LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

ATDARINOŠĀS MAŠĪNMĀCĪŠANĀS PIELIETOJUMS ROBOTIKĀ

MAĢISTRA DARBS

Autors: **Pēteris Račinskis**
Stud. apl. Nr. pr20015
Darba vadītājs: Dr. sc. comp. Modris Greitāns

RĪGA 2022

Saturs

1. IEVADS

Mašīnmācīšanās šobrīd tiek plaši uzskatīta par vienu no aktuālākajām datorzinātņu pētniecības nozarēm [12]. Pēdējās desmitgades laikā šī pētniecības lauciņa popularitāte ir daudzkārt pieaugusi, pateicoties galvenokārt diviem faktoriem: ļoti vispārīgiem neironu tīklu modeļiem un skaitļošanas resursu veiktspējai, kas beidzot ļāvusi šos teorētiski jau ļoti sen [26, 22, 13] iedomātos mākslīgā intelekta uzbūves elementus realizēt praksē. Tā risināti uzdevumi, ko izsenis daudzi uzskatījuši par neiespējamiem, un lietojuši kā argumentu pret mašīnmācīšanos kā rīku, kas spētu konkurēt ar bioloģiskas izcelsmes prātiem — semantiskas nozīmes meklēšana attēlos [21], tekstu korpusu analīze un ģenerēšana ar "izpratni" par to saturu [51] un visspējīgāko spēlētāju pārspēšana nepilnīgas informācijas spēlēs ar neaptverami milzīgiem iespējamo stāvokļu permutāciju skaitem [41].

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. Un šī pati zinātniskā fantastika radījusi arī nesaraujamu saiti starp robotiem un mākslīgo intelektu [3] — diskusijas par mākslīgo intelektu bieži plūstoši pāriet diskusijās par ar šādu intelektu aprīkotiem robotiem, un šo robotu neizbēgami kareivīgajām ambīcijām attiecībā pret cilvēci. Protams, zinātne ne vienmēr seko populārzinātniskās iedomas lidojumam, taču šāda saikne ir visnotaļ pamatota — spēja mācīties no paraugiem vai patstāvīgi un pielāgoties savai apkārtnei ir ārkārtīgi noderīga, jo daudzi uzdevumi, kuru risināšanai varētu pielietot robotus, ir sarežģīti nevis to fizikālajā izpildē, bet tieši vadības uzdevuma formulēšanā un realizācijā — lai to redzētu pietiek vien aplūkot kādu nejauši izvēlētu sarakstu ar robotikā aktuālām problēmām [8].

Atdarinošā mašīnmācīšanās (*imitation learning*) ir viens no paņēmieniem, ar kuriem tiek mēģināts risināt šādas sarežģītas vadības problēmas. Lai gan pamatu pamatos nevar apgalvot, ka tā ir tikai robotikai piemērota metožu saime, lielākā daļa izpētes virzīta tieši šajā virzienā — 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 [4]. Mašīnmācīšanās nozarē bioloģiskas analoģijas 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 analoģisku 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 dziļākai pašmācībai [16]; vai arī izmantot šādu stimulēto metodi, lai precīzāk imitētu treniņa datus [1].

1.1. Darba mērķis un struktūra

Šis ir maģistra kursa darbs — pirmais konkrētais rezultāts, kas sasniegts maģistra darba izstrādes procesā. Tātad saturs un formāts ir lielā mērā atkarīgs no nākamajā semestrī aizstāvamā gala darba izstrādei individuāli nospraustajiem mērķiem un iztecējušā semestra gaitā reāli paveiktā.

Kopējā maģistra darba tēma izvēlēta ar Elektronikas un datorzinātņu centra ekspertu palīdzību, meklējot šobrīd aktuālu pētniecības virzienu. Izvērtējot dažādus pieejamos variantus, izvēlēts pētniecības virziens — atdarinošā mašīnmācīšanās — un konkrēts motivējošais uzdevums — atkritumu šķirošanas līnijas robots, kas spēj nogādāt pudeles vai citus objektus tiem atbilstošajās tvertnēs, izmantojot mešanas kustības. Faktiski līdz šim galvenokārt veikta nozares literatūras avotu izpēte un nepieciešamo pamatzināšanu apguve. Līdz ar to šī darba mērķis ir noskaidrot, vai atdarinošās mācīšanās metodes ir piemērotas motivējošās problēmas risināšanai, atrast piemērotākos paņēmienus un aktuālākos rezultātus jau eksistējošajā zinātniskās literatūras korpusā un ieskicēt risinājuma plānu, kura praktiska izpilde un novērtēšana tad arī veidotu noslēguma darba pētniecisko jaunpienesumu. Attiecīgi, kursadarbs sastāv no trijām daļām:

1) ievada, kurā īsi izklāstīti vispārīgi jēdzieni, kas nepieciešami, lai izprastu zinātnisko literatūru nozarē, kā arī aprakstīts motivējošais uzdevums;
2) literatūras analīzes, kur izdalīti daži galvenie pētnieciskās darbības virzieni nozarē, aplūkoti konkrēti algoritmi un pētījumi kā piemēri;
3) secinājumiem, kur literatūras analīzē gūtās atziņas īsi apkopotas un izmantotas, lai piedāvātu provizorisku formu motivējošā uzdevuma risinājumam.

1.2. Terminoloģijas tulkojumi, apzīmējumi, saīsinājumi

Viena no īpatnībām, ar ko ir nācies saskarties, strādājot tieši ar mašīnmācīšanās nozari, ir nepārprotamas 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 dziļi specifisku nišu, kas nav iedvesmojusi daudz mēģinājumu tulkot to uz citām valodām, savukārt uzplaukuma laikos vēl ir ļoti daudz materiāla, ko vienkārši neviens nav paguvis iztulkot.

Patvaļīgi 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**. Šis termins pamatā tiek lietots, lai aprakstītu kādu funkciju, kas novērojumus attēlo lēmumu telpā. Pirmais ieraksts tieši tāpēc, ka varētu būt strīdīgākais. Angļu valodā pastāv divi termini, *policy* un *politics*, kas parasti latviski

tiek tulkoti vienādi — politika — par spīti radikāli atšķirīgām nozīmēm. Termins *strategy* tiek lietots kā sinonīms pirmajam abās valodās, un arī piemērojams tieši šādām lēmumu pieņemšanas funkcijām, piemēram, spēļu teorijā.

2) *reinforcement learning* — **stimulētā mašīnmācīšanās**. Meklējumi tiešsaistē atklāj [34], ka šis tulkojums jau ir apstiprināts standartā, taču varētu būt nezināms lasītājiem, kas ar to sastopas pirmo reizi — pat ja zināms metodes angliskais 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 kolēģes Lianas Blumbergas kursa darba plakāts, kur šis tulkojums ir lietots [6], tāpēc tas pats tiks lietots arī šeit.

Tekstā var bez paskaidrojuma tikt izmantoti šādi saīsinājumi:

1) RL — stimulētā mašīnmācīšanās (*Reinforcement Learning*)
2) IRL — inversā stimulētā mašīnmācīšanās (*Inverse Reinforcement Learning*)
3) GAN — ģeneratīvais pretinieku tīkls (*Generative Adversarial Network*)
4) VR — virtuālā realitāte
5) MDP — Markova lēmumu process (*Markov Decision Process*)
6) EDI — Elektronikas un datorzinātņu institūts
7) LIDAR — attāluma lāzermērīšana

Biežāk izmantoti matemātiskie apzīmējumi:

1) π, π_θ, π^* — stratēģija, stratēģija ar parametriem θ, instruktora stratēģija
2) s — sistēmas stāvoklis vai tiešais novērojums
3) s_t, s' — 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) θ, ϕ, ψ — modeļu parametru vektori
9) $\mathbb{E}$ — matemātiskā cerība

1.3. Motivējošais uzdevums

Nevarētu apgalvot, ka atdarinošā mašīnmācīšanās kā tēma šim kursa darbam un maģistra darbam kopumā tikusi izvēlēta tīri inženiertehniskā procesa rezultātā — sākot ar problēmas definīciju un nonākot pie tai piemērotākā risinājuma, pārbaudot un atsijājot visas iespējamās pieejas, priekšlaicīgi neieguldot pārāk daudz pūļu un resursu kādas konkrētas metodes pielietošanā. Daļēji orientē vēlme — gan no EDI, gan autora puses — 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. Bruņojoties ar šādu konkrētu mērķi, nozares literatūras pārskatu un teorētisko zināšanu apguvi iespējams strukturēt un prioritarizēt.

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 [42]. Konkrēti aplūkojot plastmasas iepakojumu šķirošanu, jau izstrādātas metodes to klasifikācijai un satveršanai [42]. 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, un tā masa salīdzinot ar robota pašmasu ir neliela, potenciāli iespējams ievērojami paātrināt satveršanas un pārvietošanas ciklu laikus, jo nav nepieciešams visu robotu pārvietot līdz galamērķim. 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, mēģinot manuāli programmēt katru metienu rodas virkne problēmu — dažādi trajektoriju sākumpunkti, dažādi galamērķi un to iespējamā mainība, objektu neregularitāte, u.c. Tāpēc analītisku risinājumu atrast varētu būt ļoti nepraktiski, un tiek apsvērts mašīnmācīšanās metožu lietojums.

Attiecīgi var uzskaitīt šādus nosacījumus uzdevumam:

1) var pieņemt, ka sākuma stāvoklī robots jau ir satvēris objektu, ja nepieciešams;
2) trajektorijas sākumpunkti ir mainīgi — jāspēj izdarīt metiens no dažādām pozīcijām;
3) ir zināma objekta klasifikācija un līdz ar to — tā vēlamais galamērķis;
4) vēlamie galamērķi var būt dažādi un mainīgi;
5) rezultējošo modeli jāspēj pielāgot apstākļiem darba vidē bez pārlieku darbietilpīgas un dārgas treniņa procedūras.

Nav grūti redzēt, ka šādam uzdevumam par atalgojuma kritēriju pieņemot trāpīšanu pareizajā tvertnē, funkcijas vērtības būs ļoti retinātas, kas apgrūtina klasisko stimulētās metožu pielietojumu. Nākamās nodaļas ietvaros tiek veikts vispārīgs atdarinošās mašīnmācīšanās metožu pētniecības pārskats, paturot prātā izvēlētā uzdevuma specifiku — pievēršot uzmanību tiem rezultātiem un secinājumiem, kas varētu būt nozīmīgi problēmas risināšanā. Noslēgumā tiek izdarīti secinājumi par to, kuras citur gūtās atziņas ir svarīgākās, un piedāvāts rīcības plāns motivējošā uzdevuma izpildei.

2. LĒMUMA PROCESI, REGRESIJA

Pētot un veidojot spriedumus par zinātnisko literatūru viens no lielākajiem šķēršļiem lasītājam "no malas" ir katrā nozarē pieņemtais tehnisko priekšzināšanu kopums, ko autori sagaida no auditorijas. Tas, protams, ir loģiski, jo publikācija, kas apraksta jaunākos atklājumus kādā dziļi specifiskā lauciņā, nevar veltīt visu sev atvēlēto drukas apjomu elementāras un vispārzināmas terminoloģijas skaidrojumiem. Tāpat, tālāk atskaitē iztirzājot šos rakstus, noderīgi ir ieviest tiem kopīgus apzīmējumus un definēt visus vienuviet. Tāpēc šajā ievada daļā tiek īsi aprakstīti jēdzieni, kas jāizprot, lai varētu veikt literatūras analīzi un izdarīt secinājumus.

2.1. 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:

$$M(x) \approx f(x) \tag{2.1}$$

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 θ:

$$\text{Meklē } \theta : M(x|\theta) = M_\theta(x) \approx f(x) \tag{2.2}$$

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

$$\forall f \forall x \exists \theta : M_\theta(x) \approx f(x) \tag{2.3}$$

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

$$f : x \to \mathbb{R}^k \tag{2.4}$$

$$M : x \times \theta \to \mathbb{R}^k \tag{2.5}$$

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

$$f : x \to C = \{c_1, c_2, ..., c_m\} \tag{2.6}$$

$$M : x \times \theta \to C \tag{2.7}$$

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:

$$M : x \times \theta \times c_i \to [0; 1] \tag{2.8}$$

$$M_\theta(x, c_i) = P_i \tag{2.9}$$

$$\sum_{i=1}^{m} P_i = 1 \tag{2.10}$$

Lai varētu novērtēt, cik labi modelis aproksimē nezināmo funkciju, un vadīt parametru apmācības procesu, tiek izmantotas mērķa funkcijas (*loss functions*) [2]:

$$\ell : M_\theta(x) \times f(x) \to \mathbb{R} \tag{2.11}$$

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:

$$s \sim \mathcal{D} \Leftrightarrow s \text{ ir no varbūtību sadalījuma } \mathcal{D} \tag{2.12}$$

$$y_i = f(x_i) \tag{2.13}$$

$$s_i = (x_i, y_i) \tag{2.14}$$

$$S = \{s_1, s_2, ..., s_n | s_i \sim \mathcal{D}\} \tag{2.15}$$

Datu kopai var aprēķināt empīrisku mērķa funkcijas novērtējumu,

$$L_S(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(M_\theta(x_i), y_i) \tag{2.16}$$

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*)

$$L_\mathcal{D}(\theta) = \mathbb{E}_\mathcal{D}[\ell(M_\theta(x_i), y_i)] \tag{2.17}$$

$$\text{Apmāca } M_\theta \text{ uz } \mathcal{D} \to \text{Minimizē } L_\mathcal{D}(\theta) \tag{2.18}$$

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

2.1.1. Neironu tīkli – vispārīgi

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. perceptorns, kas izteikts jau pašos pirmsākumos [26]. Perceptrons funkcija, kas piemēro nelineāru aktviācijas funkciju σ argumentu vektora $\vec{x}$ elementu savstarpējai lineārai kombinācijai, t.i,

$$f_{perceptron}(\vec{x}) = \sigma(\vec{w} \cdot \vec{x} + b) \tag{2.19}$$

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

$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ ... \\ w_k^T \end{bmatrix}; \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_k \end{bmatrix}; \tag{2.20}$$

$$f_{layer}(\vec{x}) = \sigma(W\vec{x} + \vec{b}) \tag{2.21}$$

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 slāņi 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 katra 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 [21], 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.

2.1.2. Laikrindu modelēšana

2.1.3. Rekurentie neironu tīkli

2.2. 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 [4]

$$MDP = (S, A, R, T, \gamma) \tag{2.22}$$

kur S — sistēmas iespējamo stāvokļu s kopa; A — kontrolētajam procesam ("aģentam") pieejamo darbību a kopa; $R : S \times A \to \mathbb{R}$ vai $R : S \to \mathbb{R}$— atalgojuma (*reward*) funkcija, kas ļauj kārtot sasniegtos stāvokļus pēc to lietderības; $T : S \times A \to S$ 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; γ — 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 γ, taču tie ir nepieciešami metodēm, kas lieto stimulēto mašīn-mācīšanos. Tā kā parasti spriests tiek par stratēģijām π_θ, 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 aģenta (modeļa, stratēģijas) darbību. Pie sākotnējo stāvokļu kopas S_0 un stratēģijas π 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 $t = \{1, 2, ..., T\}$, var izteikt kā laikrindu formā, kas sastāv no *state-action* pāriem — $((s_1, a_1), (s_2, a_2), ..., (s_T, a_T))$. 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.

2.3. 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 [15, 7], bet arī atdarināšanas metodes, kas tiešā veidā izmanto stimu-lēto mācīšanos, lai atdarinātu trajektoriju demonstrācijas [11]. 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 [43]. 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

$$p(s', r|s_t, a_t) = P[s_{t+1} = s', r = R(s')] \tag{2.23}$$

Nākotnē sagaidāmās atalgojuma vērtības, ņemot vērā dilšanas koeficientu γ, var izteikt kā

$$G_t = \sum_{k=0}^{T-t} \gamma^k R(s_{t+k+1}) \tag{2.24}$$

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

$$v_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \sum_a \pi(a, s) \sum_{s',r} p(s', r|s_t, a_t)[r + \gamma v_\pi(s')] \tag{2.25}$$

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 tenzors Q ar elementu, kas atbilst katrai iespējamai (s, a) vērtībai, tam tiek piešķirta kāda sākotnējā vērtība (piemēram, 0).

Apmācība notiek, izvēloties

$$a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t = s, a) \tag{2.26}$$

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

$$Q^{i+1}(s_t, a_t) = f(Q^i(s_t, a_t), G_t) \tag{2.27}$$

Kaut gan šai metodei ir teorētiskas konverģences 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 modeļi, 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 starpstāvokļiem. 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 nenulles vērtību. Tieši šādu trūkumu mēģina risināt metodes, kas kombinē ekspertu demonstrāciju aproksimēšanu ar adaptīvu pielāgošanos [28]

3. ATDARINOŠĀ MAŠĪNMĀCĪŠANĀS – PĀRSKATS

Šīs nodaļas mērķis ir izveidot aptuvenu nozares pētniecības saturisku pārskatu; 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.

Varētu sacīt, ka tieši par atdarinošo mašīnmācīšanos rakstīts ir samērā maz. Noteikti, ja salīdzina ar vispārīgākām metodēm vai rīkiem. Taču pat "samērā maz" tomēr nozīmē ļoti lielu publikāciju skaitu, kas apraksta pētījumus ļoti dažādos virzienos. Turklāt robo-tika dominē kā pielietojuma mērķis šādām metodēm. Lai radītu priekštatu par nozares pašreizējo stāvokli un tematisku dalījumu, nolemts izšķirt trīs virzienus, kas labi apraksta lielu daļu no pētījumiem par iespējām robotus apmācīt ar piemēriem:

1) trajektoriju kopēšana — mērķi šeit pamatā 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. Kā panākt, ka neaprobežojamies ar tikai piemēros esošo un spējam pielāgoties? Kā efektīvi uzsākt stimulēto mācīšanos ļoti retinātās atalgojumu telpās?

3.1. 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*) [27, 5]. 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, bet ne 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 ģenerējis 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 apakšnodaļā aprakstīti pētījumi, kuros uzsvars ir uz pietiekami detalizētu demonstrāciju atdarināšanas procesiem.

3.1.1. Vienkāršas metodes

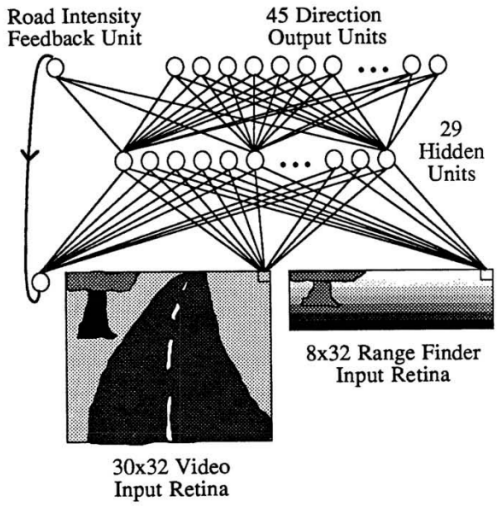
Pirmais, ko varētu darīt, ir tiešā veidā ierakstīt trajektoriju un to atkārtot. Šī nebūt nav jauna ideja — gandrīz visiem mūsdienu industriāliem robotiem ir pieejamas t.s. *lead-through* un *teach-in* programmēšanas metodes, kas ļauj fiziski un ar tālvadības ierīces palīdzību vadīt robota kustību un to ierakstīt pēcākai atdarināšanai [29], turklāt tās parādījušās jau pašos industriālās robotikas pirmsākumos 1970os gados [14].

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 koriģēt 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 [27, 5], ka, lai tik tiešām 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 [4]. Ja dots MDP un kāda eksperta stratēģija π^*, kas šo MDP optimāli risina, mērķis ir atrast maksimāli tuvu modeli π_θ, kur

$$\pi_\theta(s) \approx \pi^*(s) \tag{3.1}$$

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. 1: ALVINN modeļa uzbūve [33]

piemēru. Uzvedības klonēšana sistēmā, kur netiek veikta intensīva treniņa datu pārstrāde vai vadības algoritma argumentu pārveidošana ļoti labi kalpo šādiem mērķiem, tāpēc piemēram var izmantot 1989. gadā Kārnegija-Melona Universitātē veikto pētījumu "*Autonomous Land Vehicle in a Neural Network*" (ALVINN) [33]. Tā mērķis bija izstrādāt pašbraucošu automašīnu, kas spēj sekot ceļa kontūram.

Automašīna tikusi aprīkota ar videokameru un LIDAR sensoriem, kas devuši divus skatus uz to pašu telpas reģionu automobiļa priekšā. Par apmācāmo modeli izvēlēts neironu tīkls. Protams, 1989. gads vēl bija laiks, kad datoru veiktspēja bija stipri ierobežota, un nevienam vēl nebija ienācis prātā būvēt tik dziļas, daudzskaitlīgas un sarežģītas tīklu arhitektūras kā mūsdienu konvolūciju tīklus vai transformatorus. Tāpēc neironu tīkls ir gaužām līdzīgs jebkurā mācību grāmatā pirmajā nodaļā atrodamajiem piemēriem — tam ir viens slēptais slānis ar 29 perceptroniem, kam seko 45 izvades elementi. Video izmantots krāsainā 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.

Jau šim (šķietami) samērā vienkāršajam uzdevumam 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. Jāatzīst, ka ar modernākiem datorredzes modeļiem droši vien šī nepieciešamība mazinātos. 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 maksimāli tuvināts simulatorā realizētajam kon-

troles algoritmam izmantotā šablona iespēju robežās. Tas 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. Tas tiek lietots kā arguments par neironu tīklu pavērtajām iespējām pašbraucošo auto attīstībā, taču netiek slēpts, ka sasniegtais ir tālu no praktiskas vadības sistēmas.

Kā galvenais uzvedības klonēšanas trūkums parasti tiek minēta nespēja atgūties no faktiskā stāvokļa sadalījumu nobīdes [4] (*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, π_θ inducētais stāvokļu sadalījums diverģē no π^* inducētā. Lai iegūtu precīzāku un robustāku eksperta stratēģijas atdarinājumu, piedāvāti dažādi — sarežģītāki — apmācības paņēmieni.

3.1.2. Statistiskas korekcijas

Viens no virzieniem, kurā veikti mēģinājumi uzlabot trajektoriju kopēšanas lietderību, ir strukturēt treniņa datu ieguvi un apmācības algoritmu veidā, kas maksimāli tuvina π^* un π_θ 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 kļūt visnotaļ sarežģītas [4], 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*" [37], piedāvāts algoritms. Tas pierāda teorētiskas garantijas π^* un π_θ inducēto sadalījumu konverģencei, kombinējot instruktāžu ar apmācāmā modeļa ģenerētām stratēģijām. Lai gan raksts nedarbojas tieši 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 uzdot 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 α izvēlas, vai i-tā trajektorija tiks ģenerēta ar π^* vai π_θ;
2) iegūtās laikrindas stāvokļa elementiem s_t atrod atbilstošo $a_t^* = \pi^*(s_t)$
3) kopējai datu kopai D pievieno $D_i = \{(s_1, a_1^*), ...\}$
4) apmāca modeli π_θ uz papildinātās datu kopas

Kā jau minēts, liela daļa raksta satura veltīta tieši 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 π^* aproksimēšanas no treniņa datu kopas) rezultātus, tā lietderību stipri ierobežo instruktora interaktivitātes prasības — praksē reti kad ir iespējams kaut kas analoģisks datorspēļu aģentu treniņam izmantotajam simulatoram, kas ar dziļu pārlasi atrod labas stratēģijas jebkuram stāvoklim.

3.1.3. Inversā stimulētā mācīšanās

Cits veids, kā atdarināt instruktora dotas trajektorijas, ir pieņemt, kas 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,

$$R^*(s) = w^* \cdot \phi(s) \tag{3.2}$$

kur w^* — svaru vektors, bet $\phi : S \to [0,1]^k$ — zināmu atribūtu izpausme noteiktos stāvokļos. ϕ 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

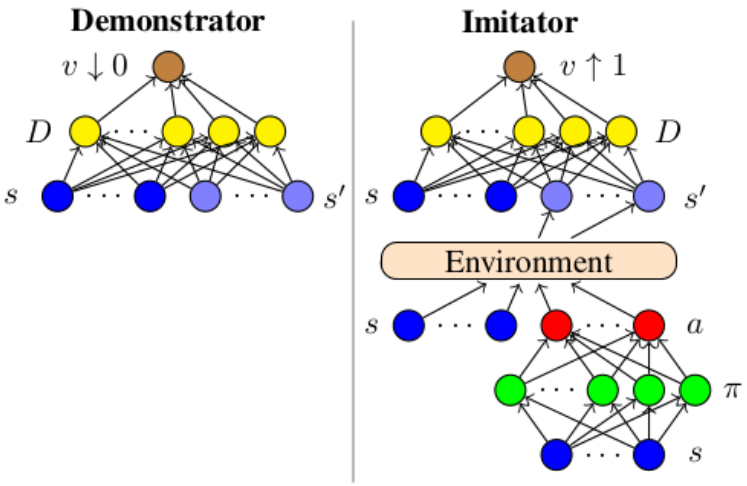
$$\mu^* = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T} \gamma^t \phi(s_t^i) \tag{3.3}$$

Tad tiek iteratīvi atkārtota procedūra, kur atrod kādu svaru vektoru w^i un attiecīgi empīrisku atalgojuma funkciju $R^i(s) = {w^i}^T \phi(s)$, ko izmanto, lai apmācītu jaunu stratēģiju π^i. Tad šai stratēģijai atrod vidējo vērtību μ^i analoģiski (2.3), un visas iepriekšējās $\mu^{j \leq i}$ tiek izmantotas, lai atrastu nākamo svaru vektoru w^{i+1}. Process turpinās, līdz ir konverģējis līdz noteiktam kļūdas 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 π^* 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.

3.1.4. Ģeneratīvie 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 ģenerētas trajektorijas un līdz ar to varētu novērtēt to sasniegto stāvokļu atribūtu sadalījumus.



Att. 2: GAN uzbūves piemērs. Augšējais tīkls — diskriminators — cenšas atšķirt eksperta demonstrācijas no ģeneratora radītām trajektorijām [50]

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*" [17]. 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 [50].

Ģeneratīvie pretinieku tīkli — GAN, *generative adversarial networks* — ir neironu tīkli, kas apmācīti visai īpatnējā veidā. Tā vietā, lai optimizētu visu modeli 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. Ģenerators no sistēmas stāvokļiem vai novērojumiem ģenerē darbības tā, lai diskriminators nebūtu spējīgs atšķirt tās no parauga. Formāli aprakstīt veidu, kā līdz šāda modeļa piemērotībai nonākts, ir sarežģīti, tāpat — pierādīt šāda optimizācijas procesa konverģenci. Taču intuitīvi diezgan skaidrs, ka pēc sekmīgas abu modeļu apmācības būs iegūta stratēģija, kas tuvu aproksimē ievades datu sadalījumu.

3.1.5. 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 skaita iemācīties katru uzdevumu pilnībā no jauna. Šī arī nebūt nav jauna ideja — izteikta jau 1980os un 1990os gados [27]. Jau 2002. gadā veikti pētījumi par algoritmiem, kas ļauj aproksimēt trajektorijas elementus ar autonomu, nelineāru diferenciālvienādojumu sistēmām [19] 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 [38], kur mašīnmācīšanās lietojums attiecināts ne tikai uz atsevišķajiem trajektorijas primitīviem, bet arī uz katrai demonstrācijai atbilstošas to secības meklēšanu.

Robotikas literatūrā pirms 2010. gada [5] 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*" [52], kas publicēts 2020. gadā, šāda pieeja tiek veiksmīgi izmantota uzdevumiem, kur svarīga ne tikai telpiskā trajektorija, bet arī uz apkārtējo vidi izdarīto spēku profils (konkrēti — galda virsmas tīrīšanā).

3.2. 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. Turklāt, 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, nav daudz sarežģītāk 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. Problēmas 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.

3.2.1. 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. Ja iepriekšējā nodaļā aplūkoto eksperimentu veicēji pārsvarā pieņēmuši, ka pieejami pareizi formatēti dati, kuros novērojums ir cieši sakarīgs ar robota vadībai svarīgiem sistēmas atribūtiem un zināmas katrā trajektorijas solī veiktās darbības, tad šajā tiek apskatīti gadījumi, kad šie dati ir kādā ziņā nepietiekami vai pārveidoti.

Pirmais sarežģījums, ko varētu ieviest, ir darbību trūkums demonstrācijās. Ar to jāsastopas ļoti daudzos uzdevumos — no nemarķētiem datiem var kā nebūt 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 rezultē pārejā no viena stāvokļa uz nākamo.

"*Behavioral Cloning from Observation*" [49] (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 tad algoritma soļi ir sekojoši:

1) doto trajektoriju kopu pārveido formā $\mathcal{T}_{dem} = \{(s_t, s_{t+1})\}$, kas ir pārejas starp stāvokļiem;
2) stratēģija π_θ un sistēmas dinamikas modelis $M_\phi(a|s, s') = P(a|s, s')$ tiek nejauši inicializēti;
3) ģenerē trajektorijas ar π_θ, 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 π_θ uz $\{(s_t, \underset{a}{\operatorname{argmax}}\, M_\phi(a|s, s'), s')|(s, s') \in \mathcal{T}_{dem}\}$
6) atkārto soļus 3-5 līdz sasniegti pieņemami rezultāti

Redzams, ka pakāpeniski tiek iegūts dinamikas modelis, kas pareizi paredz pārejas funkcijas slēpto 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, π_θ 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 tādas, ja tiek vērtēts pēc nepieciešamo demonstrācijas trajektoriju vai simulācijas iterāciju skaita.

Iepriekšējā apakšnodaļā 2. attēls ir no "*Generative Adversarial Imitation from Observation*" [49] (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 π_θ) tiek trenēts, lai tā izvēlēto darbību rezultātā iegūtās stāvokļu pārejas $s = T(s, a)$ nebūtu atšķiramas no piemēriem.

3.2.2. 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 slogojošie spēki un griezes momenti, tad var meklēt veidus, kā šīs novirzes treniņa datu ieguves procesā kompensēt. "*Online Movement Adaptation based on Previous Sensor Ex-*

periences" [30] (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. Tiek 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 līdzi nobīdēm no normas un ar klasiskās kontroles teorijas metodēm veikt korekcijas.

Nodarbojoties ar ļoti sarežģītiem uzdevumiem — piemēram, salikšanas procesiem — mēģināt vadīt robotu cauri visiem iespējamiem 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*" [40] (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 [39];
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 tīkls — π_θ tiek apmācīta paredzēt nākamo spēku/momentu vektoru laikrindā, un tas tiek izmantots robota kontrolei ar minēto vadības algoritmu.

3.2.3. Demonstrācijas no cilvēka darbībām

Ja nepieciešams ātri un lēti radīt treniņa datu kopas apmācības procesiem — kā tas noteikti būtu jebkurai 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*" [20] (2017) ir ilustratīvs piemērs. Tiek izmantota *Microsoft Kinect* — savulaik populāra, videospēļu vadībai paredzēta kustību uztveres ierīce, kas spēj ierakstīt kāda objekta kustību aprakstošu punktu virkni telpā — lai ierakstītu cilvēka kustības. Tad ar

analītiskām metodēm tiek veikta trajektoriju interpolācija, grupēšana ar klasterizācijas algoritmu, un izpildē vēlamo trajektoriju iegūst 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*" [53] (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 dziļumiem, ko arī tālāk var izmantot konvolūciju neironu tīklā, lai realizētu uzvedības klonēšanu.

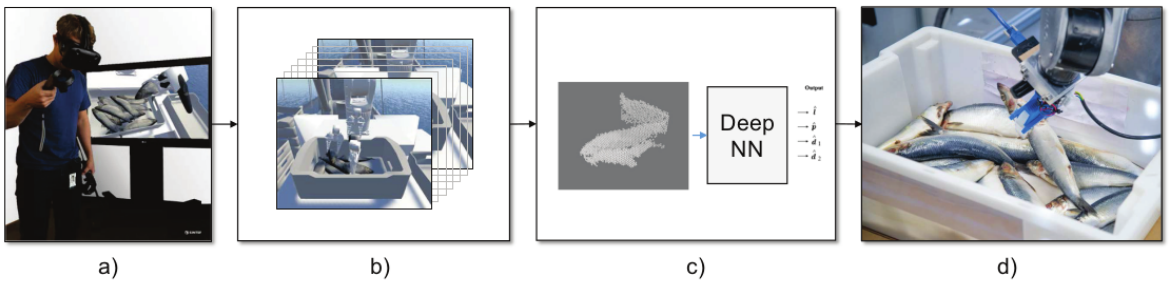
3.2.4. Video demonstrācijas, perspektīvu pārbīde

Ļoti plašas iespējas demonstrāciju iegūšanai pavērtu 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 [50] 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 kinemātiku aprakstošiem stāvokļiem, GAN vienkārši tiek apmācīts diskriminēt/ģenerēt trajektoriju vizuā-lās 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 ģenerējošo. "*Imitation from observation: Learning to imitate behaviors from raw video via context translation*" [23] (2018) risina šo atšķirīgo kontekstu sarežģījumu un iegūtu modeli izmanto arī rezultātu uzlabošanai vispār. 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 soļiem 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 ziņā sevī ietver visus nozīmīgos konfigurācijas atribūtus.

3.2.5. 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 paļauties 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



Att. 3: Zivs satveršanas modeļa apmācība: a) cilvēka radīto demonstrāciju iegūšana; b) datu kopas sintētiska pavairošana; c) telpiska konvolūciju neironu tīkla apmācība; d) veiksmīga zivs satveršana realitātē. [10]

ar cilvēka roku darbu.

"*Teaching a robot to grasp real fish by imitation learning from a human supervisor in virtual reality*" [10] ir visnotaļ interesants vairāk praktisks projekts, kura ietvaros izstrādāts modelis zivju satveršanas uzdevumam. Šo rakstu ir vērts nedaudz sīkāk aplūkot, jo problemātika ir radniecīga darba mērķī formulētajam uzdevumam — industriālā robota darbības ar neregulāras formas objektiem. Tajā robotam ir jāspēj no kastes, kurā atrodas vairākas zivis, satvert un pārvietot vienu. Zivs ir slidens objekts, kā veiksmīga satveršana ar robota efektoru iespējama tikai ļoti ierobežotā iespējamo kontakta konfigurāciju apakškopā, tāpēc no praktiskas realizācijas viedokļa projekts ir visai ambiciozs.

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 divdimensionāla 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. Rezultātā konstatēts, ka aptuveni 74% izdarīto satveršanas mēģinājumu bijuši sekmīgi.

3.3. Atdarināšana un adaptācija, vispārināšana

Ja pieejamas demonstrācijas no eksperta, to atdarināšana tiešā veidā varbūt ir pirmais 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 mēģinājumi darboties šajā jomā nav nekas jauns — var atrast senākus pētījumus, kuros, piemēram, izmantotas sarežģītas un tieši robotikas mērķiem specifiskas simboliskās dekompozīcijas metodes [31] — uzsvars šeit tiek likts uz nesenākām un vispārīgākām metodēm, kas attīstītas jau pašreizējā neironu tīklu uzplaukuma periodā.

3.3.1. Neoptimālu demonstrāciju uzlabošana

Varētu sacīt, ka 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*" [7] (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 τ, tiek meklēta tāda, kas skaidro to *sakārtojumu*. Tāpēc nepieciešams datu kopu papildināt ar kārtojumu $\prec$, lai

$$\tau_i \prec \tau_j \Rightarrow \sum_{s_t \in \tau_i} R(s) \leq \sum_{s_t \in \tau_j} R(s) \tag{3.4}$$

Protams, kārtojums var arī nebūt ideāls — trajektoriju vērtējums var būt subjektīvs vai trokšņains, ja nav zināma īstā atalgojuma funkcija, tāpēc jārēķinās ar kļūdainu pāru attiecību pastāvēšanas iespējamību ar kļūdas varbūtību ϵ

$$\exists \epsilon \geq 0 : \tau_i \prec \tau_j \Rightarrow P\left(\sum_{s_t \in \tau_i} R(s) \leq \sum_{s_t \in \tau_j} R(s)\right) = 1 - \epsilon \tag{3.5}$$

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

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

Eksperimentāli pārbaudīts, ka piemēros, kur zināmas demonstrācijām atbilstošās īstās atalgojuma funkcijas un kļūdas varbūtība saglabājas zem 15%, rekonstruētā atalgojuma funkcija r_ϕ daudzos gadījumos labi aproksimē īsto. Turklāt, ja izmantotās demonstrācijas nav optimālas, iegūtā stratēģija π_θ tās pārspēj.

3.3.2. 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*" [16] (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 soļu — lai šo telpu apstaigātu, atrastu vēlamos rezultātus un optimizētu trajektoriju caur tai.

Q-mācīšanās gadījumā stratēģiju var uzdot formā $\pi^{\epsilon Q_\theta}$, kur

$$T(s,a) = P(s') \to Q_\theta(s,a) \sim \mathbb{E}[v(s')] \tag{3.6}$$

jeb modelis Q_θ aproksimē kopējā atalgojuma sadalījumu pār iespējamām darbībām (jeb nākamo stāvokļu vērtības), un attiecīgi izvēlas vienu pēc "ϵ-alkatīgas" stratēģijas

$$a = \underset{a}{\operatorname{argmax}}\, Q(s,a) \text{ ar varbūtību } 1-\epsilon;\ \text{nejauši izvēlēta citādi} \tag{3.7}$$

Demonstrācijas tad tiek izmantotas, lai apmācītu Q_θ 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.

3.3.3. Tūlītēja trajektoriju atdarināšana

Daudzas līdz šim aplūkotās atdarinošās metodes darbojas ar pieņēmumu, ka mērķis 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 neredzētas 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 ticis optimizēts vienas stratēģijas atdarināšanai, bet gan tāds,

kas vispārina pašu atdarināšanas procesu — ievadē sistēmas pašreizējais stāvoklis tiek apvienots ar demonstrāciju, lai iegūtu vēlamo darbību.

"*One-Shot Imitation Learning*" [9] (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 kuba formas 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.

3.3.4. 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, kas iet caur abām. "*Learning Latent Plans from Play*" [25] (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.

Tiek konstatēts, ka, pilnīgi nejauši ģenerējot trajektorijas, paies visnotaļ ievērojams laiks, līdz tiks apstaigāts pietiekami plašs konfigurāciju telpas reģions, lai gūtu labas trajektorijas starp jebkuriem punktiem tajās. Atšķirībā no nejauša procesa, cilvēks jau nāk ar sagatavotu izpratni par tāda tipa uzdevumu risināšanas elementiem, kas pēc tam varētu būt interesanti citiem. Brīvi pētot dažādās simulācijas vidē pieejamās iespējas, tiks dabiski izmantotas priekšzināšanas par dažādām fizikālām sakarībām starp objektiem un to mijiedarbību — piemēram, rokturu satveršanu, lai atvērtu un aizvērtu durvis, pogu nospiešanu, objektu satveršanu un pacelšanu.

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 $s_1, s_2, ..., s_n$, 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. Vienkāršojot — ignorējot sistēmas stāvokļa reprezentāciju kodēšanas detaļas — pirmo var aprakstīt samērā vienkārši. Modelis

$\pi_\theta(s, s_g)$ saņem ievadē pašreizējo un vēlamo stāvokli, un paredz nepieciešamo darbību. Kā šablons tiek izmantots rekurentais neironu tīkls, treniņa datus veido trajektorijas — pēdējais trajektorijas stāvoklis kļūst par s_g. π_θ 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 apgrūtināt mācīšanās procesu.

Otrs, sarežģītākais modelis apkaro šo parādību, modelējot plāna jēdzienu — tiek pieņemts, ka katrai trajektorijai τ 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 analoģiski 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 diverģence starp tiem — ļaujot 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 sasniegs. Rezultātu sekcijā secināts, ka šāda mācīšanās spēj pārspēt klasisko uzvedības klonēšanas metodi 18 dažādiem uzdevumiem — kaut gan otra izmantojusi demonstrācijas, kas speciāli sagatavotas katram. Turklāt, pateicoties faktam, ka modelis apgūst trajektorijas no katra konfigurāciju telpas punkta uz katru citu, tā ir ļoti robusta pret nobīdēm. Visbeidzot, ar dimensiju redukcijas metodēm aplūkojot iegūto rīcības plānu reprezentācijas telpu redzams, ka pēc cilvēkam saprotama mērķa līdzīgas darbības veido klasterus arī šajās projekcijās.

Interesants turpinājums šim pētījumam ir "*Language conditioned imitation learning over unstructured data*" [24], kur iegūtais rezultāts atkal pagriezts otrādi — ja ir 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 daļai 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*" [15] (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.

Pamatā risinājumam ir hierarhiskā stimulētā mācīšanās — tiek iegūta nevis viena stratēģija, bet divas. π_θ^h jeb augsta līmeņa stratēģija ievadē saņem pašreizējo sistēmas stāvokli s un gala stāvokli s_g, bet tā vietā, lai izvadītu vēlamo darbību, rezultātā izdod lokālu mērķi s_g^l — argumentu zemākā līmeņa stratēģijai π_ϕ^l. Otrā tad atkarībā no s, s_g^l ģenerē vēlamo darbību vidē a. Apmācība tiek realizēta divos soļos:

1) atdarināšana — katrai no stratēģijām tiek sagatavota atsevišķa datu kopa, izmantojot "slīdošā loga" principu pār trajektorijām demonstrāciju datos. π_θ^h izvēlas plašāku logu W_h un minimālo distanci līdz lokālajam mērķim j. Katram s_t tad tiek izveidoti korteži (s_t, a'_t, s_g), kur $s_g \in \{s_{t+1}, s_{t+2}, ..., s_{t+W_h}\}$ bet $a'_t = s_{t+j}$. Zema līmeņa stratēģijai π_ϕ^l tad izvēlas īsāku logu W_l un analoģiski katru tālāk logā esošo s_{t+i} izmanto kā galamērķi s_g^l, lai izveidotu (s_t, a_t, s_g^l). Šādi pārveidotos datus tad izmanto, lai apmācītu modeļus π_θ^h, π_ϕ^l atdarināt datu kopu;
2) uzlabošana — izmantojot zināmu atalgojuma funkciju $r(s, a, s_g)$, realizē stimulēto mācīšanos. Iegūtās stratēģijas tiek darbinātas vidē, iegūtie dati tiek tāpat pārkārtoti un π_θ^h, π_ϕ^l atkal tiek optimizēti, tikai šoreiz gradientu optimizācijas mērķa funkcija (*loss*) tiek papildināta ar atalgojuma faktoru.

Tiek iegūts algoritms, kas spēj no nestrukturētas datu kopas iemācīties, kā sasniegt patvaļīgus galamērķus, bet pēc tam — uzlabot iegūtās stratēģijas ar stimulētās mācīšanās palīdzību. To parāda rezultāti — dažādos uzdevumos šāda pieeja pārspēj iepriekš aprakstīto latento plānu pieeju [25], taču attiecīgi tai ir nepieciešams papildu patstāvīgas mācīšanās etaps.

4. ROBOTIKAS TEORĒTISKIE PAMATI

4.1. Robota matemātiskais modelis – kinemātiskās ķēdes

4.1.1. Tiešie un inversie kinemātikas uzdevumi

4.1.2. Rotāciju kodēšana kvaternionos

4.2. Trajektoriju plānošana

4.2.1. Dinamika

4.2.2. Metodes, plānotāji

4.3. Kinemātikas rekonstrukcija no novērojumiem

5. IZMANTOTIE RĪKI, APRĪKOJUMS UN PLATFORMAS

5.1. ROS

5.2. Optitrack – aprīkojums un programmatūra

kas ir odometrija – vismaz tik daudz jāpasaka!!

5.3. Programmēšanas valoda, bibliotēkas

6. PRAKTISKĀ REALIZĀCIJA

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

6.1. Izvēlētā 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. Kaut gan iespējams metienus ģenerēt parametriski, izmantojot t.s. *"model-based"* pieejas – kur tiktu speciāli izveidots fizikā un robota kinemātikā balstīts sistēmas modelis, un metieni ģenerēti balstoties uz mērķa koordinātēm – šāda tradicionāla metode ir, pirmkārt,dārga, darbietilpīga un slikti visparināma, kā jau apspriests darba teorētiskajā daļā, un, otrkārt, nekādi nepalīdzētu papildināt institūta zināšanu bāzi ar ieskatu atdarinošās mašīnmācīšanās metodēs. Līdzīgi apsvērumi attiecināmi uz stimulēto mācīšanos, kaut gan tādā gadījumā vēl jāņem vērā fakts, ka tehniska realizācija būtu pat sarežģītāka un darbietilpīgāka, nekā zemāk aprakstītie paņēmieni.

Kad nolemts darboties atdarinošās mācīšanās virzienā, pirmais 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 iegūtas, 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 uztveršanu 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 šī 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 droši vien būs 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 kontrolera 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 aparatūras viedokļa vislētāk būtu izmantot video ierakstu, taču tad modelim jāveic ļoti sarežģīts uzdevums, izlobot no netiešiem novērojumiem robota kontrolei aktuālos parametrus. Var izmantot kustības uztveres tehnoloģiju un īpaši izgatavotus instrumentus, lai visnotaļ 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 marķierus manipulatoru atdarinošam instrumentam un citiem objektiem. Galvenais trūkums ir sarežģītāks datu priekšapstrādes process, taču tas tomēr šķiet mazāk sarežģīti, nekā sagatavot simulācijas vidi vai tiešā veidā izmantot pašu robotu demonstrāciju iegūšanai. Attiecīgi demonstrāciju formāts ir jau pārveidoti novērojumi – efektora odometrija Dekarta koordināšu telpā. To priekšapstrādes procesa gaitā nepieciešams papildināt ar satvērējmehānisma vadības signālu.

Kad zināms demonstrāciju formāts, iespējams izdarīt nākamo lēmumu – izvēlēties modelim un robota kontrolerim pieejamo ievades datu formātu. Pastāv divas galvenās iespējas:

1) modelis ģenerē trajektorijas Dekarta koordināšu telpā, kontroleris veic plānošanu un pārveidošanu;
2) modelis ģenerē trajektorijas jau locītavu konfigurāciju telpā, kontrolera uzdevums ir to izpilde;

Realizējot otro, būtu stipri atvieglots robota kontrolera 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 kļūst 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 visnotaļ netriviāli.

Savukārt izvēloties modeli, kas veic regresiju tīri Dekarta telpā, pats modeļa arhitektūras un apmācības jautājums kļūst 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 par šo atvieglojumu no mašīnmācīšanās viedokļa jāmaksā ar daudzkārt sarežģītāku klasisko robota kontroles uzdevumu. 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 mašīnmācīšanas metožu pētīšanu –, pieejamos resursus un laika ierobežojumus, nolemts modeli konstruēt Dekarta koordināšu telpā kā autoregresoru diskrētā laikā. T.i., katra "darbība" ir vienkārši paredzētais sistēmas stāvoklis nākamajā laika solī. Iespējami MDP formālismam atbilstoši

$$s_{t+1} = \pi_\theta(s_t) \tag{6.1}$$

vai neatbilstoši (laikrindu ekstrapolācijas)

$$s_{t+1} = \pi_\theta(s_t, s_{t-1}, ..., s_{t-n}) \tag{6.2}$$

modeļi, un iepriekš ir grūti 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 redzams zemāk, jau ar šādu risinājumu motivējošajā uzdevumā sasniegti 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 zināmām metrikām sasniedz labākus rezultātus. Tālākas pētnieciskās darbības nolūkos – meklējot veidus, kā ar MDP formālismam atbilstošu, pret trajektorijas garumu laikā un atmiņā konstantu stratēģiju sasniegt RNN pielīdzināmus rezultātus – sākta arī GAN modeļa izstrāde.

Trajektoriju izpilde uz fiziskā robota nav īpaši augsta prioritāte no pētnieciskā viedokļa, un tīri praktiski ierobežojumi pieejamajā Dekarta telpā dotu maršrutu plānošanas programmatūras klāstā nozīmē, ka laikā precīzai izpildei būtu nepieciešams veikt lielu programmēšanas darbu apjomu, kam nav zinātniskās novitātes. Taču, lai provizoriski novērtētu iegūto trajektoriju atbilstību robota kinemātiskajiem ierobežojumiem, veikta arī izpildes programmatūras prototipa izstrāde – laika paramatrizāciju 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 veiktspējas novērtēšanas metrikām ir sasaiste ar ģenerēto trajektoriju telpisko izpildījumu.

6.2. Datu priekšapstrāde

Pareizas tīkla un ierakstīšanas programmatūras iestatīšanas gadījumā *"Optitrack"* rīka iegūtie novērojumi pieejami *"ROS"* vidē, izmantojot *"topic"* – tematu – 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 maģistra darba *"github"* repozitorijā, *"trajectory_extract"* direktorijā [35]. Datu korpusi saglabāti atsevišķā repozitorijā [36].

6.2.1. Ierakstu veikšana, sākuma datu kopas ieguve

Lai saglabātu kādā *"ROS"* tematā publicētos ziņojumus noteiktā laika periodā, var izmantot *"rosbag"* lietojumprogrammu ar *"record"* komandu. Argumentos iespējams norādīt, tieši kurus tematu saturu nepieciešams ierakstīt – ietaupot vietu uzglabāšanas atmiņā. Šajā gadījumā nepieciešamā informācija pieejama augstāk aprakstīto instrumentu – efektora simulatora (*"TrashPickup"*), ar marķieriem aprīkotās pudeles (*"Bottle"*) un brīvā kritiena trajektorijas beigu indikatora (*"CatchNet"*) – odometrijās. Lietojumprogrammas izvadā tiek iegūts strukturēts fails *".bag"* formātā, ko tiešā viedā izmantot nav vienkārši.

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 ir visnotaļ vienkāršs, taču aizņem visilgāko laiku visā priekšapstrādes procesā, jo izmantotā *"bagpy"* bibliotēka, šķiet, nav sevišķi labi optimizēta. Taču šo soli pietiekami veikt vienu reizi pēc katra fizisko metienu ierakstīšanas etapa, kas tiek darīts reti, tāpēc veiktspējai nav īpaši lielas nozīmes. Datu korpusa iegūšanas procesā no visa ieraksta tiek atlasītas tikai tālākām operācijam svarīgas kolonnas.

6.2.2. 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 – kam, ja nepieciešams, lai iegūtā trajektorija spētu atspoguļot ne tikai pozīciju un orientāciju, bet arī to atvasinājums laikā, 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;
2) intervāli starp novērojumiem ir nejauši, un faktiski sastāv no īsām (dažu milisekunžu) garām virknēm, kurās saņemti novērojumi par vienu no ķermeņiem;

Abu rezultātā saņemot datu virknes ilustratīvi pieņem sekojošo formu:

$$s^a_{t1}, s^a_{t2}, ..., s^b_{tm-1}, s^b_{tm}, \tag{6.3}$$

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

$$(s^a_{t'1}, s^b_{t'1}, s^c_{t'1}), ..., (s^a_{t'p}, s^a_{t'p}, s^c_{t'p}), ... \tag{6.4}$$

kur $t'_k = \frac{n}{f} : n \in \mathbb{N}$ un f – diskrētā laika soļu frekvence. Attiecīgo pārveidojumu veic skripts *"regularize.py"*. Vispirms datu kopa tiek ielasīta atmiņā no *.csv* faila, izmantojot *"pandas"* datu korpusu apstrādes bibliotēku. Tiek atrasti tuvākie sākuma un beigu laiki, kas pieder diskrēto laika soļu rindai:

$$t'_1 = min(\{t'_k \mid t'_k > t_1\}) \tag{6.5}$$

$$t'_p = min(\{t'_k \mid t'_p > t_m\}) \tag{6.6}$$

un, izmantojot *"numpy.arange()"* funkciju, tiek ģenerēta diskrēto laika soļu virkne $t'_1, t'_2, ..., t'_p$. Datu korpuss tiek papildināts ar tukšām rindām šajos laika intervālos (kolīzijas novērstas netiek; duplikāti 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 kropļoti, 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 korpuss tātad atbilst diskrētam laika solim un tajā nav tukšu vērtību.

6.2.3. Atsevišķu demonstrāciju atdalīšana

Tā kā procesa mērķis ir apmācīt modeli, kas spēj izpildīt konkrētu uzdevumu, nevis vienkārši atdarināt visas kustības, kas ierakstītas metienu viekšanas procesā (tai skaitā atgriešanos sākuma pozīcijā, nejaušu pārvietošanos, u.t.t), nepieciešams visu ierakstu sadalīt konkrētās demonstrācijās, kas katra satur vienu metienu. Vispārīgā gadījumā šis ir netriviāls – lai neteiktu neiespējams – uzdevums, tāpēc demonstrāciju ievākšana apzināti veikta tā, lai būtu iespējams atrast katra metiena sākumu. Kustību uztveres telpā uz grīdas izvietots marķējums sākuma pozīcijai. Katrs metiens sākts, vispirms

novietojot efektora simulatoru virs marķējuma un noturot vismaz sekundi. Visā pārējā laikā pievērsta īpaša uzmanība, lai efektora simulators netiktu novietots šajās koordinātēs. Tādējādi iespējams ar vienkāršu slīdošā loga operāciju atrast vismaz vienu punktu neilgi pirms katra metiena:

$$t_{start} \in [t'_k]_1^p : t < t_{start} \wedge |t - t_{start}| < \Delta t_{max} \Rightarrow$$
$$\Rightarrow x_t \in (x_0 - \delta x, x_0 + \delta x), y_t \in (y_0 - \delta y, y_0 + \delta y) \quad (6.7)$$

kur sākuma koordinātes x_0, y_0 un pieļaujamie nobīdes intervālu rādiusi $\delta x, \delta y$ tiek atrasti, cilvēkam pārbaudot ierakstīto datu kopu, atrodot marķētā 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 vienkārši

$$t_j \in [t'_k]_{start}^{start+steps} \quad (6.8)$$

Sekmīgai demonstrāciju iegūšanai tātad 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.

6.2.4. Trajektoriju gludināšana

Ar laika ass reparametrizāciju nepietiek, lai novērstu visas neregularitātes iegūtajos datos. Odometrijas dati saņemti sadrumstalotā formā, turklāt, šķiet, ka laika izkliede starp *"ROS"* vidē saņemtajiem punktiem ir lielāka, nekā reālajiem novērojumiem – novērojama pakāpieniem vai zāģa asmenim līdzīgu kontūru esamība ierakstos. T.i., fiktīvas augstas frekvences nesinusoidālu svārstību komponentes.

Izvērtējot ierakstīto trajektoriju kvalitatīvos aspektus (salīdzinot līganus metienus video ierakstā un to raustītāš reprezentācijas trajektoriju datos), nospriests, ka tās nesastāda fiziskajā trajektorijā novē-rojamu parādību bet gan ieraksta procesā pievienotu datu kropļojumu. Tāpēc pirms modeļa apmācības tās nepieciešams likvidēt. To paveic skripts *"smoothing.py"*, kurā piemērots t.s. "slīdošās vidējās vērtības" aprēķins pēc sekojošā vienādojuma:

$$\vec{x}'_t = \frac{1}{2m+1} \sum_{i=t-m}^{t+m} \vec{x}_i \quad (6.9)$$

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

6.2.5. Kritisko punktu noteikšana un papildu signāli

Šādi ir iegūtas gludinātas, nošķirtas kinemātiku trajektorijas. Taču katrā no tām metiens var sākties pēc dažāda garuma stacionāra perioda, trajektorija satur arī ne-

jaušas kustības pēc metiena beigām, kas traucētu apmācīt modeli, un nav nekādas informācijas par satvērējmehānisma stāvokli (ierakstītas tikai efektora, pudeles un gala stāvokļa marķiera odometrijas; sk. sadaļu par kustības uztveres aprīkojumu). Turklāt, lai būtu iespējams parametrizēt metienus pēc mērķa koordinātēm, nepieciešams katru demonstrāciju papildināt ar paredzamo pudeles maršruta krustpunktu 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 apakšnodaļā aprakstītā regresijas un mērķa koordināšu noteikšanas soļa izpildei. Lai neņemtu vērā pudeles piezemēšanos tīklā, zem tā novietots beigu pozīcijas marķieris (vecais satveršanas tīkls, *"Optitrack"* kontekstā – ķermenis *"CatchNet"*). Lidojuma ballistiskā fāze uzskatāma par pabeigtu, kad pudeles x koordināte vismaz vienu reizi sasniegusi šo marķieri:

$$f_{passed}(t) = \begin{cases} 0 \text{ ja } \forall u < t, x_{Bottle}(u) \geq x_{CatchNet}(u) \\ 1 \text{ citādi} \end{cases} \tag{6.10}$$

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

$$x'(t) \approx \frac{x(t'_{k+1}) - x(t'_k)}{\Delta t'} = f \cdot \left(x(t'_{k+1}) - x(t'_k)\right) \tag{6.11}$$

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. Varētu aprēķinātajam "atvasinājumam" izmantot to pašu "slīdošās vidējās vērtības metodi", kas aprakstīta pie gludināšanas. Praksē izrādās, ka pietiekami labi strādā sekojošais, stipri vienkāršotais novērtējums, kas apvieno diferences un gludināšanas soļus:

$$x'_t \propto \overline{x'_t} = \sum_{i=t}^{t+m} \vec{x}_i - \sum_{i=t-m}^{t} \vec{x}_i \tag{6.12}$$

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:

$$f_{freefall}(t) = \begin{cases} 0 \text{ ja } \forall u < t, \overline{\left[\|\vec{r}_{Bottle}(u) - \vec{r}_{TrashPickup}(u)\|\right]'_u} < \overline{v}_{freefall} \\ 1 \text{ citādi} \end{cases} \tag{6.13}$$

kur $\vec{r}_{Bottle}$ apzīmē pudeles novietojuma vektoru, u.t.t. Lai noteiktu, vai satvērējmehānisms ir sācis atlaisties, tas pats tiek darīts ar šī attāluma otrā atvasinājuma aproksimatoru:

$$f_{release}(t) = \begin{cases} 0 \text{ ja } \forall u < t, \overline{[\|\vec{r}_{Bottle}(u) - \vec{r}_{TrashPickup}(u)\|]''_u} < \overline{a}_{release} \\ 1 \text{ citādi} \end{cases} \tag{6.14}$$

Visbeidzot, lai atrastu paša metiena – ātras kustības – sākuma momentu, slieksnis tiek piemērots efektora simulatora pozīcijas atvasinājuma novērtējumam (ievērojot, ka iepriekš "atvasināta" tiek distances vektora norma, bet šoreiz tiek meklēta norma vektora "atvasinājumam"):

$$f_{moving}(t) = \begin{cases} 0 \text{ ja } \forall u < t, \|\overline{[\vec{r}_{TrashPickup}(u)]'_u}\| < \overline{v}_{moving} \\ 1 \text{ citādi} \end{cases} \tag{6.15}$$

Sliekšņa konstantes $\overline{v}_{freefall}, \overline{a}_{release}, \overline{v}_{moving}$ tiek piemeklētas empīriski. Pieredze liecina, ka process ir diezgan robusts pret šīm vērtībām. Tālākā darbībā varētu būt vērts ievākt detalizētu informāciju par šo konstanšu izvēles ietekmi uz metienu precizitāti, taču jau pēc vizuālas laikrindu grafiku novērtēšanas izvēlēti skaitļi izrādījušies pietiekami precīzi, lai būtu iespējams realizēt pudeles metiena paraugdemonstrāciju ar fizisku robotu. Protams, it sevišķi satvērējmehānisma gadījumā, pastāv iespēja iegūt precīzākus datus, papildinot demonstrāciju ierakstīšanas aprīkojumu ar kādu sensoru, kas var nomērīt šīs vērtības tiešā veidā.

6.2.6. Brīvā kritiena ekstrapolācija, mērķa koordināšu 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. Protams, tā kā 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 marķiera z-koordināte z_{ref}. Vispirms tiek atrasta demonstrācijas brīvā kritiena fāze pēc nosacījuma

$$\mathcal{D}_{fi} = \{s \in \mathcal{D}_i \mid f_{freefall}(t(s)) \wedge \neg f_{passed}(t(s))\} \tag{6.16}$$

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 ašķirties 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 pamanīt – un, visticamāk, pie šobrīd sagaidāmās procesa precizitātes lielu iespaidu uz rezultātu neatstāj.

Lai atrastu mērķa koordinātes, var izmantot vienkāršāko ballistiskās trajektorijas modeli. Pieņem, ka xy plakne ir precīzi normāla gravitācijas paātrinājuma vektoram, tāpēc šīm koordinātēm piemēro lineāro regresiju:

$$x_{pred} = \theta_{x1}(t) + \theta_{x0} \tag{6.17}$$

$$y_{pred} = \theta_{y1}(t) + \theta_{y0} \tag{6.18}$$

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

$$z_{pred} = \theta_{z2}(t^2) + \theta_{z1}(t) + \theta_{z0} \tag{6.19}$$

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

$$\theta_{z2}(t_{-1}^2) + \theta_{z1}(t_{-1}) + \theta_{z0} - z_{ref} = 0 \tag{6.20}$$

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

$$x_{target} = \theta_{x1}(t_{-1}) + \theta_{x0} \tag{6.21}$$

$$y_{target} = \theta_{y1}(t_{-1}) + \theta_{y0} \tag{6.22}$$

$$z_{target} = \theta_{z2}(t_{-1}^2) + \theta_{z1}(t_{-1}) + \theta_{z0} \tag{6.23}$$

6.2.7. Sākuma koordināšu kompensācija

Visas demonstrācijas nav iegūtas no tā paša punkta kustību uztveres rīka koordinātu sistēmā, un tas var nesakrist ar robota efektora sākuma koordinātēm. 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 koordināšu sistēmas centrēšanu.

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

$$t_{moving} = min(\{t \mid f_{moving}(t) = 1\}) \tag{6.24}$$

$$\vec{r_0} = \vec{r}_{t_{moving}} \tag{6.25}$$

Bet metiena beigu punktu iegūst no mērķa koordinātēm:

$$\vec{r_0} = (x_{target}, y_{target}, z_{target}) \tag{6.26}$$

Tad korekciju ieviešu, vienkārši atņemot atskaites pozīciju no novērotajām:

$$\vec{r}_{tnorm} = \vec{r}_t - \vec{r}_0 \tag{6.27}$$

6.2.8. Apvienošana, stāvokļu pāreju veidošana, jaukšana

Visu iepriekšējo soļu rezultātā iegūts liels skaits atsevišķu demonstrāciju, kas katra 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 pēdējais skripts – *"preprocess_dataset.py"*, kas atrodams repozitorija *"models/"* direktorijā. Šeit arī notiek laikrindas pārveidošana stāvokļu pāreju formātā:

$$s_t, s_{t+1}, s_{t+2}, ... \rightarrow (s, s')_t, (s, s')_{t+1}, ... \tag{6.28}$$

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:

$$s_t, s_{t+1}, ..., s_{t+n}, ... \rightarrow (s, s'^1, s'^2, ..., s'^n)_t, ... \tag{6.29}$$

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 saglabāts parametriski nosauktos datu kopas *.csv* failos, kuru unikālie identifikatori iegūti, piemērojot jaucējfunkciju korpusa ģenerēšanā izmantoto demonstrāciju failu nosaukumu kopai.

Tā kā to, vai nepieciešams datu kopu jaukt, nosaka konkrētā modeļu apmācības vai validācijas uzdevuma specifika, šis solis tiek atstāts jau nākamo soļu 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.

6.3. Modeļu realizācija

Pateicoties darba ietvaros izmantoto skaitļošanas bibliotēku piedāvātajām iespējām, neironu tīkļu modeļu izstrāde programmatūras kodā ir samērā vienkārša. Par spīti šo modeļu lielajiem parametru skaitiem un sarežģītajiem aprēķiniem – piemēram, mērķa funkcijas gradientu meklēšanai vai optimizācijas metodēm, kas sastāv no vairākiem, dinamiskiem 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 dziļo neironu tīklu un rekurento autoregresoru – un uzsākts darbs arī pie pirmā soļa tālākiem pētijumiem – MDP formālismam atbistoša GAN modeļa.

Viss modeļu apmācības kods ir atrodams maģistra 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 grūti šādu kodu organizēt pēc programminženierijas labās prakses. Tā vietā, katra tipa modeļa apmācībai ir savs skripts, kas satur konfigurācijas mainīgos un visas funkcijas, kas netiek precīzi dublētas visur – pat tad, ja pastāv ievērojamas līdzības starp tām dažādu modeļu izpildījumos.

6.3.1. Parastais 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 nosakumos apzīmēta ar *"naiveBC"* – *"naive behavioural cloning"*, "naivā uzvedības klonēšana". Tas pilda klasisku MDP stratēģijas (vai lakrindu ekstrapolācijas ar vienu stāvokli garu vēsturi) uzdevumu formā

$$a_t = s'_t = \pi_\theta(s_t) \tag{6.30}$$

Līdz ar to treniņa uzdevuma mērķa funkciju iespējams definēt formā

$$\mathcal{L}_{policy} = f\left(s, s', \pi_\theta(s)\right), (s, s') \in \mathcal{D} \tag{6.31}$$

Pastāv dažādas iespējas mērķa funkcijas izvēlē. Tā kā šajā gadījumā modeļa izvads ir nepārtrauktu vērtību vektors, atkrīt dažādas kategoriskās kļūdas un varbūtību sadalījumu distances metrikas, kādas izmantotas daudzos no teorētiskajā daļā apskatītajiem pētījumiem. Tas pats sakāms arī par diskrētu 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):

$$\mathcal{L}_{ss}(\vec{s}, \vec{s'}, \theta) = \sum_{i=1}^{n} (s'_i - \pi_\theta(s)_i)^2 \tag{6.32}$$

Taču jau izsen zināms, 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 izgudrotāja P. Dž. Hūbera vārdā nosauktā *"huber loss"* funkcija [18]:

$$\mathcal{L}_\delta = \sum_{i=1}^{n} \begin{cases} \frac{1}{2}(s'_i - \pi_\theta(s)_i)^2, \text{ ja } |(s'_i - \pi_\theta(s)_i)| < \delta \\ \delta \cdot \left(|(s'_i - \pi_\theta(s)_i) - \frac{\delta}{2}|\right) \text{ citādi} \end{cases} \tag{6.33}$$

Izmantotajā *tensorflow.keras* bibliotēkā šī funkcija ir iebūvēta un pēc noklusējuma iestatīta ar parametru $\delta = 1$ [45]. Papildus veikti eksperimenti arī īpaši šim uzdevumam konstruētām mērķa funkcijām. Viens no izmēģinātajiem papildinājumiem bijis jau teorijas daļā aplūkotajā *QuaterNet* izmantota kvaternionu normas regularizācija [32]

$$\mathcal{L}_q(\theta) = \lambda_q * (\|q^\theta\| - 1)^2 \tag{6.34}$$

kur q^θ apzīmē modeļa ģenerētā izvada $\pi_\theta(s)$ tos elementus, kas kopā veido kvaternionu, bet λ_q – svara koeficientu. Šī regularizācija izmantota, jo pirmajos mēģinājumos apmācīt modeli izmantot kvadrātu summas un Hūbera funkcijas, iegūtās kvaternionu vērtības bijušas ļoti atskirīgas 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

$$\mathcal{L}_{combined}(s, s', \theta) = \lambda_p \|r^\theta - r^{s'}\|^2 + \lambda_q \|q^\theta - q^{s'}\|^2 - \lambda_g g^{s'} \log(g^\theta) \tag{6.35}$$

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 [44]. Tas darīts ar mērķi atvieglot modeļa apmācību, jau uzreiz nokodējot dažado 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 instancētu pašu modeli, 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 katra 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 [47]:

$$\sigma_{ReLU}(x) = \begin{cases} x, \text{ ja } x > 0 \\ 0 \end{cases} \tag{6.36}$$

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 novērstu 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.

Pēc apmācības 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 soļi – trajektoriju ģenerēšana uz gadījuma mērķa koordinātēm un salīdzinājumi gan ar treniņa, gan testa datu kopām. Šis process detalizēti aprakstīts zemāk.

6.3.2. Rekurentais neironu tīkls

Pētot klasiskā neironu tīkla sasniegtos rezultātus, konstatēts, ka dažus trajektorijas parametrus – piemēram, precīzu satvērējmehānisma atlaišānas brīdi – šādam modelim ir grūrības 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 un salīdzinot darba uzdevumu ar citiem, 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. Analoģiski 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 ievaddatu 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" [46], kas paredzēti īpaši šim lietojumam. Tie nodrošina visu *tensorflow* operāciju saderību ar patiešām 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,

$$(s_1, s_2, ..., s_{n-1}) \rightarrow (s_2, s_3, ..., s_n) \tag{6.37}$$

Funkcija *create_sequential_dataset* veic visus trīs uzdevumus un atgriež divus tenzorus – treniņa ievades datus un vēlamās vērtības. Demonstrciju secību jaukt nav nepieciešams, jo visa datu kopa sadalīta sērijās (*"batches"*), kas katra 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".

6.3.3. Tālāka darbība – GAN

Kaut gan šī darba pētniecisko daļu galvenokārt sastādījusi datu ieguves, priekšapstrādes, klasiskā un rekurentā modeļa apmācības un validācijas procesu izstrāde, balstoties uz teorētiskajā literatūrā gūtām atziņām par vienu no galvenajiem virzieniem tālākiem pētījumiem iezīmēta ģeneratīvā pretinieku tīkla realizācija kā paņēmiens, kas varētu apvienot vienkāršu un konstantā laikā izpildāmu stratēģijas modeli ar daudz spējīgāku apmācības metodi. Attiecīgi 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 sakarību ar to parametriem. Līdz ar to nepieciešams citādi strukturēt apmācības programmu.

Vispirms tiek instancēti divi modeļi izmantojot *keras functional API*, kas ļauj konstruēt skaitļošanas grafu ar secīgu slāņu reprezentācijas objektu izsaukšanu. Pirmais modelis – ģenerators π_θ, kas ir arī rezultējošā stratēģija – līdz šim veiktajos eksperimentos ir ticis strukturēts tāpat, kā klasiskais neironu tīkls. Otrs modelis – diskriminators d_ϕ – 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

$$\sigma_{LeakyReLU}(x) = \begin{cases} x, \text{ ja } x > 0 \\ -k, 0 < k \ll 1 \end{cases} \tag{6.38}$$

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 kompilēt 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

$$s_1 \to \left((s_1, s_2^\theta, ..., s_m^\theta), ..., (s_{n-m}^\theta, ..., s_n^\theta)\right) = \tau_{gen} \tag{6.39}$$

kur

$$s_n^\theta = \begin{cases} \pi_\theta(s_1), \text{ ja } n = 2 \\ \pi_\theta(s_{n-1}) \text{ citādi} \end{cases} \tag{6.40}$$

No treniņa datu kopas tiek sagatavots tādas pašas formas tenzors ar ierakstītām stāvokļu pārejām τ_{dem}. Diskriminators tiek izsaukts uz abiem tenzoriem ar klasikācijas uzdevumu

$$d_\phi : (s_1, s_2^\theta, ..., s_m^\theta) \to \mathbb{R} \tag{6.41}$$

un tam tiek aprēķināta mērķa funkcijas vērtība

$$\mathcal{L}_d = \sum_{pred \in \tau_{gen}} \mathcal{L}_H(pred, 0) + \sum_{obs \in \tau_{dem}} \mathcal{L}_H(obs, 1) \tag{6.42}$$

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

$$\mathcal{L}_\pi = \sum_{pred \in \tau_{gen}} \mathcal{L}_H(pred, 1) \tag{6.43}$$

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

6.4. Validācijas datu ģenerēšana, simulācija, vizualizācija

Populāriem un plaši pētītiem mašīnmācīšanās uzdevumiem – piemēram, attēlu klasifikācijai – pastāv standartizēti modeļa veiktspējas mēri un atliek tikai izvēlēties atbilstošo skaitlisko vērtību, kuras vertībai sekot, apmācot dažādus modeļus [48]. Taču risinot netipiskas vai pat iepriekš neredzētas problēmas ar netriviālu problēmas nostādni, 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 mērķī, 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, satvērējmehānisma 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.

6.4.1. Modeļu pārbaude ar gadījuma sākuma stāvokļiem

Pirmā metode, kas 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:

$$\mathbf{r}_{target} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}(\mathbf{r}_{target}), \hat{\boldsymbol{\sigma}}(\mathbf{r}_{target})) \tag{6.44}$$

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 modeļiem 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.

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 skaitļošanas 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 ieraks-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 šājā agrīnajā projekta fāzē, bijusi trajektorijas x, z koordināšu 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 – klātbūtne.

6.4.2. Validācija – demonstrāciju un autoregresoru salīdzinājumi

6.4.3. Trajektoriju telpiska vizualizācija

6.4.4. Simulācijas vide

6.5. Trajektoriju izpilde uz robota

6.5.1. Ieraksti un modeļi

6.5.2. Robota trajektorijas plānošana un izpilde

6.5.3. Satvērējmehānisma vadība

7. REZULTĀTU ANALĪZE

7.1. Novērtējuma metriku izvēle un pamatojums

7.2. Sasniegtie rezultāti dažādām modeļu klasēm

7.3. Parastie neironu tīkli

7.4. Rekurentie neironu tīkli

7.5. Trajektoriju vizualizācija un kvalitatīvie novērtējumi

8. SECINĀJUMI

Literatūras analīzē sniegts īss par atdarinošo mašīnmācīšanos līdz šim veiktās pētnieciskās darbības pārskats. Jau izvēloties, par kurām tēmām vērts rakstīt plašāk, iespaidu uz darba saturu ir atstājusi motivējošās problēmas specifika. Tagad nepieciešams pie tās atgriezties un novērtēt, kas no visa nozarē pētītā un izgudrotā attiecas uz darba ievadā aprakstīto uzdevumu — mešanas kustību iestrādāšanu atkritumu vai citu objektu pārvietošanā — un izstrādāt rīcības plānu tālākai maģistra darba izstrādei.

8.1. Svarīgākās atziņas

Tūlīt kļūst skaidrs, ka pašas vienkāršākās metodes — klasiskā uzvedības klonēšana [33] vai kādas tās tiešās korekcijas — uzdevumam piemērotas nav. Tās darbojas gadījumos, kad demonstrāciju kopa labi nosedz visas iespējamās trajektorijas sistēmas konfigurāciju telpā, nav robustas pret nobīdēm, un jaunu trajektoriju programmēšana prasa apmācības procesa atkārtošanu. Problēmu grūti nostādīt formā, kur robotam vienkārši jāapgūst maksimāli precīzi atdarināt detalizētu un pilnībā aprakstošu demonstrāciju kopu. Vēl jāatzīst, ka metodes, kas ir ļoti specifiskas tieši robotikas uzdevumiem un prasa no algoritma izstrādātājiem pieņēmumus par uzdevuma struktūru — t.i, pamatā risinājumi, kas nodarbojas ar simbolisko dekompozīciju [19, 38, 52, 30, 31] — nešķiet sevišķi perspektīvas, jo tās ir grūtāk attīstīt un vispārināt. Protams, no inženiertehniskā viedokļa tās varētu zināmos uzdevumos pārspēt vispārīgākas pieejas, tomēr ilgtermiņa pieredze rāda, ka dažādas lokālas priekšrocības mēdz izgaist, pieaugot daudzparametru neironu tīklu veiktspējai.

Runājot par novērojumiem, nozīmīgi šķiet tieši pētījumi, kas realizē demonstrāciju ievākšanu no cilvēka kustībām [20, 53] vai nu virtuālā, vai fiziskā telpā. Metienu izdarīšana ir intuitīvi labi trenēta taču analītiski sarežģīta procedūra. Ātra piemēru iegūšana varētu kalpot par pamatu rīkiem, ar ko praktisku sistēmu pielāgot tās apkārtējai videi darba zonā. Tāpat uzmanību vērts pievērst pētījumiem, kas izmanto telpiskus un video novērojumus kā arī datu kopas sintētisku pavairošanu [10]. Lai gan tiešā veidā kādu no tajos aprakstītajiem algoritmiem droši vien neizdosies izmantot, ir vērts paturēt prātā pētījumus, kur trajektorijas padarītas noturīgākas pret novirzēm — vai nu trenējot slēptu dinamikas modeli [49], vai arī izmantojot analītiskas kompensācijas [30]. Pastāv iespēja, ka būs nepieciešams izmantot novērojumus, kas iegūti no dažādām perspektīvām, tāpēc noderīgi varētu izrādīties arī perspektīvu pārneses paņēmieni [23].

Visbeidzot, no adaptīvo un imitējošo metožu kombinācijas virziena ļoti svarīgi šķiet rezultāti, kas gūti izstrādājot stratēģijas, kuras spēj pielāgoties dažādu demonstrāciju atdarināšanai [9] vai dažādu galamērķu sasniegšanai bez papildus apmācības [25, 24, 15]. Tā kā ir iespējams diezgan skaidri definēt atalgojuma funkciju objekta iekrišanai pareizajā tvertnē, bet nonākšana līdz tai ar parastu stimulētās mācīšanās procesu ir ļoti apgrūtināta, risinājuma izstrādes gaitā var nākties izmantot arī atdarināšanu kā sākumpunktu pašmācībai [16].

8.2. Plāns tālākai darbībai

Apkopojot visu augstāk minēto, var ieskicēt iespējamu risinājumu uzdevumam, kas arī kalpos par pamatu tālākās rīcības plānam.

8.2.1. Risinājums

Lai apmācītu modeli, kas realizē metiena kustību ar jau satvertu neregulāras formas objektu, tiks darīts sekojošais:

1) demonstrāciju ievākšana — tiks ierakstīti cilvēka veikti metieni fiziskā telpā vai VR, iegūtas manipulatoru trajektorijas vai metienus raksturojoši parametri;
2) datu kopas papildināšana — iegūto demonstrāciju kopu varētu sintētiski pavairot, ja tiek veikta mesto objektu trajektoriju simulācija;
3) atdarinoša modeļa apmācība — jāizmanto modelis, kas parametrizēts pēc sasniedzamā galamērķa, lai būtu iespējams ātri un lēti pielāgot robotu darbam ar dažādi izvietotām tvertnēm;
4) papildus optimizācija — arī ar nelielas un neoptimālas demonstrāciju kopas palīdzību apmācītu modeli var ievērojami uzlabot, papildus izmantojot stimu-lētās mācīšanās metodes. Atalgojuma funkciju šoreiz ir samērā viegli definēt (bet grūti sasniegt).

Šobrīd grūti paredzēt, kādi rezultāti tiks sasniegti eksperimentālajā darbībā, kuri pieņēmumi būs izrādījušies patiesi, kuri — aplami. Tāpat vēl pāragri spriest par konkrētiem tehniskiem izpildījumiem, jo tie ir ļoti atkarīgi no izmantotās datu kopas atribūtiem.

8.2.2. Maģistra darba izstrādes plāns

Vadoties pēc augstāk aprakstītās risinājuma formas, šobrīd jau iespējams plānot nepieciešamo noslēguma darba struktūru — nodaļu mērķus un to saturu. Plānu dokumenta no zinātniska viedokļa saturiski nozīmīgajām daļām var izteikt sekojoši:

1) nozares teorētiskais pārskats — visu nepieciešamo priekšzināšanu izklāsts, visticamāk dalīts trijās nodaļās vai apakšnodaļās:
   (a) vispārīgas pamatzināšanas — analoģiski šī kursa darba 1.3. apakšnodaļai;
   (b) literatūras analīze — analoģiski šī kursa darba 2. nodaļai;
   (c) konkrētā izvēlētā risinājuma teorētiskais pamatojums — izstrādāto risinājumu matemātisko aspektu detalizēts izklāsts, nepieciešamie pierādījumi;
2) praktiskās darbības apraksts:
   (a) izmantotie rīki un metodes — izvēlēto programmatūras pakotņu, fiziskā aprīkojuma apraksti, izvēles pamatojumi;
   (b) risinājuma apraksts — veiktās darbības, izveidotie programmatūras artefakti, izmantotās un iegūtās datu kopas;
3) rezultātu analīze — veiktspējas novērtējumi, salīdzinājumi ar citiem pētījumiem, ieteicamie uzlabojumi un virzieni tālākai izpētei.

Teorētisko pārskatu var uzskatīt par daļēji pabeigtu. Literatūras analīzes nodaļa, visticamāk, jau satur lielāko daļu no informācijas, kas būs nepieciešama noslēguma darbā, tāpat ar vispārīgo priekšzināšanu apakšnodaļu. Protams, korekcijas un papildinājumi

gandrīz noteikti būs nepieciešami abiem, jo neizbēgami atklāsies faktori, par kuriem šobrīd vēl nav padomāts, un, iespējams, nāksies ņemt vērā arī jaunākas publikācijas nozarē. Detalizēta risinājuma izvēle un teorētiskais pamatojums šobrīd vēl nav iespējami, jo tie cieši saistīti ar praktiskās darbības norisi.

Praktiskās darbības ietvaros var jau paredzēt aptuvenu nepieciešamo soļu secību. No demonstrāciju ievākšanas viedokļa būs nepieciešams izvērtēt EDI pieejamā telpisko kustību ierakstīšanas aprīkojuma spējas un trūkumus, apgūt VR sistēmu izmantošanu un izvēlēties labāko no šīm pieejām instruktāžas procesa realizācijai. Nāksies atrast noderīgāko metiena reprezentāciju — metienu trajektorijas iespējams pilnībā raksturot ar ātruma un leņķiskā ātruma vektoriem to sākumā. Vai labāk tiešā veidā apmācīt robotu ar pēc galamērķa parametrizētām metienu veicošā rīka kustībām? Vai tomēr izmantot šos metiena raksturotājus kā starpmērķus — uztverot pareizu metienu un trāpīšanu mērķī par diviem atsevišķiem uzdevumiem, kam jātrenē dažādas stratēģijas? Tāpat būs arī jānovērtē datu kopas pietiekamība apmācības uzdevumam — izstrādājot metodi tās sintētiskai papildināšanai, ja nepieciešams.

Kad izvēlēta metiena reprezentācija un precizēta uzdevuma matemātiskā forma, būs iespējams sākt darbu pie modeļa izstrādes. Visticamāk, nāksies pārbaudīt vairākas, potenciāli ļoti atšķirīgas metodes. Beigās vēlams iegūt modeli, kas ar minimālu papildus treniņa iterāciju skaitu spēj pielāgoties metienu izdarīšanai uz dažādiem punktiem telpā — vai nu precizētiem koordinātu formā, vai arī izmantojot robotam pieejamo ievades informāciju (piemēram, datorredzi). Svarīgi arī izstrādāt testa un novērtējumu protokolu, kas ļauj objektīvi salīdzināt dažādu izmantoto modeļu atribūtus — metienu precizitāti, modeļa izmērus vai sarežģītību, pirmreizējā treniņa resursietilpību, trajektoriju pielāgošanas vai ieprogrammēšanas sarežģītību u.c. Tikai tā iespējams no liela iteratīvas un ne visai strukturētas praktiskās darbības radītu produktu korpusa izdarīt zinātniski nozīmīgus secinājumus.

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] 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).

[3] Isaac Asimov. *I, robot*. Vol. 1. Spectra, 2004.

[4] Alexandre Attia and Sharone Dayan. "Global overview of imitation learning". In: *arXiv preprint arXiv:1801.06503* (2018).

[5] Aude Billard et al. "Handbook of robotics chapter 59: Robot programming by demonstration". In: *Handbook of Robotics. Springer* (2008).

[6] Liana Blumberga. *ĢENERATĪVO PRETINIEKU TĪKLU TEORĒTISKA IZPĒTE (kursa darba konferences stenda plakāts)*. 2019. URL: `https://www.df.lu.lv/fileadmin/user_upload/LU.LV/Apaksvietnes/Fakultates/www.df.lu.lv/Studijas/Magistrantura/Konferences_stenda_referati/2019/Blumberga_Liana.pdf` (visited on 01/20/2022).

[7] 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.

[8] Steve Crowe. *10 Biggest Challenges in Robotics*. 2018. URL: `https://www.therobotreport.com/10-biggest-challenges-in-robotics/` (visited on 01/20/2022).

[9] Yan Duan et al. "One-shot imitation learning". In: *arXiv preprint arXiv:1703.07326* (2017).

[10] 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.

[11] 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.

[12] 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).

[13] Kunihiko Fukushima. "Neocognitron: A hierarchical neural network capable of visual pattern recognition". In: *Neural networks* 1.2 (1988), pp. 119–130.

[14] ABB Group et al. "Special report: Robotics–ABB group". In: *ABB Review* (2016).

[15] Abhishek Gupta et al. “Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning”. In: *arXiv preprint arXiv:1910.11956* (2019).

[16] Todd Hester et al. “Deep q-learning from demonstrations”. In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[17] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in neural information processing systems* 29 (2016), pp. 4565–4573.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Corey Lynch and Pierre Sermanet. “Language conditioned imitation learning over unstructured data”. In: *Proceedings of Robotics: Science and Systems. doi* 10 (2021).

[25] Corey Lynch et al. “Learning latent plans from play”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 1113–1132.

[26] 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.

[27] 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.

[28] 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.

[29] 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).

[30] 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.

[31] 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.

[32] Dario Pavllo, David Grangier, and Michael Auli. “Quaternet: A quaternion-based recurrent model for human motion”. In: *arXiv preprint arXiv:1805.06485* (2018).

[33] Dean A Pomerleau. *Alvinn: An autonomous land vehicle in a neural network*. Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE and PSYCHOLOGY ..., 1989.

[34] LZA TK ITTEA protokoli. *reinforcement learning*. 2017. URL: https://termini.gov.lv/kolekcijas/97/skirklis/454193 (visited on 01/20/2022).

[35] Pēteris Račinskis. *Masters – github repository*. 2022. URL: https://github.com/peteris-racinskis/masters (visited on 05/11/2022).

[36] Pēteris Račinskis. *Masters Data – github repository*. 2022. URL: https://github.com/peteris-racinskis/masters_data (visited on 05/11/2022).

[37] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. “No-regret reductions for imitation learning and structured prediction”. In: *In AISTATS*. Citeseer. 2011.

[38] 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.

[39] 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.

[40] 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.

[41] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[42] 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).

[43] Richard S Sutton and Andrew G Barto. "Reinforcement learning: An introduction". In: MIT press, 2018, pp. 60–77.

[44] *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 (visited on 05/12/2022).

[45] *Tensorflow Core v2.8.0 API documentation – Huber loss*. Google. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber (visited on 05/12/2022).

[46] *Tensorflow Core v2.8.0 API documentation – Ragged Tensors*. Google. 2022. URL: https://www.tensorflow.org/guide/ragged_tensor (visited on 05/12/2022).

[47] *Tensorflow Core v2.8.0 API documentation – ReLU activation function*. Google. 2022. URL: https://www.tensorflow.org/api_docs/python/tf/nn/relu (visited on 05/12/2022).

[48] *Tensorflow Tutorials – image classification*. Google. 2022. URL: https://www.tensorflow.org/tutorials/keras/classification?hl=en#compile_the_model (visited on 05/12/2022).

[49] Faraz Torabi, Garrett Warnell, and Peter Stone. "Behavioral cloning from observation". In: *arXiv preprint arXiv:1805.01954* (2018).

[50] Faraz Torabi, Garrett Warnell, and Peter Stone. "Generative adversarial imitation from observation". In: *arXiv preprint arXiv:1807.06158* (2018).

[51] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[52] 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.

[53] 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.