Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Neurálny model integrácie jazykových a motorických znalostí simulovaného agenta

2011 Bc. Tomáš Malík

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Neurálny model integrácie jazykových a motorických znalostí simulovaného agenta

Diplomová práca

Študijný program: Informatika
Študijný odbor: 9.2.1 informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Vedúci práce: doc. Ing. Igor Farkaš, PhD.
Evidenčné číslo: 407cd3e9-869e-432e-bbdd-d860d7ed140d

Bratislava, 2011

Bc. Tomáš Malík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Malík
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Neurálny model integrácie jazykových a motorických znalostí simulovaného agenta.

Cieľ: 1. Naštudujte relevanté články z oblasti kognitívnej robotiky a učenia sa prirodzenému jazyku pomocou neurónových sietí. 2. Implementujte integrovaný model neurónovej siete, pozostávajúcej z motorického a z jazykového modulu. 3. Otestujte učenie sa a správanie siete v úlohách porozumenia a generovania akcií v jednoduchom simulovanom prostredí.

Vedúci: doc. Ing. Igor Farkaš, PhD.

Dátum zadania: 12.11.2009

Dátum schválenia: 18.02.2011

prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

Tomáš Malík
študent

vedúci

Ďakujem môjmu vedúcemu práce doc. Ing. Igorovi Farkašovi, PhD.
za študijné materiály, konzultácie a ostatnú odbornú pomoc.

Abstrakt

Jednou z dlho skúmaných a stále otvorených otázok výpočtovej kognitívnej vedy je mechanizmus, ktorý ľuďom umožňuje priraďovať k slovám ich význam. Kognitívna robotika sa snaží pomôcť zodpovedať túto otázku študovaním rôznych hypotéz a fenoménov pomocou reálnych či simulovaných robotických agentov. Hlavným cieľom práce je návrh, implementácia a otestovanie nového systému na báze neurónových sietí v úlohe vykonávania akcií, na základe ich jazykového opisu a taktiež pomenovania vykonanej akcie. V teoretickej časti uvádzame prehľad súvisiacich prác a použitých metód. Náš prezentovaný model sa vyznačuje použitím biologicky vierohodného učenia sa senzomotorických akcií pomocou posilňovania. Virtuálnu scénu sme vytvorili v realistickom simulátore precízne navrhnutého humanoidného robota iCub. Navrhnutý model sme úspešne implementovali a natrénovali na vykonávanie jazykom opísaných akcií s objektami na scéne prostredníctvom iCub-a. Model bol zároveň schopný vykonávanú akciu správne pomenovať. Vyhodnotili sme aj generalizačnú schopnosť systému a považujeme ju za veľmi dobrú.

Kľúčové slová: akvizícia významov, vykonávanie akcií, robot iCub.

Abstract

Despite many years of studies and research, the complete description of mechanisms that allow humans to associate meanings to words is still missing. Cognitive robotics tries to answer this question by studying different hypotheses through their realization in both real and simulated agents. The main goal of the thesis is to propose, implement and test new neural network based system in the task of performing and naming linguistically described actions. The theoretical part includes an overview of related work and description of methods we used. Our novel model is based on biologically plausible reinforcement learning of actions. The Virtual scene was created in the highly realistic simulator of humanoid robot iCub. We successfully implemented and trained the proposed model to execute actions related to objects on the table in front of the iCub robot and to name the action it was currently executing. The results confirmed very good generalization ability of presented neural system.

Keywords: embodied language, performing and understanding actions, robot iCub.

Obsah

Zoznam obrázkov

Úvod

Ľudská reč je unikátnym dorozumievacím prostriedkom, ktorý nám umožňuje vymieňať si navzájom svoje myšlienky. Napriek doterajšej snahe nebol objasnený mechanizmus, ktorý ľuďom zabezpečuje schopnosť priradiť k slovám ich význam. Pri vzájomnej komunikácii o rôznych dynamických procesoch, vlastnostiach objektov a ich vzťahoch, napríklad pri opise akcie ktorá sa má vykonať, používame slová, ktoré sú určitým spôsobom prepojené s prvkami na scéne, ako aj akciami, ktoré s nimi môžu byť vykonávané alebo na nich pozorované. Význam týchto slov je nadobúdaný už u malých detí počas interakcie s prostredím a to ešte pred tým, ako sa začínajú učiť slová. Vtelená kognícia tvrdí, že pochopenie významu pozorovanej alebo slovne opísanej akcie je založené na senzomotorickej simulácii.

V doterajších prácach sa výpočtové modelovanie ukotvenej akvizície jednoduchého jazyka zameriava na ukotvenie podstatných mien na senzomotorickú reprezentáciu objektov a slovies na vykonávané akcie (Sugita a Tani, 2005). Model reálneho, mobilného robota sa učil vykonávať rôzne akcie s objektami, v prepojení s ich slovným opisom. Akvizíciu významov mien akcií, pomocou simulátora humanoidného robota iCub (Tikhanoff a kol., 2008), prezentuje Marocco a kol. (2010). Ukotvenie významu prebiehalo fyzickou interakciou s prostredím a previazaním vplyvu vlastných akcií robota s reakciou pozorovanou pred a po vykonaní akcie. Dôraz bol kladený na previazanie pomenovania akcií s vlastnosťami objektov. V inom experimente Cangelosi a kol. (2007) pomocou simulovaných robotov skúmali akvizíciu významu formou imitácie akcií. Ukotvenie slovies bolo vo výsledku zjavne späté s motorickým vzorom akcie.

Spomínané práce spája použitie učenia s učiteľom na trénovanie neurálnych modelov. Hlavným cieľom našej práce je navrhnúť, implementovať a otestovať model systému na akvizíciu významov a generovanie jednoduchých akcií, v ktorom bude figurovať biologicky vierohodné učenie s posilňovaním, ktoré má analógiou s učením sa človeka. Navrhnutý model ovláda agenta vo virtuálnom simulovanom svete, kde sa učí vykonávať rôzne akcie s objektami na scéne, na základe jazykového príkazu a pochopiť tak jeho význam. Zároveň od agenta požadujeme, aby naučené a vykonávané akcie vedel slovne pomenovať.

V teoretickom úvode uvádzame podrobnejší opis výsledkov dosiahnutých vo

vyššie spomínaných prácach a taktiež prehľad metód použitých v našom modely. Hlavnú časť práce tvorí návrh modelu s detailným opisom jeho častí, spolu s výsledkami a vyhodnotením. Súčasťou práce je aj príručka na zoznámenie sa so simulátorom robota iCub, spolu so skúsenosťami a postrehmi, ktoré sme pri jeho použití nadobudli.

Kapitola 1

Teoretický úvod

1.1 Ciele práce

Cieľom práce je naštudovať a uviesť prehľad relevantných článkov z oblasti kognitívnej robotiky a učenia sa prirodzenému jazyku pomocou neurónových sietí. Vytvoriť jednoduché virtuálne prostredie s ramenom robota, ktoré je schopné vykonávať akcie s objektami na scéne. Na základe získaných vedomostí navrhnúť, implementovať a natrénovať model na báze neurónových sietí, ktorý sa bude učiť vykonávať pohyby smerom k objektom na scéne na základe jazykových inštrukcií. Model má mať aj opačnú schopnosť, teda vedieť pomenovať vykonávané pohyby. Súčasťou práce je aj analýza vlastností modelu, vrátane jeho generalizačnej schopnosti.

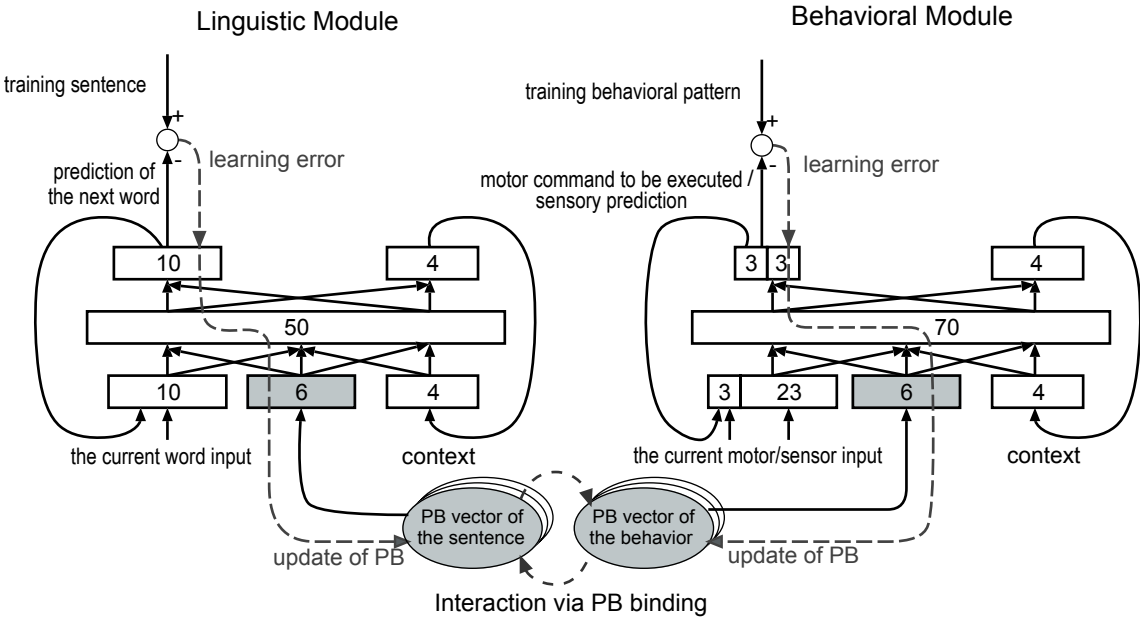
Ako virtuálne prostredie sme zvolili simulátor humanoidného robota iCub. Na scéne sú vedľa seba tri objekty rôzneho tvaru a farby. Vzájomné rozmiestnenie objektov (guľa, kocka, valec) je ľubovoľné, rovnako ako zafarbenie tromi rôznymi farbami (červená, zelená, modrá) bez opakovania. Neurálny model má ovládať rameno robota a vykonávať s objektami tri rôzne akcie (dotyk, posunutie, ukázanie). Agent dostáva na vstupe dvojslovný jazykový opis akcie, napríklad „point red“ alebo „touch cube“, kde cieľ akcie je definovaný tvarom alebo farbou objektu. Po vykonaní akcie má byť agent schopný na výstupe uviesť jej jazykové pomenovanie.

1.2 Prehľad súčasného stavu problematiky

V tejto kapitole uvádzame krátky prehľad relevantných článkov k téme našej diplomovej práce. Autori v nich prezentujú rôzne modely, zamerané na prepojenie vykonávania akcií a významu ich jazykového pomenovania.

1.2.1 Rekurentná neurónová sieť s parametrickým biasom

Nový konekcionistický model na akvizíciu významov sémantiky jednoduchého jazyka s využitím správania reálneho robota publikoval Sugita a Tani (2005). Model je postavený na rekurentnej sieti s parametrickým biasom (RNNPB) (Tani, 2003), ktorá je rozšírením klasickej Jordanovej rekurentnej neurónovej siete (RNN) (Jordan, 1990). Skladá sa z dvoch špecificky vzájomne prepojených sietí – lingvistického a behaviorálneho modulu. Na vstup je pridaný trénovaný PB vektor, ktorý dokáže modulovať správanie celej siete. Model zobrazený na obrázku 1.1 je schopný predikovať nasledujúce slovo alebo proprioreceptívny stav na základe aktuálneho stavu na vstupe. Rôzne hodnoty PB vektora podmieňujú rozličné výstupné postupnosti. Pri trénovaní viacerých výstupných postupností je ku každej z nich priradený rozdielny PB vektor, ktorý minimalizuje chybu predikcie. Dôležitou vlastnosťou je, že hodnoty PB vektorov sú pri trénovaní nastavené samoorganizáciou. Po natrénovaní siete je možné vyvolať naučené správanie privedením zodpovedajúcich hodnôt PB vektorov na vstup siete. Tieto hodnoty sa dajú po natrénovaní spätne dopočítať minimalizáciou predikčnej chyby na príslušnej postupnosti, čo môže byť považované za jej rozpoznanie.



Obr. 1.1: Architektúra dvojice RNNPB sietí. Lingvistický a behaviorálny modul sú vzájomne prepojené cez spoločný vstupný PB vektor. Obrázok je prebraný z článku Tani (2003).

Model ovláda pohyblivého robota na kolesách, ktorý je vybavený otočným ramenom a farebným obrazovým senzorom. Na scéne sa nachádzajú vedľa seba tri

objekty, pričom každej pozícii zodpovedá jedna pevne priradená farba, ktorou je objekt zafarbený. Robot sa učí s objektami vykonávať 9 behaviorálnych kategórií, konkrétne tri akcie (push, touch, point). Každá akcia môže byť vykonávaná do troch smerov, ktoré sú určené pozíciou alebo farbou objektu. Agent sa učí vykonávať akcie na základe ich slovného pomenovania.

Trénovanie prebieha za pomoci učiteľa, čiže požadované postupnosti sú vopred pripravené. Počas trénovania sú modifikované váhy prepojení, ale aj samotné PB vektory. Váhy prepojení sú pre všetky postupnosti spoločné narozdiel od PB vektora, ktorý je pre každú postupnosť odlišný. Trénujú sa súčasne pomocou algoritmu spätného šírenia chyba v čase (BPTT) (Rumelhart a kol., 1988) s cieľom minimalizovať celkovú chybovú funkciu E, cez všetky trénovacie postupnosti $q_0, \ldots, q_{s-1}$ definovanú nasledovne:

$$E(W, p_0, \ldots, q_{s-1}) = \sum_{k=0}^{s-1} E_k(W, p_k)\text{a} \tag{1.1}$$

$$E_k(W, p_k) = \sum_{t=0}^{l_k-1} \sum_{n \in N} (r_{kn}(t) - o_{kn}(W, p_k, t))^2, \tag{1.2}$$

kde E_k je chybová funkcia trénovacej postupnosti q_k s dĺžkou l_k. W je množina všetkých váh siete, p_k je PB vektor prislúchajúci trénovacej postupnosti q_k. N je množina výstupných neurónov, $r_{kn}(t)$ a $o_{kn}(W, p_k, t)$ sú cieľová a výstupná hodnota neurónu n v trénovacej postupnosti q_k v kroku t. Váhy prepojení sú upravované na minimalizovanie celkovej trénovacej chyby E. Úpravou PB vektora p_k sa zase minimalizuje trénovacia chyba E_k, pre každú postupnosť.

Pri trénovaní vety a prislúchajúceho behaviorálneho vzoru konvergujú PB vektory modulov k rôznym hodnotám. V trénovacom algoritme je preto zahrnutý mechanizmus, ktorý zabezpečuje, aby PB vektor behaviorálneho modulu a príslušný PB vektor lingvistického modulu konvergovali čo najbližšie k sebe.

V testovacej fáze pracujú moduly samostatne. Najskôr je behaviorálnym modulom rozpoznaná zadaná veta, teda je vypočítaný zodpovedajúci PB vektor. Ten je následne nastavený ako vstup pre behaviorálny modul, ktorý na základe neho vygeneruje príslušné správanie. Analogicky je možné postupovať v opačnom smere.

Model bol otestovaný v troch experimentoch. V prvom experimente bol použitý iba lingvistický modul, na zistenie jeho schopnosti osvojiť si syntax jazyka. Táto schopnosť sa potvrdila a model generoval gramaticky správne vety aj na testovacej množine. Pri skúmaní váh skrytej vrstvy sa ukázalo, že hodnoty váh každého skrytého neurónu sa dajú rozdeliť do dvoch skupín váh s rovnakou hodnotou, na váhy neurónov kódujúcich slovesá a váhy neurónov kódujúcich podstatné mená. Druhý experiment bol zameraný na trénovanie behaviorálneho modulu. Modul dokázal vy-

konávať požadované správanie aj v prípade, že cieľový objekt bol mierne posunutý. V priestore PB vektorov sa nevyskytovala žiadna zjavná štruktúra behaviorálnych kategórií.

Tretí experiment zahŕňal prepojenie oboch modulov. Model sa naučil správne vykonávať akcie po rozpoznaní naučenej vety. Behaviorálny modul bol trénovaný so všetkými možnými akciami, narozdiel od lingvistického modulu, ktorý mal niektoré vety vynechané. Napriek tomu bol robot schopný vykonávať aj akcie, ktorých vety neboli prítomné počas trénovania. Takisto bolo pozorované, že PB vektory nových viet korešpondovali s behaviorálnymi, hoci pri trénovaní neboli viazané k sebe.

1.2.2 Ukotvenie jazyka v senzomotorických akciách

Ukotvenie jazykových slov v senzomotorickej interakcii s prostredím skúmal aj Marocco a kol. (2010). V experimentoch používajú simulátor robota iCub (Tikhanoff a kol., 2008). Rameno robota je ovládané pomocou neurálneho systému trénovaného s učiteľom. Na stole pred robotom je položený jeden z troch objektov (guľa, kocka, valec). Objekty majú rôzne fyzikálne vlastnosti, či už tvar alebo odpor, ktorý kladú pri snahe o ich posunutie.

Rameno bolo ovládané len jedným kĺbom tak, aby pri pohybe došlo k dotyku, prípadne posunutiu objektu na stole. Natočenie hlavy pomocou dvoch kĺbov sledovalo pohyb objektu, ktorý bol centrovaný v zrakovom poli. Otáčanie hlavy mal na starosti samostatný modul robota. Zároveň bol z obrazu ľavej kamery vypočítaný parameter guľatosti objektu.

Neurálny systém je tvorený plne prepojenou rekurentnou sieťou s desiatimi skrytými neurónmi. Vstup reprezentuje stav agenta a skladá sa z ôsmich neurónov – 3 pre stav kĺbov, 1 pre dotyk, 1 parameter guľatosti objektu a 3 pre jazykový vstup. Identické zloženie neurónov je aj na výstupe siete, ktorá je trénovaná na predikciu nasledovného stavu na vstupe. Trénovanie prebiehalo pomocou algoritmu spätného šírenia chyby v čase (BPTT) (Rumelhart a kol., 1988). Pred trénovaním boli vygenerované trénovacie postupnosti. Rameno bolo ovládané dopredu naprogramovaným kontrolérom. Súčasne bolo aktívne aj sledovanie objektu. Zároveň boli zaznamenávané natočenia kĺbov, stav dotykového senzora a vypočítaný koeficient guľatosti objektu. Vstupné vzory boli nakoniec rozšírené o jazykovú zložku, ktorá opisovala typ objektu (gúľajúci sa, posúvajúci sa, nepohyblivý). V testovacej fáze je výstup siete použitý na nastavovanie aktuátorov robota. Takisto proprioreceptívna informácia na vstupe modelu je získaná z aktuálneho stavu robota.

Výsledky potvrdili, že robot si dokáže osvojiť a reprodukovať vykonávanie akcií s objektami na scéne. Ukázalo sa, že model je schopný priradiť konkrétnu senzomotorickú postupnosť k naučeným jazykovým označeniam aj v prípade, že jazyková časť vstupu absentuje. Testy potvrdili, že robot vie správne kategorizovať vlastnosti

objektu bez prítomnosti jazykového vstupu a rovnako aj bez prítomnosti parametra guľatosti. Ukotvenie významu bolo teda zahrnuté v celej senzomotorickej dynamike.

1.2.3 Ukotvenie symbolov pomocou imitácie

Ukotvenie jazyka do reprezentácie akcií skúmal aj Cangelosi a kol. (2007). V experimente boli použité dva simulované roboty, každý s dvoma trojčlánkovými ramenami pripevnenými k telu. Pomocou ramien môže robot interagovať s objektami položenými pred ním. Agent sa má naučiť šesť jednoduchých základných akcií spolu s ich menami. Každá akcia je asociovaná s jedným z objektov. Prvý agent je v úlohe učiteľa a je naprogramovaný na vykonávanie a demonštrovanie akcií na základe lingvistickej inštrukcie. Akcie demonštruje druhému agentovi (imitátor), ktorý sa ich snaží opakovať. Imitátor sa najskôr učí vykonávať základné akcie a potom ich aj pomenovať. Následne sa snaží samostatne aplikovať už ukotvené symboly do nových zložených akcií.

Imitátor je ovládaný viacvrstvovou neurónovou sieťou. Na vstupe má proprioreceptívnu, jazykovú a obrazovú informáciu. Na výstupe má jazykové neuróny a neuróny na ovládanie kĺbov. Trénovanie prebiehalo v troch krokoch. V prvej fáze sa agent učí vykonávať šesť základných akcií asociovaných s obrazom rôznych objektov. Na úpravu váh bol použitý imitačný algoritmus (Cangelosi a kol., 2006) využívajúci učenie s učiteľom. V druhej fáze sa imitátor učí k už natrénovaným akciám priradiť pomenovanie, pomocou štandardného algoritmu spätného šírenia chyby (Rumelhart a kol., 1988). Vo finálnej fáze nadobúda imitátor zložené akcie bez potreby demonštrácie učiteľom, ten poskytne len jazykové pomenovanie akcie vyššej úrovne.

Po ukončení trénovania dokázal imitátor bezchybne vykonávať všetkých 6 základných akcií a tri zložené akcie. Pri testovaní prenosu ukotvených významov agent úspešne vykonával akcie pomenované len menom zloženej akcie, ale tiež súčasne v kombinácii so základnými menami akcií. To potvrdzuje, že najskôr ukotvené symboly boli prenesené aj do nového správania.

Kapitola 2

Použité metódy

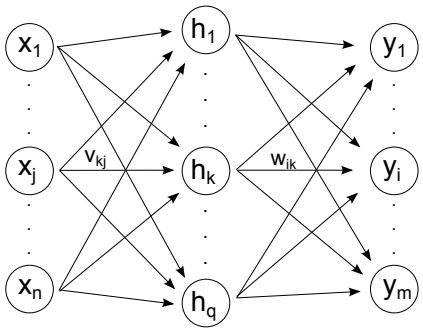
Neurálny model systému, ktorý predstavujeme v kapitole 4, je zložený z viacerých typov umelých neurónových sietí. V tejto kapitole uvádzame ich teoretický opis. Na viacerých miestach sme použili známy model viacvrstvového perceptrónu. Časť modelu, ktorá je zodpovedná za generovanie motorických pohybov, je postavená na algoritme CACLA (Continuous Actor-Critic Learning Automaton). V lingvistickom module, ktorý má za úlohu vygenerovanie slovného pomenovania vykonanej akcie, sme použili sieť s echo stavmi. Zdôvodnenie a motiváciu použitých metód uvádzame v kapitole 4.

2.1 Viacvrstvová neurónová sieť

Jednoduché neurónové siete dokážu riešiť len lineárne separovateľné problémy, čo bolo považované za ich podstatný nedostatok. Viacvrstvové neurónové siete sú výsledkom snahy o navrhnutie univerzálnejšieho modelu, ktorý bude schopný riešiť aj zložitejšie úlohy. Zmysel úvah o pridaní skrytej vrstvy potvrdil Rumelhart a kol. (1988), keď prišli s návrhom trénovacieho algoritmu spätného šírenia chyby (angl. error back propagation). Neskôr bolo dokázané, že takáto neurónová sieť s jednou skrytou vrstvou je schopná aproximovať ľubovoľnú spojitú funkciu. Náčrt viacvrstvovej siete je na obrázku 2.1

Aktivácie neurónov na jednotlivých vrstvách sú dané vzťahmi:

- $h_k = f(\sum_{j=1}^{n+1} v_{kj} x_j)$ - pre neuróny na skrytej vrstvách,
- $y_i = f(\sum_{k=1}^{q+1} w_{ik} h_k)$ - pre neuróny na výstupnej vrstve,
- $x_{n+1} = h_{n+1} = -1$ - pridaný bias vstup,
- $f(net)$ - aktivačná funkcia (sigmoida, hyperbolický tangens).



Obr. 2.1: Topológia viacvrstvovej neurónovej siete.

V krátkosti zhrnieme trénovací algoritmus spätného šírenia chyby. Pre zadaný vstup už vieme vypočítať výstup siete. Ak sa rovná požadovanému, nenastáva žiadne trénovanie. Pokiaľ je ale na výstupe chyba (rozdiel medzi požadovanou a skutočnou odozvou siete) táto sa postupne šíri od výstupnej vrstvy smerom ku vstupnej vrstve. Chyba konkrétneho neurónu prispieva k chybe nerónov na predošlej vrstve čiastkou, ktorej veľkosť závisí od sily príslušného prepojenia. Konkrétne je rovná súčinu šírenej chyby a váhy prepojenia.

Úprava váh výstupnej vrstvy:

$$w_{ik}(t+1) = w_{ik}(t) + \alpha\delta_i h_k, \text{ kde } \delta_i = (d_i - y_i)f_i' \tag{2.1}$$

a váh skrytej vrstvy:

$$w_{kj}(t+1) = w_{kj}(t) + \alpha\delta_k x_j, \text{ kde } \delta_k = (\sum_i w_{ik}\delta_i)f_k' \tag{2.2}$$

Parameter α určuje rýchlosť učenia. Na hodnoty δ_i sa môžeme pozerať, ako na chybu i-teho neurónu výstupnej vrstvy. A na hodnoty δ_k, ako na chybu k-teho neurónu skrytej vrstvy, pričom vidieť, že táto sa počíta ako suma chýb neurónov výstupnej vrstvy (resp. nasledujúcej skrytej vrstvy smerom k výstupu) prenásobených váhovým koeficientom príslušného prepojenia.

Sieť trénujeme po každom vstupe. Počas jednej epochy prejdeme v náhodnom poradí cez všetky vstupy trénovacej množiny. Pre aktuálne spracovávaný vstup vypočítame výstup siete. Spočítame všetky hodnoty δ_i, δ_k. Tie využijeme na úpravu váh w_{ik}, v_{kj} podľa vzťahov 2.1 a 2.2. Počet trénovacích epoch zavisí od zvoleného zastavovacieho kritéria.

2.2 Učenie s posilňovaním

Učenie s posilňovaním zastáva v strojovom učení miesto niekde na rozhraní učenia s učiteľom a učenia bez učiteľa. Pri učení s učiteľom dostáva agent z prostredia presne určené dvojice, pre každý vstup požadovaný výstup. Agent sa musí naučiť len korektné mapovanie sady vstupov na zodpovedajúce výstupy spolu s prijateľnou mierou generalizácie. Pri učení bez učiteľa agent nedostáva žiadnu informáciu o požadovaných výstupoch a vytvára si vlastnú reprezentáciu organizácie vstupných dát (samoorganizácia).

Pri trénovaní s posilňovaním má agent k dispozícii kvalitatívne ohodnotenie stavu v ktorom sa nachádza, poprípade ohodnotenie kvality vykonanej akcie v konkrétnom stave. Toto ohodnotenie je skalárna hodnota, ktorú nazývame odmena. Vyjadruje mieru správnosti vykonanej postupnosti akcií. Čím numericky vyššia hodnota odmeny, tým je pozorované správanie žiadúcejšie.

Učenie s posilňovaním je biologicky prijateľný spôsob trénovania. Totiž správanie agenta nemusí byť dopredu naprogramované, ale samostatne sa vyvíja na základe svojej vlastnej skúsenosti v prostredí. Že je to tak podobne aj v prírode nasvedčujú psychologické výskumy, v ktorých bolo skúmané učenie sa zvierat a ľudí pomocou odmeňovania. Šimpanzy boli schopné naučiť sa zoradiť ľubovoľnú podmnožinu číslic 1 až 9 do správneho poradia, len na základe odmeňovania arašidmi (Inoue a Matsuzawa, 2007).

2.2.1 Učenie s posilňovaním ako MDP

Problém učenia s posilňovaním je definovaný ako Markovský rozhodovací proces MDP (Markov Decision Process) (Sutton a Barto, 1998). Formálne sa naň môžeme pozerať ako na štvoricu (S, A, R, T).

- $S = \{S_0, S_1, S_2, \ldots, S_N\}$ je konečná množina stavov prostredia, kde stav v čase t budeme označovať s_t.
- $A = \{A_0, A_1, A_2, \ldots, A_N\}$ je konečná množina akcií, kde akciu vykonanú v čase t budeme označovať a_t.
- $R : S \times A \times S \to \mathbb{R}$ je funkcia odmeny. Odmenu $R(s_t, a_t, s_{t+1})$, získanú po prechode do stavu s_{t+1} vykonaním akcie a_t v stave s_t, budeme označovať r_{t+1}.
- $T : S \times A \times S \to \langle 0, 1\rangle$ je prechodová funkcia. $T(s, a, s')$ je pravdepodobnosť prechodu zo stavu s do stavu s' vykonaním akcie a.

Hovoríme, že prostredie má Markovovu vlastnosť, ak stav do ktorého sa agent dostane v čase $t + 1$ závisí len od vykonanej akcie a_t a stavu s_t. Čo sa dá vyjadriť vzťahom

$$\begin{aligned}&\Pr\left\{s_{t+1}=s, r_{t+1}=r|s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\right\} = \\ &\Pr\left\{s_{t+1}=s, r_{t+1}=r|s_t, a_t\right\}.\end{aligned} \tag{2.3}$$

Ak táto podmienka nie je splnená a novo nadobudnutý stav závisí od histórie predošlých stavov a akcií, znamená to, že stav nie je jednoznačne určený. Prostredie je tak pre agenta len čiastočne pozorovateľné.

Agent sa v prostredí správa na základe doposiaľ naučenej stratégie $\pi : S \times A \rightarrow \langle 0, 1\rangle$. Postupne svoju stratégiu vylepšuje s cieľom maximalizovať kumulatívnu odmenu

$$r_t + \gamma r_{t+1} + \gamma_2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}, \tag{2.4}$$

kde $\gamma \in \langle 0, 1\rangle$ je diskontný faktor, ktorý určuje, do akej miery sa započíta odmena budúcich stavov.

Učenie agenta sa dá realizovať ukladaním hodnôt pre stavy $V(s)$ alebo pre dvojice stav-akcia $Q(s, a)$. Hodnoty $V(s)$ reprezentujú horeuvedenú kumulatívnu diskontovanú odmenu, ktorú agent očakáva po dosiahnutí stavu s. Ohodnotenia $Q(s, a)$ zodpovedajú očakávanej kumulatívnej odmene po vykonaní akcie a v stave s.

2.2.2 Učenie ohodnotenia stavov

Stavová ohodnocovacia funkcia môže byť aktualizovaná pomocou učenia s časovým oneskorením (temporal difference):

$$V_{t+1}(s_t) = V_t(s_t) + \alpha_t \delta_t, \tag{2.5}$$

kde $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$ je zmena hodnoty stavu po vykonaní akcie a $0 \leq \alpha_t \leq 1$ je rýchlosť učenia. Použitie tohto pravidla vedie ku konvergencii ohodnotenia stavov do ich skutočných hodnôt, pre pevne zvolenú stratégiu. Dôležité je, že r_{t+1} vyjadruje odmenu získanú bezprostredne po vykonaní akcie a_t v stave s_t, čiže po prechode do stavu s_t. Napríklad (van Hasselt a Wiering, 2007) používa na tomto mieste označenie r_t s posunutým indexom časového kroku.

2.2.3 CACLA

Algoritmy učenia s posilňovaním boli navrhnuté na riešenie problémov s konečnou množinou stavov a akcií. Mnohé úlohy si ale vyžadujú použitie spojitého priestoru stavov a akcií. Napríklad v našej práci je to vykonávanie akcií pomocou robotického ramena. Takýmto zovšeobecnením na spojité priestory je algoritmus CACLA (van Hasselt a Wiering, 2007).

Namiesto tabuľky ohodnotení stavov je možné použiť neurónovú sieť ako aproximátor funkcií. Úprava hodnôt sa realizuje trénovaním váh siete počas objavovania stavov. Ohodnotenie nových stavov zabezpečuje generalizačná schopnosť siete.

Podobne je to s priestorom akcií. Pri ovládaní robotického ramena máme na výber pohyb ľubovoľným smerom. Pre rôzne stavy potrebujeme okrem ich ohodnotenia získať aj akciu, tomuto ohodnoteniu zodpovedajúcu. V práci sme pre tento účel použili architektúru aktér-kritik 2.2.5.

2.2.4 Explorácia

Explorácia je mechanizmus, ktorý umožňuje agentovi objavovať nové akcie a zároveň tak vylepšovať svoju naučenú stratégiu. V literatúre sa najčastejšie spomínajú dve metódy explorácie:

- ϵ-greedy explorácia, pri ktorej agent s pravdepodobnosťou ϵ vyberie exploratívnu akciu. Tá môže byť zvolená ako ľubovoľná náhodná akcia. S pravdepodobnosťou $1-\epsilon$ potom ponechá agent navrhovanú akciu nezmenenú. Parameter ϵ určuje mieru explorácie. Jeho zmenou je možné zvoliť vhodný pomer vykonávania naučeného správania (exploatácia) a objavovania nového (explorácia), potenciálne výhodnejšieho konania.

- Gaussovská explorácia aktuálnej aproximácie optimálnej akcie. Pravdepodobnosť, že agent v stave s_t vyberie akciu a, je:

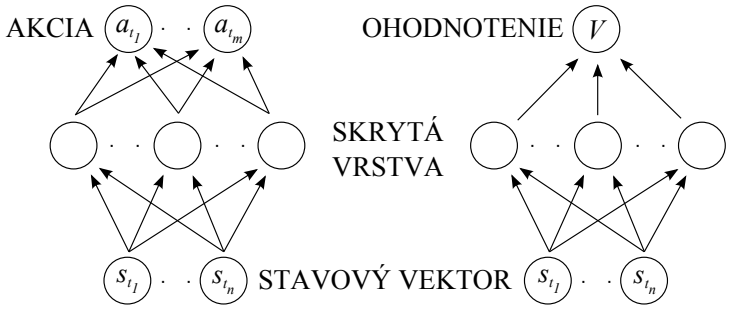
$$\pi_t(s_t, a) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(a-A_t(s_t))^2/(2\sigma^2)}, \tag{2.6}$$

  kde $A_t(s_t)$ označuje akciu navrhnutú agentom v čase t.

2.2.5 Architektúra aktér-kritik

Úlohou aktéra je naučiť sa na základe stavu agenta navrhnúť optimálnu akciu, ktorá sa má vykonať. Cieľom kritika je zase naučiť sa odhadovať ohodnotenie stavov agenta. Obe roly sa dajú jednoducho realizovať napríklad pomocou doprednej neurónovej siete. Ich vstupom je aktuálny stav agenta. Kritik je v každom kroku trénovaný na základe rovnice 2.5 tak, aby jeho výstup zodpovedal ohodnoteniu aktuálneho stavu agenta.

Trénovanie aktéra je trochu zložitejšie. Snažíme sa dosiahnuť, aby akcie ktoré navrhuje na svojom výstupe, boli blízke optimálnej stratégii. Podľa zvoleného pravidla explorácie vykonáme aktérom navrhovanú, respektíve exploračnú akciu. Po vykonaní akcie získa agent z prostredia odmenu, ktorá sa využije vo výpočte rovnice 2.5,



Obr. 2.2: CACLA - aktér a kritik pomocou neurónových sietí.

na trénovanie kritika. Zároveň ale hodnota δ_t napovedá, či bola akcia ktorú agent vykonal vhodná.

Kladná hodnota δ_t indikuje, že akcia viedla k vyššej kumulatívnej odmene a teda k lepšej stratégii. Preto v tomto prípade zvýšime pravdepodobnosť výberu aktérom vykonanej akcie pre prípad, že sa ocitne v budúcnosti v rovnakom stave. Výstupná chyba siete pre trénovanie aktéra bude rozdiel medzi navrhovanou a vykonanou akciou. Posilnením pravdepodobnosti výberu trénovanej akcie zároveň oslabíme pravdepodobnosť výberu ostatných akcií v tom konkrétnom stave.

Algorithm 1 Učenie s posilňovaním pomocou aktéra a kritika

$s_0 \leftarrow$ počiatočný stav
Inicializuj váhy aktéra
Inicializuj váhy kritika
for $t = 0, 1, 2 \ldots$ **do**
  $a_t \leftarrow A_t(s_t)$ s použitím explorácie
  vykonaj akciu a_t a prejdi do stavu s_{t+1}
  uprav váhy kritika: $V_{t+1}(s_t) \leftarrow r_{t+1} + \gamma V_t(s_{t+1})$
  if $V_{t+1}(s_t) > V_t(s_t)$ **then**
    uprav váhy aktéra: $A_{t+1}(s_t) \leftarrow a_t$
  end if
end for

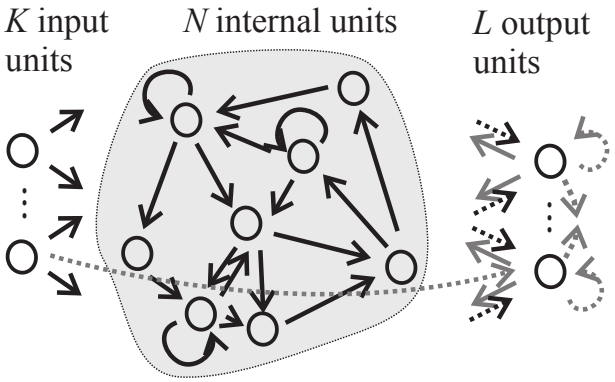
2.3 Sieť s echo stavmi

Štruktúra siete s echo stavmi (echo-state network, ESN) je podobná klasickým RNN. V oboch prípadoch sú neuróny organizované do troch vrstiev: vstupnej, vnútornej (skrytej, rekurentnej) a výstupnej. Základný rozdiel je v rekurentnej vrstve, ktorá sa v prípade ESN skladá z veľkého počtu neurónov. Tie sú navzájom poprepájané

spojeniami, ktoré sa netrénujú. Trénujú sa len prepojenia na výstupnej vrstve. Vnútorná vrstva neurónov je označovaná ako dynamický rezervoár (DR). Nevýhodou klasických RNN je nutnosť použitia komplikovaných a výpočtovo náročných trénovacích algoritmov. Keďže model ESN nemení prepojenia vrámci DR, tento môže obsahovať stovky neurónov. Odpadá nutnosť šírenia chybového signálu v čase a na trénovanie sa dá použiť akýkoľvek algoritmus lineárnej regresie. V prípade RNN sa snažíme dosiahnuť požadované správanie úpravou váh malého počtu neurónov a je preto dôležité optimálne zvoliť hodnoty váh všetkých prepojení. Naopak podstatou ESN je použitie náhodného DR s veľkým počtom neurónov, na zabezpečenie bohatej odozvy na vstupný signál. Odozva DR na vstupný signál má predstavovať jeho obraz vo vysoko rozmernom stavovom priestore rezervoára. Úlohou výstupnej vrstvy je vytvoriť výstup ako lineárnu kombináciu príznakov, ktoré zo vstupného signálu extrahovali vnútorné neuróny. V nasledujúcej časti uvedieme formálny opis ESN (Jaeger, 2001).

2.3.1 Štruktúra ESN

Diskrétny časový model ESN siete je odvodený z klasických rekurentných neurónových sietí. Je tvorený vstupnou a výstupnou vrstvou neurónov, ktoré sú prepojené s vnútornou vrstvu neurónov. Uvažujme teda K vstupných neurónov, N vnútorných neurónov a L výstupných neurónov.



Obr. 2.3: Všeobecná štruktúra ESN. Voliteľné prepojenia sú znázornené bodkovanými šípkami (Jaeger, 2001).

Aktivácie neurónov v časovom kroku n sú charakterizované:

- vstupným vektorom $\mathbf{u}(n) = (u_1(n), \ldots, u_K(n))^T$,
- aktivačným vektorom rezervoára $\mathbf{x}(n) = (x_1(n), \ldots, x_N(n))^T$,
- výstupným vektorom $\mathbf{y}(n) = (y_1(n), \ldots, y_L(n))^T$.

Hodnoty synaptických váh sú uložené v maticiach:

- $\mathbf{W}^{in} = (\mathbf{w}_{ij}^{in})$, veľkosti $N \times K$, pre vstupné váhy,
- $\mathbf{W} = (\mathbf{w}_{ij})$, veľkosti $N \times N$, pre vnútorné prepojenia,
- $\mathbf{W}^{out} = (\mathbf{w}_{ij}^{out})$, veľkosti $L \times (K + N + L)$, pre výstupné váhy,
- $\mathbf{W}^{back} = (\mathbf{w}_{ij}^{back})$, veľkosti $N \times L$, pre spätné prepojenia (angl. backprojection) od výstupných neurónov k vnútorným neurónom.

Môžu sa použiť aj prepojenia zo vstupných neurónov priamo na výstupné, ako aj prepojenia medzi výstupnými neurónmi navzájom. Pre vnútorné prepojenia sa predpokladá, že matica $\mathbf{W}$ indukuje cyklické prepojenia medzi neurónmi rezervoára. Konkrétne modely ESN sietí sa od seba navzájom odlišujú rôznymi modifikáciami všeobecnej štruktúry, s cieľom dosiahnuť optimálne správanie sa siete v danej úlohe. Typickým príkladom modifikácie je vynechanie priameho prepojenia medzi vstupnými a výstupnými neurónmi, či vzájomného prepojenia výstupných neurónov. Niektoré úlohy si naopak vyžadujú zaradenie spätného prepojenia z výstupu späť do dynamického rezervoára.

Aktivácia vnútorných neurónov je aktualizovaná podľa vzťahu

$$\mathbf{x}(n+1) = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{y}(n)), \tag{2.7}$$

kde $f = (f_1, \ldots, f_N)$ sú výstupné aktivačné funkcie vnútorných neurónov. Výstup siete sa vypočíta pomocou vzťahu

$$\mathbf{y}(n+1) = f^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n))), \tag{2.8}$$

kde $f^{out} = (f_1^{out}, \ldots, f_N^{out})$ sú aktivačné funkcie výstupných neurónov (lineárne, pre biologickú plauzibilitu sigmoidálne). Vektor $(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n))$ je spojenie vstupného vektora, aktivácie rezervoára a predošlého výstupu.

Ako funkciu f sme použili lineárnu ale aj sigmoidálnu aktivačnú funkciu

$$f(x) = x, f(x) = \tanh(x). \tag{2.9}$$

V prípade vynechania niektorého voliteľného prepojenia na výstup sa v 2.8 vypustí aj príslušná zložka z vektora $(\mathbf{u}(n+1), \mathbf{x}(n+1), \mathbf{y}(n))$. Zároveň sa použije zmenšená matica $\mathbf{W}^{out}$, v ktorej sa vynechajú stĺpce zodpovedajúce váham vynechaných prepojení.

Pre sieť, ktorá z voliteľných prepojení obsahuje len priame prepojenia zo vstupu na výstup a ktorá nepoužíva spätné prepojenie výstupu s rezervoárom, budú aktualizačné rovnice upravené nasledovne. V rovnici pre výpočet aktivácie rezervoára

$$\mathbf{x}(n+1) = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n))$$

sa vynechá príspevok $\mathbf{W}^{back}y(n)$ do aktivácie vnútorných neurónov, keďže spätné prepojenie nie je použité. Pre výpočet výstupu siete dostaneme vzťah

$$\mathbf{y}(n+1) = f^{out}(\mathbf{W}^{out}(\mathbf{u}(n+1), \mathbf{x}(n+1))),$$

kde matica $\mathbf{W}^{out}$ má rozmer $L \times (K+N)$.

2.3.2 Parametre dynamického rezervoára

Dynamický rezervoár je charakterizovaný maticou prepojení $\mathbf{W}$, preto za jeho parametre budeme považovať parametre práve tejto matice. Pre zabezpečenie prítomnosti echo stavov je dôležité preškálovať maticu vnútorných prepojení $\mathbf{W}$ tak, aby bola hodnota jej spektrálneho polomeru $\alpha < 1$.

Škálovací parameter α zohráva dôležitú úlohu aj v charaktere dynamiky rezervoára. Pre úspešný výsledok trénovania ESN siete je kľúčová práve voľba hodnoty α s prihliadnutím na vlastnosti danej úlohy. Pre malé hodnoty α dostaneme rezervoár s rýchlou dynamikou, v ktorom privedené signály veľmi rýchlo doznievajú. Naopak pre hodnoty α veľmi blízke 1 dostávame pomalý rezervoár, ktorý signály tlmí len veľmi pomaly. V takomto rezervoári sa posilňuje vplyv predošlých vstupov na nasledujúci stav. Vstupné a výstupné neuróny zase svoj vplyv strácajú. Podľa doterajších empirických pozorovaní závisí časová škála dynamického rezervoára exponenciálne od $1 - \alpha$. Pre postupnosť $\alpha = 0.99, 0.98, 0.97$ tak dostávame exponenciálny nárast rýchlosti rezervoára. Štandardne sa používajú hodnoty α v rozmedzí $0.7 - 0.98$.

2.3.3 Trénovanie ESN

Základný rozdiel v trénovaní ESN sietí oproti klasickým rekurentným sieťam je v tom, že v trénovacom procese sa upravujú len váhy prepojení výstupnej vrstvy. Matica $\mathbf{W}$ dynamického rezervoára sa teda od svojej počiatočnej inicializácie nemení. Takýto prístup značne urýchľuje proces trénovania a umožňuje pracovať s veľkým počtom neurónov v rezervoári. Kým klasické rekurentné neurónové siete používajú na skrytej vrstve maximálne zopár desiatok neurónov, pri ESN sieťach sa bežne používajú DR so stovkami neurónov. Správna inicializácia DR zabezpečí bohatú odozvu siete na vstupné signály. Úlohou výstupnej vrstvy je zrekonštruovať požadovaný výstupný signál ako lineárnu kombináciu signálov jednotlivých neurónov. Proces trénovania je zodpovedný za výpočet respektíve úpravu výstupnej matice $\mathbf{W}^{out}$ tak, aby sa minimalizovala stredná kvadratická chyba na výstupe.

Pri trénovaní ESN siete môžeme postupovať podľa nasledovného algoritmu, ktorý uvádza Jaeger (2002). V opise algoritmu budeme uvažovať sieť, ktorá používa na výstupe neuróny so sigmoidálnou aktivačnou funkciou. Ďalej budeme predpokladať

prítomnosť spätnoväzbových prepojení. Algoritmus je tak vyčerpávajúci a prípadné použitie zjednodušeného modelu si vyžaduje zrejmé úpravy vzťahov, ktoré sme načrtli už v kapitole 2.3.1.

Majme vstupno-výstupnú postupnosť $(\mathbf{u}(1), \mathbf{d}(1)), \ldots, (\mathbf{u}(T), \mathbf{d}(T))$. Potom ako výsledok trénovania požadujeme ESN sieť $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back}, \mathbf{W}^{out})$, ktorej výstupy $\mathbf{y}(n)$ aproximujú požadovaný výstup $\mathbf{d}(n)$, ak je sieť riadená vstupnou postupnosťou $\mathbf{u}(n)$.

Treba si však uvedomiť, že natrénovaná sieť je schopná aproximovať požadovaný výstup až potom, čo odznie vplyv prvotnej konfigurácie siete. Budeme teda požadovať, aby natrénovaná sieť aproximovala požadovaný signál v časoch $n = T_0, \ldots, T$, kde $T_0 > 1$. Hodnota T_0 závisí od dynamických vlastností siete a pohybuje sa v hodnotách $T_0 = 10$ (pre rýchle siete), až po $T_0 = 500$ (pre pomalé siete).

1. krok: Vytvoríme sieť $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back})$, ktorá má vlastnosť echo stavov. Na vytvorenie matice vnútorných prepojení $\mathbf{W}$ použijeme nasledovný postup:

1. Náhodne vygenerujeme maticu interných váh $\mathbf{W}_0$.
2. Normalizujeme maticu $\mathbf{W}_0$ na maticu $\mathbf{W}_1$ so spektrálnym polomerom 1 a to vynásobením prvkov matice $\mathbf{W}_0$ hodnotou $1/|\delta_{max}|$, kde $|\delta_{max}|$ je spektrálny polomer matice $\mathbf{W}_0$.
3. Preškálujeme maticu $\mathbf{W}_1$ na maticu $\mathbf{W} = \alpha\mathbf{W}_1$, kde $\alpha < 1$ je škálovací koeficient, čím získame maticu $\mathbf{W}$, ktorá má spektrálny polomer α.
4. Týmto získame nenatrénovanú sieť $(\mathbf{W}_{in}, \mathbf{W}, \mathbf{W}_{back})$, ktorá má (doposiaľ nebol preukázaný opak) vlastnosť echo stavov bez ohľadu na výber matíc $\mathbf{W}_{in}$ a $\mathbf{W}_{back}$. Ich prvky teda môžeme náhodne vygenerovať z intervalu $[-1, 1]$ a následne preškálovať podľa potreby.

2. krok: Navzorkovanie trénovacej dynamiky siete.

1. Inicializácia siete do ľubovoľného stavu, napr. $\mathbf{x}(0) = 0$.
2. Necháme sieť bežať cez trénovaciu postupnosť, pre $n = 0, \ldots, T$, so vstupným signálom $\mathbf{u}(n)$. Namiesto výstupu siete vnucujeme signál $\mathbf{d}(n-1)$. Výpočet stavu sa tak realizuje pomocou rovnice:

$$x(n+1) = f^{out}(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{back}\mathbf{d}(n)) \tag{2.10}$$

3. V čase $n = 0$, kde $d(n)$ nie je definované, použijeme $\mathbf{d}(n) = 0$.
4. V každom časovom kroku po uplynutí času T_0 zaznamenáme stav siete $\mathbf{x}(n)$, ako nový riadok do matice stavov M. Nakoniec tak dostaneme maticu veľkosti $(T - T_0 + 1) \times (K + N + L)$.

5. Tak isto v každom kroku po uplynutí času T_0 zaznamenáme invertovaný (inverznou funkciou k aktivačnej funkcii výstupných neurónov) požadovaný výstup siete $\tanh^{-1}(\mathbf{d}(n))$, ako nový riadok matice $\mathbf{D}$. Takto nakoniec dostaneme maticu o veľkosti $(T - T_0 + 1) \times L$.

Pri vytváraní matíc $\mathbf{M}$ a $\mathbf{D}$ je dôležité dať si pozor na to, aby sme pridávali vektory $\mathbf{x}(n)$ a $\tanh^{-1}(\mathbf{d}(n))$. Ľahko sa totiž môže stať, že do matíc vložíme dvojicu vektorov $\mathbf{x}(n)$ a $\tanh^{-1}(\mathbf{d}(n-1))$.

3. krok: Výpočet výstupnej matice sa redukuje na výpočet súčinu pseudoinverznej matice $\mathbf{M}^+$ s maticou $\mathbf{D}$, čím dostaneme maticu $(\mathbf{W}^{out})^+$ veľkosti $(K + N + L) \times L$, ktorej i-ty stĺpec obsahuje váhy od všetkých neurónov k i-tému výstupnému neurónu. Po transponovaní tejto matice dostávame hľadanú maticu $\mathbf{W}^{out}$:

$$\mathbf{W}^{out} = (\mathbf{M}^+\mathbf{D})^T \tag{2.11}$$

4. krok: V tejto fáze máme natrénovanú ESN sieť $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{back}, \mathbf{W}^{out})$, ktorá je pripravená na použitie. Od tohto okamihu môžeme sieť používať pomocou aktualizačných rovníc 2.7 a 2.8.

2.4 Chyba RMSE

Pri vyhodnocovaní výstupnej chyby neurónových sietí používame chybovú funkciu RMSE (Root Mean Square Error) definovanú pre dva vektory $\theta_1 = x_{1,1}, \ldots, x_{1,n}$ a $\theta_2 = x_{2,1}, \ldots, x_{2,n}$ nasledovne:

$$RMSE(\theta_1, \theta_2) = \sqrt{MSE(\theta_1, \theta_2)}, \tag{2.12}$$

kde MSE (Mean Squared Error) je stredná kvadratická odchýlka:

$$MSE(\theta_1, \theta_2) = \sum_{i=1}^{n} \frac{1}{n}(x_{1,i} - x_{2,i})^2. \tag{2.13}$$

Kapitola 3

Virtuálne prostredie

Na splnenie cieľov diplomovej práce bolo potrebné rozhodnúť sa, ako bude realizované virtuálne prostredie. Do úvahy prichádzali v zásade dve možnosti. Naprogramovanie virtuálneho simulovaného prostredia od základov s využitím už existujúcich fyzikálnych a grafických prostredí. Zvažovaný bol najmä fyzikálny engine ODE (Open Dynamic Engine). ODE je voľne dostupná knižnica na verné simulovanie vzájomnej interakcie pevných telies, zohľadňujúc silové pôsobenie. Ako druhá alternatíva sa ponúkala možnosť použiť virtuálny softvérový simulátor robota iCub, ktorý je zhodou okolností postavený na spomínanej knižnici ODE. V oboch prípadoch bolo zjavné, že so sebou nesú niekoľko výhod, ale na druhej strane aj rôzne úskalia. Na virtuálne prostredie boli kladené nasledujúce požiadavky:

- robotické rameno s dotykovými senzormi,
- možnosť ovládať rameno implementovaným neurálnym modelom,
- vizuálne zosnímanie scény vo forme obrázku,
- možnosť vytvárať objekty rôznych veľkostí, tvarov a farieb,
- gravitácia, zotrvačnosť, trenie,
- získanie pozície objektov a detekcia ich kolízií.

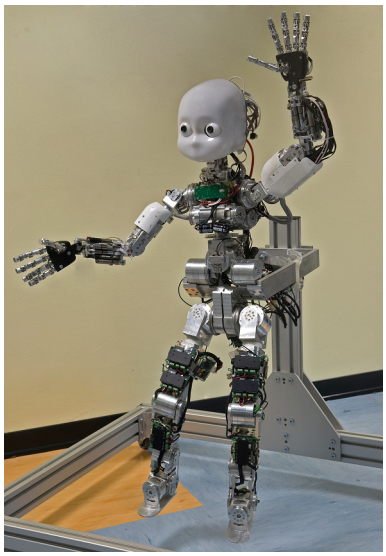
Knižnica ODE spĺňa väčšinu požadovaných bodov, bolo by ale nutné nadizajnovať robotické rameno. Taktiež by bolo potrebné navrhnúť vhodné komunikačné rozhranie s virtuálnou scénou. Odmenou by bola väčšia flexibilita pri vytváraní scény a pri prípadných neskorších zmenách požiadaviek. Nakoniec rozhodnutie padlo v prospech simulátora iCub. Ten už obsahuje dostačujúco navrhnutú scénu, ako aj verný model humanoidného robota iCub, ktorý opíšeme v nasledujúcej kapitole 3.1. Odrádzajúcim faktorom by mohla byť väčšia závislosť od existujúcej implementácie simulátora. Tvorba scény je obmedzená len dostupným rozhraním simulátora. Pridávanie novej

funkcionality, či opravovanie chýb závisí od jeho tvorcov. I keď je kód simulátora otvorený, jeho úprava by v dôsledku rozsiahlosti bola veľmi náročná. V posledných rokoch boli zverejnené viaceré experimenty napríklad Marocco a kol. (2010) a Macura a kol. (2010), ktoré potvrdzujú použitie simulátora iCub ako univerzálneho hotového riešenia.

V práci sme sa zamerali aj na preskúmanie možností využitia simulátora iCub. Prikladáme preto príručku na použitie simulátora iCub, kapitola 5, ktorá môže ďalším študentom pomôcť usúdiť, či je iCub vhodný na nasadenie v ich experimentoch. Zároveň má príručka uľahčiť zoznámenie sa s inštaláciou a používaním simulátora a umožniť tak sústredenie sa na samotné jadro práce.

3.1 Robot iCub

iCub je meno humanoidného robota vyvinutého ako výsledok projektu RobotCub. Vývoj kognitívnej robotiky a snaha porozumieť ľudskej kognícii sú jednými so základných cieľov výpočtovej kognitívnej vedy. Značný dôraz sa kladie na otvorenosť projektu čo do softvérovej, tak aj hardvérovej stránky. Zdrojové kódy programov a takisto kompletné náčrty jednotlivých komponentov robota sú voľne dostupné a môžu byť použité širokým spektrom vedeckých pracovníkov pracujúcich v príbuzných oblastiach.



Obr. 3.1: Fotografia reálneho modelu robota iCub, (robotcub.org).

Rozmery a fyzické schopnosti iCub-a verne kopírujú dva a pol ročné dieťa. iCub je určený na skúmanie v úlohách, kde sa môže učiť vykonávať rôzne akcie a interagovať

s prostredím. Umožňuje tak overovanie rôznych hypotéz a kognitívnych modelov v reálnom svete.

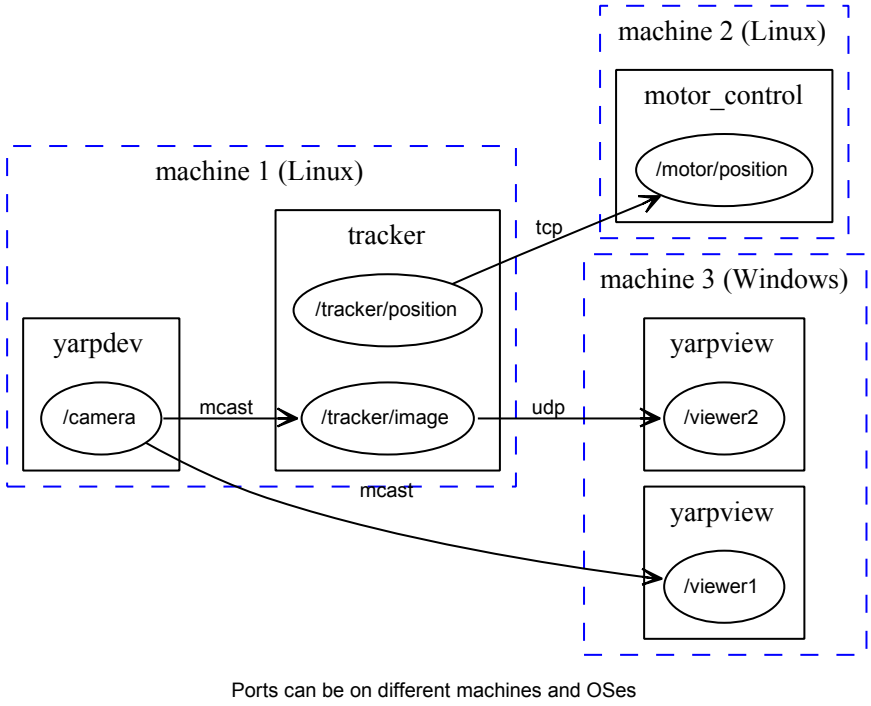
Robot je vysoký 90cm a váži necelých 23kg. Spolu obsahuje 53 stupňov voľnosti, rozdelených nasledovne: sedem pre pohyb ramena a ďalších deväť pre ruku, šesť na každej nohe a po tri pre pás, krk a pohyb očí. Bližšie detaily o návrhu jednotlivých častí, či rozsahu pohybu stupňov voľnosti je možné nájsť v článku Tsakarakis a kol. (2007) a konkrétnejšie pre návrh hlavy v Beira a kol. (2006), či spodnej časti tela robota v Tsagarakis a kol. (2006). Pre každý stupeň voľnosti je možné nastavovať a zisťovať jeho aktuálnu pozíciu. Prítomné sú aj gyroskopy, akcelerometre, mikrofóny a senzory tlaku. Prirodzenou požiadavkou je dostupnosť dotykových senzorov, o ktoré môže byť ruka robota rozšírená, pomocou špeciálnej rukavice. V každom oku je umiestnená jedna kamera, ktorá poskytuje farebný obraz s rozmermi 640x480 (VGA) a frekvenciou 30 obrázkov za sekundu. Senzorické a motorické informácie sú spracované pomocou zabudovanej dosky počítačového modulu PC104 a poskytované ďalej pomocou komunikačného rozhrania. Takto navrhnutý iCub, je najvernejšou realizáciou humanoidného robota.

3.2 YARP (Yet Another Robot Platform)

Na všetku komunikáciu s robotom sa využíva „middleware“ YARP (Yarp, 2011). Snahou open-source projektu YARP je vytvorenie univerzálneho rozhrania pre komunikáciu rôznych robotických zariadení. Dobrý návrh takéhoto rozhrania si vyžaduje značné skúsenosti, pokiaľ má spĺňať kvalitatívne nároky ako rýchlosť, univerzálnosť, modulárnosť, rozšíriteľnosť. Vedcom má uľahčiť prácu a umožniť sústrediť sa na výskum. Projekty využívajúce YARP majú vďaka opakovanej použiteľnosti a modularite väčšiu šancu, že nezapadnú do zabudnutia a budú zakomponované v ďalších projektoch. Tvorcovia sa snažia osloviť čo najväčšiu cieľovú skupinu. Preto je YARP vyvíjaný ako platformovo nezávislý. Naprogramovaný je v jazyku C++, ktorý je sám o sebe veľmi dobre prenositeľný. Jazykom C++ to však nekončí, pomocou SWIG (Simplified Wrapper and Interface Generator) sú generované „wrappery“ pre množstvo ďalších jazykov (Java, Perl, Python, Matlab, C#...).

Základnou úlohou YARP-u je prenos dát medzi senzormi, procesormi a aktuátormi. Ten prebieha po ethernetovej sieti. Použitý abstraktný model je nezávislý od spôsobu prenosu dát a umožňuje komunikáciu každého s každým. Spôsob prenosu dát sa môže pre každé spojenie meniť. Je možné použiť zdieľanú pamäť, TCP alebo UDP, unicast alebo multicast. Komunikácia prebieha cez spojenia medzi portami procesov. Každý port má svoje meno (napr. /icub/left_arm). Na preklad mien portov na adresy a číslo portu slúži YNS (Yarp Name Server). Spojenia medzi portami sa vytvárajú a rušia pomocou sieťového rozhrania Network. Bližšie informácie

o motivácii, implementácii a vlastnostiach platformy YARP možno nájsť v článku (Fitzpatrick a kol., 2008).



Obr. 3.2: Príklad komunikácie prostredníctvom platformy YARP (Fitzpatrick a kol., 2008).

Na obrázku 3.2 je znázornený príklad využitia platformy YARP na komunikáciu viacerých procesov. Komunikácia môže prebiehať medzi procesmi na jednom počítači, rovnako ale aj medzi procesmi bežiacimi na rôznych strojoch, či operačných systémoch. Zjavná je aj modularita a jednoduchosť distribuovania výkonu.

3.3 Simulátor robota iCub

Počítačové simulácie zohrávajú vo výskume dôležitú úlohu. Preto je aj v prípade robota iCub vyvíjaný softvérový simulátor (Tikhanoff a kol., 2008). Virtuálne prostredie umožňuje rýchlejší vývoj a testovanie algoritmov, bez nutnosti vyťažovania reálneho robota. Navyše je možné v bezpečí odladiť možné nebezpečné správanie robota, ktoré by mohlo viesť k jeho poškodeniu. Simulátor je navrhnutý s cieľom čo najvernejšie kopírovať fyzikálne a dynamické vlastnosti robota. Simulovaný iCub bol navrhnutý na základe špecifikácie jeho reálnej predlohy, tak aby ostali podstatné parametre zachované (rozmery, rozloženie hmotnosti, počet stupňov voľnosti,

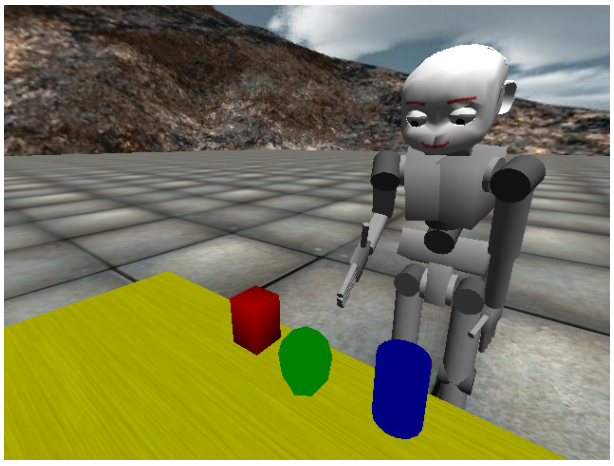
pohyblivosť...).

Architektúra simulátora zahŕňa 3D model robota a sveta, v ktorom je umiestnený. Na výpočet pohybu objektov, ich interakcie a vzájomných kolízií je použitá knižnica ODE. Fyzikálna simulácie beží kvôli rýchlosti oddelene od vizualizácie a je postavená na knižniciach OpenGL a SDL (Simple DirectMedia Layer). Na komunikáciu so simulátorom je implementované rozhranie, zhodné s reálnym modelom robota, používajúce YARP. Po spustení simulátora je vytvorená rovnaká sada portov, akú poskytuje fyzický robot.

Svet v ktorom sa nachádza simulovaný robot má vlastný port, pomocou ktorého je napríklad možné na scénu umiestňovať rôzne preddefinované objekty [1], importovať 3D modely objektov z 3D modelovacích softvérov (3ds Max, Blender...), zisťovať pozíciu objektov, ale aj pozíciu dlane robota. Vytvárané objekty môžu byť statické (pevne umiestnené na svojom mieste) a dynamické, na ktoré vplýva gravitácia a môžu sa pohybovať po priestore. Simulátor je neustále zdokonaľovaný, aj počas písania práce boli pridané niektoré užitočné funkcie, ktoré sme využili. Pre kompletný opis aktuálnej funkcionality je dobré sledovať webovú stránku (RobotCub, 2011). Podrobnejší opis použitia simulátora uvádzame v príručke 5.

3.4 Návrh scény

Návrh scény pre úlohu skúmanú v našej práci je znázornený na obrázku 3.3. Simulovaný robot iCub je postavený pred pevnú dosku. Na doske sú rozmiestnené tri objekty rôzneho tvaru a farby. Všetky sú v dosahu jeho ramena a zároveň v zornom poli kamery umiestnenej v jeho pravom oku. Kameru v ľavom oku nepoužívame.



Obr. 3.3: Ukážka použitej scény vo virtuálnom simulátore iCub.

[1] V čase písania práce boli dostupné objekty: kváder, guľa a valec.

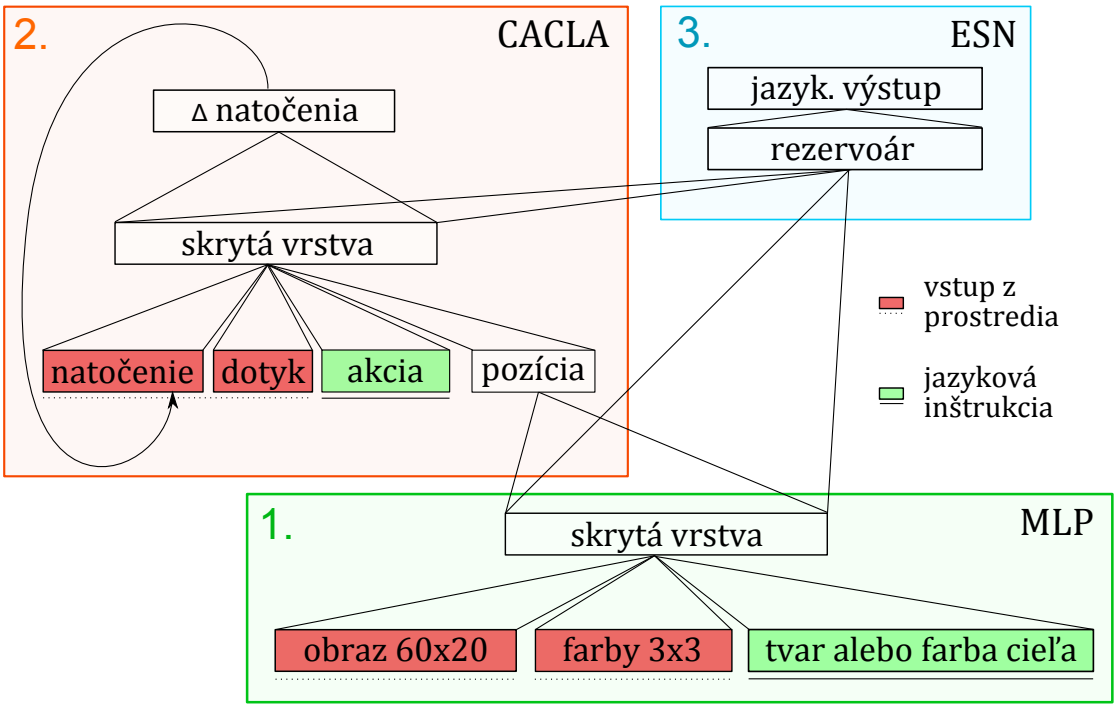
Pôvodne bolo zamýšľané použitie troch zabudovaných objektov simulátora. Pri ich použití sa ale vyskytli dva problémy. Guľa sa pri najmenšom dotyku odkotúľala preč zo stola, čo komplikovalo trénovanie akcií s týmto objektom. Ďalší problém sa objavil pri valci. Z nezistených príčin na ňom nefungovala detekcia kolízií s prstami ruky. Namiesto posunutia valca cezeň robotove prsty prechádzali. Súčasne nereagovali ani dotykové senzory umiestnené na prstoch. Tieto komplikácie nás viedli k využitiu možnosti importovania vlastných vymodelovaných 3D objektov. Valec bol nahradený totožným tvarom. Guľu sme mierne upravili sploštením jej spodnej časti tak, aby sa zabránilo prílišnému gúľaniu. Oba nové tvary sme vymodelovali pomocou programu Blender a exportovali do formátu podporovaného simulátorom.

Robot je mierne naklonený smerom k doske na zlepšenie dosahu objektov. Hlavu má trochu predklonenú a oči sú vhodne nasmerované smerom nadol. Na vykonávanie rôznych akcií s objektami používame pravé rameno. Nohy a ľavé rameno sme nepoužili. Aby sa iCub neprevrátil, zafixovali sme mu bedrá.

Kapitola 4

Návrh a implementácia modelu

V tejto kapitole predstavujeme náš model neurálneho systému na akvizíciu významov a generovanie jednoduchých akcií. Úlohou modelu je vygenerovať postupnosť motorických pohybov pre robotické rameno, ktorá reprezentuje vykonanie akcie, opísanej jazykovou inštrukciou privedenou na vstup. Zároveň požadujeme, aby bol model schopný vykonanú akciu pomenovať.



Obr. 4.1: Návrh neurálneho modelu.

Na obrázku 4.1 uvádzame schematické znázornenie modelu. Model sa skladá z troch samostatných vzájomne prepojených modulov založených na neurónových sieťach, ktoré sa trénujú nezávisle. Úlohou prvého modulu je lokalizovať pozíciu cieľového objektu na scéne, z nízko-úrovňovej vizuálnej informácie a opisu vlastností cieľového objektu. Druhý modul je trénovaný vykonávať akcie smerom k objektom na scéne na základe jazykového príkazu, ktoré vedú k požadovanej postupnosti pohybov. Tretí modul je trénovaný s cieľom získať jazykové pomenovanie vykonanej akcie.

Bližší opis implementácie jednotlivých modulov a výsledky uvádzame v nasledujúcich podkapitolách.

4.1 Modul 1 – Lokalizácia cieľového objektu

Prvý modul je klasická dopredná viacvrstvová neurónová sieť, s jednou skrytou vrstvou, trénovaná na mapovanie vstupnej trojice [IMAGE, COLOR, TARGET] na pozíciu cieľového objektu [POSITION]. IMAGE je bitová mapa získaná z reálnej snímky troch objektov na scéne (rôzneho tvaru), ktorá je spracovaná pomocou knižnice OpenCV na rozmer 60×20 pixelov v odtieňoch šedej, pričom obsahuje výrez detekovaných hrán cieľových objektov. FARBA lokalisticky kóduje farby objektov pomocou deviatich neurónov (tri pre každý objekt). Farby sú vyberané bez opakovania z množiny {červená, zelená, modrá}. Pre každý obrazový vstup sú tri neuróny aktivované na hodnotu 1, ostatné sú 0. Posledná zložka vstupu CIEĽ určuje objekt, s ktorým sa má zvolená akcia vykonať. Cieľový objekt je daný slovom špecifikujúcim jeho farbu (3 neuróny) alebo tvar (3 neuróny). POZÍCIA na výstupe modulu je reprezentovaná tromi neurónmi (vľavo, v strede, vpravo). Presné zloženie vstupného vektora je na obrázku 4.2.

IMAGE	OBJECT COLOR									TARGET					
	LEFT			MIDDLE			RIGHT								
60x20	R	G	B	R	G	B	R	G	B	B	C	S	R	G	B
01...01	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0

B-BOX
C-CYLINDER
S-SPHERE
R-RED
G-GREEN
B-BLUE

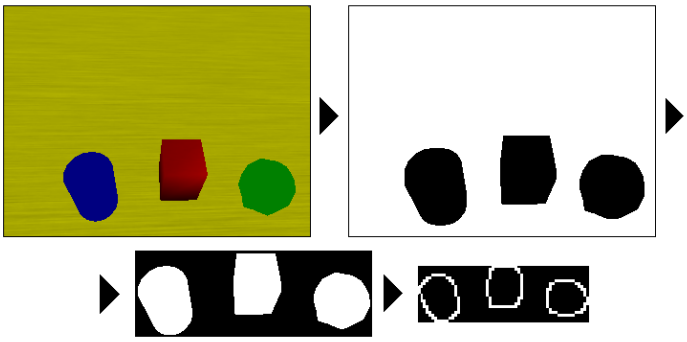
Obr. 4.2: Zloženie vstupného vektora pre modul 1.

Prvých 1200 neurónov reprezentuje za sebou poukladané riadky obrázka, potom nasledujú farby objektov. Podľa uvedenej ukážky sú objekty v poradí zľava doprava zafarbené červenou, zelenou a modrou farbou. Posledných 6 neurónov určuje cieľový objekt, v tomto prípade je ním guľa. Na skrytej a aj výstupnej vrstve sme použili sigmoidálne neuróny.

Vzhľadom na to, že TARGET na vstupe reprezentuje slová, funkcia modulu 1 môže byť interpretovaná ako porozumenie významu týchto slov so zameraním sa na pozíciu cieľového objektu, na základe tvaru a farby trojice objektov na scéne. Výsledky modulu uvádzame v podkapitole 4.1.2

4.1.1 Spracovanie vizuálneho vstupu

Farebný obraz z kamery zachytáva časť scény, v ktorej sa nachádza stôl s tromi objektami v dosahu ramena robota. Výstup z kamery simulátora má rozmer 320×240 obrazových bodov. Úlohou prvého modulu je na základe obrazovej informácie rozhodnúť, na ktorej z troch pozícií (vľavo, v strede, vpravo) sa nachádza objekt opísaný jazykovou inštrukciou. Spracovanie obrazu, lokalizácia a rozpoznávanie tvarov objektov, je sám o sebe zložitý problém, ktorého riešenie nie je cieľom tejto práce. Preto je obraz z kamery pred privedením na vstup siete spracovaný pomocou knižnice OpenCV do prijateľnej podoby. Toto spracovanie by bolo možné nahradiť modulom využívajúcim neurónovú sieť, predpokladáme, že agent takýmto modulom disponuje.



Obr. 4.3: Proces spracovania obrazu pomocou knižnice OpenCV.

Príklad postupnej úpravy pôvodného obrazu z kamery robota, až do finálnej podoby vhodnej pre spracovanie neurónovou sieťou, je zobrazený na obrázku 4.3 Pri spracovaní obrazu sme postupovali nasledovne:

1. Dopredu máme pripravený obrázok prázdneho stola s informáciou o jeho farbe. Pomocou funkcie `cvCreateHist` vypočítame jeho histogram. Ten využijeme pri spätnej projekcii na farebný obrázok z kamery. Spätná projekcia priradí každému pixelu vstupného obrázka intenzitu podľa toho, ako veľmi bola zastúpená jeho farba v obraze, ktorého histogram použijeme. My sme použili histogram obrázka stola, ktorý obsahuje len jeho farby. Výsledkom spätnej projekcie je teda obrázok v odtieňoch šedej, na ktorom sú vysvietené pixely rovnakej

farby ako stôl. Na elimináciu šumu sme aplikovali funkcie erózie (`cvErode`) a dilatácie (`cvDilate`).

2. V tomto kroku nájdeme hrany stola, na základe ktorých vyrežeme z obrázka len časť, kde sa nachádza stôl s objektami. Zavolaním funkcie `cvHoughLines2` získame zoznam úsečiek, ktoré sa nachádzajú na obrázku, pre nami zvolené parametre minimálnej dĺžky sú to práve hrany stola. Na základe nich potom spravíme vhodný výrez.

3. Z obrázka chceme získať len tú časť, na ktorej sú objekty. Z doteraz získaného obrázka spravíme negatív, čo nám stmaví stôl a vysvieti objekty na ňom. Pomocou funkcie `cvFindContours` nájdeme kontúry. Pre každú kontúru vieme vypočítať jej obsah. Ďalej budeme pracovať len s kontúrami, ktoré reprezentujú objekty na stole. Tie majú väčší obsah, ako experimentálne určená hranica. Pomocou funkcie `cvContourBoundingRect` spočítame najmenší spoločný obdĺžnik, ktorý objekty obsahuje a vyrežeme ho. Tento obdĺžnik si s určitou rozmerovou rezervou zapamätáme a používame ho bez zmeny aj pri ostatných rozloženiach objektov.

4. Pomocou Cannyho detektora (funkcia `cvCanny`) získame obrázok s vyznačenými hranami objektov. Ten na záver zmenšíme na rozmer 60 x 20 pixelov. Pred privedením na vstup modulu 1 preškálujeme hodnoty pixelov do intervalu $\langle 0, 1 \rangle$.

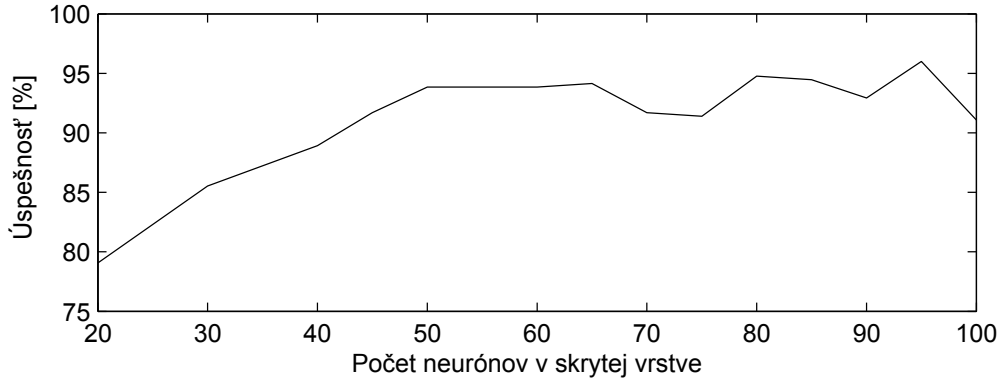
4.1.2 Výsledky modulu 1

Na vstupe modulu 1 je vektor, ktorý obsahuje informáciu o rozmiestnení objektov na scéne (3! možností[1]), o ich farbe (3! možností[2]) a cieľovom objekte (6 možností[3]). To nám dáva spolu $3! \cdot 3! \cdot 6 = 216$ rôznych vstupných vektorov. Túto množinu sme pred každým trénovaním náhodne rozdelili na trénovaciu množinu (2/3 vstupov) a testovaciu množinu (1/3 vstupov). Trénovali sme pomocou algoritmu spätného šírenia chyby spomenutého v kapitole 2.1. Cez trénovaciu množinu sme počas epochy prechádzali v náhodnom poradí. Na výstupe siete sú tri sigmoidálne neuróny, ich aktivácia je v intervale $\langle 0, 1 \rangle$. Za sieťou zvolenú cieľovú pozíciu sme považovali tú, ktorej zodpovedajúci neurón mal najvyššiu hodnotu.

Na obrázku 4.4 je znázornená závislosť testovacej úspešnosti v závislosti od počtu neurónov v skrytej vrstve. Z grafu vidieť, že s rastúcim počtom neurónov rastie aj úspešnosť na testovacej množine. Po dosiahnutí hranice 50 neurónov je úspešnosť

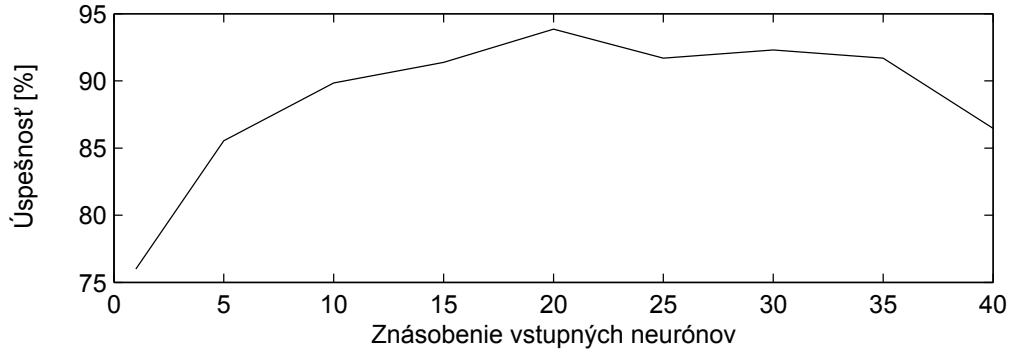
[1] Rozmiestňujeme tri rôzne objekty na tri pozície bez opakovania.
[2] Rozmiestňujeme tri rôzne farby na tri pozície bez opakovania.
[3] Pýtame sa na tvar alebo farbu. Vždy je aktívny iba jeden zo šiestich neurónov.

okolo 95% a rastie už len mierne. Ďalšie zlepšenie závisí skôr od zvoleného ukončovacieho kritéria pre trénovanie a rýchlosti učenia. Pri niektorých behoch bola dokonca dosiahnutá úspešnosť 100%. Na trénovacej množine bola sieť neomylná. Trénovanie siete z grafu na obrázku 4.4 trvalo 600 epoch, s rýchlosťou učenia 0,1. Pre každý počet neurónov bola vypočítaná priemerná úspešnosť z piatich nezávislých behov. Každý beh začínal s náhodne inicializovanými váhami prepojení.



Obr. 4.4: Úspešnosť siete na testovacej množine v závislosti od počtu neurónov v skrytej vrstve.

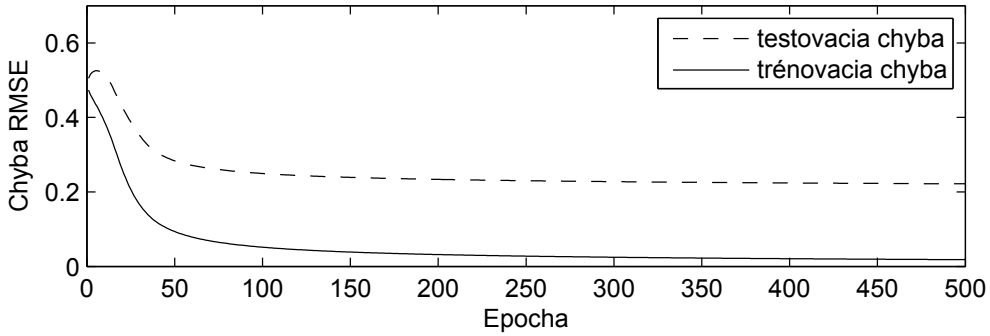
Vstupný vektor siete bol mierne modifikovaný oproti opisu na obrázku 4.2. Neuróny ktoré kódujú farby objektov a cieľ boli znásobené dvadsaťkrát. To znamená, že pre každý z týchto neurónov za ním nasledovalo devätnásť ďalších s rovnakou hodnotou.



Obr. 4.5: Úspešnosť siete na testovacej množine v závislosti od znásobenia počtu nevizuálnych neurónov vo vstupnom vektore.

V pôvodnom návrhu bola obrazová časť vstupu natoľko dominantná, že sieť mala problém správne generalizovať, najmä ak bol cieľový objekt určený farbou. Na ob-

rázku 4.5 je vykreslená úspešnosť siete v závislosti od veľkosti znásobenia počtu neobrazových neurónov. Dáta boli získané rovnakým spôsobom ako v predošlom grafe s tým, že na skrytej vrstve bolo 50 neurónov. Vykreslené hodnoty sú priemerom z piatich behov pre každé znásobenie neurónov. Bez znásobenia vstupných neurónov bola úspešnosť len okolo 75%. Najlepšie výsledky sme dosiahli pri dvadsaťnásobnom počte nevizuálnych neurónov. Za touto hranicou úspešnosť opäť klesá.



Obr. 4.6: Vývoj chyby na trénovacej a testovacej množine počas trénovania.

Na obrázku 4.6 je znázornený vývoj chyby počas 500 epoch trénovania. Trénovacia aj testovacia chyba prudko klesá počas prvých 50 epoch. Obe chyby ďalej pokračujú v miernom klesaní. Testovacia chyba nezačína časom rásť, čiže k pretrénovaniu siete pri 50 neurónoch na skrytej vrstve ešte nedochádza. Jednotlivé neuróny na výstupe sa oproti požadovanej hodnote v priemere mýlili na trénovacej množine o $\pm 0,0372$ a na testovacej množine o $\pm 0,1360$.

Modul 1 sa nám podarilo úspešne natrénovať na rozpoznávanie pozície cieľového objektu, určeného farbou alebo tvarom, z vizuálnej informácie zahŕňajúcej hrany objektov a ich farbu. Úspešnosť nad 95% na testovacej množine považujeme za dostatočnú. Z výsledkov vyplýva, že generalizácia modulu je veľmi dobrá. Po natrénovaní dokáže správne odpovedať na vstupy, ktoré pri trénovaní neboli prítomné. Napríklad, ak sa na pravej pozícii nachádza červená kocka, modul správne zodpovie na otázku: Kde sa nachádza červený objekt? To napriek tomu, že sa v trénovaní táto otázka pri danom rozložení objektov a farieb nevyskytovala. Modul teda správnu odpoveď vydedukoval na základe vstupov s inou vizuálnou informáciou (iné rozloženie farieb a objektov). Napríklad by to mohli byť tie, v ktorých sa na pravej pozícii nachádza červená guľa alebo červený valec.

4.2 Modul 2 – Vykonávanie motorickej akcie

Druhý modul, trénovaný na vykonávanie akcií, je založený na biologicky vierohodnom učení s posilňovaním (reinforcement learning, RL) (Sutton a Barto, 1998), ktoré nevyžaduje znalosť postupnosti natočenia kĺbov ramena, ktoré vedie k vykonaniu príslušnej akcie. Nakoľko modul pracuje v spojitom priestore stavov a akcií, implementovali sme ho pomocou architektúry aktér-kritik, ktorá je spojitým zovšeobecnením učenia s posilňovaním. Konkrétne sme použili algoritmus CACLA (van Hasselt a Wiering, 2007), spomenutý v podkapitole 2.2.3. Experimentovali sme aj s jeho rôznymi modifikáciami. Modul 2 pozostáva z dvoch viacvrstvových neurónových sietí. Aktér sa učí vykonávať akcie s objektami. Kritik sa učí odhadovať ohodnotenie rôznych stavov agenta, ktoré sú použité pri trénovaní aktéra.

Aktér vykonáva mapovanie vstupu [STATE, ACTION, POSITION] na výstup [STATE-CHANGE]. Pre aktéra sme zvolili 40 skrytých neurónov. Tvoril teda sieť 11-40-4. Počas trénovania bola na navrhovanú akciu aplikovaná explorácia na zabezpečenie nového správania. Kritik mapuje stav [STATE, ACTION, POSITION] na ohodnotenie stavu [VALUE]. Na skrytej vrstve kritika sme použili 20 neurónov. Obe siete sme trénovali pomocou algoritmu CACLA s využitím algoritmu spätného šírenia chyby. Všetky neuróny tohto modulu mali sigmoidálnu aktivačnú funkciu.

Zdôvodnenie zvolených parametrov, podrobnejší opis trénovania a výsledky uvádzame v nasledujúcich podkapitolách.

4.2.1 Stav agenta

Aktér a aj kritik majú na vstupe rovnaký stavový vektor. Proprioreceptívna informácia z ramena iCub-a zahŕňa natočenie štyroch kĺbov (nastavenie ruky sa nemenilo) preškálovaných na interval $\langle -1, 1 \rangle$. Dotykový senzor sa aktivuje na hodnotu 1, ak dôjde k dotyku s ľubovoľnou časťou ruky, inak je 0. Cieľová pozícia POSITION sa preberá z výstupu prvého modulu a určuje pozíciu objektu (vľavo, v strede, vpravo). Výstup aktéra reprezentuje zmenu natočenia ramena. V ďalšom časovom kroku je na vstup aktéra privedená nová pozícia ramena v stave po vykonaní navrhovanej zmeny. ACTION zodpovedá jazykovému opisu akcií (point, touch, push) a je reprezentovaná tromi neurónmi. Štruktúra stavového vektora je znázornená na obrázku 4.7. Agent je v stave vykonávanie akcie push na strednom objekte.

Hodnoty natočenia ramena j'_1 až j'_4 sú po získaní od robota v stupňoch. Rozsah použitých kĺbov je uvedený v tabuľke 4.1. V stavovom vektore sú už preškálované použitím vzťahu:

$$j_i = 2 * (j'_i + |Min_i|)/(|Min_i| + |Max_i|) - 1, \tag{4.1}$$

ARM JOINTS				TOUCH	ACTION			POSITION		
					PO	TO	PU	L	M	R
$\dot{j}_1$	$\dot{j}_2$	$\dot{j}_3$	$\dot{j}_4$	{0,1}	0	0	1	0	1	0

PO-POINT
TO-TOUCH
PU-PUSH
L-LEFT
M-MIDDLE
R-RIGHT

Obr. 4.7: Zloženie vstupného vektora pre modul 2.

kde Min_i a Max_i sú hraničné natočenia príslušných kĺbov z tabuľky 4.1.

Kĺb č.	**Rozsah pohybu**	
	Min	**Max**
1	-95	90
2	0	161
3	-37	100
4	-6	106

Tabuľka 4.1: Rozsah pohybu použitých kĺbov ramena.

4.2.2 Funkcia odmeny

Ukázalo sa, že návrh funkcie odmeny hrá podstatnú úlohy v dosiahnutí požadovaného správania agenta po natrénovaní. V našej úlohe bol spočiatku najväčší problém dosiahnuť dostatočné rozlíšenie medzi akciami dotyku (POINT) a zatlačenia objektu (PUSH). Funkcia odmeny nemôže byť príliš zložitá, inak by mal kritik problém naučiť sa odhadovať ohodnotenia stavov. Pre každú akciu sme používali odlišnú funkciu odmeny:

- Pri akcii **PUSH** dostával agent ako odmenu zápornú Euklidovskú vzdialenosť dlane od počiatočnej pozície cieľového objektu. V prípade, že agent chce maximalizovať získanú odmenu, musí cieľový objekt vytlačiť zo svojej pôvodnej pozície. To práve zodpovedá významu akcie push.

- Základ odmeny pre akciu **TOUCH** je opäť záporná Euklidovská vzdialenosť dlane od počiatočnej pozície cieľového objektu. Navyše, ak sa počas vykonávania akcie rameno dotkne iného ako cieľového objektu (prípadne ho posunie), agent dostane penalizáciu odčítaním vhodnej konštanty od pôvodnej odmeny (použili sme 0,5). V prípade, že sa dotkne cieľového objektu, dostane penalizáciu 0,3. I keď je táto penalizácia v rozpore s intuíciou, dosiahli sme s ňou najlepšie výsledky. Zdôvodnenie uvádzame nižšie, za návrhom funkcie odmeny.

- Pre akciu **POINT** bolo dôležité dosiahnuť, aby bola odmena najväčšia vtedy, keď rameno ukazuje na cieľový objekt. Euklidovská vzdialenosť dlane od objektu, ktorú sme použili na výpočet odmeny pre zvyšné dve akcie, nie je vyhovujúca. Hoci by bola dlaň akokoľvek blízko cieľového objektu, prsty ruky môžu stále ukazovať na úplne iný objekt. Pre každú pozíciu sme si zaznamenali natočenia kĺbov, ktoré zodpovedali nasmerovaniu prstov ruky na cieľovú pozíciu. Výsledná odmena bola záporná Manhattanská vzdialenosť aktuálneho natočenia ramena od pripraveného cieľového natočenia.

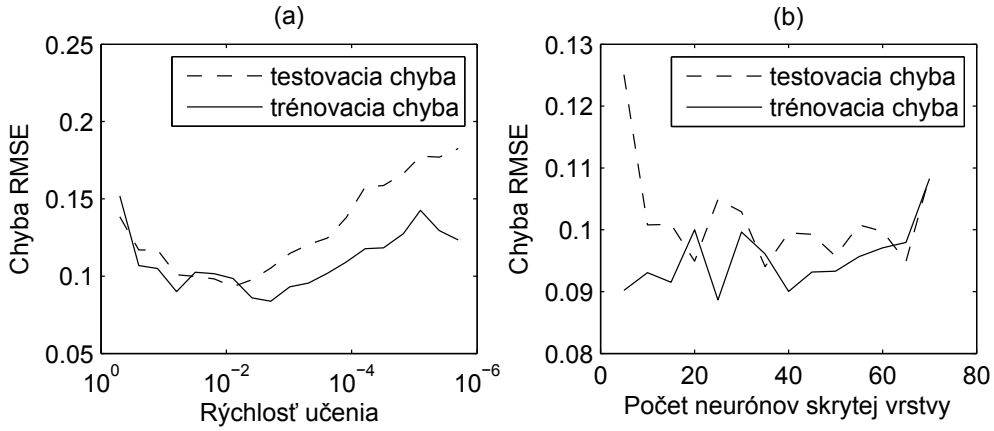
Pre akciu TOUCH by bolo intuitívne dať agentovi pri dotyku správneho objektu bonus, vtedy ale agent namiesto dotyku objekt stále odtláčal. Toto správanie sa dá vysvetliť tak, že pozícia objektov nie je pre nášho agenta plne pozorovateľná, nemá teda informáciu o ich absolútnej pozícii. Preto nevie zo svojho aktuálneho stavu zistiť, ako veľmi posunul cieľový objekt. Pri trénovaní sa vo finálnej fáze akcie objekt po dotyku o kúsok odrazil od ramena, ktoré s ním tým pádom stratilo dotyk a agent bonus. V tom momente je z jeho pohľadu potrebné urobiť pohyb smerom k objektu a zvýšiť si tým odmenu. To viedlo ale k ďalšiemu dotyku a odrazeniu objektu. Preto agent po natrénovaní namiesto akcie dotyku objekt vždy posunul. Tým že sme agentovi dávali penalizáciu aj pri dotyku s cieľovým objektom, sme dosiahli to, že nech je k nemu rameno akokoľvek blízko, agent má tendenciu sa priblížiť ešte viac. Akonáhle sa ale dotkne, posunie rameno mierne späť. Po natrénovaní sa agent pri akcii TOUCH cyklicky dotýka objektu, bez väčšieho odsunutia. Iným riešením by mohla byť história pohybu ramena, ktorú by si agent mohol uchovávať použitím rekurentnej neurónovej siete alebo zakomponovaním pozície objektov na scéne do stavu agenta.

4.2.3 Zvolenie parametrov pre aktéra a kritika

Pri použití neurónových sietí je dôležité nastavenie ich parametrov, najmä počtu neurónov a rýchlosti učenia. V ideálnom prípade je možné tieto parametre systematicky prehľadať a zvoliť optimálne hodnoty pre daný problém. V našej úlohe sme potrebovali nastaviť parametre pre siete aktéra a kritika. Systematické prehľadávanie by znamenalo nechať bežať simuláciu s robotom iCub pre rôzne parametre. To sa ukázalo ako nereálne, nakoľko simulátor beží obmedzenou rýchlosťou a potrebné prehľadávanie by trvalo príliš dlho. Preto bolo potrebné použiť iný postup.

Na zistenie vhodných parametrov sme si vytvorili trénovacie množiny pre aktéra aj kritika. Najskôr sme nechali bežať trénovanie modelu s odhadnutými parametrami sietí. Potom, čo pohyby robota prestali byť úplne chaotické a začali smerovať k cieľovým objektom, sme začali vytvárať trénovacie množiny s 5000 pohybovými inštrukciami. Pre kritika sme si zaznamenávali odmenu získanú v každom kroku, pre

aktéra zase akcie, ktoré viedli k lepším stavom. Tieto dáta boli použité ako požadovaný výstup. Zaznamenávali sme si aj príslušné vstupy (stavové vektory), zhodné pre aktéra aj kritika. Na pripravených dátach sme potom bez nutnosti použitia simulátora systematicky preskúmali vplyv parametrov na chybu siete.

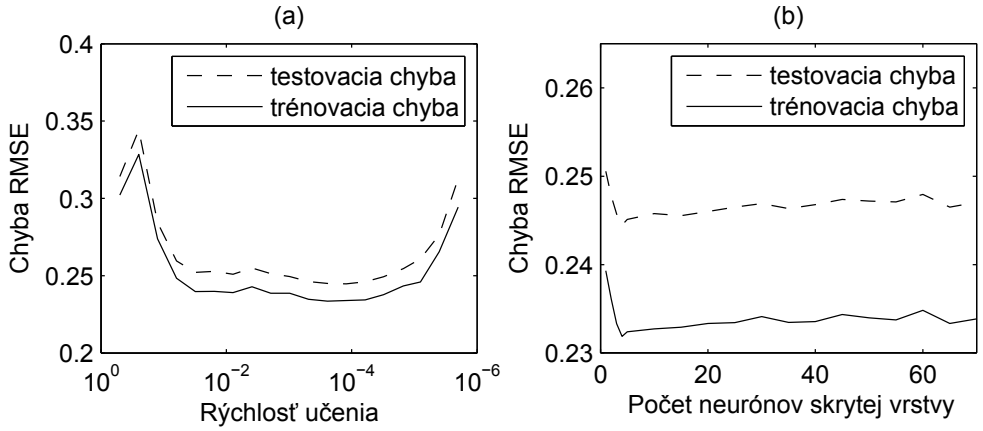


Obr. 4.8: Chyba kritika pre rôzne parametre siete. (a) Závislosť chyby od rýchlosti učenia. (b) Závislosť chyby od počtu neurónov v skrytej vrstve.

Obr. 4.8a znázorňuje chybu siete so 40 neurónmi na skrytej vrstve trénovanej počas 500 epoch. Chyba klesá spolu s klesajúcou rýchlosťou učenia a minimum dosahuje pri rýchlosti učenia okolo 0,01. Potom postupne rastie, pretože učenie je natoľko pomalé, že počas 500 epoch nestihne chyba dostatočne klesnúť. Najlepšie výsledky sme dosiahli pri nulovej hodnote diskontného faktora. Obr. 4.8b znázorňuje závislosť chyby od počtu neurónov na skrytej vrstve. Trénovanie prebiehalo počas 250 epoch, pri rýchlosti učenia 0,01. Z grafu vidieť, že počet neurónov nemá príliš veľký vplyv na úspešnosť siete. V podstate vyhovuje každý počet neurónov vyšší ako 10. My sme použili pre kritika 20 neurónov na skrytej vrstve, s rezervou pre prípadnú zmenu funkcie odmeny.

Rovnako sme analyzovali hodnoty parametrov pre aktéra. Výsledky sú v grafoch na obrázku 4.9. Aktéra sme trénovali počas 200 epoch. Podľa grafu (a) je vidieť, že optimálna rýchlosť učenia je medzi rozmedzí 0,01 až 0,0001. Ako výslednú rýchlosť učenia pre aktéra sme zvolili hodnotu 0,001. Počet neurónov na skrytej vrstve opäť nie je veľmi významný, ako naznačuje graf (b). Vzhľadom na to, že dáta na analýzu sú len čiastočne reprezentatívne, zvolili sme až 40 neurónov na skrytej vrstve aktéra.

Na základe analýzy sme pre výsledné trénovanie modulu použili parametre uvedené v tabuľke 4.2.



Obr. 4.9: Chyba aktéra pre rôzne parametre siete. (a) Závislosť chyby od rýchlosti učenia. (b) Závislosť chyby od počtu neurónov v skrytej vrstve.

	Architektúra	**Rýchlosť učenia**	**Aktivačná funkcia**
Aktér	11-40-4	0,001	tanh
Kritik	11-20-1	0,01	tanh

Tabuľka 4.2: Parametre použité na trénovanie modulu 2.

4.2.4 Výsledky modulu 2

Modul sme trénovali pomocou algoritmu 1 v dvoch fázach. V prvej fáze sme trénovali aktéra vtedy, keď sa agent dostal vykonaním akcie do stavu s väčšou odmenou ($r_t < r_{t+1}$). Dôvodom bola snaha urýchliť celé trénovanie. Aktér spočiatku vykonáva chaotické pohyby a jeho trénovanie závisí od kvality ohodnocovania stavov kritikom. Ten ich ale na začiatku trénovania ohodnocuje tiež s veľkou chybou. Aby aktér čo najskôr začal smerovať pohyby relevantným smerom, učili sme ho na základe odmeny a nie ohodnotenia stavu kritikom. Takto sme urýchlili trénovanie aktéra a zároveň aj kritika, ktoré by pomocou simulátora trvalo veľmi dlho. Po asi 1500 epizódach sme prešli do druhej fázy, počas ktorej prebiehalo trénovanie ďalších 300 epoch pomocou pôvodného algoritmu CACLA.

Jedna epizóda (epocha) predstavuje vykonanie alebo trénovanie náhodne vybranej dvojice akcia-pozícia. Dĺžka epizódy pri trénovaní bola 75 pohybových inštrukcií. Ak agent počas epizódy posunul nesprávny objekt o určitú hranicu, bolo mu povolených už len zopár pohybov a epizóda bola ukončená. V zostávajúcich krokoch mal agent možnosť zistiť, aké akcie budú viesť k zvýšeniu odmeny, ktorá mu pri posunutí bola znížená. Ak by sme epizódu neskrátili, agent by sa mohol učiť chybné správanie, pretože nemá informáciu o novej pozícii posunutých objektov. Niektoré akcie by sa mu zdanlivo zdali vhodné, aj keď pri pôvodnej pozícii objektov by boli nesprávne.

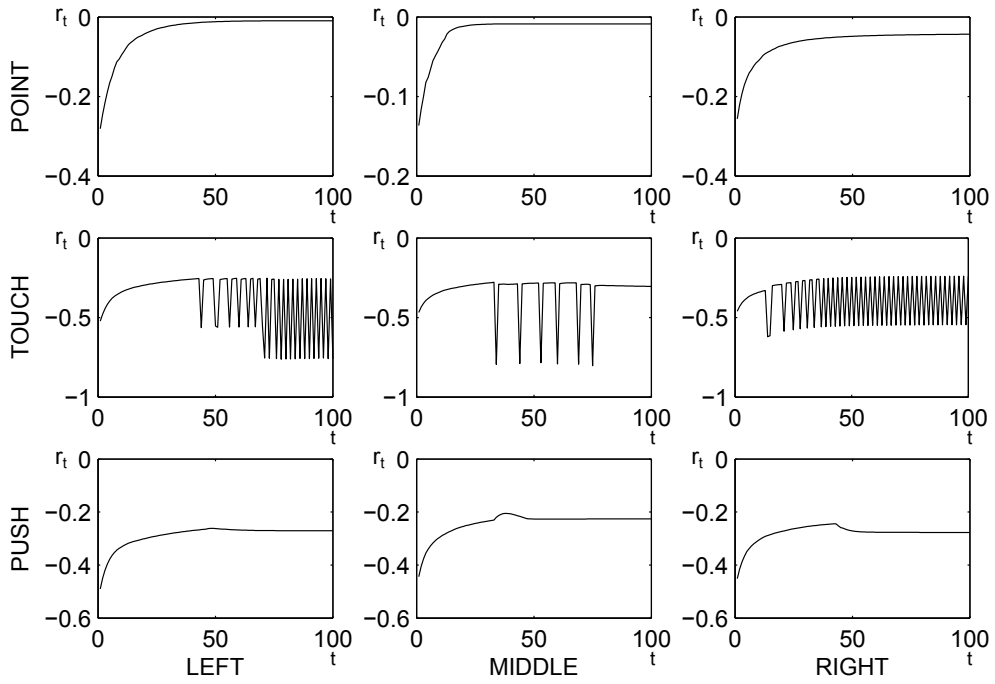
Jeden krok epizódy prebieha nasledovne:

1. Získanie aktuálneho stavu agenta s_t, odmeny r_t a ohodnotenia stavu $V_t(s_t)$.
2. Aktér navrhne akciu, ktorá je s pravdepodobnosťou 30% modifikovaná pravidlom explorácie. Veľkosť modifikácie sa zmenšuje so znižujúcou sa vzdialenosťou od cieľového objektu.
3. Vykoná sa akcia získaná v kroku 2.
4. Získanie nového stavu agenta s_{t+1}, odmeny r_{t+1} a ohodnotenia nového stavu kritikom $V_t(s_{t+1})$.
5. Úprava váh kritika. Ak $V_t(s_t) > V_t(s_{t+1})$, úprava váh aktéra.

Na začiatku trénovania, keď je aktér náhodne inicializovaný, sú navrhované akcie veľké a v náhodnom smere. Explorácia len málokedy preváži akcie do správneho smeru. Preto sme navrhnuté akcie zmenšili na tretinovú veľkosť. Ešte lepším riešením by mohla byť inicializácia váh aktéra na nižšie hodnoty, čím by sa aj navrhované akcie zmenšili.

Model sa dokázal natrénovať uspokojivo vykonávať požadované akcie. Na obrázku 4.10 je zobrazený vývoj odmien pri vykonávaní trénovaných akcií na všetky pozície objektov. Pre všetky akcie je charakteristické, že pohybové inštrukcie sú na začiatku vykonávania akcie veľké a časom sa zmenšujú.

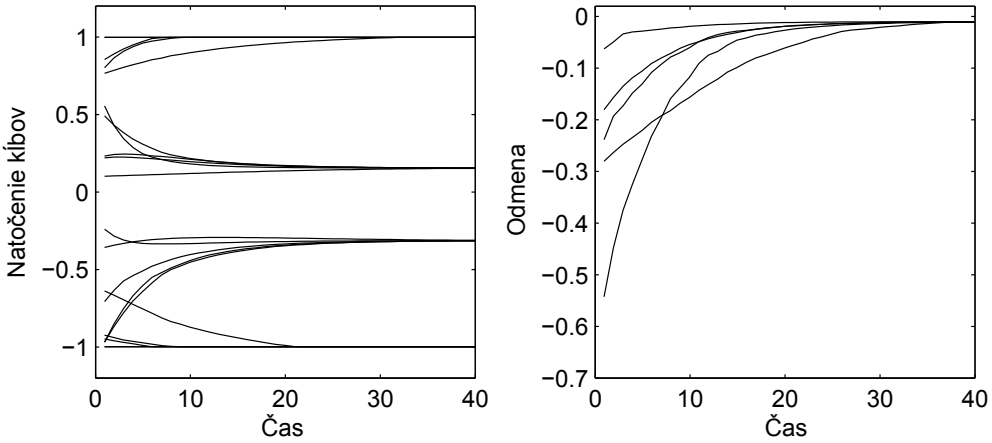
- Akcia POINT bola vykonávaná perfektne. Odmena počas celej akcie postupne rastie. Konečné natočenia kĺbov ramena boli presne zhodné s požadovanými.
- Odmena pri akcii TOUCH spočiatku rastie, až kým nenastane dotyk s objektom. Potom nasleduje fáza, kedy sa ruka robota striedavo dotýka objektu, čo sa prejavuje kolísaním odmeny (dôsledok návrhu funkcie odmeny). Veľkosť kolísania odmeny pri dotyku nezodpovedá vzdialenosti ruky od objektu. Tá sa pri striedavom dotyku pohybuje len málo. Kolísanie odmeny by mohlo byť znížené použitím menšej penalizácie pri dotyku. Pri niektorých kombináciách pozície a tvaru cieľa bol objekt až priveľmi odsunutý.
- Vykonávanie akcie push bolo veľmi dobré. Akonáhle nastal dotyk s objektom, rameno začalo objekt súčasne posúvať do strany, mimo svojej pôvodnej pozície. Na grafe sa objavuje od miesta dotyku mierny pokles odmeny. To preto, že pri posúvaní objektu sa rameno vzďaľuje od pôvodnej pozície objektu. Ak by sa podarilo objekt odsunúť tak, aby ramenu nezavadzalo, bolo by rameno presunuté na pôvodnú pozíciu objektu, kde je maximálna odmena.



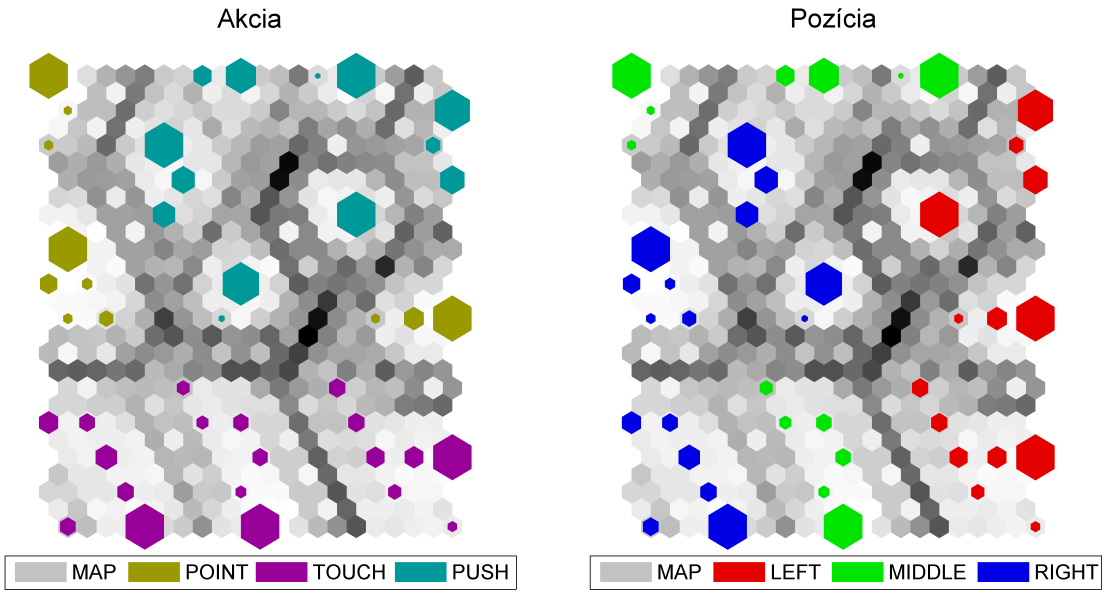
Obr. 4.10: Vývoj odmeny počas vykonávania naučených akcií.

Počas trénovania štartovalo rameno na začiatku každej epizódy z rovnakej pozície. Po natrénovaní však na začiatočnej pozícii nezáležalo. Na obrázku 4.11 je znázornená generalizácia pri vykonávaní akcie point-left z piatich výrazne odlišných počiatočných pozícií. Na ľavom grafe je priebeh natočenia kĺbov. Každý kĺb konverguje do svojej cieľovej pozície. Na pravom grafe je znázornený vývoj odmeny, ktorá taktiež dosiahne na konci akcie rovnakú hodnotu. Ak je rameno premiestnené v priebehu vykonávania akcie, agent sa prispôsobí a správne pokračuje vo vykonávaní akcie z novej pozície. Podobné správanie je aj pri vykonávaní ostatných akcií.

Počas vykonávania akcií sme zaznamenávali aktivácie skrytej vrstvy aktéra, na základe ktorých sme skonštruovali vizualizáciu vysokorozmernej reprezentácie akcií do dvojrozmernej mapy (Obr. 4.12). Na redukciu dimenzie sme použili samoorganizujúcu sa mapu SOM (Kohonen a kol., 2001). Na mape vidieť jasne identifikovateľné zhluky (svetlé miesta), ktoré reprezentujú jednotlivé akcie. Na ľavom obrázku sú farebne odlíšené akcie, na pravom zase pozícia cieľa. Veľkosť farebnej značky zodpovedá početnosti prislúchajúcich aktivácií. Každá akcia má vo svojom zhluku menšie a jednu väčšiu značku. Najväčšia značka reprezentuje cieľovú pozíciu, v ktorej rameno zotrvávalo už od polovice času prideleného na testovanie akcie. Menšie značky zodpovedajú trajektórii počas vykonávania akcie, pred dosiahnutím cieľovej pozície.



Obr. 4.11: Vývoj natočenia ramena a odmeny, pri vykonávaní akcie point-left z rôznej počiatočnej pozície.



Obr. 4.12: Vizualizácia aktivácií skrytej vrstvy aktéra pomocou samoorganizujúcej sa mapy SOM s toroidnou topológiou.

4.3 Modul 3 – Pomenovanie vykonávanej akcie

Tretí modul využíva sieť s echo stavmi na generovanie slovného pomenovania práve vykonávanej akcie. Modul uskutočňuje mapovanie dvojice [M1-HID, M2-HID] na výstup [SENTENCE]. M1-HID je aktivácia skrytej vrstvy prvého modulu, ktorá umožňuje pomenovať vlastnosti objektu (farbu alebo tvar). M2-HID je aktivácia skrytej vrstvy aktéra modulu 2, ktorá umožňuje predikovať a teda aj pomenovať typ vykonávanej akcie. SENTENCE je reprezentácia výstupnej vety, 3 neuróny pre akciu a 6 neurónov pre tvar a farbu. Vety sú dvojslovné, vždy sú aktívne dva neuróny, jeden pre akciu a druhý pre identifikáciu objektu.

4.3.1 Výsledky modulu 3

ESN sme použili s 50-timi neurónmi, v rezervoári so spektrálnym polomerom 0,9 a konektivitou 20%. Matica vstupných váh bola vygenerovaná náhodne z intervalu $\langle -1, 1 \rangle$. Spätné prepojenia z výstupu do rezervoára sme nepoužili. V celej echo sieti sme použili lineárne neuróny. Voľba parametrov pre tento modul nebola kritická.

Na trénovanie sme zvolili dávkový algoritmus uvedený v podkapitole 2.3.3. Pred trénovaním sme si pripravili dáta z prvého a druhého modulu. Z natrénovaného prvého modulu sme si uložili 256 aktivácií skrytej vrstvy pre všetky rôzne kombinácie vstupov. Jedna aktivácia obsahovala aktivačné hodnoty 50-tich neurónov skrytej vrstvy. Podobne sme si zaznamenali aktivácie skrytej vrstvy aktéra, so 40-timi neurónmi, počas vykonávania všetkých troch akcií, do všetkých troch smerov. Dĺžka jednej akcie bola 100 krokov.

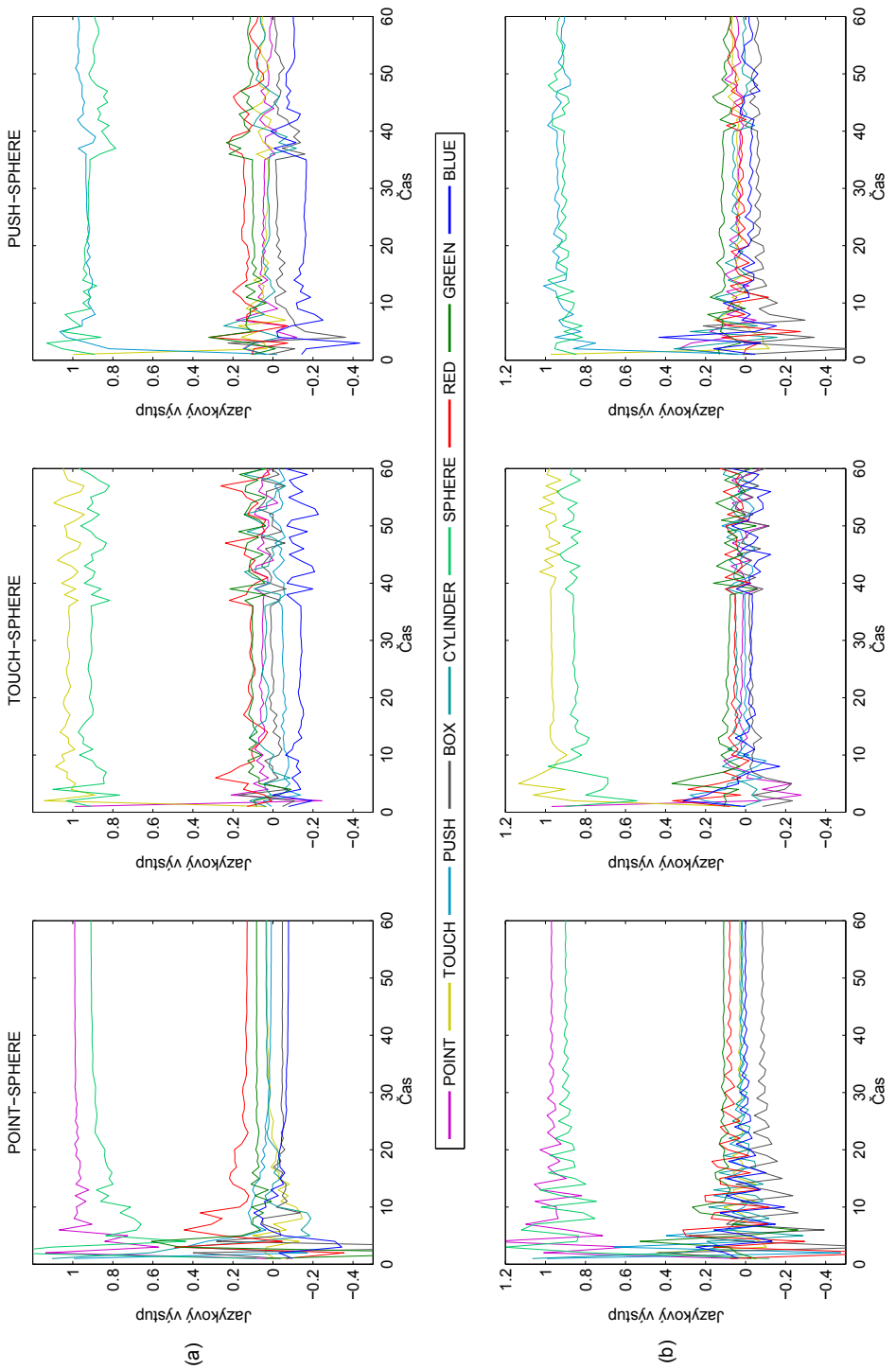
Každá z 216 skrytých aktivácií modulu 1 zodpovedá konkrétnej cieľovej pozícii a na túto pozíciu je možné vykonať každú z troch akcií (trikrát postupnosť 100 skrytých aktivácii aktéra). Vstupné dáta pre modul 3 sú zložené z 216 blokov dĺžky 300. Bloky sú poukladané za seba. Každej skrytej aktivácii modulu 1 prislúcha jeden blok, ktorý obsahuje postupnosť 300 aktivácií aktéra, pre tri akcie smerujúce na pozíciu určenú príslušnou aktiváciou modulu 1, zreťazených s príslušnou skrytou aktiváciou prvého modulu. Jeden blok teda obsahuje 300 zložených aktivácií dĺžky 90. Prvých 40 prvkov zloženej aktivácie v bloku sa mení (podľa priebehu aktivácií aktéra pre tri akcie) a zvyšných 50 prvkov je statických (príslušná aktivácia modulu 1). Pre každý vstup trénovacej množiny sme si uložili požadovaný výstup siete, ktorý obsahoval aktivované dva príslušné neuróny.

Najlepšie výsledky sme dosiahli, keď sme vo fáze zbierania stavov pre výpočet pseudoinverzie ukladali všetky stavy. Takto sme dosiahli na trénovacej množine úspešnosť vyše 99%. Vo fáze testovania sme nechali natrénovaný modul pomenovať akcie, ktoré boli vykonávané z náhodnej začiatočnej pozície. Aj v tomto prípade bola úspešnosť modulu nad 99%. Výstup modulu pri pomenovaní akcií s guľou sú

znázornené na grafoch obrázka 4.13. V prvom riadku (a) je vývoj výstupu siete na trénovacej množine. V druhom riadku (b) je znázornené pomenovanie akcií, ktoré začínajú z náhodnej začiatočnej pozície.

Na grafoch vidieť, že jazykový výstup sa veľmi rýchlo (do piatich krokov) ustáli na správnom pomenovaní akcie. Predpokladali sme, že najskôr sa prejaví informácia o cieľovom objekte a až potom typ akcie. Očakávali sme, že rozlíšiť medzi akciami PUSH a TOUCH bude možné až ku koncu akcií, keď sa ich trajektórie začnú líšiť. Táto hypotéza sa ale nepotvrdila. Pre všetky akcie sú stavy skrytých vrstiev dostatočne odlišné na to, aby ich už na začiatku vykonávania akcie bolo možné rozlíšiť.

Požadovaná aktivácia výstupných neurónov nemá hodnotu presne 1, ale pohybuje sa okolo nej. Je však dostatočne odlišná od aktivácií zvyšných výstupných neurónov, ktoré kolíšu prevažne v rozmedzí $\pm 0,2$ okolo nuly. Pri akciách začínajúcich z náhodnej pozície je možné pozorovať, že ustálenie výstupu trvá trochu dlhšie, ako pri trénovacích, štartovacích pozíciách. Pri akciách TOUCH a PUSH nastáva v 40-tom kroku dotyk s cieľovým objektom. V tomto mieste sa začína meniť pohyb ramena, čo sa premieta vo forme miernych oscilácií, nielen na neurónoch kódujúcich akciu, ale rovnako aj na neurónoch, ktoré opisujú cieľový objekt.



Obr. 4.13: Jazykový výstup modulu 3, počas vykonávania akcií s guľou. Akcia začína z pozície použitej pri trénovaní (a) a z náhodnej pozície (b).

Kapitola 5

Príručka simulátora iCub

Softvérový simulátor iCub je najkomplexnejší simulátor humanoidného robota. V rámci vypracovania tejto diplomovej práce som pomerne veľa času venoval skúmaniu jeho možností a využiteľnosti. Návrh algoritmov pre túto platformu nesie viacero výhod, no má aj svoje slabé miesta. V tejto kapitole preto zhrnieme získané poznatky, ktoré môžu pomôcť ďalším študentom rýchlejšie sa zorientovať a umožnia im sústrediť sa na jadro svojej práce. Uvádzame odkazy na relevantné zdroje dôležitých informácií potrebných pre vývoj softvéru pre iCub. Ďalej poskytujeme ukážky zdrojových kódov na komunikáciu s robotom, ako aj postrehy, pozorovania a rady nadobudnuté počas práce so simulátorom.

Aktuálna dokumentácia pre projekt robota iCub sa nachádza na oficiálnej wiki stránke (`http://eris.liralab.it/wiki/Main_Page`), ktorá je hlavným rázcestím k manuálu (`http://eris.liralab.it/wiki/Manual`), softvérovej dokumentácii pre iCub (`http://eris.liralab.it/iCub/main`) a pre YARP (`http://eris.liralab.it/yarpdoc/index.html`).

5.1 Inštalácia simulátora iCub

V práci sme používali simulátor v prostredí operačného systému Windows. Prvé experimenty sme realizovali s už skompilovanými binárnymi verziami, tie ale nie sú poskytované na oficiálnych stránkach. Narazili sme pri nich na problém s rýchlosťou simulácie. Preto je nanajvýš vhodné si simulátor skompilovať vo vlastnej réžii. Komunita simulátor neustále vylepšuje a pridáva doň novú funkcionalitu, čo sa môže hodiť. Prekompilovanie simulátora je potom už len otázkou niekoľkých minút. Aj počas písania práce boli pridané niektoré funkcie, ktoré sme využili.

Celú kompiláciu je možné realizovať postupovaním podľa šiestej kapitoly manuálu uvedenej na oficiálnych stránkach `http://eris.liralab.it/wiki/Manual#Six._Software.2C_Compiling_YARP_and_iCub`. Opísaná je inštalácia pre systémy

Windows a Linux, v šestnástej kapitole manuálu je neoficiálny návod pre MacOS. Nakoľko sme používali simulátor na platforme Windows, ďalej zhrnieme inštaláciu v tomto prostredí.

Keď hovoríme o inštalácii, nejde len o samotný simulátor, ale aj o množstvo modulov, pomocných aplikácií, či tutoriálov pre robota iCub. Tieto sú závislé na viacerých knižniciach, ktoré je potrebné nainštalovať najskôr. Patria medzi ne ACE, GTK+ a GTKMM, QT3, OpenGL, GLUT, GSL, OpenCV, IPOPT. Posledná menovaná nie je nevyhnutá, bez nej ale budú vynechané napríklad moduly používajúce inverznú kinematiku. Opis inštalácie jednotlivých knižníc sa nachádza na stránke `http://eris.liralab.it/wiki/PrepareWindows`. V našom prípade pri jeho dodržaní prebehla inštalácia bez väčších problémov.

Aby autori čo najviac uľahčili vývoj, na začiatku stránky je odkaz na predkompilované knižnice, či projektové súbory pre rôzne verzie vývojového prostredia MS Visual Studio. V prípade kompilácie priamo so zdrojových kódov knižníc je uvedený ku každej knižnici postup, väčšinou s využitím multi-platformového prekladača CMake, ktorý vytvorí potrebné projektové súbory pre cieľové vývojové prostredie.

Po nainštalovaní prerekvizít môžeme prejsť k inštalácii YARP-u a samotného softvéru iCub. Postupujeme podľa návodu na stránke `http://eris.liralab.it/wiki/CompileWindows`.

Ak prebehla inštalácia úspešne, mali by sme byť schopní spustiť simulátor. To si overíme tak, že najskôr spustíme YARP server `yarpserver.exe` z priečinka so skompilovaným YARP-om. Ten zabezpečuje komunikáciu a spravuje prepojenie portov. Potom spustíme samotný simulátor `iCub_SIM.exe`. Malo by sa otvoriť okno simulovaného prostredia s virtuálnym robotom iCub. Od tohto momentu sa dá s robotom a prostredím komunikovať.

Aktuálna dokumentácia funkcionality simulátora je spísaná na stránke `http://eris.liralab.it/wiki/Simulator_README`. Zaujímavé informácie a riešenia niektorých problémov je možné nájsť aj na fóre `http://robotcub-hackers.2198711.n2.nabble.com/`.

5.2 Použitie simulátora iCub

Konfiguračný súbor simulátora obsahuje zopár nastavení pre jeho beh. Po ich zmene je nutné simulátor reštartovať. V krátkosti uvádzame bližšiu charakteristiku pre niektoré z nich:

- Sekcia `[PARTS]` umožňuje zapínať jednotlivé časti robota. Vypnuté časti neodpovedajú na komunikáciu a znižujú výpočtovú náročnosť.
- `fixed_hip` pevne zafixuje bedrá robota. Bez zafixovania sa môže robot pri hýbaní rukami vlastnou tiažou prevrátiť.

- `elevation` - mierne zdvihne robota do výšky.
- `pressure` - zapína dotykové senzory na prstoch. Ak sú senzory vypnuté, prsty aj s dlaňou sa správajú ako jeden veľký dotykový senzor s výstup 0 bez dotyku a 1, ak sa ruka niečoho dotýka. Po zapnutí parametra, začnú reagovať jednotlivé prsty na dotyk samostatne. Výstup je z intervalu $\langle 0, 1 \rangle$ a zodpovedá veľkosti tlaku pôsobiaceho na senzor.
- `objects` - určuje, či sú pri spustení simulátora vytvorené pred robotom objekty: stôl, guľa, hranový model kocky. S vypnutým parametrom je stále možné manuálne vytváranie objektov.
- `cover` - zapína vykresľovanie krytu robota.

Pri spustení simulátora je vytvorená sada portov, pomenovaná identicky ako v reálnej verzii robota, pomocou ktorých môžeme s robotom komunikovať. Porty majú textové adresy v tvare `/robotName/robotPart`. Meno robota v simulátore je prednastavené na hodnotu `icubSim`. Robot je rozdelený na šesť častí, každá má svoj vlastný port. Zoznam častí a prislúchajúcich portov je v tabuľke 5.1.

Časť robota	**Adresa portu**
Hlava	`/icubSim/head`
Ľavé rameno	`/icubSim/left_arm`
Pravé rameno	`/icubSim/right_arm`
Ľavá noha	`/icubSim/left_leg`
Pravá noha	`/icubSim/right_leg`
Trup	`/icubSim/torso`
Obraz z kamier	`/icubSim/cam/left` (resp. `right`)
Dotykové senzory	`/icubSim/skin/left_hand` (resp. `right_hand`)
Virtuálnym svetom	`/icubSim/world`

Tabuľka 5.1: Prehľad portov a ich adresy.

Každý z portov má tri verzie s príponami `/rpc:i` pre RPC port, `/state:o` pre port stavu a `/command:i` pre príkazový port.

5.3 Komunikácia pomocou YARP a ovládanie robota

Komunikovať s portami robota sa dá viacerými spôsobmi. Jedna možnosť je použitie príkazového riadka a programu `yarp.exe` na vzdialené volanie funkcií (Remote procedure call - RPC). Natočenie prvého kĺbu ramena ľavej ruky môžeme realizovať nasledovnými príkazmi:

```
yarp rpc /icubSim/left_arm/rpc:i
set pos 0 45
```

Prvý príkaz spustí program yarp a pripojí sa na RPC port vzdialeného volania funkcií pre ľavé rameno. Nasleduje slovný opis vzdialenej procedúry a jej parametrov. Príkaz `help`, na pripojenom RPC porte vypíše syntax dostupných procedúr.

Prirodzenou požiadavkou je komunikácia so simulátorom z prostredia nejakého programovacieho jazyka. YARP poskytuje rôzne spôsoby komunikácie s portami. Úvod do ich použitia je spracovaný na stránke `http://eris.liralab.it/yarpdoc/port_expert.html`. Horeuvedený príklad sa dá v jazyku C++ realizovať podľa algoritmu 2.

Algorithm 2 Použitie RPC v jazyku C++

```
Network yarp;
RpcClient world_client; // Vytvorenie RPC klienta

world_client.open("/sim/armRPC"); // Vytvorenie portu
world_client.addOutput("/icubSim/right_arm/rpc:i");

Bottle cmd, response; // Pripravenie správy
cmd.addString("set");
cmd.addString("pos");
cmd.addInt(0);
cmd.addInt(45);

world_client.write(cmd, response); // Odoslanie správy
printf(response.toString());
```

Obdobne možno využívať aj ostatné funkcie simulátora uvedené v jeho dokumentácii. Či už pomocou programu `yarp.exe` alebo volaním RPC v programovacom jazyku sa napríklad dajú vytvárať a rotovať objekty, zisťovať a nastavovať ich pozícia, zisťovať pozícia dlane ramena, čítať dotykové senzory, importovať vlastné 3D objekty atď. Bližší opis použitia RPC portov je dostupný na adrese `http://eris.liralab.it/yarpdoc/rpc_ports.html`.

Komunikácia pomocou RPC by pre pohyblivé časti robota s veľkým počtom kĺbov bola veľmi nepraktická. Vhodnejšie je použitie motorické rozhrania YARP-u. To umožňuje flexibilné nastavovanie a čítanie rýchlosti, či pozície kĺbov. Detailný opis použitia je uvedený na stránke `http://eris.liralab.it/iCub/main/dox/html/icub_motor_control_tutorial.html`. Ukážka algoritmu 3 vychádza z tutoriálového súboru `%ICUB_ROOT%/iCub/tutorials/src/tutorial_arm.cpp`.

Po zdĺhavejšej úvodnej inicializácii získame oveľa jednoduchší spôsob komunikácie. Ak máme pripravený vektor s natočením kĺbov, jeho odoslanie robotovi je otázka jedného riadku kódu.

Algorithm 3 Použitie motorického rozhrania YARP v jazyku C++

```
Network yarp;

Property options;
options.put("device", "remote_controlboard");
options.put("local", "/test/client");
options.put("remote", "/icubSim/right_arm");

PolyDriver robotDevice(options); // Vytvorenie rozhrania
if (!robotDevice.isValid()) {
  printf("Zariadenie nie je dostupné. Rospoznané zariadenia:\n");
  printf("%s", Drivers::factory().toString().c_str());
  return 0;
}

IPositionControl *pos;
IEncoders *encs;

bool ok = robotDevice.view(pos) && robotDevice.view(encs);

if (!ok) {
  printf("Problém s inicializáciou rozhrania\n");
  return 0;
}

int nj=0;
pos->getAxes(&nj);

Vector encoders, command, tmp;

encoders.resize(nj);
command.resize(nj);
tmp.resize(nj);

for (int i = 0; i < nj; i++) {
  tmp[i] = 30.0;
  pos->setRefSpeed(i, tmp[i]); // Nastavenie rýchlosti kĺbov.
}

command=0;  // Pripravenie natočenia kĺbov.
command[0]=-50;
command[1]=20;
command[2]=-10;

pos->positionMove(command.data()); // Nastavenie natočenia kĺbov.
encs->getEncoders(encoders.data()); // Prečítanie natočenia kĺbov.
```

5.4 Prenos obrazu z kamery

Samotný robot má dve kamery, v každom oku jednu. Simulátor poskytuje navyše jednu kameru prostredia. My sme v práci použili len kameru v pravom oku. Obraz z nej sme spracovávali pomocou knižnice OpenCV. Na prenos obrazovej informácie je možné použiť zdrojový kód z ukážky algoritmu 4. Uvedená je aj funkcia na prevod obrázka do typu **IplImage**, knižnice OpenCV. Opis základného spracovania obrazu je spracovaný na stránke `http://eris.liralab.it/iCub/main/dox/html/icub_basic_image_processing.html`.

Algorithm 4 Prenos obrazu z kamery v jazyku C++ a jeho konverzia do obrazového formátu knižnice OpenCV.

```
BufferedPort<ImageOf<PixelRgb>> cam_right;
cam_right.open("/sim/image/in");
Network::connect("/icubSim/cam/right", "/sim/image/in");

// Prečítanie snímky.
ImageOf<PixelRgb> *yarpImage = cam_right.read();

if (yarpImage != NULL) { // Ak bol prenos úspešný.
  IplImage *img = cvCreateImage(cvSize(yarpImage->width(),
                                       yarpImage->height()),
                                IPL_DEPTH_8U,
                                3);
  // Konverzia do OpenCV formátu.
  cvCvtColor((IplImage*)yarpImage->getIplImage(),
             img,
             CV_RGB2BGR);
  ...
}
```

5.5 Softvér pre robota iCub

Pre robota iCub bolo napísané množstvo aplikácií a modulov. Sú to samostatné programy komunikujúce s robotom. Zoznam sa nachádza na stránke `http://eris.liralab.it/iCub/main/dox/html/group__icub__modules__all.html`. Dajú sa tu nájsť moduly na spracovanie obrazu, uchopovanie objektov, detekciu tváre a mnoho ďalších. Zaujímavé sú najmä grafické používateľské rozhrania na manuálne ovládanie a monitorovanie robota. Veľmi intenzívne sme využívali grafické rozhranie – modul `robotMotorGui`, ktorý umožňuje jednoducho nastavovať natočenia jednotlivých kĺbov.

5.6 Problémy a ich riešenie

V tejto podkapitole zhrnieme problémy, s ktorými sme sa počas používania simulátora stretli v čase písania práce a uvádzame ich prípadné riešenia. Nakoľko simulátor je aktívne vyvíjaný, je možné, že uvedené problémy budú časom odstránené.

5.6.1 Padanie simulátora

Zaznamenali sme viacero situácií, v ktorých simulátor padá. Prvá problémová situácia nastávala po mnohonásobne opakovanom pripájaní sa k portom simulátora. Jednoduchým riešením je nezatvárať zbytočne spojenia s portami, ale držať ich od prvého pripojenia otvorené.

Druhá situácia pri ktorej simulátor padal nastávala veľmi sporadicky. Prejavom boli chybové hlášky obsahujúce text `ODE INTERNAL ERROR...`. O tomto probléme bola zmienka aj na fóre RobotCub Hackers. Domnievame sa, že tieto chyby sa vyskytujú pri veľkej rýchlosti objektov na scéne, eliminovali sa totiž pri nastavení nižšej rýchlosti ramena. Frekvencia výskytu tohto javu sa menila aj podľa verzie knižnice ODE, použitej pri kompilácii simulátora. Extrémne často sa vyskytuje pri skompilovaní simulátora s jednoduchou presnosťou na rozdiel od dvojitej, ktorú odporúča použiť manuál.

5.6.2 Zvýšenie rýchlosti simulácie

Počas trénovania modelov založených na neurónových sieťach je často vykonávaný veľký počet trénovacích epoch. Preto je relevantná otázka maximálnej možnej rýchlosti simulácie. Simulátor iCub je navrhnutý tak, aby rýchlosť pohybov robota bola zhodná s rýchlosťou ľudského ramena. V čase písania práce nebola implementovaná žiadna možnosť nastavenia rýchlosti simulácie. Tento fakt môže byť v istých úlohách limitujúci a preto je vhodné s ním počítať dopredu.

Jedným spôsobom ako zrýchliť učenie je nastavenie vysokej rýchlosti ramena. Tým získame rýchlejšie pohyby ramena, čím zároveň nadobúda veľkú energiu, ktorá pôsobí na zasiahnuté objekty. Ak sa rameno aj pri malej zmene pozície dotkne objektu, ten je odstrčený na podstatne väčšiu vzdialenosť, ako pri nižších rýchlostiach ramena. Nedá sa teda príliš dobre vykonávať jemná motorika.

Na ďalšie zrýchlenie môžeme využiť fakt, že máme k dispozícii zdrojový kód simulátora. Na tomto mieste chceme zdôrazniť výhodu vlastnej kompilácie simulátora iCub. V takomto prípade máme vyriešené všetky závislosti a môžeme smelo skúšať modifikovať zdrojový kód. Po jednoduchej úprave kódu sa nám podarilo získať niekoľkonásobné zrýchlenie simulácie. Skúšali sme modifikovať nastavenie dvoch konštánt v súbore `iCub_Sim.cpp`.

1. **stepsize** - vo volaní funkcie `dWorldStep(odeinit.world, 0.01);`, v tele funkcie `Uint32 OdeSdlSimulation::ODE_process`, ktorá počíta jeden krok simulácie. Podľa dokumentácie tento parameter určuje, koľko sekúnd (virtuálneho sveta) sa má spracovať v jednom kroku simulácie. Určuje teda presnosť simulácie. Zmenou parametra sme ale nedosiahli dobré výsledky. Pri zvýšení hodnoty tohto parametra sa pri najvyšších rýchlostiach ramena objavovali anomálie. Rameno sa začalo nekontrolovateľne pohybovať medzi jeho krajnými polohami. V extrémnych prípadoch zanikla súdržnosť kĺbov a časti robota sa rozleteli po priestore. Zmena tohto parametra sa ukázala ako nevhodná.

2. **int delay** - v tele funkcie `int OdeSdlSimulation::thread_func`. Hodnota tejto premennej určuje v akom časovom intervale tiká časovač, ktorý spúšťa jeden krok simulácie. Jej hodnota slúži na synchronizáciu simulácie s reálnym časom. Znížením hodnoty tejto premennej teda zvýšime frekvenciu krokovania simulácie. Bez problémov sme používali hodnotu `delay = 0`, čím sme získali podstatné zrýchlenie.

5.6.3 Zaseknutie ramena

Pri dotyku ruky so statickými objektami sa nám stávalo, že rameno sa zaseklo a ďalšie inštrukcie neviedli k zmene jeho pozície. Tento jav sa vyskytoval najmä pri vypnutých prstoch ruky. V prípade, že sa robotovi zasekla ruka o stôl zabudovaný v prostredí, jedinou možnosťou odblokovania bolo reštartovanie simulátora. Preto sme v konfiguračnom súbore vypli parameterom `objects` vytváranie preddefinovaných objektov na scéne pri spustení. Namiesto stola sme použili „dosku“ z dostatočne veľkého a tenkého kvádra. Zároveň je možné takto manuálne vytvorené objekty zmazať vzdialenou procedúrou `world del all`, volanou na porte virtuálneho sveta. Ak nám v takto namodelovanej scéne nastalo zaseknutie ramena, zmazaním všetkých objektov sa odblokovalo a bolo možné ho premiestniť na bezpečnú pozíciu. Potom sme nanovo vytvorili potrebné objekty na scéne.

V kontexte tohto problému ešte spomenieme metódu `checkMotionDone` objektu `IPositionControl`, ktorá vracia `TRUE` v prípade, že bol posledný pohyb ramena dokončený. Metóda kontroluje, či boli dosiahnuté cieľové natočenia poslednej motorickej inštrukcie. To sa v prípade zaseknutého ramena nikdy nestane a funkcia neustále vracia `FALSE`. Preto nie je vhodné ju používať ako test ukončenia vykonávania motorickej inštrukcie. V našej úlohe sa osvedčila vlastná jednoduchá metóda, ktorá kontroluje, či sa natočenie ramena od poslednej inštrukcie zmenilo alebo len bez pohnutia stojí na mieste.

Po krátkom pozorovaní najnovšej verzie dostupnej počas písania práce nasvedčuje, že problém zaseknutia ramena bol do istej miery eliminovaný. Rameno už

nezostávalo pevne zaseknuté o statický predmet. Po zaslaní motorickej inštrukcie sa naďalej pohybovalo, aj keď spomalene.

5.6.4 Problém s dotykom valca

Vo verzii simulátora ktorú sme používali sa vyskytoval zvláštny problém z objektom valca, u ktorého nám nefungovala detekcia kolízií na prstoch ruky. Namiesto toho, aby sa valec posunul, cez neho prsty robota voľne prechádzali. Zároveň na dotyk ani nereagovali dotykové senzory na prstoch. Kolízia s dlaňou už ale funkčná bola. Tento problém sme vyriešili namodelovaním vlastného valca v softvéri Blender a jeho importovaní do virtuálneho prostredia pomocou procedúry `world mk model`.

Importované modely už korektne reagovali na dotyk s prstami ruky. Vyriešením jednej záludnosti sa ale objavila iná. Po zmazaní objektov a ich novom importe so vzájomne vymenenými súbormi textúr boli dva objekty vykreslené s rovnakou textúrou. To by až tak nevadilo, keby textúry neurčovali požadovanú farbu objektov.

5.6.5 Oneskorenie obrazu z kamery

Pri čítaní obrazu z kamery sme sa stretli ani nie tak s chybou, ako skôr z vlastnosťou portov s vyrovnávacou pamäťou. Obraz prečítaný z portu nebol zhodný s aktuálnym stavom scény. Problém sme pozorovali pri spracovaní vizuálneho vstupu bezprostredne po vytvorení objektov na scéne. Toto správanie si vysvetľujeme tak, že sme z portu prečítali obrazovú informáciu z vyrovnávacej pamäte, ktorá ešte nebola od vytvorenia objektov aktualizovaná. Toto správanie uvádzame napriek tomu že je štandardné, no ľahko môže byť zdrojom problémov.

Záver

Ciele diplomovej práce sa nám podarilo splniť. Úspešne sme implementovali nový model na báze umelých neurónových sietí, ktorý je schopný vykonávať jednoduché, jazykom opísané akcie s objektami na scéne. Takisto je schopný vykonávanú akciu pomenovať. Náš model na rozdiel od niektorých predošlých, založených na učení s učiteľom, nedostáva pri trénovaní informáciu o nasledujúcom proprioreceptívnom stave. Túto informáciu získava z interakcie s prostredím, pomocou biologicky vierohodného učenia s posilňovaním. Ku každému z troch modulov nášho modelu sme uviedli jeho podrobný návrh, voľbu parametrov, spôsob trénovania a vyhodnotenie úspešnosti.

Prvý modul dokáže spoľahlivo identifikovať pozíciu cieľového objektu na scéne z obrazovej informácie získanej z kamery robota, na základe slovného opisu jeho vlastností. Tento modul sa vyznačuje vynikajúcou generalizačnou schopnosťou, keďže sa nám podarilo pre vhodne zvolené parametre dosiahnuť 100% úspešnosť nielen na trénovacej, ale aj na testovacej množine. Agent bol teda schopný identifikovať pozíciu objektov pri ich rozložení a zafarbení, s ktorým sa pri trénovaní nestretol.

Druhý modul preukázal schopnosť vykonávať tri rôzne akcie na cieľových objektoch, pomocou priameho ovládania ramena simulovaného humanoidného robota iCub. Agent svoje správanie upravuje na základe odmeny získavanej z prostredia. Ukázalo sa, že jej návrh zohráva kľúčovú úlohu v úspešnosti trénovania. Nakoľko agent nedisponuje súradnicami pozícií objektov, je preňho prostredie čiastočne pozorovateľné, čo viedlo k istým neintuitívnym prvkom v návrhu funkcie odmeny. Námetom na rozšírenie práce by mohlo byť preskúmanie vplyvu použitia rekurentnej neurónovej siete na pozícii aktéra a kritika. Aj druhý modul preukázal dobrú generalizačnú schopnosť a akcie dokázal vykonávať aj z nových počiatočných pozícií ramena.

Tretí modul umožnil agentovi slovne opísať svoje naučené správanie. Na základe interných stavov prvých dvoch modulov sme úspešne natrénovali sieť s echo stavmi pomenovať cieľový objekt a vykonávanú akciu. Hypotéza, že počas vykonávania akcie bude najskôr možné definovať objekt a až potom akciu sa nepotvrdila. Aj keď majú akcie PUSH a TOUCH spoločný začiatok svojej trajektórie, stavy skrytej vrstvy aktéra sú dostatočne odlišné na rozlíšenie týchto akcií už na začiatku ich

vykonávania.

V kapitole 5 sme uviedli príručku pre použitie simulátora robota iCub. Obsahuje informácie, ktoré môžu pomôcť ďalším študentom rýchlo sa zoznámiť s možnosťami simulátora. Odkazujeme na oficiálne relevantné dokumenty s dokumentáciou a návodom na inštaláciu. Zahrnuli sme aj ukážky zdrojových kódov, ktoré demonštrujú rôzne spôsoby komunikácie s robotom. Na záver kapitoly sme spomenuli rôzne postrehy a riešenie problémov, s ktorými sme sa pri používaní simulátora stretli.

Literatúra

Beira, R., Lopes, M., Praça, M., Santos-Victor, J., Bernardino, A., Metta, G., Becchi, F. a Saltarén, R. (2006). Design of the robot-cub (icub) head. In *in IEEE - International Conference on Robotics and Automation (ICRA'06)*.

Cangelosi, A., Hourdakis, E. a Tikhanoff, V. (2006). Language acquisition a symbol grounding transfer with neural networks a cognitive robots. Na *IJCNN 2006*.

Cangelosi, A., Tikhanoff, V., Fontanari, J. F. a Hourdakis, E. (2007). Integrating language a cognition: A cognitive robotics approach. *IEEE Computational Intelligence Magazine*, str. 65–70.

Fitzpatrick, P., Metta, G. a Natale, L. (2008). Towards long-lived robot genes. *Robot. Auton. Syst.*, 56:29–45. ISSN 0921-8890.

Inoue, S. a Matsuzawa, T. (2007). Working memory of numerals in chimpanzees. *Current Biology*, 17(23):R1004–R1005, December 2007.

Jaeger, H. (2001). The „echo state“ approach to analysing a training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science.

Jaeger, H. (2002). Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF a the " echo state network" approach. Technical report, Fraunhofer Institute AIS, St. Augustin-Germany.

Jordan, M. I. (1990). *Attractor dynamics a parallelism in a connectionist sequential machine*, str. 112–127. IEEE Press, Piscataway, NJ, USA. ISBN 0-8186-2015-3.

Kohonen, T., R. Schroeder, M. a Huang, T. S. (2001). *Self-Organizing Maps*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3. vydanie. ISBN 3540679219.

Macura, Z., Cangelosi, A., Ellis, R. a Bugmann, D. (2010). A cognitive robotic model of grasping.

Marocco, D., Cangelosi, A., Fischer, K. a Belpaeme, T. (2010). Grounding action words in the sensorimotor interaction with the world: experiments with a simulated icub humanoid robot. *Frontiers in Neurorobotics*, 4(0). ISSN 1662-5218.

RobotCub (2011). Simulator readme - wiki for robotcub a friends. URL `http://eris.liralab.it/wiki/Simulator_README`.

Rumelhart, D. E., Hinton, G. E. a Williams, R. J. (1988). *Learning internal representations by error propagation*, str. 673–695. MIT Press, Cambridge, MA, USA. ISBN 0-262-01097-6.

Sugita, Y. a Tani, J. (2005). Learning semantic combinatoriality from the interaction between linguistic a behavioral processes. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 13:33–52. ISSN 1059-7123.

Richard, S., Sutton a Andrew, G. Barto (1988). *Reinforcement Learning: An Introduction (Adaptive Computation a Machine Learning)*. The MIT Press. ISBN 0262193981.

Tani, J. (2003). Learning to generate articulated behavior through the bottom-up and the top-down interaction processes. *Neural Netw.*, 16:11–23. ISSN 0893-6080.

Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L. , a Nori, F. (2008). An opensource simulator for cognitive robotics research: The prototype of the icub humanoid robot simulator. In *In Proceedings of IEEE Workshop on Performance Metrics for Intelligent Systems (PerMIS08)*.

Tsagarakis, N. G., Sinclair, M. D., Becchi, F., Metta, G., Sandini, G. a Caldwell, D. G. (2006). Lower body design of the icub a human-baby like crawling robot. *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, str. 450–455. 2006 6th IEEE-RAS International Conference on Humanoid Robots 4-6 December 2006, Genoa, Italy ISBN: 142440200X.

Tsakarakis, N.G., Metta, G., Sandini, G., Vernon, D., Beira, R., Becchi, F., Righetti, L., Santos-Victor, J., Ijspeert, A.J., Carrozza, M.C. a Caldwell, D.G. (2007). iCub - The Design a Realization of an Open Humanoid Platform for Cognitive a Neuroscience Research. *Journal of Advanced Robotics, Special Issue on Robotic platforms for Research in Neuroscience*, 21(10):1151–1175.

van Hasselt, H. a Wiering, M. A (2007). *Reinforcement Learning in Continuous Action Spaces*, str. 272–279. IEEE Press.

Yarp (2011). Yarp - yet another robot platform. URL `http://eris.liralab.it/yarp/`.

Dodatok A

Príloha – CD médium

Na priloženom CD je uložený text tejto diplomovej práce v elektronickej podobe. Okrem toho sú na ňom uložené zdrojové kódy použité pri implementácii a vyhodnocovaní neurálneho modelu.