ne tik, operacijas, būtinas kuo didesnis tikslumas. Naudojant robotizuotas rankas, operacijų metu, yra išvengiami kai kurie žmogaus daromi faktoriai, kaip operuojančio chirurgo rankos drebėjimas, per ilgas operacijas sumažėjęs tikslumas ir kiti. Taip pat, naudojant robotizuotas rankas galima prieiti prie giluminių operuojamo žmogaus kūno vietų, darant pakankamai mažus pjūvius [WOS4]. Naudojimas neuroninius tinklus roboto rankos kelio optimizavimui leidžia sutaupyti daromas operacijas, kas yra labai svarbu medicinos srityje. Daugelis operacijų visame pasaulyje dažnai nepasiseka, kaip būna planuota, vien tik dėl ilgo operacijos laiko. Robotizuotos rankos operacijas padaro itin tikslias bei neuroninių tinklų pritaikymas optimizuojant rankos keliavimo kelią sutrumpina operacijos laiką bei pjūvio dydžius.

Taip pat, neuroniniais tinklais pagrysta roboto ranka gali veikti dinamiškai bei prisitaikyti prie aplinkos pakeitimų. Jei roboto ranka turi sensorius, pavyzdžiui, atstumo jutiklius iki kliūties ar vaizdo medžiagą, ranka, naudodama neuroninius tinklus, gali sugebėti prisitaikyti prie bet kokios aplinkos ir optimizuotai pasiekti kelio tikslą.

Kaip yra matoma iš apklausos rezultatų, žmonių roboto rankos panaudojimo aplinkos įsivaizdavimas yra daugiausiai susijęs su kasdieninio gyvenimo palengvinimu buityje bei medicina. Kolkas ši sritis yra daugiausiai paplitusi, kaip ir buvo minėta įžangoje, pramonės srityje bei gamyboje, tad yra daug erdvės tobulėjimui kitose srityse. Pramonėje smarkiai paplitusios šešių laisvės mazgų roboto rankos. Neuroniniai tinklai pritaikyti, jog suoptimizuotų robotizuotos rankos judėjimą keliu, visapusiškai sutrumpina rankos judėjimo darbo laiką bei sutrumpina judesio amplitudę. Taip pramonėje, gamyklose yra sutaupomi elektros resursai bei gamybos laikas, medicinoje yra padidinami kokybiškos bei greitos operacijos rodikliai.

3. Tikslai ir uždaviniai.

Kaip jau buvo minėta įžangoje, šio magistrinio darbo metu bus siekiama įgyvendinti vieną pagrindinių tikslų – pritaikyti gilaus pastiprinto neuroninio tinklo algoritmą aštuonių laisvės mazgų roboto rankos optimizavimui ir tikslo pasiekimui.

Jog šie tikslai būtų įgyvendinti, tam buvo išsikelti uždaviniai, kuriuos įgyvendinant bus galima lengviau pasiekti išsikeltus tikslus. Uždaviniai:

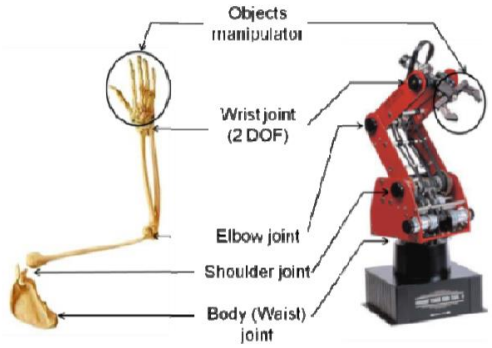
- Išsiaiškinti roboto rankos veikimo principus.
- Ištirti neuroninių tinklų panaudojimus optimizavimo srityje.
- Apmokyti pritaikytą neuroninį tinklą roboto rankos judesio kelio optimizavimui.
- Ištestuoti apmokytą tinklą.
- Vizualiai susimuliuoti gautus rezultatus, jog būtų matomas aiškus darbo rezultatas.

4. Trumpa roboto rankos apžvalga, 6 laisvės laipsniai.

Robotai, kurie turi aukštą laisvo judėjimo rodiklį (*DOF – degrees of freedom*) savo jungtyse, yra turbūt svarbiausia, bet kokio, roboto modeliavimo tema. Ganėtinai populiarus ir išspręstas uždavinys yra šešių laisvės mazgų roboto rankos judėjimas. Problema kinematinio valdymo yra dažniausiai skiriama į dvi dalis:

- Tiesioginė (*Forward or direct*) kinematika, kuri yra mechanizmo Dekarto pozicijos ir orientacijos problemos sprendimo būdas.
- Atvirkštinė (*Inverse*) kinematika, kuri skaičiuoja roboto rankos mazgų reikšmes pagal galinio efektoriaus (*end-effector*) poziciją bei orientaciją. Ši problema yra ganėtinai sudėtingesnė nei TK (tiesioginės kinematikos) [IIK12].

Paveikslėlyje nr. 2 matoma 6 mazgų roboto rankos suvienodinimas su žmogaus ranka.



Pav. 1. 6 mazgų roboto rankos palyginimas su žmogaus ranka

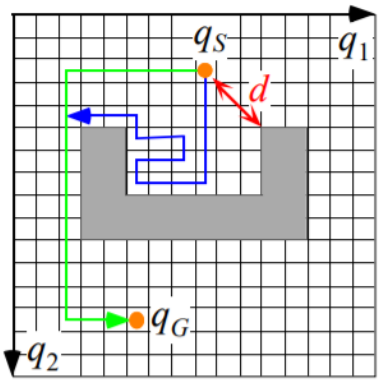
Kiekvienas iš šių mazgų turi po vieną DOF, išskyrus riešo mazgą, šis turi net du. Taigi, roboto „riešas“ gali judėti dviem plokštumomis: riedėjimo bei žingsnio (*roll and pitch*), dėl ko galutinis efektorius (*end – effector*) tampa lankstesnis objekto atžvilgio manipuliacijai. Daugiau apie roboto rankos judėjimą, jo optimizavimą naudojantis neuroniniais tinklais bei konstrukcijas bus nagrinėjama vėlesniuose skyriuose.

5. Trumpa roboto rankos kliūčių apėjimo problema.

Labiausiai vertinami robotai yra tie, kurie sugeba prisitaikyti dinaminėje aplinkoje. Tokioje aplinkoje gali būti išskiriami keli skirtingi atvejai:

- Kai aplinkoje yra objektų (kliūčių).
- Kai robotas laiko skirtingų dydžių bei formų objektus. Tokiu atveju, roboto ranka gali pasiekti kelionės tikslą, tačiau laikydama tam tikro dydžio objektą, šis gali sutrukdyti įveikti kelią.

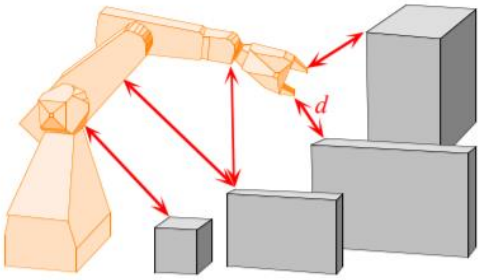
Kai aplinkoje yra dinaminių objektų, kurie kliudo pasiekti roboto rankos kelionės tikslą, galima šią problemą išspręsti keliais būdais. Pirmasis būdas būtų naudoti „C-space“ metodą, kuris roboto rankos keliavimą erdvėje sukonvertuotu į C formą. Tai reiškia, jog mechanizuota ranka keliautų lanku aplink kliūtį. Paveikslėlyje nr. 3 žalia linija iš Qs iki Qg parodo šį metodą dviejų dimensijų erdvėje. Taip pat yra sugalvoti dirbtinio intelekto algoritmai kaip A, A* ir kiti, kurie dinamiškai skaičiuoja kelio žingsnius ir bando glaustai apeiti kliūtį. Trečiame paveikslėlyje tai būtų mėlyna linija [GFG].



Pav. 2. C-space ir A algoritmai.*

Kai robotas laiko skirtingų dydžių bei formų objektus anksčiau išvardinti kelio paieškos algoritmai yra netinkami, nes roboto rankos bei laikomo objekto visumos dydis gali gerokai padidėti ir ranka su objektu gali kliudyti kliūtį. Yra sugalvotas būdas spręsti šią problemą. Mechanizuotos rankos su laikomu objektu kliūties apėjimui gali padėti atstumo laikymasis (žr. pav.

4). Jei roboto rankos dalyse būtų galima sumontuoti daviklius arba kameras, kurios fiksuotų ranką bei kliūtis, būtų galima palaikyti tam tikrą atstumą nuo kliūties ir skaičiuoti kelią iki koordinačių [HWW].



Pav. 3. Roboto rankos atstumo laikymasis.

Šie aukščiau aptarti keletas būdų robotizuotai rankai pasiekti kelionės tikslą yra ganėtinai lėti bei reikalaujantys daug resursų, tad šio magistrantūros darbo metu bus stengtasi naudojantis neuroniniais tinklais sumažinti rankos naudojamus resursus.

6. Roboto rankos veikimo principas.

Kaip jau buvo minėta ankstesniame skyriuje, kur buvo trumpai apžvelgiama roboto ranka, pastebėjome, jog ji, neatsižvelgiant į išvaizdą, yra labai panaši į žmogaus ranką. Jau aptartas, šešių laisvės mazgų, roboto rankos judėjimas gali pilnai atstoti žmogaus rankos judėjimą. Roboto rankos laisvės mazgų judėjimo spindulys net gali būti didesnis nei realaus žmogaus rankos. Tačiau įdomu, kaip yra tai pasiekiama, kaip ji iš tikrųjų veikia. Tad šiame skyriuje bus glaustai apibendrinta ir paaiškinta roboto rankos judėjimo paslaptis.

6.1. Išvirkštinės kinematikos problema.

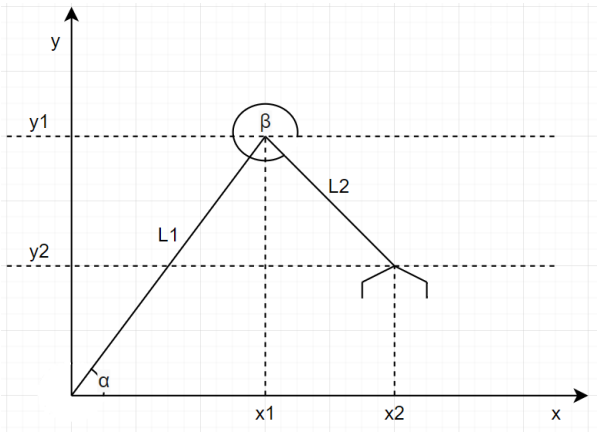
Išvirkštinė kinematika, trumpai tariant, yra funkcija (žr. pav. 5), kuri pasako, kaip ir kiek kiekvienas roboto rankos laisvės mazgas turi būti pasisukęs, jog ranka būtų tam tikroje pozicijoje erdvėje. Toliau bus pateikta lengvų bei sunkesnių pavyzdžių.

6.2. Dviejų laisvės mazgų skaičiavimo pavyzdys dvimatėje erdvėje (2D). Paprastoji kinematika.

Šiame skyrelyje bus aptartas pavyzdys, kaip yra skaičiuojami roboto rankos dviejų laisvės mazgų kampai dvimatėje erdvėje. Paveikslėlyje nr. 6 matome dviejų laisvės mazgų roboto ranką dvimatėje erdvėje[Twa18]. Įsivaizduokime, jog mums reikia pajudinti šią ranką, bei priversti ją paspausti mygtuką, kuris yra x = 200 mm. ir y = 20 mm. koordinatėse. Šios užduoties palengvinimui, tarkime jog α = 80°, β = 350°, L_1 = 130 mm., L_2 = 180 mm. Jog rastume, kuriame taške yra roboto rankos galas, pritaikysime taško radimo tiesėje formules $x = x_2 = L_1 \cos\alpha + L_2 \cos\beta$ ir $y = y_2 = L_1 \sin\alpha + L_2 \sin\beta$. Įstačius į šias formules atitinkamą duotą informaciją, gauname lygtis

- $x = x_2 = L_1 \cos(\alpha) + L_2 \cos(\beta) = 130mm \cos(80°) + 180mm \cos(350°) = 22{,}57\ mm + 177{,}27\ mm = 199{,}84\ mm.$
- $y = y_2 = L_1 \sin(\alpha) + L_2 \sin(\beta) = 130mm \sin(80°) + 180mm \sin(350°) = 127{,}4\ mm - 30{,}6\ mm = 96{,}8\ mm.$

Taigi, pagal viršuje atliktus skaičiavimus, gavome, jog roboto rankos galo koordinatės dvimatėje koordinačių plokštumoje yra x = 199,84 mm. ir y = 96,8 mm. Šis skaičiavimo būdas yra vadinamas „paprastąja kinematika“.



Pav. 6 3 laisvės mazgų roboto ranka.

Roboto rankos judesiai yra atvaizduojami matricomis (dviejų matmenų masyvais) (žr. pav. 7). Lengviausias būdas sužinoti kelią tarp roboto rankos galutinio taško koordinatės iki galutinio tikslo yra apskaičiuoti α ir β kampus kiekvienam galimam roboto rankos taško judėjimo krypties koordinačių dvejetui. Taip žinosime, kiekvieno roboto rankos laisvės mazgo pasisukimą laipsniais, kiekvienoje roboto rankos galo koordinatėje.

$$\alpha = \begin{bmatrix} 99 & 98 & 98 & 98 & 97 & 96 & 96 & 95 & 94 & 92 \\ 97 & 97 & 97 & 96 & 95 & 95 & 94 & 93 & 92 & 91 \\ 95 & 95 & 95 & 94 & 94 & 93 & 92 & 92 & 90 & 89 \\ 94 & 94 & 93 & 93 & 92 & 92 & 91 & 90 & 89 & 88 \\ 92 & 92 & 92 & 91 & 91 & 90 & 89 & 88 & 87 & 86 \\ 90 & 90 & 90 & 90 & 89 & 88 & 88 & 87 & 86 & 84 \\ 89 & 89 & 88 & 88 & 88 & 87 & 86 & 85 & 84 & 82 \end{bmatrix}$$

$$\beta = \begin{bmatrix} 111 & 111 & 111 & 112 & 113 & 113 & 114 & 115 & 116 & 117 \\ 112 & 112 & 113 & 113 & 114 & 114 & 115 & 116 & 117 & 118 \\ 113 & 113 & 114 & 114 & 115 & 116 & 116 & 117 & 118 & 119 \\ 114 & 115 & 115 & 115 & 116 & 117 & 117 & 118 & 119 & 120 \\ 115 & 116 & 116 & 117 & 117 & 118 & 119 & 119 & 120 & 121 \\ 116 & 117 & 117 & 118 & 118 & 119 & 120 & 120 & 121 & 122 \\ 117 & 118 & 119 & 119 & 119 & 120 & 121 & 122 & 122 & 123 \end{bmatrix}$$

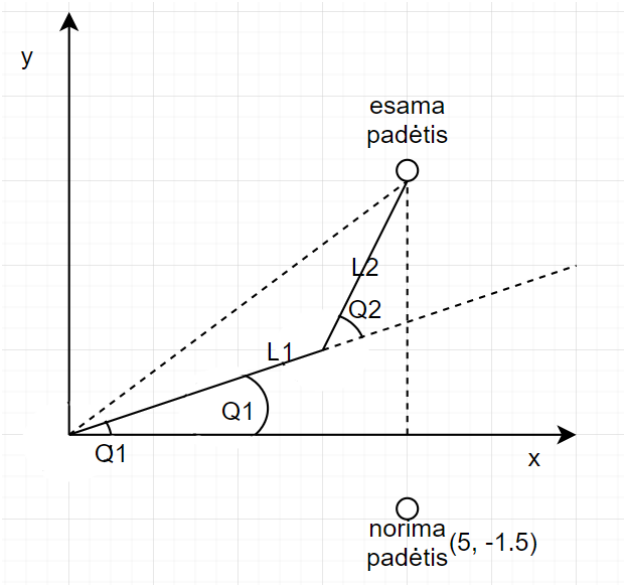
Pav. 7 Roboto rankos judesių matricos.

Taip pat roboto rankos judėjimo skaičiavimams galima naudoti ir kitokį metodą. Jei prieš tai skaičiavome roboto rankos galo koordinates pagal duotus kampus bei kraštines, kitas būdas

skaičiuoja kokioje pozicijoje turi būti roboto ranka, jog galo koordinatės sutaptų su tikslo koordinatėmis. Kitaip tariant, yra skaičiuojamas roboto rankos laisvės mazgų kampas. Šis metodas vadinasi „atvirkštinė kinematika“.

6.3. Dviejų laisvės mazgų skaičiavimo pavyzdys dvimatėje erdvėje (2D). Išvirkštinė kinematika.

Trumpai išnagrinėsime išvirkštinės kinematikos pavyzdį [UOR]. Pagal pav. nr. 8, jei mes norime, jog roboto rankos „esama padėtis“ nukeliautų iki „norima padėtis“. Norint tai įvykdyti, pradžioje reikia įsitikinti, jog roboto ranka yra pajėgi pasiekti norimas koordinates. Motorizuota ranka gali pasiekti minimalų $L_1 - L_2$ atstumą nuo pradinės pusės, jei alkūnė yra sulenkta taip jog dilbis atsidurtų ant viršutinės rankos. Taip pat, jei ranka ištiesta tiesiai, ji gali pasiekti maksimalų atstumą $L_1 + L_2$ nuo pražios vietos.

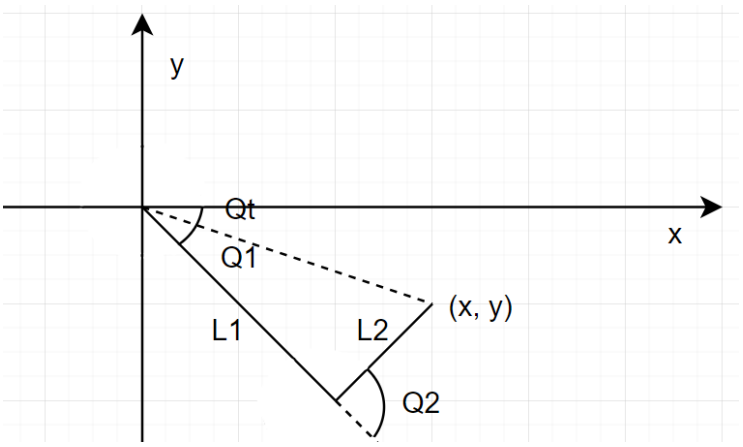


Pav. 8 Roboto ranka dvimatėje erdėje.

Atstumą iki norimos lokacijos (x, y) nuo pradinės padėties galima rasti naudodami formulę $\sqrt{x^2 + y^2}$. Taigi, pasinaudojus, šia formule galima įsitikinti, ar norimos padėties taškas yra rankos pasiekiamumo diapazone.

$$L_1 - L_2 \leq \sqrt{x^2 + y^2} \leq L_1 + L_2 \;\Rightarrow\; 4 - 2 \leq \sqrt{5^2 + (-1.5)^2} \leq 4 + 2 \Rightarrow\; 2 \leq \sqrt{5.22} \leq 6$$

Matome, jog norimas taškas, koordinačių plokštumoje, yra pasiekiamas, tad galima ieškoti motorizuotos rankos laisvės mazgų kampų (žr. pav. 9.).



Pav. 9 Roboto ranka dvimatėje erdvėje.

Pasinaudojus kosinuso taisykle ($\cos(A) = \frac{b^2+c^2-a^2}{2ac}$), surandame Q1 ir Q2 kampus.

$$Qt = cos^{-1}\left(\frac{x}{\sqrt{x^2+y^2}}\right) = -16.7°$$

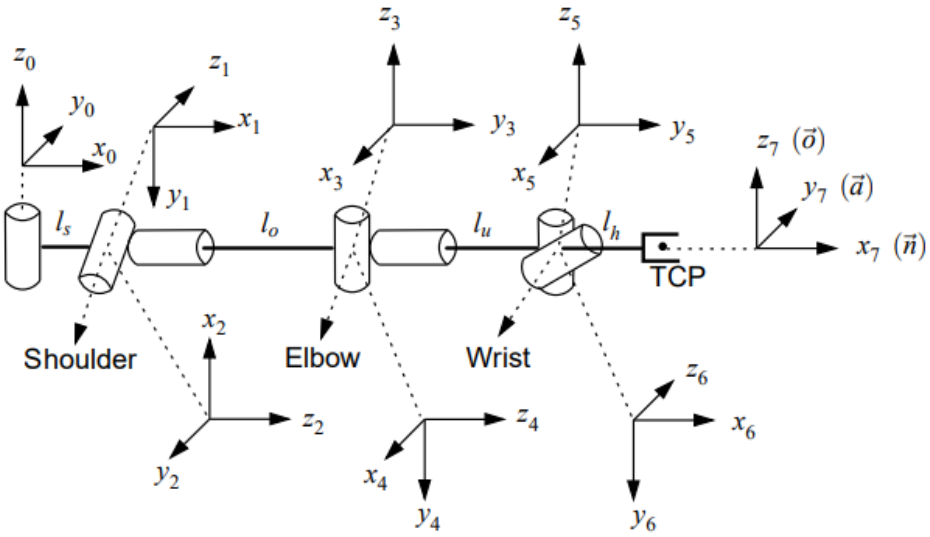
$$\cos(Q1 - Qt) = \frac{L_1^2+\left(\sqrt{x^2+y^2}\right)-L_2^2}{2L_1\sqrt{x^2+y^2}} => Q1 = cos^{-1}\left(\frac{L_1^2+x^2+y^2-L_2^2}{2L_1\sqrt{x^2+y^2}}\right) + Qt = \text{-}36.7°$$

$$Q2 = -cos^{-1}\left(\frac{L_1^2+L_2^2-(x^2+y^2)}{2L_1L_2}\right) + 180° = 63°$$

Gauti rezultatai: roboto rankai reikia pasukti Q1 kampą -36.7° ir Q2 63°, kad pasiektų norimas koordinates (5, -1,5).

Šiuose dvejuose poskyriuose yra pateiktu du, paprastosios kinematikos ir atvirkštinės kinematikos, pavyzdžiai. Išvirkštinė kinematika yra dažniausiai naudojama ir plačiai paplitusi robotikos srityje. Išvirkštinė kinematikos metodas yra naudojamas ne tik kuriant roboto rankas, bet ir kitus komponentus, kurie reikalauja rotacijos kaip žmogaus sąnarys.

6.4. Kinematika roboto rankoje. (3D)



Pav. 10 Rankos koordinačių sistema. Nuotrauka paimta iš [AD03] šaltinio.

Dešimtame paveikslėlyje yra pavaizduota septynių laisvės mazgų roboto rankos posūkių bei judėjimo krypčių pavyzdys, naudojant Denavit – Hartenberg konvenciją[AD03]. Ši ranka susideda iš trijų laisvės mazgų dalių: peties, alkūnės bei riešo. Rankos duomenys yra pavaizduoti pirmoje lentelėje (žr. lent. 1).

i	θ_i	α_i	$a_i[mm]$	$d_i[mm]$	range
1	θ_1	$-90°$	l_s	0	$-90° \cdots 90°$
2	$\theta_2 - 90°$	$-90°$	0	0	$-90° \cdots 90°$
3	$\theta_3 + 90°$	$90°$	0	l_u	$-230° \cdots 90°$
4	θ_4	$-90°$	0	0	$0 \cdots 145°$
5	θ_5	$90°$	0	l_f	$0 \cdots 350°$
6	$\theta_6 + 90°$	$-90°$	0	0	$-45° \cdots 45°$
7	θ_7	$90°$	l_h	0	$45° \cdots 45°$

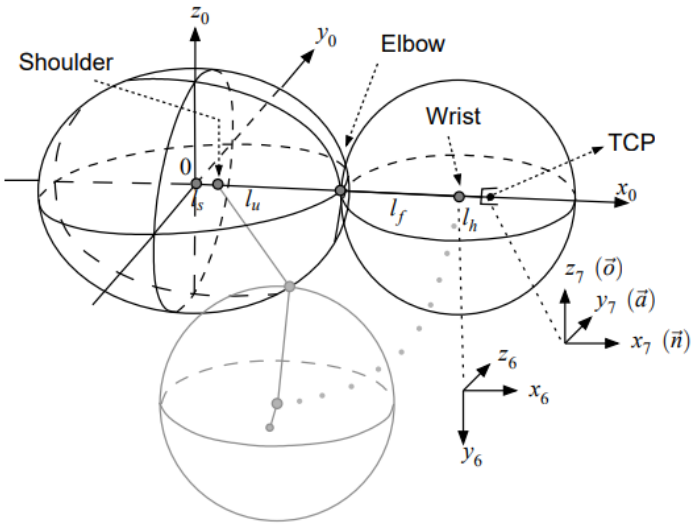
Lent. 1 Rankos duomenys. Lentelė paimta iš [AD03] šaltinio.

Denavit-Hartenberg konvencija leidžia sukurti kinematikos funkciją sudedant koordinačių transformacijas į vieną vienalytė transformacijos matrica. Aprašymas tarp koordinačių transformacijos tarp i ir i – 1 yra nurodyta žemiau pavaizduotoje vienalytėje transformavimo matricoje. Matricos *Ai* simboliai $s\ ir\ c$ reprezentuoja sinusą ir kosinusą.

$$A_i = \begin{pmatrix} c_{\theta_i} & -c_{\alpha_i} s_{\theta_i} & s_{\alpha_i} s_{\theta_i} & a_i c_i \\ s_{\theta_i} & c_{\alpha_i} \theta_{\theta_i} & -s_{\alpha_i} c_{\theta_i} & a_i s_i \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

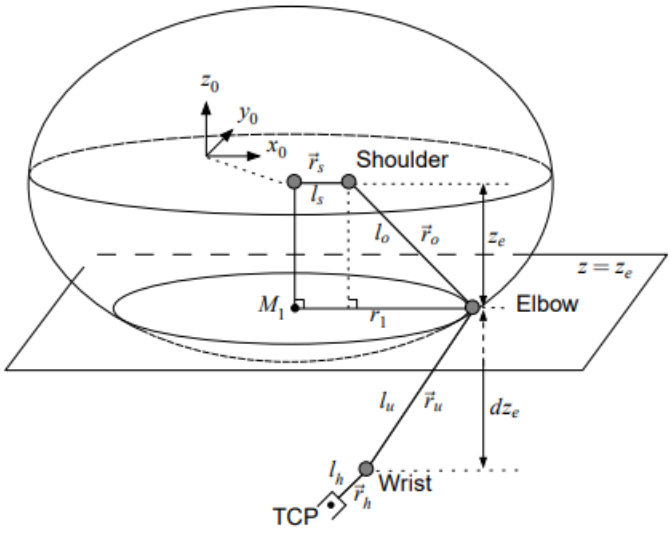
Apibrėžus kiekvieną kadrą, koordinačių transformacija, apibūdinanti galinio efektoriaus poziciją bei orientaciją bazinio rėmo atžvilgiu yra $T_{tcp}(\sim \theta) = \prod_{i=1}^{7}\left(A_i(\theta_i)\right)$, kur ~ θ yra jungtinių kintamųjų (7 × 1) vektorius, ~ n, ~ o ir ~ a yra rėmelio, pritvirtinto prie galinio efektoriaus, o ~ p yra vietos padėties vektorius atskaitos rėmo atžvilgiu (x_0, y_0, z_0).

Roboto rankos atleidimą galima apibūdinti alkūnės sąnario centro sukimasis apie savo ašį, kuris praeina per riešą ir pagrindą (x_0, y_0, z_0). Roboto rankos alkūnės padėtis aplink šią ašį yra apibrėžta kreive. Taip pat, dilbio taškas vienuoliktame paveikslėlyje nurodo sferą, kurios centras yra riešas (x_w, y_w, z_w), kuris atitinka koordinačių sistemą (x_6, y_6, z_6). Kadangi šie du taškai turi sutapti, atleidimo kreivė susidaro iš elipsoido ir sferos sankirtos. Vienuoliktame paveikslėlyje matome elipsoidą, kuris apskaičiuojamas pagal $\frac{x}{(l_s+l_o)^2} + \frac{y}{(l_s+l_o)^2} + \frac{z}{(l_o)^2} = 1$. Taip pat, matome sferą, kurios spindulys yra l_f ir centras yra duotas pagal riešo vektoriaus padėtį $\overrightarrow{w} = (x_w, y_w, z_w)^T = \vec{p} - l_h * \vec{n}$, kur $\vec{p}$ yra galinio efektoriu vektoriau pozicija, l_h yra distancija tarp riešo ir $\vec{n}$ vektoriaus galinio efektoriaus.



Pav. 11 Viršutinės rankos ir dilbio atvaizdavimas. Nuotrauka paimta iš [10] šaltinio.

6.5. Alkūnės padėtis ir orientacijos nustatymas.



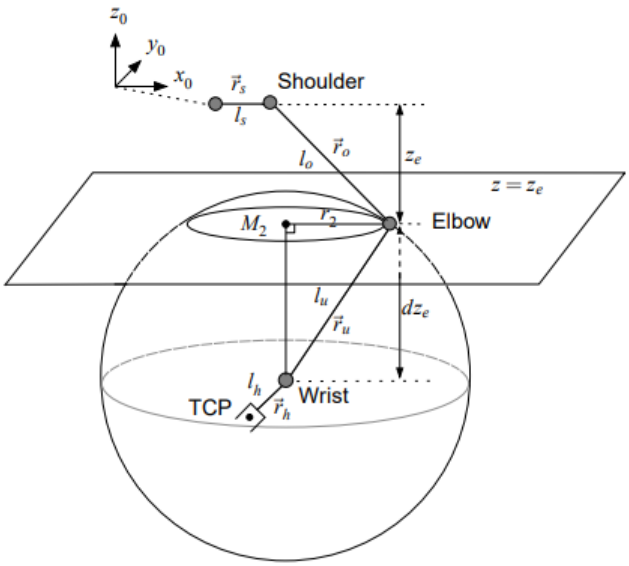
Pav. 12 Elipsoidas ir plokštuma $z = z_e$. Nuotrauka paimta iš [10] šaltinio.

Tarkime, jog $z_e \in [z_e^{min}, z_e^{max}]$ yra alkūnės z koordinatės padėtis. Elipsės sankirta su plokštuma $z = z_e$ (žr. pav. 12) sudaro apskritimą. Galime gauti spindulį r_1 ir apskritimo centrą M_1 pagal duotas formules:

- $r_1 = l_s + \sqrt{l_u^2 - z_e^2}$
- $M_1 = (0,0,z_e)^T$

Šių dviejų sferų susikirtimas su plokštuma $z = z_e$ atsivaizduoja apskritimu, kurio skersmenį r_2 ir centrą M_2 galima rasti vadovaudamasis šiomis formulėmis:

- $r_2 = l_s + \sqrt{l_f^2 - dz_e^2},\ dz_e = z_e - z_w$
- $M_2 = (x_w, y_w, z_e)^T$



Pav. 13 Sfera ir plokštuma z = z_e. Nuotrauka paimta iš [10] šaltinio.

Du taškai $\overrightarrow{e_1}\ ir\ \overrightarrow{e_2}$, kurie yra dviejų pavaizduotų apskritimų susikirtimo taškai, nurodo dvi galimas alkūnės pozicijas. Taigi, galima prieiti iki vektoriaus (x_4, y_4, z_4) (žr. pav. 10) skaičiavimo. (x_4, y_4, z_4) gali būti išreikštas taip:

- $\overrightarrow{z_4} = \frac{\vec{w} - \vec{e_i}}{\|\vec{w} - \vec{e_i}\|}$
- $\overrightarrow{y_4} = \frac{\overrightarrow{z_4} * (\vec{r_s} - \vec{e_i})}{\|\overrightarrow{z_4} * (\vec{r_s} - \vec{e_i})\|}\ , kai\ i = 1, 2.$
- $\overrightarrow{x_4} = \overrightarrow{y_4} * \overrightarrow{z_4}$

Apskaičiavus šių vektorių pozicijas, galime jas atvaizduoti homogenine transformacijos matrica, kuri atvaizduos dvi įmanomas alkūnės pozicijas.

- $T_{alkūnė} = \begin{pmatrix} \overrightarrow{x_4}\overrightarrow{y_4}\overrightarrow{z_4}\overrightarrow{e_i} \\ 0\ 0\ 0\ 1 \end{pmatrix}, kai\ i = 1, 2.$
  - Jog gauti $z_e^{min}\ ir\ z_e^{max}$ mes apsibrėžiame nelygybę $\sqrt{x_w^2 + y_w^2} \leq r_1 + r_2$. Į $r_1\ ir\ r_2$ įstatome $r_1 = l_s + \sqrt{l_u^2 - z_e^2}$ ir $r_2 = l_s + \sqrt{l_f^2 - dz_e^2},\ dz_e = z_e - z_w$

6.6. Atvirkštinės kinematikos sprendimas.

Atvirkštinės kinematikos uždavinio sprendimo užduotis lemia rankos sąnario kampų nustatymą pagal galinio efektoriaus padėtį ir orientaciją. Santykis tarp alkūnės ir atskaitos pagrindo yra jau minėtoje $T_{alkūnė} = \begin{pmatrix} \overrightarrow{x_4}\overrightarrow{y_4}\overrightarrow{z_4}\overrightarrow{e_i} \\ 0\ 0\ 0\ 1 \end{pmatrix}, kai\ i = 1, 2$ lygtyje. Ji išreikšta taip:

$$T_{elbow} = A_1 \cdot A_2 \cdot A_3 \cdot A_4 = \begin{pmatrix} n'_x & o'_x & a'_x & p'_x \\ n'_y & o'_y & a'_y & p'_y \\ n'_z & o'_z & a'_z & p'_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Atkreipiame dėmesį, jog pozicijos ir orientacijos išraiška (x_4, y_4, z_4) priklauso tik nuo pirmųjų keturių sąnario kintamųjų: $\theta_1, \theta_2, \theta_3, \theta_4$. Pozicija i orientacija galinio efektoriaus alkūnės atžvilgiu:

$$T'' = A_4^{-1} \cdot A_3^{-1} \cdot A_2^{-1} \cdot A_1^{-1} \cdot T_{elbow} = A_5 \cdot A_6 \cdot A_7 = \begin{pmatrix} n''_x & o''_x & a''_x & p''_x \\ n''_y & o''_y & a''_y & p''_y \\ n''_z & o''_z & a''_z & p''_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Taigi, atvirkštinę kinematikos problemą galima išskaidyti į prieš tai atvaizduotas dvi mažesnių matmenų antrines problemas. Sprendimas prasideda matricos lygybe:

$$\underbrace{A_1^{-1} \cdot T_{elbow}}_{L_1} = \underbrace{A_2 \cdot A_3 \cdot A_4}_{R_1}$$

Suprastinant žymėjimą, teigsime, jog $s_i ir\ c_i\ yra\ \sin\theta_i ir\ cos\theta_i$. Notacija W(i, j) žymi matricos i eilutės ir j stulpelio elementą. Taigi, visi septyni roboto rankos mazgų kampai apskaičiuojami taip:

- $\theta_{1_{1,2}} = atan2(\pm p'_y, \pm p'_x)$
- $\theta_2 = atan2(-p'_z, c_1 p'_x + s_1 p'_y - l_s)$
- $\theta_3 = atan2(-s_1 o'_x + c_1 o'_y, -s_2 c_1 o'_x - s_2 c_1 o'_x - s_2 s_1 o'_y - c_2 o'_z)$
- $\theta_4 = atan2(c_2 c_1 n'_x + c_2 s_1 n'_y - s_2 n'_z, c_2 c_1 a'_x + c_2 s_1 a'_y - s_2 a'_z$
- $\theta_{5_{1,2}} = atan2(\pm o''_y, \pm o''_x)$
- $\theta_6 = atan2(-o''_z, -c_2 o''_x - s_5 o''_y)$
- $\theta_7 = atan2(-s_5 n''_x, +c_2 n''_y, s_5 a''_x, -c_5 a''_y)$

7. Dirbtiniai neuroniniai tinklai.

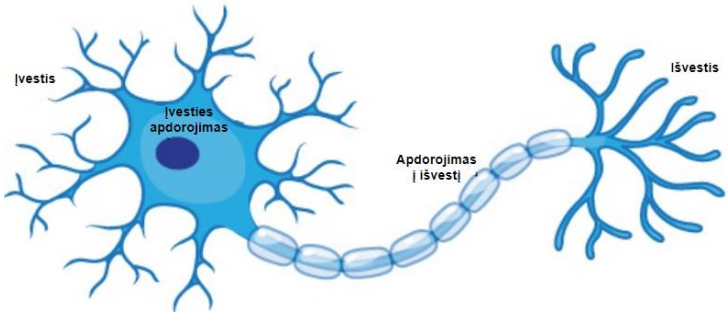
Roboto rankos judėjimui trimatėje plokštumoje yra naudojama keletas metodų. Vieni iš jų yra Dalelių Spiečiaus Optimizacija, Dirbtinė Bičių kolonija, „Firefly“*(FFA)* algoritmas bei hibridiniai metodai. Tačiau vienas populiariausių yra Dirbtinių Neuroninių Tinklų *(ANN)* metodas[DK16].

Dirbtiniai neuroniniai tinklai buvo sukurti semiantis įkvėpimo iš žmogaus, tiksliau jo galvos smegenų bei pačio žmogaus elgsenos. Žmonės pastebėjo, jog kompiuterio galimybė mokytis naudojant realaus gyvenimo pavyzdžius ir patirtis yra geresnis nei naudojantis kitais metodais. Tačiau tai nėra visiškai paprasta. Pats neuroninio tinklo mokymosi procesas turbūt yra didžiausias jo pačio trūkumas. Taip yra dėl sudėtingo algoritmo ir pakankamai didelio resursų poreikio. Tačiau,

jei pats mokymosi procesas yra įgyvendintas atitinkamai, tai neuroninio tinklo atsakymas į užduotą klausimą yra gana paprastas: klaida arba teisingas variantas.

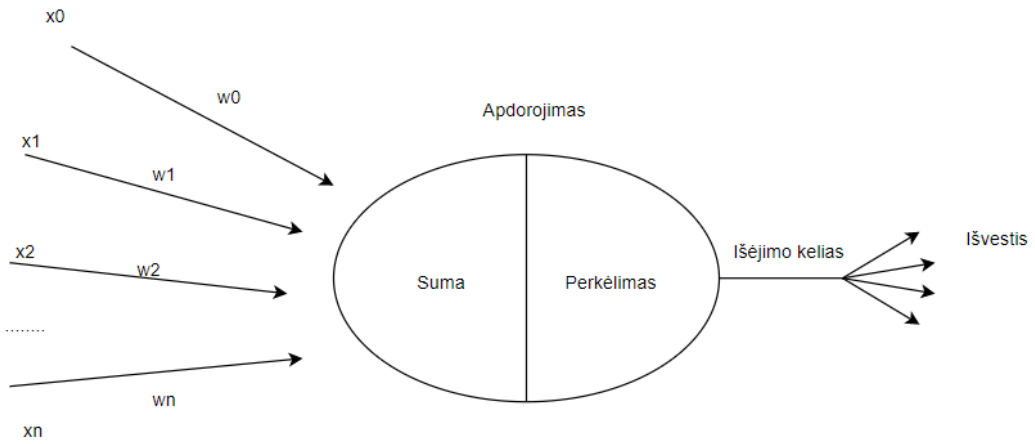
7.1. Dirbtinis neuronas.

Pagrindinis neuroninio tinklo apdorojimo elementas yra neuronas [AM92]. Iš esmės žmogaus galvos biologinis neuronas gauna duomenis iš kitų šaltinių (jutiklių/neuronų), juos tam tikru būdu sujungia/apdoroja ir tuomet išveda galutinį rezultatą, jį atiduodamas kitiems neuronams. Paveikslėlyje numeriu 14 matomas žmogaus neurono pavyzdys [LSW+15].



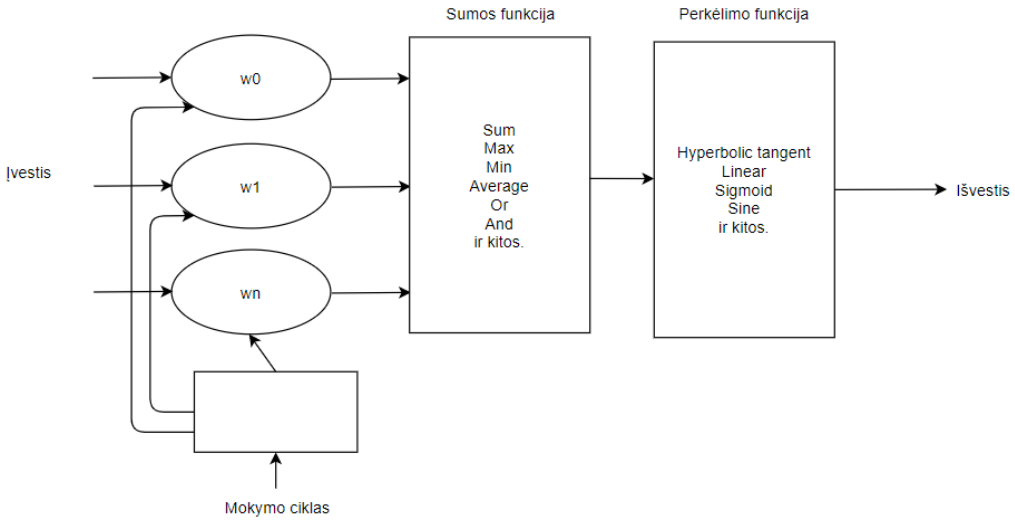
Pav. 14 Žmogaus galvos smegenų neuronas.

Žmogaus neuronas yra kur kas sudėtingesnis organizmas nei prieš tai pavaizduotame paveikslėlyje, tačiau jo nagrinėjimas nėra šio rašto darbo tikslas kaip ir dirbtinių neuroninių tinklų tikslas nėra žmogaus galvos smegenų atkūrimas. Jo tikslas yra siekimas suprasti bei pasiekti tokių pačių galimybių kaip pats biologinis žmogus sąveikauja su pasauliu ir geba mąstyti bei apdoroti informaciją[Suz13]. Norint tai pasiekti, buvo sugalvota, jog pagrindinis dirbtinio neuroninio tinklo vienetas bus dirbtinis neuronas, kuris taip pat turi pagrindines biologinio neurono dalis: įvestį, įvesties apdorojimo bloką, informacijos apdorojimo į išvestį bloką bei išvestį [AM92] (žr. pav. 15).

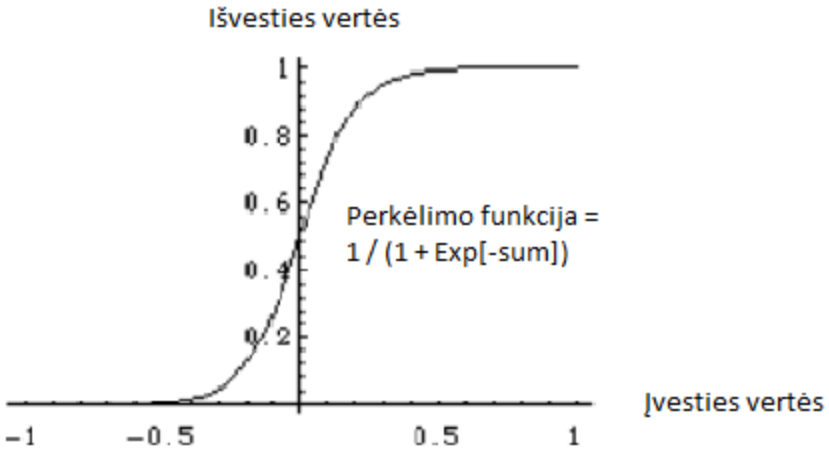


Pav. 15 Paprastas dirbtinis neuronas.

Tačiau pats dirbtinis neuronas (žr. pav. 15) nėra toks paprastas. Paveikslėlyje nr. 16 matome detalesnę dirbtinio neurono schemą. Pirmasis neurono žingsnis yra kiekvieną įvesties elementą sudauginti su įvesties svorių koeficientu, kurie pavaizduoti w0, w1, ..., wn. Tuomet ši sandauga yra paduodama į sumos funkciją. Sumos funkcijoje yra keletas skirtingų operacijų: suma(*sum*), max(*max*), min(*min*), vidurkis(*average*), arba(*or*), ir(*and*) ir kitos. Tuomet sumos funkcijos išvestis yra perkeliama į perkėlimo(*transfer*) funkciją. Perkėlimo funkcijos išvestis yra realus skaičius: 0, 1, -1 arba kitas realus skaičius. Perkėlimo funkcijos palaikomos funkcijos yra sigmoidė, sinusas, hiperbolinis tangentas ir kitos. Pavyzdys, kaip perkėlimo funkcija veikia yra 17 paveikslėlyje.



Pav. 16 „Apdorojamo elemento“ modelis

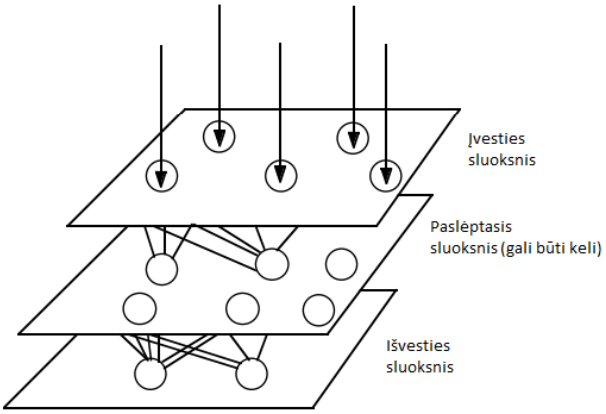


Pav. 17 Sigmoidinė transformavimo funkcija.

7.2. Dirbtinių neuronų junginiai

Neuroninių tinklų sąvokoje ne veltui yra žodis „tinklas“. Šį tinklą sudaro jau prieš tai minėti ir apibūdinti dirbtiniai neuronai. Visi neuroniniai tinklai yra sukonstruoti iš šio, prieš tai aprašyto, pagrindinio, bloko. Biologiškai neuroniniai tinklai yra konstruojami trijų dimensijų pasaulyje (3D) iš mikroskopinių komponentų, neuronų. Tačiau taip nėra žmogaus sukurtame dirbtiniame neuroniniame tinkle. Žmogaus sukurtas tinklas yra dviejų dimensijų (2D) pasaulyje ir yra sudarytas iš riboto kiekio neuronų sluoksnių.

Paprastai, visi dirbtiniai neuroniniai tinklai turi panašią struktūrą (žr. pav. 18). Šioje struktūroje, kai kurie neuronai (pirmasis/įvesties sluoksnis) sąveikauja su išoriniu pasauliu, jog gautų pradžios duomenis. Paskutinis/išvesties sluoksnis, taip pat, sąveikauja su pasauliu, jog grąžintų apdorotą informaciją. Visi kiti neuronai yra „paslėptuose“ sluoksniuose.

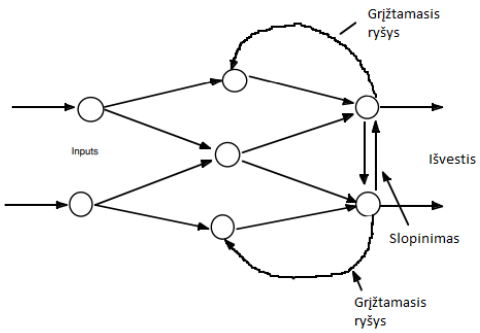


Pav. 18 Paprasta neuroninio tinklo sluoksnių diagrama.

Daugelyje dirbtinių tinklų kiekvienas neuronas iš paslėptojo sluoksnio gauna signalą iš prieš tai jau esančio neurono. Po to, kai neuronas apskaičiuoja savo funkciją, jis atiduoda rezultatą visiems neuronams, kurie priklauso sluoksniui, žemiau jo. Šios susisiekimo tarp neuronų linijos yra svarbus neuroninių tinklų aspektas. Šių jungčių yra du tipai. Vienas tipas sumuoja, kitas atima.

Kai kurie tinklai siekia, jog neuronai slopintų kitus neuronus tame pačiame sluoksnyje. Tai yra vadinama šoniniu slopinimu (*lateral inhibition*). Dažniausiai tai yra naudojama išvesties sluoksniuose. Pavyzdžiui, dirbtiniame neuroniniame tinkle, kuris „skaito“ tekstą ir jį atpažįsta yra dažnas variantas, jog vienu metu vieną raidę gali įvardinti keliomis raidėmis. Jei gaunama situacija yra, jog P raidę tinklas įvardija, kaip 85 % kaip P raidę, o 65 % kaip S raidę, tinklas turi nuslopinti neuroną, kurio išvestis bus S raidė, nes P raidė surinko didesnį procentų kiekį.

Kitas neuronų sujungimo tipas yra „grįžtamasis ryšys“. Tai yra kai išvestis vieno sluoksnio grįžta į prieš tai esanti sluoksnį ir tampa jo įvestis. Paveikslėlyje nr. 19 matome grįžtamojo ryšio pavyzdį.



Pav. 19 Paprastas tinklas su atsiliepimais ir konkurencija.

7.3. Dirbtinių neuroninių tinklų mokymas.

Neuroninio tinklo architektūra yra tokia, jog davus tinkamą įvesties duomenų rinkinį, dirbtinis neuroninis tinklas pateikia norimą išvestį [SK13]. Egzistuoja įvairūs metodai sąsajų stiprybėms nustatyti. Vienas būdas yra aiškiai nustatyti svorius, o kitas yra „maitinti“ dirbtinio neuroninio tinklo mokymosi modelį ir leisti jam keisti jo svorius pagal pasirinktą mokymosi taisyklę.

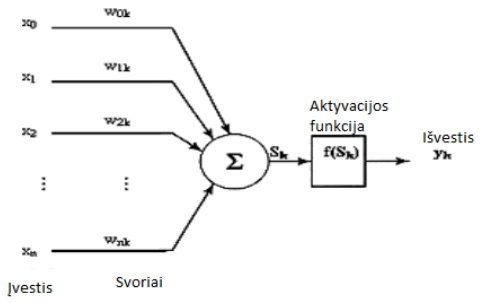
7.4. Daugiasluoksnis perceptronas.

Daugiasluoksnį perceptroną sudaro trys dalys: įvesties, paslėptieji ir išvesties sluoksniai (žr. pav. 20). Kiekvienas sluoksnis naudoja aktyvavimo funkciją, kuri sujungia juos visus. Daugiasluoksniai tinklai yra pagrįsti svorių reguliavimais visuose skirtinguose sluoksniuose. Svoriai lemia tikslią norimą išvesties funkciją.

Kaip jau buvo minėta, biologinis bei dirbtinis neuronas turi tuos pačius pagrindinius komponentus. Roboto manipuliatoriaus atveju, neuronas susideda iš šių komponentų:

- Nuorodų, apibūdinančių neuronų įvestis, skaičius su svoriais W1, W2,..., Wm.
- Loginę sudėties operaciją, kuri sudeda įvestis, kurios turi svorius.
- $u \sum_{j=1}^{m} W_j X_j$

Sigmoidinė aktyvacijos funkcija φ, kuri riboja neurono išvestį: $\varphi = \phi(u + b)$



Pav. 20 Daugiasluoksnis perceptronas.

- Visos įvesties sumos pridedamos prie jų svorių ir praeina per kiekvieną sluoksnį link išvesties.
- Įvertinus išvestį yra įvertinamos visų išvesčių klaidos. Tuomet šios klaidos yra pašalinamos koreguojant svorius paslėptuose sluoksniuose.

8. Dirbtiniai neuroniniai tinklai 6 mazgų roboto rankoje.

8.1. Šešių laisvės mazgų „Denso“ robotas.

Šiame skyriuje bus nagrinėjama 6 laisvės mazgų „Denso“ roboto judėjimas naudojantis neuroniniais tinklais, kurį atliko Ahmed R. J. Almusawi, L. Canan Dülger ir Sadettin Kapucu.

8.1.1. Roboto specifikacija.

Šio roboto homogeninė transformacijos matrica yra $A_i = Rot_{\mathrm{z},\theta i}, Trans_{\mathrm{x,d}i} Trans_{x,\alpha i} Rot_{x,\alpha i}$.

$$A_i = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

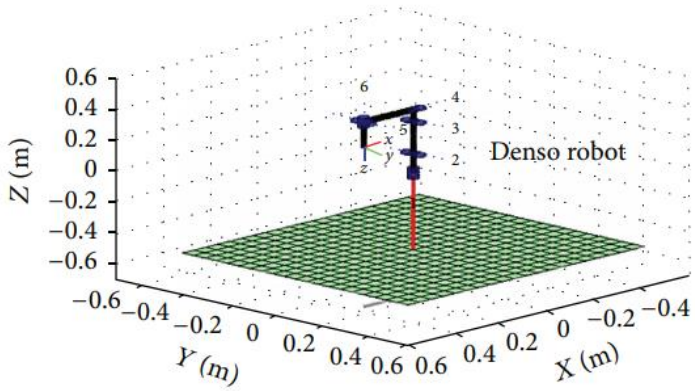
kur i yra mazgo numeris, $S\theta i$ = sin θ_i, $C\theta_i$ = cos θ_i, θ_i yra mazgo rotacijos kampas, a_i yra rankos dalių ilgis, α_i yra posūkių kampai, θ yra mazgų kampai, d_i yra nuorodos kompensacijos.

Denavit – Hertenberg roboto rankos parametrai yra atvaizduoti lentelėje nr. 2 [ADK16].

Link i	θ_i	d_i	a_i	α_i
1	q_1	0.125	0	pi/2
2	q_2	0	0.21	0
3	q_3	0	−0.075	−pi/2
4	q_4	0.21	0	pi/2
5	q_5	0	0	−pi/2
6	q_6	0.07	0	0

Lent. 2 Denavit – Hertenberg roboto rankos parametrai.

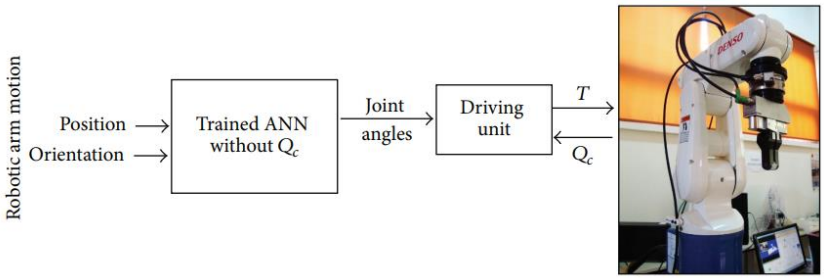
Grafinis roboto sistemos atvaizdavimas, atliktas su MATLAB programa:



Pav. 21 Denso robotas.

8.1.2. Tradiciniai neuroniniai tinklai.

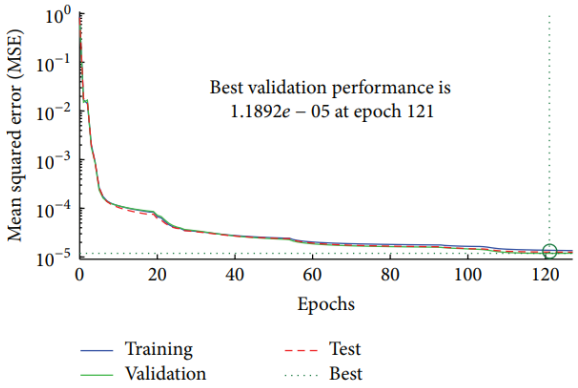
Tradicinė architektūros dirbtiniai neuroniniai tinklai yra naudojami labai plačiai. Šiuo atveju, tradicinio tinklo įvestis bus šeši kintamieji, kurie atspindi pozicijas ir orientacijas „grėblio“. Paslėptųjų sluoksnių skaičius yra 10. Išvestis yra, taip pat, šeši elementai, kurie nurodo mazgo Q kampus. MATLAB programinė įranga buvo naudota mokymui, testavimui ir patvirtinimui. Pav. nr. 22 vaizduoja tradicinio neuroninio tinklo diagramą bei modelį [Q] = ANN_Traditional_Net(P,R).



Pav. 22 Tradicinio neuroninio tinklo diagrama bei modelis [Q] = ANN_Traditional_Net(P,R). Paveikslėlis paimtas iš [ADK16] šaltinio.

Įvestis/išvestis yra apskaičiuojama „forward“ kinematikos problema. Tokiu būdu kiekviena roboto pozicija turi savo atskirą mangų konfiguraciją, neuroninio tinklo įvesties/išvesties atžvilgiu. Mokymo algoritmas – *Levenberg-Marquardt back-propogation.* Jis yra naudojamas norint užtikrinti greitą mokymo klaidų suartėjimą ir yra labai populiarus kreivės pritaikymo algoritmas.

Mokymo MSE krito iki kol validacijos klaida sustabdė epochas 121-ojoje, kur MSE buvo $1.1892e{-}5$.



Pav. 23 MSE diagrama. Paveikslėlis paimtas iš [ADK16] šaltinio.

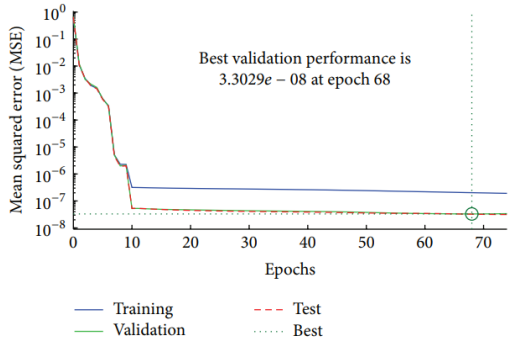
8.1.3. Siūlomas dirbtinio neuroninio tinklo dizainas.

Šiame skyriuje autoriai siūlo naudoti kitokius dirbtinius neuroninius tinklus. Siūlomas metodas remiasi roboto rankos kinematikos apribojimais. Taip pat, priekinė kinematika buvo naudojama sugeneruoti įvesties duomenis, kurie buvo panaudoti mokymui. Neuroninio tinklo įvestis yra pozicija/orientacija ir dabartinė roboto laisvės mazgų konfiguracija/duomenys. Siūlomas

neuroninis tinklas, įvesties sluoksnyje, gali priimti 13 įvesties elementų, kur P ir R yra „grėblio“ pozicija/orientacija ir laisvės mazgų kampai Q_c. Išvesties sluoksnis turi šešis laisvės mazgų kampus Q.

$$
\begin{aligned}
[Q] &= \text{ANN\_Proposed\_Net}\left(P, R, Q_c\right), \\
Q &= \left[q_1, q_2, q_3, q_4, q_5, q_6\right], \\
P &= \left[x, y, z\right], \\
R &= \left[R_x, R_y, R_z\right], \\
Q_c &= \left[q_{1c}, q_{2c}, q_{3c}, q_{4c}, q_{5c}, q_{6c}\right].
\end{aligned}
$$

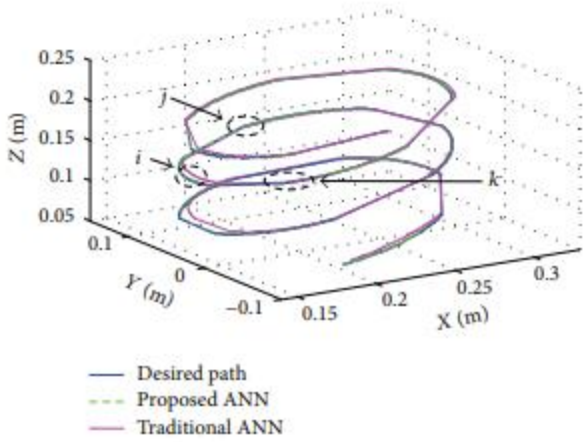
Tikslumas neuroninio tinklo buvo skaičiuojamas pagal MSE tarp neuroninio tinklo išvesties ir to, kas buvo tikėtasi. Tikslumas pavaizduotas paveikslėlyje 24. Kaip matome, MSE krenta su epochų skaičiumi. Mokymo MSE kritimas buvo užfiksuotas iki 68 – os epochos, kai MSE buvo $3.3029e{-}8$. Mokymo duomenų buvo 2800, testavimo – 600.



Pav. 24 MSE diagrama. Paveikslėlis paimtas iš [ADK16] šaltinio.

8.1.4. Nagrinėjamas uždavinys.

Nagrinėjamas uždavinys yra roboto rankos galinio efektoriaus sekimas linija trimatėje erdvėje. Linijos išdėstymas koordinačių plokštumoje yra spiralės formos. X ašyje plotis užima 0.15 m, – 0.3 m., Y ašyje nuo -0.1 m. iki 0.1 m., Z – 0.05 m. – 0.25 m. Žemiau yra pateiktas paveikslėlis, kuris nurodo tikslo trajektoriją ir galinio efektoriaus keliavimo kelią remiantis siūlomais dirbtiniais neuroniniais tinklais ir tradiciniais.



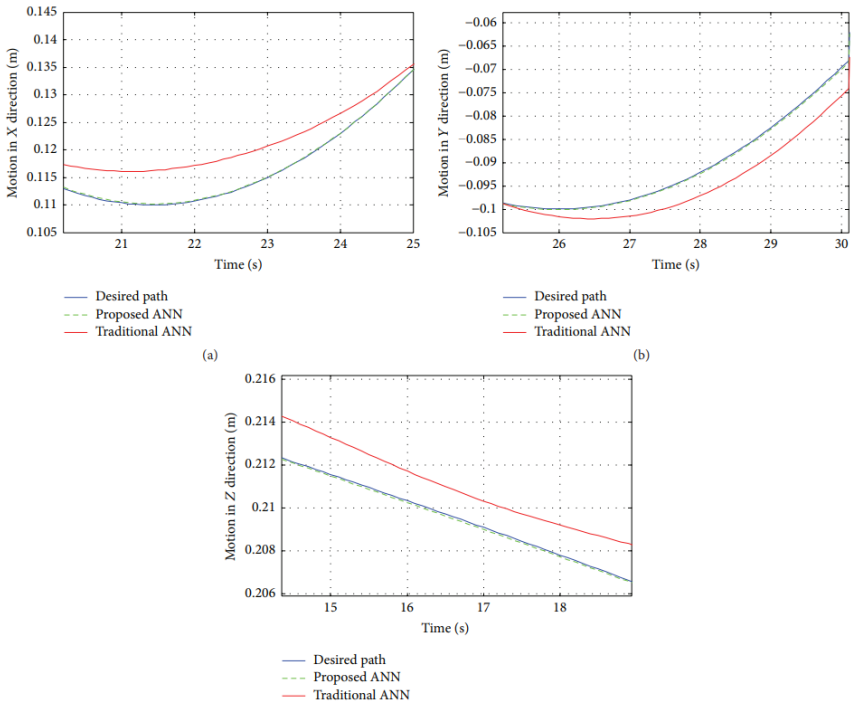
Pav. 25. Tradicinių neuroninių tinklų ir siūlomų roboto rankos galinio efektoriaus palyginimas. Paveikslėlis paimtas iš [ADK16] šaltinio.

8.1.5. Rezultatai.

Žemiau pateiktoje lentelėje (žr. pav. nr. 25) pateiktas dviejų, šiame skyriuje nagrinėtų, neuroninių tinklų tikslumas. Matome, jog siūlomas ANN tinklas yra daug tikslesnis nei tradicinis. Taip pat, žemiau yra pateikta grafinių pavyzdžių, kur yra lyginami x, y, z judėjimo krypčių tikslumas (žr. pav. 26). Siūlomas ANN dizainas parodė geresnį galinio efektoriaus tikslumą nei tradicinis neuroninis tinklas.

Parameters	Proposed ANN	Traditional ANN
P_x error%	0.17	5.78
P_y error%	0.36	7.25
P_z error%	0.12	1.28
MSE	$3.3029e^{-8}$	$1.1892e^{-5}$
Regression	0.99999	0.99758

Pav. 24 Rezultatai. Paveikslėlis paimtas iš [ADK16] šaltinio.



Pav. 25 Rezultatai. Paveikslėlis paimtas iš [ADK16] šaltinio.

8.2. Neuroninių tinklų, šešių mazgų roboto rankos judėjimas su kompensacija.

Dirbtinis neuroninis tinklas naudodamas atvirkštinio sprendimo (back propogation) algoritmą geba spręsti išvirkštinės kinematikos problemas. Šiame skyriuje bus taip pat nagrinėjamas 6 laisvės mazgų robotas. Du Turkijos dėstytojai: Z. Bingul, H.M. Ertunc and C. Oysu, atliko šį tyrimą [BEO05].

8.2.1. Dirbtinio neuroninio tinklo atvirkštinio sprendimo (backpropogation) struktūra.

Šio neuroninio tinklo įvesties duomenys yra imami iš trijų atskirų šaltinių, kurių pavadinimai buvo:

- pirmas pristatymas – 12 parametrų,

- antras pristatymas – 6 parametrai,
- trečias pristatymas – 7 parametrai.

Neuroninis tinklas buvo mokytas tris kartus. Kiekvienas įvesties duomenų šaltinis buvo parinktas vienam mokymo ciklui. Kitaip tariant, vienas šaltinis – vienam ciklui. Pirmame pristatyme dvylika parametrų reprezentuoja devynis elementus rotacijos matricos $(r_{ij} i, j = 1{:}\,3)$ ir trys vektorių pozicijas (P_x, P_y, P_z). Antrame pristatyme pirmosios trys įvestys yra Eulerio kampai, apskaičiuojami taip: $R(\alpha, \beta, \gamma) = \begin{bmatrix} c\alpha c\beta & c\alpha s\beta s\gamma - s\alpha c\gamma & c\alpha s\beta c\gamma + s\alpha s\gamma \\ s\alpha c\beta & s\alpha s\beta s\gamma + c\alpha c\gamma & s\alpha s\beta c\gamma - c\alpha s\gamma \\ -s\beta & c\beta s\gamma & c\beta c\gamma \end{bmatrix}$.

Paskutiniame pristatyme pirmieji keturi parametrai (vienas vektorius ir vienas kampas) yra apskaičiuojami taip: $R(K, \theta) = \begin{bmatrix} k_x k_x v\theta c\theta & k_x k_y v\theta - k_z s\theta & k_x k_z v\theta + k_y s\theta \\ k_x k_y v\theta + k_z s\theta & k_y k_y v\theta + c\theta & k_y k_z v\theta - k_x s\theta \\ k_x k_z v\theta - k_y s\theta & k_y k_z v\theta + k_x s\theta & k_z k_{zv\theta + c\theta} \end{bmatrix}$,

kur $K = k_x i + k_y j + k_z k$ ir $v\theta = (1 - c\theta)$. Denavit Hartenberg roboto rankos kinematiniai parametrai yra išdėstyti žemiau esančioje lentelėje (žr. pav. 28).

i	θ_i	α_{i-1}	a_{i-1}	d_i
1	θ_1	0	0	h_1
2	θ_2	90	0	d_2
3	θ_3	-90	0	l_2
4	θ_4	0	l_3	0
5	θ_5	-90	l_4	0
6	θ_6	90	0	d_6

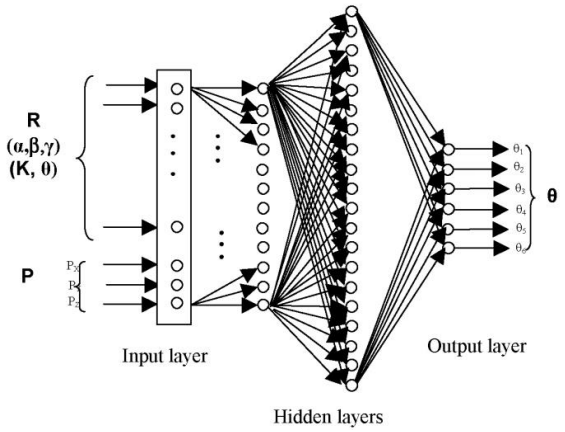
Pav. 28 laisvės mazgų rėžiai.

Šis, nagrinėjamas, neuroninis tinklas turi šešias išvestis visiems trims pristatymams(*representation*). Išvestis yra šeši roboto rankos laisvės mazgų kampai.

Simuliaciniai duomenys dirbtiniam neuroniniam tinklui mokyti ir testuoti yra generuojami iš prieš tai aprašytų lygčių. Duomenų rinkiniai turi $1.5x10^6$ įvesties-išvesties porų kiekvienam 10^o kampui. Darbinis duomenų rinkinys sudaro 8000 duomenų taškus, kurie buvo sudaryti iš įvesties ir išvesties vektorių porų. Išvesties vektorius, arba dar vadinamas tiksliniu vektoriumi, apima šešis

jungiamuosius kampus. 70 % duomenų buvo skirta mokymui ir 30 % - testavimui. Tiek įvesties, tiek išvesties kintamieji buvo normalizuoti [-1, 1] intervale.

Atvirkštinio sprendimo (backpropogation) neuroninio tinklo paslėptieji sluoksniai yra sudaryti tik iš dviejų sluoksnių, kur pirmasis turi 12 neuronų, o antrasis – 24. Taip pat, išvesties sluoksnis turi neuronus (žr. pav. 29).



Pav. 29 Atvirkštinio sprendimo (backpropogation) neuroninio tinklo paslėptieji sluoksniai. Paveikslėlis iš [BEO05] šaltinio.

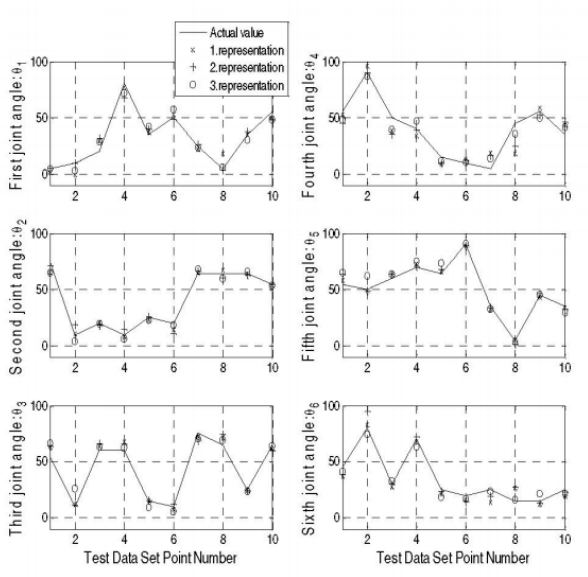
Įvesties ir paslėptųjų sluoksnių neuronai turi sigmoidinę aktyvacijos funkciją. Išvesties neuronai yra linijiniai, dėl to jie gali pateiktį reikšmes ne tik [-1; 1] rėžiuose. Įvesties ir tikėtini/spėjami mokymo duomenų vektoriai yra naudojami apmokyti tinklą iki tol kol jis pradeda atrasti sąsajas tarp įvesties ir išvesties. Tinklo mokymui buvo koreguoti svorio koeficientai naudodamasis Levenberg-Marquardt algoritmu, kuris yra greičiausias mokumo algoritmas vidutinio dydžio tinklams [Mor77]:

- išjungimo kriterijus klaidos tikslui yra nustatytas 0.01,
- visi svorio koeficientai buvo priskirti atsitiktinai.

8.2.2. Rezultatai.

Pagrindinis šio neuroninio tinklo naudojimo trūkumas yra, jog reikia labai didelio mokymo duomenų kiekio, jog būtų pasiektas patenkinamas tikslumas. Po 70 % pasirinktų įvesties-išvesties vektorių apmokymo, kuris buvo pasirinktas atsitiktinai iš 8000 porų, tinklas buvo pratestuotas ant

likusių 30 %, kurie nebuvo naudojami mokymo duomenų aibėje. Tikimasi, jog treniruotas (backpropogation) tinklas pagrįsta reaguos į pateiktus duomenis. Žemiau pateiktame paveikslėlyje (žr. pav. nr. 30) vaizduojamas kiekvieno laisvės mazgo kampas po kiekvieno iš trijų pristatymų. Taip pat, nupiešta ir tikslo linija/reikšmės.



Pav. 30 Rezultatai. Paveikslėlis iš [BEO05] šaltinio.

9. Simuliacinės aplinkos.

9.1. OpenAI Gym

OpenAi Gym [BCP+16] yra mokymosi tyrimų rinkinys, kuris su *reinforcement learning* padeda apmokyti tam tikrus robotus su dideliu duomenų kiekiu. OpenAI Gym yra geras tuo, jog su juo galima paduoti labai didelius duomenų kiekius ir prasukti didelį ciklų skaičių. Kai kuriems žmonėms, kurie stengėsi su kitomis simuliacinėmis aplinkomis prasukti didelį kiekį duomenų su kitomis simuliacinėmis aplinkomis, nepasisekdavo, nes susimuliuoti tai užtrukdavo ilgą laika. Rašant šį magistrinį darbą buvo padaryta prielaida, jog OpenAI Gym leis išspręsti tokią problemą, teoriškai. Kaip bus praktiškai, bus aprašyta praktinėje dalyje.

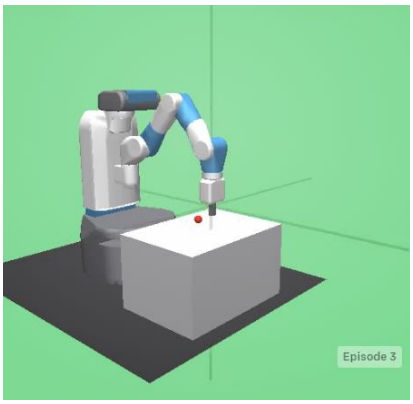
8.1.1 OpenAI Gym aplinkos.

OpenAI Gym, kol kas, turi keletą aplinkų iš kurių ir bus viena naudojama šiame magistriniame darbe – robotikos aplinka. Internetinė nuoroda: https://gym.openai.com/envs/FetchReach-v1/

Aplinkų sąrašas:

- Algoritminė
- Atari
- Box2d
- Klasikinė
- Mujoco
- Robotikos
- Žaislų/Teksto

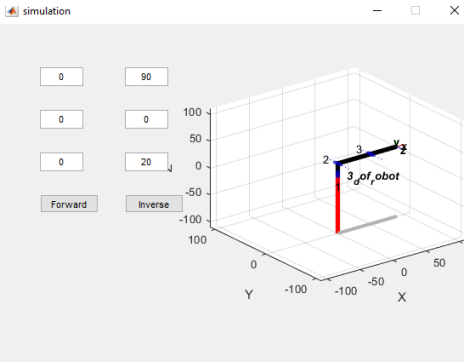
Kaip jau buvo minėta, bus naudojama robotikos simulicinė aplinka. Kaip ji atrodo, galima išvysti paveikslėlyje nr. 31.



Pav. 31 Roboto rankos simuliacija.

9.2. MATLAB simuliacinė aplinka.

Matlab yra daugiaplatforminė prograinė įranga, skirta įvairių mokslo šakų problemoms spręsti. Šiame magistriniame darbe, ji yra pasirinkta viena iš variantų simuliuoti roboto rankos judėjimą. Kolkas buvo padaryti pora bandymų ir susimuliuota 3 laisvės mazgų roboto ranka išvirkštinės kinematikos problemai spręsti(žr. pav. 32.) [YEB+16].



Pav. 32 Roboto rankos simuliacija.

9.2.1. MATLAB simuliacinės aplinkos kodas.

Susimuliuoti roboto rankai reikėjo „rvctools“ bibliotekos. Žemiau matomas kodas, kuris skaičiuoja išvirkštinės kinematikos problemą bei atvaizduoja ją roboto simuliacijoje (žr. pav. 33).

```
function btn_Invere_Callback(hObject, eventdata, handles)
% hObject    handle to btn_Invere (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
PX = str2double(handles.Pos_X.String);
PY = str2double(handles.Pos_Y.String);
PZ = str2double(handles.Pos_Z.String);

L_1 = 20;
L_2 = 50;
L_3 = 40;

L(1) = Link([0 L_1 0 pi/2]);
L(2) = Link([0 0 L_2 0]);
L(3) = Link([0 0 L_3 0]);

Robot = SerialLink(L);
Robot.name = '3_dof_robot';

T = [1 0 0 PX;
     0 1 0 PY;
     0 0 1 PZ;
     0 0 0 1];
J = Robot.ikine(T, [0 0 0], [1 1 1 0 0 0]) * 180/pi;
handles.Theta_1.String = num2str(floor(J(1)));
handles.Theta_2.String = num2str(floor(J(2)));
handles.Theta_3.String = num2str(floor(J(3)));

Robot.plot(J*pi/180);
```

Pav. 33 Roboto rankos simuliacijos kodas MATLAB aplinkoje.

Ateities darbuose Matlab simuliacinėje aplinkoje bus įgyvendintas 8 mazgų roboto ranka, nupieštas kelias ir pateikus įvesties duomenis, bus stebima, ar roboto galinio efektorius keliaujo pieštu keliu. Taip bus atskirta, ar testuojamas algoritmas yra efektyvus ar ne.

10. Sustiprinimo mokymasis (reinforcement learning).

Reinforcement learning (RL) yra mašininio mokymosi sritis. RL naudoja įvairios programinės įrangos ir mechanizuotos sistemos, kurių tikslas yra rasti geriausią įmanomą elgesį ar kelią, kurį reiktų pasirinkti, jog robotas pasiektų tikslą.[Baj20]. RL mokymasis skiriasi nuo prižiūrimo (supervised) mokymosi tuo, jog prižiūrimo mokymosi duomenyse yra duota, koks yra teisingas rezultatas. Tad modelis iškart treniruojasi iš teisingų duomenų. RL mokymosi metu to nėra. Jis pats sprendžia, kokius sprendimus priimti, ir iš to mokosi. Kitaip tariant, RL mokosi iš savo patirties.

10.1.1. Pagrindiniai RL aspektai.

- Įvestis turėtų būti pradinė būsena, nuo kurios prasidės modelis.
- Išvestis. Yra daug galimų išvesties variantų, nes tam tikros problemos sprendimo būdai yra įvairūs.
- Mokymas. Mokymas grindžiamas įvestimi. Modelis grąžins būseną, o vartotojas nuspręs apdovanoti modelį arba nubausti už jo rezultatą.
- Modelis nenustoja mokytis.
- Geriausias sprendimas nusprendžiamas atsižvelgiant į maksimalų atlygį.

10.1.2. RL ir (supervised) mokymo skirtumai bei RL tipai.

RL	(Supervised)
Mokymasis susijęs tiesiogiai su modulio sprendimų priėmimu.	Mokymosi sprendimas priimamas tik nuo pradinės įvesties duomenų.
Visi padaryti sprendimas yra susiję vieni su kitais.	Visi sprendimai yra skirtingi.
Pavyzdys: šachmatų žaidimas.	Pavyzdys: nuotraukos atpažinimas.

Yra du RL tipai:

- Teigiamas – vadinamas įvykiu, kai tam tikras teigiamas įvykis atsiranda dėl tam tikro elgesio.

  RL mokymosi pranašumai:

  - Maksimaliai padidina našumą.
  - Palaiko pokyčius ilgą periodą.

  RL trūkumai:

  - Per didelis sugeneruotų duomenų kiekis gali susilpninti rezultatus.

- Neigiamas – yra apibrėžiamas kaip elgesio stiprinimas, nes neigiama būklė sustabdoma arba stengiamasi jos išvengti.

  RL mokymosi pranašumai:

  - Padidina elgesį.
  - Nepaiso minimalaus našumo standarto.

  RL trūkumai:

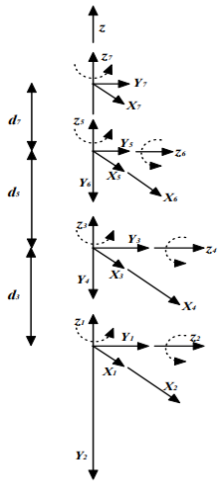
  - Teikia tik tiek, jog patenkintų minimalius reikalavimus.

11. Optimizacija.

11.1. PA-10 manipuliatoriaus optimizacijos uždavinys.

PA-10 manipuliatorius tai yra mobili(nešiojama), intelektuali bendros paskirties ranka [KD05]. Ši ranka buvo sukurta Mitsubishi ir turi 7 laisvės mazgus (žr. pav. 34). PA-10 rankos specifikacijos:

- 35 kg. svoris.
- Maksimalus visų ašių bendrasis greitis yra 155 m/s.
- Keliamoji galia 10 kg.
- Sukimo momentas 9.9 nm.



Pav. 34 Roboto rankos atvaizdavimas koordinačių ašimis.

Vienalytę transformacijos matricą šios T_{07} manipuliatoriaus rankos, kuri atvaizduoja finalinę galinio efektoriaus poziciją ir orientaciją, bazinės koordinačių sistemos atžvilgiu. Ją galima gauti iš grandinėje iš eilės einančių koordinačių transformacijos matricų. Finalinė rankos matrica gali būti išreikšta taip: $T_{07} = \prod_{i=1}^{7} T_{i-1,i} = \begin{pmatrix} nsap \\ 0001 \end{pmatrix}$, kur n yra normalus galinio efektoriaus vektorius bazinėje koordinačių sistemoje, s yra slenkamasis galinio efektoriaus vektorius, a yra požiūrio vektorius galinio efektoriaus atžvilgiu, $p = [x, y, z]^T$ yra galinio efektoriaus padėties vektorius. $T_{i-1,i}$, kai i = 1, 2, ..., 7 yra pateikti žemiau, kur $c_i = \cos q_i$ ir $s_i = \sin q_i$. Rankos matrica yra gaunama padauginus žemiau esančias matricas.

$$T_{01} = \begin{pmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} T_{12} = \begin{pmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_2 & -c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{23} = \begin{pmatrix} c_3 & -s_3 & 0 & 0 \\ 0 & 0 & -1 & -d_3 \\ s3 & c3 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} T_{34} = \begin{pmatrix} c_4 & -s_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_4 & -c_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{45} = \begin{pmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & -1 & -d_5 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} T_{56} = \begin{pmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_{67} = \begin{pmatrix} c_7 & -s_7 & 0 & 0 \\ 0 & 0 & -1 & -d_7 \\ s_7 & c_7 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Pav. 35

11.2. PA-10 siūlomas optimizavimo algoritmas.

Norint sužinoti q(t) duotam $r_d(t)$ mums reikia išspręsti kintamą netiesinio optimizavimo problemą su lygybės apribojimais: $minimize\ \frac{1}{2}q(t)^T W q(t)$ kuriems taikoma $J\big(q(t)\big)q(t) = r_d(t)$, kur $q(t)^T$ žymi perkėlimą $q(t)$ ir W yra simetrinis svoris M x M matricos, kuris gali būti teigiamas arba ne. Visų pirma, jei W yra tapatybės matrica, tai tikslinė funkcija, kuria reikia sumažinti yra lygi 2-osios normos jungčiai $\|q(t)\|_2^2$. Jei W yra iteracijos matrica, tada tikslo funkcijos sumažinimas lygus kinetinei energijai.

Energijos funkcija netiesinio optimizavimo programavimo problemai, kuriai taikomi lygybės apribojimai, aprašyti aukščiau, atrodo taip:

$$E(\dot{q}(t),\ \lambda(t)) = \frac{1}{2}\ \dot{q}(t)^T\ W\ \dot{q}(t) + \lambda(t)^T[J(q(t))\dot{q}(t) - \dot{r}_d(t)] + \frac{K}{2}[J(q(t))\dot{q}(t) - \dot{r}_d(t)]^T[J(q(t))\dot{q}(t) - \dot{r}_d(t)]$$

Pav. 36 Energijos funkcija netiesinio optimizavimo programavimo problemai spręsti.

kur λ(t) yra n - dimencinio stulpelių vektorius, priklausantis R^{n*1}, K yra bausmės parametras, kuris visada yra didesnis arba lygus nuliui. Taikant nuolydžio metodą, tinklo atnaujinimo lygtį galime atlikti taip:

$$\dot{q}(k+1) = \dot{q}(k) - \mu \nabla_{\dot{q}} E(\dot{q}, \lambda)$$
$$\lambda(k+1) = \lambda(k) + \eta \nabla_{\lambda} E(\dot{q}, \lambda)$$

kur μ ir η yra mokymosi greičio parametrai. Nustačius nuolydį, prieš tai aprašytose dvejose lygtyse, gaunamos mokymosi taisyklės:

$$\dot{q}(k+1) = \dot{q}(k) - \mu[W\dot{q}(k) + J(q(t))^T \lambda(k) + KJ(q(t))^T (J(q(t)))\dot{q}(t) - \dot{r}_d(t)]$$

$$\lambda(k+1) = \lambda(k) + \eta[J(q(t))\dot{q}(t) - \dot{r}_d(t)]$$

Šios dvi lygtys padeda gauti laisvės mazgų greičio reikšmes.

12. Praktinis aštuonių laisvės mazgų roboto rankos įgyvendinimas.

Šiame skyriuje bus aprašytas praktinis aštuonių laisvės mazgų roboto rankos įgyvendinimas bei apmokymas pasiekti galutinį tikslą. Taip pat, bus aptarta simuliacinė aplinka, mokymosi algoritmas, rankos mokymosi eiga bei rezultatai.

Įgyvendintą kodą galima rasti nuorodoje https://github.com/mykolas1/RobotArm.

12.1. Simuliacinė aplinka.

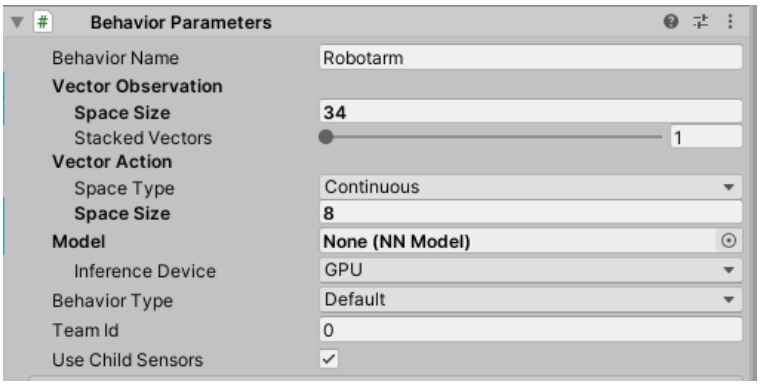
Prieš tai buvo išbandytos jau aprašytos OpenAI bei Matlab aplinkos. Ties šiomis dviem aplinkomis paieška nebuvo sustabdyta. Buvo norėta surasti naudotojui (programuotojui) draugiškesnę aplinką bei aplinka, kurioje jau turima patirties. Po keleto bandymų, simuliacinei aplinkai buvo pasirinkta Unity žaidimų kūrimo platforma bei variklis su Unity ML integruotu įrankiu. Ši platforma yra sukurta kaip variklis kurti kompiuterinius žaidimus, tačiau jos sukurta aplinka gali būti puikiai pritaikoma simuliuoti įvairius scenarijus. Su šia platforma galima kurti 2 dimencijų bei 3 dimencijų žaidimus bei aplikacijas. Nuoroda į Unity naudotojo dokumentaciją - https://docs.unity3d.com/Manual/index.html.

12.1.1. Unity ML Įrankis.

Unity Machine Learning (ML) Toolkit yra atviro kodo mašininio mokymosi įrankis, kurį galima pridėti prie bet kurio *Unity* egzistuojančio projekto [Met20], leidžiantis kurti intelektualių agentų mokymosi aplinką. Šis *Unity* platformos priedas buvo sukurtas kaip būdas skatinti kurti naujas sudėtingas aplinkas bei agentus, kurie galėtų egzistuoti tose aplinkose.

Agentu gali būti, bet kuris žaidimo ar simuliacinės aplinkos objektas, kuris turi šias savybes bei scenarijus:

- *Unity* scenarijų, kuris apibrėžia žaidimo objektą (*Game Object*) kaip *ML – Agents Toolkit* agentą.
- Elgsenos parametrus, kitaip vadinamas agento smegenimis (žr. pav. 36).
- Komunikacijos scenariju, kuris susieja *Academy* su agentu.



Pav. 36

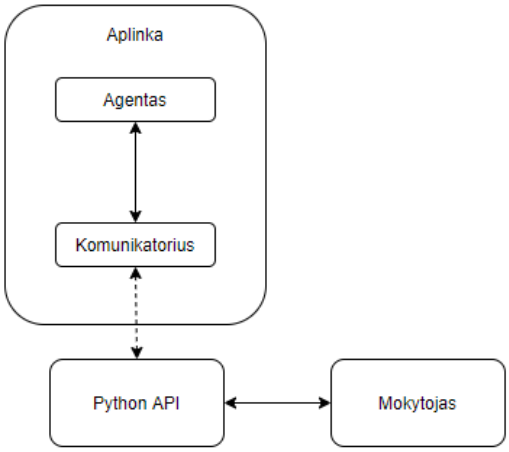
Academy yra klasė struktūrizuota su *ML – Agent Toolkit*. Ji yra atsakinga už keletą veiksnių:

- Agentų inicijavimu.
- Žingsnių skaičiaus sekimu.
- Epizodų sekimu nuo mokymosi pradžios.

ML – Agent Toolkit turi keturias pagrindines funkcijas/savybes:

- Kalbant apie mokymosi aplinką, įrankis leidžia agentui daryti stebėjimus ir spręsti, kokius veiksmus daryti. Už tai gauti atlygį.
- *Python API*. Išorinis komponentas sujungti „mokytojus“ su mokomu komponentu.

- Komunikatorius. Jo tikslas suteikti komunikacijos galimybes tarp agento smegenų bei *Python API*.
- *Python Trainer*. Išorinis funkcionalumas, kuris leidžia apibrėžti mokymosi algoritmus. Finalinis produktas, pateiktas „mokytojo“ yra neuroninio tinklo modelis, kuris kontroliuoja agento veiksmus.



Pav. 37, ML – Agent Toolkit architektūros pavyzdys.

12.1.2. Agento smegenys.

Agento smegenys yra jo *Behavior Parameters script‘as* [žr. pav. 36] prisegtas prie žaidimo objekto. Smegenų tikslas yra komunikuoti tarp agento ir mokymosi API bei pranešti agentui kurį veiksmą jam atlikti.

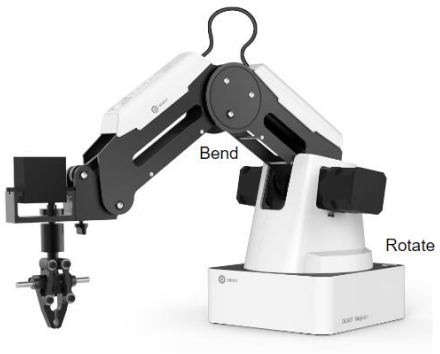
Smegenų sudėtis:

- Vardas: suteiktas trenerio naudojamų *hiper* parametrų rinkimui.
- Vektorių stebėjimas: kintamųjų kiekis, kurį agentas pasiekti iš aplinkos.
- Vektorių veiksmai: realių arba sveikų skaičių eilė, atsižvelgiant ar vektoriaus tipas yra tęstinis ar diskretus.
- Modelis: nusako kurį NN modelį smegenys naudoja agento veiksmams atlikti.
- Elgsenos tipas: smegenys nuspręs kurį veiksmą atlikti, atsižvelgiantį elgsenos tipą. Yra trys elgsenos tipai:
  - *Interface* – veiksmai, kuriuos turės agentas atlikti yra nusprendžiami po mokymosi. Agentas gauna veiksmus iš modelio naudodamasis *OnActionReceived* funkcija.
  - *Heuristic* – veiksmai yra aprašomi ranka *Heuristic* funkcijoje ir tuomet paduodami *OnActionReceived* funkcijai.
  - *Default* – veiksmai yra paduodami *OnActionReceived* funkcijai mokymosi proceso metu.

12.2. Aštuonių laisvės mazgų roboto ranka.

12.2.1. Rankos laisvės mazgų tipai.

Paprastai rinkoje naudojamos rankos turi tik 4 arba 6 laisvės mazgus. Kiekvienas laisvės mazgas yra klasifikuojamas į vieną iš dviejų tipų: sukamasis arba lenkiamasis (žr. pav. 38).

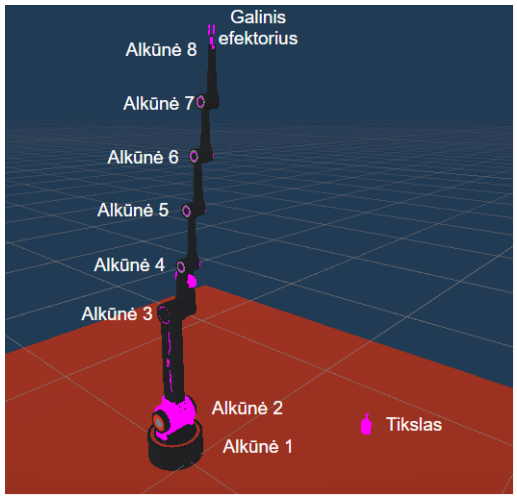


Pav. 38 roboto ranka, vaizduojanti sukamąjį bei lenkiamąjį mazgą.

Kiekvieno roboto rankos mazgo sulenkimo bei sukimo diapazonas visada bus nuo 0 iki 360 laipsnių. Dėl roboto, kaip fizinio asmens savybių, kai kurių mazgų diapazone yra sudedamos ribos. Taip yra dėl to, jog robotas neatsitrenktų į patį save. Toliau bus pateiktas realus 8 laisvės mazgų bei galinio efektoriaus modelis, naudotas šiame magistriniame darbe.

12.2.2. Sukurtos 8 laisvės mazgų rankos modelis.

Sukurta roboto ranka turi 8 laisvės mazgus ir galini efektorių. Kuriant šią ranką, buvo pasitelktas 5 laisvės mazgų rankos pavyzdys [Kan20].



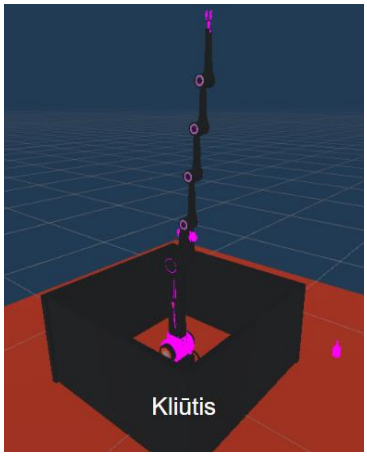
Pav. 39, įgyvendintas modelis.

- Tikslas – tikslo taškas erdvėje, atsitiktinai keičiantis vietą erdvėje aplink ranką. X, Y, Z koordinatės yra apskaičiuojamos C# Random.Range() funkcija, 0.5f – 0.8f intervale.
- Alkūnė 1 – sukamoji. Apsisukimo laipsnis – 360.
- Alkūnė 2 – lenkiamoji. Lenkimo diapazonas nuo -90 iki 90 laipsnių.
- Alkūnė 3 – lenkiamoji. Lenkimo diapazonas nuo -80 iki 80 laipsnių.
- Alkūnė 4 – lenkiamoji. Lenkimo diapazonas nuo -80 iki 80 laipsnių.
- Alkūnė 5 – lenkiamoji. Lenkimo diapazonas nuo -80 iki 80 laipsnių.
- Alkūnė 6 – lenkiamoji. Lenkimo diapazonas nuo -80 iki 80 laipsnių.
- Alkūnė 7 – lenkiamoji. Lenkimo diapazonas nuo -80 iki 80 laipsnių.
- Alkūnė 8 – lenkiamoji. Lenkimo diapazonas nuo -90 iki 90 laipsnių.
- Galinis efektorius – roboto rankos galūnė.

Roboto ranka turi 8 lenkiamąsias alkūnes ir vieną sukamąją. 3D modelis yra paimtas iš https://sketchfab.com/3d-models/robot-arm-22d9367c8d2f4457b3a4e74193e86ac9 [Joe19] ir modifikuotas iki 8 laisvės mazgų roboto rankos.

12.2.3. Aplinkos kliūtys.

Šiame magistriniame darbe buvo iškelti keli tikslai. Sukurti 8 laisvės mazgų roboto ranką bei apmokyti ją optimaliai siekti savo tikslo, mūsų atveju - butelio (žr. pav. 39). Darbo eigoje buvo sugalvota pasunkinti užduotį ir įdėti į roboto galimų judesių erdvę kliūčių. Taip buvo nuspręsta, nes norima ištirti, kaip 8 laisvės mazgų robotas mokinsis erdvėje su kliūtimis bei, koks bus galutini rezultatas.



Pav. 40, roboto ranka su kliūtimis.

Kliūtis imituoja tvorą aplink roboto ranką. Tvora susideda iš 4 atskirų sienelių. Sienų bei roboto rankos aukščio santykis yra 1:3. Buvo daryta daug eksperimentų bei pastebėta, jog jei padidintume tvoros aukštį iki santykio 2:3, tai proporcingai sumažintų roboto rankos judėjimo galimybes. Dėl to roboto mokymosi laikas būtų labai smarkiai prailginamas. Praėjus 10 mln. mokymosi iteracijų su 2:3 dydžio siena, rankos surinktas atlygis vos siekė 0,5, kai paprastai be sienų siekia apie 40 – 60.

Prie kiekvieno iš sienelių buvo pridėta programa, kuri tikrina ar roboto ranka nekirto/neprilietė sienelės. Jei ranka vis dėl to ją kirto, tai AI modeliui yra pridedamas minusinis

atlygis, šiuo atveju -0.0005f bei sustabdomas epizodas. Jei viso epizodo metu roboto ranka neprilietė nors vienos iš sienelių, pridedamas atlygis – 0.5f. (žr. priedą. pav. 41, 42).

Taip pat, grindis, ant kurių robotas stovi, galima įvardinti antrąja kliūtimi. Jos įgyvendinimo principas yra toks pat, kaip ir sienų. Tačiau, yra tik vienas skirtumas. Roboto rankos alkūnė 1 gali liesti grindis (žr. pav. 39).

12.3. Aštuonių laisvė mazgų roboto rankos mokymas bei AI specifikacija.

12.3.1. Roboto rankos pagrindinis tikslas, AI išvestis, duomenų atvaizdavimo būdas.

Roboto rankos mokymosi tikslas yra, iš bet kurios būsenos pasiekti buteliuką simuliacinėje aplinkoje. Ranka negali kirsti grindų bei sienų aplink ją. Tikslo komponentas, mūsų atveju – butelis, visą laika bus virš žemės bei roboto rankos pasiekiamumo trajektorijoje.

Apmokyto dirbtinio intelekto išvestis bus judesių rinkinys. Kiekvienas judesys reprezentuos kiekvieno laisvės mazgo pasisukimą iki tol, kol paskutinis rinkinio judesys bus tokie mazgų pasisukimo laipsniai, jog galinis efektorius pasieks tikslą.

AI modelyje viskas yra atvaizduojama normalizuota forma. Tai reiškia, jog veiksmai bus vaizduojami kaip realieji skaičiai [-1; 1] rėžiuose. Mokymosi metu šios reikšmės yra konvertuojamos į laipsnius.

Mokymosi sesijos yra suskirstytos į epizodus. Kiekvienas epizodas vyksta kol:

- Pasiekiamas veiksmų limitas.
- Pasiekiamas tikslas.
- Pasiekiama tokia būsena, iš kurios nebeįmanoma atlikti gero veiksmo.

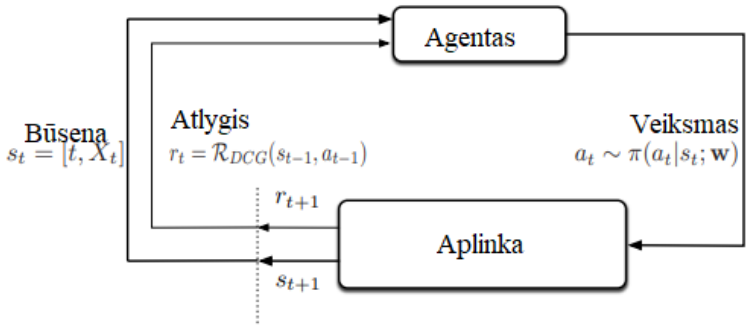
MDP (žr. skyrių 11.3.2) mokymas yra paremtas tuo, jog AI modulis bando pasiekti maksimalų atlygį, kiekvieno epizodo gale.

12.3.2. Markov sprendimų procesas (MDP).

Unity ML agentas naudoja Markov sprendimų procesą [WXL+17]. Tai procesas, kuris leidžia priimti iš dalies atsitiktinius tačiau ir kontroliuojamus sprendimus. MDP yra naudingas tiriant optimizavimo problemas. Jis yra plačiai naudojamas dinaminiam programavime. MDP yra sugalvotas 1950 metais bei pirmoji tėvinė knyga buvo „Dynamic Programming and Markov Processes“, Ronald Howard 1960 m.

MDP susideda iš 5 dalių:

- Būsena – tai yra sąrašas apibūdinimų, kurie AI modeliui apibrėžia aplinką.
- Veiksmai – tai yra diskretus veiksmų masyvas, kuriuos agentas gali vykdyti. Veiksmų masyvas yra glaudžiai susijęs su būsena.
- Perėjimas – tai yra T(S,A) funkcija $T: S \times A \rightarrow S$, kuri, atsižvelgiant į pasirinką veiksmą, transformuoja seną būseną į naują.
- Atlygis – tai padarytų teisingų/neteisingų veiksmų matavimo vienetas.
- Strategija – ji apibūdina agento elgseną, kuri yra tikimybinis pasiskirstymas galimiems jo veiksmams.



Pav. 43, MDP agento bei aplinkos sąveika.

MDP agento bei aplinkos sąveikos ciklas (žr. pav. 43):

1. Agentas mato būseną.
2. Agentas atlieka veiksmus.

3. Agentas gauna atlygį/įvertinimą.
4. Agentas kartoja viską nuo pradžių (1).

12.3.3. Atlygio schema.

Pagal MDP modelį, kiekvienam roboto rankos atliktam veiksmui, kurį sugeneravo ML agentas reikia gražinti atlygį. Atlygis gali būti teigiamas arba teigiamas. Jei mokoma ranka atlieka veiksmą, kuris ją priartina prie tikslo ir nėra atliekamų draudžiamų veiksmų, tokiu atveju duodamas teigiamas atlygis. Tačiau, jei roboto ranka atlieka daugiau draudžiamų judesių, arba tokių, kurie nepriartina jos prie tikslo, tokiu atveju yra duodamas neigiamas atlygis, kitaip – bauda. Taip pat, duodama didelė bauda, jei mokomas modelis patenka į nebesugrįžtamą būseną, pavyzdžiui, atsitrenkia į žemę ar sienas. Duodamas didelis atlygis tokiu momentu, kai ranka pasiekia tikslą.

Naudojama atlygio schema:

- Kai ranka atsitrenkia į sieną, -0,0005f bauda.
- Kai ranka atsitrenkia į grindis, -1,0f bauda.
- Kai ranka pasiekia tikslą, 1,0f atlygis.
- Kai ranka priartėja prie tikslo, yra skaičiuojamas atlygis, kai iš buvusios visos distancijos atimama likusi distancija tarp tikslo ir rankos galinio efektoriaus.
- Kai ranka patolėja nuo tikslo, tai skaičiuojama bauda, kai atimama iš pradinės distancijos esama distancija ir gaunamas minusinis realus skaičius, kurį atimame iš bendro rankos sukaupto atlygio.
- Kai rankos absoliuti visų laisvės mazgų laipsnių suma yra arčiau 0, tokiu atveju yra duodamas bonusas. Kuo arčiau nulio, tuo bonusas didesnis. Rėžiai – [0,5f; -0,5f]. Taip norėtasi išlaikyti, jog ranka būtų kuo mažiau sulenkta.
- Kai rankai pasiseka greičiau pasiekti rezultatą, nei praeitą karta, gauna +0,5f bonusą. Atvirkščiu atvejų -0,5f baudą.

12.3.4. Mokymo modelio konfigūracija.

Modelio konfigūracija yra laikoma *config.yaml* faile (žr. priedą, pav. 44).

Šis failas yra suskirstytas į dvi dalis: bendri bei „Robotarm“ parametrai. Juose laikomas ne mažas skaičius konfigūracijos kintamųjų, tačiau pagrindiniai yra trys:

- *Batch_size, hidden_units* – mazgų skaičius kiekviename neuroninio tinklo sluoksnyje.
- *Num_layers* – paslėptų neuroninio tinklo sluoksnių skaičius.
- *Max_steps* – mokymo žingsnių skaičius.

12.3.5. Agentas (kodas).

Mūsų atveju roboto kontroleris turi nustatyti agentą, kuris privalo būti apmokytas. Jog tai įgyvendinti, mes turime pridėti prie roboto modelio simuliacinėje aplinkoje programą, kur bus įgyvendintas agento funkcionalumas. Šis agentas išplečia jau integruoto agento funkcionalumą, tad lieka perrašyti tam tikras funkcijas - public class RobotControllerAgent : Agent.

Pirmas perrašytas tėvinės agento klasės metodas yra *Initialize()* (žr. pav. 45). Mes nustatome visus rankos laisvės mazgus į numatytas pozicijas – visi mazgų laipsniai įgauna reikšmę 0 (žr. pav. 46).

```
public override void Initialize()
{
    ResetAllAxis();
    MoveToSafeRandomPosition();
}
```

Pav. 45, metodas Initialize().

```
private void ResetAllAxis()
{
    armAxes.All(c =>
    {
        c.transform.localRotation =  Quaternion.Euler(0f, 0f, 0f);
        return true;
    });
}
```

Pav. 46, metodas ResetAllAxis().

Tuomet iškviečiam *MoveToSafeRandomPosition()* metodą, kuris bando pajudinti ranką atsitiktine kryptimi bei patikrina ar galinis efektorius nesikerta su žeme.

Kitas perrašytas metodas yra *OnEpisodeBegin()* (žr. priedą, pav. 47.). ML agento mokymai yra paleidžiami epizodais. Kiekvienas epizodas turi 5000 ciklų. *OnEpisodeBegin()* metodas yra vieta, kur paruošiama aplinka po praeito epizodo bei inicijuoja naują epizodą. Mūsų atveju mes gražinam ranką į pradinę būsena su *ResetAllAxis()* funkcija (žr. pav. 46), iškviečia jau aptartą *MoveToSafeRandomPosition()* metodą bei pakeičia tikslo koordinates erdvėje (žr. priedą, pav. 48). Taip pat, yra nustatomas laikas, kada prasidėjo epizodas. *UpdateNearestComponent()* metode yra ne tik atnaujinamos tikslo koordinatės, bet ir apskaičiuojama pradinė distancija tarp galinio efektoriaus ir tikslo bei pagrindo kampas.

Trečiojo metodo pavadinimas yra *OnActionReceived(float[] vectorAction)* (žr. priedą, pav. 49). Šis metodas bus iškviestas vykstant „spėjimo“ stadijai. Tai taip pat gali būti sutapatinta ir su neuroninio tinklo išvesties mazgais. Aptariamas metodas gauna *vectorAction* float tipo masyvą, kuris reprezentuoja neuroninio tinklo numatytus rankos veiksmus. Kiekvienas rankos mazgas yra transformuojamas iš realaus skaičiaus į sukimosi laipsnius. Tuomet yra skaičiuojama distancija nuo galinio efektoriaus iki tikslo. Jei jis yra didesnis, nei praeitų epizodų geriausio distancijos rezultato, tai reiškia, jog rankos galinis efektorius patolėjo nuo tikslo ir iš rankos surinkto atlygio yra atimama buvusios geriausios distancijos ir dabartinio epizodo distancijos skirtumas. Kitu atveju yra apskaičiuojamas skirtumas tarp pradinės epizodo distancijos ir galutinės. Skirtumas gaunasi teigiamas skaičius ir yra pridedamas prie atlygio. Šio veiksmo gale dabartinė distancija yra išsaugojama kaip geriausia praeitų atliktų veiksmų distancija. Taip daroma dėl to, jog ateities veiksmuose būtų galima palyginti ar ranka priartėjo prie tikslo ar ne. Kiekvieno veiksmo gale yra pridedama bauda, kurios svoris yra -0,0001f. Tai laidžia neuroniniam tinklui suprasti, jog siekti tikslo reikia per kuo mažesnį veiksmų skaičių.

Kai galinis efektorius priliečia/kerta tikslą/butelį, tuomet teigiama, jog tikslas yra įgyvendintas. Mūsų atveju, yra iškviečiamas *JackpotReward(Collider other)* funkcija (žr. priedą, pav. 50). Šis metodas apskaičiuoja bonusą, prideda jį prie 1,0f. Taip pat, apskaičiuoja visų roboto rankos mazgų kampų sumą. Ją paverčia absoliučiu skaitmeniu bei duoda jį *AddBonusWhenAnglesAreSmall* metodui (žr. priedą, pav. 51), kuris sprendžia ar pridėti didesnį bonusą prie atlygio, ar atimti baudą.

Po to, kai apskaičiuojamas atlygis, atsižvelgiant į absoliučią kampų sumą, yra iškviečiama *AddTimeReward()* funkcija (žr. priedą, pav. 52), kuri apskaičiuoja, kiek ranka užtruko pasiekti tikslą. Tuomet palygina, ar tai geriausias rezultatas lyginant su praeitų epizodų rezultatas. Jei tai yra geriausias, tai pridedamas laiko bonusas 0,5f, jei ne atimamas -0,5f.

Gale *JackpotReward(Collider other)* metodo yra pridedamas atlygis prie sukaupto bei atnaujinamos tikslo koordinatės.

Taip pat, prie tikslo objekto yra pridėtas funkcionalumas, kuris tikrina, ar atnaujinus tikslo koordinates, jis nesikirs su sienos koordinatėmis. Jei taip įvyks, bus per nauja atnaujinamos tikslo koordinatės (žr. priedą, pav. 53).

12.4. Mokymosi rezultatai bei išvados.

Mokymo modelis turi 3 paslėptus neuroninio tinklo sluoksnius bei atlieka 5 arba 10 mln. žingsnių.

12.4.1. Aštuonių laisvės mazgų mokymosi rezultatai be kliūčių (5 mln.).

Buvo apmokytas prieš tai sukonstruotas/aptartas 8 laisvės mazgų robotas. Mokymo metu buvo atlikti 5 mln. žingsnių. Geriausio mokymo metu, buvo surinktas ~45.0 atlygis.

Nors ir buvo įgyvendintas funkcionalumas, kuris pridėdavo bonusą prie rankos atlygio, kai ji būdavo vis mažiau susilenkusi – ranka vis tiek išliko sulinkusi. Buvo pastebėta, jog jos išsitiesimo greitis pagreitėjo ir ji tapo judresnė.

Įrašą, kaip ranka siekia tikslo, galima rasti šioje nuorodoje - https://streamable.com/3tp1i6. Atlygio, *Policy Loss* bei *Value Loss* funkcijų diagramas galima rasti 54, 55, 56 prieduose.

12.4.2. Aštuonių laisvės mazgų mokymosi rezultatai su kliūtimis. (10 mln.)

Kai simuliacinėje aplinkoje atsirado kliūtys, atsirado keturi labai svarbūs objektai, kurių negali robotas paliesti. Tie keturi objektai ir yra sienos aplink ranką. Iš pradžių, kaip jau buvo minėta, jų aukštis buvo pasiekęs $\frac{2}{3}$ rankos aukščio. Paleidus mokymą, po 3 mln. iteracijų buvo pastebėta, jog modelis neapsimoko. Jo atlygis nesiekė nei 1,0, svyravo ~0,3. Taip nutiko todėl, kad

tokio aukščio tvora ženkliai sumažino rankos judėjimo plotą ir, kiekvieną kartą jai pajudėjus, buvo prisiliesta prie tvoros. Tokiu atveju agentas nešdavo didelį baudų skaičių modeliui ir jam būdavo sunku atskirti, ką jis daro blogai. Jog tai išspręsti, tektų išbandyti paleisti labai ilgą mokymą, kuris siektų ne 10 mln. iteracijų, o galbūt 100 mln. Tokio ilgio mokymas truktų ~ 10 val., nes buvo pastebėta, jog 1 mln. iteracijų mokymas užtrunka apie 60 min.

Jog būtų išspręsta aukščiau paminėta problema, buvo nuspręsta sieną, juosiančia ranką, sumažinti dvigubai. Siena buvo sumažinta iki $\frac{1}{3}$ roboto aukščio. Tai suteikė roboto rankai daug daugiau erdvės judėti tikslo link. Mokymosi diagramas galima rasti žemiau (žr. priedus, pav. 57, 58, 59). Jose matoma, kaip kito atlygis kiekvienos iteracijos metu bei *Policy/Value Loss* funkcijos.

Atlygio, *Policy Loss* bei *Value Loss* funkcijų diagramas galima rasti 54, 55, 56 prieduose.

Sienų atsiradimas simuliacinėje aplinkoje toliau sukėlė problemų. Kai jos atsirado, roboto rankai ne tik reikėjo pasiekti tikslą, bet ir reikėjo neatsitrenkti į sienas. Keičiant konfigūraciją bei atlygio kiekio pridėjimą/atėmimą, buvo gauta, jog roboto rankai didesnis tikslas tapo ne pasiekti tikslą, o neatsitrenkti į sieną. Aukščiau matomos diagramos yra šio probleminio modelio.

Vaizdo įrašą galima rasti šioje nuorodoje - https://streamable.com/0gg31v.

12.4.3. Aštuonių laisvės mazgų mokymosi rezultatai su kliūtimis, kai tolstant iš atlygio atimamas nutolęs atstumas.

Visuose prieš tai aprašytuose bandymuose, kai ranka tolo nuo galutinio tikslo – butelio, iš bendro atlygio buvo atimamas geriausio buvusio atstumo bei dabartinio atstumo tarp galinio efektoriaus ir tikslo skirtumas. Kai aplinkoje atsirado daugiau kliūčių, mūsų atveju – tvora, atsirado daugiau tikslų. Modeliui išmokstant, jog negalima atsitrenkti į sienas, jo pagrindinis tikslas – pasiekti butelį, pradėjo blankti. Norėta jį sustiprinti, duodant didesnę baudą, kai ranka nutolsta. Dėl to buvo pakeistas kodas ir pradėta atiminėti ne anksčiau minėtą atstumų skirtumą, bet bendrą atstumą iki butelio. Išvados tokios, jog ranka nebeapsimokė. Ji nesurinko atlygio, nes baudos buvo per didelės. Atlygis geriausiu metu siekia vos daugiau 0.8. Atlygio funkciją galima rasti priede nr. 60.

12.5. Mokymosi rezultatų palyginimas.

Buvo išbandyta 8 laisvės mazgų roboto ranka be kliūčių, su kliūtimis bei skirtingomis konfigūracijomis. Lyginant mokymosi rezultatus tarp aplinkos su kliūtimis bei laiko ir susilenkimo funkcijomis ir be, galima teigti, jog ranka, be kliūčių bei papildomų funkcijų geriau apsimokė. Tai matosi ne tik iš grafikų, bet ir vaizdo įrašų, pateiktų ankstesniuose skyriuose. Žemiau (žr. priedą, pav. 61) matome šių dviejų konfigūracijų mokymosi atlygio kreives.

Iš priedo pav. 61 yra matoma, jog kai kuriuose momentuose, ranka be kliūčių generavo dvigubą atlygį, nei su kliūtimis. Taip pat, jų atlygių sumų skirtumas svyruoja tarp 35-45 procentų.

Taip pat, buvo pastebėta, jog 8 laisvės mazgų ranka yra ganėtinai didelė mokymo moduliui. Jis epizodo pradžioje sulenkdavo ranką ir epizodui einant ją po truputi vis labiau ištiesdavo, jog būtų pasiektas tikslas.

Lyginant rankų apsimokymą, kai bauda už rankos nutolimą nuo tikslo buvo skaičiuojama dviejų atstumu skirtumu ir absoliutaus atstumo atėmimu, yra matomas aiškus skirtumas. Kai iš rankos surinkto atlygio buvo atimama bauda, kuri buvo lygi absoliučiam atstumui tarp galinio efektoriaus ir tikslo, matoma, jog bauda tapo per didelė ir ranka nebeapsimokė. Mokymosi atlygio diagrama galima rasti priede nr. 62.

13. Aštuonių laisvės mazgų mokymosi rezultatai su kliūtimis – pasisekęs bandymas.

Visi prieš tai aprašyti aštuonių laisvės mazgų roboto rankos su kliūtimis bandymai buvo nesėkmingi. Po gilesnių tyrimų buvo nuspręsta iš pradžių įgyvendinti šešių laisvės mazgų roboto ranką su kliūtimis ir pasižiūrėti, kokie bus rezultatai. Informaciją galima rasti 13 – ame skyriuje. Po sėkmingo 6 laisvės mazgų roboto rankos mokymo, ši ranka buvo patobulinta iki 8 laisvės mazgų bei ištaisyta keletas klaidų.

13.1. Ištaisytos praeities bandymų klaidos.

1. Pirmoji, esminė ištaisyta klaida, kuri neleisdavo pasiekti praeitiems robotams ganėtinai gero rezultato, yra ta, jog anksčiau robotams buvo duodamas atlygis už tai, dėl ko jie netūrėtų gauti atlygio. Anksčiau robotai, jei atsitrenkdavo į sieną, gaudavo baudą, o jeigu

neatsitrenkdavo į sieną, gaudavo atlygį. Šis sienos atsitrenkimo metodas yra blogas, nes buvo pastebėta, jog duodant robotui atlygį už tai, kad jis tiesiog neatsitrenkė į sieną, jį klaidindavo. Tad buvo nuspręsta duoti tik baudą už tai, ką robotas blogai padarė, ir atsisakyta duoti atlygį už tai, ko robotas blogai nepadarė. Šis metodas leido robotui AI moduliui tiksliai suprasti, ką robotas blogai daro ir ateities iteracijose to išvengti.

2. Antroji, ganėtinai paprasta klaida, buvo ta, jog ne visos roboto rankos dalys tūrėjo daviklius, kurie duotų baudą už atsitrenkimą į kliūtį. Dėl šios klaidos, už daug atsitrenkimų į sieną nebuvo duodamos baudos, kas labai suprastindavo roboto rankos mokymąsi. Dažnu atveju, roboto ranka apsimokydavo taip, jog ji nevengdavo kliūčių.
3. Trečioji ištaisyta klaida, tai buvo visų papildomų atlygių davimų atsisakymas. Anksčiau roboto rankai būdavo pridedamas bonusas, jei ji mažai susilenkdavo, jei ji pasiekdavo rezultatą greičiau nei praeitoje epochoje. Šių papildomų atlygių davimų atsisakymas padėjo aštuonių laisvės mazgų roboto rankai tiksliau suprasti, kas yra gerai ir kas daroma blogai. Robotas gaudavo teigiama atlygį tik tais atvejais, kai tai yra susiję su tikslo pasiekimų, o baudas gaudavo, kai toldavo nuo tikslo arba atsitrenkdavo į sienas.
4. Ketvirtasis patobulinimas buvo gana paprastas – mokomų roboto rankų papildymas mokymo aplinkoje. Praeityje mokymo aplinkoje mokydavosi 12 robotų vienu metu, kurių rezultatai buvo įrašomi į vieną AI modulį. Dabar skaičius buvo išplėstas iki 162 robotų. Tai padėjo gerokai sutaupyti laiko.

13.2. Rezultatai.

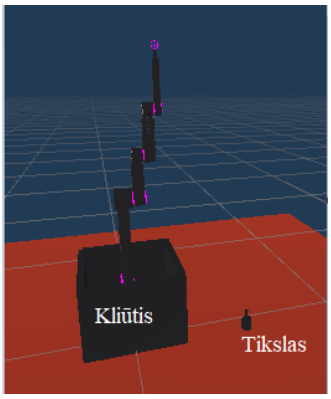
Rezultatai, lyginant su praeities bandymais, kai aštuonių laisvės mazgų roboto ranka nesurinkdavo net +1.0 atlygio, yra ryškiai matomi. Bandymai buvo daromi su 10 mln. iteracijų ir geriausias surinktas atlygis buvo 132.1. Tai yra geriausias gautas rezultatas iš visų bandymų. Kaip atlygio suma keitėsi per visas 10 mln. iteracijų galima pamatyti priede nr. 63. Kreivėje galima pastebėti keletą atlygio duobių. Pavyzdžiui, ties 2.5 milijonine iteracija atlygis buvo kritęs nuo 40 iki 5, ties 4.6 mln. iteracija, taip pat buvo kritimas per ~63 vienetus. Taip įvykdavo todėl, nes kol dar AI modulis nėra pilnai apmokytas, mokomos rankos atsidurdavo situacijose, kur nerasdavo geros išeities ir joms tekdavo tik vienas kelias – atsitrenkti į sieną. Po tokių situacijų AI modulis gaudavo dideles baudas ir tai lemdavo, jog daugiau ranka nebeatsidurdavo tokiose situacijose. Taip pat visas mokymas truko apie 5.5 val. Vidutinė 50000 iteracijų mokymo trukmė buvo apie 100 sekundžių. Rezultatų įrašas: https://streamable.com/gniqvb.

14. Praktinis šešių laisvės mazgų roboto rankos įgyvendinimas.

Bandant įgyvendinti aštuonių laisvės mazgų roboto ranką, kilo idėja sukurti panašią simuliacinę aplinką, kurioje būtų įgyvendinta populiarioji šešių laisvės mazgų roboto ranka. Ši ranka būtų pratestuota aplinkoje su kliūtimis ir aplinkoje be kliūčių. Ši idėja kilo, nes buvo pastebėta, jog 8 laisvės mazgų roboto ranką yra tikrai sunku apmokyti, dėl to šešių laisvės mazgų ranka būtų palyginimas aštuonių alkūnių rankai.

14.1. Simuliacinė aplinka bei roboto ranka.

Simualiacinė aplinka taip pat kaip ir aštuonių laisvės mazgų rankai buvo pasirinkta Unity. Taip pat buvo naudota *Unity Machine Learning Toolkit*. Žemiau pateiktame paveikslėlyje yra matoma roboto ranka simuliacinėje aplinkoje (žr. pav. 47).



Pav. 47, 6 laisvės mazgų roboto ranka.

Roboto ranka turi 6 laisvės mazgus bei galūnę. Laisvės mazgų pasisukimo laipsniai:

- 1 alkūnė – 360 laipsnių,
- 2 alkūnė – 140 laipsnių,
- 3 alkūnė – 160 laipsnių,
- 4 alkūnė – 140 laipsnių,
- 5 alkūnė – 160 laipsnių,
- 6 alkūnė – 180 laipsnių,

Alkūnės lenkimosi laipsnių amplitudė nėra vienoda, nes taip buvo išvengta bereikalingo perdėto rankos susilenkimo.

Taip pat, integruotos kliūtys buvo aprašomos būsenoje bei paduotos agentui. Tai padeda padaryti *Unity.MLAgents.Sensors* biblioteka. Žemiau yra matoma būsena, kuri yra paduodama agentui (žr. pav. 48).

```
sensor.AddObservation(angles);
sensor.AddObservation(transform.position.normalized);
sensor.AddObservation(wall.transform.position.normalized);
sensor.AddObservation(wall1.transform.position.normalized);
sensor.AddObservation(wall2.transform.position.normalized);
sensor.AddObservation(wall3.transform.position.normalized);
sensor.AddObservation(nearestComponent.transform.position.normalized);
sensor.AddObservation(endEffector.transform.TransformPoint(Vector3.zero).normalized);
Vector3 toComponent = (nearestComponent.transform.position - endEffector.transform.TransformPoint(Vector3.zero));
sensor.AddObservation(toComponent.normalized);
sensor.AddObservation(Vector3.Distance(nearestComponent.transform.position,endEffector.transform.TransformPoint(Vector3.zero)));
sensor.AddObservation(StepCount / 5000);
```

Pav. 48, būsena paduodama agentui.

Agento programavime nenaudota papildomų rankos lenkimuisi mažinančių funkcijų.

Naudojama atlygio schema:

- Kai ranka atsitrenkia į sieną, -1f bauda.
- Kai ranka atsitrenkia į grindis, -1f bauda.
- Kai ranka pasiekia tikslą, +0.5f atlygis.
- Kai ranka priartėja prie tikslo, yra skaičiuojamas atlygis, kai iš buvusios visos distancijos atimama likusi distancija tarp tikslo ir rankos galūnės.
- Kai ranka patolėja nuo tikslo, tai skaičiuojama bauda, kai atimama iš pradinės distancijos esama distancija ir gaunamas minusinis realus skaičius, kurį atimame iš bendro rankos sukaupto atlygio.
- Kai ranka žiūri tikslo link, yra skaičiuojamas bonusas, kuris pridedamas prie atlygio. Jis svyruoja nuo -1f iki +1f.

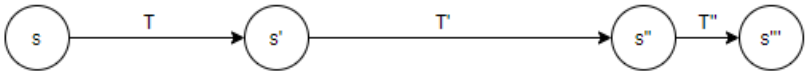
14.2. Rezultatai.

Mokymosi atlygio kreivė ganėtinai nepastovi. Dažnai atlygio suma keisdavosi per 30 vienetų. Grafiką galima pamatyti pav. 69. Pagal mokymo rezultatus, galima teigti, jog 6 laisvės mazgų robotas šį kartą sunkiau mokėsi nei 8 laisvės mazgų. Buvo vienas didelis atlygio kritimas iki -37.12 ties 4.65 mln. iteracija, po kurio sekė greitas kilimas. Taip įvyko dėl to, jog roboto rankos atsidūrė aklavietėje, kur nebežinojo, ką daryti. Tokiu atveju atsitrenkus į sieną, gavo baudą, po

kurios daugiau į tokią situaciją nebepapuolė. Vaizdo įrašą galima rasti: https://streamable.com/xp889d.

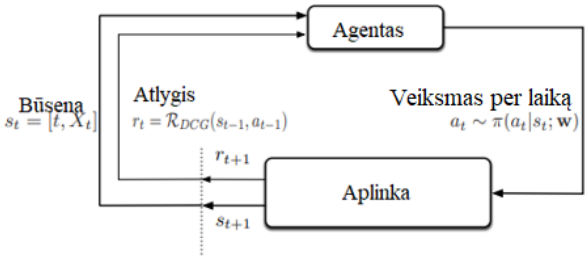
15. Dalinis Markov sprendimų procesas. (*Semi-MDP*).

Dar vienas mašininio mokymosi algoritmas, priskiriamas *reinforcement learning* algoritmų klasei, yra dalinis Markov sprendimų procesas *(semi-MDP)*. Jis yra panašus jau prieš tai aptarta Markov sprendimų procesą, bet šių dviejų procesų skirtumas padaro *Semi-MDP* unikaliu algoritmu. Pagrindinis šių algoritmų skirtumas yra tas, jog *semi-MDP* algoritmas veikia taip pat kaip *MDP*, tačiau atsiranda dar vienas ryškus rodiklis – laikas[21]. *SMDP* yra kaip ir *MDP* penketukas <S, A, P, F, r>, tačiau yra pridedamas laikas. Tai reiškia, jog kiekviena simuliacijos būsena turės tik tam tikrą laiką egzistencijos ir po to kai jis praeis, bus apskaičiuojamas atlygis ir pasikeis aplinkos būsena į naują. Dar labiau supaprastinant, roboto ranka turės tik tam tikrą apibrėžtą laiko tarpą, per kurį galės pasiekti tikslą. Kiekvienas laiko tarpas varijuoja – yra skirtingas vienas nuo kito.



Pav. 49, SMDP laiko intervalai.

Paveikslėlyje nr. 49 yra matomi SMDP galimi laiko intervalai. Kiekviena būsena (S, S`, S`` ir S```) turi savo egzistavimo laiką (T, T`, T``, T```). Taip pat linijų ilgis parodo, jog kiekvienas intervalas yra skirtingas.



Pav. 50, agento ir aplinkos sąveika SMDP.

16. Dalinio Markov sprendimų proceso įgyvendinimas.

Buvo įgyvendintas šis metodas 8 - ių laisvės mazgų roboto rankoje su kliūtimis.

16.1. Duoto laiko skaičiavimas iteracijai.

Praeitas agento modulis buvo patobulintas laiko funkcijomis. Pirmoji jų buvo *calculateGivenTime*. *SMDP* procese yra svarbu, jog duotas laiko tarpas atlikti veiksmus skirtųsi nuo prieš tai buvusio. *CalculateGivenTime* funkcija taip ir padaro. Ji kiekvieną iteraciją skaičiuoja perskaičiuoja, kiek laiko yra duodama iteracijai pabaigti veiksmus. Duoti laiko vienetai svyruoja tarp 5 bei 10 – ies sekundžių. Kodo pavyzdį galima rasti prieduose, 65 paveikslėlyje. Jei duotos sekundės mažesnės arba lygios 9 – ioms sekundėms, tai prie jų būdavo pridedama dar viena sekundė ir patikrinama, ar naujos duotos sekundės nėra lygios dešimčiai sekundžių. Jei taip, tai kintamasis, kuris nusako, ar didinti duotas sekundes, ar mažinti, yra pakeičiamas į *False*, kas reiškia, jog kitos iteracijos metu, duotas laikas bus pradėtas mažinti nuo dešimties sekundžių iki penkių. Taip pat yra daroma ir mažėjimo tvarkoje. Kiekvieną iteraciją yra atimama po vieną sekundę ir yra tikrinama ar duotos sekundės nepasiekė 5 sekundžių. Jei pasiekė, kintamasis, kuris nusako ar didinti/mažinti sekundes yra pakeičiamas į *True,* kas pasako, jog kitą iteraciją duotos sekundės bus didinamos. Jei kokiu nors būdu duotos sekundės nepateks į prieš tai aprašytus rėžius, duotos sekundės bus nustatytos nuo pradžių – 5, ir pridėjimo/atėmimo kintamasis bus nustatytas kaip *True*. Taip pat, duotas laikas iteracijai yra paduodamas į būseną.

16.2. Laiko sekimas iteracijoje.

Kiekieno epizodo pradžioje yra įsiminama, koks tuo metu yra laikas *Float* tipu. Kiekvieno roboto rankos judesio metu yra iškviečiama funkcija, kuri patikrina ar jau robotas peržengė duotą laiką pasiekti tikslą intervalą. Agentas iš tuo metu dabartinio laiko *Float* tipu atima laiką, kurį įsiminė epizodo pradžioje. Patikrina ar jis nėra didesnis už duotą laika, kurį paskaičiuoja *CalculateGivenTime* metodas, ir jei taip atsitinka, iškviečia *TimePenalty* metodą. Šį funkcionalumą galima rasti prieduose, 66 paveikslėlyje. *TimePenalty* metodas atima iš sukaupto atlygio -1f ir pabaigia epizodą. (Žr. pav. 67).

16.3. Rezultatai.

Roboto surinktas atlygis yra ganėtinai didelis. 9.4 mln. iteracijoje jis siekė net 81.89, po kurio pradėjo po truputi kristi. Mokymas baigėsi ties 10 mln. (Žr. pav. 68). Tačiau, kai pradėta

testuoti apmokytą modulį pagal *SMDP*, pastebėta, jog šis metodas nepasiteisino. Rezultato įrašą galima rasti: https://streamable.com/96phmo. Ranka neapsimokė, nes pasiekti tikslą jai dažnai reikėdavo daugiau laiko nei yra duota. Nuspręsta, jog *SMDP* nėra geriausias *RL* algoritmas roboto rankai mokyti.

17. Išvados

Po daug bandymų surasti gerą AI agento konfigūraciją, jog roboto ranka siektų savo tikslo ir išvengtų kliūčių, ji buvo rasta, tačiau tai dar galima tobulinti, jog pasiektume sklandų roboto rankos judėjimą. Išvados tokios, jog roboto ranka geriau apsimokys, kai atlygio ir baudų taisyklės bus kuo paprastesnės ir jų kiekis bus kuo mažesnis. Taip pat pastebėta, jog robotui reikia duoti baudą už tai, ką jis blogai padarė, tačiau nereikia duoti atlygio už tai, kad jis kažko blogai nepadarė. Taip pat reikia išvengti papildomų, ganėtinai nereikšmingų, taisyklių, kurios koreguoja atlygio kreivę – kuo atlygio/baudų taisyklių bus mažiau, tuo roboto ranka aiškiau supras, ko iš jos yra tikimąsi.

Aštuonių laisvės mazgų roboto ranka surinko truputi didesnę atlygio sumą nei šešių mazgų ranka. Tad galima teigti, jog du laisvės mazgai nepadaro didelės įtakos atlygio kreivei mūsų sukurtoje simuliacinėje aplinkoje. Mokymosi aplinkose robotų skaičius buvo padidintas iki 162. Aštuonių laisvės mazgų AI modulio mokymas užtruko 20 % ilgiau nei šešių laisvės mazgų. Lyginant šešių ir aštuonių laisvės mazgų roboto rankas, nors ir mūsų atveju aštuonių laisvės mazgų ranka gavo didesnį atlygį nei šešių, galima teigti, jog šešių mazgų ranka apsimoko greičiau ir paprasčiau. Nes kuo didesnis mazgų skaičius, tuo agento konfigūraciją sunkiau pritaikyti simuliacinei aplinkai. Gali kilti papildomų kliūčių kaip roboto rankos per didelis lenkimasis ar rankos atsitrenkimas į save.

MDP algoritmas yra puikiai pritaikytas roboto rankos kelio optimizacijai, nes jo pagrindinis tikslas yra mokytis iš praeityje įvykdytų klaidų bei laimėjimų ir taip bandyti surasti trumpiausią bei greičiausią kelią tikslo link. Aišku, daug faktorių priklauso nuo AI agento konfigūracijos bei kaip jis buvo suprogramuotas, tačiau, jei jis buvo kruopščiai įgyvendintas, galimi roboto rankos rezultatai optimalumo kontekste bus ganėtinai geri.

Taip pat buvo įgyvendintas *SMDP RL* algoritmas. Gautas rezultatas nėra idealus ir perteklinis. Roboto ranka nebesuprato savo pagrindinio tikslo, kai iteracijos buvo nutraukiamos, jei

ji per tam tikrą laiką nepasiekdavo taikinio. Prieita prie išvadų, jog roboto rankai reikia leisti iki galo užbaigti judesius, jog būtų įgyvendintas pakankamai geras mokymas. Šio magistrinio darbo metu labiau tiko *MDP* algoritmas, nei *SMDP*.

Renkantis, kokį robotą įgyvendinti, reikia didelį dėmesį atkreipti į aplinką, kurioje jis egzistuos. Šio magistro darbo simuliacinės aplinkos kliūtis buvo tvora, dėl to aštuonių laisvės mazgų roboto ranka puikiai tiko pasiekti tikslą už tvoros, siekiant jos per kliūties viršų. Jei aplinkoje jokių kliūčių nebus arba ranka bus mokoma su tikslu, kuris nėra pakankamai nutolęs nuo jos, kitaip tariant, nebus išnaudojamas rankos ilgumo pranašumas, ranka apsimokys taip, jog kai kurie mazgai nebus reikalingi. Ranka atrodys susisukus arba persilenkus. Taip AI modulis mokydamasis sutrumpina rankos ilgumą iki reikiamo. Taip pat dėl to padidėja šansai, jog ranka atsitrenks pati į save.

18. Santrauka (*Summary*).

In the rapidly modernized world of the 21st century, robots are beginning to play a significant role. Looking back, robotic manipulators and contractors were the very first robots used in the industry. They have been used since the 1960s. Nowadays, when computers have evolved very dramatically by the technical possibilities, humans managed to implement self learning algorithms to robots in simulation environments and reality. In those environments, robots which are using self learning algorithms like reinforcement learning (*RL*), can learn how to interact with different environments. This master's thesis will aim to implement one of the main goals - to apply the algorithm of deep enhanced neural network to optimize the robot arm of eight freedom nodes and achieve the goal.

In order of this goal to be achieved, tasks have been set:

- To find out the principles of operation of the robot arm.
- To investigate the applications of neural networks in the field of optimization.
- Train a custom neural network to optimize the path of robot arm movement.
- Test a trained network.
- Visually simulate the obtained results to see a clear result of the work.

During this master's thesis a simulation environment was created with *Unity* platform. For machine learning algorithm, the *Unity ML* plugin was used which has *Tensorflow* algorithm integrated inside. Experiments were made with 8 and 6 freedom knots in environment with obstacles and without. Markov – decision process algorithm was used for AI learning.

Results are positive because learned arms had quite accurate learning. Robotic arm tries to reach a goal and do not touch obstacles. Robot tries to find optimal path from beginning to the goal. Also, there are always space for improvements. One of them could be to teach robot arm to move smoothly.

19. Literatūros sąrašas.

[WMH+12]. Mahisorn Wongphati, Yushi Matsuda, Hirokata Osawa and Michita Imai. „Where do you want to use a robotic arm? And what do you want from the robot?“. Rugsėjis 9-13 d. 2012, Paryžius, Prancūzija.

[Cab12]. Josh Cable. „The Future of Robotics in Manufacturing: Moving to the Other Side of the Factory". Vasario 12 d. 2012 m.

[Kel10]. Tod. R. Kelley. „Optimization, an Important Stage of Engineering Design". Purdue University, 2010 m.

[WOS+]. Gregory S. Weinstein, Bert W. O'Malley Jr, Wendy Snyder, BS; Eric Sherman, Harry Quon. „Transoral Robotic Surgery". Amerikos Medicininė Asociacija, 2007 m.

[IIK12]. Jamshed Iqbal, Raza ul Islam, and Hamza Khan. „Modeling and Analysis of a 6 DOF Robotic Arm Manipulator", kanadiečių žurnalas „Electrical and Electronics Engineering“ tomas 3, Nr. 6, liepa 2012 m.

[GFG]. „A* Search Algorithm". Prieiga per Internetą: <https://www.geeksforgeeks.org/a-search-algorithm/>.

[HWW]. Dominik Henrich, Christian Wurll and Heinz Worn, „6 DOF path planning in dynamic environments – A parallel on-line approach", „Process Control and

Robotics” institutas, kompiuterių mokslo departamentas, Karlsruhe universitetas, Vokietija, 1998 m.

[Twa18]. Marcin Twardak, „Inverse Kinematics In a Robotic Arm“. Balandis 25 d. 2018. Prieiga per Internetą: <https://medium.com/@marcin_97686/inverse-kinematics-in-a-robotic-arm-learn-how-to-calculate-it-54156a63b65b>.

[UOR]. University of Regina, CS 408/8080 kursas, Prieiga per Internetą: <http://www2.cs.uregina.ca/~anima/408/Notes/Kinematics/Inverse_Kinematics.htm>.

[AD03]. T. Asfour and R. Dillmann, „Human-like Motion of a Humanoid Robot Arm Based on a Closed-Form Solution of the Inverse Kinematics Problem“ Karlsruhe Haid-und-Neu-Str universitetas, 27-31 d. spalis 2003 m., Karlsruhe, Vokietija.

[DK16]. Serkan Dereli, Rasit Koker „In a research on how to use inverse kinematics solution of actual intelliggent optimization method“, „Research Gate“ lapkritis 2016 m.

[AM92]. Dave Anderson, George McNeill, „Artificial Neural Networks Technology“, „Kaman Sciences“ įmonė, Rugpjūtis 20 d. 1992 m., Roma, Vokietija.

[LSW+15]. Gongning Luo, Dong Sui, Kuanquan Wang, Jinseok Chae, „ Neuron anatomy structure reconstruction based on a sliding filter“, „ BMC Bioinformatic“, 2015 m.

[Suz13]. Kenji Suzuki, „Artificial Neural Networks“. Čikagos universitetas, leidykla „InTech“, 2013 m., Illinois, Jungtinės Amerikos Valstijos.

[KD05]. C. W. Kennedy, P. J. Desai, „Modeling and control of the Mitsubishi PA-10 robot arm harmonic drive system“, 2005 m.

[SK13]. Ashish Srivastava, Ajay Kumar, „An Optimization Technique to Solve the Inverse Kinematics of Robot Manipulator“ „Department of Mechanical Engineering Noida Institute of Engineering & Technology“, 2013 m., Noida, Indija.

[ADK16]. Ahmed R. J. Almusawi, L. Canan Dülger ir Sadettin Kapucu, „A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm (Denso VP6242)“, „Hindawi“ leidykla, 2016 m.

[Baj20]. Prateek Bajaj, „Reinforcement learning“, gegužės 17d. 2020 m. Prieiga per Internetą: <https://www.geeksforgeeks.org/what-is-reinforcement-learning/>.

[BEO05]. Z. Bingul, H.M. Ertunc and C. Oysu, „Comparison of Inverse Kinematics Solutions Using Neural Network for 6R Robot Manipulator with Offset“, Kocaeli universitetas, sausis 2005 m., Turkija.

[Mor77]. Jorge J. More, „The Levenberg-Marquardt Algorithm: Implementation And Theory“ sausis, 1 d. 1977 m. Jungtinės Amerikos Valstijos.

[BCP+16]. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, „OpenAI Gym“, birželis, 5 d. 2016 m.

[Mat]. Matlab dokumentacija. Prieiga per Internetą: <https://se.mathworks.com/help/matlab/>.

[YEB+16]. Jargalbaatar Yura , Mandakh Oyun-Erdene , Bat-Erdene Byambasuren, and Donghan Kim, „Modeling of Violin Playing Robot Arm with MATLAB/SIMULINK“, „Electronics and Radio Engineering“ departamentas, „School of Power Engineering“, „Mongolian University of Science and Technology “, liepos 9d. 2016 m.

[Met20]. Lucas A.E Pineda Metz, „An Evaluation of Unity ML-Agents Toolkit for Learning Boss Strategies“, „School of Technology, Department of Computer Science at Reykjavík University”, rugsėjis, 2020 m.

[Joe19]. Joel, „Robot Arm“ roboto rankos 3D modelis, birželio 12 d. 2019 m. Prieiga per Internetą: <https://sketchfab.com/3d-models/robot-arm-22d9367c8d2f4457b3a4e74193e86ac9>.

[WXL+17]. Zeng Wei, Jun Xu, Yanyan Lan, Jiafeng Guo, Xueqi Cheng, „Reinforcement Learning to Rank with Markov Decision Process“, „Institute of Computing Technology, Chinese Academy of Sciences“, rugpjūtis, 2017.

[Kan20]. Raju Kandasamy., „How to train your Robot Arm?”, rugpjūčio 9 d., 2020. Prieiga per Internetą: <https://medium.com/xrpractices/how-to-train-your-robot-arm-fbf5dcd807e1>.

[RFF+08]. Emmanuel Rachelson, Patrick Fabiani, Frédérick Garcia, and Gauthier Quesnel „A Simulation-based Approach for Solving Generalized Semi-Markov Decision Processes“, sausis, 2008 .

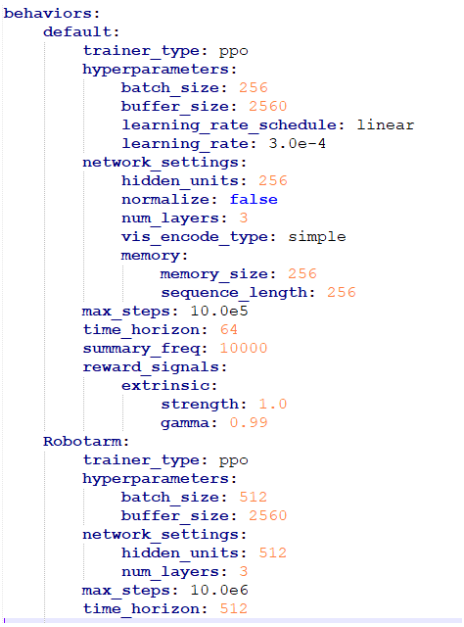
20. Priedai.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PenaltyCollidersBox : MonoBehaviour
{
    public RobotControllerAgent parentAgent;

    private void OnTriggerEnter(Collider other)
    {
        if (parentAgent != null)
        {
            if (other.transform.CompareTag("RobotInternal"))
            {
                parentAgent.ObstacleHitPenalty();
            }
            else
            {
                parentAgent.BoxNotHitReward();
            }
        }

    }
}
```

Pav. 41, PenkaltyColiderBox klasė.

```
public void BoxNotHitReward()
{
    AddReward(0.5f);
}

public void ObstacleHitPenalty()
{
    AddReward(-0.0005f);
    EndEpisode();
}
```

Pav. 42, BoxNotHitReward() ir ObstacleHitPenalty() metodai.



```yaml
behaviors:
    default:
        trainer_type: ppo
        hyperparameters:
            batch_size: 256
            buffer_size: 2560
            learning_rate_schedule: linear
            learning_rate: 3.0e-4
        network_settings:
            hidden_units: 256
            normalize: false
            num_layers: 3
            vis_encode_type: simple
            memory:
                memory_size: 256
                sequence_length: 256
        max_steps: 10.0e5
        time_horizon: 64
        summary_freq: 10000
        reward_signals:
            extrinsic:
                strength: 1.0
                gamma: 0.99
    Robotarm:
        trainer_type: ppo
        hyperparameters:
            batch_size: 512
            buffer_size: 2560
        network_settings:
            hidden_units: 512
            num_layers: 3
        max_steps: 10.0e6
        time_horizon: 512
```

Pav. 44, modelio konfigūracija.

```
public override void OnEpisodeBegin()
{
    if(trainingMode)
        ResetAllAxis();
    startTime = Time.time;
    MoveToSafeRandomPosition();
    UpdateNearestComponent();
}
```

Pav. 47, OnEpisodeBegin() metodas.

```
public void UpdateNearestComponent()
{
   if (trainingMode)
   {
     inFrontOfComponent = UnityEngine.Random.value > 0.5f;
   }
   if(!inFrontOfComponent)
     {
         nearestComponent.transform.position = transform.position + new Vector3(Random.Range(0.5f, 0.8f), Random.Range(0.3f, 0.4f), Random.Range(0.5f, 0.8f));
     }
     else
     {
         nearestComponent.transform.position = endEffector.transform.TransformPoint(Vector3.zero) + new Vector3(Random.Range(0.05f,0.18f),Random.Range(0.05f,0.18f), Random.Range(0.05f,0.18f));
     }
   beginDistance = Vector3.Distance(endEffector.transform.TransformPoint(Vector3.zero), nearestComponent.transform.position);
   prevBest = beginDistance;

   baseAngle = Mathf.Atan2( transform.position.x - nearestComponent.transform.position.x, transform.position.z - nearestComponent.transform.position.z) * Mathf.Rad2Deg;
   if (baseAngle < 0) baseAngle = baseAngle + 360f;
}
```

Pav. 48, UpdatenearestComponent() metodas.

```
public override void OnActionReceived(float[] vectorAction)
{
   angles = vectorAction;
   if (trainingMode)
   {
      armAxes[0].transform.localRotation =
         Quaternion.AngleAxis(angles[0] * 180f, armAxes[0].GetComponent<Axis>().rotationAxis);
      armAxes[1].transform.localRotation =
         Quaternion.AngleAxis(angles[1] * 90f, armAxes[1].GetComponent<Axis>().rotationAxis);
      armAxes[2].transform.localRotation =
         Quaternion.AngleAxis(angles[2] * 80f, armAxes[2].GetComponent<Axis>().rotationAxis);
      armAxes[3].transform.localRotation =
         Quaternion.AngleAxis(angles[3] * 80f, armAxes[3].GetComponent<Axis>().rotationAxis);
      armAxes[4].transform.localRotation =
         Quaternion.AngleAxis(angles[4] * 80f, armAxes[4].GetComponent<Axis>().rotationAxis);
      armAxes[5].transform.localRotation =
         Quaternion.AngleAxis(angles[5] * 80f, armAxes[5].GetComponent<Axis>().rotationAxis);
      armAxes[6].transform.localRotation =
         Quaternion.AngleAxis(angles[6] * 80f, armAxes[6].GetComponent<Axis>().rotationAxis);
      armAxes[7].transform.localRotation =
         Quaternion.AngleAxis(angles[7] * 90f, armAxes[7].GetComponent<Axis>().rotationAxis);


      float distance = Vector3.Distance(endEffector.transform.TransformPoint(Vector3.zero),
         nearestComponent.transform.position);
      float diff = beginDistance - distance;

      if (distance > prevBest)
      {
         AddReward(prevBest - distance);
      }
      else
      {
         AddReward(diff);
         prevBest = distance;
      }
      AddReward(stepPenalty);
   }
}
```

Pav. 49, OnActionReceived metodas.

```
public void JackpotReward(Collider other)
{
    if (other.transform.CompareTag("Components"))
    {
        float SuccessReward = 1.0f;
        float bonus = Mathf.Clamp01(Vector3.Dot(nearestComponent.transform.up.normalized,
                          endEffector.transform.up.normalized));
        float reward = SuccessReward + bonus;
        absoluteSumOfAngles = 0;

        absoluteSumOfAngles = absoluteSumOfAngles + armAxes[2].transform.localEulerAngles.z;
        absoluteSumOfAngles = absoluteSumOfAngles + armAxes[3].transform.localEulerAngles.z;
        absoluteSumOfAngles = absoluteSumOfAngles + armAxes[4].transform.localEulerAngles.z;
        absoluteSumOfAngles = absoluteSumOfAngles + armAxes[5].transform.localEulerAngles.z;
        absoluteSumOfAngles = absoluteSumOfAngles + armAxes[6].transform.localEulerAngles.z;
        absoluteSumOfAngles = absoluteSumOfAngles + armAxes[7].transform.localEulerAngles.z;

        AddBonusOrPenaltyOnAngleAbsSize(Math.Abs(absoluteSumOfAngles));

        AddTimeReward(reward);

        if (float.IsInfinity(reward) || float.IsNaN(reward)) return;
         Debug.LogWarning("Great! Component reached. Positive reward:" + reward );
        AddReward(reward);
        //EndEpisode();
        UpdateNearestComponent();
    }
}
```

Pav. 50, JackPotReward metodas.

```
private void AddBonusOrPenaltyOnAngleAbsSize(float absolute)
{
    if (0.0f < absolute && absolute < 100.0f)
    {
        AddReward(0.5f);
    }
    else if (100.0f <= absolute && absolute < 200.0f)
    {
        AddReward(0.4f);
    }
    else if (200.0f <= absolute && absolute < 300.0f)
    {
        AddReward(0.3f);
    }
    else if (300.0f <= absolute && absolute < 400.0f)
    {
        AddReward(0.2f);
    }
    else if (400.0f <= absolute && absolute < 500.0f)
    {
        AddReward(0.1f);
    }
    else if (500.0f <= absolute && absolute < 600.0f)
    {
        AddReward(0.0f);
    }
    else if (600.0f <= absolute && absolute < 700.0f)
    {
        AddReward(-0.1f);
    }
    else if (700.0f <= absolute && absolute < 800.0f)
    {
        AddReward(-0.2f);
    }
    else if (800.0f <= absolute && absolute < 900.0f)
    {
        AddReward(-0.3f);
    }
    else if (900.0f <= absolute && absolute < 1000.0f)
    {
        AddReward(-0.4f);
    }
    else
    {
        AddReward(-0.5f);
    }
}
```

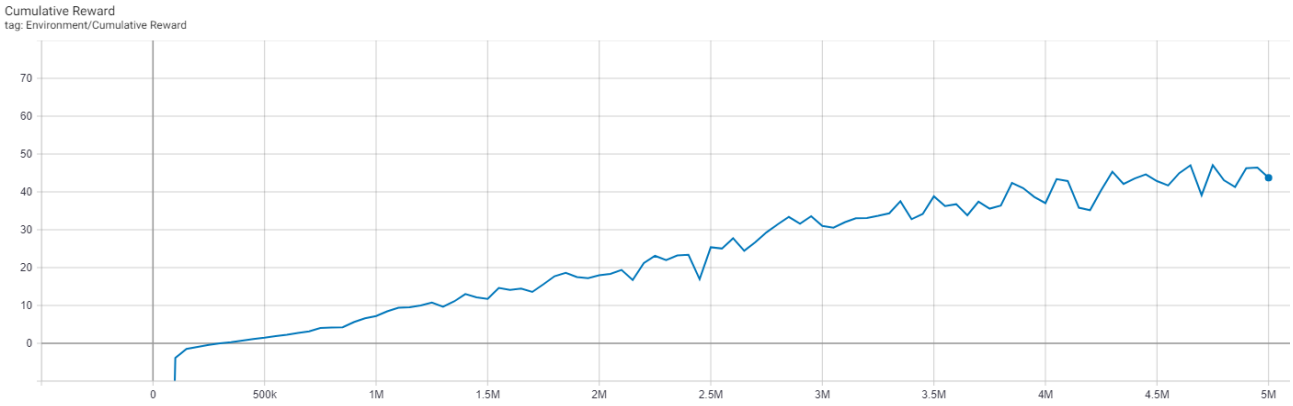
Pav. 51, AddBonusOrPenaltyOnAngleAbsSize metodas.

```
private void AddTimeReward()
{
    //time
    float timeSpent = Time.time - startTime;

    if (timeSpent < bestTime)
    {
        Debug.LogWarning("BETTER TIME");
        reward = reward + 0.5f;
    }
    else
    {
        reward = reward - 0.5f;
    }
}
```

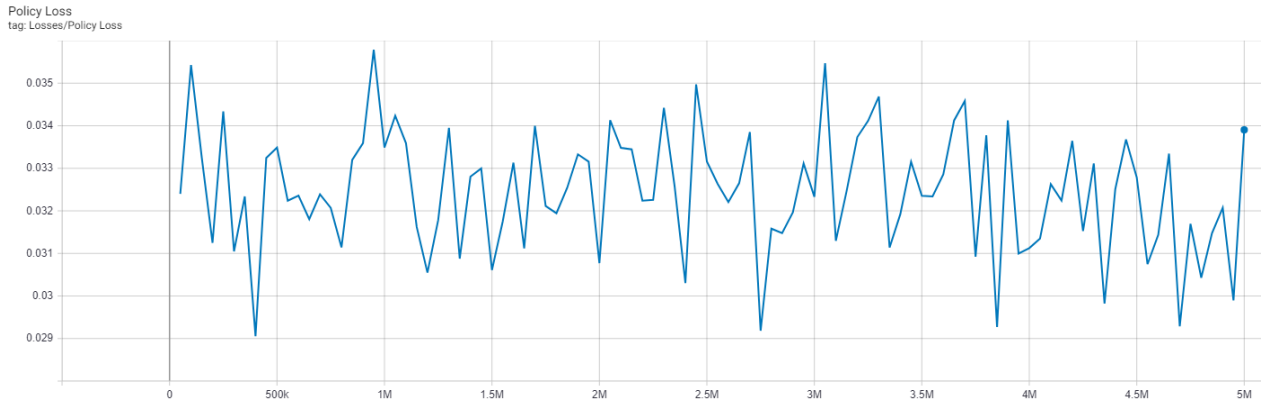
Pav. 52, AddTimeReward() metodas.

```
public class NearestComponentCollision : MonoBehaviou
{
    public RobotControllerAgent parentAgent;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }
    void OnCollisionEnter(Collision col)
    {
        if (col.gameObject.name == "Obstacle1" ||
            col.gameObject.name == "Obstacle2" ||
            col.gameObject.name == "Obstacle3" ||
            col.gameObject.name == "Obstacle4")
        {
            Debug.Log("Collision");
            parentAgent.UpdateNearestComponent();
        }
    }
}
```

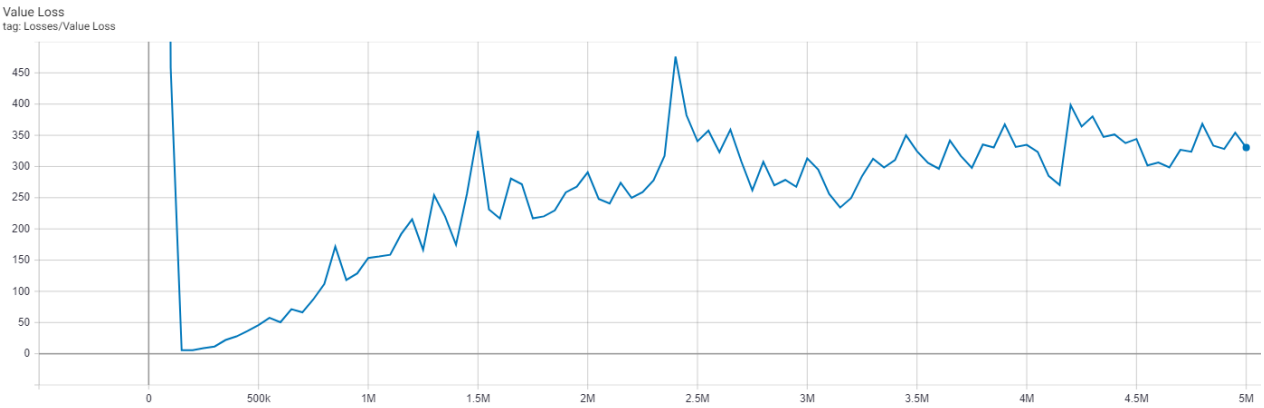
Pav. 53, NearestComponentCollision klasė.



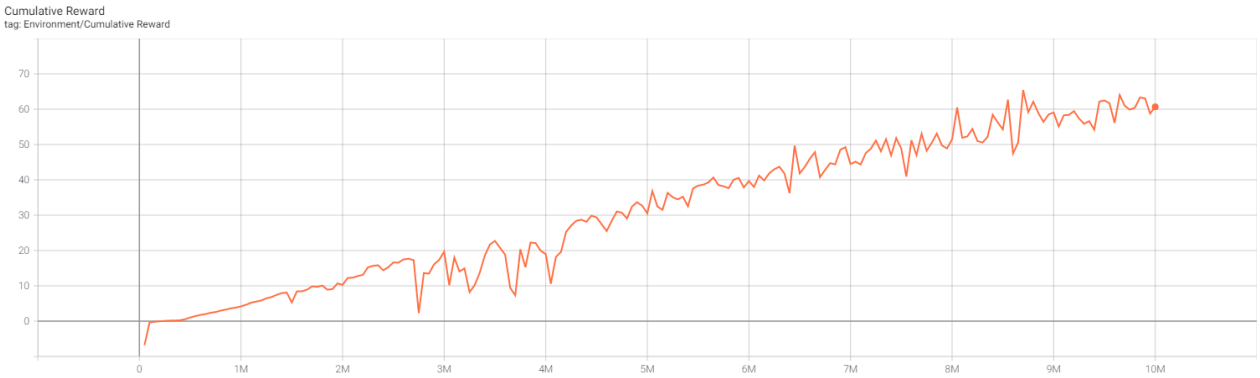
Pav. 54. Atlygio funkcija.



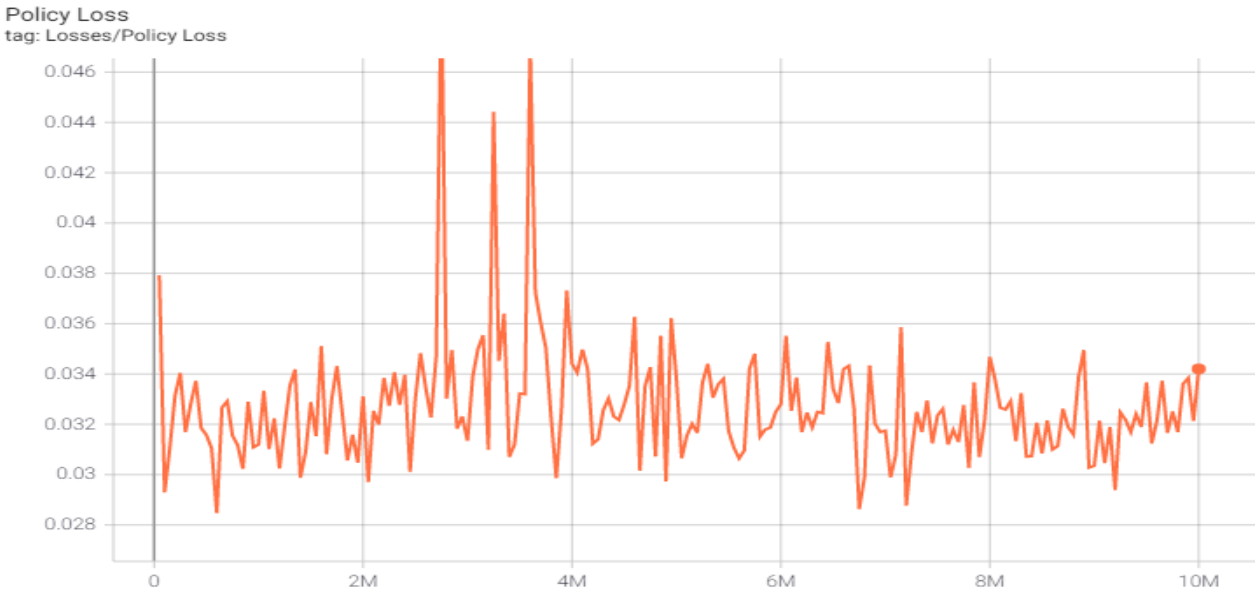
Pav. 55. Policy Loss funkcija.



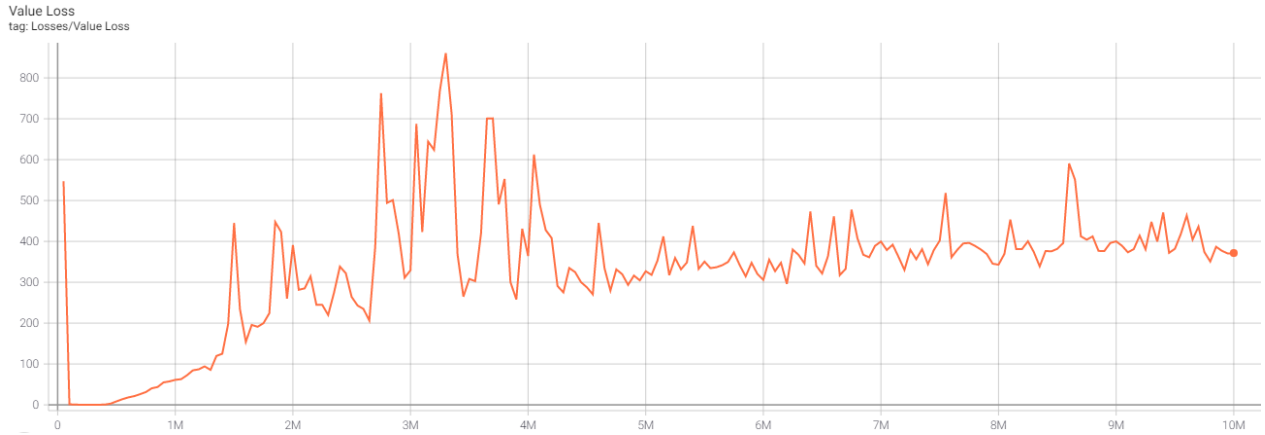
Pav. 56. Value Loss funkcija.



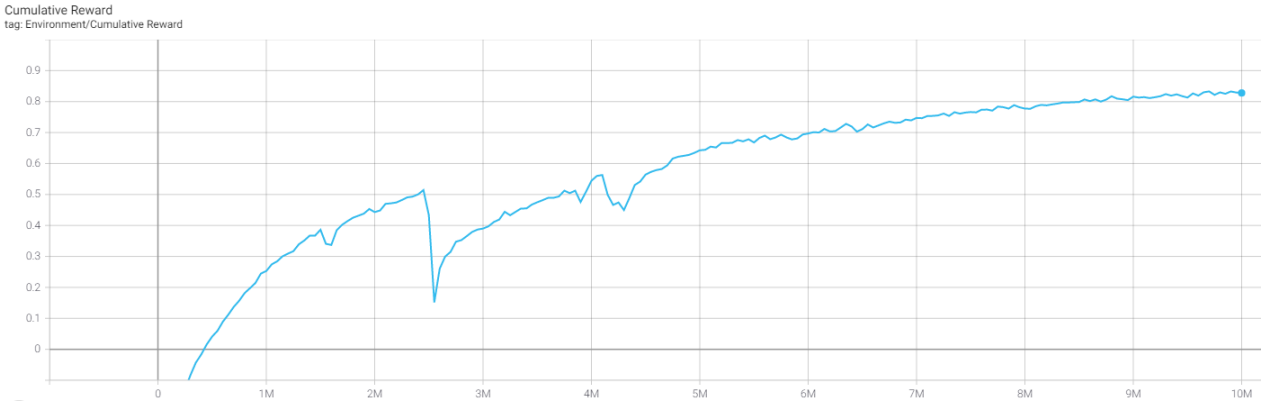
Pav. 57, Atlygis/Iteracijų skaičius.



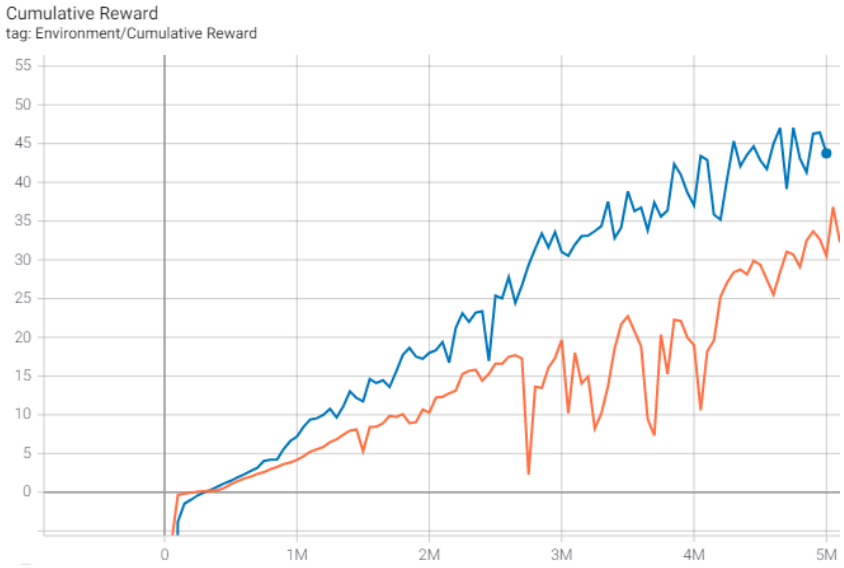
Pav. 58, Policy Loss funkcija.



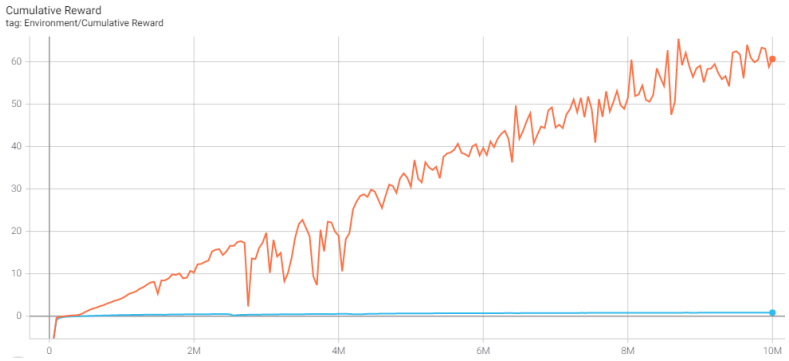
Pav. 59, Value Loss funkcija.



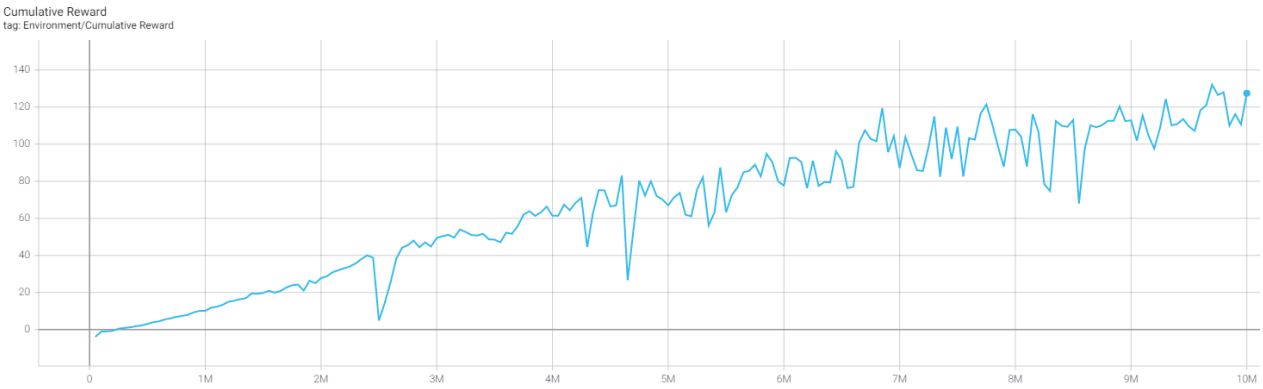
Pav. 60, Surinktas atlygis.



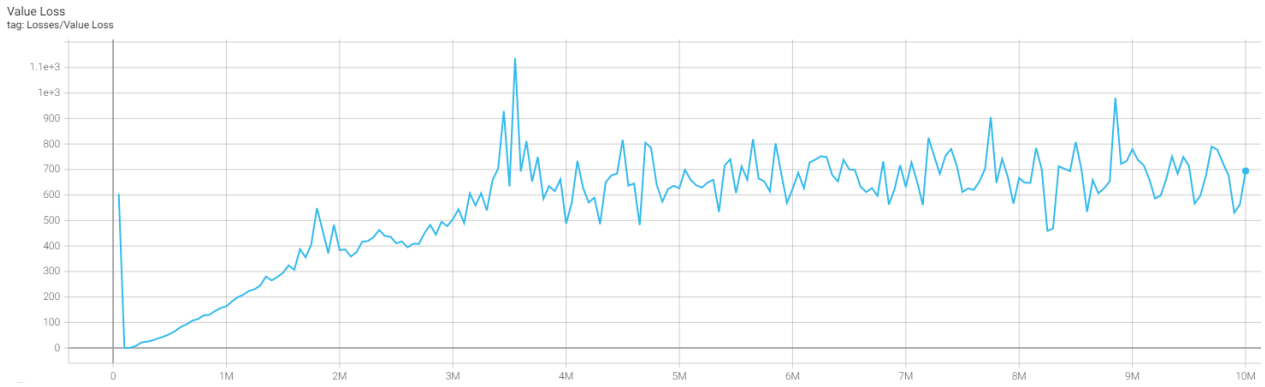
Pav. 61,. Dviejų mokymų palyginimas. Mėlyna – be kliūčių, oranžinė – su kliūtimis.



Pav. 62, Oranžinė – ranka su kliūtimis, mėlyna – atimama bauda – absoliutus atstumas tarp tikslo ir galinio efektoriaus.



Pav. 63, Atlygio/iteracijų skaičius funkcija. 8 Laisvės mazgai su kliūtimis.



Pav. 64, Value loss funkcija. 8 Laisvės mazgai su kliūtimis.

```
private void calculateGivenTime()
{
    if (givenTime <= 9.0f && increase == true)
    {
        givenTime = givenTime + 1.0f;
        if (givenTime >= 10.0f)
        {
            increase = false;
        }
    }
    else if (givenTime >= 10.0f && increase == false)
    {
        givenTime = givenTime - 1.0f;
        if (givenTime <= 5.0f)
        {
            increase = true;
        }
    } else
    {
        givenTime = 5.0f;
        increase = true;
    }
}
```

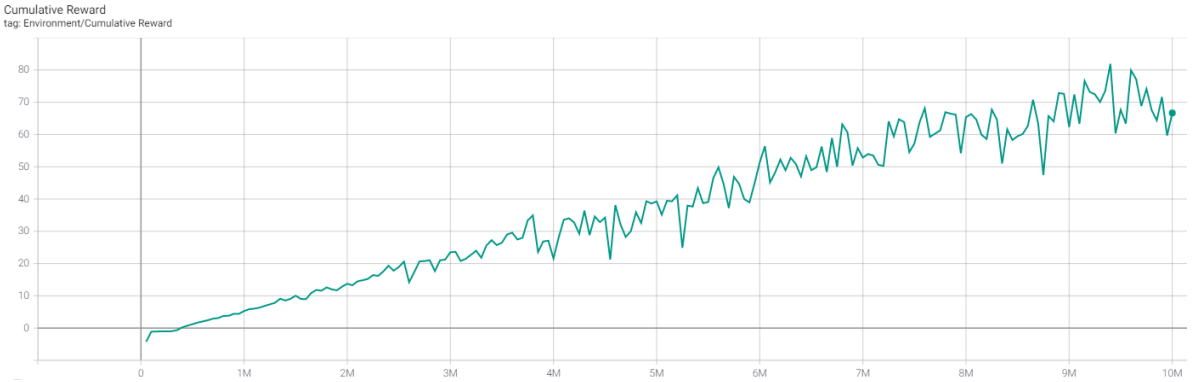
Pav. 65, CalculateGivenTime metodas.

```
if (Time.deltaTime - timer > givenTime)
{
    TimePenalty();
}
```

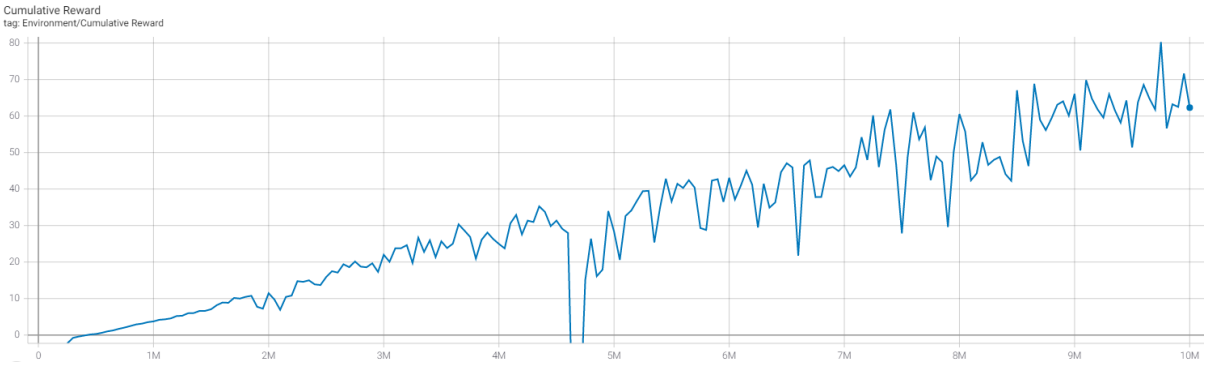
Pav. 66, praleisto laiko epizode patikrinimas.

```
private void TimePenalty()
{
    AddReward(-1f);
    Debug.LogWarning("Time penalty");
    EndEpisode();
}
```

Pav. 67, TimePenalty metodas.



Pav. 68, Atlygio/iteracijų skaičius funkcija. 8 Laisvės mazgai su kliūtimis (SMDP).



Pav. 69, Atlygio/iteracijų skaičius funkcija. 6 Laisvės mazgai su kliūtimis (MDP).