

LATVIJAS UNIVERSITĀTE  
DATORIKAS FAKULTĀTE

**ATDARINOŠĀS MAŠĪNMĀCĪŠANĀS PIELIETOJUMS  
ROBOTIKĀ**

MAGISTRA DARBS

Autors: **Pēteris Račinskis**

Stud. apl. Nr. pr20015

Darba vadītājs: Dr. sc. comp. Modris Greitāns

RĪGA 2022

## ANOTĀCIJA

## **ABSTRACT**

## AUTOREFERĀTS

# Saturs

IEVADS . . . . .	7
Darba mērķis un struktūra . . . . .	8
Terminoloģijas tulkojumi . . . . .	8
Apzīmējumi un saīsinājumi . . . . .	9
Motivējošais uzdevums . . . . .	10
<b>1. VISPĀRĪGI TEORĒTISKIE PAMATI . . . . .</b>	<b>11</b>
1.1 Parametriski modeļi, šabloni . . . . .	11
1.1.1 Neironu tīkli . . . . .	13
1.1.2 Laikrindu modelēšana, autoregresori, rekurentie neironu tīkli . . . . .	13
1.2 Markova lēmumu procesi . . . . .	15
1.3 Stimulētā mašīnmācīšanās . . . . .	16
1.4 Ieskats industriālo rbotu darbībā . . . . .	17
1.4.1 Kinemātika . . . . .	17
1.4.2 Trajektoriju plānošana . . . . .	21
<b>2. ATDARINOŠĀ MAŠĪNMĀCĪŠANĀS . . . . .</b>	<b>22</b>
2.1 Labi definētu trajektoriju kopēšana . . . . .	22
2.1.1 Vienkāršas metodes . . . . .	23
2.1.2 Statistiskas korekcijas . . . . .	25
2.1.3 Inversā stimulētā mācīšanās . . . . .	25
2.1.4 Generatīvie pretinieku tīkli . . . . .	26
2.1.5 Uzdevumu simboliska dekompozīcija . . . . .	27
2.2 Novērojumu iegūšana, interpretācija, papildināšana . . . . .	28
2.2.1 Nezināmas darbības . . . . .	28
2.2.2 Dinamikas novērojumu iegūšana, izmantošana, vispārināšana . . . . .	29
2.2.3 Demonstrācijas no cilvēka darbībām . . . . .	30
2.2.4 Video demonstrācijas, perspektīvu pārbīde . . . . .	31
2.2.5 Datu sintēze, telpiski modeļi . . . . .	31
2.3 Atdarinošu modeļu vispārināšana un adaptācija . . . . .	32
2.3.1 Neoptimālu demonstrāciju uzlabošana . . . . .	33
2.3.2 Demonstrācija – sākumpunkts apmācību procesam . . . . .	33
2.3.3 Tūlītēja trajektoriju atdarināšana . . . . .	34
2.3.4 Nestrukturētas demonstrācijas, plānu veidošana no galamērķiem . .	35
2.3.5 ļoti lielu, vispārināmu laikrindu regresoru pielietošana . . . . .	36
<b>3. PRAKTISKĀ REALIZĀCIJA . . . . .</b>	<b>38</b>
3.1 Izmantotie rīki, aprīkojums un platformas . . . . .	38
3.1.1 ROS . . . . .	38
3.1.2 Optitrack – aprīkojums un programmatūra . . . . .	39
3.1.3 Programmēšanas valoda, bibliotēkas . . . . .	40

3.2	Uzdevuma risinājuma pieeja . . . . .	41
3.3	Datu priekšapstrāde . . . . .	44
3.3.1	Ierakstu veikšana, sākuma datu kopas ieguve . . . . .	44
3.3.2	Laika ass reparametrizācija . . . . .	45
3.3.3	Atsevišķu demonstrāciju atdališana . . . . .	45
3.3.4	Trajektoriju gludināšana . . . . .	46
3.3.5	Kritisko punktu noteikšana . . . . .	47
3.3.6	Brīvā kritiena ekstrapolācija, mērķa koordinātu noteikšana . . . . .	48
3.3.7	Sākuma koordinātu kompensācija . . . . .	49
3.3.8	Apvienošana, stāvokļu pāreju veidošana, secības jaukšana . . . . .	50
3.4	Modeļu realizācija . . . . .	50
3.4.1	Klasiskais neironu tīkls . . . . .	51
3.4.2	Rekurentais neironu tīkls . . . . .	53
3.4.3	GAN (tālāka darbība) . . . . .	54
3.5	Validācijas datu ģenerēšana, simulācija, vizualizācija . . . . .	56
3.5.1	Modeļu pārbaude ar gadījuma sākuma stāvokļiem . . . . .	56
3.5.2	Validācija – demonstrāciju un autoregresoru salīdzinājumi . . . . .	58
3.5.3	Trajektoriju telpiska vizualizācija . . . . .	58
3.6	Trajektoriju izpilde uz robota . . . . .	60
3.6.1	Modeļa savietošana ar kontrolleri . . . . .	60
3.6.2	Robota trajektorijas plānošana un izpilde . . . . .	62
3.6.3	Asinhrona satvērējmehānisma vadība, ātrdarbība . . . . .	63
<b>4.</b>	<b>REZULTĀTI . . . . .</b>	<b>65</b>
4.1	Novērtējumi, to aprēķins . . . . .	65
4.1.1	Datu kopu tuvības mēri . . . . .	65
4.1.2	Vidējās novirzes laika soļa ietvaros . . . . .	66
4.1.3	Metiena parametru aplēšana . . . . .	67
4.1.4	Grafiku ģenerēšana un saturs . . . . .	69
4.2	Sasniegtie rezultāti dažādām modeļu klasēm . . . . .	70
4.3	Klasiskie neironu tīkli . . . . .	71
4.4	Rekurentie neironu tīkli . . . . .	73
<b>SECINĀJUMI . . . . .</b>	<b>75</b>	
<b>ATSAUCES . . . . .</b>	<b>76</b>	

## IEVADS

Mašīnmācīšanās šobrīd tiek plaši uzskatīta par vienu no aktuālākajām datorzinātņu pētniecības nozarēm [17]. Pēdējās desmitgades laikā šī pētniecības lauciņa popularitāte ir daudzkarīt pieaugusi, pateicoties galvenokārt diviem faktoriem: ļoti vispārīgiem neironu tīklu modeļiem un skaitlošanas resursu veikspējai, kas beidzot ļāvusi šos, teorētiski jau pagājušajā gadsimtā [38, 34, 18] iedomātos, mākslīgā intelekta uzbūves elementus realizēt praksē. Starp uzdevumiem, kurus izdevies realizēt, aizstājot mēginājumus rast analītiskus risinājumus ar pietiekami jaudīgiem neironu tīklu modeļiem, ir semantiskas nozīmes meklēšana attēlos [32], tekstu korpusu analīze un generēšana ar "izpratni" par to saturu [76] un visspejīgāko spēlētāju pārspēšana nepilnīgas informācijas spēlēs ar neap-tverami milzīgiem iespējamo stāvokļu permutāciju skaitiem [63].

Nav arī īpaši grūti atrast vēsturisko saikni starp mākslīgo intelektu un robotiku. Tautas iztēlē termins "robots" drīzāk droši vien iezīmēs zinātniskās fantastikas radītos personāžus – mehāniskas būtnes, kas spēj patstāvīgi darboties neierobežotā vidē un risināt sarežģītus uzdevumus – nevis pieticīgākus, reāli pastāvošus un ražotnēs rodamus industriālos robotus. Šī pati zinātniskā fantastika radījusi arī nesaraujamu saiti starp robotiem un mākslīgo intelektu [4]. Neformālas diskusijas par mākslīgo intelektu bieži vien plūstoši pāriet diskusijās par ar šādu intelektu aprīkotiem robotiem, un, neizbēgami, dažnedažādiem šo intelektuālo sistēmu radītiem draudiem mūsu sabiedrībai. Protams, zinātne ne vienmēr seko populārzinātniskās iedomas lidojumam, taču šāda saikne ir visnotaļ pamatota – spēja mācīties no paraugiem vai patstāvīgi, pielāgoties savai apkārtnei ir ārkārtīgi noderīga. Daudzi uzdevumi, kuru risināšanai varētu pielietot robotus, ir sarežģīti nevis to fizikālajā izpildē, bet tiesi vadības uzdevuma formulēšanā un realizācijā – tai to ilustrētu pietiek vien aplūkot kādu nejauši izvēlētu sarakstu ar robotikā aktuālām problēmām [13].

Atdarinošā mašīnmācīšanās (*imitation learning*) ir viens no paņēmieniem, ar kuriem tiek mēgināts risināt šādas sarežģītas vadības problēmas. Lai gan pamatu pamatos nevar apgalvot, ka tā ir tikai robotikai piemērota metožu saime, lielākā daļa izpētes virzīta tiesi šajā virzienā [5]. Problēmas tiek formulētas kā fizikālu (vai nosacīti fizikālu – virtuālās vidēs simulētu) procesu kontroles uzdevumi, un risinājumi tiek rasti no pēc iespējas mazāka skaita veiksmīgas darbības piemēru [5]. Mašīnmācīšanās nozarē biologiskas analogijas un iedvesma nav nekāds retums, un savā ziņā šāda mācīšanās atspoguļo vienu no izplatītiem paņēmieniem, kā cilvēki vai sabiedriski dzīvnieki nodod prasmes viens otram - demonstrējot. Nevar nepieminēt, ka izpēte šajā jomā bieži aizņemas pieejas un iespaidojas no rezultātiem, kas gūti ar stimulēto mašīnmācīšanos (*reinforcement learning*) - savā ziņā vispārīgu, pašmācībai un treniņam analogisku paņēmienu. Arī abu metožu apvienojums ir ideja, kas pavīd visai regulāri – cerībā, ka, atdarinot ekspertus, var ātrāk nonākt pie derīgām stratēģijām, kas var kalpot kā sākumpunkts dzīlākai pašmācībai [26]. Visbeidzot, pavisam nesen sasniegti ļoti pārliecinoši rezultāti ar teksta un cita veida virķu regresijai paredzētiem modeļiem arī atdarināšanas jomā [55]. Tas liecina, ka varbūt tieši laikrindu modeļi ir atslēga veiksmīgai apmācībai ar paraugdemonstrācijām.

## Darba mērķis un struktūra

Magistra darba tēma izvēlēta ar Elektronikas un datorzinātņu institūta personāla palīdzību, meklējot šobrīd aktuālu pētniecības virzienu. Izvērtējot dažādus pieejamos variantus, izvēlēts pētniecības virziens – atdarinošā mašīnmācīšanās – un konkrēts motivējošais uzdevums – atkritumu šķirošanas līnijas robots, kas izmanto mešanas kustības. Darba izstrāde attiecīgi notikusi institūta Robotikas un mašīnuzterveres laboratorijā. Tur paralēli noris cita, radniecīga pētnieciska darbība – gan atkritumu šķirošanas procesu automatizācijā, gan stimulētās mācīšanās metodēs.

Šī darba mērķis ir noskaidrot, vai atdarinošās mašīnmācīšanās metodes ir piemērotas motivējošās problēmas risināšanai, izvērtēt dažādu paņēmienu un aktuālāko rezultātu izmantošanas iespējas un izstrādāt prototipu procesam, kura gaitā tiek iegūtas demonstrācijas un apmācīti modeļi, kas izpilda doto uzdevumu. Attiecīgi, magistra darba pamatdaļa sastāv no četrām nodaļām:

- 1) vispārīga nepieciešamo teorētisko pamatu izklāsta, kur īsi apskatīti parametrisku modeļu, lēmuma procesu, robotikas matemātiskie formālismi;
- 2) atdarinošās mašīnmācīšanās pārskata, kur apkopota nozares zinātniskā literatūra, izdalīti galvenie pētniecības virzieni un aprakstīti to sasniegumi;
- 3) praktiskās realizācijas atskaites, kur aprakstīti darba gaitā izstrādātie rīki;
- 4) rezultātu analīzes, kur salīdzināti dažādi apmācītie modeļi.

## Terminoloģijas tulkojumi

Viena no īpatnībām, ar ko ir nācies saskarties, strādājot tieši ar mašīnmācīšanās nozari, ir viennozīmīgas terminoloģijas trūkums latviešu valodā. Pati zinātnes nozare, lai arī nebūt ne tik jauna kopumā, piedzīvojusi milzīgas izmaiņas un nepieredzētu uzplaukumu pēdējās desmitgades laikā. Protams, datorzinātnes laukā pirmā un galvenā saziņas valoda ir angļu. Attiecīgi novērojami divējādi un saistīti fenomeni - publikācijas un terminoloģija, kas radītas senāk, veidojušas dzīļi specifisku nišu, kas nav iedvesmojusi daudz mēginājumu tulkot to uz citām valodām, savukārt uzplaukuma laikos vēl ir ļoti daudz materiāla, ko vienkārši neviens nav paguvis iztulkot.

Patvalīgi izvēloties tulkojumu, pastāv risks mulsināt lasītāju un sadrumstalot jau tā nelielo literatūras kopu dažādu atslēgas vārdu izvēles rezultātā. Kur vien iespējams, ieteicams izmantot oficiālā terminu datubāzē pieejamus tulkojumus, taču ne vienmēr tādi ir pieejami. Tāpēc šeit izveidots saraksts ar potenciāli mulsinoši tulkoto terminoloģiju tās oriģinālajā formulējumā angļu valodā, izvēlētajiem tulkojumiem un īsiem pamatojumiem.

- 1) **policy – stratēģija.** Apraksta funkciju, kas novērojumus attēlo lēmumu telpā. Šis tulkojums varētu būt strīdīgs. Angļu valodā pastāv divi termini, *policy* un *politics*, kas parasti latviski tiek tulkoti vienādi – politika – par spīti radikāli atšķirīgām nozīmēm. Terms *strategy* tiek lietots kā sinonīms pirmajam abās valodās, un arī piemērojams tieši šādām lēmumu pieņemšanas funkcijām, piemēram, spēļu teorijā. Kad stratēģija tiek izmantota kāda procesa vadībā, to mēdz saukt par **agentu**.

- 2) *reinforcement learning* – **stimulētā mašīnmācīšanās**. Meklējumi tiešsaistē atklāj [51], ka šis tulkojums jau ir apstiprināts standartā, taču varētu būt nezināms lasītājiem, kas ar to sastopas pirmo reizi – pat ja zināms metodes anglikais nosaukums.
- 3) *imitation learning* – **atdarinošā mašīnmācīšanās**. Paša autora piedāvāts tulkojums, izmantojot iepriekšējo kā piemēru, jo nav izdevies atrast alternatīvas. Latviskais vārds ”atdarināt” izvēlēts pār internacionālismu ”imitēt”, jo to vieglāk izlocīt formā, kas neizklausās lauzīta un neveikla. Taču procesā zūd spēja viegli atrast sākotnējo vārdu svešvalodā, kas ļoti svarīga zinātniskajā vidē, kurā latviski pieejamo resursu ir maz.
- 4) *reward function* – **atalgojuma funkcija**. Oficiālā terminu datubāzē tulkojuma šim terminam nav. Tuvākie tulkojumi lietojumam, ar kādu šis termins parasti sastopams tekstos angļu valodā, būtu ”lietderība” vai ”atdeve”, taču ”atalgojums” labi ietver sevī faktu, ka šo funkciju paredzēts maksimizēt.
- 5) *generative adversarial network* – **ģeneratīvie pretinieku tīkli**. Šim terminam arī pastāv vairāki konkurējoši tulkojumi, un terminu datubāzē skaidru atbildi nevar rast. Taču ir atrodams jau 2019. gadā sagatavotais kolēges Lianas Blumbergas kursa darba plakāts, kur šis tulkojums ir lietots [8], tāpēc tas pats tiks lietots arī šeit.

## Apzīmējumi un saīsinājumi

Tekstā var bez paskaidrojuma tikt izmantoti šādi saīsinājumi:

- 1) EDI – Elektronikas un datorzinātņu institūts
- 2) GAN – generatīvais pretinieku tīkls (*Generative Adversarial Network*)
- 3) IRL – inversā stimulētā mašīnmācīšanās (*Inverse Reinforcement Learning*)
- 4) LIDAR – attāluma lāzermērišana
- 5) MDP – Markova lēmumu process (*Markov Decision Process*)
- 6) RL – stimulētā mašīnmācīšanās (*Reinforcement Learning*)
- 7) RNN – rekurentais neironu tīkls (*Recurrent Neural Network*)
- 8) VR – virtuālā realitāte

Biežāk izmantoti matemātiskie apzīmējumi:

- 1)  $\pi, \pi_\theta, \pi^*$  – stratēģija, stratēģija ar parametriem  $\theta$ , instruktora stratēģija
- 2)  $s$  – sistēmas stāvoklis vai tiešais novērojums
- 3)  $s_t, s'$  – laika solī  $t$ , nākamais  $s$
- 4)  $a, a_t$  – darbība, darbība laika solī  $t$
- 5)  $o_t$  – netiešais novērojums
- 6) trajektorija – laikrinda ar elementiem  $s_t, o_t, (s_t, a_t)$  vai  $(o_t, a_t)$ , ja nav norādīts
- 7)  $R(s), R(s, a)$  – atalgojuma funkcija
- 8)  $\theta, \phi, \psi$  – modeļu parametru vektori
- 9)  $\mathbb{E}$  – matemātiskā cerība
- 10)  $\mathbf{x}, x_i$  – vektors, tā elements
- 11)  $\mathcal{L}(y, \pi_\theta(x))$  – mērķa funkcija optimizācijā

## Motivējošais uzdevums

Nevar apgalvot, ka atdarinošā mašīnmācīšanās kā tēma šim kursa darbam un magistra darbam kopumā tikusi izvēlēta tīri inženiertehniskā procesa rezultātā – sākot ar problēmas definīciju un nonākot pie tai piemērotākā risinājuma, pārbaudot un atsijājot visas iespējamās piejas, priekšlaicīgi neieguldot pārāk daudz pūļu un resursu kādas konkrētas metodes pielietošanā. Daļēji orientē vēlme – gan no EDI, gan autora putas – izzināt jaunas kompetences un likt pamatu tālākiem pētījumiem. Tomēr vienu šobrīd aktuālu praktisku problēmu, kas šķiet piemērota tieši atdarinošās mašīnmācīšanās metodēm, var izteikt kā motivējošu uzdevumu.

Situāciju var aprakstīt sekojoši. Atkritumu šķirošanas līnijas ir cilvēkam nedraudzīga un nepatīkama darba vide. Šobrīd tiek aktīvi meklēti veidi, kā palielināt to automatizācijas pakāpi – gan izmaksu samazināšanas, gan darba drošības nolūkos [65]. Konkrēti aplūkojot plastmasas iepakojumu šķirošanu, jau izstrādātas metodes to klasifikācijai un satveršanai [65].

Viens no virzieniem, kurā varētu pilnveidot šo un citādu robustu, neregulāras formas objektu *pick-and-place* uzdevumu risinājumus, ir metienu iestrādāšana trajektorijās. Ja pārvietojamais objekts ir noturīgs pret trieciena slodzēm vai tā stāvoklis nav svarīgs, turklāt tā masa salīdzinot ar robota pašmasu ir neliela, potenciāli iespējams pārvietot objektu tālāk par robota maksimālā izstiepuma distanci. Turklāt paveras iespējas tieši šķirošanas uzdevumos, jo iespējams telpā brīvāk izvietot tvertnes, uz kurām objekti jānogādā – metiens var nogādāt objektu tālāk, nekā maksimāli izstiepies robots.

Protams, iespējams manuāli programmēt katru metienu vai veidot programmatūru, kas, balstoties uz fizikas likumiem, saplāno atbilstošu robota trajektoriju. Taču, mēginot manuāli programmēt katru metienu, rodas virkne problēmu – dažādi trajektoriju sākumpunkti, dažādi galamērķi un to iespējamā mainība, objektu neregularitāte, u.c. Savukārt parametrizējamas programmatūras izstrāde ir dārgs un laikielipīgs process, ko katram uzdevumam atkārtot būtu nepraktiski. Tā vietā pastāv cerība, ka var izstrādāt procesu, kura gaitā tiek apmācīts modelis no demonstrācijām. Šī darba ietvaros mērķis ir realizēt prototipu tādam procesam mešanas uzdevumā un gūt atziņas, kas noderēs šī un cita uzdevumu praktiskā automatizācijā ar atdarinošām metodēm.

# 1. VISPĀRĪGI TEORĒTISKIE PAMATI

Pētot zinātnisko literatūru viens no lielākajiem šķēršļiem lasītājam “no malas” ir katrā nozarē pieņemtais tehnisko priekšzināšanu kopums, ko autori sagaida no auditorijas. Tas, protams, ir logiski, jo publikācija, kas apraksta jaunākos atklājumus kādā dzīļi specifiskā lauciņā, nevar veltīt visu sev atvēlēto drukas apjomu elementāras un vispārzināmas terminoloģijas skaidrojumiem. Tāpat, tālāk literatūras analīzē iztirzājot šos rakstus vai aprakstot pašu veiktu praktisku darbību, noderīgi ir ieviest tiem kopīgus apzīmējumus un definēt visus vienuviet. Tāpēc šajā nodaļā tiek īsi aprakstīti jēdzieni, kas jāizprot, lai varētu orientēties šajā darbā patstāvīgi izmantotajā terminoloģijā.

## 1.1. Parametriski modeļi, šabloni

Viens no visplašāk izmantotajiem formālismiem datizraces un mašīnmācīšanās laukos ir parametriskais modelis. Pamatā tam ir ideja, ka nezināmu funkciju, kuras rezultātus vēlamies paredzēt, var aproksimēt ar citu funkciju jeb modeli:

$$M(x) \approx f(x) \quad (1.1)$$

Protams, šādu modeļu varētu būt bezgalīgi daudz, un tie visi var atšķirties pēc tā, cik labi spēj paredzēt nezināmās funkcijas vērtības. Tāpēc modeļu meklēšanai parasti izmanto šablonus - funkcijas, kuru argumentā papildus ievades datiem ir brīvi maināmi un kopīgi (tātad ”apmācāmi”) parametri  $\theta$ :

$$\text{Meklē } \theta : M(x|\theta) = M_\theta(x) \approx f(x) \quad (1.2)$$

Iegūtā šablona funkcijas un apmācīto parametru kombinācija  $\{M, \theta\}$  tad veido konkrētu modeli. Labs šablonis ir tāds, kas spēj pielāgoties ļoti daudzām dažādām funkcijām:

$$\forall f \forall x \exists \theta : M_\theta(x) \approx f(x) \quad (1.3)$$

Atkarībā no uzdevuma specifikas, izplatīti modeļi mēdz būt regresori, kas aproksimē funkcijas ar  $k$ -dimensionālām reālām (vai citādi skaitliskām) vērtībām,

$$f : x \rightarrow \mathbb{R}^k \quad (1.4)$$

$$M : x \times \theta \rightarrow \mathbb{R}^k \quad (1.5)$$

un klasifikatori, kas paredz ievades datu punkta piederību kādai diskrētai klasei

$$f : x \rightarrow C = \{c_1, c_2, \dots, c_m\} \quad (1.6)$$

$$M : x \times \theta \rightarrow C \quad (1.7)$$

Bieži vien noderīgi ir ne tikai spēt attēlot datu punktu kā diskrētu klasi, bet iegūt varbūtību sadalījumu, kas apraksta tā iespējamību piederēt jebkurai no klasēm:

$$M : x \times \theta \times c_i \rightarrow [0; 1] \quad (1.8)$$

$$M_\theta(x, c_i) = P_i \quad (1.9)$$

$$\sum_{i=1}^m P_i = 1 \quad (1.10)$$

Lai varētu novērtēt, cik labi modelis aproksimē nezināmo funkciju, un vadīt parametru apmācības procesu, tiek izmantotas mērķa funkcijas (*loss functions*) [3]:

$$\ell : M_\theta(x) \times f(x) \rightarrow \mathbb{R} \quad (1.11)$$

Strādājot ar reāliem datiem, datu punkti veido datu kopu, kas parasti tiek uzskatīta par gadījuma izlasi no punktus ģenerējošā varbūtību sadalījuma. Praktiskiem apmācības uzdevumiem datu kopa parasti jāiegūst formā, kas satur gan sagaidāmos ievades datus, gan pareizu rezultātu:

$$s \sim \mathcal{D} \Leftrightarrow s \text{ ir no varbūtību sadalījuma } \mathcal{D} \quad (1.12)$$

$$y_i = f(x_i) \quad (1.13)$$

$$s_i = (x_i, y_i) \quad (1.14)$$

$$S = \{s_1, s_2, \dots, s_n | s_i \sim \mathcal{D}\} \quad (1.15)$$

Datu kopai var aprēķināt empīrisku mērķa funkcijas novērtējumu,

$$L_S(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(M_\theta(x_i), y_i) \quad (1.16)$$

bet apmācības process parasti kādā veidā tiecas minimizēt šīs vērtības matemātisko cerību ģenerējošam sadalījumam (nevis tikai pašai datu kopai - ja modelis ļoti cieši pielāgots konkrētai datu izlasei bet zaudē precizitāti sadalījumam kopumā, to sauc par pārpielāgošanos – *overfitting*)

$$L_{\mathcal{D}}(\theta) = \mathbb{E}_{\mathcal{D}}[\ell(M_\theta(x_i), y_i)] \quad (1.17)$$

$$\text{Apmāca } M_\theta \text{ uz } \mathcal{D} \rightarrow \text{Minimizē } L_{\mathcal{D}}(\theta) \quad (1.18)$$

Ja modelis ir stratēģija (*policy*), stimulētās vai atdarinošās mašīnmācīšanās literatūrā to ļoti bieži izsaka kā  $\pi_\theta(x)$ . Mazliet mulsinoš ir tieši ar imitējošām metodēm saistītos rakstos lietotais apzīmējums  $\pi^*$ , ar ko apzīmē t.s. “ekspertu stratēģijas” – kas pašas ir nezināmās funkcijas, ko cenšamies aproksimēt pēc to ģenerēto punktu kopām.

### 1.1.1. Neironu tīkli

Neironu tīkli ir izplatīta modeļu šablonu saime, ko var izmantot dažādas formas funkciju aproksimēšanai – tie var būt gan klasifikatori, gan regresori. Neironu tīklu kopīgais elements ir t.s. perceptrons, kas izteikts jau pašos pirmsākumos [38]. Perceptronu funkcija, kas piemēro nelineāru aktivācijas funkciju  $\sigma$  argumentu vektora  $\mathbf{x}$  elementu savstarpējai lineārai kombinācijai, t.i,

$$f_{\text{perceptron}}(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) \quad (1.19)$$

, kur  $\mathbf{w}$  ir t.s. svaru vektors, bet  $b$  – nobīde. Perceptrona parametri tātad ir brīvie mainīgie  $\mathbf{w}$  un  $b$ . Neironu tīkls parasti sastāv no slāņiem – perceptronu  $f_i$  kopām, kas visi apstrādā to pašu argumentu vektoru, bet katrs ar saviem parametriem  $w_i, b_i$ . Tad slāni algebriski izsaka formā

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \dots \\ w_k^T \end{bmatrix}; \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_k \end{bmatrix}; \quad (1.20)$$

$$f_{\text{layer}}(\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b}) \quad (1.21)$$

Ja slānis tīklā ir pēdējais un tā vērtības ir modeļa izvadē, to sauc par izvades (*output*) slāni. Ievades datu vektoru sauc par ievades (*input*) slāni. Pārējos slāņus sauc par slēptajiem (*hidden layers*). Saka, ka slāni savā starpā pilnīgi savienoti (*fully connected*), ja katram viena slāņa perceptronam argumentā parādās visi iepriekšējā slāņa izvades elementi. Svarīga neironu tīkla īpašība – ja tā aktivācijas funkcijas ir diferencējamas, tad arī tīkls kopumā ir diferencējams pēc katras tā parametra, pat ar perceptroniem daudzos slāņos. Līdz ar to var izmantot atpakaļizplatīšanas algoritmu, kas atrod mērķa funkcijas parciālos atvasinājumus pēc modeļa parametriem un izmanto kādu gradientu optimizācijas metodi apmācībai.

Pastāv dažādas šo tīklu arhitektūras. Vienkāršākās sastāv no viena vai vairākiem slāņiem (neskaitot ievades slāni), taču ir plaši izplatīti arī, piemēram, konvolūciju neironu tīkli [32], ko izmanto attēlu apstrādē, tai skaitā šajā atskaitē aplūkotajos pētījumos, kur nepieciešams gūt informāciju no video datiem. Galvenā atšķirība konvolūcijā tīklā ir kodolu (*kernel*) izmantošana - konvolūciju slāņi vienā līmenī piemēro identiskas perceptrona funkcijas nelieliem iepriekšējā slāņa (matricas vai tenzora formā) reģioniem. Tas palīdz identificēt dažādas lokālas struktūras, piemēram, attēlā.

### 1.1.2. Laikrindu modelēšana, autoregresori, rekurentie neironu tīkli

Ja modelējamā parādība vai datu kopa izsakāma formā

$$(x_1, x_2, \dots) \quad (1.22)$$

, kur starp punktiem pastāv izteikta secības sakarība – piemēram, tie ir novērojumi laikā vai simboli teksta korpusā – šādam fenomenam var piemērot t.s. laikrindu datu kopu

analīzes metodes [80]. Šī darba ietvaros plaši tiek izmantots autoregresora jēdziens [64]. Principā ar šo terminu var apzīmēt jebkuru laikrindu regresijas metodi, kas balstīta uz pieņēmumu, ka nākotnes vērtības var modelēt kā funkciju no pagātnes vērtībām vai kādas to apakškopas, t.i.

$$x_t = \pi_\theta(x_{t-1}, x_{t-2}, \dots, x_1) \quad (1.23)$$

turklāt šo procesu var atkārtot ar paša modeļa ģenerētiem rezultātiem

$$x_{t+m} = \pi_\theta(x_{t+m-1}, x_{t+m-2}, \dots, x_1) \quad (1.24)$$

, kur  $x_t = \pi_\theta(x_{t-1}, x_{t-2}, \dots, x_1)$ ,  $x_{t+1} = \pi_\theta(x_t, x_{t-1}, \dots, x_1)$  u.t.t. Autoregresoru modeļus šādi var izmantot laikrindu ģenerēšanā, izmantojot zināmu sākuma stāvokli vai sākuma stāvokļu virkni. Principā ir iespējams jebkuru modeli ar pietiekami lielas dimensijas datu izvadu izmantot kā autoregresoru-ģeneratoru – ja tas ir formā

$$\pi_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^n \quad (1.25)$$

, tad šī darba ietvaros tiek sacīts, ka “modelis lietots autoregresijā”, ja iegūta sekojošās formas datu virkne

$$(s_1, \pi_\theta(s_1), \pi_\theta(\pi_\theta(s_1)), \dots) \quad (1.26)$$

Laikrindu modelešanas uzdevumos tradicionāli plaši izmantoti rekurentie neironu tīkli [2]. Īsumā, tos varētu definēt kā

$$y_t, h_t = \pi_\theta(x_t, h_{t-1}) \quad (1.27)$$

, kur  $y_t$  ir modeļa izpildes (nosacīti – laika) solī iegūtais izvads,  $x_t$  ir tajā pašā solī izmantotais ievaddatu vektors bet  $h_t$  – modeļa iekšējā stāvokļa vektors jeb tie elementi no modeļa izvaddatiem, kas nākamajā izpildes solī tiek izmantoti modeļa ievadā. Tātad pēc augstāk dotās autoregresijas modeļa definīcijas, rekurentais tīkls vienmēr vismaz daļēji strādā šādā režīmā – izmantojot kādu funkciju no iepriekšējo laika solu vērtībām pat tad, ja tas strādā t.s. “sequence-to-sequence” režīmā – kur jau pieejama katram solim atbilstoša ievaddatu vērtība. Protams, tāpat kā jebkura cita veida modeli, arī rekurento neironu tīklu var darbināt “tīrā” autoregresijā. Tad to aprakstošais vienādojums ir vienkārsī

$$y_t, h_t = \pi_\theta(y_{t-1}, h_{t-1}) \quad (1.28)$$

Šādu modeli iespējams apmācīt ar novērojumu virknēm  $(x_t, y_t)$ , bet pēc tam izmantot, lai ġenerētu jaunas virknes, par ievadiem nēmot jau paša modeļa paredzējumus. Viena problēma, ar ko saskaras rekurentie neironu tīkli praksē, ir t.s. izgaistošo gradientu problēma. Piemērojot atpakaļplatīšanas algoritmu rekurentam neironu tīklam, nepieciešams daudzas reizes virzīties caur visiem tā slāņiem. Līdz ar to nelineāro aktivācijas funkciju rezultātā mērķa funkcijas gradienti var sasniegt nulli vai neierobežoti

pieaugt, algoritmam virzoties tālāk no laika soļa, kurā piemērota mērķa funkcija. Tāpēc vienkāršas rekurento tīklu arhitektūras cieš no “īsas atmiņas”.

Divas populāras arhitektūras, kas piedāvātas ar mērķi apkarot šo fenomenu, ir “*Long Short-Term Memory*” (LSTM) [28] un “*Gated Recurrent Unit*” (GRU) [11], kas abas ievieš strukturētus un apmācāmus mehānismus, kas ļauj modelim “izvēlēties” – “atcerēties” vai “aizmirst” slēptā stāvokļa vektorā kodētu informāciju. Daudzos pētījumos to sasnieg tie rezultāti ir visnotaļ līdzīgi un pārspēj vienkāršāku arhitektūru modeļus [12], taču šķiet, ka pie nelielām datu kopām un garām apstrādājamām virknēm GRU sasniedz labākus rezultātus [79].

## 1.2. Markova lēmumu procesi

Pastāv dažādi formālismi procesu definēšanai vadības sistēmu izstrādes mērķiem, lai ar tiem varētu veikt matemātiskas operācijas. Izplatīti atdarinošās un stimulētās mašīnmācīšanās literatūrā ir Markova lēmumu procesi (MDP – *Markov decision processes*). Šis formālisms ir piemērojams situācijām, kurās, lai paredzētu procesa atribūtu vērtības pēc laika soļa  $t$ , netiek izmantota nekāda informācija par to vērtībām pagātnē – laika soļos  $t' < t$ . Dažādi autori, kas darbojas dažādos izpētes virzienos, mēdz piedāvāt atšķirīgus tā formulējumus, taču parasti tie ir ekvivalenti sekojošam [5]

$$MDP = (S, A, R, T, \gamma) \quad (1.29)$$

, kur  $S$  – sistēmas iespējamo stāvokļu  $s$  kopa;  $A$  – kontrolētajam procesam (“āgen-tam”) pieejamo darbību  $a$  kopa;  $R : S \times A \rightarrow \mathbb{R}$  vai  $R : S \rightarrow \mathbb{R}$  – atalgojuma (*reward*) funkcija, kas ļauj kārtot sasniegto stāvokļus pēc to lietderības;  $T : S \times A \rightarrow S$  vai  $P(s' \in S)$  – pārejas (*transition*) funkcija, kas nosaka nākamo stāvokli  $s'$  vai tam atbilstošu varbūtību sadalījumu, ja pie iepriekšējā stāvokļa  $s$  izvēlēta darbība  $a$ ;  $\gamma$  – koeficients nākotnes atalgojumu vērtību samazināšanai. MDP ir *galīgs* ja  $S, A$  ir galīgas kopas. Ja  $s' = T(s, a)$  ir determinēts, MDP ir *determinēts*. Ja  $s'$  ir gadījuma lielums, kas pieder sadalījumam  $P(s') = T(s, a)$ , MDP ir *stohastisks*.

Atdarinošās mašīnmācīšanās metodēm ne vienmēr ir nepieciešams definēt atalgojuma funkciju un attiecīgi arī  $\gamma$ , taču tie ir nepieciešami metodēm, kas lieto stimulēto mašīn-mācīšanos. Tā kā parasti spriests tiek par stratēģijām  $\pi_\theta$ , kas izvēlas nākamo darbību  $a$  atkarībā no sistēmas stāvokļa  $s$ , tad bieži vien faktiskā pārejas funkcija ir formā  $P(s') = T(s, \pi_\theta(s), s')$ , t.i., pārejas funkcija apraksta “vides” (*environment*) reakciju uz āagenta (modeļa, stratēģijas) darbību. Pie sākotnējo stāvokļu kopas  $S_0$  un stratēģijas  $\pi$  var spriest par stratēģijas inducēto stāvokļu sadalījumu  $s_t \sim P(S|S_0, \pi)$  – tas nosaka, kādas trajektorijas vispār ir iespējamas pie šādiem nosacījumiem.

Faktiski gandrīz nekad nav pieejams vides momentānā stāvokļa pilns raksturojums. Tā vietā zināma ir kāda to aprakstošu atribūtu apakškopa – novērojums  $o_t = g(s_t)$ . Gadījumos, kad šie novērojumi veido vadības algoritma vajadzībām pietiekami precīzu un tieši pielāgojamu stāvokļa aprakstu, nekādi papildu formālismi nav nepieciešami – tos var saukt par *tiešiem novērojumiem* un uztvert kā ekvivalentus stāvoklim  $s_t$ . Ja nepieciešami

papildu soļi, lai no novērojuma iegūtu vadības algoritmam derīgu stāvokļa raksturojumu, tos sauc par *netiešiem novērojumiem*.

Trajektoriju caur sistēmas stāvokļu (konfigurāciju) telpu, kādai process seko ar laika soļiem  $t = \{1, 2, \dots, T\}$ , var izteikt kā laikrinda formā, kas sastāv no *state-action* pāriem –  $((s_1, a_1), (s_2, a_2), \dots, (s_T, a_T))$ . Ja šī laikrinda tiek izmantota kā paraugs treniņa procesā, to var saukt par demonstrāciju. Stāvokļus tajā, protams, iespējams aizstāt ar novērojumiem situācijās, kad tiek izmantota nepilnīga informācija. Ne vienmēr vēlams vai iespējams modelēt sistēmu ar MDP. Ir iespējami gadījumi, kad pārejas funkcija vai stratēģija ir atkarīga no laika soļa, t.i.,  $T(s, a|t), \pi(s, a|t)$ . Var būt arī tā, ka ar novērojumiem nepietiek lēmuma pieņemšanai un nepieciešams ņemt vērā iepriekšējo stāvokļu un darbību virkni, lai pareizi spriestu par slēptiem stāvokļa atribūtiem.

### 1.3. Stimulētā mašīnmācīšanās

Stimulētā mašīnmācīšanās ir pati par sevi ļoti aktuāla izpētes nozare, un nereti nodarbojas ar to pašu vai līdzīgu uzdevumu risināšanu kā atdarinošā. Pastāv ne tikai kombinēti paņēmieni [20, 10], bet arī atdarināšanas metodes, kas tiešā veidā izmanto stimu-lēto mācīšanos, lai atdarinātu trajektoriju demonstrācijas [16]. Tāpēc nav nekāds pārsteigums, ka šis termins visnotaļ bieži parādās ar atdarinošo mašīnmācīšanos saistītos pētījumos, citreiz bez nekādiem papildus paskaidrojumiem.

Stimulētās mašīnmācīšanās teorētiskie pamati ir galīgi MDP un Belmana vienādojums [66]. Pieņem, ka katram stāvoklim ir kāds atalgojums  $R(s_t)$ , bet uzdevums – maksimizēt šo atalgojumu summu visā trajektorijas garumā  $\sum_{t=1}^T R(s_t)$ . Tad var izteikt arī varbūtību sadalījumu atalgojumam katram stāvokļa un darbības pārim

$$p(s', r|s_t, a_t) = P[s_{t+1} = s', r = R(s')] \quad (1.30)$$

Nākotnē sagaidāmās atalgojuma vērtības, ņemot vērā dilšanas koeficientu  $\gamma$ , var izteikt kā

$$G_t = \sum_{k=0}^{T-t} \gamma^k R(s_{t+k+1}) \quad (1.31)$$

Jebkura stratēģija katram stāvoklim nosaka darbību vai darbību sadalījumu  $p(a|s) = \pi(a, s)$ . Var izmantot rekursīvu sakārību, lai katram stāvoklim piekārtotu sagaidāmo atalgojumu jeb vērtību  $v_\pi(s)$ , kas atkarīga no izmantotās stratēģijas – Belmana vienādojumu.

$$v_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] = \sum_a \pi(a, s) \sum_{s', r} p(s', r | s_t, a_t) [r + \gamma v_\pi(s')] \quad (1.32)$$

Atrisināt mācīšanās uzdevumu tādā gadījumā nozīmē atrast stratēģiju, kas maksimizē atalgojumu. Pastāv dažādas metodes, kā to darīt. Kā ilustratīvu piemēru var minēt Q-mācīšanās algoritmu. Tā strādā samērā vienkārši – tiek izveidots tensors  $Q$  ar elementu, kas atbilst katrai iespējamai  $(s, a)$  vērtībai, tam tiek piešķirta kāda sākotnējā vērtība (piemēram, 0).

Apmācība notiek, izvēloties

$$a_t = \underset{a}{\operatorname{argmax}} Q(s_t = s, a) \quad (1.33)$$

un sasniedzot trajektorijas beigas – vai nu pēc noteikta solu skaita  $T$ , vai arī kāda pārtraukšanas nosacījuma. Tad iegūtajai trajektorijai  $(s_1, a_1), (s_2, a_2), \dots, (s_T, a_T)$  no bei-gām aprēķinot  $G_1, G_2, \dots, G_T$  atbilstoši katram solim var korigēt vērtības tenzorā, kur  $Q^i$  apzīmē tenzora vērtības i-tajā treniņa solī

$$Q^{i+1}(s_t, a_t) = f(Q^i(s_t, a_t), G_t) \quad (1.34)$$

Kaut gan šai metodei ir teorētiskas konvergences garantijas pēc pietiekama iterāciju skaita, ļoti strauji pieaug tās modeļa – tenzora  $Q$  – parametru skaits, pieaugot iespējamo stāvokļu un darbību skaitam – nepieciešams atsevišķi optimizēt katru iespējamo kombināciju, iespējams, ļoti daudzās iterācijās. Tāpēc praksē parasti tiek lietoti modeli, kas aproksimē  $v_\pi(s)$ , piemēram, aģenta-kritiķa (*actor-critic*) neironu tīkli, kas reizē iemācās paredzēt gan sagaidāmo vērtību, gan labāko darbību katram stāvoklim ar potenciāli daudz kompaktāku modeli.

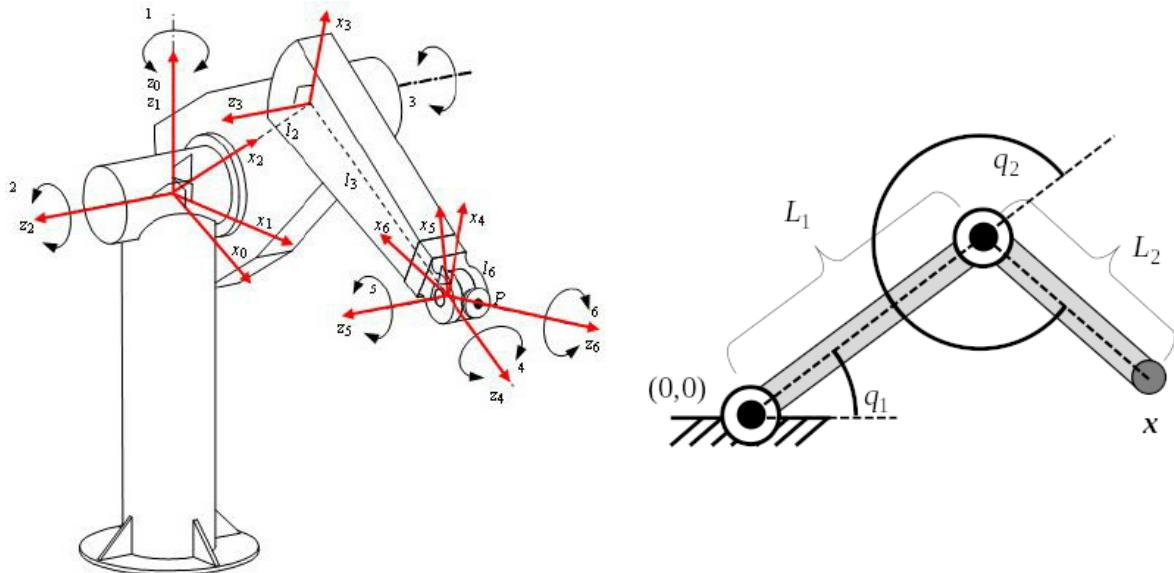
Lai uzdevumu varētu risināt, nepieciešams spēt izteikt kādu analītisku funkciju, kas apraksta pašreizējā stāvokļa lietderību – izšķir labus rezultātus no sliktiem, vai starp-stāvokļiem. Robotikā var būt sarežģīti šādu funkciju izdomāt, turklāt tā var būt ļoti retināta stāvokļu-darbību telpā, t.i., tikai ļoti nelielam skaitam (vai ar ļoti nelielu varbūtību) sasniegto stāvokļu atalgojuma funkcija  $R(s_t)$  pieņem nenuelles vērtību. Tieši šādu trūkumu mēģina risināt metodes, kas kombinē ekspertu demonstrāciju aproksi-mēšanu ar adaptīvu pielāgošanos [40]

## 1.4. Ieskats industriālo rbotu darbībā

Pirms iespējams padziļināti analizēt ar robotu kontroli saistītu zinātnisko literatūru vai patstāvīgi izstrādāt programmatūru to vadībai, nepieciešams iepazīties ar matemātiskajiem formālismiem, kas izmantoti šajā nozarē. Kā jebkurā citā briedumu sasniegušā inženierzinātņu laukā, pastāv virkne vairāk vai mazāk standartizētu paņēmienu, kā iz-paust un risināt dažādas problēmas. Svarīgi izprast ģeometriskās un algebriskās konven-cijas robotu stāvokļa aprakstam un kā tās tiek izmantotas praktisku uzdevumu izpildē.

### 1.4.1. Kinemātika

Robotu mehāniskie izpildījumi var būt ļoti dažādi, taču kopīga ir matemātisko modeļu saime, kas tiek izmantoti, lai attēlotu to izpildmehānismu tiesā veidā nomērāmos stāvokļus – locītavu leņķus, bīdes pāru lineāros pārvietojumus – telpiskajā (Dekarta) koordinātu telpā un otrādi. Tos dēvē par kinemātiskajām kēdēm [23]. Kinemātiskā kēde sastāv no savienojumu virknes, kas katrs ietver kinemātisko pāri un nobīdi. Ka-trs kinemātiskais pāris pieļauj kādu mainīgu cietķermeņa ģeometrisko pārveidojumu – rotāciju un pārvietojumu – kombināciju. Nobīde definē saiti starp diviem kinemātiskiem pāriem. Piemēram, ja divas rotācijas asis ir savienotas ar robota rokas segmentu, to



Att. 1: Pa kreisi – tipiska 6-brīvības pakāpju industriālā robopta shematisks attēls ar kinemātiskās lēdes elementiem – sešiem rotācijas pāriem jeb locītavām, kam katrai ir viena brīvības pakāpe [21]. Pa labi – apzīmējumi kinemātiskajā kēdē ar rotācijas pāriem un fiksētām nobīdēm [23].

attālums veido fiksētu telpisku pārveidojumu. Katram kinemātiskā pāra veidam ir t.s. brīvības pakāpju skaits, kas nosaka to, cik dimensiju vektors ir nepieciešams, lai viennozīmīgi aprakstītu šī pāra konfigurāciju.

Atkarībā no mehānisma uzbūves kinemātiskās lēdes var būt atvērtas (ar vienu fiksētu punktu) vai slēgtas (ar diviem); virknē (katrs nākamais kinemātiskais pāris sa-vienots ar iepriekšējā pārveidotu koordinātu sistēmu) vai paralēlas (vairāki pāri savienoti ar vienu elementu un to stāvokļi ir savstarpēji neatkarīgi). Par kinemātiskās lēdes konfigurāciju var saukt vektoru, kas apraksta visu kinemātisko pāru individuālās konfigurācijas – rotācijas pāru jeb locītavu gadījumā (pieņemot, ka rotācija nav ierobežota) tie var būt leņķi

$$\boldsymbol{\alpha} \in \mathcal{C} \quad (1.35)$$

$$\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \quad (1.36)$$

, kur  $n$  – locītavu skaits,  $\mathcal{C}$  – robota locītavu konfigurāciju telpa. Tā kā darba ietvaros strādāts ar 6-asu industriālajiem manipulatoriem, ilustratīvam piemēram ir vērts izmantot tieši šādu robotu. Kinemātikas kontekstā pamatā tiek strādāts ar diviem uzdevumiem – tiešo un inverso. Robota izpildmehānisms ir piestiprināts pie kinemātiskās lēdes pēdējā elementa, un parasti nepieciešams pārveidot pozīcijas no šīs pārvietojamās koordinātu sistēmas uz robota pamatnes – apkārtējās vides – fiksēto. Tiešās kinemātikas uzdevums ir iegūt summāro pārvietojumu un rotāciju, ko piemērojot kādam vektoram efektoru koordinātu sistēmā, var iegūt atbilstošu pārvietojuma vektoru un orientācijas reprezentāciju (sk. zemāk) globālajā koordinātu sistēmā.

Viens no veidiem, kā definēt pārveidojumu katrā locītavā un savienojumā, ir ar pārveidojuma matricām formā

$$T_{l_i}(q_i) = \begin{bmatrix} R(\mathbf{a}_i, \alpha_i) & L_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.37)$$

, kur  $\alpha_i$  – locītavas leņķis (mainīgais parametrs),  $\mathbf{a}_i$  – rotācijas ass vienības vektors savienojuma koordinātu sistēmā (ap to tiek piemērota rotācija),  $L_i$  – pārvietojums uz nākamo atskaites punktu (3-dimensionāls vektors).  $R(\mathbf{a}_i, \alpha_i)$  ir  $3 \times 3$  telpiskā rotācijas matrica, kas atbilst rotācijai ap doto asi pa doto leņķi. Tā kā šie ir lineāri pārveidojumi, tos var kombinēt

$$T_n(\boldsymbol{\alpha}) = T_{l_n}(\alpha_n)T_{l_{n-1}}(\alpha_{n-1})\dots T_{l_1}(\alpha_1) \quad (1.38)$$

un piemērot pārvietojuma vektoram gala efektoru koordinātu sistēmā ar matricu reizināšanu

$$\mathbf{x}_{world} = T_n(\boldsymbol{\alpha})\mathbf{x}_{ee} \quad (1.39)$$

Summāro rotāciju attiecīgi apzīmē kombinētās pārveidojuma matricas rotācijas komponente augšējā kreisajā stūrī, ko iespējams izteikt Eilera leņķos vai citādi. Viens populārs veids, kā apzīmēt rotācijas 3-dimensiju telpā, ir izmantojot kvaternionus [22]. Tos var uztvert kā kompleksu skaitļu 4-dimensjonālu ekvivalentu. Šī darba ietvaros padziļināti aprakstīt to algebriskās īpašības nav nepieciešams, pietiek tikai minēt, ka, lai izteiktu telpisku rotāciju, tiek izmantota t.s. versora forma

$$q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k} \quad (1.40)$$

$$\|q\| = \sqrt{w^2 + x^2 + y^2 + z^2} = 1 \quad (1.41)$$

, kur  $w$  ir rotācijas skalārā komponente bet  $x, y, z$  veidotais 3-dimensiju vektors ir paralēls atbilstošai rotācijas asij. Svarīgi ievērot, ka kaut gan kvaternions vispārīgā gadījumā ir četrdimensionāls objekts, konstantās normas nosacījums visus versorus novieto uz 3-dimensionālas hipersfēras 4-dimensiju telpā, kas nozīmē, ka telpiskām rotācijām tik un tā ir tikai 3 brīvības pakāpes – tāpat, kā izsakot tās ar leņķiem. Kvaternioniem ir reizinājuma operācija, kam ir vienības vērtība un inversās vērtības

$$qq^{-1} \equiv 1 + 0\mathbf{i} + 0\mathbf{j} + 0\mathbf{k} \quad (1.42)$$

Rotāciju vektoram 3-dimensiju telpā piemēro, to papildinot ar 0 un piemērojot rotācijas operatoru.

$$L_q(\mathbf{v} \in \mathbb{R}^3) \equiv q(0, \mathbf{v})q^{-1} \quad (1.43)$$

Šādām rotācijas operācijām iespējama kompozīcija

$$L_q \circ L_p(\mathbf{v}) = L_{qp}(\mathbf{v}) = (qp)(0, \mathbf{v})(qp)^{-1} \quad (1.44)$$

, kas ļauj kinemātisko ķēdes konfigurācijai atbilstošo koordinātu sistēmas pārvei-dojumu aprakstīt kā rotētu nobīžu vektoru summu

$$\mathbf{x}_{ee} = L_{q_1}(\mathbf{L}_1) + L_{q_2 q_1}(\mathbf{L}_2) + \dots + L_{q_n \dots q_1}(\mathbf{L}_{ee}) \quad (1.45)$$

Robota vadības programmatūras kontekstā bieži var nākties strādāt ar t.s. robota efektora pozu, kas sastāv no divām komponentēm – efektora koordinātu sistēmas sākumpunkta nobīdes un summārās rotācijas

$$\mathbf{x}_{position} = \mathbf{x}_{world} \quad (1.46)$$

$$\mathbf{q}_{orientation} = \prod_{i=1}^n q_i \quad (1.47)$$

Ja tiešās kinemātikas uzdevumā nepieciešams no zināmas robota locītavu konfigurācijas iegūt telpisko pozīciju, tad inversā kinemātika nodarbojas ar atbilstošu locītavu leņķu piemeklēšanu konkrētai telpiskai pozīcijai [24]. Formāli, tipiskais inversās kinemātikas uzdevums ir atrast  $f^{-1}$  pie konkrētas orientācijas un pozīcijas (kopā – pozas)

$$f(\boldsymbol{\alpha}) \equiv \mathbf{x}_{desired} = \begin{bmatrix} \mathbf{x}_{position} \\ \mathbf{x}_{orientation} \end{bmatrix} \quad (1.48)$$

, kur  $\mathbf{x}_{position}$ ,  $\mathbf{x}_{orientation}$  ir pozīcijas un orientācijas ierobežojumi, kas atbilst konkrētam pārvietojumam un orientācijai, izteikti 3-dimensionāli. Protams, iespējams izmantot arī cita veida ierobežojumus, un to skaits var būt dažāds. Atkarībā no dotā lineāri neatkarīgu ierobežojumu skaita  $m$  un robota brīvības pakāpju skaita  $n$ , iespējamas sekojošās situācijas:

- 1)  $n > m$  – problēma ir nepietiekami ierobežota. Ja atrisinājums pastāv (nav ārpus robota sasniedzamās darba zonas) un nav singulārs, tad atrisinājumu ir bezgalīgi daudz;
- 2)  $n = m$  – problēma ir precīzi ierobežota (tipiski 6-asu robotam 3-dimensiju telpā). Ja atrisinājums pastāv, tad to ir galīgā skaitā;
- 3)  $n < m$  – problēma ir pārlieku ierobežota. Risinājums pastāv tikai punktu apakškopai ar samazinātu dimensionalitāti.

Vienkāršākais veids, kā varētu aprakstīt singulārus risinājumus jeb singularitātes robotikas kontekstā, varētu būt sekojošs: tie ir punkti pozu telpā  $x_{singularity}$ , kuriem pēc tiešās kinemātikas piekārtotās locītavu konfigurācijas telpa  $\{\boldsymbol{\alpha} \mid f(\boldsymbol{\alpha}) = x_{singularity}\}$  ir ar citu dimensionalitāti, nekā jebkurā tā topoloģisko kaimiņu kopā. Piemēram, precīzi ierobežotās problēmas gadījumā tas varētu būtu punkts, kam iespējama tikai viena atbilstoša robota konfigurācija, bet jebkurā tā kaimiņu kopā pastāv punkti ar lielāku, galīgu iespējamu konfigurāciju skaitu. Viena šādu pozīciju saime ir visas maksimāli iztaisnota robota sasniegtās pozas.

Singularitāšu pastāvēšana, vairāku vai bezgalīgi daudzu konfigurāciju pastāvēšana (gandrīz) katrai sasniedzamai pozai un nelineārās sakarības starp locītavu leņķi un efektoru atrašanās vietu telpā nozīmē to, ka izmantot vienkāršus lineārus operatorus inversās kinemātikas uzdevuma risināšanai nav iespējams. Tā vietā iespējams vai nu sastādīt ne-lineāras vienādojumu sistēmas, kas atbilst robota ģeometrijai, vai rēķināt rezultātu ar skaitliskām metodēm.

#### 1.4.2. Trajektoriju plānošana

Ar tiešo un inverso kinemātiku vien nepietiek, lai būtu iespējams realizēt robota vadību. Pat ja zināms robota sākuma un beigu stāvoklis pozu vai konfigurāciju telpā, jebkura kustība starp tiem sastāda līknes – gan konfigurācijā, gan pozā. Parasti abās telpās pastāv arī ierobežojumi, kas jāievēro – robots nedrīkst sadurties ar objektiem apkārtējā vidē (absolūtās vērtības pozu telpā), tā locītavu ātrumi vai paātrinājumi (parciālie atvasinājumi konfigurāciju telpā) nedrīkst pārsniegt ierobežojumus, u.t.t. Robotikas kontekstā kustības plānošanas problēmu var izteikt kā [25]

$$y(s) : [0, 1] \rightarrow \mathcal{C} \quad (1.49)$$

, kas piekārto katru vērtību nepārtrauktajā intervālā  $[0, 1]$  atbilstošai robota konfigurācijai, ar nosacījumu, ka  $y(s)$  nav pārtraukuma punktu.  $y(0), y(1)$  apzīmē robota sākuma un beigu stāvokļus, pārējās  $s$  vērtības – jebkuru trajektorijas punktu starp tiem.

Telpiskās pozīcijas un locītavu konfigurāciju – kinemātiskie – ierobežojumi nosaka atļauto apgabalu  $\mathcal{F} \subseteq \mathcal{C}$ , un attiecīgi

$$\forall s, y(s) \in \mathcal{F} \quad (1.50)$$

Savukārt ierobežojumus, kas attiecināmi uz  $y(s)$  atvasinājumiem, sauc par dinamiskajiem. Svarīgi piebilst, ka ne vienmēr iespējams izpildīt nepārtrauktu kustību pozu telpā – var gadīties, ka, lai nonāktu no viena punkta nākamajā, nepieciešams tāds kustības segments konfigurāciju telpā, kura  $f(y(s))$  nepieder telpiskajai trajektorijai. Šādu parādību sauc par “lēcienu”, un no tās ir svarīgi izvairīties, ja uzdevuma ietvaros ir svarīga pareiza kustības telpiskā ģeometrija, kā tas ir ar metieniem.

## 2. ATDARINOŠĀ MAŠĪNMĀCĪŠANĀS

Šīs nodaļas mērķis ir pārskatīt ar nozares pētniecību saistīto zinātnisko literatūru; aprakstīt galvenos sasniegtos rezultātus, gūtās atziņas katrā no tematiskajiem apakšvirzieniem. Protams, ne visus pētījumus iespējams vienkārši klasificēt pēc to piederības šeit izvēlētajām kategorijām, un daudzi varbūt tajā vispār neiederas – taču cenšoties gūt personisku izpratni par kādu tēmu, lai motivētu tālākus pētījumus, ir svarīgi nostatīt iepriekšējus rezultātus to kontekstā. Saprast, kāpēc tieši šobrīd aktuālie pētniecības virzieni ir tādi, kādus tos varam redzēt kādā akadēmisko publikāciju datubāzē vai neseno pētījumu pārskatā.

Salīdzinot ar citām aktuālām mašīnmācīšanās pētniecības nozarēm – piemēram, jau minēto stimulēto mācīšanos – tieši par atdarināšanu veiktās pētniecības apjoms ir relatīvi neliels. Taču tik un tā pieejams liels skaits publikāciju, kas apraksta pētījumus ļoti dažādos virzienos. Turklat robotika ir viens no izplatītākajiem pielietojuma mērķiem šādām metodēm, konkurējot ar datorspēļu aģentiem. Lai radītu priekšstatu par nozares pašreizējo stāvokli un tematisku dalījumu, izšķirti trīs virzieni, kas labi apraksta lielu daļu no pētījumiem par iespējām robotus apmācīt ar piemēriem:

- 1) trajektoriju kopēšana – mērķis ir panākt robustu, precīzu atdarināšanu ar nelielām treniņa datu kopām, ja pieejama visa nepieciešamā informācija par sistēmas stāvokli;
- 2) novērojumu iegūšana, interpretācija, papildināšana – ne vienmēr ir pieejami dati formā, ko var tiešā veidā izmantot imitējamu trajektoriju iegūšanā. Daudzi pētījumi nodarbojas tieši ar apmācībai derīgu novērojumu iegūšanu – netiešu (piemēram, video) novērojumu pārveidošanu, labākām cilvēka-robota saskarnes metodēm, trajektorijām ar nezināmām darbībām, u.c.;
- 3) atdarināšana un adaptācija, vispārināšana – atdarinošās mācīšanās pielietojums, lai uzlabotu stimulēto, un otrādi; vispārīgu prasmju iegūšana no demonstrācijām, tūlītēja atdarināšana. Mēģinājumi panākt, ka modelis neaprobežojas tikai ar piemēros esošo un spēj pielāgoties. Stimulētās mācīšanās uzsākšana ļoti retinātās atalgojumu telpās.

### 2.1. Labi definētu trajektoriju kopēšana

Pirmā, varētu teikt galvenā, taču ne vienmēr vienkāršākā problēma, ir atrast veidu, kā pieejamās ekspertu zināšanas – robotikas kontekstā tās parasti būs pareizas trajektorijas dažādu pārvietojumu un smalku manipulācijas uzdevumu risināšanai – tiešā veidā atdarināt. Šo procesu mēdz saukt arī par programmēšanu ar demonstrācijām (*PBD – programming by demonstration*) [39, 7]. Idealizētā vidē ar determinētām stāvokļu pārejām un pilnīgu informāciju par tās pašreizējo konfigurāciju šis uzdevums varētu būt pat triviāls, taču praksē saskaramies ar problēmām:

- 1) darbs notiek ar novērojumiem, nevis stāvokļiem. Pat ja pieejami, piemēram, trajektoriju ieraksti, bieži vien trūkst svarīgas informācijas (varētu būt zināma trajektorijas kinemātika, bet ne tās dinamika – paātrinājumi, spēki);

- 2) atšķirības vidē: izpildelementos – varbūt robots ir nedaudz citāds; apkārtnē – varbūt manipulējamo objektu masas, forma vai izvietojums ir nedaudz atšķirīgi no demonstrācijās esošajiem;
- 3) ja trajektoriju generējis eksperts, kam, iespējams bijusi pieejama informācija, kuras aģentam nav – piemēram, manipulāciju veicis cilvēks ar redzi, bet robotam pieejami tikai kontakta sensori.

Problēmas faktiski nozīmē to, ka reālā sistēmā stāvokļu pārejas nav determinētas attiecībā pret novērojumiem un darbībām. Lai labāk saprastu šos trūkumus, vispirms noderīgi ir aplūkot “naivākos” veidus, kā varētu imitēt piemērus. Šajā apakšnodalā aprakstīti pētījumi, kuros uzsvars ir uz pietiekami detalizētu demonstrāciju atdarināšanas procesiem.

### **2.1.1. Vienkāršas metodes**

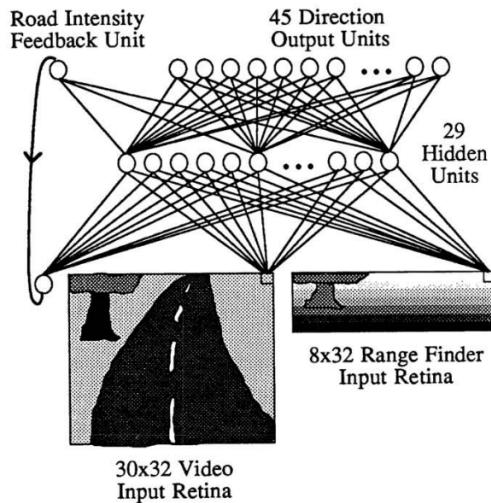
Pirmais, ko varētu darīt, ir tiešā veidā ierakstīt trajektoriju un to atkārtot. Šī nebūt nav jauna ideja – gandrīz visiem mūsdienu industriāliem robotiem ir pieejamas t.s. *lead-through* un *teach-in* programmēšanas metodes, kas ļauj fiziski un ar tālvadības ierīces palīdzību vadīt robota kustību un to ierakstīt pēcākai atdarināšanai [44], turklāt tās parādījušās jau pašos industriālās robotikas pirmsākumos 1970os gados [19].

Tā kā trajektorijai jābūt ierakstītai tieši ar robotu, lai tā būtu atkārtojama bez papildus datizraces uzdevumu risināšanas, var paredzēt, ka dabiski radīsies zināma tendence dominēt viegli realizējamiem, bet ne garantēti optimāliem ceļiem telpā – vieglāk ierakstīt dažus pagrieziena punktus un ļaut programmatūrai interpolēt nekā fiziski vadīt robotu visā kustības ceļā. Pie tam var parādīties neparedzēti trūkumi, pārejot no lēnas, nenoslogotas izpildes programmēšanas procesā uz ātru un noslogotu ekspluatācijā, kas apgrūtina procesu. Faktiski sākotnējais ieraksts bieži vien kalpo par starta punktu, bet, lai nonāktu pie lietojamas programmas, nepieciešams iegūto kodu korigēt un iteratīvi pielāgot. Lai arī principā tiek izmantota demonstrācija trajektorijas iegūšanai, procesa veikšanai tik un tā nepieciešams personāls ar robotu programmēšanas prasmēm. Jau sen atzīts [39, 7], ka, lai tik tiesām robotus varētu apmācīt tikai ar piemēriem, nepieciešamas metodes, kas ir robustākas pret nobīdēm no paraugu ģenerējošā procesa apstākļiem un vispārināmākas, tāpēc uzsākti pētījumi ar mašīnmācīšanās metodēm.

Kad jāspēj atdarināt kas vairāk nekā viena, nemainīga trajektorija, nepieciešams atdarināt nevis pašu trajektoriju, bet gan procesu, kas tādas ģenerē – “eksperta” jeb “instruktora” stratēģiju. Viena no vienkāršākajām metodēm, kas bieži tiek lietots kā piemērs, taču praksē reti kad ir pielietojuma, ir uzvedības klonēšana (*behavioural cloning*). Vispārīgi to definēt ir samērā vienkārši [5]. Ja dots MDP un kāda eksperta stratēģija  $\pi^*$ , kas šo MDP optimāli risina, mērķis ir atrast maksimāli tuvu modeli  $\pi_\theta$ , kur

$$\pi_\theta(s) \approx \pi^*(s) \quad (2.1)$$

Parasti, protams, ir pieejama datu kopa ar eksperta izietajām stāvokļu-darbību laikrindām, turklāt jāstrādā ir ar novērojumiem, nevis stāvokļiem. Lai veidotu vispārīgu izpratni par atdarinošo mašīnmācīšanos, noderīgi ir detalizētāk aplūkot kādu vienkāršu tās



Att. 2: ALVINN modeļa uzbūve [49]

piemēru. Uzvedības klonēšana sistēmā, kur netiek veikta intensīva treniņa datu pārstrāde vai vadības algoritma argumentu pārveidošana, ļoti labi kalpo šādiem mērķiem, tāpēc var aplūkot 1989. gadā Kārnegija-Melona Universitātē veikto pētījumu “*Autonomous Land Vehicle in a Neural Network*” (ALVINN) [49]. Tā mērķis bija izstrādāt pašbraucošu automašīnu, kas spēj sekot ceļa kontūram.

Automašīna tikusi aprīkota ar videokameru un LIDAR sensoriem, kas devuši dienus skatus uz to pašu telpas reģionu automobiļa priekšā. Par apmācāmo modeli izvēlēts neironu tīkls. Protams, 1989. gads vēl bija laiks, kad datoru veikspēja bija stipri ierobežota, tāpēc neironu tīkls ir klasiskas uzbūves – tam ir viens slēptais slānis ar 29 perceptroniem, kam seko 45 izvades elementi. Video izmantots krāsinā attēla zilais kanāls, jo tajā ceļa virsma visvairāk kontrastē ar apkārtējo vidi. Gan video, gan LIDAR radītie attēli tīkla ievadē veido vienkāršu vektoru bez nekādiem telpiskiem kodējumiem, visi slāņi savstarpēji pilnībā savienoti.

Modeļa izvades slānis apzīmē vēlamo stūrēšanas virzienu 45 diskrētos soļos. Treniņa datu kopā faktisko virziena komandu atspoguļo neprecizēta veida “zvana” funkcija ar modu pie pareizā virziena. Ieviests viens papildus perceptrons, kas (teorētiski) novērtē ceļa gaišumu, salīdzinot ar apkārtējo vidi, un tiek pievienots nākamās iterācijas ievades vektoram. Konstatēts, ka ievākt treniņa datus fizikālā vidē – braucot ar automašīnu pa ceļiem un ierakstot vadītāja veiktās korekcijas – nav praktiski, jo nepieciešama ļoti liela treniņa datu kopa. Tāpēc dati ģenerēti sintētiski – tā kā gan video, gan attāluma datu izšķirtspēja ir gaužām neliela, pat ar 1989. gadā pieejamām datorgrafikas iespējām šādi gūtus attēlus ir grūti atšķirt no īstiem. Simulatorā iegūtie attēli un vadības komandas izmantoti klasifikatora apmācībā.

Iegūtais rezultāts – modelis, kas bijis pietiekami labs, lai spētu vadīt ar kameru un attāluma sensoru aprīkotu automobili pa 400m garu slēgta ceļa posmu saulainos dienas apstākļos, ar ātrumu 0,5m/s. Kā galvenais uzvedības klonēšanas trūkums parasti tiek minēta nespēja atgūties no faktiskā stāvokļa sadalījumu nobīdes [5] (*distribution shift*). Ja reālais modelis  $\pi_\theta(s)$  nevar pilnīgi precīzi atdarināt eksperta  $\pi^*(s)$  darbības vai MDP

pārejas funkcija ir stohastiska, tātad treniņa datu kopa neietver visas iespējamās trajektorijas ar atbilstošajām  $\pi^*(s)$  vērtībām,  $\pi_\theta$  inducētais stāvokļu sadalījums diverģē no  $\pi^*$  inducētā. Lai iegūtu precīzāku un robustāku eksperta stratēģijas atdarinājumu, piedāvāti dažādi – sarežģītāki – apmācības paņēmieni.

### **2.1.2. Statistiskas korekcijas**

Viens no virzieniem, kurā veikti mēģinājumi uzlabot trajektoriju kopēšanas lietderību, ir strukturēt treniņa datu ieguvi un apmācības algoritmu veidā, kas maksimāli tuvina  $\pi^*$  un  $\pi_\theta$  inducētos stāvokļu sadalījumus. Bieži vien tas nozīmē, ka vienkārši ievākt datus un veikt apmācību uz tiem vairs nav iespējams – nepieciešama aktīva instruktora iesaiste. Piedāvātie risinājumi ir dažādi, un metodes var klūt visnotaļ sarežģītas [5], tāpēc, lai ilustrētu pieejas būtību kopumā, izvēlēts viens, vairāk teorētisks piemērs.

DAgger – *dataset aggregation* – ir 2011. gadā publicētajā rakstā “*A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*” [59], piedāvāts algoritms. Tas pierāda teorētiskas garantijas  $\pi^*$  un  $\pi_\theta$  inducēto sadalījumu konvergēncēi, kombinējot instruktāžu ar apmācāmā modeļa ģenerētām stratēģijām. Lai gan raksts nav tieši saistīts ar robotiku, izmantotais MDP kontroles formālisms ir vispārīgs.

Algoritma darbību var vienkāršoti aprakstīt sekojoši: pieņem, ka ir pieejami ne tikai eksperta ģenerēti trajektoriju dati, bet ir iespējams pašam ekspertam uzzot vaicājumus par katrā stāvoklī optimālu darbību – kas, ja instruktāžu nodrošina cilvēks un laika soļu pārejas ir biežas, reti kad būs praktiski iespējams. Tādā gadījumā iteratīvi atkārto šādus soļus:

- 1) ar kādu varbūtību  $\alpha$  izvēlas, vai i-tā trajektorija tiks ģenerēta ar  $\pi^*$  vai  $\pi_\theta$ ;
- 2) iegūtās laikrindas stāvokļa elementiem  $s_t$  atrod atbilstošo  $a_t^* = \pi^*(s_t)$
- 3) kopējai datu kopai  $D$  pievieno  $D_i = \{(s_1, a_1^*), \dots\}$
- 4) apmāca modeli  $\pi_\theta$  uz papildinātās datu kopas

Kā jau minēts, liela daļa raksta satura veltīta tiesi algoritma teorētisko īpašību pierādīšanai, taču beigās arī veikti daži eksperimenti – divi ar personāžu vadību datorspēļu vidē, viens ar rokraksta zīmju atpazīšanu teksta virknēs. Lai gan visos gadījumos DAgger pārspēj uzvedības klonēšanas (vienkāršas  $\pi^*$  aproksimēšanas no treniņa datu kopas) rezultātus, tā lietderību stipri ierobežo instruktora interaktivitātes prasības – praksē reti kad ir iespējams kaut kas analogisks datorspēļu aģentu treniņam izmantotajam simulatoram, kas ar dziļu pārlasi atrod labas stratēģijas jebkuram stāvoklim.

### **2.1.3. Inversā stimulētā mācīšanās**

Cits veids, kā atdarināt instruktora dotas trajektorijas, ir pieņemt, ka tā stratēģija optimizē kādu slēptu atalgojuma funkciju  $R^*(s)$  un mēģināt to atjaunot no pieejamās informācijas. Šādā veidā ar stimulētās mašīnmācīšanās metodēm var iegūt meklēto rezultātu. Kā jau parasti, iespējami dažādi veidi, kā formalizēt uzdevumu un tehniski to realizēt. 2004. izdotais “*Apprenticeship learning via inverse reinforcement learning*” [1] no Džordžijas Tehnoloģiju institūta, viens no citētākajiem rakstiem par šo tēmu (lai arī ne pirmais), piedāvā iteratīvu algoritmu nezināmas atalgojuma funkcijas atjaunošanai un izmantošanai. Galvenā atkāpe no tipiska MDP formālisma ir pieņēmums, ka nezināmā

atalgojuma funkcija  $R^*$  ir formā

$$R^*(s) = w^* \cdot \phi(s) \quad (2.2)$$

, kur  $w^*$  – svaru vektors, bet  $\phi : S \rightarrow [0, 1]^k$  – zināmu atribūtu izpausme noteiktos stāvokļos.  $\phi$  nozīme ir tāda, ka ir iespējams noteikt, kādu sakarību lineāra kombinācija varētu būt īstā atalgojuma funkcija. Kā piemērs tiek piedāvāts autovadītāja uzdevums – viens no atribūtiem varētu būt 1, ja mašīna atrodas uz ceļa, bet 0 citādi, u.t.t.  $m$  treniņa kopas trajektorijām aprēķina vidējo faktoru vērtību summu, izteiktu kā

$$\mu^* = \frac{1}{m} \sum_{i=1}^m \sum_{t=1}^T \gamma^t \phi(s_t^i) \quad (2.3)$$

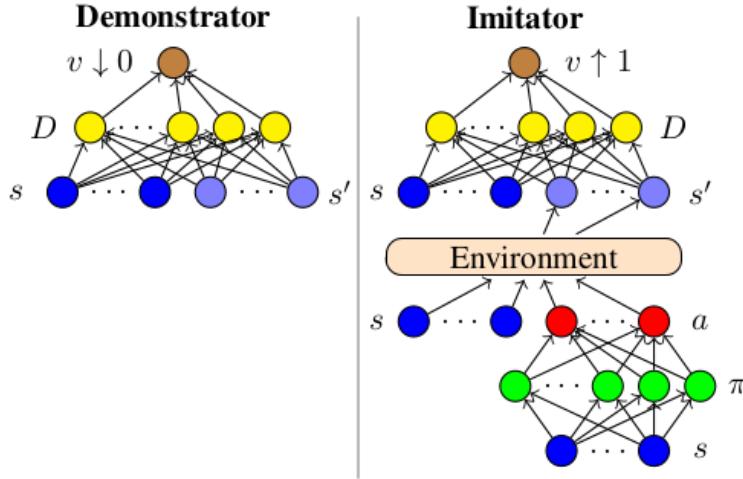
Tad tiek iteratīvi atkārtota procedūra, kur atrod kādu svaru vektoru  $w^i$  un attiecīgi empīrisku atalgojuma funkciju  $R^i(s) = w^{iT} \phi(s)$ , ko izmanto, lai apmācītu jaunu stratēģiju  $\pi^i$ . Tad šai stratēģijai atrod vidējo vērtību  $\mu^i$  analogiski (2.3), un visas iepriekšējās  $\mu^{j \leq i}$  tiek izmantotas, lai atrastu nākamo svaru vektoru  $w^{i+1}$ . Process turpinās, līdz ir konvergējis līdz noteiktam klūdas hiperparametram. Tādējādi beigās iegūta stratēģija, kas maksimizē līdzīgu atribūtu  $\phi(s)$  kombināciju nezināmajai instruktora stratēģijai, un robusti seko demonstrācijām.

Rezultāti parāda, ka šī metode pārspēj dažādas vienkāršas, statistiskas  $\pi^*$  aproksimācijas metodes (uzvedības klonēšanu). Kopā risināti divi dažādi uzdevumi. Viens ir “gridworld” – spēle, kurā aģents pārvietojas pa režģa formas vidi un dažos lauciņos ir pieejams atalgojums. Taču pārejas process ir stohastisks, tāpēc metodes, kas atdarina tikai telpiskos pārvietojumus un nemēģina atjaunot slēpto atalgojuma funkciju darbojas sliktāk. Otrs ir divdimensionāla spēle, kurā aģents vada automobili. Šeit tika pārbaudīts, vai var iemācīt aģentam atšķirīgus “braukšanas stilus” tikai ar demonstrācijām, kas arī izdevies.

#### 2.1.4. Generatīvie pretinieku tīkli

Iedvesmojoties no inversās stimulētās mācīšanās, attīstītas arī citas metodes, kas tiešā vai netiešā veidā nodarbojas ar instruktora stratēģijas aproksimēšanu. Bieži kā trūkums IRL metodēm tiek minēta nepieciešamība katrai iteratīvi iegūtajai stratēģija no jauna veikt stimulēto apmācības procesu, lai būtu iespējams iegūt šīs metodes generētas trajektorijas un līdz ar to varētu novērtēt to sasniegto stāvokļu atribūtu sadalījumus. Meklējot veidus, kā tiešā veidā optimizēt stratēģiju, lai tā sasniegtu tādus pašus novērtējumus kā demonstrācijas pie plašām iespējamo atalgojuma funkciju klasēm, izlaižot pašu šo atalgojuma funkciju meklēšanu, 2016. gadā publicēts “Generative Adversarial Imitation Learning” [27]. Tas piedāvā risināt trajektoriju atdarināšanas uzdevumu ar pretinieku tīklu metodi, un ir bijis samērā ietekmīgs uz tālākiem pētījumiem nozarē, jo šobrīd tā jau ir visai populāra metode, un vēl salīdzinoši nesen tika uzskatīta par labāko tieši trajektoriju kopēšanas uzdevumā [75].

Ģeneratīvie pretinieku tīkli – GAN, *generative adversarial networks* – tiek apmācīti ar metodi, kas ļauj izmantot adaptīvu mērķa funkciju. Tā vietā, lai optimizētu visu modeli



Att. 3: GAN uzbūves piemērs. Augšējais tīkls – diskriminators – cenšas atšķirt eksperta demonstrācijas no ģeneratora radītām trajektorijām [75]

vienam mērķim, tiek izdalīti divi elementi – ģenerators un diskriminators – ar pretējiem uzdevumiem. Diskriminators ir klasifikators, kas apmācīts atšķirt demonstrāciju kopas trajektorijas vai to elementus no visām pārējām. Generators no sistēmas stāvokļiem vai novērojumiem ģenerē darbības tā, lai diskriminators nebūtu spējīgs atšķirt tās no parauga. Formāli pierādījumi šī modeļa konvergēncēi ir pārāk apjomīgi, lai tos apskatītu šajā darbā. Tomēr no augstāk minētā pētījuma un citiem tā iedvesmotajiem rakstiem, kas aprakstīti zemāk, var secināt, ka pēc sekmīgas abu modeļu apmācības būs iegūta stratēģija, kas tuvu aproksimē ievades datu sadalījumu.

#### 2.1.5. Uzdevumu simboliska dekompozīcija

Vēl viena metode atdarināšanas spēju uzlabošanai ir telpisku kustības trajektoriju pārvēršana simbolisku, diskrētu darbību virknē. Pamatideja ir tāda, ka vieglāk iemācīties robusti izpildīt primitīvas kustības un tad šādu primitīvo kustību secību kāda uzdevuma izpildē, nekā no neliela demonstrācija skaits iemācīties katru uzdevumu pilnībā no jauna. Šī arī nebūt nav jauna ideja – izteikta jau 1980os un 1990os gados [39]. Jau 2002. gadā veikti pētījumi par algoritmiem, kas ļauj aproksimēt trajektorijas elementus ar autonomu nelineāru diferenciālvienādojumu sistēmām [30] un iemācīt pietiekami sarežģītas kustības – piemēram, tenisa bumbas sišanu – ar samērā nedaudziem piemēriem (ap 20). Ap to pašu laiku piedāvātas arī pieejas šādu primitīvu kustību kombinēšanai [60], kur mašīnmācīšanās lietojums attiecināts ne tikai uz atsevišķajiem trajektorijas primitīviem, bet arī uz katrai demonstrācijai atbilstošas to secības meklēšanu.

Robotikas literatūrā pirms 2010. gada [7] plaši rakstīts par šādiem paņēmieniem, taču pēdējos gados aizvien biežāk tiek izmantotas metodes, kuru pamatā ir vispārīgāki neironu tīklu modeļi. Jebkurā gadījumā, aktīva pētniecība nozarē vēl joprojām notiek, it sevišķi pielietojumiem, kur robots tiek mācīts ar kinestētiskām metodēm. Piemēram, “A Framework of Hybrid Force/Motion Skills Learning for Robots” [77], kas publicēts 2020. gadā, šāda pieeja tiek veiksmīgi izmantota uzdevumiem, kur svarīga ne tikai telpiskā

trajektorija, bet arī uz apkārtējo vidi izdarīto spēku profils (galda virsmas tīrišanā).

## 2.2. Novērojumu iegūšana, interpretācija, papildināšana

Pat ja demonstrācijas ir iespējams ļoti precīzi atdarināt, praksē paveras jauna problēma – ne vienmēr iespējams tiešā veidā iegūt pietiekami labus paraugus no instruktoriem. Ja novērojums  $o_t$  ir netiešs – iegūts formā, kas neatbilst robota vadības algoritmam nepieciešamajam sistēmas konfigurācijas aprakstam – to nepieciešams pārveidot. Turklat, darbojoties ar netiešiem novērojumiem, bieži vien tiešā veidā iespējams atjaunot tikai stāvokļu  $s_t$  laikrindu – darbības  $a_t$  paliek nezināmas.

Strādājot apstākļos, kur vienkārši analītiski modeļi ar labu precizitāti var paredzēt stāvokļa atribūtus un sakarības starp tiem, šī atšķirība var nebūt īpaši svarīga. Piemēram, darbojoties ar robotu, kas nav sevišķi smagi noslogots un kura kontroles sistēma spēj bez lielām novirzēm atdarināt no tās prasīto kinemātiku, var ar augstu precizitāti darboties ar novērojumiem, kuros zināmi tikai šie kinemātiskie atribūti. Tas varbūt pat ir vienkāršāk, nekā izmantot smalkāku konfigurācijas aprakstu, kur pieejami arī visi dinamikas atribūti.

Taču daudziem potenciāli ļoti noderīgiem lietojumiem galvenais šķērslis apmācībai var būt tieši derīgu treniņa datu kopu iegūšana no nepilnīgiem novērojumiem. Problemas var sagādāt demonstrāciju ģenerēšana ar citādas ģeometrijas instruktoru (piemēram, cilvēku), citu objektu sarežģītu un iepriekš nezināmu konfigurāciju modelēšana (manipulējamo objektu novietojums, orientācija, u.t.t.), trajektoriju automātiska iegūšana no datu kopām, kas nav tiešā veidā paredzētas šim mērķim (video ieraksti). Tāpat jāpārvar zināmi izaicinājumi, lai datus varētu ģenerēt ar netiešām metodēm – piemēram, attālinātu vadību vai virtuālās realitātes simulācijām.

### 2.2.1. Nezināmas darbības

Pētījumos, kuru galvenā būtība ir dažādu veidu pārveidojumi ar kinemātiskiem vai dinamiskiem trajektoriju datiem, varētu teikt, ka uzsvars ir uz kinestētiskajiem mācīšanās aspektiem. Iepriekšējā apakšnodalā aplūkoto eksperimentu veicēji pārsvarā pieņemuši, ka pieejami pareizi formatēti dati, kuros novērojuma atribūti ir cieši saistīti ar robota vadībai svarīgiem sistēmas atribūtiem un zināmas katrā trajektorijas solī veiktās darbības. Savukārt šajā tiek apskatīti gadījumi, kad dati ir kādā ziņā nepietiekami vai pārveidotī.

Pirmais sarežģījums, ko varētu ieviest, ir darbību trūkums demonstrācijās. Ar to jāsastopas ļoti daudzos uzdevumos – no nemarķētiem datiem var vienkārši iegūt, piemēram, robota gala efektora pozīciju un rotāciju, taču nekas nav zināms par tā locītavu leņķiskajiem paātrinājumiem. Lai varētu izmantot jau zināmos atdarinošās mācīšanās algoritmus, rodas nepieciešamība uzminēt attiecīgās darbības, kas novērtē pie pārejas no viena stāvokļa uz nākamo.

“*Behavioral Cloning from Observation*” [74] (2018) ir viena šāda metode. Tās mērķis ir realizēt jau aprakstīto uzvedības klonēšanas algoritmu laikrindu datiem, kuros pieejami tikai novērojumi. Lai to panāktu, tiek ieviests papildus modelis, tikai šoreiz nevis stratēģijas, bet gan paša robota (vai cita aģenta) dinamikas aproksimēšanai.

Nedaudz vienkāršojot, algoritma soli ir sekojoši:

- 1) doto trajektoriju kopu pārveido formā  $\mathcal{T}_{dem} = \{(s_t, s_{t+1})\}$ , kas ir pārejas starp stāvokļiem;
- 2) stratēģija  $\pi_\theta$  un sistēmas dinamikas modelis  $M_\phi(a|s, s') = P(a|s, s')$  tiek nejauši inicializēti;
- 3) generē trajektorijas ar  $\pi_\theta$ , pievieno trajektorijas elementus kopai  $\mathcal{T} = \{(s_t, a_t, s_{t+1})\}$ ;
- 4) apmāca  $M_\phi(a|s, s')$  uz  $\mathcal{T}$ ;
- 5) trenē  $\pi_\theta$  uz  $\{(s_t, \underset{a}{\operatorname{argmax}} M_\phi(a|s, s'), s') | (s, s') \in \mathcal{T}_{dem}\}$
- 6) atkārto solus 3-5 līdz sasniegti pienemami rezultāti

Redzams, ka pakāpeniski tiek iegūts sistēmas slēptās dinamikas modelis, kas pareizi paredz pārejas funkcijas darbības parametru, un, tāpat kā parastajā gadījumā, tiek apmācīta atbilstoša stratēģija. Interesanti arī, ka šajā situācijā potenciāli nedaudz lielāks uzsvars ir tieši uz nākamo novērojumu laikrindā, nevis izvēlēto darbību – pie  $s$ ,  $\pi_\theta$  iemācās paredzēt  $s'$  sasniegšanai labāko darbību, nevis vienkārši atkārtot pašu darbību bez konteksta, tātad savā ziņā sistēmas dinamika un vēlamais rezultāts tiek ņemti vērā. Rezultātos autori salīdzina šo metodi ar citām, kas izmanto arī darbību datus no demonstrācijām, un konstatē, ka šis algoritms pat strādā labāk nekā tās, ja tiek vērtēts pēc nepieciešamo demonstrācijas trajektoriju vai simulācijas iterāciju skaita.

Iepriekšējā apakšnodalā 2. attēls ir no “*Generative Adversarial Imitation from Observation*” [74] (2018), kas turpina darboties tajā pašā virzienā, tikai šoreiz ar GAN modeļa palīdzību. Tā vietā, lai modelētu sistēmas dinamiku, diskriminators klasificē trajektoriju laikrindās sastopamās stāvokļu pārejas  $(s, s')$  pēc tā, vai tās būtu sastopamas demonstrācijā, bet ģenerators (kas reizē ir arī stratēģija  $\pi_\theta$ ) tiek trenēts, lai tā izvēlēto darbību rezultātā iegūtās stāvokļu pārejas  $s = T(s, a)$  nebūtu atšķiramas no piemēriem.

### **2.2.2. Dinamikas novērojumu iegūšana, izmantošana, vispārināšana**

Cita veida problēma, kas arī prasa korekciju ieviešanu, ir atdarināšana sistēmām ar mainīgu dinamiku. Robotiem izmantojot “*lead-through*” programmēšanas metodi, kustības tiek programmētas bez slodzes un, iespējams, neievērojot ātrumu – zināma tikai daļēja sistēmas kinemātika. Ja var paredzēt, ka pēc tam ekspluatācijā atšķirsies robotu slogojosie spēki un griezes momenti, tad var meklēt veidus, kā šīs novirzes treniņa datu ieguves procesā kompensēt. “*Online Movement Adaptation based on Previous Sensor Experiences*” [46] (2011) paredz jau iepriekš aprakstīto simboliskās dekompozīcijas modeļu papildināšanu ar korekcijām reālā laikā, izmantojot atgriezenisko saiti ar gan paša robota iekšējiem devējiem, gan ārējiem sensoriem.

Pētījumos, kas nodarbojas ar simbolisko dekompozīciju, bieži vien tiek aprakstīti visai sarežģīti un tieši robotu dinamikai specifiski matemātiskie modeļi. Taču vienkāršoti procesu var aprakstīt sekojoši: kustības raksturojošie diferenciālvienādojumu modeļi tiek iegūti, robotu manuāli vai citādi pārvietojot. Tad kustība tiek veikta ar pareizu kinemātiku (ātrumiem, paātrinājumiem), bet bez papildu slodzes. Tieki ierakstīti sensoru novērojumi un veidoti kustībai atbilstoši modeļi, kas paredz šos raksturlielumus trajektorijas gaitā. Autoru vārdiem – robots iemācās, kā kustībai vajadzētu “justies”. Tad reāli

ekspluatācijā var sekot līdzī nobīdēm no normas un ar klasiskās kontroles teorijas metodēm veikt korekcijas.

Nodarbojoties ar ļoti sarežģītiem uzdevumiem (piemēram, salikšanas procesiem), mēģināt vadīt robotu cauri visiem iespējamajiem detaļu savstarpējiem stāvokļiem var būt neiespējami. Ja var iegūt demonstrācijas kādā citā, vieglāk realizējamā veidā un izstrādāt vispārīgu metodi, kā tos atdarināt neatkarīgi no robota uzbūves, rodas iespējas automatizēt arī šādus procesus. “*Contact Skill Imitation Learning for Robot-Independent Assembly Programming*” [62] (2019) realizē šādu procedūru, izmantojot divus būtiskus elementus:

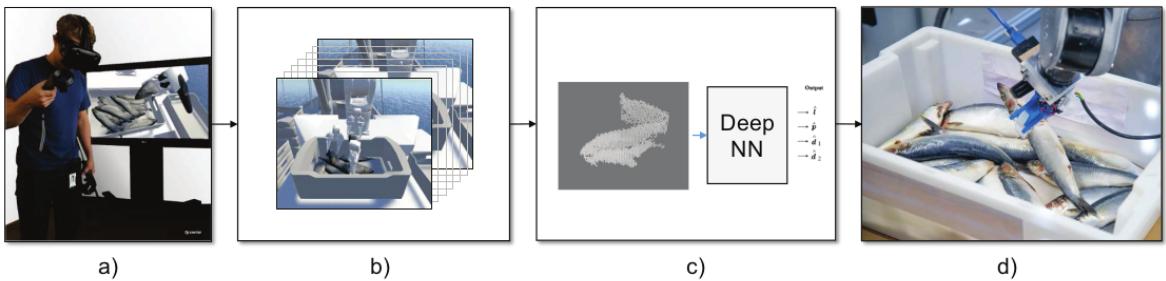
- 1) *forward dynamics compliance control* – robota vadības algoritmu, kurā tiek kontrolēti uz efektoru iedarbojošos spēku un griezes momentu vektori, nevis izpildelementu pozīcija [61];
- 2) rekurentos neironu tīklus laikrindas nākamā elementa paredzēšanai – t.i., tīklus, kuru ievadē parādās iepriekšējā laika soļa izvades vektors no tā paša modeļa;

Demonstrāciju iegūšanai cilvēks simulācijas vidē ar datorpeles palīdzību veic salikšanas procesu, izmantojot tikai vizuālo uztveri un intuitīvu izpratni par sadursmju dinamiku. Novērojumus veido manipulējamā objekta masas centrā reģistrēto griezes momentu un spēku vektori. Stratēģija – rekurentais neironu tīklis –  $\pi_\theta$  tiek apmācīta paredzēt nākamo spēku/momentu vektoru laikrindā, un tas tiek izmantots robota kontrolei ar minēto vadības algoritmu.

### **2.2.3. Demonstrācijas no cilvēka darbībām**

Ja nepieciešams ātri un lēti radīt treniņa datu kopas apmācības procesiem – kā tas būtu praktiski izmantojamai robotu programmēšanas sistēmai – svarīgi radīt cilvēkam draudzīgu saskarni. Tā vietā lai programmētu robota trajektoriju ar tradicionālām metodēm, ir veikti mēģinājumi attīstīt paņēmienus, kas ļautu dabiski ierakstīt cilvēka izpildītas kustības un izteikt tās formā, ko var izmantot robota apmācībai. Viens no virzieniem, kurā veikta izpēte, ir tieša fiziskās kinemātikas ierakstīšana un pārveidošana. “*Imitation learning in industrial robots: a kinematics based trajectory generation framework*” [31] (2017) ir piemērs šādam pētījumam. Tieki izmantota *Microsoft Kinect* – savulaik populāra, videospēļu vadībai paredzēta kustību uztveres ierīce, kas spēj ierakstīt kāda objekta kustību aprakstošu punktu virknī telpā – lai ierakstītu cilvēka kustības. Tad ar analītiskām metodēm tiek veikta trajektoriju interpolācija, grupēšana ar klasterizācijas algoritmu, un izpildē vēlamo trajektoriju ieguve ar tuvāko kaimiņu metodēm.

Lai mazinātu atšķirības starp cilvēkam un robotam pieejamām novērojumu un darbību telpām, var izmantot virtuālo realitāti – gan simulācijās, gan robota tālvadībai. “*Deep imitation learning for complex manipulation tasks from virtual reality teleoperation*” [81] (2018) ir aprakstīta fiziska robota tālvadības sistēma, kas robota novēroto vidi pārveido sintētiskā telpā, kur cilvēks var ar manipulatoru palīdzību intuitīvi vadīt robota kustības. Šis uzdevums nav gluži triviāls, jo jāpārvar atšķirības starp, piemēram, robota kameras un cilvēka galvas kustību ātrumu, lai neradītu diskomfortu lietotājam. Tādējādi iegūta demonstrāciju datu kopa, kas sastāv no attēliem un telpiskiem dzīļumiem, ko arī



Att. 4: Zīvs satveršanas modeļa apmācība: a) cilvēka radīto demonstrāciju iegūšana; b) datu kopas sintētiska pavairošana; c) telpiska konvolūciju neironu tīkla apmācība; d) veiksmīga zīvs satveršana realitātē. [15]

tālāk var izmantot konvolūciju neironu tīklā, lai realizētu uzvedības klonēšanu.

#### 2.2.4. Video demonstrācijas, perspektīvu pārbīde

Ļoti plašas iespējas demonstrāciju iegūšanai radītu spēja tās iegūt no video datiem un izmantot šādas “redzes” sistēmas arī tiešā vadības uzdevuma risināšanai. Izrādās, ka daži no jau iepriekš aplūkotajiem algoritmiem ir pietiekami vispārīgi, lai tos varētu pielāgot šim uzdevumam. Piemēram, GAN no novērojumiem [75] ir ļoti robusts attiecībā pret ievades datu formu un saturu, ja vien tajos ir iespējams pietiekami labi atšķirt demonstrāciju no citām trajektorijām. Tāpat kā gadījumā, kad strādā ar kinematiku aprakstošiem stāvokļiem, GAN vienkārši tiek apmācīts diskriminēt/generēt trajektoriju vizuālās reprezentācijas, ar samērā labiem rezultātiem.

Brīdī, kad vairs netiek izmantoti robota konfigurāciju aprakstoši novērojumi, rodas jauna problēma – iegūtie novērojumi ir atkarīgi no izmantotās perspektīvas, kas vispārīgā gadījumā var arī nesakrist ar robota vadības algoritmu ievades datus generējošo. *“Imitation from observation: Learning to imitate behaviors from raw video via context translation”* [35] (2018) risina šo atšķirīgo kontekstu sarežģījumu un iegūto modeli izmanto arī rezultātu uzlabošanai kopumā. Pieņemot, ka trajektorijai atbilstošie novērojumi tiek iegūti no dažādām perspektīvām telpā, vispirms tiek apmācīts enkodera-dekodera tipa neironu tīkls, kas iegūst novērojumu vektoriālas reprezentācijas, ko pēc tam iespējams dekodēt par tai pašai sistēmas konfigurācijai atbilstošu novērojumu no jebkuras perspektīvas. Tas ļauj ne tikai tiešā veidā “tulkot” novērojumus no vienas kameras uz citu, bet arī izmantot iegūtās kodētās reprezentācijas Eiklīda distanci no demonstrāciju trajektoriju soliem atbilstošajām kā atalgojuma funkciju stimulētās mašīnmācīšanās algoritmam. Arī intuitīvi skaidrs, ka modelis, kas spēs atjaunot situācijas attēlu kādā perspektīvā, ja zināmi tikai no citas perspektīvas gūtie attēli, kaut kādā zinā sevī ietver visus nozīmīgos konfigurācijas atribūtus.

#### 2.2.5. Datu sintēze, telpiski modeļi

Daudzi uzdevumi, kam tiek izmantoti roboti, sevī ietver manipulāciju ar citiem objektiem, kuru telpisko novietojumu un citus atribūtus ne vienmēr viegli iegūt, un nākas palauties uz netiešiem novērojumiem, ko sagādā, piemēram, datorredzes sistēmas. Ja

objekti ir paredzamā orientācijā un regulāras formas, no šīs problēmas parasti iespējams izvairīties, taču vēl joprojām daudzi pārvietošanas un pozicionēšanas procesi tiek veikti ar cilvēka roku darbu.

“*Teaching a robot to grasp real fish by imitation learning from a human supervisor in virtual reality*” [15] ir labs piemērs šāda tipa pētījumam, kura ietvaros izstrādāts modelis zivju satveršanas uzdevumam. Robotam ir jāspēj no kastes, kurā atrodas vairākas zivis, satvert un pārvietot vienu. Risinājuma pamatā ir trīs idejas:

- 1) zivs pozīcijai un orientācijai piemērota satveršanas punkta un leņķa paredzēšana, izmantojot ar telpisko kameru gūtus punktu mākoņa datus. Tam pielieto konvolūciju neironu tīklu – izmantojot vispārinātas metodes, kas sākumā izmantotas divdimenšionāla attēlu klasifikācijai;
- 2) demonstrāciju ievākšana virtuālās realitātes vidē, kas ļauj cilvēkam viegli un dabiski ģenerēt satvērienus;
- 3) datu kopas sintētiska pavairošana.

Novērojumi gūti, cilvēkam VR vidē brīvā veidā satverot zivis dažādos veidos. Lai viennozīmīgi aprakstītu katru iespējamo satvērumu, izmantoti trīs vektori – pozīcija (punkta), garenās ass orientācija un rotācija ap to. Pēc tam ģenerēts liels skaits zivju izvietojuma konfigurāciju. Lai sintētiskie dati būtu lietojami, nepieciešams nodrošināt, ka satvēruma matrica tiek koriģēta atbilstoši zivs izliekumam, novietojumam un orientācijai. Katrā sagatavotajā sistēmas konfigurācijā katrai zivij sākumā piešķirti visi iespējamie satvēruma vektori, no kuriem atsijāti visi, kas kolīziju dēļ nav sasniedzami. Šie dati tad izmantoti 3-dimensionālā konvolūciju neironu tīkla apmācībai – simulatorā ģenerēti ainai atbilstoši telpiskie attēli, modelis apmācīts kā tipisks klasifikators.

### **2.3. Atdarinošu modeļu vispārināšana un adaptācija**

Ja pieejamas demonstrācijas no eksperta, to atdarināšana tiešā veidā varbūt ir pirmsais solis noderīgu stratēģiju iegūšanā, taču nebūt ne vienīgais, ko iespējams darīt. Līdz šim aplūkotas metodes, kas izrāda zināmu adaptācijas pakāpi, lai precīzāk un robustāk imitētu paraugus, taču vienmēr izdarīts pieņēmums, ka instruktors kādā ziņā optimāli izpildījis uzdevumu. Turklāt visos gadījumos, lai apmācītu sistēmu izpildīt jaunu uzdevumu, nepieciešams veikt intensīvu treniņu ar cilvēka starpniecību.

Šajā apakšnodaļā aprakstīti pētījumi, kuros meklēti veidi, kā apvienot atdarināšanu ar adaptāciju, lai vispārinātu demonstrācijās ietverto informāciju, atvieglotu jaunu uzdevumu programmēšanas procesu vai pārspētu instruktoru sasniegto rezultātu kvalitātē. Lai gan iespējams atrast desmit gadus un vairāk senus pētījumus, kuros, piemēram, izmantojas sarežģītas un tieši robotikas mērķiem specifiskas simboliskās dekompozīcijas metodes [47] – uzsvars šeit tiek likts uz nesenākām un vispārīgākām metodēm, kas attīstītas jau pašreizējā neironu tīklu uzplaukuma periodā.

### 2.3.1. Neoptimālu demonstrāciju uzlabošana

Jau pati atdarinošās mācīšanās pamatnostādne – apgūt stratēģiju problēmas risināšanai, pēc iespējas tuvāk sekojot kādai demonstrāciju kopai – sevī ietver pieņēmumu par demonstrācijas optimalitāti. Taču izrādās, ka iespējams pašu treniņa datu kopu izmantot, lai gūtu informāciju par to, ko instruktors patiesi vēlējies sasniegt, un attiecīgi optimizēt modeli šī slēptā mērķa sasniegšanai.

*“Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations”* [10] (2019) izmanto jau iepriekš aprakstīto inversās stimulētās mācīšanās algoritmu saimi par pamatu jaunai metodei. Tā vietā, lai atrastu atalgojuma funkciju, kas pamato demonstrāciju kopas trajektorijas  $\tau$ , tiek meklēta tāda, kas skaidro to *sakārtojumu*. Tāpēc nepieciešams datu kopu papildināt ar kārtojumu  $\prec$ , lai

$$\tau_i \prec \tau_j \Rightarrow \sum_{s_t \in \tau_i} R(s) \leq \sum_{s_t \in \tau_j} R(s) \quad (2.4)$$

Protams, kārtojums var arī nebūt ideāls – trajektoriju vērtējums var būt subjektīvs vai trokšķains, ja nav zināma īstā atalgojuma funkcija, tāpēc jārēķinās ar klūdainu pāru attiecību pastāvēšanas iespējamību ar klūdas varbūtību  $\epsilon$

$$\exists \epsilon \geq 0 : \tau_i \prec \tau_j \Rightarrow P \left( \sum_{s_t \in \tau_i} R(s) \leq \sum_{s_t \in \tau_j} R(s) \right) = 1 - \epsilon \quad (2.5)$$

Ja kārtojums ir pieejams, radoši nosauktais *Trajectory-ranked Reward EXtrapolation* jeb T-REX algoritms darbojas divos solos:

- 1) izmantojot demonstrāciju datus, tiek apmācīta nezināmās atalgojuma funkcijas  $R(s)$  aproksimācija  $r_\phi(s)$  – parametisks modelis, konkrēti neironu tīkls;
- 2) tāpat kā IRL gadījumā, rekonstruētā atalgojuma funkcija tiek izmantota, lai apmācītu stratēģiju  $\pi_\theta$  ar stimulētās mašīnmācīšanās metodēm.

Eksperimentāli pārbaudīts, ka piemēros, kur zināmas demonstrācijām atbilstošās īstās atalgojuma funkcijas un klūdas varbūtība saglabājas zem 15%, rekonstruētā atalgojuma funkcija  $r_\phi$  daudzos gadījumos labi aproksimē īsto. Turklāt, ja izmantotās demonstrācijas nav optimālas, iegūtā stratēģija  $\pi_\theta$  tās pārspēj.

### 2.3.2. Demonstrācija – sākumpunkts apmācību procesam

Atšķirībā no IRL un augstāk aprakstītā T-REX, ir iespējams uz atdarinošās un stimulētās mācīšanās kombināciju skatīties arī no otras puses – nevis izmantot stimulētās metodes, lai realizētu atdarināšanu, bet gan izmantot demonstrācijas, lai uzlabotu parasto stimulētās mašīnmācīšanās procesu ar zināmu atalgojuma funkciju.

*“Deep Q-learning from demonstrations”* [26] (2018) piedāvā vispārīgu metodi stimulētās mācīšanās procesa inicializācijai ar demonstrāciju datu kopas palīdzību. Šādu pieeju motivē viens no lielākajiem klupšanas akmeņiem RL algoritmu apmācībā – ja atalgojuma funkcijas nenulles vērtības konfigurāciju telpā ir stipri retinātas, kā tas ļoti bieži ir arī dažādos robotikas uzdevumos, tad, praktiski apmācot stratēģiju izpildīt tai uzdoto uzdevumu, nepieciešams ļoti liels skaits treniņa solu – lai šo telpu apstaigātu, atrastu vēlamos

rezultātus un optimizētu trajektoriju caur tai.

Q-mācīšanās gadījumā stratēģiju var uzdot formā  $\pi_{\epsilon Q_\theta}$  ar  $\epsilon, Q_\theta$  tā, ka

$$T(s, a) = P(s' | s, a) \rightarrow Q_\theta(s, a) \approx \mathbb{E}[v(s')] \quad (2.6)$$

– t.i., modelis  $Q_\theta$  aproksimē matemātisko cerību atalgojumam attiecībā iespējamām darbībām (jeb nākamo stāvokļu vērtības)  $v(s')$  – un attiecīgi izvēlas vienu darbību pēc “ $\epsilon$ -alkatīgas” stratēģijas

$$a = \begin{cases} \operatorname{argmax}_a Q(s, a) \text{ ar varbūtību } 1 - \epsilon; \\ \text{nejauši izvēlēta no iespējām ar varbūtību } \epsilon \end{cases} \quad (2.7)$$

Demonstrācijas tad tiek izmantotas, lai apmācītu  $Q_\theta$  vēl pirms notiek jebkāda modeļa mijiedarbība ar vidi, un attiecīgi tas jau uzreiz darbojas daudz kvalitatīvāk nekā nejauši inicializēts modelis. Rezultāti liecina, ka jau sākotnējās iterācijās šāds algoritms var darboties visai labi, kas paver iespējas to izmantot situācijās, kad nav iespējams veikt apmācību simulācijas vidē, bet ir pieejami demonstrāciju dati. Robotikas kontekstā tas nozīmē, ka, ja jāveic apmācība uz fiziska robota, modeļa sagatavošana ar atdarināšanu varētu būt visai noderīgs paņēmiens.

### 2.3.3. Tūlīteja trajektoriju atdarināšana

Līdz šim aplūkotās atdarinošās metodes darbojas ar pieņēmumu, ka mērkis ir no demonstrāciju datu kopas iegūt modeli, kas pēc apmācības spēj izpildīt vienu uzdevumu, un attiecīgi treniņa algoritms neparedz iespēju bez papildus iterācijām atdarināt iepriekš nerēdzētas demonstrācijas – tipiski pieejams tikai pašreizējais sistēmas stāvoklis ievadē un vēlamā darbība izvadē.

Salīdzinot ar bioloģiskām sistēmām, uzreiz ir skaidrs, ka šādi procesi nekad nespēs izdarīt cilvēkam šķietami pašsaprotamo – novērot darbību un uzreiz to atkārtot bez liela skaita treniņa iterāciju. Lai būtu iespējams sasniegt šādu rezultātu, ir nepieciešams nevis modelis, kas ar paraugiem tīcis optimizēts vienas stratēģijas atdarināšanai, bet gan tāds, kas vispārīna pašu atdarināšanas procesu – ievadē sistēmas pašreizējais stāvoklis tiek apvienots ar demonstrāciju, lai iegūtu vēlamo darbību.

“One-Shot Imitation Learning” [14] (2017) šādu rezultātu sasniedz, apmācot sarežģītas uzbūves modeli, kas apmācīts no vienas demonstrācijas paredzēt citas vispārīgā veidā. Tā ievades vektors satur veselu treniņa kopas trajektoriju. Modelis tiek apmācīts uzreiz veselai līdzīgu uzdevumu kopai, nevis tikai vienam konkrētam.

Konkrēti, pētītā uzdevumu saime ir kubisku detaļu kraušana vienai uz otras ar robotu. Novērojumu ģenerēšanas un interpretēšanas problēmas gan tiek apietas, jo sistēmas stāvokli raksturojošie vektori satur gan robota iekšējo konfigurāciju, gan detaļu relatīvās pozīcijas attiecībā pret robota efektoru. Tad, izmantojot konvolūciju un uzmanības mehānismus, dažādu garumu trajektoriju laikrindas tiek reducētas uz vienu sistēmas stāvokļa vektoru, kas paredz piemērotāko darbību attiecībā pret doto demonstrāciju un momentāno sistēmas stāvokli.

### 2.3.4. Nestrukturētas demonstrācijas, plānu veidošana no galamērķiem

Droši vien lielākā atkāpe no “klasiskā” atdarinošā mašīnmācīšanās uzdevuma ir tajos pētījumos, kur tiek atmests pieņēmums, ka ir dotas diskrētas parauga trajektorijas ar zināmu uzdevumu. Tā vietā pat pārraudzītā trajektoriju marķēšanas stadija tiek automatizēta un atstāta apmācības algoritma pārziņā. Tā vietā var novērot, ka, ja mērķis ir iemācīties modeli, kas spēj vispārīgi atrast ceļu no vienas sistēmas konfigurācijas uz citu, pietiek atrast trajektorijas, kuras satur abas. “*Learning Latent Plans from Play*” [37] (2019) ierosina demonstrāciju strukturētas ievākšanas vietā izmantot cilvēka dabisko tendenci spēlēties ar dažādiem objektiem, lai virtuālās realitātes vidē ģenerētu demonstrāciju kopas. Sistēma šajā gadījumā ir virtuālā realitātē modelēta vide, bet pieejamie novērojumi – robota iekšējā konfigurācija un simulēti attēli.

Lai no šādas nestrukturētas informācijas iegūtu praktiski izmantojamas stratēģijas, nepieciešams papildināt stratēģijas formālismu ar mērķa jēdzienu – ne vairs vienkārši atrodot katram stāvoklim piekārtotu  $\pi(s)$ , bet gan parametrizētu pēc vēlamā beigu stāvokļa (*goal*) –  $\pi(s, s_g)$ . Ja pieejamas trajektoriju laikrindas formā  $s_1, s_2, \dots, s_n$ , nav grūti pamanīt, ka jebkura apakšvirkne veido sākuma-gala stāvokļu pāri ar visām to starpā esošajām pārejām.

Tiek piedāvāti divi dažādi modeļu šabloni un apmācības algoritmi, kas spētu iegūt reālas stratēģijas no nestrukturētu datu kopas. Pirmais modelis  $\pi_\theta(s, s_g)$  saņem ievadē pašreizējo un vēlamo stāvokli, un paredz nepieciešamo darbību. Kā šabloni tiek izmantots rekurentais neironu tīkls, treniņa datus veido trajektorijas, un pēdējais trajektorijas stāvoklis kļūst par  $s_g$ .  $\pi_\theta$  optimizē, lai katram  $(s_t, a_t, s_g)$  būtu  $\pi_\theta(s_t, s_g) \approx a_t$ . Taču pastāv problēma – var pastāvēt dažādas trajektorijas starp jebkuriem  $s, s_g$ , kas draud aprūtināt apmācības procesu.

Otrs, sarežģītākais modelis apkaro šo parādību, modelējot plāna jēdzienu – tiek pieņemts, ka katrai trajektorijai  $\tau$  var piekārtot rīcības plānu, visiem iespējamiem rīcības plāniem pastāv vektoriālas reprezentācijas  $z$ , un vieglāk ir vispirms paredzēt atbilstošāko plānu, pēc tam – piemeklēt tam viennozīmīgi piekārtotu trajektoriju. Modelis tad sastāv no trim daļām:

- 1) plānu enkodera  $q_\phi(z|\tau)$ , kas izsaka trajektorijas ar matemātisko cerību un standartnovirzi vektoriem  $\mu_\phi, \sigma_\phi = q_\phi(\tau)$ ;
- 2) plānu selektora  $q_\psi(z|s, s_g)$ , kas analogiski paredz  $\mu_\psi, \sigma_\psi = q_\psi(s, s_g)$  un iegūst no sadalījuma konkrētu vektoru  $z$  diferencējamā veidā (pieskaitot vidējām vērtībām gadījuma lieluma reizinājumu ar standartnovirzi);
- 3) dekodera  $\pi_\theta(z, s, s_g)$ , kas rekonstruē nepieciešamo darbību no plāna reprezentācijas, galamērķa un sistēmas momentānā stāvokļa.

Divi varbūtību sadalījumi  $q$  nepieciešami, jo optimizācijas procesā tiek minimizēta Kulbaka-Leiblera divergēnce starp tiem – ņaujot visu trajektoriju aizstāt ar tikai diviem tās punktiem. Otrs mērķa funkcijas faktors ir kļūda darbībā  $a_t$ . Tā kā visi slāņi ir diferencējami, viss modelis kopā veido vienu neironu tīklu, ko kopā arī apmāca uz trajektoriju apakšvirknēm. Stratēģija ir  $\pi_\theta(q_\psi(s, s_g), s, s_g)$ .

Tātad, lai programmētu robotu izpildīt kādu konkrētu uzdevumu, vairs nav nepieciešamas ne papildus treniņa iterācijas, ne demonstrācijas – pietiek norādīt vēlamo sistēmas gala stāvokli, un stratēģija to pati sasniegts. Secināts, ka šāda mācīšanās pārspēj klasisko uzvedības klonēšanas metodi 18 dažādiem uzdevumiem.

Interesants turpinājums šim pētījumam ir “*Language conditioned imitation learning over unstructured data*” [36], kur iegūtais veikta vēl viena uzdevuma inversija – ja pastāv metode, kas spēj pati izdalīt uzdevumus un grupēt tos pēc būtības, tad apvienojot to ar sagatavotiem marķējumiem, var iegūt stratēģiju, kas vadāma ar šādiem marķējumiem. Konkrēti, ja dalai no trajektorijām tiek *a posteriori* piekārtots dabiskā valodā izteikts mērķis, var apmācīt modeli, kas sasaista rīcības plānu reprezentācijas ar frāžu reprezentācijām – ļaujot vadīt robotu ar jau iepriekš apmācītu valodas enkoderu ģenerētiem kodiem. Praktiski tas izpaužas tā, ka iespējams robotam dot pavisam dabiskas komandas, turklāt neatkarīgi no izvēlētajiem sinonīmiem un pat dažādās valodās.

Ar tīri atdarinošām metodēm iegūtas stratēģijas var nebūt optimālas, it sevišķi gadījumos, kad tiek prasīts sasniegt mērķi, kam nav sagatavota speciāla demonstrāciju datu kopa. “*Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning*” [20] (2019) piedāvā apvienot nestrukturētas demonstrāciju datu kopas un galamērķus kā uzdevuma specifikāciju ar zināmu atalgojuma funkciju. Tā varētu iegūt labas inicializācijas apmācības procesiem, kas citādi būtu nepraktiski vai teju neiespējami ar vienkāršām konfigurāciju telpas pārstaigāšanas metodēm to retināto atalgojuma vērtību dēļ.

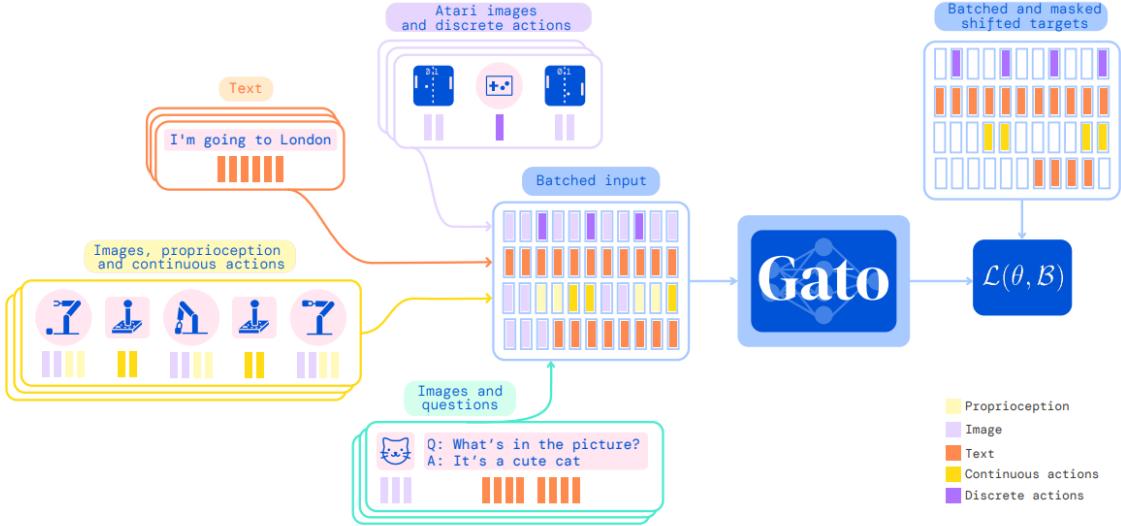
### **2.3.5. ļoti lielu, vispārināmu laikrindu regresoru pielietošana**

Sekojoši līdzi tikai literatūrai, kas uzsver tieši atdarinošās un stimulētās mašīnmaičīšanās nozarē pielietotās terminoloģijas lietojumu, būtu viegli palaist garām problemātikas ziņā tuvu radniecīgus rezultātus, kuru autori līdz tiem nonākuši no citu nozaru pētniecības. MDP formālisms tā klasiskajā formā izslēdz dažādus sarežģītāka izpildījuma autoregresorus, kaut gan tiem iespējams piemeklēt visnotaļ produktīvus pielietojumus arī atdarināšanas uzdevuma risināšanā.

Pavisam nesen radusies iespēja šo darbu papildināt ar pavisam jaunu, ļoti aktuālu publikāciju no DeepMind. “*A Generalist Agent*” ir 2022. gada 12. maijā pirmspublikācijā izplatīts raksts, kura pamatā ir viena laikrindu autoregresora pielietojums dažāda veida sekvenciālu datu atdarināšanai, tai skaitā – atdarinošajam uzdevumam robotikā [55].

Izmantotais modelis ir t.s. transformers jeb pašuzmanības tīkls – pirmo reizi piedāvāts šobrīd jau ļoti slavenajā 2017. gada rakstā “*Attention is all you need*” kā risinājums teksta virķu tulkošanas un autoregresijas uzdevumiem [76]. Tā ir modulāra neironu tīkla arhitektūra, kas sastāv no potenciāli daudziem līdzīgiem transformeru blokiem, kuram katram ir klasisku neironu tīklu un uzmanības mehānisma elementi. Šādus modeļus ar ļoti lieliem parametriem skaitiem pēdējo gadu laikā veiksmīgi izmanto dažādu dabiskās valodas apstrādes uzdevumu risināšanā, kur tie pakāpeniski aizstāj vēl nesen populārās rekurentās tīklu arhitektūras [54].

Konkrētajā pētījumā dažādu modalitāšu ievaddati tiek pārveidoti atbilstošos seman-



Att. 5: *Gato* – vispārināms atdarināšanas aģents, kas ar vienu un to pašu transformatora modeli spēj paredzēt radikāli dažādu sistēmu vektorizētu reprezentāciju laikrindas [55].

tisko markieru vektoros ( “tokens” ), kas pēc tam tiek attēloti modeļa ievadā izmantotajā vektoru telpā ar apmācāmu iegulšanas modeli. Tieki nodalīti novērojumu markieri un darbību markieri. Modelis tiek darbināts kā autoregresors un paredz visus markierus ievada virknē, taču mērķa funkcija tiek piemērota tikai “darbībām” – teksta autoregresijas gadījumā tie būtu visi izvadītie markieri, jo katrs atbilst apakšvārda iegultai formai, bet robotu vadības uzdevumā daļa no markieriem atbilst novērojumam un daļa – darbībai. Atkarībā no uzdevuma problemātikas, novērojuma un darbības reprezentācijai var atbilst dažādu garumu markieru virknes. Par to rekonstrukciju izmantojamā formā atbild katram uzdevumam atsevišķa dekodera sekcija, kas ir inversā operācija iepriekš aprakstītajai markieru kodēšanai un attēlošanai vektoru telpā. Robotu vadības kontekstā apmācībai izmantotas iepriekš sagatavotu stimulētās mācīšanās modeļu sagatavotas demonstrācijas, kas padara šo par klasisku atdarinošās mācīšanās uzdevumu.

Rezultāti liecina, ka šāds modelis ļoti veiksmīgi apgūst ekspertru stratēģijas un sasniedz augstus novērtējumus, piemērojot to apmācībā izmantotās atalgojuma funkcijas – kaut gan pats atdarinošais modelis apmācības procesā tās neizmanto. Tas vieš pārliecību, ka laikrindu paredzēšanas modeli ir perspektīvs virziens tālākai atdarinošo metožu attīstībai.

### 3. PRAKTISKĀ REALIZĀCIJA

Šī maģistra darba ietvaros aptuveni 4 mēneši pavadīti EDI robotikas un mašīnuzvērzes laboratorijā, nodarbojoties ar motivējošā uzdevuma praktiska risinājuma meklēšanu un izstrādi. Rezultātā ar kustības uztveršanas metodēm ierakstītas demonstrāciju datu kopas, veikta to priekšapstrāde un pārveidošana neironu tīklu apmācībai derīgā formātā, apmācīti vairāku veidu modeli pie dažādiem hiperparametriem, veikta to autoregresijas procesā ģenerēto trajektoriju ierakstīšana, vizualizācija un darbināšana uz robota – gan simulācijas vidē, gan uz fiziskas aparatūras.

#### 3.1. Izmantotie rīki, aprīkojums un platformas

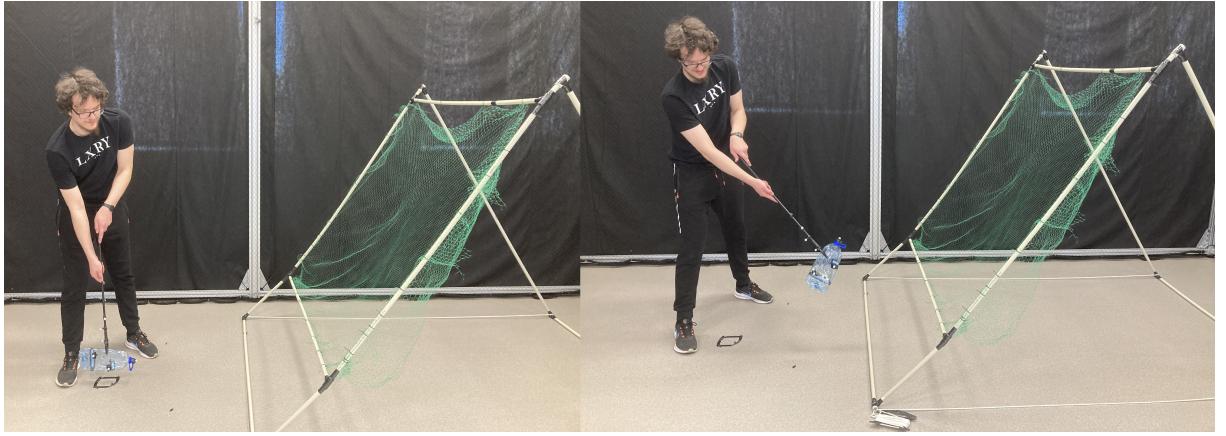
Robotikā tiek integrētas tehnoloģijas no ļoti daudzām nozarēm, tāpēc lietoto rīku ietekme uz darbības virzienu ir ļoti jūtama. Tas, kurus no iepriekšējā nodaļā aprakstītajiem teorētiski iespējamajiem risinājumiem ir iespājams realizēt praksē, ir atkarīgs no pieejamā aprīkojuma un citiem resursiem, tāpēc svarīgi iezīmēt kontekstu, kādā attīstīta darba praktiskā daļa.

##### 3.1.1. ROS

Viens no Robotikas un mašīnuzvērzes laboratorijā veiktās pētniecības stūrakmeniem ir atvērtā koda “*Robot Operating System*” (turpmāk – ROS) platforma [56]. Par spīti nosaukumam, tā ir nevis operētājsistēma, bet gan programmatūras pakotņu kopums, kas paredzēts tiražējamai dažādu robotikas problēmu risināšanai. Šie rīki pieejami pamatā *Linux* saimes operētājsistēmās, turklāt īpaši pielāgoti “*Ubuntu LTS*” versijām. Līdzīgi kā *Python* programmēšanas valodā, šobrīd platforma tiek izplatīta vecākajā ROS1 un jaunākajā, nesaderīgajā ROS2 versijā. Laboratorijā vēl joprojām tiek lietota platformas pirmā versija, tāpēc par to arī tālāk runāts.

Varētu teikt, ka ROS platformas pamatlīdzīgums ir starpprocesu komunikācijas mehānisma nodrošināšana, kam piedāvāti trīs dažādi paveidi [57]:

- 1) Tematu-abonentu modelis (“*topics*”) – paredzēts nepārtrauktu datu plūsmu organizēšanai. Nenoteikts skaits publicētāju, kas jebkurā brīdī var saziņas kanālā publicēt stingri noteikta formāta ziņojumu. Abonenti ir procesi, kas reģistrē atzīmēšanas funkcijas datu saņemšanai. Nav nekādas plūsmas kontroles, un ziņojumi var “izkrist” bez nekādām iespējamām korekcijām;
- 2) Pakalpojumu modelis (“*services*”) – paredzēts momentāniem ārēju funkciju izsaukumiem, kurus nav iespējams pārtraukt. Klienta-servera modelis, kur klienta process uzsāk komunikāciju;
- 3) Darbību modelis (“*actions*”) – paredzēts garāku darbību izpildei, kur nepieciešams sekot līdzi procesa stāvoklim. Arī realizēts kā klienta-servera saziņas process, taču šoreiz jau ar sarežģītāku galīgo automātu, kas nosaka savstarpējo komandu un informācijas apmaiņas modeli, un paredz iespēju ar ārēju izsaukumu pārtraukt (“*preempt*”) uzsāktu darbību.



Att. 6: Demonstrāciju ievākšana kustības uztveres sistēmā *Optitrack*. Robota efektoru simulē atkritumu savākšanas instruments. Tīkls izgatavots, lai netiktu bojāti markieri uz pudeles.

Papildus šai pamatinfrastrukūrai, ROS nodrošina arī pakotņu komplilācijas un instalācijas sistēmu, kā arī ļauj ar vides mainīgo un lietotņu palīdzību viegli apvienot dažādu ROS pakotņu izsaukumus lietotāja skriptos. Pakotnes tiek izstrādātas kā *C++* un/vai *Python* lietojumprogrammas. To izsaukumi tiek kontrolēti ar speciāliem XML formāta palaišanas failiem ( “*Launch Files*”), bieži vien vienlaicīgi izsaucot daudzas izpildāmās programmas ar dinamiski konfigurētiem parametriem [58]. *C++* programmatūras izstrādei pieejamas daudzas bibliotēkas, kam parasti ir arī automātiski generētas *Python* saskarnes.

Apjomīgākās lietojumprogrammas, kas izmantotas šī darba ietvaros, ir telpisku objektu vizualizācijas rīks *rviz*, laikrindu datu kopu attēlotājs *plotjuggler*. Bet saskarni ar robotiem nodrošina to ražotāju izstrādātie robotu dziņi un vairāki abstrakcijas slāņi virs tiem – no kuriem nozīmīgākais ir *MoveIt*, kas nodrošina darbību serveri sazināti ar robotu dziņiem. Jāpiemin, ka arī kustību uztveres sistēma *Optitrack* piedāvā iespēju saņemt reālā laikā straumētos odometrijas datus ar ROS tematu – tā tie arī ierakstīti.

### 3.1.2. *Optitrack – aprīkojums un programmatūra*

Laboratorijā pieejams *Optitrack* kustību uztveres aprīkojums, kas izmantots demonstrāciju trajektoriju ieguvei. Sistēma sastāv no 8 kamerām, kas nosaka īpašu, ļoti atstarojošu markieru pozīciju telpā. Lai varētu izsekot kādu objektu, nepieciešams tam piestiprināt pietiekami daudzus šādus markierus, lai jebkurā iespējamā orientācijā vairākas kameras spētu viennozīmīgi identificēt konkrēto ķermenī un triangulēt tā pozu. Kaut gan ROS kontekstā pieņemts jebkādi novērotas un rekonstruētas ķermenē pozas datus apzīmēt ar terminu “odometrija”, jāpiebilst, ka šis termins gan īstenībā ir attiecīnāms tikai uz braucošu iekārtu pozīcijas aprēķinu, integrējot to riteņu leņķiskos pārvietojumus [43].

Saskarni ar kustību uztveres aprīkojumu nodrošina ražotāja izplatītā *Motive* programmatūras pakotne, kas gan pieejama tikai operētājsistēmā *Windows*. Tāpēc nepieciešama atsevišķa darbstacija šim mērķim, un iegūtie dati tiek straumēti iekšējā tīklā. Lai šiem straumētiem datiem varētu pieķūt ar tematu metodi ROS vidē tiek izmantota



Att. 7: No kreisās pusēs: *Optitrack* aprīkojuma būris; ar markieriem apriņķotie rīki – pirmais tīkls (izmantots kā lidojuma beigu atskaites punkts), efektora simulators, pudele; cietu ķermeņu definīcijas *Motive* programmatūras vidē.

ROS pakotne *mocap\_optitrack*, kas publicē saņemtās datu paketes kā tematu ziņojumus. Ziņojumu ierakstīšanu realizē ar ROS lietotni *rosbag*, kas visus tematā publicētos ziņojumus apkopo izvaddatu failā.

Demonstrāciju ievākšanas mērķiem izstrādāti vairāki fiziski darbarīki, kas redzami 7. attēlā. Pirmkārt nepieciešams reģistrēt pozīciju kādam ķermenim, kas kalpo kā robota efektora simulators. Lai ķermenī būtu iespējams izsekot, tam jābūt nemainīgas formas, un tā unikālais markieru izvietojums jāreģistrē *Motive* lietojumprogrammā (redzams 7. attēlā, pa labi). Turklat attālumiem starp markieriem būtu jābūt pietiekami lieliem, lai tie varētu kalpot objekta atpazīšanai. Tāpēc markieru stiprināšana pie cilvēka rokas tika noraidīta kā potenciāls rīcības plāns.

Tā vietā tika atrasts instruments, kas tuvu aproksimē robota satvērējmehānisma darbību un aprīkots ar markieriem. Tas pats darīts ar plastmasas pudeli. Lai varētu izdarīt metienus, nesalaužot markierus, nepieciešams nokert pudeli. Sākumā izmantots neliels makšķerēšanas tīkls, taču zemās mešanas precizitātes dēļ ar to iespējamo metienu attālums ir visai neliels. Turklat iespējamo trajektoriju dispersija ir neliela. Tā vietā izgatavots vēl viens, daudz lielāks tīkls pudeles ķeršanai. Mazais tīkls bijis aprīkots ar markieriem, lai būtu iespējams noteikt pudeles pozīciju relatīvi tam datu priekšapstrādes procesos. Tāpēc to turpināts izmantot šiem mērķiem arī ar lielo tīklu, vienkārši novietojot to vēlamajā attālumā uz grīdas. Trajektoriju ierakstīšanas process ilustrēts 6. attēlā, kur redzama viena metiena izpilde.

### 3.1.3. Programmēšanas valoda, bibliotēkas

Datu priekšapstrādes, modeļu apmācības, robota vadības, veikspējas novērtējuma mēru un trajektoriju vizualizāciju iegūšanai galvenokārt izmantota programmēšanas valoda *Python*. Pamatā izmantotā tās versija ir 3.8+, kas savietojama ar jaunāko ROS1 platformas versiju (paredzētu *Ubuntu LTS 20.04* operētājsistēmai). Taču robota vadības mērķiem daļa koda pārnesta uz *Python 2.7*, jo to izmanto vecāka ROS1 versija, kas uzstādīta uz robotu kontrolei paredzētās darbstacijas.

Datu korpusi projekta ietvaros saglabāti *.csv* – “comma-separated values” – formāta failos, un to apstrādei lielākoties izmantota bibliotēka *pandas* [45]. Tā sniedz iespēju viegli lasīt un rakstīt *.csv* formāta failus, pārveidojot tos par *DataFrame* objektiem atmiņā.

*DataFrame* ir abstrakcija, kas ļauj ar dažādu tipu informāciju saturošiem divu dimensiju datu korpusiem strādāt funkcionālā vai objektorientētā stilā, ievērojami atvieglojot koda izstrādes procesu. Ľoti reti nepieciešams programmētājam pašam domāt par iterācijas procesiem, jo pieejams plašs klāsts vektorizētu metožu datu kolonnu vai rindu apstrādei – turklāt šīs darbības ir ļoti ātras, jo realizētas jau no bibliotēkas kompilētā mašīnkodā.

Vispārīgām matemātiskām operācijām papildus izmantota *numpy* [42] bibliotēka, kas paredzēta ļoti ātrai skaitlisku un algebrisku operāciju veikšanai ar daudzdimensio-nāliem datu masīviem situācijās, kad tas nav bijis iespējams tikai ar *pandas* pieejamo rīku klāstu. Neironu tīklu modelji izstrādāti ar *tensorflow 2* [72] bibliotēkas palīdzību. Izstrādāta īpaši mašīnmācīšanās lietojumiem, tā piedāvā kompilējamu skaitļošanas grafu un datu tenzoru infrastruktūru. Tieši ar neironu tīklu realizāciju saistītās darbības šajā bibliotēkā ir pieejamas caur augsta līmeņa abstrakcijām, taču vietām tik un tā ne-pieciešams rakstīt kodu ar šajos grafos izmantojamiem tenzoru objektiem, lai nodrošinātu ātrdarbīgu programmas izpildi.

### 3.2. Uzdevuma risinājuma pieeja

Tā kā darba mērķis ir pētīt atdarinošās mašīnmācīšanās metodes un to pielietošanas iespējas praktisku industriālajā robotikā sastopamu vadības problēmu risināšanā, jau no paša sākuma tiek krietni sašaurināts motivējošā uzdevuma potenciālo risinājumu klāsts. Iespējams metienus ģenerēt parametriski, izmantojot t.s. “*model-based*” metodes – kur tiktū speciāli izveidots fizikā un robota kinemātikā balstīts sistēmas modelis, un metieni ģenerēti balstoties uz mērķa koordinātēm. Taču šāda tradicionāla metode ir, pirmkārt, dārga, darbietilpīga un slikti vispārināma, kā jau apspriests darba teorētiskajā daļā, un, otrkārt, nepalīdzētu sasniegt darba pamatmērķi – sniegt ieskatu atdarinošās mašīnmācīšanās metodēs un to praktiska pielietojuma iespējās. Līdzīgi apsvērumi attiecināmi uz stimulēto mācīšanos, kaut gan tādā gadījumā vēl jāņem vērā fakts, ka tehniska re-alizācija būtu pat sarežģītāka un darbietilpīgāka, nekā zemāk aprakstītie paņēmieni.

Strādājot ar atdarinošām metodēm, pirms lēmums, no kā atkarīgs viss tālākais process, ir demonstrāciju iegūšanas metodes izvēle. Atsaucoties uz teorētiskajā daļā pārskatīto literatūru, var izvirzīt vairākus priekšlikumus:

- 1) fiziska vai simulēta robota iekšējo stāvokļu ierakstīšana – demonstrācijas tiek iegū-tas, ar programmu-ekspertu vai (iespējams, simulētu) ārēju fizisku iedarbību vadot robotu pa trajektorijām un saglabājot locītavu pozīciju, ātrumu, paātrinājumu, slodžu datus;
- 2) efektora konfigurāciju ierakstīšana simulācijas vidē – izmantojot VR vai tradicionālu saskarni, izveidot simulācijas vidi, kurā iespējams precīzi reģistrēt vismaz efektora stāvokļa vektoru laikrindas;
- 3) efektora konfigurācija iegūšana no ārējiem novērojumiem – izmantojot kustību uz-tveršanu vai cita veida fizikālās vides novērojumus (piemēram, video), rekonstruēt gala efektora pozīcijas, orientācijas un satvērējmehānisma stāvokļa informāciju no netiešu novērojumu laikrindām.

Vistiešākā pārnese no novērojumiem uz robota kontroles sistēmu, kas darbojas locītavu konfigurāciju telpā, būtu metodēm, kurās jau pašas demonstrācijas sastāv no punktiem šajā telpā. Taču no visām metodēm šādas ir visciešāk piesaistīta konkrētam fizikālam izpildījumam – demonstrācijas ir atkarīgas no konkrētā robota kinemātikas, un nepieciešams izmantot vai nu pašu robotu, vai precīzu tā simulāciju jau treniņa datu kopas ieguvē. Tas prasa lielu sākotnējo laika un darba ieguldījumu šādas vides sagatavošanā, demonstrāciju iegūšanas process kļūst samērā sarežģīts un iegūstamie rezultāti ir slikti vispārināmi.

Ierakstot gala efektora konfigurācijas simulētā vidē nav jāuztraucas par datu kropļojumiem, kas neizbēgami jebkādos no fizikālām sistēmām iegūtos novērojumos. VR cilvēka-datora saskarni padara ļoti tiešu un vienkāršu, ļaujot realizēt gandrīz jebkādas darbības, ko instruktors spētu veikt realitātē. Taču, kaut gan ņemot vērā tikai gala efektoru, tiek ievērojami samazināta datu kopas sasaiste ar kādu konkrētu mehānisku izpildījumu, tik un tā nepieciešams sagatavot adekvātu virtuālu vidi katram uzdevumam.

Fizikālu novērojumu metodēm saikne starp iegūtajiem datiem un robota kontrollera izmantoto informāciju ir visnetiešākā, turklāt jāveic ievērojami pārveidojumi, pirms dati ir pietiekami regulāri, lai tos varētu praktiski izmantot modeļu apmācībā – jārēķinās ar troksni, neregulāru laika parametrizāciju un nepietiekamu sistēmas konfigurācijas aprakstu (ne visus lielumus iespējams tiešā veidā novērot). Pastāv virkne dažādu uztveres metožu un jāmeklē kompromiss starp sistēmas izmaksām/pieejamību un iegūto datu kvalitāti. No aparātūras viedokļa vislētāk būtu izmantot video ierakstu, taču tad priekš-apstrādes procesam vai modelim jāveic ļoti sarežģīts uzdevums, izlobot no netiešiem novērojumiem robota kontrolei aktuālos parametrus. Var izmantot kustības uztveres tehnoloģiju un īpaši izgatavotus instrumentus, lai visnotāl precīzi reģistrētu visus konfigurācijas aspektus – pat papildinot tos ar slēptiem lielumiem kā paātrinājumiem un precīzu satvērējmehānismu vadības signālu stāvokli. Taču kustības uztveres sistēmas ir saistītas ar augstām izmaksām un ne vienmēr būs pieejamas praksē.

Novērtējot Robotikas un mašīnuztveres laboratorijas tehnisko nodrošinājumu un paredzamo lietojumu klāstu – cilvēkam veicamu samērā lielu telpisko izmēru darbību (0,1-1m izmēru diapazona objektu satveršanas, pārvietošanas, metienu, u.c.) atdarināšanu – izdarīta izvēle izmantot laboratorijā pieejamo, augstāk aprakstīto “Optitrack” kustību uztveres aprīkojumu. Ar tā palīdzību iespējams ar samērā augstu precizitāti ierakstīt šāda tipa kustību kinemātiku, un pielāgoties dažādiem uzdevumiem iespējams ātri un vienkārši, piestiprinot markierus manipulatoru atdarinošam instrumentam un citiem objektiem. Trūkums šādai metodei ir sarežģītāks priekšapstrādes process, nekā simulācijas vidē iegūti precīzi novērojumi vai jau ierakstītas robota konfigurāciju laikrindas. Attiecīgi demonstrāciju formāts ir jau pārveidoti novērojumi – efektora odometrija Dekarta koordinātu telpā. To priekšapstrādes procesa gaitā nepieciešams papildināt ar satvērējmehānisma vadības signālu.

Pēc demonstrāciju formāta izvēles, iespējams izdarīt nākamo lēmumu – izvēlēties modelim un robota kontrollerim pieejamo ievades datu formātu. Pastāv divas galvenās

iespējas:

- 1) modelis ģenerē trajektorijas Dekarta koordinātu telpā, kontrolleris veic plānošanu un pārveidošanu;
- 2) modelis ģenerē trajektorijas jau locītavu konfigurāciju telpā, kontrollera uzdevums ir to izpilde;

Realizējot otro, tiktu stipri atvieglošs robota kontrollera uzdevums un nebūtu nepieciešams papildus plānošanas solis. Taču tad nepieciešams priekšapstrādes procesā datu kopu stingri sasaistīt ar konkrētu mehānisku realizāciju (īstā robota kinemātiku), faktiski pārnesot inversās kinemātikas un kustības plāna aprēķinu uz priekšapstrādes soli. Turklāt jārēķinās, ka modeļa uzdevums klūst sarežģītāks – liela daļa lietotāju interesējošo trajektorijas aspektu ir atkarīgi no sistēmas stāvokļa konfigurāciju telpā (piemēram, metiena virziens, attālums, mērķa koordinātes). Tāpēc papildus trajektorijas atdarināšanas uzdevumam tam arī jāspēj veikt attēlošanu starp locītavu konfigurācijas un Dekarta telpām, kas var būt visnotāl netriviāli.

Savukārt izvēloties modeli, kas veic regresiju tikai Dekarta telpā, pats modeļa arhitektūras un apmācības jautājums klūst stipri vienkāršāks. Tam jāspēj tikai ģenerēt demonstrāciju kopai tuvu trajektoriju sadalījumu bez papildu pārveidojumiem, un rezultāts ir tikai netieši sasaistīts ar konkrētu mehānisku izpildījumu – demonstrācijām jāņem vērā robota dinamikas un kinemātikas iespēju ierobežojumi, neprasot neiespējamas pozīcijas, orientācijas, ātrumus vai paātrinājumus. Tomēr šis atvieglojums no mašīnmācīšanās viedokļa noved pie sarežģītāka klasiskā robota kontroles uzdevuma. Nepieciešams iegūtajām telpisko konfigurāciju laikrindām piemeklēt atbilstošu inverso kinemātiku, turklāt bez lēcieniem un, ja metiena precizitātei ir nozīme, ar vismaz aptuveni pareizu laika parametrizāciju (telpas punktiem atbilstošo locītavu konfigurāciju sasniegšana tādā pašā vai tuvā laika momentā) – kas ar ROS platformā pieejamo rīku klāstu nebūt nav vienkārši sasniedzams mērķis.

Izvērtējot darba pamatmērķi (tieši mašīnmācīšanās metožu pētīšanu), pieejamos resursus un laika ierobežojumus, tika izvēlēts modeli konstruēt Dekarta koordinātu telpā kā autoregresoru diskrētā laikā. T.i., katrā “darbība” ir vienkārši paredzētais sistēmas stāvoklis nākamajā laika solī. Iespējami MDP formālismam atbilstoši

$$s_{t+1} = \pi_\theta(s_t) \quad (3.1)$$

vai neatbilstoši (laikrindu ekstrapolācijas)

$$s_{t+1} = \pi_\theta(s_t, s_{t-1}, \dots, s_{t-n}) \quad (3.2)$$

modeļi, un iepriekš nevar spriest, vai MDP stratēģija būs pietiekama konkrēta uzdevuma risināšanai. Tāpēc nolemts sākt ar vienkāršāko MDP atbilstošo stratēģiju klasi – “*behavioural cloning*” pieejā konstruētu klasisku vairāku slāņu neironu tīklu. Kā aprakstīts pie rezultātiem, jau ar šādu risinājumu motivējošajā uzdevumā sasniegsti pozitīvi (bet ne izsmeļoši rezultāti). Metienu precizitātes uzlabojumu meklējumos vispirms papildus attīstīts rekurentais autoregresors – MDP neatbilstošs un izpildes laika ziņā mazāk

efektīvs laikrindu ekstrapolācijas modelis, kas pēc dažiem novērtējumiem sasniedz labākus rezultātus. Tālakas pētnieciskās darbības nolūkos – meklējot veidus, kā ar MDP formālismam atbilstošu, pret trajektorijas garumu laikā un atmiņā konstantu stratēģiju sasniegt RNN pielīdzināmus rezultātus – uzsākta arī GAN modeļa izstrāde.

Trajektoriju izpilde uz fiziskā robota nav tikusi prioritarizēta no pētnieciskā vie-dokļa, un praktiski ierobežojumi pieejamajā Dekarta telpā dotu maršrutu plānošanas programmatūras klāstā nozīmē, ka laikā precīzai izpildei būtu nepieciešams veikt lielu program-mēšanas darbu apjomu, kam nav zinātniskās novitātes. Taču, lai provizoriiski novērtētu iegūto trajektoriju atbilstību robota kinemātiskajiem ierobežojumiem, veikta arī izpildes programmatūras prototipa izstrāde – laika paramatrīzāciju šobrīd iespējams rekonstruēt tikai aptuveni, bet sasniegtās pozīcijas un orientācijas var rekonstruēt precīzi. Rezultāts ir pietiekami labs, lai varētu izpildīt metienu paraugdemonstrācijas un pārliecināties par to, ka demonstrāciju iegūšanas un modeļu apmācības metodes ir adekvātas motivējošā uzdevuma realizācijai ilgtermiņā, un modeļu veikspējas novērtēšanas metrikām ir sasaiste ar generēto trajektoriju telpisko izpildījumu.

### 3.3. Datu priekšapstrāde

Pareizas tīkla un ierakstīšanas programmatūras iestatīšanas gadījumā “Optitrack” rīka iegūtie novērojumi pieejami “ROS” vidē, izmantojot tematu (“topic”) saskarni. Taču, lai no tiem iegūtu demonstrācijas un veiktu modeļu apmācību, nepieciešams veikt virkni priekšapstrādes operāciju. Darba izstrādes gaitā izveidota daļēji automatizēta datu kopu sagatavošanu sistēma, kas balstīta ar “make” sistēmas palīdzību secīgi izsauktos “Python” skriptos. Tie visi pieejami magistra darba “github” repozitorijā, “trajectory\_extract” di-rektorijā [52]. Datu korpusi saglabāti atsevišķā repozitorijā [53].

#### 3.3.1. Ierakstu veikšana, sākuma datu kopas ieguve

Lai saglabātu “ROS” tematā publicētos ziņojumus noteiktā laika periodā, var izmantot “rosbag” lietotni ar “record” komandu. Argumentos iespējams norādīt, tieši kurus tematu saturu nepieciešams ierakstīt – ietaupot vietu uzglabāšanas atmiņā. Šajā gadījumā nepieciešamā informācija pieejama augstāk aprakstīto instrumentu – efekторa simulatora (“TrashPickup”), ar markieriem aprīkotās pudeles (“Bottle”) un brīvā kritiena trajektorijas beigu indikatora (“CatchNet”) – odometrijas datos. Lietotnes izvadā tiek iegūts strukturēts fails “.bag” formātā, kā nolasīšanai nepieciešams izmantot bibliotēku *C++* vai *Python* vidē.

Tāpēc pirmais priekšapstrādes procesa solis ir šī specifiskā formāta datu pārveidošana vispārīgi izmantojamā .csv – “comma separated values” – datu korpusā. To paveic skripts “extract.py”, kas aizņem visilgāko laiku visā priekšapstrādes procesā, jo izmantotā “bagpy” bibliotēka nav sevišķi labi optimizēta ātrdarbības ziņā. Taču šo soli jāveic tikai vienu reizi pēc katra fizisko metienu ierakstīšanas etapa, kas tiek darīts reti, tāpēc veikspējai nav īpaši lielas nozīmes. Datu korpusa iegūšanas procesā no visa ieraksta tiek atlasītas tikai tālākām operācijām svarīgas kolonas.

### 3.3.2. Laika ass reparametrizācija

Pirmā problēma, ar ko nākas saskarties, strādājot ar kustību uztveres procesā iegūtiem kinemātikas datiem, ir neregulārā laika parametrizācija. Jau iepriekš minēts, ka modeli paredzēts izstrādāt kā autoregresoru ar diskrētu laika soli. Ja nepieciešams, lai iegūtā trajektorija spētu atspoguļot ne tikai pozīciju un orientāciju, bet arī to atvasinājums laikā, tam jāatbilst konstantam laika intervālam. Taču no “*Optitrack*” iegūtajos datos pastāv divas nevēlamas parādības:

- 1) katrā novērojuma ietverta tikai vienā odometrijas tematā pieejamā informācija – t.i., katrā laika momentā zināma tikai viena no triju izsekoto ķermeņu konfigurācijām, pārējās nav zināmas;
- 2) intervāli starp novērojumiem ir nejauši, un faktiski sastāv no īsām (dažu milisekunžu) garām virknēm, kurās saņemti novērojumi par vienu no ķermeņiem – veidojot neregulāra garuma apakšvirknes ar informāciju tikai par vienu no ķermeņiem; Abu rezultātā, saņemot datu virknes ilustratīvi pieņem sekojošo formu

$$s_{t1}^a, s_{t2}^a, \dots, s_{tm-1}^b, s_{tm}^b \quad (3.3)$$

, kur  $s^a, s^b$  – dažādus ķermeņu aprakstošie konfigurāciju vektori,  $t_k$  – nejauši laika momenti, kur  $k < j \Rightarrow t_k < t_j$ . Lai datus varētu izmantot tālāk, nepieciešams laikrindu pārveidot formā

$$(s_{t'1}^a, s_{t'1}^b, s_{t'1}^c), \dots, (s_{t'p}^a, s_{t'p}^b, s_{t'p}^c), \dots \quad (3.4)$$

, kur  $t'_k = \frac{n}{f} : n \in \mathbb{N}$  un  $f$  – diskrētā laika soļu frekvence, un katrā laika solī pieejama informācija par visiem ķermeņiem. Attiecīgo pārveidojumu veic skripts “*regularize.py*”. Vispirms datu kopa tiek ielasīta atmiņā no *.csv* faila, izmantojot “*pandas*” datu korpusu apstrādes bibliotēku. Tieki atrasti tuvākie sākuma un beigu laiki

$$t'_1 = \min(\{t'_k \mid t'_k > t_1\}) \quad (3.5)$$

$$t'_p = \min(\{t'_k \mid t'_p > t_m\}) \quad (3.6)$$

, kas pieder diskrēto laika soļu rindai un, izmantojot “*numpy.arange()*” funkciju, tiek ģenerēta diskrēto laika soļu virkne  $t'_1, t'_2, \dots, t'_p$ . Datu korpuiss tiek papildināts ar tukšām rindām šajos laika intervālos (kolīzijas novērstas netiek; dublikāti pēc laika tiek atmesti – kolīzijas varbūtība ir ārkārtīgi zema, strādājot ar peldošā komata skaitļiem). Pēc tam visas tukšās vērtības datu korpusā tiek aizpildītas ar pēdējo zināmo – pēc t.s. “*forward fill*” interpolācijas metodes. Šādi, protams, dati tiek nedaudz kroploji, taču pie pietiekami augstām novērojumu frekvencēm rezultējošās nobīdes ir nelielas un grūti manāmas. Visbeidzot, tikai tās korpusa rindas, kuru laika solis pieder pie  $t'_k$  virknes, tiek saglabātas. Iegūtais datu korpuiss tātad atbilst diskrētam laika solim un tajā nav tukšu vērtību.

### 3.3.3. Atsevišķu demonstrāciju attdalīšana

Tā kā procesa mērķis ir apmācīt modeli, kas spēj izpildīt konkrētu uzdevumu, nevis vienkārši attdarināt visas kustības, kas ierakstītas metienu veikšanas procesā (tai skaitā

atgriešanos sākuma pozīcijā, nejaušu pārvietošanos, u.t.t), nepieciešams visu ierakstu sadalīt konkrētās demonstrācijās, kas katru satur vienu metienu. Vispārīgā gadījumā šis ir netriviāls uzdevums, tāpēc demonstrāciju ievākšana apzināti veikta tā, lai būtu iespējams atrast katra metiena sākumu. Kustību uztveres telpā uz grīdas izvietots markējums sākuma pozīcijai. Katrs metiens sākts, vispirms novietojot efektora simulatoru virs markējuma un noturot vismaz sekundi. Visā pārējā laikā pievērsta īpaša uzmanība, lai efektora simulators netiku novietots šajās koordinātēs. Tādējādi iespējams ar vienkāršu slīdošā loga operāciju atrast vismaz vienu punktu neilgi pirms katra metiena

$$\begin{aligned} t_{start} \in [t'_k]_1^p : t < t_{start} \wedge |t - t_{start}| < \Delta t_{max} \Rightarrow \\ \Rightarrow x_t \in (x_0 - \delta x, x_0 + \delta x), y_t \in (y_0 - \delta y, y_0 + \delta y) \end{aligned} \quad (3.7)$$

, kur sākuma koordinātes  $x_0, y_0$  un pieļaujamie nobīdes intervālu rādiusi  $\delta x, \delta y$  tiek atrasti, cilvēkam pārbaudot ierakstīto datu kopu, atrodot markētā sākuma punkta koordinātes ieraksta koordinātu sistēmā un nosakot, cik lielas nobīdes ļaus precīzi identificēt visu (vai vismaz daudzu) trajektoriju sākuma punktus. Ievērojot pietiekami ilgu intervālu starp metieniem – garāku par  $\frac{steps}{f}$  ar empīriski izvēlētu soļu skaitu  $steps$  – katra demonstrācija tad ir

$$t_j \in [t'_k]_{start}^{start+steps} \quad (3.8)$$

Sekmīgai demonstrāciju iegūšanai ir nepieciešams, katru reizi mainot sākuma marķiera novietojumu telpā vai demonstrāciju iegūšanas stilu (īsākus, garākus laika intervālus pirms/pēc metiena), nomainīt arī sākuma koordinātes  $x_0, y_0$ , sākuma pozīciju intervālu rādiusus  $\delta x, \delta y$  un soļu skaitu  $steps$ . Katra demonstrācija tiek saglabāta atsevišķā .csv failā.

### 3.3.4. Trajektoriju gludināšana

Ar laika ass reparametrizāciju nepietiek, lai novērstu visas neregularitātes iegūtajos datos. Odometrijas dati saņemti sadrumstaloti, turklāt laika izkliede starp “ROS” vidē saņemtajiem punktiem ir lielāka, nekā reālajiem novērojumiem – novērojama pakāpieniem vai zāgim līdzīgu kontūru esamība ierakstos. T.i., fiktīvas augstas frekvences nesinusoidālu svārstību komponentes.

Izvērtējot ierakstīto trajektoriju kvalitatīvos aspektus (salīdzinot līganus metienus video ierakstā un to raustītās reprezentācijas trajektoriju datos), nospriests, ka tās ne-sastāda fiziskajā trajektorijā novērojamu parādību bet gan ieraksta procesā pievienotu datu kroplojumu. Tāpēc pirms modeļa apmācības tās nepieciešams novērst. To paveic skripts “smoothing.py”, kurā piemērotsslīdošās vidējās vērtības aprēķins pēc sekojošā vienādojuma

$$\mathbf{x}'_t = \frac{1}{2m+1} \sum_{i=t-m}^{t+m} x_i \quad (3.9)$$

, kur  $\mathbf{x}$  – novērojuma vektors ar tām vērtībām, kam tiek piemērota gludināšana.

### 3.3.5. Kritisko punktu noteikšana

Šādi ir iegūtas gludinātas, nošķirtas kinemātiku trajektorijas. Taču katrā no tām metiens var sākties pēc dažāda garuma stacionāra perioda, trajektorija satur arī nejaušas kustības pēc metiena beigām, kas traucētu apmācīt modeli, un nav nekādas informācijas par satvērējmehānisma stāvokli (ierakstītas tikai efektora, pudeles un gala stāvokļa markiera odometrijas; sk. sadaļu par kustības uztveres aprīkojumu). Turklat, lai būtu iespējams parametrizēt metienus pēc mērķa koordinātēm, nepieciešams katru demonstrāciju papildināt ar paredzamo pudeles maršruta krustpunktū ar grīdu.

Kritisko punktu meklēšanu veic skripts “*threshold.py*”. Pirmais un vienkāršākais uzdevums ir atrast brīvā kritiena beigu nosacījumu. Brīvā kritiena posms ir nepieciešams nākamajā apakšnodalā aprakstītā regresijas un mērķa koordinātu noteikšanas soļa izpildei. Lai neņemtu vērā pudeles piezemēšanos tīklā, zem tā novietots beigu pozīcijas markēris (vecais satveršanas tīklis, “*Optitrack*” kontekstā – ķermenis “*CatchNet*”). Lidojuma ballistiskā fāze uzskatāma par pabeigtu, kad pudeles  $x$  koordinātē vismaz vienu reizi sasniegusi šo markēri:

$$f_{passed}(t) = \begin{cases} 0 & \text{ja } \forall u < t, x_{Bottle}(u) \geq x_{CatchNet}(u) \\ 1 & \text{citādi} \end{cases} \quad (3.10)$$

Pārējo punktu meklēšanai nepieciešams iegūt pozīciju atvasinājuma aproksimāciju. Diskrētai virknei, protams, atvasinājumi nepastāv. Taču nepārtrauktajai telpiskajai trajektorijai, no kurās iegūti diskrētie novērojumi, atvasinājumi pastāv. Vienkāršākais veids, kā tos atrast, būtu izmantot novērojumu starpības:

$$x'(t) \approx \frac{x(t'_{k+1}) - x(t'_k)}{\Delta t'} = f \cdot (x(t'_{k+1}) - x(t'_k)) \quad (3.11)$$

Ja atvasinājumu absolūtās vērtības konkrētās mērvienībās nav svarīgas, konstanti  $f$  – novērojumu frekvenci, šajā gadījumā 100 Hz – nav nepieciešams ņemt vērā. Empīriski novērots, ka pēc šādas metodes aproksimējot atvasinājumu, katrā solī pieaug svārstību amplitūda. Var aprēķinātajam “atvasinājumam” izmantot to pašu “slīdošās vidējās vērtības metodi”, kas aprakstīta pie gludināšanas. Taču praksē izrādās, ka pietiekami labi strādā sekojošais, stipri vienkāršotais novērtējums, kas apvieno differences un gludināšanas soļus:

$$x'_t \propto \overline{x'_t} = \sum_{i=t}^{t+m} x_i - \sum_{i=t-m}^t x_i \quad (3.12)$$

Piemērojot šo soli divreiz, iespējams atrast arī pozīcijas otrā atvasinājumam – paātrinājumam – proporcionālu vērtību. Lai noteiktu, vai pudele ir brīvajā kritienā, tiek piemērota tāda pati sliekšņa funkcija ar histerēzi – t.i., visas vērtības pēc pirmās, kas ir 1, arī ir 1 – pudeles un efektora simulatora savstarpējās distances atvasinājuma aproksimatoram

$$f_{freefall}(t) = \begin{cases} 0 & \text{ja } \forall u < t, [\|\overline{\mathbf{r}_{Bottle}(u) - \mathbf{r}_{Trash Pickup}(u)}\|]_u' < \bar{v}_{freefall} \\ 1 & \text{citādi} \end{cases} \quad (3.13)$$

, kur  $\mathbf{r}_{Bottle}$  apzīmē pudeles novietojuma vektoru, u.t.t. Lai noteiktu, vai ir sākta satvērēj-mehānisma atlaišana, tas pats tiek darīts ar šī attāluma otrā atvasinājuma aproksimatoru:

$$f_{release}(t) = \begin{cases} 0 & \text{ja } \forall u < t, \overline{\|\mathbf{r}_{Bottle}(u) - \mathbf{r}_{TrashPickup}(u)\|}_u'' < \bar{a}_{release} \\ 1 & \text{citādi} \end{cases} \quad (3.14)$$

Visbeidzot, lai atrastu paša metiena sākuma momentu (ko šajā gadījumā pieņem par ātras kustības sākumu), slieksnis tiek piemērots efektoru simulatora pozīcijas atvasinājuma novērtējumam (ievērojot, ka iepriekš “atvasināta” tiek distances vektora norma, bet šoreiz tiek meklēta norma vektora “atvasinājumam”):

$$f_{moving}(t) = \begin{cases} 0 & \text{ja } \forall u < t, \|\overline{[\mathbf{r}_{TrashPickup}(u)]_u'}\| < \bar{v}_{moving} \\ 1 & \text{citādi} \end{cases} \quad (3.15)$$

Sliekšņa konstantes  $\bar{v}_{freefall}, \bar{a}_{release}, \bar{v}_{moving}$  tiek piemeklētas empīriski. Pieredze liecina, ka process ir diezgan robusts pret šīm vērtībām. Detalizētas informācijas ievākšana par šo konstanšu izvēles ietekmi uz metienu precizitāti ir viens no potenciāliem virzieniem tālākai pētnieciskai darbībai, taču jau pēc vizuālas laikrindu grafiku novērtēšanas izvēlēti skaitļi novērtēti kā pietiekami precīzi, lai būtu iespējams realizēt pudeles metiena paraugdemonstrāciju ar fizisku robotu. Satvērējmeħānisma gadījumā pastāv iespēja iegūt precīzākus datus, papildinot demonstrāciju ierakstīšanas aprīkojumu ar kādu sensoru, kas var nomērīt šīs vērtības tiešā veidā.

### 3.3.6. Brīvā kritiena ekstrapolācija, mērķa koordinātu noteikšana

Demonstrāciju ievākšanas procesā, ja vien netiek ļoti cieši kontrolēti metiena parametri (piemēram, visu laiku metot nelielā mērķī) tīri dabiski rodas ievērojama dispersija pudeles telpiskajās trajektorijās. To varētu pilnībā ignorēt un apmācīt modeli bez papildu brīvības pakāpēm vienkārši atdarināt kādu paraugu no kopas, taču no pētnieciskā viedokļa interesantāk ir jau uzreiz paredzēt iespēju parametrizēt metienus pēc mērķa koordinātēm. Tā kā realizētais kustības uztveres process neparedz demonstrāciju ierakstīšanu ar zināmu galamērķi, šīs koordinātes nav zināmas – tās ir nepieciešams noteikt.

Lai to darītu, izveidots skripts “*regression.py*”, kas veic ekstrapolāciju pudeles trajektorijas brīvā kritiena fāzei un nosaka aptuveno punktu telpā, kur tā šķērsotu grīdas plakni. Par atskaites punktu pieņemta beigu pozīcijas markiera z-koordināte  $z_{ref}$ . Vispirms tiek atrasta demonstrācijas brīvā kritiena fāze pēc nosacījuma

$$\mathcal{D}_{fi} = \{s \in \mathcal{D}_i \mid f_{freefall}(t(s)) \wedge \neg f_{passed}(t(s))\} \quad (3.16)$$

, kur  $\mathcal{D}_i$  – demonstrācijas novērojumu kopa,  $\mathcal{D}_{fi}$  – tās apakškopa, kuras novērojumos pudeles konfigurācijas sastāda ballistisku trajektoriju. Jāņem vērā, ka šādi iegūtā trajektorija tikai aptuveni atbilst īstajai, jo faktiskais pudeles smaguma centrs var atšķirties no kustību uztveres procesā izmantotās cietā ķermeņa definīcijas centroīdas. Taču līdz šim ievāktajās trajektorijās nav manītas lielas nobīdes – tālāk aprakstītās idealizētās regresijas līknes ļoti tuvu atbilst faktiskajām laikrindām. Tātad, pat ja centru nobīdes inducēta

svārstību komponente šajā signālā pastāv, tā ir pārāk neliela, lai to varētu pamānīt – un, visticamāk, pie šobrīd sagaidāmās procesa precizitātes lielu iespaidu uz rezultātu neatstāj.

Lai atrastu mērķa koordinātes, var izmantot vienkāršako ballistiskās trajektorijas modeli. Pieņem, ka gravitācijas paātrinājuma vektors ir precīzi normāls xy plaknei, tāpēc šīm koordinātēm piemēro lineāro regresiju:

$$x_{pred} = \theta_{x1}(t) + \theta_{x0} \quad (3.17)$$

$$y_{pred} = \theta_{y1}(t) + \theta_{y0} \quad (3.18)$$

Savukārt z-koordinātes vērtības ekstrapolē, veicot kvadrātisko regresiju – jeb lineāro regresiju ar divām mainīgā parametra  $t$  (laika) pakāpēm:

$$z_{pred} = \theta_{z2}(t^2) + \theta_{z1}(t) + \theta_{z0} \quad (3.19)$$

Laika momentu, kad kritiens krusto grīdas plakni, var noteikt, atrodot sekojošā vienādojuma pozitīvo sakni:

$$\theta_{z2}(t_{-1}^2) + \theta_{z1}(t_{-1}) + \theta_{z0} - z_{ref} = 0 \quad (3.20)$$

Paksē demonstrācijas ir garākas nekā lidojumi, tāpēc var arī aprēķināt šo vērtību visiem punktiem datu korpusā un atrast pēdējo pozitīvo izteiksmes vērtību. Tas ir asimptotiski lēnāk, taču praktiski “*pandas*” bibliotēkā šādas operācijas ir ļoti ātras. Tad mērķa koordinātes atrod, ievietojot šo laika vērtību regresijas vienādojumos:

$$x_{target} = \theta_{x1}(t_{-1}) + \theta_{x0} \quad (3.21)$$

$$y_{target} = \theta_{y1}(t_{-1}) + \theta_{y0} \quad (3.22)$$

$$z_{target} = \theta_{z2}(t_{-1}^2) + \theta_{z1}(t_{-1}) + \theta_{z0} \quad (3.23)$$

### 3.3.7. Sākuma koordinātu kompensācija

Demonstrācijas sāktas no dažādiem punktiem kustību uztveres rīka koordinātu sistēmā, un tie nesakrīt ar robota efektora sākuma koordinātēm, kas arī ir mainīgas. Tāpēc, lai varētu apmācīt modeli ar vairākās epizodēs iegūtiem novērojumiem, un izmantot šo modeli, vadot robotu, kas nav novietots tādā pašā telpiskā pozīcijā, nepieciešams veikt koordinātu sistēmas centrēšanu.

Pastāv divi varianti, centrēšana pēc metiena sākuma un mērķa. Lai iegūtu metiena sākuma punktu, izmantota iepriekš aprakstītā sliekšņa funkcija:

$$t_{moving} = \min(\{t \mid f_{moving}(t) = 1\}) \quad (3.24)$$

$$\mathbf{r}_0 = \mathbf{r}_{t_{moving}} \quad (3.25)$$

Bet metiena beigu punktu iegūst no mērķa koordinātēm:

$$\mathbf{r}_0 = (x_{target}, y_{target}, z_{target}) \quad (3.26)$$

Tad korekciju ievieš, vienkārši atņemot atskaites pozīciju no novērotajām:

$$\mathbf{r}_{tnorm} = \mathbf{r}_t - \mathbf{r}_0 \quad (3.27)$$

### 3.3.8. Apvienošana, stāvokļu pāreju veidošana, secības jaukšana

Visu iepriekšējo solū rezultātā iegūts liels skaits atsevišķu demonstrāciju, kas katrā ietverta savā failā. Lai tās varētu izmantot modeļa apmācībā, šīs datu korpusa vērtības jāielasa atmiņā tenzora formā. Tā kā kopējais datu apjoms nav ārkārtīgi liels, iespējams izveidot datu kopu, kas ietver visas demonstrācijas uzreiz. Tādu uzdevumu veic automatizētā procesā pēdējais skripts – “*preprocess\_dataset.py*”, kas atrodams repozitorija “*models/*” direktorijā. Šeit arī notiek laikrindas pārveidošana stāvokļu pāreju formātā

$$s_t, s_{t+1}, s_{t+2}, \dots \rightarrow (s, s')_t, (s, s')_{t+1}, \dots \quad (3.28)$$

ar iespēju veidot arī datu kopas, kurās katram rezultējošam stāvoklim zināmi vairāki iepriekšējie. Šī funkcionalitāte ieviesta, sākot strādāt pie nākamā posma – GAN modeļu izstrādes, kur diskriminatoram var būt nepieciešamas garākas virknes:

$$s_t, s_{t+1}, \dots, s_{t+n}, \dots \rightarrow (s, s'^1, s'^2, \dots, s'^n)_t, \dots \quad (3.29)$$

Šajā posmā katrs novērojums tiek reducēts uz tikai modelim nepieciešamo kolonnu apakškopu. Pastāv iespēja pievienot laika signālu, bez kuras sākotnēji veikti modeļu apmācības mēģinājumi. Daļa demonstrāciju tiek ietvertas treniņa datu kopā, daļa – testa un validācijas kopā. Attiecība starp to izmēriem ir iestatāma. Rezultāts tiek sa- glabāts parametriski nosauktos datu kopas *.csv* failos, kuru unikālie identifikatori iegūti, piemērojot jaucējfunkciju korpusa ģenerēšanā izmantoto demonstrāciju failu nosaukumu kopai.

To, vai nepieciešams datu kopu jaukt, nosaka konkrētā modeļu apmācības vai vali- dācijas uzdevuma specifika, tāpēc šis uzdevums tiek atstāts jau nākamajiem soļiem pare- dzēto programmu pārziņā. Failā “*helpers.py*” definētas dažādas funkcijas, kas izmantotas vairākās vietās. Viena no tām ir “*data\_and\_label*”, kas nolasa sagatavoto treniņa datu kopu, sadala to modeļa ievadu (“*input data*”) un vēlamo rezultātu (“*label*”) tenzoros. Ja nepieciešams, šīs kopas elementu secība tiek sajaukta.

## 3.4. Modeļu realizācija

Pateicoties darba ietvaros izmantoto skaitļošanas bibliotēku piedāvātajām iespējām, neironu tīkļu modeļu izstrāde programmatūras kodā ir samērā vienkārša. Par spīti šo modeļu lielajiem parametriem skaitiem un sarežģītajiem aprēķiniem – piemēram, mērķa funkcijas gradientu meklēšanai vai optimizācijas metodēm, kas sastāv no vairākiem, di- namiskiem etapiem – mūsdienās iespējams konfigurēt un apmācīt neironu tīklu ar dažām augsta abstrakcijas līmeņa komandām. Līdz ar to viena pētnieciska darba ietvaros ir bijis iespējams izstrādāt divu dažādu un radikāli atšķirīgu veidu regresorus – klasisku dzīlo

neironu tīklu un rekurento autoregresoru – un uzsākts darbs arī pie viena no tālākiem pētījumiem – MDP formālismam atbilstoša GAN modeļa.

Viss modeļu apmācības kods ir atrodams magistra darba repozitorijā, direktorijā “*models/*”. Tas ir organizēts atsevišķos skriptos. Viena no īpatnībām, ar ko nākas saskarties, strādājot eksperimentālā zinātnes nozarē, ir visu darbību mainīgā un iteratīvā daba. Uzrakstītais kods nav sevišķi apjomīgs, bet provizoriskas izpētes un strukturētu eksperimentu gaitā mainot paņēmienus modeļu uzbūvē, apmācībā, datu priekšapstrādē un izvades failu formātos, nepārtraukti tiek veiktas izmaiņas. Bieži vien pat lielas programmas daļas tiek pilnībā pārrakstītas vai atmestas. Tāpēc katra tipa modeļa apmācībai ir sava skripts, kas satur konfigurācijas mainīgos un visas funkcijas, kas netiek precīzi dublētas visur – pat tad, ja pastāv ievērojamas līdzības starp tām dažādu modeļu izpildījumos.

### 3.4.1. Klasiskais neironu tīkls

Pirmais no izstrādātajiem variantiem un tas, ar kuru strādāts visvairāk, ir klasiskais “*feedforward*” jeb viena virziena dziļais neironu tīkls ar pilnībā savienotiem slēptiem slāņiem. Šī projekta ietvaros šāda pieeja visu pirmkoda un datu failu nosaukumos apzīmēta ar “*naiveBC*” – “*naive behavioural cloning*”, “naivā uzvedības klonēšana”. Tas pilda klasisku MDP stratēģijas (vai lākrindu ekstrapolācijas ar vienu stāvokli garu vēsturi) uzdevumu formā

$$a_t = s'_t = \pi_\theta(s_t) \quad (3.30)$$

Līdz ar to treniņa uzdevuma mērķa funkciju iespējams definēt formā

$$\mathcal{L}_{policy} = f(s, s', \pi_\theta(s)), (s, s') \in \mathcal{D} \quad (3.31)$$

, kur  $\mathcal{D}$  apzīmē konkrētu demonstrāciju. Pastāv dažādas iespējas mērķa funkcijas izvēlē. Tā kā šajā gadījumā modeļa izvads ir nepārtrauktu vērtību vektors, nevar izmantot dažādas kategoriskās kļūdas un varbūtību sadalījumu distances metrikas, kādas izmantotas daudzos no teorētiskajā daļā apskatītajiem pētījumiem. Tas pats sakāms arī par diskretnu darbību kopu stratēģijām paredzētām regularizācijas metodēm. Tā vietā nākas izvēlēties kādu no regresijas mērķiem paredzētajiem novērtējumiem. Pirmā iespēja būtu izmantot klasisko kvadrātu summas metodi, kas ir pamatā jau priekšapstrādes sadaļā apskatītajā lineārās regresijas modeļu ieguvei (atceroties, ka  $s, s'$  apzīmē  $n$ -dimensionālus vektorus):

$$\mathcal{L}_{ss}(s, s', \theta) = \sum_{i=1}^n (s'_i - \pi_\theta(s)_i)^2 \quad (3.32)$$

Taču jau sen pierādīts, ka šī metode nav sevišķi noturīga pret lielām nobīdēm atsevišķu dimensiju dispersijās – izlēcējiem – un piedāvātas alternatīvas mērķa funkcijas dažādu optimizācijas uzdevumu risināšanai, no kurām šeit izvēlēta tās izgudrotāja P. Dž. Hūbera vārdā nosauktā “*huber loss*” funkcija [29]:

$$\mathcal{L}_\delta = \sum_{i=1}^n \begin{cases} \frac{1}{2}(s'_i - \pi_\theta(s)_i)^2, & \text{ja } |(s'_i - \pi_\theta(s)_i)| < \delta \\ \delta \cdot \left(|(s'_i - \pi_\theta(s)_i) - \frac{\delta}{2}| + |(s'_i - \pi_\theta(s)_i) + \frac{\delta}{2}|\right) & \text{citādi} \end{cases} \quad (3.33)$$

Izmantotajā *tensorflow.keras* bibliotēkā šī funkcija ir iebūvēta un pēc noklusējuma iestatīta ar parametru  $\delta = 1$  [69]. Papildus veikti eksperimenti arī īpaši šim uzdevumam konstruētām mērķa funkcijām. Viens no izmēģinātajiem papildinājumiem ir kvaternionu normas regularizācija, kas izmantota, piemēram, cilvēka kustību aprakstošu locītavu orientāciju laikrindu prognozes modeļos, kuru izvadā visi dati veido normētus kvaternionus [48]

$$\mathcal{L}_q(\theta) = \lambda_q * (\|q^\theta\| - 1)^2 \quad (3.34)$$

, kur  $q^\theta$  apzīmē modeļa ģenerētā izvada  $\pi_\theta(s)$  tos elementus, kas kopā veido kvaternionu, bet  $\lambda_q$  – svara koeficientu. Šī regularizācija izmantota, jo pirmajos mēģinājumos apmācīt modeli izmantot kvadrātu summas un Hūbera funkcijas, iegūtās kvaternionu vērtības bijušas ļoti atšķirīgas no 1-normētiem versoriem, kas apzīmē telpisko orientāciju robotu vadības kontekstā.

Vēl viena speciāli šim uzdevumam paredzēta mērķa funkciju saime, kas izmēģināta agrīnās projekta fāzēs, ir tradicionālās regresijas funkcijas papildināšana vai aizstāšana ar dažādu metriku piemērošanu dažādiem izvades vektora elementiem, piemēram

$$\mathcal{L}_{combined}(s, s', \theta) = \lambda_p \|r^\theta - r^{s'}\|^2 + \lambda_q \|q^\theta - q^{s'}\|^2 - \lambda_g g^{s'} \log(g^\theta) \quad (3.35)$$

, kur  $r^\theta, r^{s'}$  apzīmē  $\pi_\theta(s)$  un  $s'$  pārvietojuma vektora komponentes,  $g^\theta, g^{s'}$  – to satvērējmehānisma (“gripper”) konfigurāciju (atvērts/aizvērts), bet  $-x \log(\hat{x})$  ir savstarpējās entropijas mērķa funkcija  $\mathcal{L}_H$ , kas parasti tiek izmantota diskrētas klasifikācijas tipa uzdevumos [68]. Tas darīts ar mērķi atvieglot modeļa apmācību, jau uzreiz nokodējot dažādo izvades vektora elementu savstarpējās sakarības – pirmajos eksperimentos modeļa izvadā iegūtie rezultāti nav pat aptuveni sakrituši ar demonstrāciju kopas datiem.

Taču vēlāki eksperimenti atklājuši, ka abas augstāk minētās problēmas var novērst, vienkārši izmantojot citus hiperparametrus modeļa apmācības procesā (perceptronu skaitu, apmācības ātrumu, treniņa epohu skaitu). Tāpēc visiem rezultātu analīzes sadaļā apskatītajiem “naiveBC” modeļiem izmantota neizmainīta Hūbera funkcija.

Lai izveidotu paša modeļa instanci, izmantota *tensorflow.keras.Sequential* klases piedāvātā augsta līmeņa saskarne. Šīs klases konstruktora metodei argumentā iespējams nodot sarakstu ar *tensorflow.keras.layers.Layer* apakšklašu instancēm, kas katrā apzīmē vienu modeļa slāni. Atgrieztajā vērtībā tad tiek saņemta *tensorflow.keras.Model* klases instance, kuras aprēķinu grafā ietverti visi šie slāņu objekti.

Pēc sākotnējiem izmēģinājumiem konstatēts, ka uzdevumam piemērots ir modelis ar diviem vienāda platuma slēptiem slāņiem, kam katram piemērota “Rectified Linear Unit” nelineārā aktivācijas funkcija [71]:

$$\sigma_{ReLU}(x) = \begin{cases} x, & \text{ja } x > 0 \\ 0 & \end{cases} \quad (3.36)$$

Pilnīgi savienotu slāni *keras* vidē apzīmē klase *keras.layers.Dense*. Aktivācijas funkcijas šajā kontekstā tiek modelētas kā atsevišķi slāņi, un tos pievieno ar *keras.layers.ReLU* instancēm. Ievaddatu formātu definē *keras.layers.Input*, kuras konstruktorā iespējams norādīt ievaddatu formu, bet pēdējā norādītā slāņa perceptronu vektors sastāda modeļa izvadu. Lai izmantotu *keras* iebūvēto treniņa metodi *Model.fit()*, nepieciešams definēt optimizatoru un mērķa funkciju, izsaucot *Model.compile()* metodi. Šajā solī arī tiek noteikti modeļa parametru izmēri un sagatavoti tenzori to vērtībām, ja nav iepriekš atsevišķi norādīta modeļa ievada tenzora forma. Iespējams reģistrēt agrīnas apstāšanās nosacījuma funkciju (*tf.keras.callbacks.EarlyStopping* instanci), lai pārtrauktu treniņa procesu, balstoties uz pārbaudes datu kopā iegūtu modeļa novērtējumu un no-vērstu pārpielāgošanos. Šī funkcija jānodod kopā ar citiem argumentiem – treniņa datu kopu, validācijas datu kopu, epohu skaitu, u.t.t – izsaucot *Model.fit()* funkciju, kas pilnībā automatizē visu modeļa apmācības procesu.

Apmācītais modelis tiek saglabāts ar aprakstošu nosaukumu. Skriptā paredzēta iespēja pārbaudīt, vai pastāv ar atbilstošajiem parametriem jau iepriekš apmācīts modelis pastāv modeļu direktorijā un izlaist treniņa soli. Pēc tam tiek veikti divi validācijas procesa soli – trajektoriju ģenerēšana uz gadījuma mērķa koordinātēm un salīdzinājumi gan ar treniņa, gan testa datu kopām. Šis process detalizēti aprakstīts zemāk.

### 3.4.2. Rekurentais neironu tīkls

Pētot klasiskā neironu tīkla sasniegtos rezultātus, konstatēts, ka dažus trajektorijas parametrus – piemēram, precīzu satvērējmehānisma atlaišanas brīdi – šādam modelim ir grūtības atdarināt. Vēlāk atkārtojot eksperimentus ar citiem hiperparametriem rezultātus izdevies uzlabot, taču pirms tam jau nolemts papildināt darba ietvaros izstrādāto risinājumu klāstu ar sarežģītākas arhitektūras modeli. Atkāpjoties no stingrā MDP formālisma kļuvis skaidrs, ka šādai atdarināšanai pastāv lielas līdzības ar laikrindu ekstrapolāciju. Tāpēc izvēlētā alternatīvā pieeja ir rekurenta autoregresora izstrāde.

Kā jau minēts darba teorētiskajā daļā, pastāv vairākas labi zināmas un visnotaļ spējīgas rekurento tīklu arhitektūras. Divas no tām – LSTM un GRU – iebūvētas *keras* vidē kā *Layer* apakšklases. Papildus iespējams arī realizēt vienkāršu pilnīgi savienotu rekurento tīklu. Šīs klases ietver realizāciju iterācijai, inicializācijai un slēptā stāvokļa izvada vektora padošanai atpakaļ rekurentā modeļa ievadā. Līdz ar to programmētājam nav jādomā par šo sarežģīto sistēmas aspektu – vienīgais, ko nepieciešams precizēt, ir izvadā atgriezto datu formāts. Ar argumenta *return\_sequences* palīdzību tiek norādīts, vai modeļa izvads ir viens vektors, vai vektoru virkne, kuras garums ir vienāds ar ievadā saņemto laika soļu skaitu.

Tā kā atdarinošās mācīšanās kontekstā tiek strādāts ar nelielām datu kopām, un zināms, ka GRU pārspēj LSTM rezultātus šādā situācijā, par modeļa pamatu izvēlēts *layers.GRU* slānis. Analogiski augstāk aprakstītajam klasiskajam neironu tīklam tad

iespējams konstruēt rekurento neironu tīklu. Tāpat kā iepriekš, apmācība ir ļoti vienkārši konfigurējama un izsaucama.

Galvenos sarežģījumus rekurenta modeļa realizācijā rada pareizi strukturētu ievadu sagatavošana. Izņemot īpašo gadījumu, kad tiek strādāts ar vienāda garuma laikrindām, nepieciešams nodrošināt, ka viena no tenzora asīm ir neregulāra garuma. Šeit ļoti noderīgi izrādās nesen ieviestie “robainie tenzori” [70], kas paredzēti īpaši šim lietojumam. Tie nodrošina visu *tensorflow* operāciju saderību ar patiesām neregulāras formas tenzoru – bez aizpildītām fiktīvām vērtībām vai nepieciešamības programmētājam izstrādāt pašam savu treniņa procedūru.

Attiecīgi, izņemot pašu modeļa deklarāciju un pēcāko trajektoriju ģenerēšanu validācijas mērķiem, galvenā atšķirība starp rekurentā un klasiskā tīkla realizācijas skriptiem ir datu kopas sagatavošanas funkcija. Pirmkārt, tā kā rekurentais tīkls uzreiz apstrādā veselu trajektoriju, datu kopas elementus nedrīkst sajaukt – vismaz ne pirms tam, kad iegūtas pilnas trajektorijas. Otrkārt, nepieciešams atšķirt katras trajektorijas sākumu un beigas, lai varētu veidot atsevišķu laikrindu datu tenzorā. Treškārt, nepieciešams ievaddatu laikrindai piekārtot vēlamo vērtību laikrindu, t.i,

$$(s_1, s_2, \dots, s_{n-1}) \rightarrow (s_2, s_3, \dots, s_n) \quad (3.37)$$

Funkcija *create\_sequential\_dataset* veic visus trīs uzdevumus un atgriež divus tenzorus – treniņa ievades datus un vēlamās vērtības. Demonstrētu secību jaukt nav nepieciešams, jo visa datu kopa sadalīta sērijās (“*batches*”), kas katru satur vairākas demonstrācijas. Pie katras no tām tiek vienreiz aprēķināti mērķa funkcijas gradienti un piemēroti modeļa parametriem. Sērijas tiek automātiski jauktas savā starpā. Jaukšanas procedūrai ir lielāka nozīme pie klasiskā neironu tīkla, jo citādi katra sērija, iespējams, saturēs tikai vienas demonstrācijas datus – kas varētu novest pie optimizācijas procesa “lēkāšanas”.

### 3.4.3. GAN (tālāka darbība)

Kaut gan šī darba pētniecisko daļu galvenokārt sastādījusi datu ieguves, priekš-apstrādes, klasiskā un rekurentā modeļa apmācības un validācijas procesu izstrāde, balstoties uz teorētiskajā literatūrā gūtām atzinībām par vienu no galvenajiem virzieniem tālākiem pētījumiem iežīmēta ģeneratīvā pretinieku tīkla realizācija kā paņēmiens, kas varētu apvienot vienkāršu un konstantā laikā izpildāmu stratēģijas modeli ar daudz spējīgāku apmācības metodi. Tāpēc uzsākta arī GAN modeļa realizācijas izstrāde.

Atšķirībā no abiem augstāk minētajiem variantiem, kur iespējams izmantot *keras.Sequential* šablonu klasi, GAN apmācības procesā nepieciešams izmantot divus atsevišķus modeļus un sarežģītāku mērķa funkcijas sakārību ar to parametriem. Līdz ar to nepieciešams citādi strukturēt apmācības programmu.

Vispirms tiek instancēti divi modeļi izmantojot *keras functional API*, kas ļauj konstruēt skaitlošanas grafu ar secīgu slāņu reprezentācijas objektu izsaukšanu. Pirmais modelis – ģeneratorς  $\pi_\theta$ , kas ir arī rezultējošā stratēģija – līdz šim veiktais eksperimentos ir tīcīs strukturēts tāpat, kā klasiskais neironu tīkls. Otrs modelis – diskriminators  $d_\phi$  – kalpo kā mērķa funkcija ģeneratora apmācībā, un ievadā saņem vai nu stāvokļu pāreju,

vai garāku virkni ar stāvokļiem. Neironu tīkla iekšēja struktūra arī pagaidām ir tāda pati, kā klasiskajam neironu tīklam. Galvenā atšķirība abos gadījumos ir tāda, ka izmēģināta arī “*Leaky ReLU*” aktivācijas funkcija, kas ievieš nelielu gradientu citādi plakanos mērķa funkcijas atvasinājuma apgabalos

$$\sigma_{\text{LeakyReLU}}(x) = \begin{cases} x, & \text{ja } x > 0 \\ -k, & 0 < k \ll 1 \end{cases} \quad (3.38)$$

Lai veiktu apmācību, nepieciešams lokāli definēt procedūru. Vienu treniņa soli ietver procedūra *train\_step*, kas, pateicoties *tensorflow.function* dekoratoram, tiek kompilēta uz daudzkārt ātrāku reprezentāciju pirms izpildes. Visas diferencējamās darbības tiek ietvertas *Python* konteksta blokā, kura ietvaros divi *GradientTape* objekti – viens katram modelim – ieraksta mērķa funkciju parciālos atvasinājumus pret to parametriem. Iegūtie gradienti tiek izmantoti, lai izmainītu modeļu parametru vērtības.

Pats diferencējamais etaps sastāv no diviem soļiem. Vispirms tiek izsaukta ģeneratora autoregresijas funkcija (vienu no *generator\_multi\_iterate* versijām atkarībā no tā, vai tiek izmantots viens vai vairāki iepriekšējie stāvokļi). Lai būtu iespējams kompiletēt visas darbības uz ātri izpildāmu grafu reprezentāciju, svarīgi izmantot tikai ar tenzoru saskarni savietojamas datu struktūras. Šī funkcija atgriež tenzoru ar ģeneratora ievadiem un izvadiem

$$s_1 \rightarrow ((s_1, s_2^\theta, \dots, s_m^\theta), \dots, (s_{n-m}^\theta, \dots, s_n^\theta)) = \tau_{gen} \quad (3.39)$$

, kur

$$s_n^\theta = \begin{cases} \pi_\theta(s_1), & \text{ja } n = 2 \\ \pi_\theta(s_{n-1}) \text{ citādi} & \end{cases} \quad (3.40)$$

No treniņa datu kopas tiek sagatavots tādas pašas formas tensors ar ierakstītām stāvokļu pārejām  $\tau_{dem}$ . Diskriminators tiek izsaukts uz abiem tenzoriem ar klasifikācijas uzdevumu

$$d_\phi : (s_1, s_2^\theta, \dots, s_m^\theta) \rightarrow \mathbb{R} \quad (3.41)$$

un tam tiek aprēķināta mērķa funkcijas vērtība

$$\mathcal{L}_d = \sum_{pred \in \tau_{gen}} \mathcal{L}_H(pred, 0) + \sum_{obs \in \tau_{dem}} \mathcal{L}_H(obs, 1) \quad (3.42)$$

, kur  $\mathcal{L}_H$  ir jau iepriekš aprakstītā savstarpējās entropijas funkcija. Tā kā ģeneratora uzdevums ir pretejs, bet tā parametri ietekmē tikai ģenerēto trajektoriju klasifikāciju, mērķa funkcija ir preteja:

$$\mathcal{L}_\pi = \sum_{pred \in \tau_{gen}} \mathcal{L}_H(pred, 1) \quad (3.43)$$

Tāpat kā klasiskā neironu tīkla variantā, izmēģinātas dažu veidu regularizācijas – kvaternionu normas, attāluma starp ģenerētās trajektorijas soliem – taču nopietna to iedarbības izpēte atstāta turpmākai pētnieciskai darbībai.

### 3.5. Validācijas datu ģenerēšana, simulācija, vizualizācija

Populāriem un plaši pētītiem mašīnmācīšanās uzdevumiem – piemēram, attēlu klasifikācijai – pastāv standartizēti modeļa veikspējas mēri un atliek tikai izvēlēties atbilstošo skaitlisko vērtību, kuras vērtībai sekot, apmācot dažādus modeļus [73]. Taču risinot netipiskas vai pat iepriekš neredzētas problēmas ar netriviālu problēmas nostādi, var būt grūti spriest par modeļa kvalitāti pēc virspusējiem skaitliskiem novērtējumiem kā mērķa funkcijas vērtībai.

It sevišķi problēma saasinās situācijās ka šī darba agrīnajās stadijās, kad vēl nav nekādas skaidrības par izvēlēto metožu spēju pat ļoti aptuveni pietuvoties vēlamajiem rezultātiem – nevar vērtēt modeļa spēju sviest pudeli mērķi, ja vēl nav sasniegts modelis, kā ģenerētā trajektorija pat aptuveni izskatās pēc metiena. Tāpēc ļoti svarīgi ir izveidot attīstīt metodes, kas ļauj novērtēt iegūto rezultātu kvalitatīvos aspektus – trajektorijas “formu” telpā, satvērējmeħānisma atlaišanas signāla esamību vai neesamību dažādos to punktos, u.t.t. Tā kā uzdevums pamatā ir kinemātikas atdarināšana, ļoti noderīga ir spēja vizualizēt iegūtos rezultātus – gan kā līknes laikā, gan maršrutus telpā, gan robota kustību simulatorā. Attiecīgi projekta gaitā izstrādātas metodes ģenerētu demonstrāciju ierakstīšanai un attēlošanai.

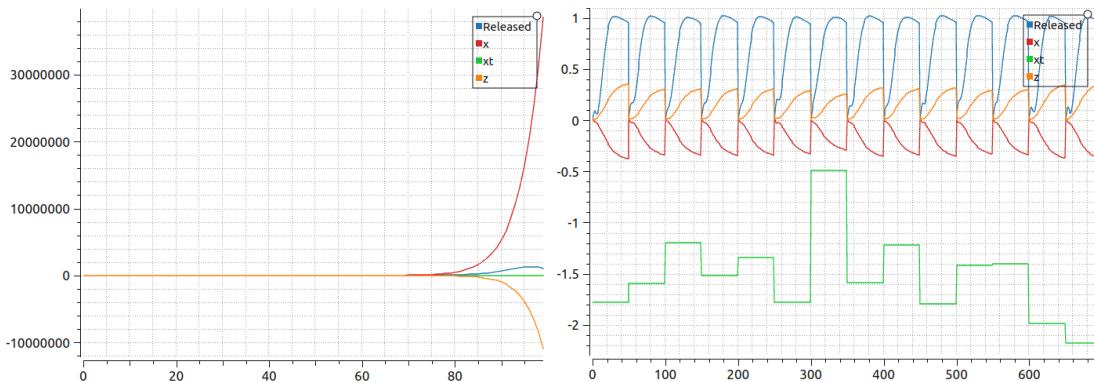
#### 3.5.1. Modelu pārbaude ar gadījuma sākuma stāvokļiem

Pirmā metode, kas darba gaitā izstrādāta, lai aptuveni novērtētu, vai izvēlētie apmācības parametri noved pat pie ļoti aptuvenas vēlamo rezultātu aproksimācijas, ir modeļu autoregresijas rezultātu ierakstīšana pie gadījuma mērķa koordinātēm. Tā balstīta uz pieņēmuma, ka treniņa kopā sastaptās mērķa koordinātes  $x_{target}, y_{target}, z_{target}$  veido gadījuma izlasi no nezināma normālā sadalījuma:

$$\mathbf{r}_{target} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}(\mathbf{r}_{target}), \hat{\boldsymbol{\sigma}}(\mathbf{r}_{target})) \quad (3.44)$$

Tādā gadījumā iespējams ģenerēt pēc vajadzības lielu pārbaudes trajektoriju kopu, veicot gadījuma izlasi no šī sadalījuma ar empīriski nosakāmiem parametriem. Tā kā demonstrāciju dati ir normalizēti pēc sākuma koordinātēm, sākuma pārvietojums vienmēr ir vienāds ar 0. Savukārt sākuma orientācijai pietiek piemeklēt tādu kvaternionu, kas atbilst aptuveni centrētam efektora simulatora stāvoklim.

Dažādiem modeliem ir nedaudz atšķirīgas autoregresijas trajektoriju ģenerēšanas prasības. Visi MDP formālismam atbilstošie modeļi ir ļoti vienkārši izpildāmi – pietiek sākt ar pirmo stāvokli un atkārtoti izsaukt modeli uz tā izvades datiem, līdz sasniegts prasītais trajektorijas garums, tāpat kā vienādojumā (6.39). Šāda tipa autoregresors ir lineārs tā asimptotiskajā sarežģītībā – gan atmiņas, gan laika – pret trajektorijas garumu.



Att. 8: Pa kreisi, trajektorija ierakstīta ar modeli, kam nepareizi definēta mērķa funkcija – novēdot pie autoregresora rezultātu divergences. Pa labi, trajektorija, kuras kvalitatīvie aspekti liecina, ka parametru izvēle ir aptuveni pareiza. Abos gadījumos – atlaišanas signāls, paredzētās  $x$  un  $z$  koordinātes, mērķa  $x$  koordinātes attiecībā pret laika soli.

Savukārt rekurentā neironu tīkla gadījumā sarežģījumus rada fakts, ka visa *tensorflow* saskarne būvēta ap sērijās apkopotu tenzoru apstrādi. Lai nodrošinātu lineāru asimptotisko sarežģītību būtu nepieciešams rakstīt sevis definētu modeļa apakšklasi ar speciālu izsaukuma metodi, kas jau skaitlošanas grafa līmenī pieļauj autoregresiju no viena sākuma ievada. Izmantojot augsta līmeņa *Sequential* šablonu, vienkāršākais veids, kā realizēt autoregresoru, ir atkārtoti izsaukt modeli un katru reizi papildināt tā ievades datu tenzoru ar jaunāko izvadu. Protams, šāda metode ir kvadrātiskas tās asimptotiskajā sarežģītībā izpildes laika ziņā. Taču, tā kā tiek stādāts ar īsām datu virknēm un nelielu kopējo datu apjomu, šī daudzkārt vienkāršākā pieeja arī izvēlēta realizēšanai praksē.

Kad iegūtas autoregresoru ģenerētās trajektorijas, tās nepieciešams apvienot un saglabāt. Tās tiek papildinātas ar mērķa koordinātēm, kvaternionu normu orientācijas elementiem un, gadījumos, kad modelis apmācīts bez laika signāla ievades datos, laika soli. Šim mērķim tiek izmantota *pandas* bibliotēka, un izvades *.csv* formāta faila struktūra ir līdzīgs treniņa datu korpusam. Kā jau minēts, apmācītie modeļi tiek saglabāti, lai būtu iespējams jebkuru no tiem izmantot vizualizācijas vai validācijas datu ieraksts-tīšanai bez atkārtotas apmācības.

Pirms zemāk aprakstīto telpisko vizualizācijas rīku izstrādes, galvenais līdzeklis iegūto rezultātu novērtēšanai bija ROS vidē pieejamā *plotjuggler* lietojumprogramma, kas ļauj uzskatāmi attēlot laikrindu datus *.csv* formātā. Galvenie kvalitatīvie aspekti, kam pievērsta uzmanība šajā agrīnajā projekta fāzē, bijusi trajektorijas  $x, z$  koordinātu līkņu forma atkarībā no  $x_{target}$  parametra, atlaišanas signāla pārejas moments un vērtība (vai vispār tiek komandēta atlaišana? Vai novērojama tās atkarība no metiena attāluma?) un vispārīgi nevēlamu pazīmju – pārtraukuma punktu, svārstību – klātbūtnē. Lai gan grūti izdarīt objektīvus spriedumus par cilvēka intuīcijā balstītiem mēriem, tiem ir nesamērojama nozīme procesa atklūdošanā – 8. att. redzamo salīdzinājumu var ņemt kā paraugu situācijai, kad nepieciešami uzskatāmi rezultāti.

### **3.5.2. Validācija – demonstrāciju un autoregresoru salīdzinājumi**

Taču kā jau minēts, ar subjektīviem vizuāliem spriedumiem vien nepietiek, lai būtu iespējams salīdzināt dažādu modeļu arhitektūru un hiperparametru kopu ietekmi uz rezultātiem. Kad pabeigta procesa sākotnējā atklūdošana un iegūti modeļi, kas konverģē uz šķietami derīgiem rezultātiem, nākamais solis ir kvantitatīvu salīdzinājuma rādītāju ieguve. Problēma ar no gadījuma izlases ģenerētām trajektorijām ir objektīvu kvalitātes kritēriju trūkums.

Nemot vērā atdarinošās mašīnmācīšanās (un laikrindu ekstrapolācijas) problēmu pamatnostādnes, modeļa mērķis ir radīt tādas laikrindas konfigurāciju telpā, kas nav atšķi-ramas no tāda slēptā sadalījuma, kura izlase veido demonstrāciju kopu. Nekas nav zināms par pašu šo sadalījumu, un kaut kāda veida mākslīga tā izlases papildināšana ir ekvivalenta šīs pašas problēmas atrisinājumam. Taču pastāv divi teorētiski krasī atšķirīgi, bet praktiskā izpildījuma ziņā tuvi radniecīgi paņēmieni, ko var izmantot, lai izdarītu spriedumus par apmācības procesa spēju izpildīt pamatnostādni:

- 1) novērtēt modeļa spēju aproksimēt pašu treniņu datu kopu. No teorētiskā viedokļa, jebkurš pietiekami liels un “elastīgs” modelis var pilnībā apgūt jebkuru tā apmācībā izmantoto datu kopu, taču praksē tas ne vienmēr izdodas. Var salīdzināt autoregresijas rezultātus ar treniņa kopas demonstrācijām pie tādiem pašiem sākuma parametriem;
- 2) salīdzināt modeli ar izlasi no tā paša sadalījuma, kas veidojis apmācības datu kopu, bet kuras elementi tajā neietilpst. Jebšu, nošķirt pieejamos datus treniņa un validācijas kopās, no kurām tikai pirmā izmantota modeļa parametru iestatīšanai. Šādi tiek gūts priekšstats par iegūtā modeļa vispārināmību nezināmā sadalījuma ietvaros.

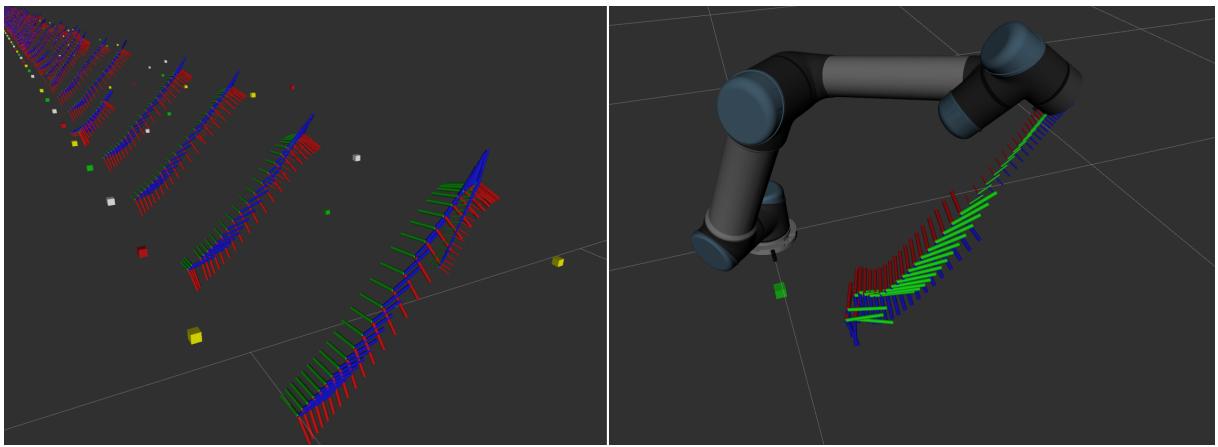
Reālā izpildījuma ziņā abas metodes ir pilnīgi identiskas, atšķiras tikai datu kopa, kas tiek izmantota salīdzināšanai. To veic procedūras ar nosaukumu *validation\_on\_test*, kas dažādajiem neironu tīklu izpildījumiem ir atšķirīgas, taču principā veic to pašu darbību secību. Tieki nolasīta izvēlētais datu korpuiss – apmācības vai validācijas. Tieki nošķirtas visas tā demonstrācijas, un katrai no tām tiek piekārtota atbilstoša tāda paša garuma autoregresora trajektorija un pievienota datu korpusa rindai

$$(s_1, s_2, \dots, s_n) \rightarrow ((s_1, s_1), (s_2, s_2^\theta), \dots, (s_n, s_n^\theta)) \quad (3.45)$$

Apvienotās trajektorijas attalā tiek savietotas vienā datu korpusā, un tas ar modeļa parametriem atbilstošu nosaukumu tiek saglabāts *models/validation/* direktorijs. Visos apmācības skriptos tiek piemērota šī operācija gan ar apmācības, gan pārbaudes datu kopu. Kad pieejamas statistiski salīdzināmas laikrindas, iespējams aprēķināt dažādus skaitliskus novērtējumus to līdzībai – par ko sīkāk diskutēts nākamajā nodaļā.

### **3.5.3. Trajektoriju telpiska vizualizācija**

Ne divdimensionālas līknes, kas katru attēlo vienu konfigurācijas koordināti, ne dažādi abstrakti statistiski skaitliskie rezultāti nesniedz stabili priekšstatu par to, vai iegūtais rezultāts telpā atgādina metienu, vai tas ietilpst robota kinemātisko ierobežojumu



Att. 9: Telpiski attēlotas autoregresora trajektorijas. Pa kreisi vizualizēts korpuiss ar no gadījuma stāvokļiem ģenerētiem metieniem, pa labi – trajektorija, kam piemēroti visi nepieciešamie pārveidojumi, lai tā sakristu ar robota efektora koordinātu sistēmu.

definētajā darba zonā, un vai tas radikāli nepārkāpj robota dinamiskās iespējas. Pirmkārt, šīs iekārtas ir jaudīgas un vērtīgas – nepieciešams ar lielu piesardzību novērtēt jebkuru vadības sistēmu, pirms tiek pielauta to radīto komandu izpilde uz fiziskas aparatūras. Otrkārt, pat, ja pieejama simulācijas vide, dažādu koordinātu sistēmu nesakritības, pārtraukuma punktu klātbūtne un citi faktori var nozīmēt, ka trajektoriju vai nu nav iespējams izpildīt – plānošanas fāzē tiek pārkāpti kādi drošības nosacījumi – vai arī grūti saprast, kas notiek, jo tiek veikti lieli lēcieni konfigurāciju telpā.

Tāpēc izstrādāta programma, kas spēj attēlot vienu vai daudzas pozīcijas un orientācijas vektoru laikrindas 3-dimensiju telpā. Kā svarīgākās prasības šim rīkam noteiktas sekojošās:

- 1) jāspēj viennozīmigi noteikt katru laikrindas punkta telpiskā pozīcija un orientācija robota pamatnes koordinātu sistēmā;
- 2) jābūt iespējai uzreiz vizualizēt daudzas trajektorijas ar tām piekārtotajām mērķa koordinātēm. Lietotājam jāspēj skaidri atšķirt blakus esošie metieni un to mērķi;
- 3) jābūt iespējai savietot šīs vizualizācijas ar robota modeli, lai būtu iespējams tās izmantot dažādu nevēlamu nobīžu atklūdošanā;
- 4) jābūt uzskatāmam veidam kā noteikt, ka padots satvērējmehānisma atlaišanas signāls.

Šiem mērķiem ar Robotikas un mašīnuztveres personāla palīdzību izveidota ROS pakotne *trajectory\_vis*, kas sastāv no viena valodā C++ rakstīta izpildfaila – *visualize.cpp*. Par grafisko attēlošanas vidi izvēlēts ROS platformā iekļautais rīks *rviz*. Šajā rīkā iespējams attēlot patvalīgi novietotus, patvalīgas formas markierus vai to masīvus. Šo markieru iestatīsanai tiek izmantota tematu saskarne. Saskaņi ar robota koordinātu sistēmām un markieru modeļu zīmēšanu nodrošina bibliotēkas *MoveItVisualTools* un *Rviz-VisualTools*.

Izsaucot lietojumprogrammu, tiek norādīti sekojošie argumenti:

- 1) failsistēmas ceļš uz pareizi strukturētu .csv failu, kas satur trajektorijas;

- 2) attēlojamo trajektoriju skaits;
- 3) pirmās trajektorijas sākuma indekss;
- 4) pirmās trajektorijas beigu indekss.

Programma nolasa datu korpusu un atrod tajā pozīcijas un orientācijas vektorus. Šie vektori tiek izmantoti, lai zīmētu koordinātu sistēmas markierus – simbolus, kas parāda, kā tiktu pārveidota pamata koordinātu sistēma, piemērojot tai attiecīgo pārvietojuma vektoru un rotācijas kvaternionu. Katru reizi, kad tiek sasniegts beigu indekss, pārvietojuma vektora  $y$  ass tiek pārbīdīta un skaitīšana tiek atsākta – tādējādi nodrošinot, ka daudzas trajektorijas var tikt attēlotas viena otrai blakus.

To mērķa koordinātes apzīmē ar kuba formas markieriem. Tā kā vizuāli ir grūti nošķirt, kurš beigu markieris sakrīt ar kuru trajektoriju, ieviests vēl viens markieris zem katras sākuma punkta. Katrai trajektorijai atbilstošie markieri iezīmēti vienā krāsā, un secīgi tiek atkārtota virkne ar kontrastējošām krāsām.

Papildus tiek nolasīts arī satvērējmehānisma konfigurācijas parametrs. To, vai satvērējmehānisms tiek uzskatīts par atvērtu, nosaka ar sliekšņa funkciju, kas piemērota tā skaitliskajai vērtībai. Vizuāli, tie punkti laikrindā, kad pudele teorētiski vēl būtu satverta, attēloti ar lielākiem markieriem, bet tie, kuros tā jau ir atlaista – ar samazinātiem.

## **3.6. Trajektoriju izpilde uz robota**

No pētnieciskā viedokļa galvenā praktiskā darba daļa veltīta datu ieguves un modeļu apmācības uzdevumiem. Tā arī veido galveno zinātnisko pienesumu. Taču pēdējais un, iespējams, sarežģītākais uzdevums, kas veikts šī darba ietvaros, ir iegūto rezultātu izpilde uz fiziskas aparātūras. Tādi identificēti daudzi potenciālie šķēršļi iepriekš pētīto datizraces metožu pielietojumam ekspluatācijā un atrasts vismaz viens svarīgs virziens tālākai darbībai. Taču par spīti sastaptajiem sarežģījumiem izdevies realizēt paraug-demonstrāciju – modeļa ģenerētu, uz reālas iekārtas izpildītu metiena kustību, kas veiksmīgi aizmet satvertu pudeli.

### **3.6.1. Modeļa savietošana ar kontrolleri**

Kā jau sīkāk apspriests darba nodalā par robotiem un to vadības problemātiku, ar pozīciju Dekarta koordinātu sistēmā vēl ne tuvu nepietiek, lai būtu iespējams izpildīt kādas robota darbības. Nepieciešams pārveidot telpisku pozīcijas informāciju robota asu dziņu vadības sistēmām saprotamos signālos, kas saistīti ar ātrumiem un paātrinājumiem locītavu konfigurācijas telpā.

Atgriežoties pie izvēlētās modeļa arhitektūras un tā ģenerēto datu formāta, redzams, ka tie sastāda ar nemainīgu frekvenci atjaunotu pozīcijas un orientācijas mērķu virknī. Šāda tipa kontroles signāls sevī ietver visu nepieciešamo informāciju par vadāmo lielumu atvasinājumiem pēc laika, taču netiešā veidā. Darba gaitā aplūkoti augsta līmena paņēmieni veidi, kā pārveidot šādu mērķu secību vadības signālos:

- 1) kontrolleris – pozas un orientācijas sekotājs, kas darbojas reālā laikā un potenciāli veido atgriezenisko saiti ar modeli;

2) pilna kustības plāna aprēķināšana pirms izpildes;

Uzskatāmākā priekšrocība kontrollera realizācijai ir iespēja izmantot modeli jau vairs ne kā tīru autoregresoru, bet ieviest atgriezenisko saiti caur fizikālo vai simuleto izpildes vidi. Kā ievadu nākamā stāvokļa mērķa iegūšanai varētu izmantot pozīcijas un orientācijas faktisko mēriju, iespējams, mazinot sistemātiskas nobīdes starp modeļa inducēto stāvokļu kopu tīrā autoregresijā un reāli ieņemto konfigurāciju sadalījumu. Protams, jārēķinās arī ar iespēju, ka šādas sistemātiskas nobīdes pārāk daudz diverģē no inducētā sadalījuma, uz kura modelis ir apmācīts, un noved pie pasliktinātiem rezultātiem. Taču, ja paredzams kādreiz papildināt sistēmu ar stimulētās mācīšanās soli kā dažos literatūras analīzē aplūkotos pētījumos, atgriezeniskā saite ar vidi ir svarīga. Tāpat nav jāuztraucas par trajektoriju plānotāju piedāvātām laika parametrizācijas iespējām – ja izdevies realizēt pozas sekotāju, ātruma informācija jau automātiski nokodēta vadības signālos.

Veikta šāda kontrollera prototipa izstrāde, izmantojot *MoveIt\_servo* bibliotēku. Tā spēj pārvērst 6-dimensiju komandas, kas sastāv no telpiskā un leņķiskā ātruma vektoriem, robota locītavu vadību signālos bez lieliem lēcieniem konfigurāciju telpā, kamēr vien tas netiek novests tuvu singularitātēm vai kolīzijām. Taču sarežģījumu rada fakts, ka, lai pārvērstu stāvokļu virkni tās atvasinājumiem pielīdzināmos signālos, nepieciešams vēl viens kontrolleru slānis. Izmantojot atsevišķu PID kontrolleri katrai asij, kas ievadā saņem pozīcijas vai leņķa nobīdi novēlamās, to koeficientu atrašana klūst par netriviālu uzdevumu. Tā kā vadības signāli atbilst pozīciju atvasinājumiem, nelielas nobīdes tajos var nozīmēt lielas klūdas robota sasniegtajos novietojumos. Tomēr reāla laika kontrollera izmantošana varētu sniegt labākus rezultātus, strukturējot modeļus tā, lai tie paredz darbību – telpisko ātrumu un paātrinājumu – nevis stāvokļu laikrindu, kas būtu interesants virziens tālākiem pētījumiem.

Savukārt pilna trajektoriju plāna aprēķināšana ir metode, kuras izstrādes process saistīts ar mazākiem riskiem. Pastāv plašs atbalsts šādiem risinājumiem, un ROS kontekstā faktiski visas saskarnes ar robotu kontrolleriem balstītas uz pieņēmuma, ka uzdevums tiek realizēts kā diskrētu, iepriekš plānotu kustību virkne. Lai sagatavotu kustības plānu no punktu virknes Dekarta telpā, pietiek vien izsaukt augsti abstrahētu plānošanas komandu *MoveIt* bibliotēkā.

Galvenā problēma ar plānošanu ROS platformā ir tas, ka, kaut gan ļoti vienkārši precīzēt konkrētiem laika punktiem piekārtotu locītavu konfigurāciju punktu iekļaušanu plānā un izpildi, nepastāv gatavi rīki, kas to pašu darītu ar telpiskiem punktiem. Tāpēc precīzi laikā parametrizēta telpiska plāna izpilde šobrīd nav iespējama – nepieciešams izstrādāt rīku, kas spēj pārrēķināt plāna punktu laika vērtības, lai tās maksimāli tuvu atbilstu laikā parametrizētai telpisku punktu virknei. Tomēr, atšķirībā no kontrollera pieejas, kur nepieciešams pilnīgi strādājošs risinājums, lai kustības ģeometrija atbilstu vēlamajai, ar plānotāju iegūtās trajektorijas tai ir garantēti tuvas. Iespējams piemērot dažus vienkāršus uzlabojumus, lai plānu padarītu mazāk saraustītu un tā kopējo izpildes ilgumu pielīdzinātu prasītajam. Tad var novērtēt, vai modeļa ģenerētā trajektorija ie-

kļaujas robota kinemātiskajos ierobežojumos, un pat izdarīt metienus ar samazinātu pre-cizitāti.

### 3.6.2. Robota trajektorijas plānošana un izpilde

Tīri praktisku ierobežojumu dēļ trajektoriju plānošanas un izpildes programmatūru nācīs radīt ar spēju darboties gan ar modeli, gan iepriekš ierakstītām laikrindām *.csv* failos. Tas tāpēc, ka visa modeļu izstrāde šī projekta ietvaros ir veikta ar *tensorflow* otro versiju, savukārt datori, kas tiek izmantoti fizisko robotu kontrolei, izmanto novecojušu ROS versiju – kuras *Python* versija nav savietojama ar jauno *tensorflow*. Tāpēc sākotnēji simulācijas videi sagatavotais skripts *setpoint\_from\_trajectory.py*, kas pieejams *testpackage/scripts* direktorijā, papildināts ar iespēju nolasīt iepriekš sagatavotus failus un pārveidots saderībai ar *Python 2.7*.

Vadības programmas pamatā ir *MoveIt* bibliotēkas piedāvātā saskarne. Atkarībā no datu avota iespējams vai nu ielādēt iepriekš apmācītu modeli un nogenerēt autoregresora trajektoriju, vai arī tiešā veidā šādu trajektoriju iegūt no faila. Svarīgi, ka atkarībā no izvēlētā robota un kustību uztveres procesā izmantoto cieto ķermeņu konfigurācijas, iespējamas atšķirības starp to izmantotajām koordinātu sistēmām. Tāpēc nepieciešams veikt orientācijas pārveidojumus un novietojuma pārbīdi.

Tiek atrasti orientācijas nobīdes versori

$$q_{offset} = q_r^{-1} q_{base}^{\pi} \quad (3.46)$$

$$q_{restore} = q_{offset}^{-1} \quad (3.47)$$

$q_0^{\pi}$  apzīmē sākuma rotāciju – kvaternionu, kura apzīmētā efektora simulatora orientācija ir ekvivalenta robota neitrālai robota sākuma orientācijai, ko apzīmē ar  $q_r$ .  $q_{offset}$  tad ir nobīde starp šīm orientācijām, ņemot vērā arī atšķirīgo koordinātu sistēmu. Modela izmantošanai nepieciešams pārnest robota sākuma stāvokli uz tā apmācības kopā pielīdzināmu efektora stāvokli

$$s_0^{\pi} = (r_0^{\pi}, q_0^{\pi}, g_0) \quad (3.48)$$

$$r_0 = x_0, y_0, z_0 = 0 = r_0^r - r_0^r \quad (3.49)$$

$$q_0^{\pi} = q_0^r q_{offset} \quad (3.50)$$

$$g_0 = \begin{cases} 1, & \text{ja atvērts} \\ 0 & \end{cases} \quad (3.51)$$

, kur  $r^r, r^{\pi}$  apzīmē robota, modeļa pozīciju,  $q^{\pi}, q^r$  apzīmē to orientācijas, bet  $g_0$  – sākotnējo satvērējmehānisma stāvokli. Iegūto laikrindu pēc tam pārveido atpakaļ uz robota koordinātu sistēmu

$$s_t^r = (r_t^r, q_t^r, g_t) \quad (3.52)$$

$$r_t^r = r_t^{\pi} + r_0^r \quad (3.53)$$

$$q_t^r = q_t^\pi q_{restore} \quad (3.54)$$

Satvērējmehānisma vadības signālam tiek piemērota sliekšņa funkcija ar histerēzi

$$g_t^r = \begin{cases} 0, & \text{ja } \forall u < t, g_u^\pi < \delta \\ 1 & \end{cases} \quad (3.55)$$

Faktiski vienīgais pārveidojums, kas jāveic tiešajā virzienā (no robota uz modeļa sistēmu) ir sākuma orientācija, ja tā atšķiras no neitrālas efektoru orientācijas. Strādājot ar ierakstītu trajektoriju, notiek tikai inversie pārveidojumi (no modeļa uz robota sistēmu).

Lai varētu punktus izmantot trajektorijas plāna ieguvē, nepieciešams no tiem izveidot ROS komunikācijas modeļa robota pozas (pozīcijas un orientācijas) apraksta ziņojuma struktūras *trajectory\_msgs.msg.Pose*. No šādu objektu masīva var iegūt kustības plānu ar *move\_group.compute\_cartesian\_path* metodi, taču šādam plānam ir patvalīga laika parametrizācija. Lai to tuvinātu reālai demonstrācijai, vispirms tiek veikta laika reparametrizācija, kas minimizē telpiskos un leņķiskos paātrinājumus, izmantojot “*Time-Optimal Trajectory Generation*” algoritmu [33]. Pēc tam visi laika intervāli tiek proporcionāli pārveidoti, lai kopējais plāna ilgums sakristu ar trajektorijā paredzēto.

### 3.6.3. Asinhrona satvērējmehānisma vadība, ātrdarbība

Vienkāršākais veids, kā realizēt kustības plāna izpildi, būtu izmantot to pašu *move\_group* saskarni, kas piedāvā *execute* metodi. Tai kā argumentu nepieciešams padot trajektoriju locītavu konfigurācijas telpā, un, iestatot argumentu *wait* uz nepatiesu vērtību, šis izsaukums nav bloķeošs. Taču jāatceras, ka *move\_group* īstenībā sastāv no klienta abstrakcijām *MoveGroup* darbības serverim ROS vidē. Kā jau minēts ROS platformā pieejamo saziņas mehānismu pārskatā, komunikācija ar šāda tipa serveriem seko galīgā automāta modelim, un visas šādas augsta līmeņa komandas bloķē izsaukumus uz serveri, pat ja izsauceja kodā tās ļauj turpināt izpildi, negaidot atbildi ar darbības pieprasījuma rezultātu. Tas nozīmē, ka, pat ja serveris tiktu konfigurēts kontrolēt gan robotu, gan tā darbarīku, nebūtu iespējams ar tā palīdzību veikt pudeles atlaišanu, kamēr robots vēl ir kustībā.

Tāpēc nepieciešams izmantot paralēlus un neatkarīgus ziņapmaiņas kanālus, lai nodotu atlaišanas komandu pareizajā kustības izpildes brīdī. Vienkāršākais variants, kā to varētu mēģināt realizēt, ir noteikt laiku kopš darbības sākuma, kad jāatlaiž pudele, un ar laika aizturi padot komandu. Taču tas būtu ārkārtīgi neprecīzi bez stingrām reālā laika garantijām un ar vērā nemamu aizturi starp kustības pieprasījuma nodošanu un reālās kustības sākumu roblota kontrollerī.

Tāpēc nolemts tiešā veidā izmantot robota locītavu kontrollera piedāvāto darbības serveri – apejot *MoveGroup* – un izmantot pēc robota reālās pozīcijas parametrizētu atzvanīšanas funkciju, lai nodotu satvērēja atlaišanas signālu. Tā tiek izsaukta katru reizi, kad darbības serveris publicē atgriezeniskās saites ziņu. Principā iespējams izsaukuma ietvaros iegūt robota telpisko pozu, taču tas nozīmē, ka jāveic papildu saziņa ziņampaiņas



Att. 10: Paraugdemonstrācija – klasiskā neironu tīkla generēta trajektorija.

sistēmā – ar papildu aizturēm. Tā kā izpildāmās trajektorijas ir ātras – vēlamā atlaišanas signāla izpildes precīzitāte ir ne zemāka kā 0,05 sekundes – nolemts jau iepriekš noteikt vēlamo locītavu konfigurāciju un salīdzināt robota pašreizējo stāvokli ar to. Šī vēlamā konfigurācija tiek noteikta, aprēķinot kustības plānu telpiskajai trajektorijai līdz atlaišanas punktam un ķemot pēdējo. Visas darbības, kas veicamas ar locītavu stāvokļa kontrollera darbību servera palīdzību, apkopotas klasē “*FollowTrajectoryWrapper*”, jo nepieciešama iespēja nodot dinamiskus papildu argumentus atzvanīšanas funkcijai.

Tā kā citu pētījumu ietvaros izvēlētais robots parasti ir aprīkots ar “*Robotiq 2f*” efektoru, kas piedziņai izmanto elektromotorus, saziņu realizē pār *EtherCAT* protokolu un arī kontrolējams ar darbības servera palīdzību, tam izveidota klienta klase *GripperActionWrapper*. Simulācijas vidē šis ir pietiekams risinājums, un ar to izdodas veikt trajektoriju izpildi.

Taču mēģinot veikt paraugdemonstrāciju ar fizisko robotu konstatēts, ka elektromotoru piedziņa nav pietiekami ātrdarbīga, lai paredzami un īstajā trajektorijas punktā veiktu elastīga metamā objekta atlaišanu – pat iestatot maksimālu darbības ātrumu. Var, protams, ieviest manuālas korekcijas programmas kodā – mākslīgi pārbīdīt atlaišanas punktu, to pielāgojot katras trajektorijas formai, vai ieregulējot satveršanas ciešumu tā, lai tas ļoti precīzi atbilstu konkrētajai pudelei un tās pašreizējam tvērienam. Taču tādā gadījumā vairs netiek objektīvi vērtēta modeļa spēja pareizā laika un telpas punktā padot atlaišanas signālu. Lai risinātu šo problēmu, robots aprīkots ar daudz ātrāku pneimatisko satvērējmehānismu. Tā darbībai pietiek ar robota kontrollera digitālā izvada pieslēgšanu 5/2 solenoīdvārstam. *Ur5e* robota kontrolleris nepiedāvā darbības serveri digitālo izvadu vadībai, tāpēc jāizmanto trešā no ROS pieejamajām saziņas metodēm – pakalpojuma serveris. Komunikāciju ar *set-io* pakalpojumu realizē klase *GripperServiceWrapper*.

## 4. REZULTĀTI

Paraugdemonstrācijas uz robota atstāj iespaidu uz skatītājiem no malas un pālīdz popularizēt gūtos rezultātus, taču tās nav darba pašmērkis, un pilnīgi noteikti nav pietiekamas, lai izdarītu objektīvus spriedumus par dažādus datu ieguves un modeļu apmācības procesa aspektu priekšrocībām un trūkumiem. Nepieciešams atrast piemērotus skaitliskus novērtējumus, kas ļauj pēc iespējas vienlīdzīgāk salīdzināt dažādus risinājumus.

### 4.1. Novērtējumi, to aprēķins

Iepriekšējā nodaļā aprakstīts, kā tiek iegūti trajektoriju paraugi validācijai – apmācības un pārbaudes datu kopu demonstrācijas ar tām piekātotām modeļu ģenerētām trajektorijām pie tādiem pašiem sākuma nosacījumiem. Lai aprēķinātu novērtējuma mērus un apkopotu tos vienā datu korpusā, izvedots skripts *generate\_evaluation\_metrics.py*, kas atrodams projekta repozitorijā, *models/* direktorijā.

Tā kā pie katras ar dažādiem parametriem apmācīta modeļa iegūtas divi validācijas datu faili – treniņa un testa – to skaits ir pārāk liels, lai būtu praktiski veikt aprēķinu katram atsevišķi, iestatot nepieciešamos parametrus manuāli. Tāpēc uzrakstītais kods iegūst katru modeli aprakstošo informāciju no saglabātā faila nosaukuma. Tie ir lieumi kā modeļa paveids, parametru skaits, apmācībā izmantotā datu kopa – neilgi pirms darba noslēguma ievākta jauna demonstrācijas datu kopa, kas labāk atbilst robota kinemātiskajiem ierobežojumiem un ļauj praktiski izpildīt trajektorijas, neuztraucoties par pozām ārpus tiem – apmācības epohu skaits, laika signāla esamība vai neesamība (agrīni modeli apmācīti bez tā). Šī informācija var tikt izmantota, lai grupētu un filtrētu rezultātu tabulas rindas analīzes mērķiem.

Lai varētu veikt dažādo novērtējumu aprēķinus, pats datu fails tiek nolasīts, izmantojot *pandas* datu korpusu apstrādes bibliotēku, un dažādās skaitliskās vērtības tiek aprēķinātas pamatā ar tensoru operācijām *numpy* skaitlošanas bibliotēkā.

#### 4.1.1. Datu kopu tuvības mēri

Pirmā skaitlisko novērtējumu kategorija, ko var aprēķināt, lai salīdzinātu jebkādus datus, ir datu kopu tuvības mēri, kas aizgūti no statistikas – Pīrsona korelācijas koeficients, dažādu pakāpju distances metrikas, kosinusu līdzība. Problēma ar šādu metožu izmantošanu ir fakts, ka tās visas ir operācijas, kurām argumentā nepieciešams padot divus vektorus – vienā rindā sakārtotas skaitļu virknes. Efektoru konfigurāciju laikrindas sastāv no vektora katrā laika solī – tātad tās ir matricas no lineārās algebras viedokļa. Šeit pielietotais risinājums ir gaužām vienkāršs – datu kopas sašķeltas demonstrācijas un ģenerētajās matricās. Katra no tām tiek “saplacināta”:

$$\begin{bmatrix} x_{11} & \dots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix} \rightarrow (x_{11}, \dots, x_{n1}, x_{12}, \dots, x_{nm}) \quad (4.1)$$

Tad tām var aprēķināt vektoru līdzības novērtējumus. Pirmais ir statistikā ļoti plaši izmantotais Pīrsona korelācijas koeficients  $r_{xy}$ , kam var atrast vienkāršu vektoriālu izteiksmi, kura izmanto centrētus vektorus  $x^c, y^c$  [9].

$$\bar{x} = \frac{1}{nm} \sum_{i=1}^{nm} x_i \quad (4.2)$$

$$x^c = (x_1 - \bar{x}, x_2 - \bar{x}, \dots, x_{nm} - \bar{x}) \quad (4.3)$$

$$r_{xy} = \frac{x^c \cdot y^c}{\|x^c\| \|y^c\|} \quad (4.4)$$

Āoti līdzīga izteiksme ir kosinusu līdzībai, kas ir vienāda ar 1 paralēliem vektoriem, 0 – ortogonāliem, bet pretejiem ir  $-1$  [50]. Aplūkojot formulu uzreiz arī klūst skaidrs, ka Pīrsona korelācijas koeficienta geometriskā interpretācija ir šī pati kosinusu līdzība, tikai centrētiem vektoriem.

$$d_{cos}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} \quad (4.5)$$

Eiklīda un Manhetenas distance pieder pie distances metrikām – funkcijām kas ir simetriskas un kam izpildās trīsstūra nevienādība [78].

$$d_{euclidean}(x, y) = \|x - y\| \quad (4.6)$$

$$d_{manhattan}(x, y) = \|x - y\|_1 = \sum_{i=1}^{nm} |x_i - y_i| \quad (4.7)$$

Kā redzams, pirmā atbilst telpiskajam attāluma jēdzienam, savukārt otrā vienkārši pilnīgi neatkarīgi novērtē atšķirību katrā vektora dimensijā, tad saskaita visas.

#### 4.1.2. Vidējās novirzes laika soļa ietvaros

Iespējams mērīt ne tikai tuvību starp datu kopām kopumā, bet novērtēt to vidējās, maksimālās, minimālās, u.c. nobīdes katra laikrindas soļa ietvaros. Tieki aprēķinātas vidējās Eiklīda un Manhetenas distances, kur tiek uzreiz novērtēti konfigurāciju vektori  $x_t, y_t$

$$\overline{d_{euclidean}}(x, y) = \frac{1}{n} \sum_{t=1}^n \|x_t - y_t\| \quad (4.8)$$

$$\overline{d_{manhattan}}(x, y) = \frac{1}{n} \sum_{t=1}^n \|x_t - y_t\|_1 \quad (4.9)$$

Eiklīda distanci var aprēķināt arī konkrēti pozīcijas klūdai.

$$\overline{\Delta r}(x, y) = \frac{1}{n} \sum_{t=1}^n \|r_t^x - r_t^y\| \quad (4.10)$$

Savukārt orientācijas klūdai var aprēķināt leņķisko nobīdi starp kvaternioniem [6].

$$\overline{\alpha}(x, y) = \frac{1}{n} \sum_{t=1}^n \arccos (2(q_t^x \cdot q_t^y)^2 - 1) \quad (4.11)$$

Primitīvs atlaišanas signāla kvalitātes novērtējums būtu kategorisks salīdzinājums (vienāds ar 0, ja signāli laika solī sakrīt, bet 1, ja nesakrīt):

$$g'_t = \begin{cases} 1, & \text{ja } g_t > 0,5 \\ 0 & \end{cases} \quad (4.12)$$

$$\overline{\Delta g}(x, y) = \frac{1}{n} \sum_{t=1}^n |g_t^x - g_t^y| \quad (4.13)$$

#### 4.1.3. Metiena parametru aplēšana

Augstāk uzskaitītie vispārīgi pielietojamie rādītāji var būt noderīgi dažādu modeļu apmācības aspektu vērtēšanai, taču, tā kā sasniedzamais rezultāts ir metiena realizācija, iespējams sastādīt daudz specifiskākus modeļa novērtējumu. Pirmās kārtas aproksimācijai var pieņemt, ka metienu pilnībā definē tā palaišanas punkts un ātruma vektors. Lai salīdzinātu ātrumu vektorus, ar kosinusu līdzību vien nepietiek – svarīga ir arī vektoru norma. Vispirms, atceramies, ka katrs validācijas datu korpuiss sastāv no demonstrāciju un ģenerētu trajektoriju pāriem

$$\mathcal{D}_{val} = ((\tau_{d1}, \tau_{g1}), \dots (\tau_{dk}, \tau_{gk})) \quad (4.14)$$

Tāpēc sastādīts sekojošais vienādojums

$$v_t = r_t - r_{t-1} \quad (4.15)$$

$$\overline{\epsilon_v}(\mathcal{D}_{val}) = \frac{1}{k} \sum_{\mathcal{D}_{val}} |\|v_{trd}^d\| - v_{trd}^d \cdot v_{trg}^g| \quad (4.16)$$

, kur  $t_{rd}, t_{rg}$  ir attiecīgi  $\tau_d, \tau_g$  atlaišanas signāla paceļošās frontes laika soli, bet  $v_t$  apzīmē aptuvenos ātrumus (pār-vietojumu diferences). Lai šādam novērtējumam būtu jebkāda nozīme, gadījumos, kad kādā no ģenerētajām trajektorijām  $\tau_g$  atlaišanas signāls nav padots, tas pieņem nedefinētu vērtību.

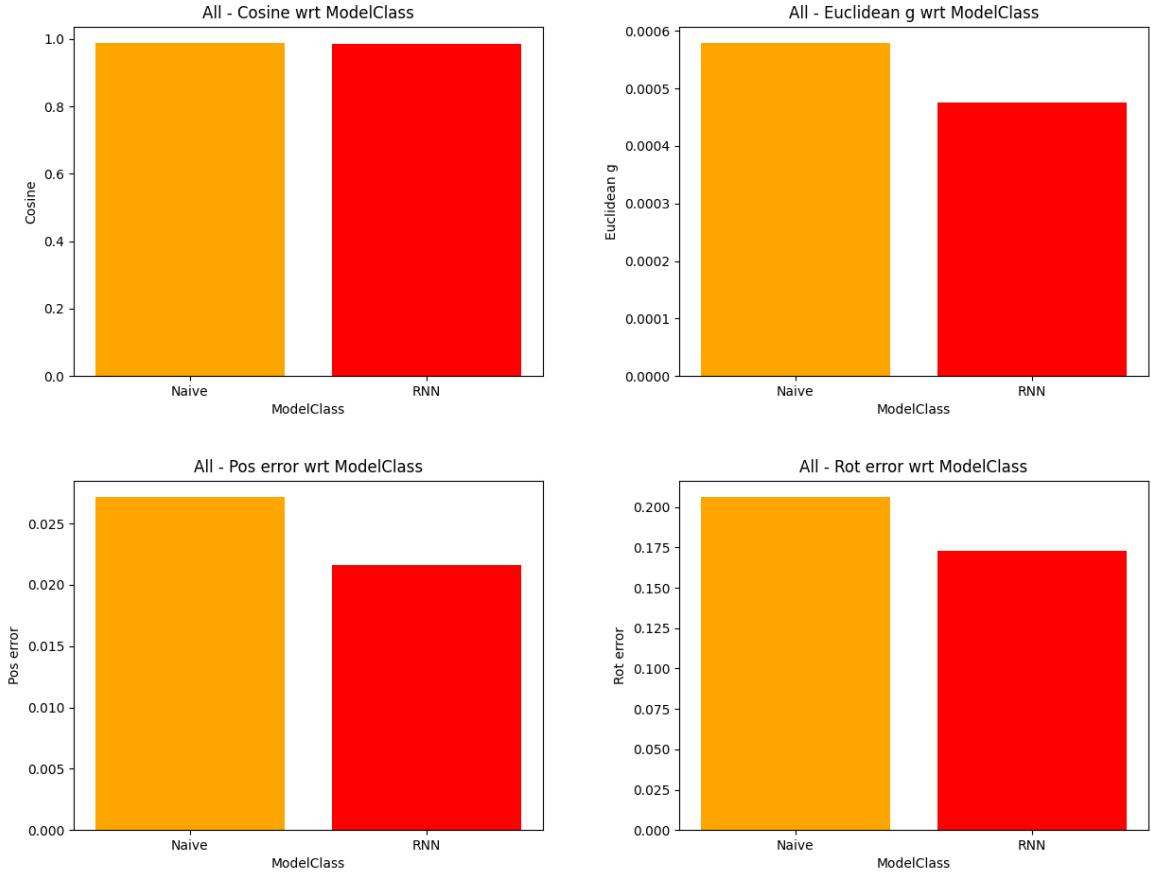
$$\nexists s_t \in \tau_g, g(s_t) = 1 \Rightarrow \overline{\epsilon_v}(\tau_d, \tau_g) \equiv +\infty \quad (4.17)$$

Līdzīgi var novērtēt arī kļūdu palaišanas momenta pārvietojumos.

$$\overline{\epsilon_r}(\mathcal{D}_{val}) = \frac{1}{k} \sum_{\mathcal{D}_{val}} \|r_{trd}^d - r_{trg}^g\| \quad (4.18)$$

Pacilājot jautājumu par to, kā abas šīs vērtības varētu apvienot vienā novērtējumā, secināts, ka tik tālu izdarīto var novest līdz galam un aplēst novirzi starp abu palaišanas ātruma un pozīcijas vektoru pāru definēto brīvā kritiena parabolu krustpunktiem ar mērķa koordinātu horizontālo plakni. Šādā, ģeometriski “simulēt” metienu un novērtēt, cik tālu no demonstrācijas modelis ir trāpījis. No pamatskolas matemātikas zināms, ka brīvo kritieni var aprakstīt ar vienādojumiem

$$x(t) = x_0 + v_x t \quad (4.19)$$



Att. 11: Datu korpusu vispāriņgās tuvības rādītāji (kosinusu līdzība, Eiklīda distance) un vidējās telpiskās/rotācijas novirzes laika soļa ietvaros – RNN un klasiskajam neironu tīklam.

$$y(t) = y_0 + v_y t \quad (4.20)$$

$$z(t) = z_0 + v_{z0}t - \frac{g}{2}t^2 \quad (4.21)$$

, kur  $q \approx 9,81 \frac{m}{s^2}$  ir tipiska gravitācijas paātrinājuma vērtība uz zemes virsmas. Tad lai atrastu  $x, y$  koordinātes parabolas krustpunktam ar plakni, vispirms tiek atrasta pozitīvā sakne  $t_{intersect}$  sekojošam vienādojumam

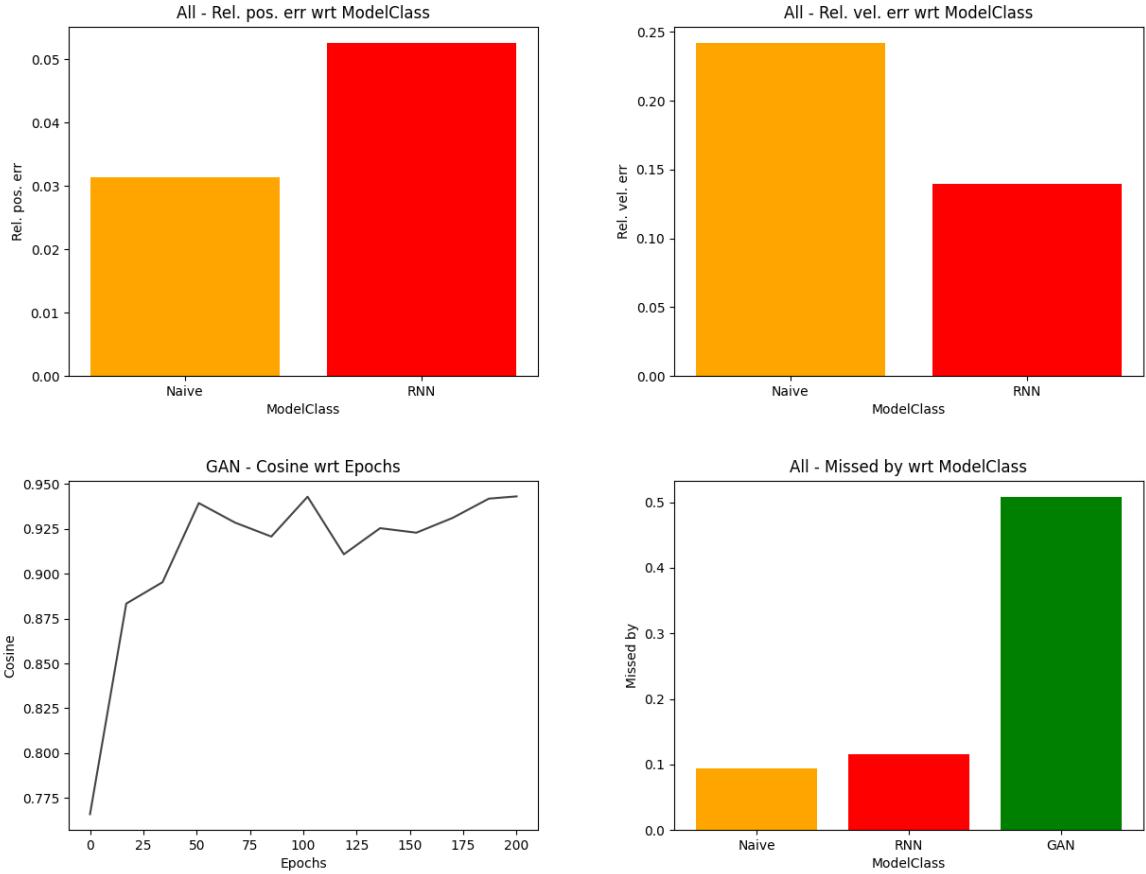
$$z_0 - z_{target} + v_{z0}t_{intersect} - \frac{g}{2}t_{intersect}^2 = 0 \quad (4.22)$$

un ievietota horizontālo koordinātu vienādojumos

$$r_{intersect} = (x(t_{intersect}), y(t_{intersect}), z_{target}) \quad (4.23)$$

Vidējo metiena kļūdu tad var novērtēt kā

$$\overline{\epsilon_{throw}}(\mathcal{D}_{val}) = \frac{1}{k} \sum_{\mathcal{D}_{val}} \|r_{intersect}^d - r_{intersect}^g\| \quad (4.24)$$



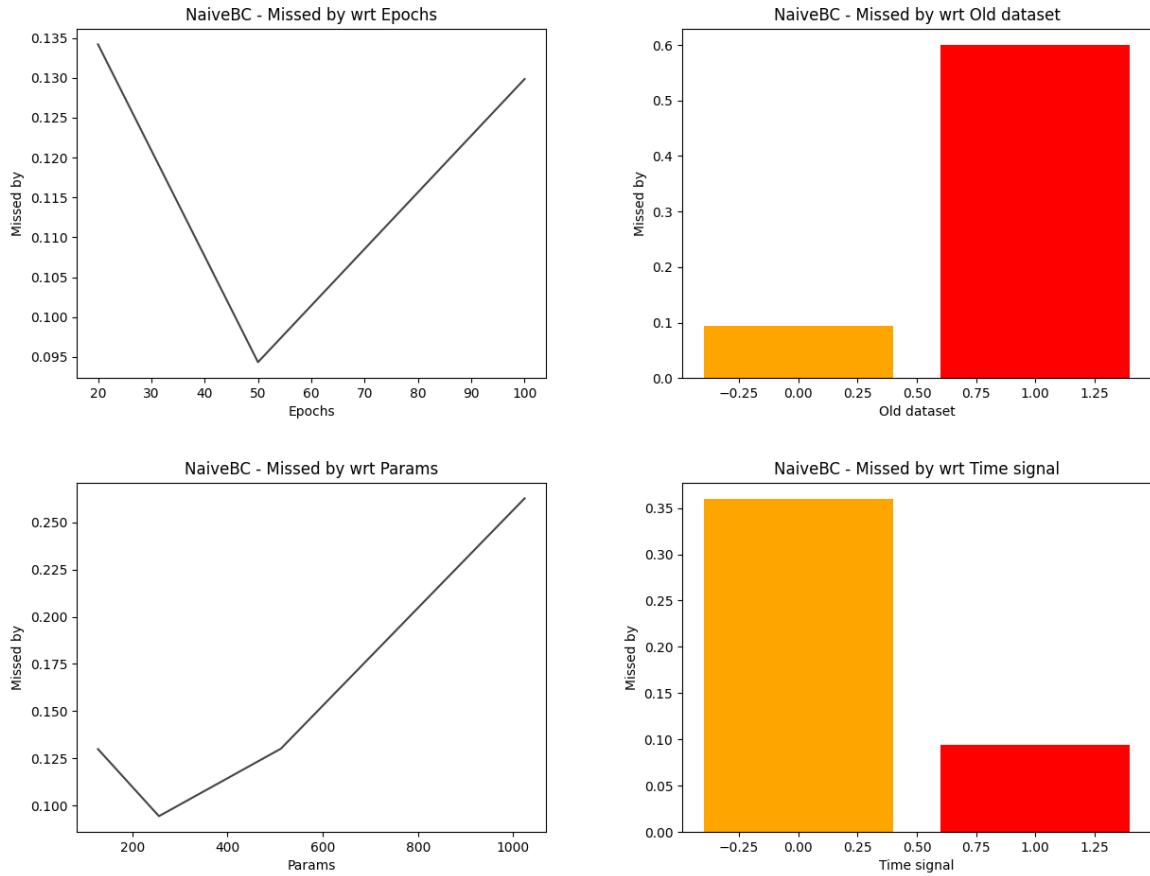
Att. 12: Augšējā rindā – labākās sasniegtās vidējā atlaišanas punkta novietojuma un ātruma vektora noviržu vērtības. Apakšā pa kreisi – pietiek ar jau rudimentāru GAN arhitektūru, lai vispār notiku modeļa konverģence. Tomēr no RNN un klasiskā neuronu tīkla sasniegtajiem rezultātiem tā tomēr atpaliek.

#### 4.1.4. *Grafiku ģenerēšana un saturs*

Visas šajā nodaļā izmantotās diagrammas ir ģenerētas programmatiski divos soļos. Vispirms no iepriekš aprēķinātā metriku datu korpusa tiek izdalītas katras sakarības attēlošanai nepieciešamo datu kopas, kas pēc tam tiek izmantotas grafikus saturošu attēlu ģenerēšanai. Par šiem uzdevumiem atbild darba repozitorija [52] *models*/ direktorijā atrodamie skripti *comparison\_table\_generator.py* un *draw\_plots.py*.

Situācijās, kad tiek salīdzinātas dažādas modeļu klases vai veikti vispārīgi modeļu veiktspējas novērtējumi pie dažādiem hiperparametriem, uzskatāmības labad tiek atrastas novērtējumu maksimālās (vai minimālās, atkarībā no novērtējuma būtības) vērtības pie katra, bez fiksēto hiperparametru kopas. Tas nesniedz tik detalizētu ieskatu apmācības procesa atkarībās no katra argumenta, bet ņauj globāli novērtēt, ar kuru modeli sasniegti labāki rezultāti, vai lielāki modeli sasniedz labākus rezultātus, u.t.t.

Papildus veikta arī detalizēta novērtējumu attēlošana atkarībā no konkrētiem liešumiem, turot pārējos konstantus – taču lielais apmācīto modeļu skaits nozīmē, ka grūti šādus datus attēlot uzskatāmi. Tāpēc ieviests kompromiss – situācijās, kad apskatītas liknes ar daudzām fiksēto argumentu kopām, izvēlētas pa divām “labākajām” – pie lielākās



Att. 13: Klasiskā neironu tīkla ģenerēto trajektoriju simulēto metienu nobīdes atkarībā no: apmācības epohu skaita; jaunās/vecās datu kopas; parametru skaita; laika signāla neesamības/esamības.

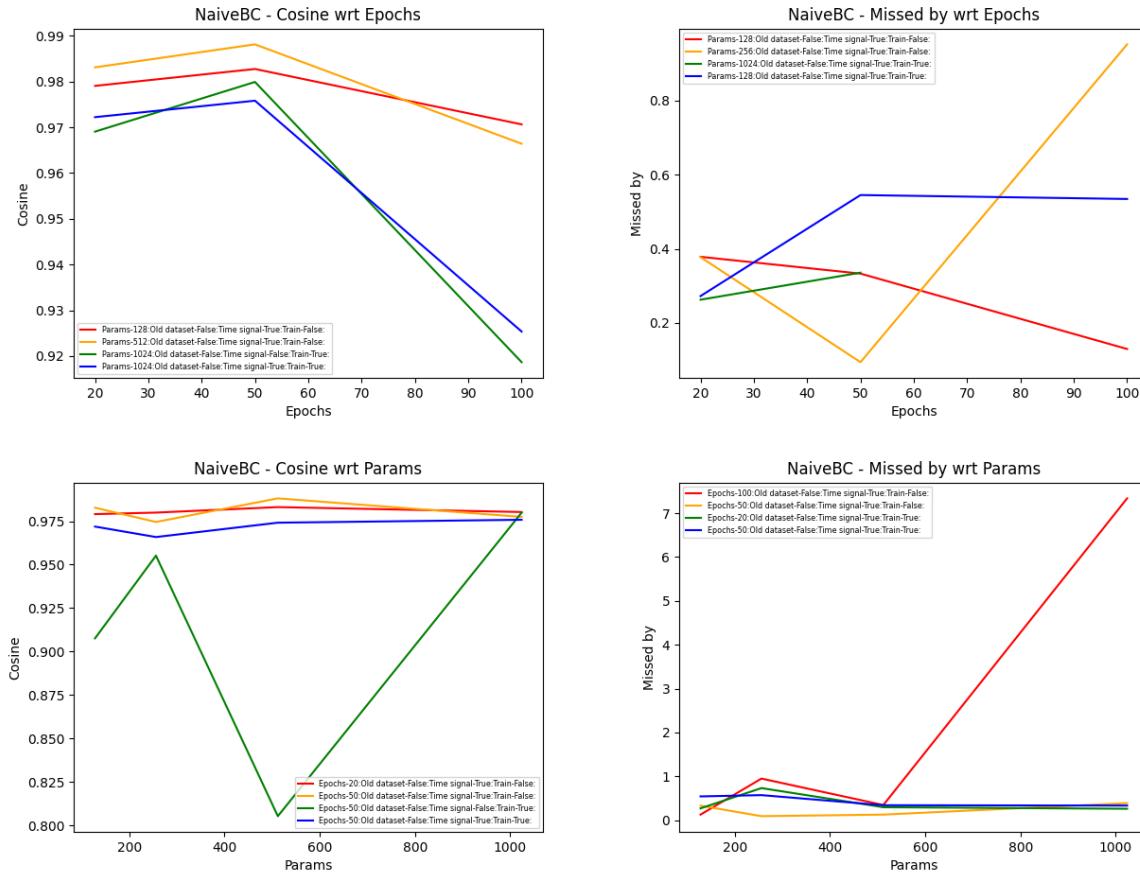
apmācības kopas (atceroties, ka tas ir novērtējums modeļa spējai aproksimēt tā apmācības datus) un mazākās validācijas kopas (kas sniedz ieskatu slēptā datus ģenerējošā sadalījuma apguvē, taču mazāko izmēru dēļ noved pie neskaidrākiem rezultātiem un sakarībām).

## 4.2. Sasniegtie rezultāti dažādām modeļu klasēm

Pirmais, ko var darīt, kad iegūti šādi sistematiski aprēķināti kvalitātes mēri, ir salīdzināt ar dažādām modeļu arhitektūrām sasniegtos labākos rezultātus. 11. attēlā redzamas diagrammas, kurās salīdzināti labākie globālie  $\tau_d$ ,  $\tau_g$  tuvības mēri un novirzes katrā laika solī iegūtajās konfigurācijās. Kā redzams, kosinusu līdzība ir ļoti augsta abiem modeļiem, liecinot, ka datu kopa tik tiesām tiek apgūta. Metrikas bieži vien labākas ir rekurentiem modeļiem, tas pats attiecas arī uz vidējām nobīdēm.

Var ar pārliecību apgalvot, ka rekurentais neironu tīkls mazāk diverģē no treniņa kopā novērotajiem metieniem, taču šī atšķirība nav radikāla. Izsakot grafiku mērogu cilvēkam saprotamās mērvienībās, rekurentais tīkls uzlabo vidējo pozīcijas klūdu no 2,5 uz 2cm, bet vidējo orientācijas vektora leņķisko nobīdi – no 11,5 uz 9,8 grādiem.

12. attēlā savukārt salīdzināti metiena novērtējumi. Redzams, ka ātruma vektora nobīdes vērtējumos rekurentais tīkls pārspēj klasisko. Pozīcijas klūda abos gadījumos ir

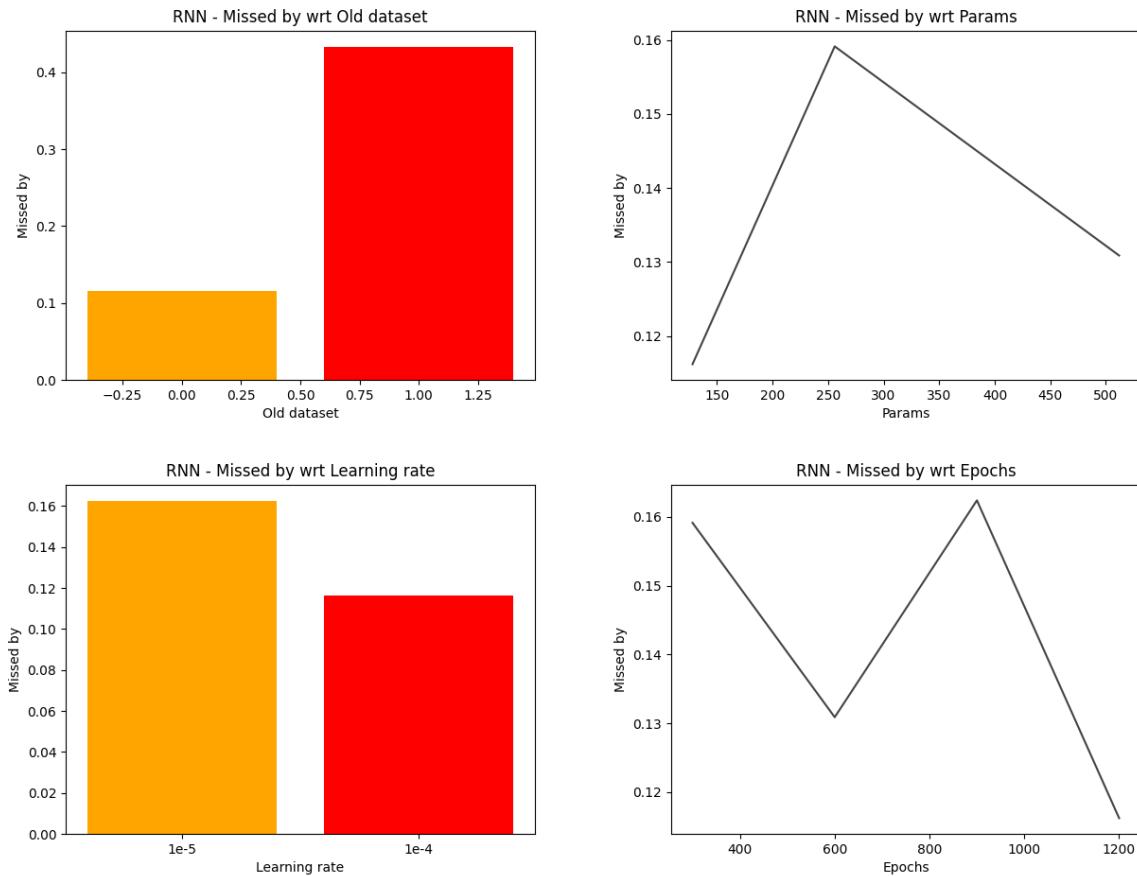


Att. 14: Atsevišķu klasisko modeļu novērtējumi, mainot tikai vienu hiperparametru. Augšējā rindā kosinusu līdzība un metiena kļūda atkarībā no epohas, apakšējā – no parametru skaita.

neliela, taču labāko rezultātu šajā gadījumā ir sasniedzis klasisks modelis. Taču pārsteidzošā kārtā, arī labāko rezultātu simulētā metiena nobīdē sasniedz viens no parastajiem modeļiem, kaut gan atšķirība ir ļoti neliela un precizitāte abiem modeļiem ir 0,1m diapazonā. Kā apspriests zemāk pie divu izmantoto demonstrāciju kopu salīdzinājuma, tas visticamāk skaidrojams ar nelielo validācijas kopu “mazo metienu” korpusā – novēdot pie lielākas dispersijas šajā kvalitātes mērā. Attēloti arī daži rezultāti, kas gūti, sākot GAN modeļa attīstību. Kā redzams, sasniegta modeļa konverģence – vērtējamie kritēriji manāmi uzlabojas, pieaugot treniņa epohu skaitam. Taču vēl nav sasniegta konkurentspejīga veikspēja, tāpēc šī darba ietvaros sīkāk rezultāti pētīti netiks.

### 4.3. Klasiskie neuronu tīkli

13. attēlā redzamas galvenās atzinās, kas gūtas strādājot ar parasto neuronu tīklu – labākie rezultāti pie katras attēlotā hiperparametra vērtības. Gan pret epohu skaitu, gan modeļa izmēriem novērotās sakarības liecina, ka problēmas dinamikas modelēšanai varētu būt samērā nedaudzas brīvības pakāpes. Līdz ar to neliels modelis un ne pārāk ilga apmācība ir pietiekami – pēc tam veikspēja atkal samazinās, visticamāk pārpielāgošanās rezultātā. Labākie rezultāti gūti pie 256 perceptroniem katrā slēptajā slānī un 50 treniņa



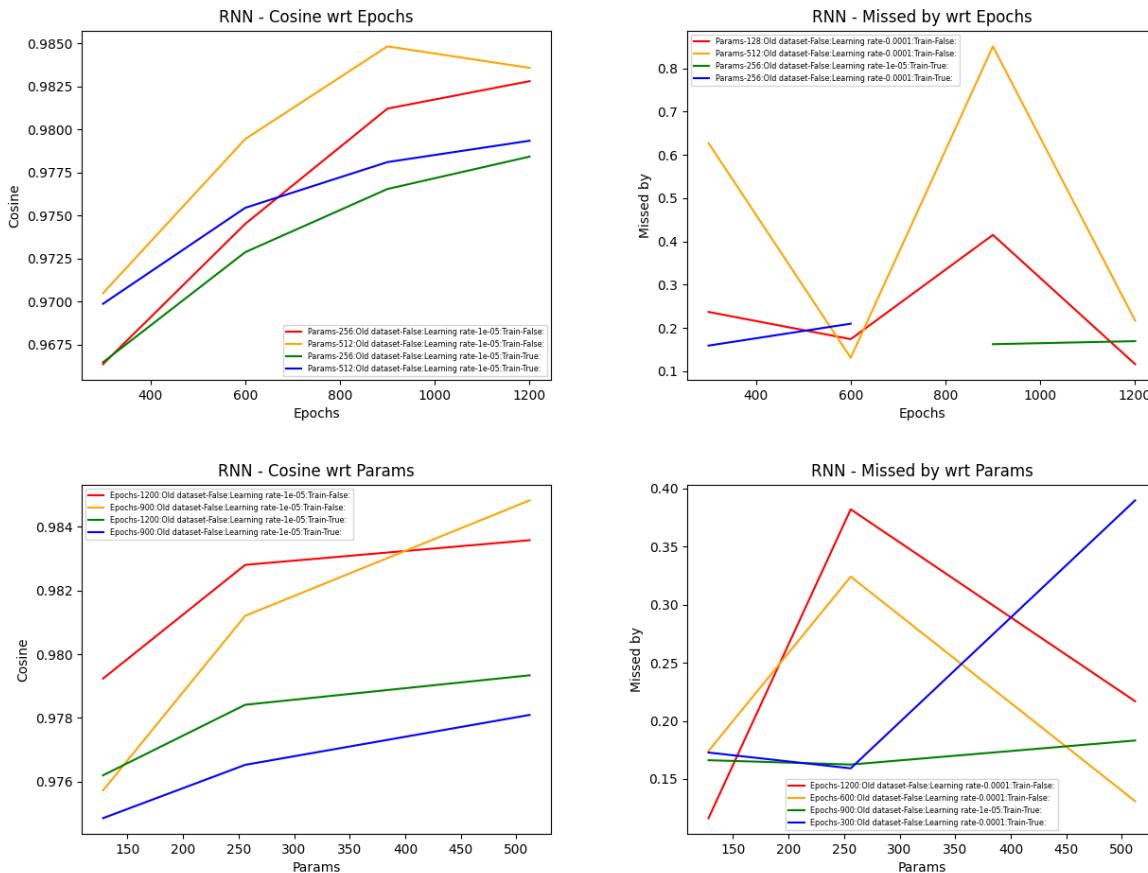
Att. 15: Rekurentā neironu tīkla ģenerēto trajektoriju simulēto metienu nobīdes atkarībā no: datu kopas; parametru skaita; apmācības ātruma; epohu skaita.

epohām.

14. attēlā redzamas izmaiņas novērtējumos atkarībā tikai no viena mainīgā lieluma. Var secināt, ka klasiskā modeļa līdzība ar demonstrāciju sadalījumu sāk kristies, pieaugot epohu skaitam, un abi labajā pusē redzami grafiki parāda, ka iespējams novērot pārpielāgošanos uz validācijas kopas. Interesanti, ka kosinusu līdzībai pie dažādiem parametru skaitiem var būt ievērojama dispersija pat bez skaidri redzama kāpuma vai krituma.

Pirmās datu korpusa versijas tika sagatavotas bez atsevišķa laika signāla – balstoties uz pieņēmumu, ka orientācijas izmaiņa varētu būt pietiekama, lai dotu stratēģijai kontekstu par to, kurā metiena fāzē tā šobrīd atrodas. Savā ziņā pieņēmums apstiprinājās – modelis konverģē, un arī kaut kas līdzīgs metienam tiek nogenerēts. Taču, kā redzams rezultātos, pievienojot atsevišķu laika signālu vektoram rezultāti tiek radikāli uzlaboti.

Visbeidzot, lielākā daļa demonstrāciju tikušas ierakstītas vairākās stadijās vēl mēnešus pirms autoregresijas rezultātus mēģināts atkārtot ar fizisku robotu. Izrādās, ka tās neatbilst robota izmēriem – veiktie vēzieni ir pārāk lieli, un mēģinājumi proporcionāli korigēt tos novēd pie nesakritības starp efektoru pozīciju un orientāciju (tieki mēģināts lielas rotācijas apvienot ar nelieliem pārvietojumiem, kas novēd pie bremzēšanas reālā robotā). Tāpēc neilgi pirms darba noslēguma iegūta jauna demonstrāciju kopa, kas gan ir mazāka par veco. Protams, arī validācijas kopa ir attiecīgi mazāka. Tajā ir līdzīga



Att. 16: Atsevišķu rekurento modelu novērtējumi, mainot tikai vienu hiperparametru. Augšējā rindā kosinusu un metiena klūda atkarībā no epohas, apakšējā – no parametru skaita.

mērķa koordinātu dispersija un, varbūt svarīgāk, tās ir tuvāk metiena sākumpunktam. Augstākās simulēto metieni precizitātes sasniegtas tieši uz šīs kopas. Mazākas absolūtās vērtības kombinētas ar relatīvi lielāku dispersiju liek domāt, ka varētu būt ievērojams nejaušības faktors labāko metienu mērā. Turpmākiem pētījumiem nepieciešams strādāt ar lielāku validācijas kopu. Tomēr jāpiebilst, ka vecajā datu kopā sasniegtās precizitātes patiesām varētu būt zemākas, jo tajā ir daudz lielāka *sākuma orientāciju* dispersija – rezultātā modelis paredz agresīvākas rotācijas, bet validācijas trajektorijas ievāktas ar neitrālu sākuma stāvokli.

#### 4.4. Rekurentie neuronu tīkli

15. attēlā salīdzināta rekurentā tīkla simulēto metienu precizitāte pie dažādiem hiperparametriem. Pirmkārt, redzams, ka ne pēc epohām, ne modeļa izmēriem skaidras sakarības nav novērojamas – vērtībām ir ļoti neliela dispersija. Tas rada zināmas cerības, ka vēl nav atrastas optimālas hiperparametru kombinācijas, un iespējami tālāki uzlabojumi. Svarīgi arī pamanīt, ka pie vecās datu kopas – kas ir lielāka un ar daudz lielākām efektora orientācijas izmaiņām – rekurentais modelis sasniedz ievērojami labākus rezultātus, nekā klasiskais. Tomēr pie nelielās jaunā datu korpusa validācijas kopas sasniegtā

precizitāte ir mazliet zemāka.

16. attēlā redzami rekurentā tīkla novērtējumu izmaiņas, noturot fiksētus pārējos hiperparametrus. Redzams, ka epohu skaits lielākiem modeļiem (ar 256, 512 perceptroniem katrā slēptajā slānī) noved pie augstākiem kopējās līdzības novērtējumiem (šajā gadījumā, kosinusu), taču sakarība nav tik viennozīmīga, apskatot metiena kļūdu. Tāpat kā iepriekš, katrā grafikā attēloti četri modeļi ar absolūti labākajām sasniegta jām vērtībām – divi pret treniņa kopu un divi pret testa. Redzams, ka, salīdzinot ar nelielās validācijas kopas trajektorijām, abi modeļi visos gadījumos ir veiksmīgi padevuši atlaišanas signālu, taču pie lielākās apmācības kopas abiem modeļiem tas notiek tikai konkrētā epohu diapazonā. Turklat izteikta kļūst apmācības ātruma ietekme uz rezultātiem. 256-neironu modelis ar apmācības ātrumu  $\lambda = 10^{-5}$  ir lietojams pie 900 vai 1200 epohām, bet ar ātrumu  $\lambda = 10^{-4}$  – pie 300 vai 600.

Apakšējā rindā kā mainīgais hiperparametrs izmantots perceptronu skaits modelī. Kosinusu līdzības grafiks parāda, ka faktiski modeļa spēja apgūt datu kopu vai to ģenerējošo sadalījumu vēl joprojām pieaug (kaut gan lēni), palielinot parametru skaitu pie dažādām pārējo hiperparametru kombinācijām. Savukārt metienu kļūda pret validācijas kopu liecina, ka pie 300 epohām pat ar augstāku apmācības ātrumu lielākais 512-neironu modelis vēl nav pietiekami labi apguvis sadalījumu. Pie 900 epohām iegūtie rezultāti uz validācijas kopas ir diezgan stabili, kas gan iespējams skaidrojams ar šīs kopas nelielo izmēru. Interesanti, ka pret apmācības kopu un lielāka epohu skaita gan 128, gan 512 perceptronu varianti pārspēj vidējo 256-neironu modeli – iespējams, liecinot par t.s. divkāršā krituma jeb “*double descent*” fenomenu [41].

## SECINĀJUMI

Veicot literatūras analīzi, identificēti Robotikas un mašīnuztveres laboratorijas mērķiem un iespējām atbilstošākie potenciālie risinājumi motivējošajam uzdevumam – MDP formālismam atbilstoša uzvedības klonēšana ar klasisku neironu tīklu kā stratēģiju un no laikrindu prognozēšanas metodēm aizgūta rekurentā tīkla pielietojums. Attīstītā automatizētā datu ieguves un priekšapstrādes sistēma ir adekvāta, lai ātri un sistemātiski sagatavotu demonstrāciju trajektoriju kopas no ierakstiem ar minimālu cilvēka iesaisti. Ar iegūtajiem datiem apmācītie klasiskie un rekurentie modeļi kvalitatīvi labi aproksimē metienus, un pārbaudot rezultātu uz fiziska robota var pārliecināties, ka pat ar neprecīzu laika parametrizāciju jau iespējams izpildīt metiena kustību.

Salīdzinot apmācītos modeļus, redzams, ka daudzos statistiskos novērtējumos sarežģītākais rekurentais tīkls pārspēj klasisko neironu tīklu, taču pagaidām grūti spriest, vai sniegtie uzlabojumi ir pietiekami, lai atsvērtu modelja lielāko sarežģītību gan no izpildījuma, gan asimptotiskās laika sarežģītības puses. Katrā ziņā ļoti aptuvenajā metētā trāpīguma novērtējumā pagaidām labāko rezultātu sasniedz klasiskais neironu tīkls, taču atšķirība ir neliela un tās nozīmīgums ir strīdīgs, jo rezultāts iegūts uz ļoti nelielas validācijas kopas.

Nemot vērā augstāk minēto pieredzi un atziņas no teorētiskās literatūras, iezīmējas skaidri virzieni tālākai pētnieciskai un inženiertehniskai darbībai:

- 1) nepieciešams izstrādāt programmnodrošinājumu precīzi laikā parametrizētu telpisku trajektoriju izpildei ar robotu ROS vidē. Šī darba zinātniskā prioritāte nav augsta, jo komerciāli pielietojamā industriālo robotu vidē šāda tipa funkcionalitāte tiek piedāvāta (piemēram, lineāras kustības instrukcijās ar telpiska ātruma argumentu) [67]. Taču šāds rīks būtu nepieciešams tālākai eksperimentālai darbībai, piemēram, salīdzinot dažādu modeļu faktiski sasniegto precizitāti;
- 2) nepieciešams ievākt lielākas datu kopas, kas atbilst robota kinemātikas ierobežojumiem, it sevišķi validācijas mērķiem;
- 3) pastāv iespēja, ka ar noteiktiem uzlabojumiem apmācības procesā pilnīgi pietiek tikai ar MDP formālismam atbilstošām stratēģijām. Viens no papildinājumiem varētu būt jau uzsāktā GAN modeļu izveide, kas teorētiskajā literatūrā sasniedz labākos rezultātus;
- 4) modeļus varētu apmācīt tā, lai tie paredz nevis nākamo pozīciju un orientāciju, bet gan telpiskos un leņķiskos ātrumus. Tādā veidā būtu triviāli integrēt modeli ar servokontrolleri;
- 5) motivējošais uzdevums – pudeles metiens – ir tikai viens no daudziem potenciāliem pielietojumiem sistēmai, kas kustību uztveres radītos datus pārvērš apmācītos robota vadības modeļos. Noderīgi būt atrast citus šādus uzdevumus un pārbaudīt, cik sarežģīti ir vispārināt šī darba ietvaros izveidotās iestrādes citām problēmām. It sevišķi jādomā par to, cik liels papildus darbs nepieciešams, lai pielāgotu priekšapstrādes sistēmu darbam ar citādiem uzdevumiem, kam pastāv citi kritiskie punkti un aprēķināmie papildu parametri.

## ATSAUCES

- [1] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.
- [2] Shervine Amidi Afshine Amidi. *Recurrent Neural Networks cheatsheet*. Stanford. URL: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks> (visited on 05/18/2022).
- [3] Beijing Academy of Artificial Intelligence. *Suggested Notation for Machine Learning*. 2020. URL: <http://ctan.math.utah.edu/ctan/tex-archive/macros/latex/contrib/mlmath/mlmath.pdf> (visited on 01/14/2022).
- [4] Isaac Asimov. *I, robot*. Vol. 1. Spectra, 2004.
- [5] Alexandre Attia and Sharone Dayan. “Global overview of imitation learning”. In: *arXiv preprint arXiv:1801.06503* (2018).
- [6] Jim Belk. *Quaternion distance*. URL: <https://math.stackexchange.com/q/90098>.
- [7] Aude Billard et al. “Handbook of robotics chapter 59: Robot programming by demonstration”. In: *Handbook of Robotics*. Springer (2008).
- [8] Liana Blumberga. *GENERĀTĪVO PRETINIEKU TĪKLU TEORĒTISKA IZPĒTE (kursa darba konferences stenda plakāts)*. 2019. URL: [https://www.df.lu.lv/file/admin/user\\_upload/LU.LV/Apaksvietnes/Fakultates/www.df.lu.lv/Studijas/Magistrantura/Konferences\\_stenda\\_referati/2019/Blumberga\\_Liana.pdf](https://www.df.lu.lv/file/admin/user_upload/LU.LV/Apaksvietnes/Fakultates/www.df.lu.lv/Studijas/Magistrantura/Konferences_stenda_referati/2019/Blumberga_Liana.pdf) (visited on 01/20/2022).
- [9] Matthew Brett. *Correlation and projection*. 2016. URL: [https://matthew-brett.github.io/teaching/correlation\\_projection.html](https://matthew-brett.github.io/teaching/correlation_projection.html) (visited on 05/13/2022).
- [10] Daniel Brown et al. “Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations”. In: *International conference on machine learning*. PMLR. 2019, pp. 783–792.
- [11] Kyunghyun Cho et al. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).
- [12] Junyoung Chung et al. “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555* (2014).
- [13] Steve Crowe. *10 Biggest Challenges in Robotics*. 2018. URL: <https://www.therobotreport.com/10-biggest-challenges-in-robotics/> (visited on 01/20/2022).
- [14] Yan Duan et al. “One-shot imitation learning”. In: *arXiv preprint arXiv:1703.07326* (2017).
- [15] Jonatan S Dyrstad et al. “Teaching a robot to grasp real fish by imitation learning from a human supervisor in virtual reality”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7185–7192.

- [16] Peter Englert and Marc Toussaint. “Learning manipulation skills from a single demonstration”. In: *The International Journal of Robotics Research* 37.1 (2018), pp. 137–154.
- [17] Mathilde Frot. *5 Trends in Computer Science Research*. 2021. URL: <https://www.topuniversities.com/courses/computer-science-information-systems/5-trends-computer-science-research> (visited on 01/21/2022).
- [18] Kunihiko Fukushima. “Neocognitron: A hierarchical neural network capable of visual pattern recognition”. In: *Neural networks* 1.2 (1988), pp. 119–130.
- [19] ABB Group et al. “Special report: Robotics–ABB group”. In: *ABB Review* (2016).
- [20] Abhishek Gupta et al. “Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning”. In: *arXiv preprint arXiv:1910.11956* (2019).
- [21] Khalifa H Harib et al. “Parallel, serial and hybrid machine tools and robotics structures: comparative study on optimum kinematic designs”. In: *Serial and Parallel Robot Manipulators-Kinematics, Dynamics, Control and Optimization*. IntechOpen, 2012.
- [22] Kris Hauser. *Robotic Systems – Chapter 4. 3D Rotations*. University of Illinois. 2020. URL: <http://motion.cs.illinois.edu/RoboticSystems/3DRotations.html> (visited on 05/17/2022).
- [23] Kris Hauser. *Robotic Systems – Chapter 5. Robot Kinematics*. University of Illinois. 2020. URL: <http://motion.cs.illinois.edu/RoboticSystems/Kinematics.html> (visited on 05/17/2022).
- [24] Kris Hauser. *Robotic Systems – Chapter 6. Inverse Kinematics*. University of Illinois. 2020. URL: <http://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html> (visited on 05/17/2022).
- [25] Kris Hauser. *Robotic Systems – Chapter 8. What is motion planning?* University of Illinois. 2020. URL: <http://motion.cs.illinois.edu/RoboticSystems/WhatIsMotionPlanning.html> (visited on 05/17/2022).
- [26] Todd Hester et al. “Deep q-learning from demonstrations”. In: *Thirty-second AAAI conference on artificial intelligence*. 2018.
- [27] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29 (2016), pp. 4565–4573.
- [28] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [29] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101. doi: 10.1214/aoms/1177703732. URL: <https://doi.org/10.1214/aoms/1177703732>.

- [30] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. “Movement imitation with nonlinear dynamical systems in humanoid robots”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 2. IEEE. 2002, pp. 1398–1403.
- [31] Abhishek Jha et al. “Imitation learning in industrial robots: a kinematics based trajectory generation framework”. In: *Proceedings of the Advances in Robotics*. 2017, pp. 1–6.
- [32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.
- [33] Tobias Kunz and Mike Stilman. “Time-optimal trajectory generation for path following with bounded acceleration and velocity”. In: *Robotics: Science and Systems VIII* (2012), pp. 1–8.
- [34] Seppo Linnainmaa. “The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors”. In: *Master’s Thesis (in Finnish)*, Univ. Helsinki (1970), pp. 6–7.
- [35] YuXuan Liu et al. “Imitation from observation: Learning to imitate behaviors from raw video via context translation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1118–1125.
- [36] Corey Lynch and Pierre Sermanet. “Language conditioned imitation learning over unstructured data”. In: *Proceedings of Robotics: Science and Systems*. doi 10 (2021).
- [37] Corey Lynch et al. “Learning latent plans from play”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 1113–1132.
- [38] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
- [39] S Muench et al. “Robot programming by demonstration (rpd)-using machine learning and user interaction methods for the development of easy and comfortable robot programming systems”. In: *Proceedings of the International Symposium on Industrial Robots*. Vol. 25. INTERNATIONAL FEDERATION OF ROBOTICS, & ROBOTIC INDUSTRIES. 1994, pp. 685–685.
- [40] Ashvin Nair et al. “Overcoming exploration in reinforcement learning with demonstrations”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6292–6299.
- [41] Preetum Nakkiran et al. “Deep double descent: Where bigger models and more data hurt”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2021.12 (2021), p. 124003.
- [42] *numpy documentation*. NumPy Developers. URL: <https://numpy.org/doc/stable> (visited on 05/21/2022).

- [43] *Odometry*. MIT CS and AI Laboratory. URL: <https://groups.csail.mit.edu/dr1/courses/cs54-2001s/odometry.html> (visited on 05/18/2022).
- [44] Alex Owen-Hill. *The Decade of Artificial Intelligence*. 2021. URL: <https://blog.robotiq.com/what-are-the-different-programming-methods-for-robots> (visited on 01/16/2022).
- [45] *pandas documentation*. Pandas Development Team. URL: <https://pandas.pydata.org/docs> (visited on 05/21/2022).
- [46] Peter Pastor et al. “Online movement adaptation based on previous sensor experiences”. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 365–371.
- [47] Peter Pastor et al. “Skill learning and task outcome prediction for manipulation”. In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3828–3834.
- [48] Dario Pavllo, David Grangier, and Michael Auli. “Quaternet: A quaternion-based recurrent model for human motion”. In: *arXiv preprint arXiv:1805.06485* (2018).
- [49] Dean A Pomerleau. *Alvinn: An autonomous land vehicle in a neural network*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE and PSYCHOLOGY . . ., 1989.
- [50] Selva Prabhakaran. *Cosine Similarity - Understanding the math and how it works (with python codes)*. 2018. URL: <https://www.machinelearningplus.com/nlp/cosine-similarity/> (visited on 05/13/2022).
- [51] LZA TK ITTEA protokoli. *reinforcement learning*. 2017. URL: <https://termini.gov.lv/kolekcijas/97/skirklis/454193> (visited on 01/20/2022).
- [52] Pēteris Račinskis. *Masters – github repository*. 2022. URL: <https://github.com/peteris-racinskis/masters> (visited on 05/11/2022).
- [53] Pēteris Račinskis. *Masters Data – github repository*. 2022. URL: [https://github.com/peteris-racinskis/masters\\_data](https://github.com/peteris-racinskis/masters_data) (visited on 05/11/2022).
- [54] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).
- [55] Scott Reed et al. “A Generalist Agent”. In: *arXiv preprint arXiv:2205.06175* (2022).
- [56] *Robot Operating System*. Open Robotics. URL: <https://www.ros.org> (visited on 05/18/2022).
- [57] *ROS wiki – communication patterns*. Open Robotics. URL: <http://wiki.ros.org/ROS/Patterns/Communication> (visited on 05/18/2022).
- [58] *ROS wiki – roslaunch*. Open Robotics. URL: <http://wiki.ros.org/roslaunch> (visited on 05/18/2022).
- [59] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. “No-regret reductions for imitation learning and structured prediction”. In: *In AISTATS*. Citeseer. 2011.

- [60] Stefan Schaal, Auke Ijspeert, and Aude Billard. “Computational approaches to motor learning by imitation”. In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 358.1431 (2003), pp. 537–547.
- [61] Stefan Scherzinger, Arne Roennau, and Rüdiger Dillmann. “Forward dynamics compliance control (FDCC): A new approach to cartesian compliance for robotic manipulators”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 4568–4575.
- [62] Stefan Scherzinger, Arne Roennau, and Rüdiger Dillmann. “Contact skill imitation learning for robot-independent assembly programming”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 4309–4316.
- [63] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.
- [64] *STAT 501, 14.1 – Autoregressive Models*. Penn State. 2021. URL: <https://online.stat.psu.edu/stat501/lesson/14/14.1> (visited on 05/18/2022).
- [65] Andrea D. Steffen. *Robotics Takes Plastic Recycling To The Next Level*. 2021. URL: <https://www.intelligentliving.co/robotics-plastic-recycling-next-level/> (visited on 01/20/2022).
- [66] Richard S Sutton and Andrew G Barto. “Reinforcement learning: An introduction”. In: MIT press, 2018, pp. 60–77.
- [67] *Technical Reference Manual – RAPID*. ABB. URL: [https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual\\_RAPID\\_3HAC16581-1\\_revJ\\_en.pdf](https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf) (visited on 05/19/2022).
- [68] *Tensorflow Core v2.8.0 API documentation – Binary Cross-Entropy loss*. Google. 2022. URL: [https://www.tensorflow.org/api\\_docs/python/tf/keras/losses/BinaryCrossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy) (visited on 05/12/2022).
- [69] *Tensorflow Core v2.8.0 API documentation – Huber loss*. Google. 2022. URL: [http://www.tensorflow.org/api\\_docs/python/tf/keras/losses/Huber](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber) (visited on 05/12/2022).
- [70] *Tensorflow Core v2.8.0 API documentation – Ragged Tensors*. Google. 2022. URL: [https://www.tensorflow.org/guide/ragged\\_tensor](https://www.tensorflow.org/guide/ragged_tensor) (visited on 05/12/2022).
- [71] *Tensorflow Core v2.8.0 API documentation – ReLU activation function*. Google. 2022. URL: [https://www.tensorflow.org/api\\_docs/python/tf/nn/relu](https://www.tensorflow.org/api_docs/python/tf/nn/relu) (visited on 05/12/2022).
- [72] *TensorFlow documentation*. Google. URL: [https://www.tensorflow.org/api\\_docs/python/tf](https://www.tensorflow.org/api_docs/python/tf) (visited on 05/21/2022).

- [73] *Tensorflow Tutorials – image classification*. Google. 2022. URL: [https://www.tensorflow.org/tutorials/keras/classification?hl=en#compile\\_the\\_model](https://www.tensorflow.org/tutorials/keras/classification?hl=en#compile_the_model) (visited on 05/12/2022).
- [74] Faraz Torabi, Garrett Warnell, and Peter Stone. “Behavioral cloning from observation”. In: *arXiv preprint arXiv:1805.01954* (2018).
- [75] Faraz Torabi, Garrett Warnell, and Peter Stone. “Generative adversarial imitation from observation”. In: *arXiv preprint arXiv:1807.06158* (2018).
- [76] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.
- [77] Ning Wang, Chuize Chen, and Alessandro Di Nuovo. “A framework of hybrid force/motion skills learning for robots”. In: *IEEE Transactions on Cognitive and Developmental Systems* 13.1 (2020), pp. 162–170.
- [78] Eric Weisstein. *Wolfram Math World – Metric*. 2022. URL: <https://mathworld.wolfram.com/Metric.html> (visited on 05/13/2022).
- [79] Shudong Yang, Xueying Yu, and Ying Zhou. “Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example”. In: *2020 International workshop on electronic communication and artificial intelligence (IWECAI)*. IEEE. 2020, pp. 98–101.
- [80] Fangyi Yu. *A Thorough Guide to Time Series Analysis*. 2021. URL: <https://towardsdatascience.com/a-thorough-guide-to-time-series-analysis-5439c63bc9c5> (visited on 05/18/2022).
- [81] Tianhao Zhang et al. “Deep imitation learning for complex manipulation tasks from virtual reality teleoperation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 5628–5635.

## DOKUMENTĀRĀ LAPA

Magistra darbs “Atdarinošās mašīnmācīšanās pielietojums robotikā” izstrādāts LU Datorikas fakultātē.

Darba teksta galīgā versija izgatavota 23.05.2022.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: \_\_\_\_\_ (Autora paraksts un datums)

Ar savu parakstu apliecinu, ka esmu lasījis augstāk minēto magistra darbu un atzīstu to par p i e m ē r o t u / n e p i e m ē r o t u (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu magistrantūrā.

Darba vadītājs: \_\_\_\_\_ (Autora paraksts un datums)

Darbs iesniegts magistrantūras sekretariātā \_\_\_\_\_ (Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Studiju metodiķe: \_\_\_\_\_ (Metodiķes paraksts)

Recenzents: prof. Dr. dat. Guntis Bārzdiņš

Darbs aizstāvēts magistra gala pārbaudījuma komisijas sēdē

\_\_\_\_\_ prot. Nr. \_\_\_\_\_

Komisijas sekretārs: \_\_\_\_\_ (Sekretāra paraksts)