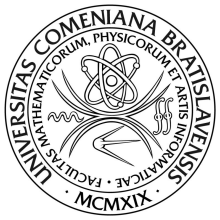


Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky
Katedra Aplikovanej Informatiky

Modelovanie uchopovania objektov pomocou neurónových sietí v robotickom simulátore iCub

Diplomová práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: doc. Ing. Igor Farkaš, PhD.

Bratislava, 2012 Bc. Lukáš Zdechovan

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Lukáš Zdechovan
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Modelovanie uchopovania objektov pomocou neurónových sietí v robotickom simulátore iCub.

Cieľ: Navrhnite a implementujte neurálny model umožňujúci kvázi prirodzene uchopovať objekty v prostredí robotického simulátora iCub (v Linuxe) s využitím vizuálnej spätnej väzby. Modul implementujte v jazyku C/C++ a otestujte ho na rôznych pevných objektoch uchopiteľných jednou rukou.

Literatúra: http://eris.liralab.it/iCub/main/dox/html/index.html
ďalšia literatúra bude dodaná

Poznámka: pasívna znalosť angličtiny, absolvovaný predmet Neurónové siete, relatívna samostatnosť, ochota pracovať priebežne

Vedúci: doc. Ing. Igor Farkaš, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky

Dátum zadania: 01.10.2010

Dátum schválenia: 03.11.2010

doc. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

študent

Vedúci

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Svojmu vedúcemu doc. Ing. Igorovi Farkašovi, PhD. ďakujem za jeho čas, početné konzultácie a cenné rady počas tvorby mojej diplomovej práce. Taktiež ďakujem ľuďom vo svojom okolí za neustálu podporu.

Abstrakt

V našej práci riešime jednu zo základných úloh kognitívnej robotiky, a to naučiť robota uchopovať objekty. Keďže cieľom kognitívnej robotiky je vytvárať biologicky inšpirované riešenia, je náš model založený na neurónových sieťach a učení s posilňovaním v spojitom priestore stavov a akcií. Pracujeme so simulátorom humanoidného robota iCub, ktorý patrí k najprecíznejšie navrhnutým robotom súčasnosti. Uchopovanie modelujeme ako súbeh dvoch samostatných akcií a to siahanie rukou na ľubovoľnú pozíciu v pracovnom priestore a následne samotné uchopenie objektu zapojením viacerých stupňov voľnosti na zápästí a prstoch. Prezentujeme architektúry použitých neurónových sietí ako aj vizualizáciu výsledkov použitia nášho modelu.

Kľúčové slová: uchopovanie objektov, kognitívna robotika, humanoid, učenie s posilňovaním, CACLA

Abstract

In our thesis we solve one of the fundamental tasks in cognitive robotics – object grasping. Since one of the goals of cognitive robotics is to create biologically plausible solutions, our model is based on neural networks and reinforcement learning in continuous state and action spaces. We work with the simulator of humanoid robot iCub, which belongs to the most precisely designed robots at present. We model object grasping as a concurrent run of two independent actions: reaching to an arbitrary position in the workspace and object grasping with the use of many degrees of freedom at wrist and fingers. We present an architecture of used neural networks and the visualization of our model results as well.

Keywords: object grasping, cognitive robotics, humanoid, reinforcement learning, CACLA (Continuous Actor-Critic Learning Automaton)

Predhovor

Cieľom našej diplomovej práce je navrhnúť model založený na neurónových sieťach pre kvázi prirodzené uchopovanie objektov v simulátore humanoidného robota iCub.

V práci analyzujeme existujúce modely uchopovania objektov a inšpirujeme sa nimi neskôr pri návrhu vlastného modelu. Popisujeme použité metódy z oblasti umelej inteligencie, ktoré sú základom pre pochopenie návrhu a implementácie nášho modelu. Obzvlášť sme sa venovali učeniu s posilňovaním v spojitom priestore stavov a akcií (CACLA) a rôznym modifikáciám tohto algoritmu.

Podrobne opisujeme návrh nášho modelu pre siahanie v priestore ako aj modelu pre uchopovanie objektov. Následne prezentujeme spôsob implementácie modelov v aplikácii pre simulátor robota iCub.

Na záver analyzujeme výsledky učenia modelov v mnohých experimentoch, kde porovnávame rôzne nastavenia parametrov a taktiež rôzne modifikácie použitého učiaceho algoritmu.

Obsah

Zoznam obrázkov

Úvod

Humanoidným robotom sa v súčasnej robotike začína pripisovať veľký význam a elektrotechnickí konštruktéri po celom svete ich vedia zostrojiť veľmi precízne s dôrazom na čo najväčšiu podobu s ľuďmi. Takýto robot je však po skonštruovaní stále len telo bez duše. Priemyselne využívané roboty sú zväčša dopredu naprogramované na konkrétny druh činnosti, ktorú vykonávajú s nesmiernou presnosťou a rýchlosťou.

Cieľom našej diplomovej práce je navrhnúť a implementovať model pre uchopovanie objektov v simulátore humanoidného robota iCub a otestovať ho na rôznych pevných objektoch.

Pokročilé techniky umelej inteligencie umožňujú robotom, aby sami získali rôzne druhy zručností. V spojení s kognitívnou vedou dospejeme k oblasti kognitívnej robotiky, ktorej cieľom je vytvárať biologicky inšpirované modely učenia pre humanoidných robotov (Asada et al., 2009). V ideálnom prípade takýto robot nielen vyzerá ako človek, ale sa ako človek aj učí.

V našej práci sa čo najviac držíme tohto princípu, a preto sme si naštudovali články o existujúcich modeloch uchopovania založených predovšetkým na kognitívnom výskume opíc makaka[1]. Mnohé z týchto modelov sú však trénované učením na predpripravených príkladoch (tzv. učenie s učiteľom), ktoré v prípade uchopovania nie je tak biologicky vierohodné ako učenie s posilňovaním, kde agent získa určitú znalosť vlastnou interakciou s prostredím.

Roboty pracujú v spojitom priestore, preto sme náš model postavili na učení CACLA (Continuous Actor-Critic Learning Automaton), ktoré využíva neurónové siete v postavení aktéra a krtitika pre vykonávanie a ohodnocovanie akcií v spojitom priestore a čase.

Humanoidný robot iCub je vyvýjaný európskym konzorciom RobotCub a z pohľadu konštrukcie je pre nás najdôležitejší fakt, že má až 53 stupňov voľnosti. Na jedinej ruke sa pritom nachádza 16 stupňov voľnosti čo robí úlohu uchopovania náročnejšou, ale

[1]Rod primátov často používaný na laboratórne a výskumné účely.

na druhej strane aj oveľa zaujímavejšou. Pre výskumné tímy je voľne dostupný aj simulátor, ktorý by mal byť vernou kópiou skutočného iCuba. Práve tento simulátor využívame aj my v našej záverečnej práci.

Na záver analyzujeme výsledky z vykonaných experimentov a prezentujeme rady k dosiahnutiu čo najvyššej úspešnosti modelu.

Kapitola 1

Východiská

1.1 Modely uchopovania objektov

Uchopovanie objektov je veľmi dôležitým spôsobom manipulácie s našim okolím a je základným kameňom pre vykonávanie rôznych úloh. V oblasti kognitívnej vedy a kognitívnej robotiky sa skúma proces učenia sa siahať v priestore efektorom (angl. reaching) a učenie sa uchopovať rôzne druhy objektov. Naša práca vychádza z modelov opierajúcich sa o využitie vizuálnej informácie a proprioceptívnej informácie agenta. V posledných rokoch sa uskutočnilo niekoľko výskumov v tejto oblasti a najdôležitejšie z nich zhrnieme v tejto kapitole.

Existujú dve základné teórie ovládania ramena a ruky pre správne uchopenie objektu (Kawato a Oztop, 2009). Prvá teória je podľa nášho názoru prirodzenejšia a predpokladá nezávislé, avšak koordinované ovládanie ramena a ruky tak, aby agent úspešne uchopil objekt. Proces uchopovania je tak rozdelený na 2 časti: siahanie na pozíciu v priestore s pripravením tvaru ruky (angl. reaching and hand preshaping) a samotné uchopenie objektu (angl. grasping). Druhá, alternatívna, teória predpokladá, že rameno a ruka sa ako 1 končatina ovládajú pomocou jediného ovládacieho mechanizmu. Z pohľadu strojového učenia je možnosť učenia sa uchopovania jedného ovládacieho mechanizmu omnoho náročnejšia v porovnaní s dvoma samostatnými mechanizmami. Učenie ovládačov siahania a uchopovania a ich vzájomná koordinácia pre úspešné vykonanie úlohy je navyše viac biologicky plauzibilné (Kawato a Samejima, 2007).

1.1.1 FARS model

Model FARS (Fagg-Arbib-Sakata-Rizzolatii model) popisuje proces uchopovania objektov z neurofyziologického pohľadu a vysvetľuje aké oblasti ľudského mozgu spolupracujú pri úspešnou vykonaní uchopenia (Fagg a Arbib, 1998). Neurofyziologické dáta tímu výskumníkov stojacich za týmto modelom naznačujú, že na správnom vykonaní a koordinovaní pohybových aktivít sa podieľa temenná mozgová kôra (parietal cortex).

Výsledky výskumu ukazujú, že v prednej medzitemennej oblasti (anterior intraparietal area, AIP) opíc makaka sa extrahujú črty trojrozmerného pozorovaného objektu, z ktorých sa odvodia afordancie pre možné uchopenia objektu. Táto oblasť je následne silne prepojená s oblasťami F4 a F5 dolnej premotorickej kôry (inferior premotor cortex). V oblasti F4 mozgu opice makaka sa z týchto afordancií naplánuje trajektória a cieľová pozícia ramena a v oblasti F5 sa určí najideálnejší typ úchopu (silový, precízny, bočný a pod.). Oblasti F4 a F5 ďalej riadia oblasť F1 primárnej motorickej mozgovej kôry (primary motor cortex), ktorá vysiela konkrétne signály do svalov ramena a ruky. FARS model je znázornený na obrázku 1.1.

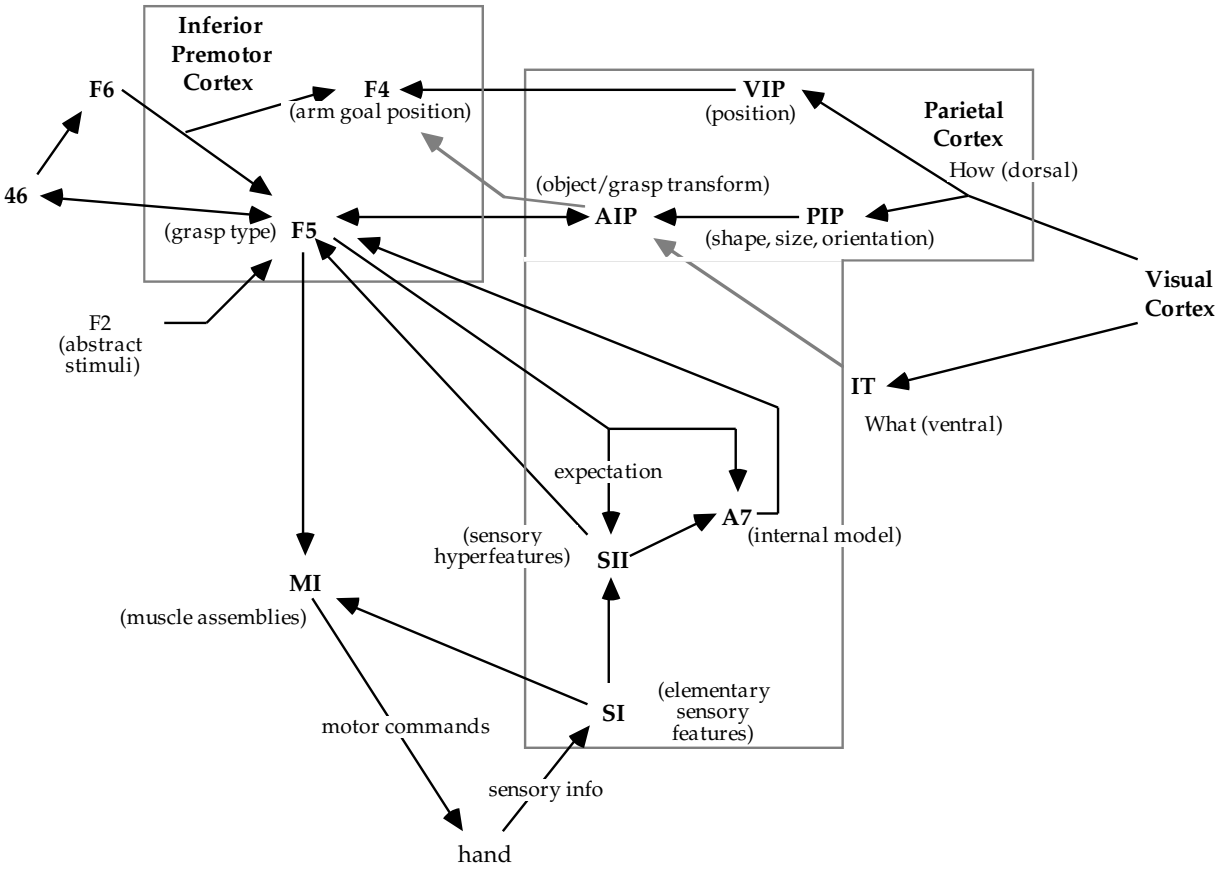
Hlavný dôraz modelu FARS bol na správne využitie afordancií, ktoré sa v AIP extrahujú zo spracovanej vizuálnej informácie z vizuálnej kôry (visual cortex) pre naučenie sa určiť správny typ uchopenia v oblasti F5 pre rôzne konkrétne objekty. Schéma funkcie AIP je znázornená na obrázku 1.2.

Afordancie agentovi udávajú akcie, ktoré je možné s objektom vykonať a ten by mal následne vybrať tú, ktorá najviac súvisí s úspešným naplnením jeho cieľa.

1.1.2 MNS – model založený na zrkadliacich neurónoch

Model MNS I sa snažil vysvetliť učenie rozpoznávania akcií pomocou zrkadliacich neurónov (mirror neurons) v mozgu opíc makaka. Zrkadliace neúrony sa u makaka nachádzajú v oblasti premotorickej kôry a aktivujú sa počas vykonávania akcie ako aj pri pozorovaní iného subjektu (človek, makak) vykonávajúceho akciu. Učenie kanonických neurónov, ktoré ovládajú samotné vykonávanie akcie makaka je podľa tvorcov modelu MNS I podmienené aj učením zrkadliacich neurónov. To umožňuje makakovi učiť sa rôzne akcie pozorovaním iných podobných jedincov. V modeli MNS I sa ďalej bližšie venovali jedinej akcii – uchopovaniu objektu (Oztop a Arbib, 2002).

Rozšírený model MNS II využíva biologicky prijateľnejší mechanizmus učenia pomocou rekurentných neurónových sietí metódou spätného šírenia chyby v čase (angl.

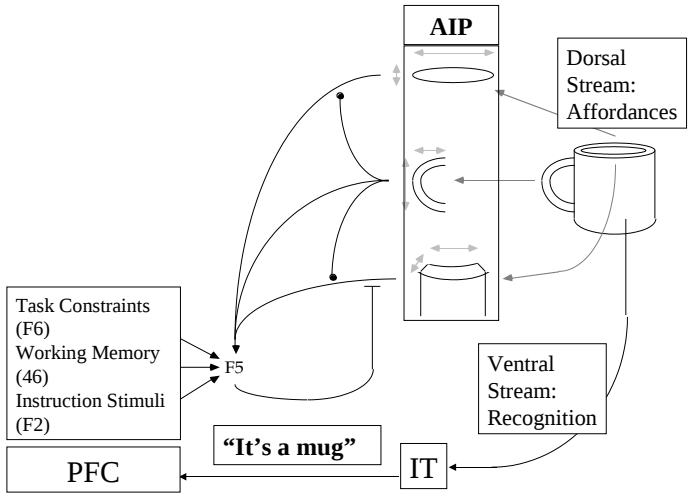


Obr. 1.1: FARS model (Fagg a Arbib, 1998)

error back-propagation through time - BPTT). Takto sa simuluje pracovná pamäť a umožňuje správnu aktiváciu zrkadliacich neurónov aj pri dočasnom skrytí objektu. Zrkadliace neuróny boli rozšírené o zvukovú zložku, takže učenie sa zintenzívni s použitím zvukových signálov (Bonaiuto et al., 2007).

Základom architektúry modelu je rekurentná neurónová sieť so 7-rozmerným vstupným vektorom (reprezentujúcim extrahované informácie o stave ruky vo vzťahu k objektu), vrstvou skrytých neurónov a 3-rozmerným výstupným vektorom (reprezentujúcim zrkadliace neuróny), z ktorého sa pomocou lokalistického[1] (one-hot) kódovania určuje cieľový typ uchopenia. Pre našu prácu bol veľkou inšpiráciou práve vstupný vektor, ktorý reprezentuje niekoľko geometrických vzťahov v rámci ruky a medzi rukou a cieľovým objektom. Stav ruky odráža vzdialenosť ruky od objektu, rýchlosť pohybu ruky, vzájomnú polohu palca, ukazováka a prostredníka a natočenia ruky vzhľadom na os uchopenia (Obrázok 1.3).

[1]Diskrétne kódovanie výberu prvku z N-prvkovej množiny pomocou N neurónov, z ktorých je aktívny vždy len jediný.



Obr. 1.2: Extrakcia afordancií v oblasti AIP (Fagg a Arbib, 1998)

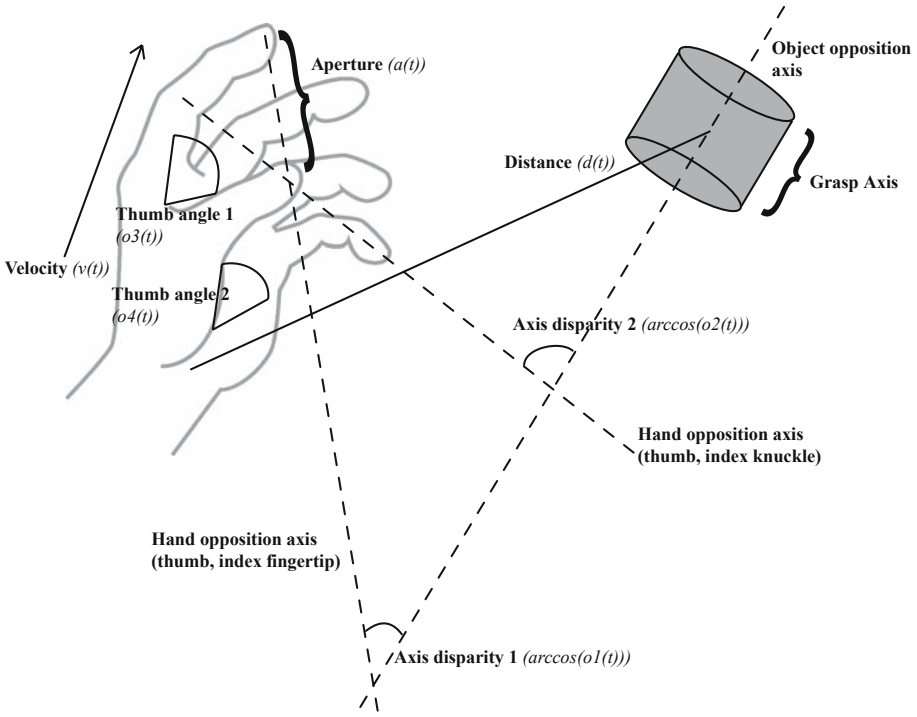
Autor modelu využil Jordanovskú rekurentnú architekúru s 5 kontextovými neurónmi, ktoré boli prepojené so skrytou vrstvou. Výstup skrytej vrstvy teda prispieval do pracovnej pamäte, ktorej výstup vchádza opäť do skrytej vrstvy. Súčasťou modelu je aj druhá rekurentná neurónová sieť špecializovaná na spracovanie zvukového signálu, ktorej výstup vchádza ako vstup do 3 výstupných (zrkadliacich) neurónov hlavnej siete.

Sieť je trénovaná štýlom učenia s učiteľom, kde sa najskôr pripraví množina trénovacích dát, v prípade MNS II to boli generované trajektórie pohybu ruky k 3 rôznym objektom (guľa, disk a kocka) rôznej veľkosti. Následne sa sieť učila metódou BPTT, pokým nedokázala správne identifikovať 95% uchopení. Cieľom tohto modelu teda nebolo generovať trajektórie pre úspešné dosiahnutie cieľovej pozície a uchopenie objektu, ale len určenie správneho typu uchopenia pre konkrétny tvar a veľkosť objektu.

1.1.3 Výpočtový model učenia sa detí uchopovať objekty

Porozumenie procesu učenia sa uchopovať objekty u živých bytostí (opice, ľudia) nám pomôže pri tvorbe neurálnych modelov a voľbe vhodnej metódy strojového učenia pre dosiahnutie čo najvyššej biologickej vierohodnosti. Vývin schopnosti uchopovania u detí skúmajú Oztop a Arbib (2004) pomocou výpočtového modelu ILGM (infant learning to grasp, a computation model).

Už novorodenci preukazujú neohrabanú schopnosť siahania na cieľovú pozíciu ramenom, ktorá sa postupne po 4 až 5 mesiacoch vyvinie do presnejšie ovládaného siahania a jednoduchého uchopovania objektov. Schopnosti blízko úrovne dospelého človeka získa dieťa už po 9 mesiacoch s výnimkou presného uchopovania malinkých objektov (angl.



Obr. 1.3: Stav ruky $(a(t), o1(t), o2(t), o3(t), o4(t), d(t), v(t))$ (Oztop a Arbib, 2002)

precision grasping), ktoré sa doladí po 12 až 18 mesiacoch od narodenia dieťaťa. Medzi 9. až 13. mesiacom sa pri siahaní na objekt zároveň pripravuje aj tvar a natočenie ruky podľa veľkosti, typu a orientácie uchopovaného objektu.

V modeli ILGM sa pri plánovaní uchopenia vypočítavajú 3 zložky (*p, r, v*): *cieľová pozícia dlane* (p), *natočenie zápästia* (r) *a zoznam prstov pre obklopenie objektu* (v). Mechanizmus, ktorý vykonáva samotnú akciu uchopovania najskôr presunie ruku tak, aby sa dlaň nachádzala v pozícii p a súčasne otáča zápästím po cieľový uhol r. Následne sa dlaň posúva smerom k ťažisku objektu. Po kontakte dlane s objektom prsty špecifikované vo v obklopia daný objekt (simulácia palmárneho reflexu u novorodencov). V prípade, že zovretie dlane viedlo k úspešnému uchopeniu objektu, výpočtový modul vráti pozitívny stimul (odmena) a váhy spojení medzi modulmi sa posilnia tak, aby za rovnakých okolností bol vygenerovaný práve vykonaný plán (*p, r, v*). Naopak, v prípade neúspešného úchopu sa váhy upravia tak, aby sa v budúcnosti naplánovali iné parametre.

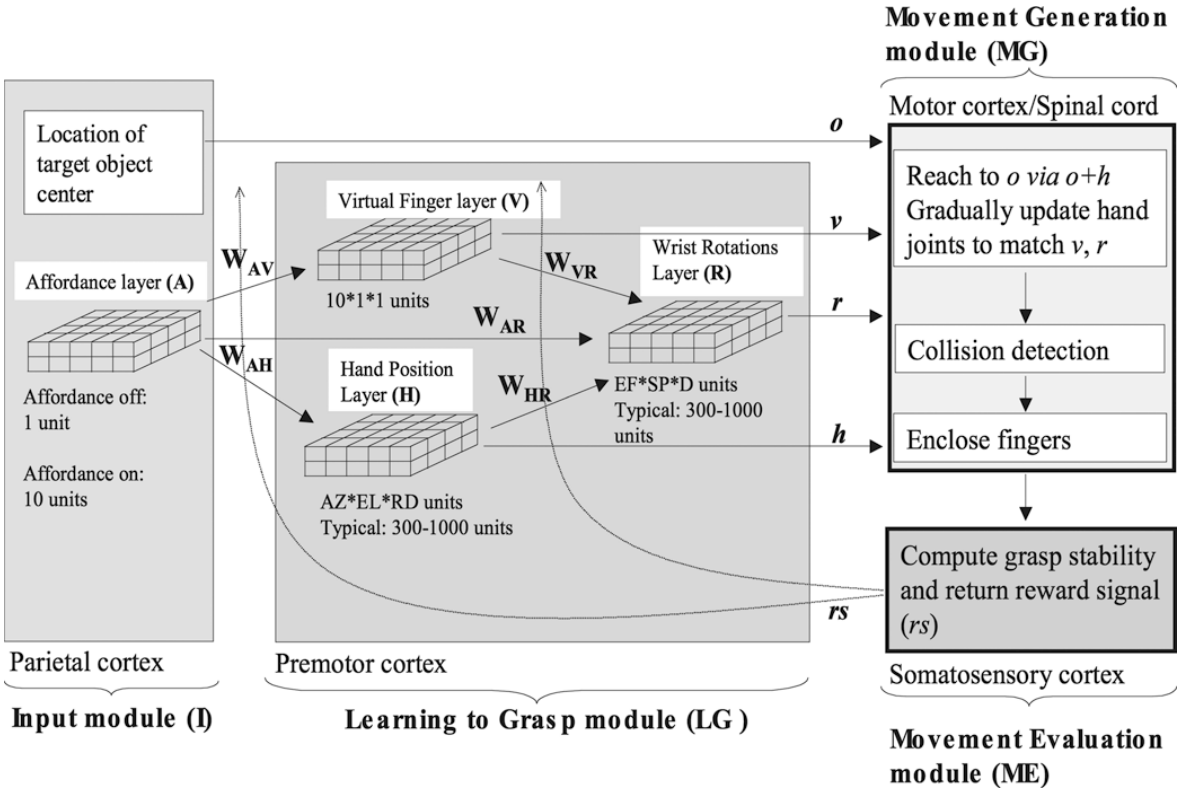
Architektúra modelu vychádza taktiež z neurofyziologického pohľadu na uchopovanie u opíc makaka a teda kooperáciu temennej, premotorickej, motorickej a somatosenzorickej mozgovej kôry. Každej oblasti prislúcha jeden modul:

- temennej kôre vstupný modul (I) kódujúci cieľovú pozíciu objektu s vrstvou

neurónov reprezentujúcich afordancie

- premotorickej kôre modul učenia sa uchopovania (LG) s 3 vrstvami neurónov pre plánovanie úchopu, ich výstupom je (p, r, v)
- motorickej kôre modul (MG), ktorá zabezpečuje posun ruky do cieľovej pozície p, otáčanie zápästia na uhol r, zovieranie prstov v a detekciu kolízie
- somatosenzorickej kôre modul (ME) pre vyhodnocovanie stability úchopu a vysielanie spätnej väzby (odmena)

V najjednoduchších experimentoch bola vo vrstve afordancií zakódovaná len informácia o existencii objektu. Neskôr v pokročilejších experimentoch sa do afordancií populačným kódovaním doplnili informácie o orientácii a pozícii cieľového objektu. Viac o populačnom kódovaní pojednávame v nasledujúcej podkapitole.



Obr. 1.4: Architektúra modelu ILGM (Oztop a Arbib, 2004)

Model ILGM dokázal počas učenia správne odhadnúť úchopy špecifické pre rôzne konkrétne objekty. Simulácie ďalej dokázali, že jednoduché akcie pre siahanie a uchopovanie spojené s vyhodnocovaním hmatovej (haptickej) spätnej väzby postačujú na naučenie sa rôznych štýlov uchopenia. Experiment potvrdil, že tzv. silný úchop (angl.

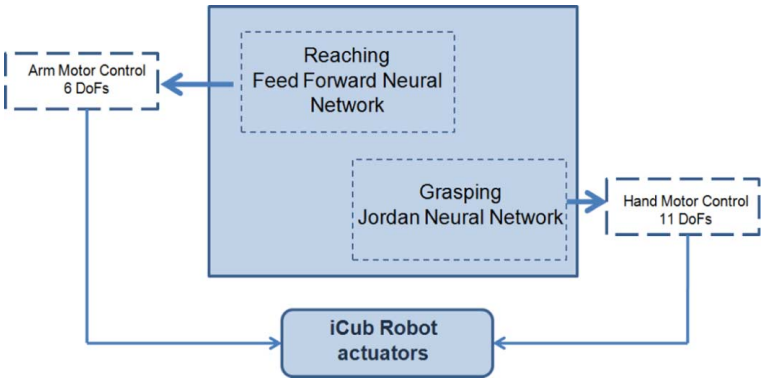
power grasp) je dominantným typom uchopenia v počiatočnej fáze učenia, a to aj pre malé objekty. Až v neskorej fáze učenia sa pre malé objekty stal dominantným presný úchop (angl. precision grasp).

1.1.4 Integrácia reči a akcií v humanoidných robotoch

Robotika ako taká je fenomén 20. storočia a úspešne sa v praxi osvedčili najmä priemyselné roboty, ktoré dokážu veľmi kvalitne, lacno a rýchlo vykonávať rôzne mechanické činnosti. Tieto roboty však boli dopredu naprogramované na konkrétne činnosti. Strojové učenie sa v praxi v robotoch používalo len veľmi zriedka a keď tak len v jednoduchej forme (napr. robotický vysávač). V poslednom desaťročí sa úspešne začína rozvíjať odvetvie kognitívnej humanoidnej robotiky, ktorá sa podľa očakávaní, stane fenoménom 21. storočia.

Myšlienkou kognitívnej robotiky je vytvoriť humanoidé roboty, ktoré budú schopné interagovať s prostredím podobne ako človek, pričom toto správanie nemajú predprogramované, ale dokážu sa ho naučiť. Dôležitým prvkom učenia je schopnosť komunikovať (napr. vhodne interpretovať príkaz na vykonanie úlohy), a preto vznikajú neurálne modely za účelom učenia sa jazyka (za cieľom porozumenia ľudskej reči).

Jednou z najnovších prác z tejto oblasti je integrácia reči a akcií v modeli pre siahanie a uchopovanie objektov vedcov Tikhanoffa, Cangelosiho a Mettu (2011). Vo svojom projekte využili taktiež simulátor robota iCub, ktorého sa s využitím neurónových sietí a strojového učenia snažili naučiť manipulovať s objektami podľa príkazu zadaného formou ľudskej reči (napr. reach blue ball, grasp blue ball) (Tikhanoff et al., 2011).

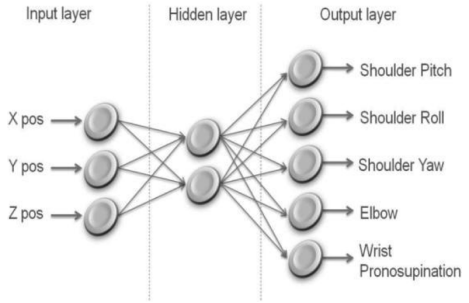


Obr. 1.5: Časť modelu pre siahanie a uchopovanie (Tikhanoff et al., 2011)

Tím talianskych vedcov navrhol architektúru kognitívneho systému pre humanoidného robota iCub, ktorý z najvrchnejšieho pohľadu tvoria modul pre spracovanie

obrazu z oboch kamier, modul pre spracovanie reči, neurónová sieť pre výber cieľa, modul pre siahanie v priestore a modul pre uchopovanie objektov. Jeho schému ilustruje obrázok 1.7. Výstupom obrazového modulu sú pozícia, farba, zaoblená aproximácia objektov a ich počet. Výstupom rečového modulu je zas signál, ktorý je pokynom pre vykonanie úlohy. Oba tieto moduly sú vstupom pre neurónovú sieť, ktorá určí cieľ v symbolovej reprezentácii a ten posunie na vstup modulu pre siahanie a modulu pre uchopovanie.

Siahanie na objekty modelujú autori klasickou doprednou dvoj-vrstvovou neurónovou sieťou, ktorá sa učí s učiteľom na 5000 príkladoch, ktoré boli získané náhodným pohybovaním ramena robota v priestore. Autori rozdelili príklady na trénovacie a testovacie v pomere 1:1. Sieť sa dokázala naučiť siahať na pozície v priestore po 50000 iteráciách (teda 20 epochách) s chybou 0.156 (root-mean-square error, str. 25) na testovacích príkladoch, pri rýchlosti učenia 0.05 a počte 10 skrytých neurónov. Architektúru siete opisuje obrázok 1.6.

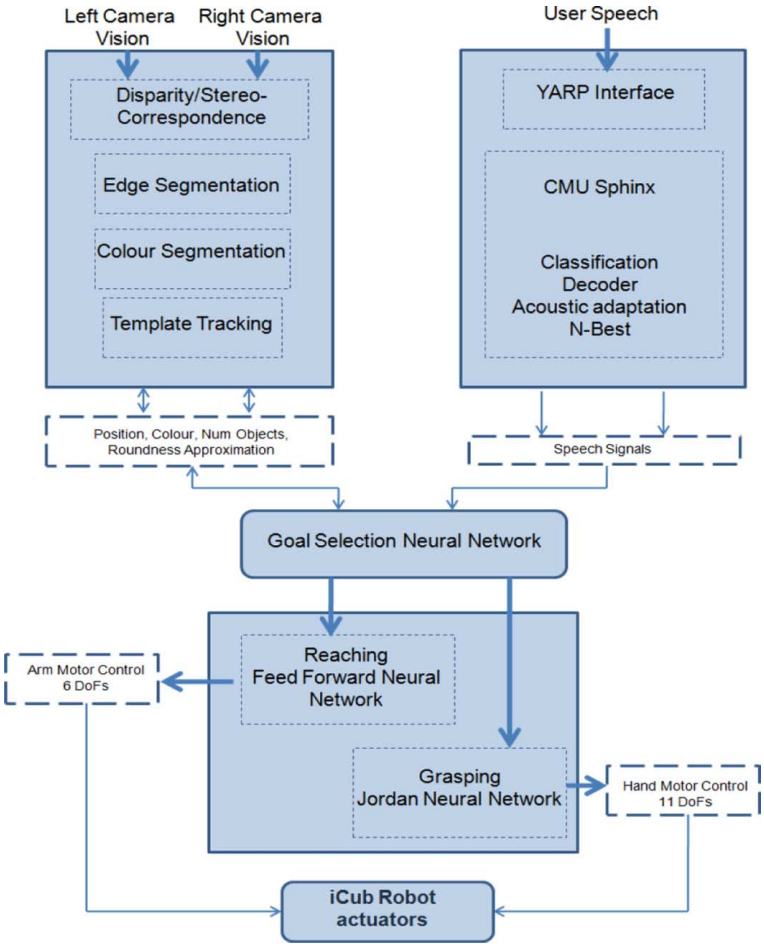


Obr. 1.6: Architektúra neurónovej siete pre siahanie (Tikhanoff et al., 2011)

Model pre uchopovanie objektov je založený na senzorovo orientovanom prístupe k uchopovaniu (angl. sensory-driven grasping approach). Pre tento model navrhli zložitejšiu neurónovú sieť s rekurentnými spojeniami na základe Jordanovej architektúry. Simuluje sa tým pracovná (krátkodobá) pamäť agenta. Učenie je zabezpečované online ARP (associative reward penalty) algoritmom. Ide teda o metódu asociatívneho učenia s posilňovaním.

Robot bol trénovaný na jedinom objekte (kocka) a následne po natrénovaní (keď sa už nezvyšovala dosahovaná odmena) testovali generalizačnú schopnosť siete na uchopovaní 3 iných objektov (malá kocka, guľa a komplexný objekt – plyšová hračka). Použité boli statické objekty (pevne „visiace“ v priestore), ktoré boli následne po uchopení zmenené za bežné (podliehajúce gravitácii) a podľa toho, ako dlho sa pozícia objektu počas 250 časových krokov nezmenila, bola vypočítaná odmena.

Po implementácii modulu pre spracovanie reči a neurónovej siete na voľbu cieľa bol iCub testovaný na úspešnosť vykonania rečou zadaných príkazov. Táto neurónová sieť bola taktiež učená metódou s učiteľom a dosiahla po 50000 trénovacích epizódach veľmi nízku testovaciu chybu RMSE na úrovni 0.037.



Obr. 1.7: Kognitívny model pre robota iCub (Tikhanoff et al., 2011)

Kapitola 2

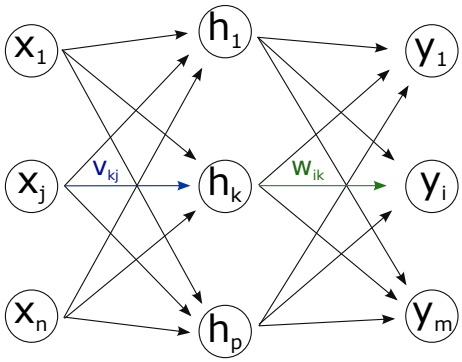
Použité metódy

Cieľom našej práce je umožniť humanoidnému robotovi naučiť sa uchopovať objekty. Vytvorili sme neurálny model pre siahanie na pozíciu efektorom a neurálny model pre samotné uchopovanie rôznych typov objektov. Učenie oboch modelov prebieha prostredníctvom algoritmu CACLA (Continuous Actor-Critic Learning Automaton) a využívajú sa pritom klasické viacvrstvové perceptróny (Hasselt, 2012). V tejto kapitole vysvetlíme teoretické pozadie a princípy fungovania týchto metód a algoritmov.

2.1 Viacvrstvové neurónové siete

Neurónová sieť je výpočtový model inšpirovaný biologickými princípmi vzájomného prepojenia a spolupráce viacerých neurónov, ktoré reagujú na vstupné podnety. Jednoduché neurónové siete majú jedinú vrstvu výstupných neurónov, ktoré sú váhovanými prepojeniami spojené so vstupmi. Tieto siete však dokážu riešiť len lineárne separovateľné problémy, čo môžeme považovať za veľký nedostatok.

Rozšírením siete o ďalšiu, skrytú vrstvu, získame model dvojvrstvovej neurónovej siete, ktorá je už schopná aproximovať ľubovoľnú spojitú funkciu (Baldi a Hornik, 1989). Tento typ siete sa trénuje známym trénovacím algoritmom spätného šírenia chyby (angl. error back propagation), za návrhom ktorého stoja McClelland a Rumelhart (1988). Obrázok 2.1 znázorňuje model dvojvrstvovej neurónovej siete so vstupným vektorom $\mathbf{x}$, skrytou vrstvou $\mathbf{h}$ a výstupnou vrstvou $\mathbf{y}$. Medzi vstupným vektorom a skrytou vrstvou sú prepojenia váhované maticou $\mathbf{V}$ a medzi skrytou vrstvou a výstupnou vrstvou prepojenia váhované maticou $\mathbf{W}$. Pomocou takejto siete dokážeme modelovať nelineárne zobrazenie vstupného vektora $\mathbf{x}$ na výstupný vektor $\mathbf{y}$.



Obr. 2.1: Dvojvrstvový perceptrón.

Výstupom každého neurónu je výsledok aktivačnej funkcie $f(net)$, ktorá ako vstup dostane lineárnu kombináciu vstupovného vektora a prislúchajúcich váh - *net*. Pre úplnosť pripájame vzorce pre výpočet aktivácií neurónov.

- $h_k = f(\sum_{j=1}^{n+1} v_{kj} x_j)$ ... aktivácia neurónov na skrytej vrstve
- $y_i = f(\sum_{k=1}^{q+1} w_{ik} h_k)$ ... aktivácia neurónov na výstupnej vrstve
- $x_{n+1} = h_{n+1} = -1$ ... bias u vstupného vektora a v skrytej vrstve

2.1.1 Algoritmus spätného šírenia chyby

Najčastejšie používaným algoritmom na učenie neurónových sietí je algoritmus spätného šírenia chyby (angl. error back propagation). Pre daný vstupný vektor sieť vypočíta výstupný vektor. Pokiaľ sa rovná požadovanému, nenastáva žiadne učenie. Ak je však na výstupe chyba (rozdiel medzi požadovaným a skutočným výstupom 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 váhy príslušného prepojenia.

Váhy medzi skrytou a výstupnou vrstvou sa upravujú podľa nasledujúceho vzorca:

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

Váhy medzi vstupom a skrytou vrstvou sa upravujú podľa vzorca:

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

Hodnoty δ_i vyjadrujú chybu i-teho neurónu výstupnej vrstvy a ako sme už spomínali, táto chyba je rozdielom požadovaného výstupu t_i a skutočného výstupu y_i prenásobeného deriváciou aktivačnej funkcie daného neurónu f_i'. Šírenie chyby vidieť z výpočtu chyby neurónu na skrytej vrstve δ_k, kde táto je sumou chýb neurónov na výstupnej vrstve prenásobených váhou prepojenia w_{ik} s daným neurónom. Parameter α ovplyvňuje rýchlosť učenia.

Najčastejšie používaná metóda učenia viacvrstvovej neurónovej siete je učenie s učiteľom. Ide o učenie, kde máme pripravenú sadu príkladov (dvojíc vstup - požadovaný výstup). Tieto príklady sa rozdelia na dve časti: trénovacie a testovacie dáta. Pomocou trénovacích príkladov sieť trénujeme, pričom po každom kroku aplikujeme algoritmus spätného šírenia chyby. Sieť sa po čase naučí aproximovať závislosti v dátach, čo vieme overiť na testovacích dátach.

Metóda učenia s učiteľom je však pre problém uchopovania objektov humanoidného robota nevhodná, nakoľko nie je biologicky vierohodná a navyše, nie je jednoduché pripraviť vhodné trénovacie dáta. Išlo by v podstate o také učenie, ako keby sme ťahali ruku robota požadovaným smerom. Preto používame iné pokročilé a biologicky vierohodnejšie metódy strojového učenia - učenie s posilňovaním a jeho špecializácie (angl. reinforcement learning).

2.1.2 Vyhodnocovanie chyby výstupu siete

Pre vyhodnotenie chyby výstupu siete sa štandardne používa chybová funkcia RMSE (Root-Mean-Square Error). Chyba naznačuje vzdialenosť aktuálneho výstupu od požadovaného výstupu pre daný vstup. Nech výstupom siete je vektor $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ a cieľovým výstupom vektor $\mathbf{d} = (t_1, t_2, \ldots, t_n)$. Chyba je potom daná nasledovným vzorcom:

$$\mathrm{RMSE(y,\ t)} = \sqrt{\mathrm{MSE(y,\ t)}} \tag{2.3}$$

kde MSE (Mean-Square Error) je stredná kvadratická chyba:

$$\mathrm{MSE(y,\ t)} = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2 \tag{2.4}$$

2.2 Učenie s posilňovaním

Učenie s posilňovaním je metóda strojového učenia, pri ktorej agent vykonáva akcie v prostredí, pričom mení svoj stav a po každom kroku dostáva spätnú väzbu z prostredia. Pre túto metódu sa používa aj názov učenie odmenou a trestom, keďže spätná

väzba z prostredia môže byť ako kladná, tak aj záporná a samotné učenie je založené na tom, že agent sa snaží maximalizovať získanú odmenu. Prehľadávanie priestoru je zabezpečené fázou explorácie, kedy agent vykonáva zašumené, či náhodné akcie a pozoruje výsledok svojho konania (Sutton a Barto, 1998).

Ide o biologicky vierohodnú metódu strojového učenia, nakoľko akcie agenta nie sú dopredu naprogramované, ani nie sú umelo vyučované pomocou trénovacích príkladov. Agent sám spoznáva svoje prostredie a to, ako ho dokáže ovplyvňovať svojimi akciami. Biologickú vierohodnosť potvrdzujú psychologické výskumy, v ktorých sa skúmalo učenie ľudí a zvierat pomocou odmeňovania.

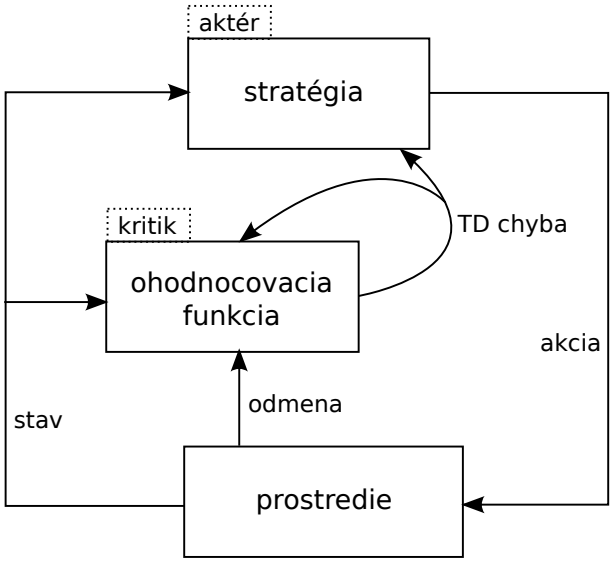
Základná schéma učenia s posilňovaním je založená na Markovovských rozhodovacích procesoch (angl. Markov Decision Processes - MDP) a operuje s diskrétnou množinou stavou a akcií. Úlohou agenta je nájsť takú stratégiu (angl. policy), ktorá mu v ľubovoľnom stave odporučí takú akciu, ktorá ho z dlhodobejšieho pohľadu dovedie k získaniu čo najvyššej odmeny (vo väčšine problémov určuje policy najbezpečnejšiu a najrýchlejšiu cestu stavovým priestorom k cieľu). Asi najdôležitejšou fázou implementácie učenia s posilňovaním pre konkrétny problém je navrhnutie odmeňovacej funkcie (angl. reward function). Tá musí byť navrhnutá tak, aby zahŕňala všetky požiadavky a obmedzenia problému.

2.2.1 Učenie s posilňovaním pomocou aktéra a kritika

V našom probléme pracujeme s agentom v spojitom priestore, v ktorom vie vykonávať taktiež spojité akcie. Jednou z možných špecializácií učenia s posilňovaním v spojitom priestore stavov a akcií je algoritmus CACLA (Continuous Actor-Critic Learning Automaton) (Hasselt, 2012). Tento algoritmus je postavený na architektúre zvanej aktér-kritik, kde aktér zabezpečuje generovanie akcie a_t v danom stave s_t a kritik ohodnocuje výber danej akcie v súvislosti so zmenou stavu na s_{t+1}, pričom berie do úvahy ako odmenu z prostredia r_{t+1}, tak aj odhad ohodnotenia pravdepodobných budúcich stavov.

Aktér a kritik

Aktér aj kritik sa modelujú ako funkčné aproximátory (dvojvrstvové neurónové siete so spojitou aktivačnou funkciou), kde vstup je aktuálny stav prostredia a výstupom je vygenerovaná akcia resp. ohodnotenie vykonanej akcie. Učenie sa upravuje pomocou 2 parametrov: diskontného faktora γ a exploračného faktora σ. Aktér spočiatku gene-



Obr. 2.2: Schéma CACLA učenia

ruje náhodné akcie kvôli náhodnej inicializácii váh siete, následne sú tieto akcie ešte zašumené (k akcii sa pripočíta náhodná zložka, ktorej veľkosť závisí od exploračného faktora) a až takúto zašumenú (explorovanú) akciu agent v prostredí vykoná. Hodnoty parametrov sa často určujú experimentálne, v okolí určitých zaužívaných „dobrých“ hodnôt. Niekedy je vhodné meniť hodnotu exploračného faktora v čase. Agent potom viac exploruje prostredie na začiatku učenia, neskôr sa už len málo odchyľuje od naučenej stratégie.

Učenie kritika prebieha v každom kroku t behu algorirmu. Kritik ohodnotí akciu vykonanú v stave s_t hodnotou $V(s_t)$. Prostredie vráti odmenu r_{t+1} za nový stav s_{t+1}, do ktorého sa agent dostal vykonaním akcie a_t a následne sa kritik učí tak, aby v budúcnosti stav s_t ohodnotil touto odmenou plus odhadom do búducnosti $\gamma V_t(s_{t+1})$. V prípade, že $\gamma = 0$ nehľadí agent vôbec na budúci vývoj a učí sa odhadovať vždy len odmenu dosiahnutú po vykonaní akcie aktérom. Inak sa do úvahy berú aj budúce odmeny $r_t + \gamma r_{t+1} + \gamma^2 rt + 2 + \gamma^3 r_{t+3} \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ (Doya, 2000).

Aktéra chceme učiť len vtedy, pokiaľ nás vykonaná explorovaná (zašumená) akcia a_t viedla k stavu s_{t+1} s lepším ohodnotením, než kritikove ohodnotenie stavu, do ktorého by sme sa dostali aktérom pôvodne vygenerovanou akciou ac_t. Teda pokiaľ $r_{t+1} + \gamma V_t(s_{t+1}) > V(s_t)$, aktérove váhy posilníme tak, aby v budúcnosti zvolil v stave s_t radšej akciu a_t.

Algoritmus 1 CACLA - Učenie s posilňovaním pomocou aktéra a kritika

1: $s_0 \leftarrow$ počiatočný stav
2: inicializuj váhy aktéra
3: inicializuj váhy kritika
4: **for** $t = 0, 1, 2 \ldots$ **do**
5: $ac_t \leftarrow A_t(s_t)$
6: $a_t \leftarrow$ exploruj ac_t
7: vykonaj akciu a_t a prejdi do stavu s_{t+1}
8: **if** $r_{t+1} + \gamma V_t(s_{t+1}) > V(s_t)$ **then**
9: aktualizuj váhy aktéra: $A_{t+1}(s_t) \leftarrow a_t$
10: **end if**
11: aktualizuj váhy kritika: $V_{t+1}(s_t) \leftarrow r_{t+1} + \gamma V_t(s_{t+1})$
12: **end for**

Urýchlenie učenia algoritmu CACLA

Keďže algoritmus využíva na ohodnocovanie svojich stavov viacvrstvovú neurónovú sieť, ktorá má iniciálne vygenerované náhodné váhy, a preto hodnoty vyprodukované dopredným prechodom sú spočiatku tiež náhodné. Pri použití grafického simulátora sa však musíme snažiť o čo najrýchlejšie učenie, preto sme za cieľom urýchlenia skúšali rôzne modifikácie algoritmu CACLA.

Ako sme už spomínali, spočiatku sú kvôli náhodnému vygenerovaniu váh v neurónových sieťach akcie aktéra chaotické a ohodnotenia stavov kritikom náhodné. Aby sme aktéra neučili zbytočne hlúpe akcie, môžeme ho prvých N krokov algoritmu učiť len na základe rozdielu odmeny v stave po vykonaní akcie a odmeny predchádzajúceho stavu. Aktéra budeme teda učiť podľa pravidla $r_{t+1} > r_t$.

Ďalšie urýchlenie algoritmu môžeme získať tak, že budeme následne ďalších M krokov učiť aktéra podľa aktuálneho ohodnotenia vygenerovanej a explorovanej akcie. Agent si mentálne predstaví akciu ac_t, ktorú vygeneroval aktér a určí stav s'_{t+1}, do ktorého by sa dostal vykonaním akcie ac_t. Následne porovná kritikove ohodnotenia tohto stavu a skutočného stavu, do ktorého sa dostal po vykonaní explorovanej (zašumenej) akcie. Aktér sa teda bude učiť podľa pravidla $V_t(s_{t+1}) > V_t(s'_{t+1})$.

V našich modeloch sme používali hodnoty $N = 400 \times L_e$ a $M = 1000 \times L_e$, kde L_e je dĺžka epizódy a epizódou myslíme pokus dosiahnuť daný cieľový stav na určitý počet krokov. Dĺžku epizódy sme obmedzili na 20 krokov (agent môže teda vykonať najviac 20 akcií na to, aby sa dostal do cieľového stavu).

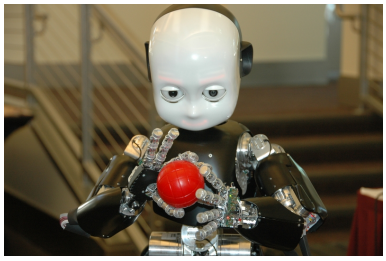
Algoritmus 2 Modifikovaná CACLA - urýchlenie učenia

$s_0 \leftarrow$ počiatočný stav
$t_{uc\_odmenou} \leftarrow$ # epizód, ktoré budeme aktéra učiť na
$p_{nahodna} \leftarrow$ pravdepodobnosť vygenerovania úplne náhodnej akcie
inicializuj váhy aktéra
inicializuj váhy kritika
for $t = 0, 1, 2 \ldots$ **do**
  $ac_t \leftarrow A_t(s_t)$
  if $rand() \leq p_{nahodna}$ **then**
    $a_t \leftarrow$ vygeneruj náhodnú akciu
  else
    $a_t \leftarrow$ exploruj ac_t
  end if
  vykonaj akciu a_t a prejdi do stavu s_{t+1}
  vypočítaj potenciálny stav s_{t+1} v prípade vykonania akcie ac_t
  $r_{t+1} \leftarrow$ odmena z prostredia
  if $(t \leq t_{uc\_odmenou}$ and $r_{t+1} > r_t)$ **then**
    aktualizuj váhy aktéra: $A_{t+1}(s_t) \leftarrow a_t$
  else if $(t > t_{uc\_odmenou}$ and $V_t(s_{t+1}) > V_t(s'_{t+1}))$ **then**
    aktualizuj váhy aktéra: $A_{t+1}(s_t) \leftarrow a_t$
  end if
  aktualizuj váhy kritika: $V_{t+1}(s_t) \leftarrow r_{t+1} + \gamma V_t(s_{t+1})$
end for

2.3 Simulátor humanoidného robota iCub

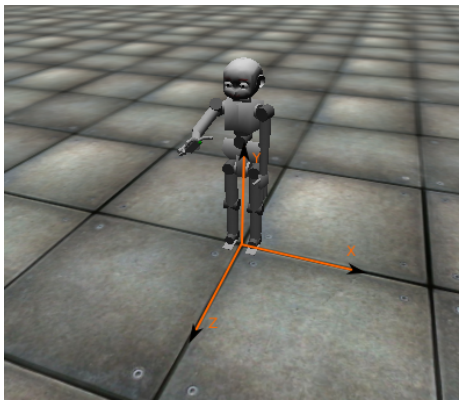
Už v názve našej práce je špecifikované, že je našim cieľom navrhnúť neurálny model pre uchopovanie objektov s využitím robotického simulátora iCub. Robot iCub je vyvijaný konzorciom RobotCub a ide podľa rozšíreného názoru o najlepšieho na akademické účely využívaného humanoidného robota v Európe. Avšak keďže nie každý výskumný tím si môže dovoliť takéhoto robota, zostrojili talianski vedci hodnoverný simulátor iCuba kompatibilný so skutočným robotom (Tikhanoff et al., 2008). Ďalšou motiváciou pre použitie simulátora je využitie v strojovom učení, kde je nutné, aby agent vykonával určitú úlohu často aj 1000 opakovaní, než sa dokáže naučiť požadované správanie. Pri pohľade na najznámejších humanoidov na svete si ľahko všimneme, že

iCub je jeden z mála robotov, ktorý sa skutočne veľmi podobá človeku. Na obrázku 2.3 je pohľad na iCuba a nižšie na obrázku 2.4 zas pohľad na simulátor. Na humanoidných robotov sa kladú vždy náročné požiadavky na množstvo stupňov voľnosti a početnosť senzorov. Robot iCub (reálny aj v simulátore) má až 53 stupňov voľnosti (na hlave, trupe, ramenách, rukách aj nohách) a rôzne druhy senzorov (kamera v oboch očiach, tlakové senzory na dlani a prstoch).



Obr. 2.3: Ukážka humanoidného robota iCub

Simulátor je postavený na fyzikálnej knižnici ODE a grafickej knižnici OpenGL a je vyvijaný ako softvér s otvoreným zdrojovým kódom. V súčasnosti je čoraz viac intenzívnejšie využívaný vo výskumných prácach v oblasti kognitívnej vedy a najmä kognitívnej robotiky. Pri našej práci sme postrehli drobné nedostatky týkajúce sa najmä spracovania fyziky v simulátore. Občas sa vyskytli nerealistické javy ako prilepenie objektu k dlani iCuba alebo prechádzanie ramenom cez vlastné telo. Naša práca si viažadala rozličné úpravy simulátora, ktoré sme popísali v kapitole Implementácia.



Obr. 2.4: Ukážka simulátora so znázorneným súradnicovým systémom

2.4 Neurálne kódovanie

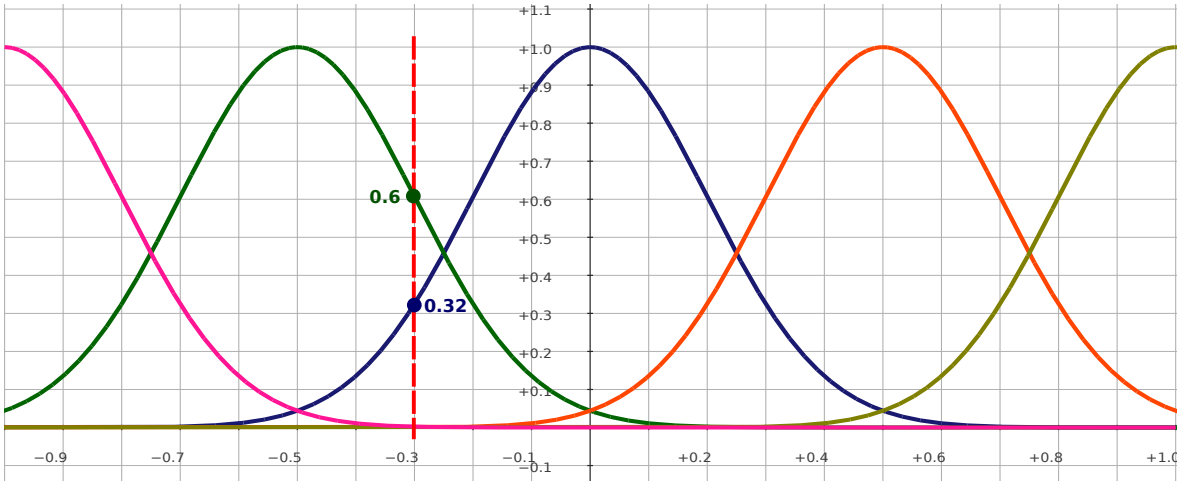
Pri kódovaní informácií v neurónových sieťach je užitočné poznať rozmanité spôsoby kódovania, ktoré boli pozorované v mozgu zvierat. Vedci z oblasti neurovedy popísali niekoľko možností kódovania, z ktorých sme sa rozhodli popísať tie, ktoré sú užitočné pre pochopenie článkov z teórie uchopovania objektov a pre návrh nášho neurálneho modelu.

2.4.1 Populačné kódovanie

Populačné kódovanie sa vyznačuje tým, že informáciu kóduje skupina neurónov, z ktorých každý svojou aktiváciou prispieva k formovaniu výsledného signálu. Výstup teda odráža akýsi vzor aktivít celej danej populácie neurónov (Pouget et al., 2000).

2.4.2 Hrubé (Coarse) kódovanie

Špeciálny typ populačného kódovania je hrubé kódovanie (angl. coarse coding), kde sa informácia kóduje vždy len úzkou skupinou neurónov. Napríklad na kódovanie reálneho čísla z intervalu $\langle -1, 1 \rangle$ vieme vytvoriť populáciu neurónov, ktoré na intervale rovnomerne rozmiestnime a nad každým umiestnime gaussovskú funkciu. Následne pre zakódovanie reálneho čísla x nájdeme nenulové prieniky s gaussiánmi, ktoré prechádzajú ponad x a prislúchajúce veľkosti y budú aktiváciou príslušných neurónov.
Obrázok 2.5 ilustruje kódovanie reálneho čísla 0.3 pomocou populácie 5 neurónov, ktorých aktivácia bude vektor $\langle 0.0, \mathbf{0.6}, \mathbf{0.32}, 0.0, 0.0 \rangle$.



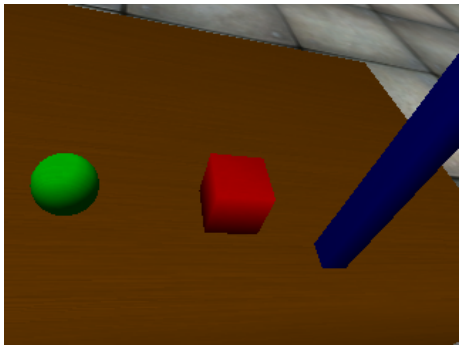
Obr. 2.5: Hrubé kódovanie reálneho čísla z intervalu $\langle -1, 1 \rangle$ pomocou 5 neurónov

2.5 Spracovanie obrazu s využitím knižnice OpenCV

Počítačové videnie a spracovanie obrazu sú samostatné vedné disciplíny, ktoré vyprodukovali stovky algoritmov, ktoré sa dnes úspešne využívajú v komerčnej praxi ako aj vo vede a výskume. V kognitívnej robotike sa kladie dôraz na to, aby čo najväčší podiel vizuálnych informácií bol získavaný spracovaním obrazu.

My sme si pre potreby spracovania obrazu zvolili známu knižnicu OpenCV, ktorá obsahuje mnohé algoritmy napr. pre prahovanie obrazu, vyhľadávanie kontúr a následne obrazových momentov kontúr, z ktorých sa dá určiť rozmer, pozícia a relatívna orientácia pozorovaného objektu.

Robot iCub ako aj simulátor má v očiach vstavané kamery, z ktorých sa priebežne získava trojkanálový obraz v rozmeroch 320×240 pixelov (viď obrázok 2.6). Následne z tohto obrazu s využitím niekoľkých algoritmov extrahujeme rozmer a orientáciu cieľového objektu, pretože aj tieto vizuálne informácie využívame v našom modeli pre uchopovanie.



Obr. 2.6: Obraz z pravej kamery robota

2.5.1 Použité OpenCV algoritmy

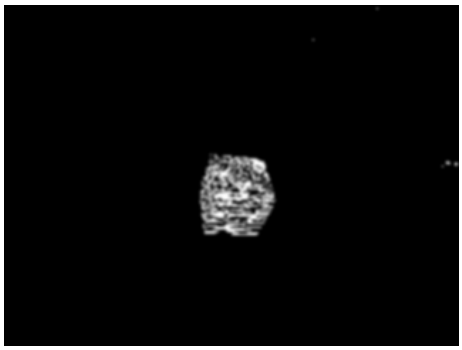
Thresholding (Prahovanie)

Na prahovanie existuje v OpenCV viacero funkcií. My sme využili funkciu **cvInRangeS**, ktorá zo vstupného obrazu prekreslí na výstupný obraz každý pixel čiernou, ak nepatrí do zadaného HSV[1] farebného intervalu. Pixle patriace do intervalu sa vyfarbia bielou a tým vznikne binárny čiernobiely obraz. Keďže v našom prípade, každý objekt na scéne ma jedinečnú farbu, vieme takto objekty podľa farby identifikovať.

[1]Farebný model HSV - Hue (odtieň), Saturation (sýtosť), Value (svetlosť)

Blur (Rozostrenie)

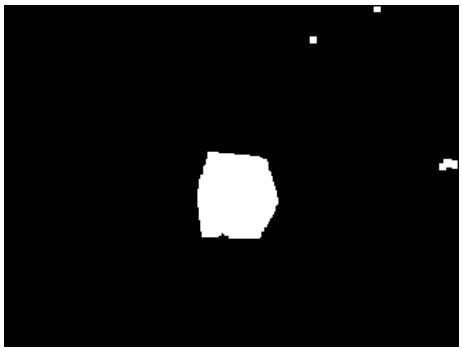
Rozostrenie na spracovaný obraz po prahovaní sa využíva ako medzistupeň pre jednoduchšie vyhľadávanie kontúr, keďže často vznikne po prahovaní na čiernej ploche biely tvar s veľkým počtom malých čiernych dier. Tie, ako môžeme vidieť na obrázku 2.7, sa dostatočne silným rozostrením zaplnia. Na rozostrenie využívame funkciu **cvSmooth** s gaussovským typom rozostrenia.



Obr. 2.7: Rozostrený obraz po prahovaní na odtiene červenej farby

Vyhľadávanie kontúr

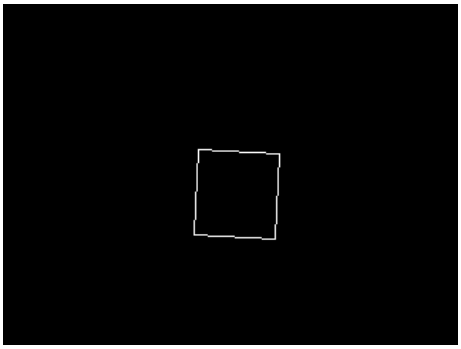
V jemne rozostrenom prahovanom obraze následne vyhľadávame kontúry identifikovaného objektu. Na tento účel sme využili funkciu **cvFindContours**, ktorá nájde pole sekvencií bodov (sekvencia bodov tvorí kontúru). Následne využívame funkcie **cvContourArea** a **cvArcLength**, aby sme pre každú nájdenú kontúru určili jej obsah resp. obvod. Tieto výstupy využívame na určenie rozmeru pozorovaného objektu. Obrázok 2.8 znázorňuje výsledok po hľadaní kontúr.



Obr. 2.8: Obraz po rozostrení a hľadaní kontúr

Obrazové momenty a ohraničenia

Orientáciu objektu sme skúšali odvodzovať pomocou dvoch rôznych metód. Jednou je vypočítanie obrazového momentu kontúry pomocou funkcie **cvMoments**, z ktorého zložiek je možné vypočítať uhol momentu. Táto metóda však neposkytovala také spoľahlivé určenie orientácie ako s využitím orientovaného ohraničujúceho obdĺžnika (cv::RotatedRect). Pomocou funkcie **cv::minAreaRect** určíme tento obdĺžnik (viď obrázok 2.9) a následne z pomeru výšky a šírky vieme odvodiť orientáciu objektu (či je bežný objekt alebo podlhovastý vertikálne/horizontálne umiestnený objekt).



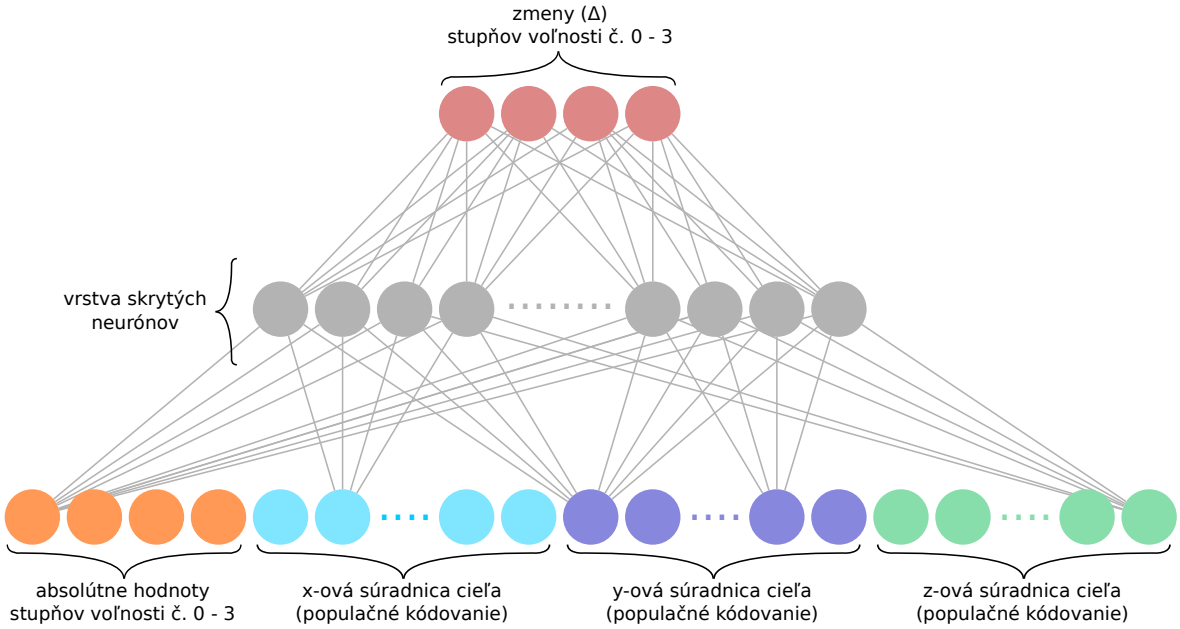
Obr. 2.9: Orientovaný ohraničujúci obdĺžnik

Kapitola 3

Návrh

Ako sme už spomenuli v kapitole Teoretické východiská, existujú dva prístupy k modelovaniu uchopovania objektov. My sme sa pri návrhu rozhodli stavať na prístupe, kde uvažujeme dva nezávislé no koordinované procesy približovania ruky k objektu (reaching) a uchopenia objektu (grasping). Navrhli sme teda dva samostatné modely založené na učení CACLA. V tejto kapitole popíšeme návrh neurálneho modelu pre siahanie v priestore a návrh neurálneho modelu pre uchopovanie.

3.1 Model pre siahanie v priestore



Obr. 3.1: Návrh neurálneho modelu pre siahanie

Pri návrhu neurálneho modelu predstavuje vstupná vrstva informácie o stave agenta, ktoré potrebuje pre úspešné splnenie cieľa (v našom prípade dosiahnutie určitej pozície rukou v priestore). My sme sa rozhodli poskytnúť agentovi na vstupe súradnice cieľovej pozície a propriocentrické informácie o stupňoch voľnosti, ktoré na výstupe motoricky ovláda (ovládanie kĺbov v ramene a lakti). Kvôli efektívnejšiemu učeniu neurónovej siete sme sa rozhodli všetky hodnoty preškálovať do intervalu $\langle -1, 1 \rangle$. Súradnice cieľovej pozície vyjadrujeme populačným kódovaním a to 9 neurónmi pre každú súradnicu.

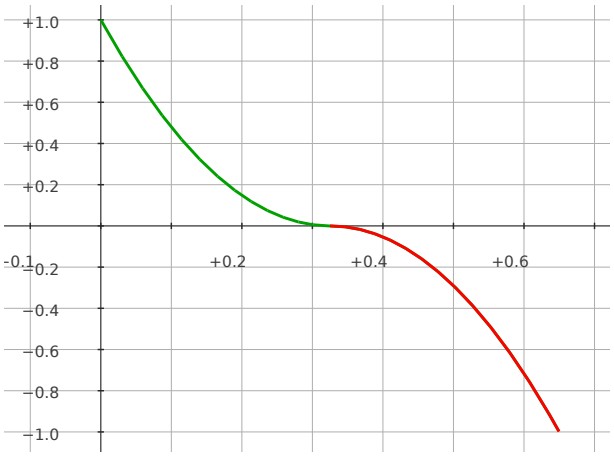
Počet neurónov v skrytej vrstve sme určovali najskôr experimentálne a neskôr sme si zostrojili program, ktorý na zozbieraných dátach zo simulátora testoval učenie siete s rôznym počtom neurónov na skrytej vrstve.

Pri návrhu architektúry sme sa spočiatku inšpirovali obdobným prístupom k siahaniu na 3 pevne umiestnené objekty v práci Farkaš et al. (2012).

Na výstupnú vrstvu sme umiestnili neuróny, ktoré posielajú motorické signály pre relatívnu zmenu príslušných stupňov voľnosti. Vektor na výstupe sa teda pripočíta k aktuálnym hodnotám stupňov voľnosti a tie následne slúžia ako nový vstup do siete.

Iniciálne veľkosti váh sme navrhli vygenerovať náhodne z intervalu $\langle -0.1, 0.1 \rangle$. Nízke hodnoty sme zvolili preto, aby váhy pri vysokom exploračnom faktore rýchlo nedivergovali.

3.1.1 Návrh funkcie odmeny



Obr. 3.2: Funkcia odmeny modelu pre siahanie

Pre správne učenie s posilňovaním je nutné vhodne navrhnúť funkciu odmeny, ktorou vieme agenta „motivovať“ k vykonávaniu akcií vedúcich k naplneniu stanoveného cieľa. Intuitívne je pre agenta najvýhodnejšia akcia, ktorá ho čo najviac priblíži k

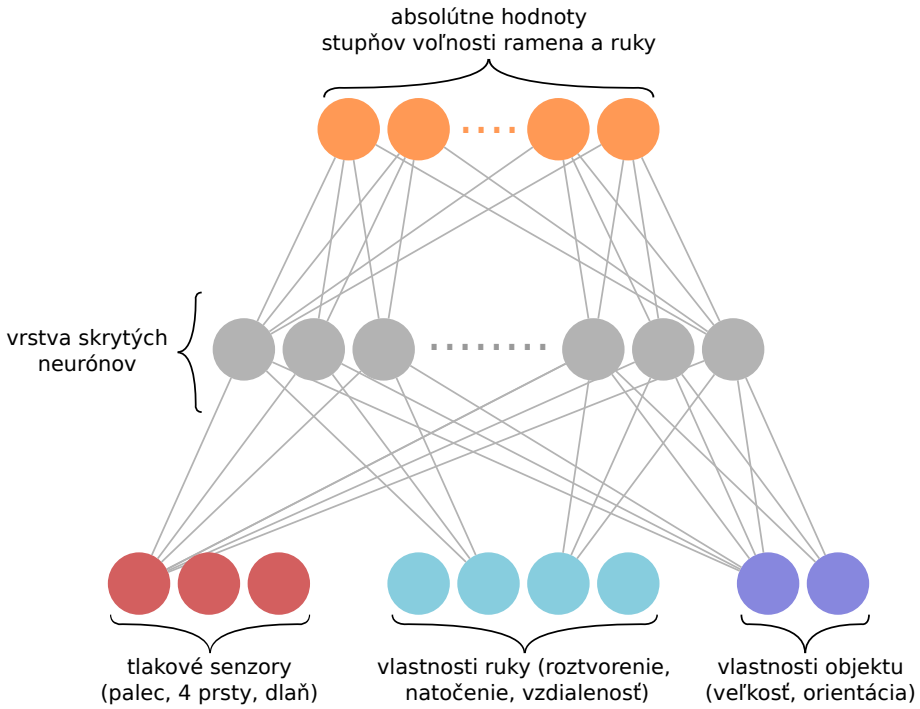
cieľovej pozícii. Navrhli sme teda jednoduchú funkciu závislú len od euklidovskej vzdialenosti ťažiska dlane od cieľovej pozície. Odmenu za vzdialenosť škálujeme do intervalu $\langle -1, 1 \rangle$ a umocňujeme na druhú so zachovaním znamienka. Tým dosiahneme efekt, že stavy blízko cieľa sú oveľa hodnotnejšie, než keby sme použili lineárnu funkciu.

Vzorec podľa ktorého počíta prostredie odmenu r po vykonaní akcie:

$$r(d) = r_l(d)^2 sgn(r_l(d)), \tag{3.1}$$

kde $r_l(d) = 1 - 2 * (d/d_{\mathrm{MAX}})$, $d_{\mathrm{MAX}} = 0.65$ je maximálna uvažovaná vzdialenosť a d je aktuálna vzdialenosť od cieľa.

3.2 Model pre uchopovanie objektov



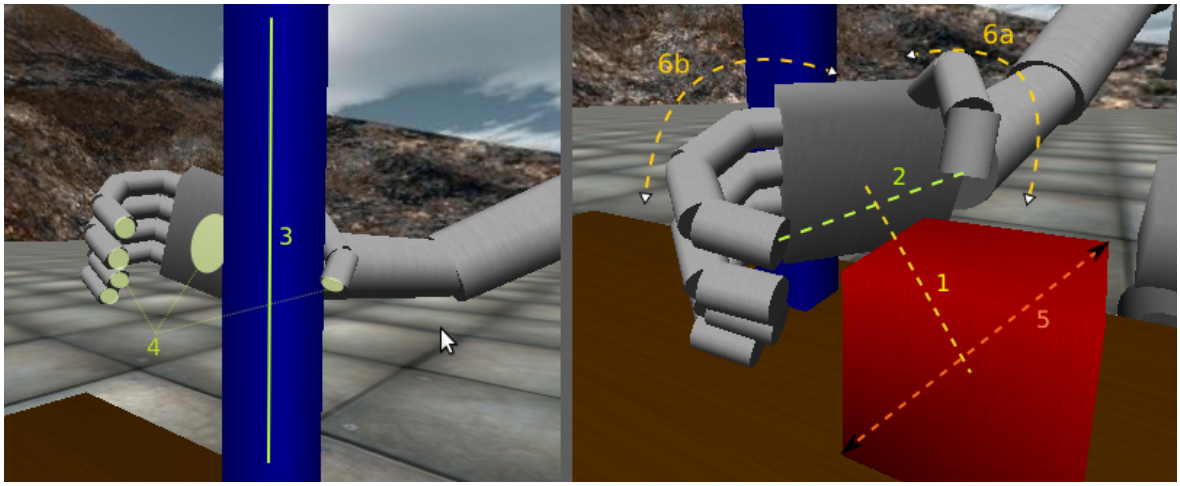
Obr. 3.3: Návrh neurálneho modelu pre Grasping

Stavový vektor pre model uchopovanie je zložitejší než pri siahaní v priestore. Je logické, aby agent zohľadňoval informácie z dotykových/tlakových senzorov, veľkosť a orientáciu objektu a vzdialenosť dlane od objektu. Ďalej si myslíme, že je vhodné, aby agent využíval pri uchopovaní aj informácie o natočení dlane v priestore a roztvorenia prstov dlane. Neurón $tp\alpha$ určuje uhol natočenia dlane v smere priamky z dlane cez vystretý palec. Neurón $pp\alpha$ zas určuje uhol natočenia dlane v smere priamky z dlane cez vystretý ukazovák. Vidíme teda, že stavový vektor obsahuje vizuálne informácie, haptické informácie a taktiež propriocentrické informácie (natočenia svojej dlane a roztvorenie prstov vnímame aj bez vizuálneho vstupu).

Počet neurónov v skrytej vrstve sme určovali rovnako ako u predchádzajúceho modelu pre siahanie.

Na výstupnú vrstvu sme umiestnili neuróny, ktoré posielajú motorické signály na nastavenie príslušných stupňov voľnosti ramena, dlane a prstov. Vektor na výstupe obsahuje absolútne cieľové hodnoty príslušných stupňov voľnosti.

Iniciálne veľkosti váh sme navrhli vygenerovať náhodne z intervalu $\langle -0.1, 0.1 \rangle$. Nízke hodnoty sme zvolili z rovnakého dôvodu ako pri predchádzajúcom modeli, aby váhy pri vysokom exploračnom faktore rýchlo nedivergovali.



Obr. 3.4: Stavové informácie, ktoré robot vníma z prostredia

Tabuľka 3.1: Prehľad informácii, ktoré vystupujú v stavovom vektore modelu pre uchopovanie objektu

	názov	popis
1	vzdialenosť od objektu	eukl. vzdialenosť ťažiska dlane od ťažiska objektu
2	roztvorenie ruky	eukl. vzdialenosť koncových článkov palca a ukazováka
3	orientácia objektu	horizontálna/vertikálna orientácia objektu
4	dotyk	informácie z tlakových senzorov
5	veľkosť objektu	odhad objemu uchopovaného objektu
6	natočenie dlane	uhly pozdĺžneho a bočného natočenia dlane

3.2.1 Návrh funkcie odmeny

Pre vyhodnotenie úspešnosti uchopenia je potrebné zvážiť viacero faktov. Agent musí pocítiť haptickú spätnú väzbu po uchopení objektu a rovnako musí priebežne vyhodnocovať vzdialenosť dlane (častí dlane) od cieľového objektu. Odmena by taktiež mala súvisieť s možnosťou udržať pri danom uchopení objekt vo vzduchu, nie vždy je totiž vysoká haptická odozva postačujúca.

Funkcia odmeny pozostáva z dvoch váhovaných hodnôt a to odmeny za vzdialenosť od ťažiska objektu a haptickej spätnej väzby. Tieto hodnoty sú váhované v pomere 1:3. Rovnako hodnoty z jenodlivých senzorov sú škálované a váhovane kombinované do odmeny za dotyk.

$$r(d,p) = 0.25 * r_d + 0.75 * r_p, \tag{3.2}$$

kde $r_p(p) = 2*(0.2*p_t+0.6*p_p+0.2*p_f)-1$, $r_d(d) = 1-2*(d/d_{MAX})$, d je vzdialenosť od objektu, $d_{MAX} = 0.15$ je maximálna uvažovaná vzdialenosť a p je zoznam hodnôt z tlakového senzora (t-palec, p-dlaň, f-4 prsty).

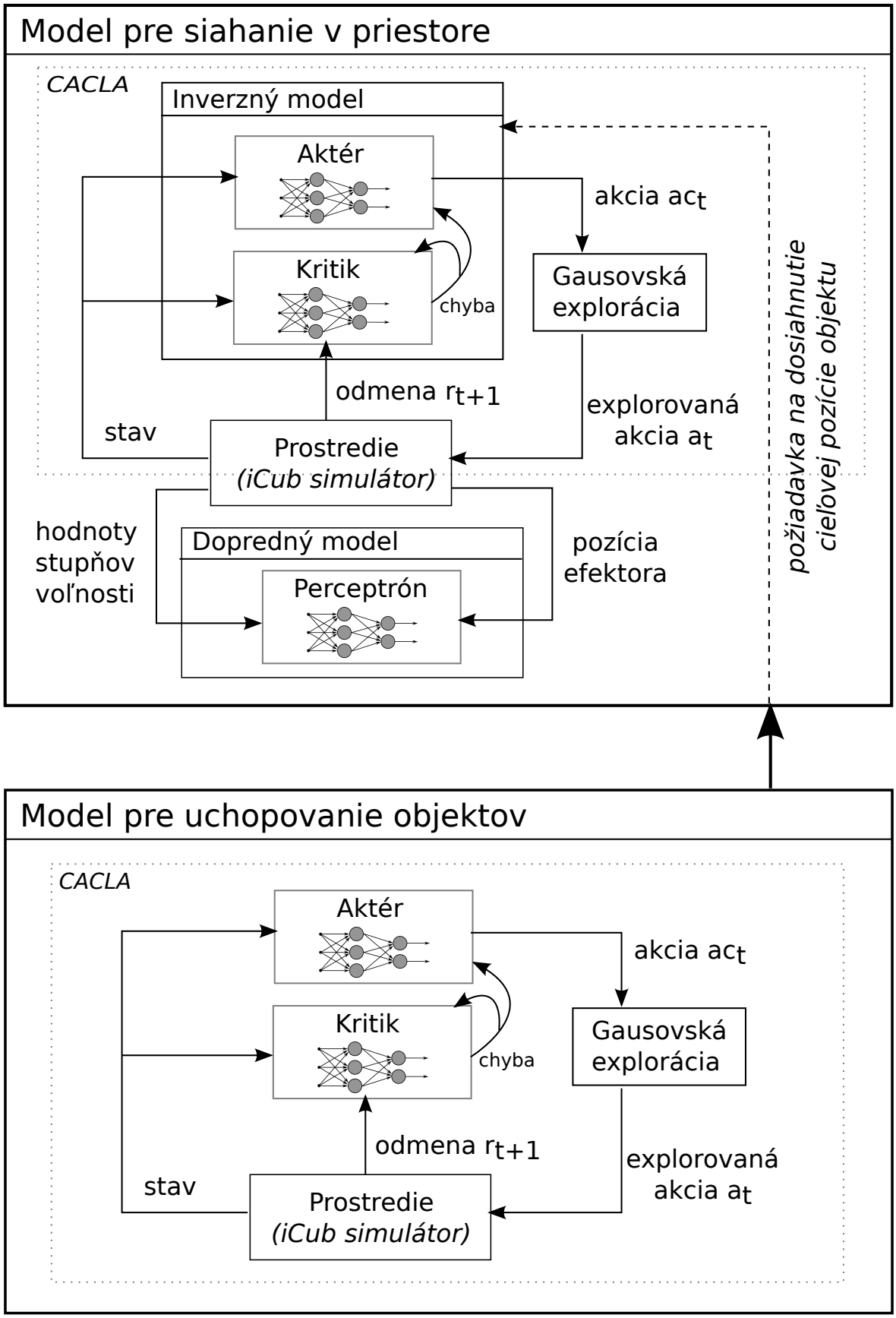
Uvažovali sme aj o rozšírení funkcie odmeny o zložku, ktorá by ohodnocovala ako dokáže iCub daný objekt udržať vo vzduchu. Podobnú odmeňovaciu funkciu využili vo svojom modeli aj Tikhanoff et al. (2011). Jednou z možných realizácií je v prípade dostatočne vysokej dotykovej spätnej väzby zdvihnúť ruku a počas N časových krokov sledovať, či iCub objekt udrží alebo zvíťazí gravitácia. Zistili sme však, že fyzika v simulátore má rôzne chyby a jednou z nich je prilepenie objektu k iCubovej ruke pri silnejšom náraze do objektu. Preto sa stávalo, že iCub objekt síce dokázal zdvihnúť a udržať, nešlo však o prirodzený úchop a v reálnom svete by sa takým spôsobom nepodarilo objekt ani zdvihnúť, nie to ešte udržať vo vzduchu.

3.2.2 Integrácia s modelom pre siahanie v priestore

Navrhli sme vloženie modelu pre siahanie priamo do modelu pre uchopovanie, ktorý je potom použitý na umiestnenie ruky nad cieľový objekt. Predpokladali sme, že sekvenčné volanie modelov za sebou je dostačujúce pre úspešné a kvázi prirodzené uchopenie objektu.

Náš model pre uchopovanie sme zo začiatku trénovali s napevno zadanými príkazmi, pre správny presun a nastavenie ruky nad objekt. Toto zjednodušenie nám umožnilo sústrediť sa na samotné uchopovanie už v čase, keď ešte samotný modul pre siahanie v priestore neposkytoval uspokojujúce výsledky.

Po uspokojivých výsledkoch modelu pre siahanie sme umožnili uchopovaciemu modulu aplikácie po identifikácii pozície cieľového objektu (informácia pochádza z prostredia simulátora) využil natrénovaný model pre siahanie k posunu ruky nad danú pozíciu objektu. Schéma použitia oboch modelov je znázornená na obrázku 3.5. CACLA predstavuje inverzný model siahania v priestore (keďže hľadá správne natočenie uhlov pre danú cieľovú pozíciu). Vytvorili sme aj jednoduchý viacvrstvový perceptrón, ktorý sa po každej epizóde trénuje ako dopredný model (pre absolútne hodnoty stupňov voľnosti určí pozíciu efektora). Dopredný model však nenašiel v našej práci uplatnenie. Veríme, že v budúcnosti by mohol byť použitý pre zefektívnenie učenia inverzného modelu.



Obr. 3.5: Schéma použitia našich modelov pre siahanie a uchopovanie

Kapitola 4

Implementácia

Program, v ktorom sme implementovali navrhnuté neurónové modely sme naprogramovali v jazyku C++, nakoľko v tomto jazyku je naprogramovaný aj iCub a YARP. Sú teda dostupné aj C++ knižnice pre programovanie s iCubom. Všetky zdrojové kódy vrátane rozsiahlej technickej dokumentácie vygenerovanej aplikáciou Doxygen sú dostupné na CD priloženom k diplomovej práci.

Implementovali sme knižnicu pre podporu dopredných neurónových sietí v našej aplikácii, ďalej knižnicu s infraštruktúrou aplikácie (logovanie, systémové utility, špeciálne matematické funkcie...) a knižnicu so základnou implementáciou učiaceho algoritmu CACLA. Za učenie sa dosahovať pozíciu v priestore a uchopovať objekty je zodpovedná hlavná konfigurovateľná aplikácia, ktorá cez YARP priamo komunikuje s iCubom. Keďže iCub simulátor neposkytoval všetku takú funkcionalitu, akú by sme si priali, museli sme zdrojové kódy simulátora upraviť podľa vlastných potrieb.

4.1 Infraštruktúra

4.1.1 Neurónové siete

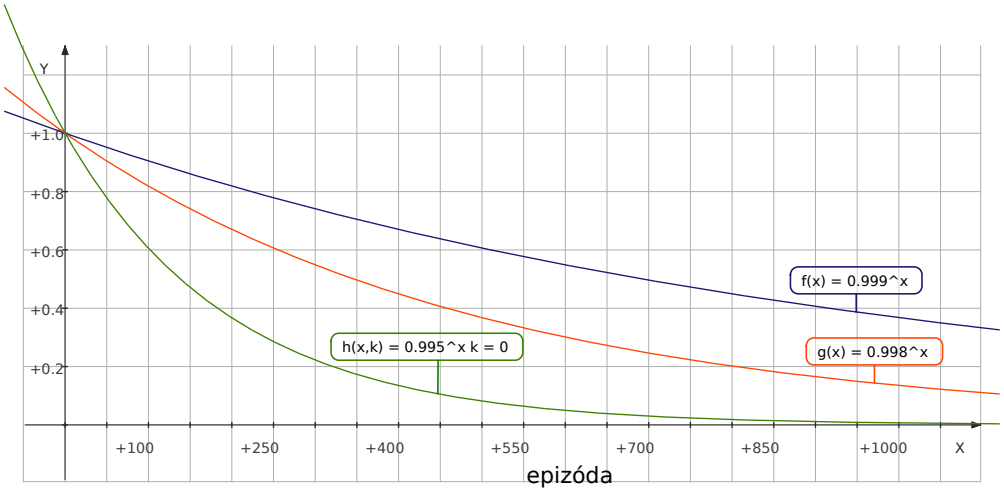
Neurónové siete sú základným stavebným kameňom našej práce, preto sme implementovali knižnicu, ktorá je zodpovedná za prácu s neurónovými sieťami. Obsahuje triedy zastrešujúce dopredné neurónové siete s implementovaným algoritmom spätného šírenia chyby.

4.1.2 Učenie s posilňovaním CACLA

Navrhli sme knižnicu zodpovednú za učenie s posilňovaním rl_lib, ktorú využívame v našej aplikácii. Implementovali sme triedy predstavujúce aktéra a kritika a vytvorili sme abstraktné triedy pre prostredie, a CACLA algoritmus. Jednotlivé modely (pre siahanie v priestore a pre uchopovanie) následne odvodzujú svoje prostredie a CACLU z týchto abstraktných tried.

Aktér aj kritik sú modelované ako funkčné aproximátory a sú závislé na knižnici nn_lib zodpovednú za prácu s neurónovými sieťami.

Využívali sme postupné znižovanie exploračného faktora v priebehu učenia. Po každej epizóde sa exploračný faktor prenásobil zvolenou konštantou. Testovali sme konštanty 0.995, 0.998 a 0.999. Na obrázku 4.1 je znázornený pomer exploračného faktoru pri danej epizóde voči počiatočnej hodnote. Pri konštante 0.995 sa exploračný faktor znižoval príliš dramaticky, preto sme neskôr vo všetkých experimentoch využívali len zvyšné 2 konštanty.



Obr. 4.1: Vývoj exploračného faktoru pri 3 rôznych násobiacich konštantách

4.1.3 Komunikácia s iCub simulátorom

Robot iCub je postavený na otvorenej robotickej platforme YARP (Yet Another Robot Platform), ktorú vytvorilo konzorcium RobotCub. Ide o implementáciu strednej vrstvy ISO OSI modelu, tzv. middleware pre komunikáciu jednotlivých častí humanoidných robotov. Samotné časti robota (ruka, hlava, kamery, dotykové senzory...) vystupujú z pohľadu architektúry ako klienti, ktorí sa cez jedinečné názvy portov registrujú

v YARP serveri, ktorý zabezpečuje ich vzájomnú komunikáciu na základe návrhového vzoru Observer[1].

Dôležitým aspektom použitia YARPu je fakt, že budúce potenciálne roboty postavené na tejto platforme budú vedieť spolu komunikovať. Cieľom je, aby neboli roboty stavané na proprietárnych platformách a ich vzájomná komunikácia je tak veľmi obmedzená.

Jednotlivé časti robota si teda medzi sebou posielajú správy (napr. príkaz na nastavenie určitého stupňa voľnosti pravej ruky), alebo čítajú tok dát (napr. z dotykových senzorov na dlani). Aby sme vedeli robota ovládať z aplikácie, musíme sa cez YARP pripojiť na porty, na ktorých sú registrované požadované súčasti.

My sme implementovali samostatnú triedu YarpAdapter, ktorá obaľuje všetku potrebnú inicializáciu, týkajúcu sa YARPu. YarpAdapter inicializuje interný ovládač pravého ramena, ovládač pohľadu očí (v orig. gaze controller) a pripojí sa na porty trupu, simulovaného sveta, dotykových senzorov pravej ruky a taktiež ku obrazu z kamery v pravom oku.

Zdrojový kód 4.1: Inicializácia ovládača ramena - ukážka zdrojového kódu

```
1  void YarpAdapter::initICubDriver() {
2      ...
3
4      // Port names
5      std::string remotePorts="/" + robotName + "/right_arm";
6      std::string localPorts="/icub_cacla/client_arm";
7
8      Property options;
9      options.put("device", "remote_controlboard");
10     // Set local port name
11     options.put("local", localPorts.c_str());
12     // Set remote port name
13     options.put("remote", remotePorts.c_str());
14
15     // Connect local YARP device as controller of remote device
16     robotDevice = new PolyDriver(options);
17
18     if (!robotDevice->isValid()) {
19         printf("Device not available. Here are the known devices:\n");
20         printf("%s", Drivers::factory().toString().c_str());
21     }
22     else printf("Device is available.\n");
23
24     ...
25 }
```

Z ovládača ramena následne získame inštancie implementujúce IPositionController a IEncoders. Rozhranie IPositionController umožňuje volaním metódy setPosition

[1]Observer je známy návrhový vzor GoF pre komunikáciu, kde subjekt notifikuje (posiela správu) pri určitej príležitosti ďalšie objekty - svojich pozorovateľov.

asynchrónne poslať príkaz na nastavenie dostupných stupňov voľnosti. Rozhranie IEncoders umožňuje prečítať aktuálne hodnoty stupňov voľnosti.

4.1.4 Úpravy iCub simulátora

Pri našej práci sme sa niekedy dostali do ťažkostí, kvôli nedostatočnej funkcionalite poskytovanej simulátorom, preto sme si museli naštudovať jeho zdrojový kód obsahujúci pomerne veľké množstvo kódu pracujúceho s ODE a OpenGL. Rozhodli sme sa vykonať nasledovné úpravy simulátora, ktoré buď upravovali parametre simulácie alebo rozširovali funkcionalitu simulátora.

Urýchlenie simulácie

Keďže pri učení využívame robota interagujúceho vo virtuálnom prostredí, musíme sa uspokojiť s faktom, že jedna epizóda bude trvať toľko času, koľko je reálne potrebné na vykonanie danej akcie v simulátore. Neurónové siete vyžadujú k úspešnému učeniu často tisíce epizód, preto sme spočiatku čelili problému s príliš pomalým učením.

O radu sme požiadali komunitu pracujúcu s iCubom a od talianskeho vedca Valeria Sperati sme vzápätí dostali odpoveď, ktorá nás nasmerovala k správnym častiam zdrojového kódu simulátora. Zdrojové súbory k simulátoru sa nachádzajú v repozitári projektu iCub v priečinku *main/src/simulators/iCubSimulation*. Dôležité sú najmä zdrojové súbory umiestnené v priečinku *odesdl*, ktoré pracujú s fyzikálnou knižnicou ODE. Pre urýchlenie je potrebné upraviť konštanty zabezpečujúce beh simulácie v reálnom čase.

V súbore *iCub_Sim.cpp* je možné upraviť metódy `OdeSdlSimulation::thread_func` a `OdeSdlSimulation::ODE_process`. V metóde thread_func sa nachádza premenná `delay` inciálne nastavená na hodnotu 50ms. Jej znížením na hodnotu 0ms dosiahneme výrazné urýchlenie simulátora. Tento parameter určuje, v akom intervale sa volá metóda
ODE_process. V tejto metóde je podstatné nastavenie veľkosti kroku vo virtuálnom ODE svete pomocou funkcie `dWorldStep`, ktorý sa má vypočítať. Predvolená hodnota je 0.01s a pri intervale 50ms to znamená, že aktualizujeme v simulácii fyziku každých 20-krát za sekundu, pričom sa vo virtuálnom svete pohneme o 0.20s za 1s.

My sme simuláciu urýchlili znížením intervalu ODE výpočtov na 0, avšak dWorldStep sme ponechali na pôvodnej hodnote, čím sme dosiahli to, že výpočet kroku virtuálneho sveta o dĺžke 0.01s sa vykonáva tak často, ako je to len výpočtovo možné.

Ak nám hardvér umožní vypočítať krok v ODE svete častejšie než 100-krát za sekundu, získame zrýchlenú simuláciu.

Možnosť nastavenia hustoty objektov

V našej práci sme chceli vyskúšať uchopovanie objektov rôznych hmotností. Simulátor podporuje vytváranie vlastných objektov na scéne (3 základné objekty - kváder, guľa a valec) ako aj importovanie vymodelovaných 3D objektov. Ďalej umožňuje tieto objekty posúvať, otáčať a odstraňovať. Neumožňuje však nastavenie váhy alebo hustoty vytváraného objektu. Preto sme museli nájsť odpovedajúcu metódu, ktorá spracúva požiadavky na vytváranie objektov a rozšíriť ju o možnosť parametrizovania hustoty objektu.

Spracúvanie požiadaviek na manipuláciu so svetom je implementované v súbore *WorldManager.cpp* v priečinku *odesdl*. Metóda `OdeWorldManager::respond` je zodpovedná za sparsovanie príkazu a vykonanie požadovanej úpravy sveta.
Pre ilustráciu uvedieme príkaz: `world mk box 0.1 0.1 0.1 1 2 3 1 0 0`
WorldManager spracuje príkaz nasledovne:

- Reťazec na druhej pozícii označuje typ akcie (mk - make, set - nastaviť pozíciu, get - zistiť pozíciu, rot - otočiť objekt, del - odstrániť objekt).
- Reťazec na tretej pozícii špecifikuje typ objektu (box - kváder, cyl - valec a sph - guľa). Predponou s môžeme definovať statický objekt, ktorý nebude ovplyvnený žiadnou silou (takže sa môže aj vznášať).
- Trojica (vyfarbená modrou) nastavuje veľkosť nového objektu v metroch (líši sa podľa typu).
- Ďalšia trojica (vyfarbená zelenou) určuje pozíciu vytvoreného objektu v metroch v pravotočivej karteziánskej súradnicovej sústave.
- Posledná trojica (vyfarbená červenou) udáva farbu objektu vo farebnom modeli RGB.

My sme metódu respond upravili tak, že očakávame v zadanom príkaze pre mk ešte jeden dodatočný parameter - hustotu objektu, ktorá je následne cez funkcie ODE (dMassSetBoxTotal, dMassSetCylTotal, dMassSetSphereTotal) priradená novovytvorenému ODE objektu.

Zdrojový kód 4.2: Ukážka modifikovaného zdrojového kódu simulátora

```
bool OdeWorldManager::respond(const Bottle &command, Bottle &reply) {
    ...

    if (subcmd=="mk"){ //this allows the user to create some objects around the world
        if (setBody==2){ // box with gravity
            if (num < MAXNUM){
                i = odeinit._wrld->OBJNUM;
            }
            odeinit._wrld->obj[i].size[0] = command.get(3).asDouble();
            odeinit._wrld->obj[i].size[1] = command.get(4).asDouble();
            odeinit._wrld->obj[i].size[2] = command.get(5).asDouble();
            double x = command.get(6).asDouble(); // x position
            double y = command.get(7).asDouble(); // y position
            double z = command.get(8).asDouble(); // z position
            double R = command.get(9).asDouble(); // colour R
            double G = command.get(10).asDouble();// colour G
            double B = command.get(11).asDouble();// colour B
            double density = command.get(12).asDouble(); // -> parsovanie hustoty

            odeinit.mutex.wait();

            dMass m;
            dMassSetZero(&m);
            odeinit._wrld->obj[i].boxbody = dBodyCreate (odeinit.world);
            dMassSetBoxTotal (&m,density, odeinit._wrld->obj[i].size[0], ...); // -> vypocet hmotnosti
            dBodySetMass (odeinit._wrld->obj[i].boxbody, &m); // -> priradenie hmoty k objektu
```

Možnosť získať informáciu o pozícii ľubovoľnej časti tela robota

Agent pre voľbu správnej akcie potrebuje rozmanité stavové informácie, ktoré vníma z prostredia. Keďže abstrahujeme od spracovania niektorých vizuálnych informácii priamo z obrazu, získavame ich hodnoty priamo zo simulátora. Základná verzia simulátora umožňuje získať informácie o 3D pozícii ťažiska pravej a ľavej dlane. Informácie o pozíciách iných častí robota nie sú štandardne dostupné. Preto sme museli analyzovať kód zodpovedný za vytváranie tela robota a ten následne modifikovať.

Robot iCub je v simulátore vytvorený z veľkého množstva primitívnych objektov, ktoré sú jednoznačne identifikovateľné pomocou tzv. dBodyId. Tieto objekty sú potom pomocou kĺbov spojené do väčších celkov (prsty, ruka a podobne). V metóde `OdeWorldManager::respond` je možné vyžiadať si aktuálnu pozíciu ťažiska objektu v priestore. Z objektov tvoriacich telo iCuba sa však dá získať len pozícia ťažiska pravej a ľavej dlane. V našej práci sme chceli využiť aj inú dodatočnú informáciu ako napr. natočenie dlane alebo vzdialenosť palca a ukazováka. Preto sme rozšírili túto metódu tak, aby bolo možné pýtať sa na pozíciu ľubovoľnej časti iCuba na základe hodnoty dBodyId.

Opäť uvádzame ilustračný príkaz: `world get rhand` alebo `world get cyl 2`

Na uvedené príkazy by sme dostali v odpovedi súradnice pozície ťažiska pravej dlane resp. ťažiska druhého valca. Po našej úprave je možné zadať príkaz: `world get icub 31`

Tento príkaz sa spracuje ako požiadavka na získanie pozície objektu s dBodyId 31 - posledný článok pravého ukazováka.

Najdôležitejšia zmena spočívala v úprave pomocnej funkcie `static int nameToBody(int id, dBodyID& bid, dGeomID& bid2, int index)`, ktorá pre identifikátor (rhand, cyl, icub...) vráti jeho dBodyID resp. dGeomID (pre statické objekty). Táto funkcia sa používa v metóde `OdeWorldManager::respond` a následne používa už len získané dBodyID resp. dGeomID. My sme zneužili posledný parameter index, v ktorom posielame požadované id časti tela ako číslo a v tele funkcie získame konkrétne dBodyID.

Zdrojový kód 4.3: Úprava zdrojového kódu v súbore WorldManager.cpp

```
1  static int nameToBody(int id, dBodyID& bid, dGeomID& bid2, int index = 0) {
2      OdeInit& odeinit = OdeInit::get();
3
4      int setBody = 0;
5      switch (id) {
6          ...
7          case VOCAB4('i','c','u','b'): // dopyt na cast tela iCuba
8              bid = odeinit._iCub->body[index]; // dBodyID danej casti
9              printf("Body part with index %d\n", index);
10             setBody = 4; // v respond docielime rovnake spracovanie ako pri kvadri
11             break;
12         ...
13     }
14     return setBody;
15 }
16
17 bool OdeWorldManager::respond(const Bottle &command, Bottle &reply) {
18     OdeInit& odeinit = OdeInit::get();
19     ...
20
21     if (subcmd=="get"||subcmd=="set"||subcmd=="mk"||...) {
22         int id = command.get(2).asVocab(); // retazec rhand/icub/.. ako VOCAB
23         int index = command.get(3).asInt(); // id objektu iCuba
24
25         dBodyID bid = NULL;
26         dGeomID bid2 = NULL;
27         setBody = nameToBody(id,bid,bid2,index);
28         ...
```

4.2 Modul pre siahanie v priestore

Zodpovednosť za pohyb ramena v priestore za účelom dosiahnutia pozície blízko cieľa je zodpovedný modul pre siahanie v priestore. Implementovali sme špecializované triedy pre CACLA algoritmus a prostredie odvodené od abstraktných tried v knižnici pre strojové učenie. Prostredie špecifické pre siahanie zahŕňa výpočet aktuálnej vzdialenosti dlane od cieľovej pozície, reštartovanie stavu prostredia pre každú epizódu a najmä výpočet odmeny po vykonaní akcie.

Modul v počiatočnom stave obsahuje nenatrénované neurónové siete aktéra a kritika. Trénovanie spočíva v generovaní náhodných cieľových pozícií z ohraničujúceho kvádra (pracovného priestoru robota) podľa tabuľky, ktoré sa transformujú na počiatočný stav pre každú epizódu.

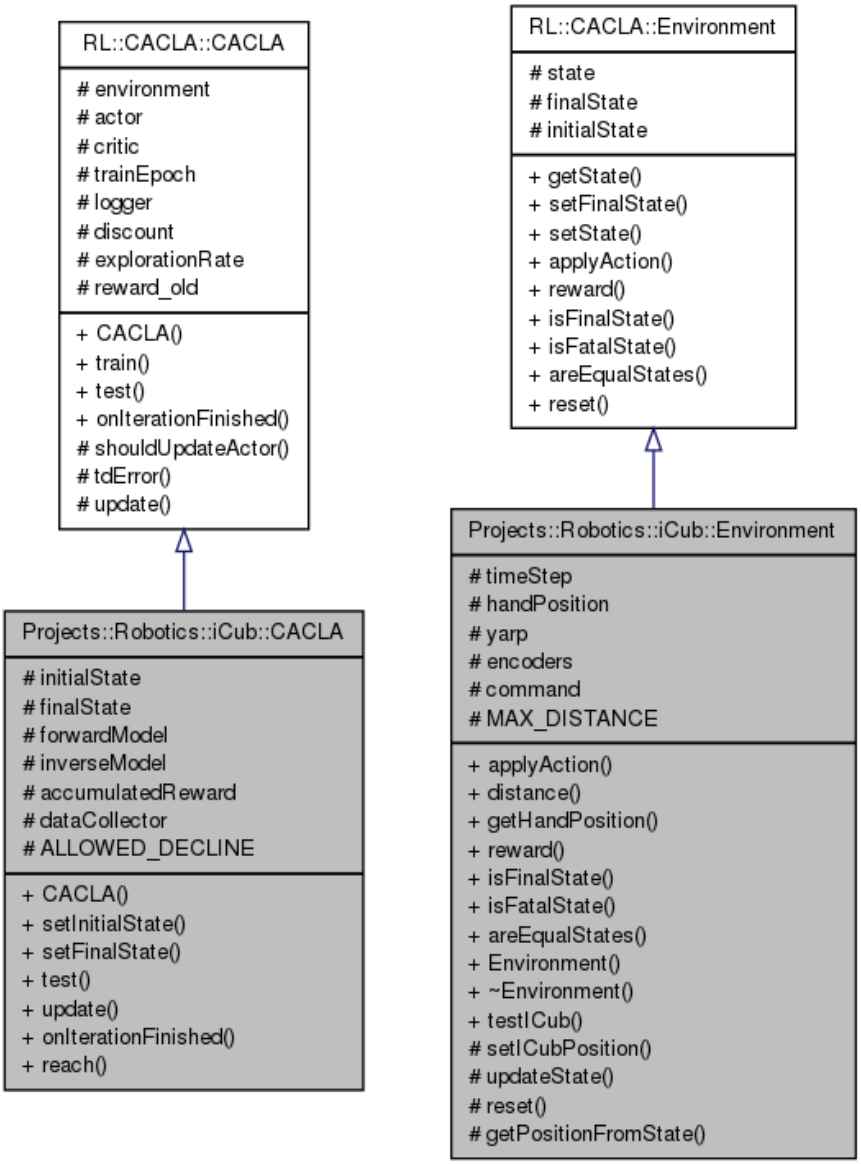
os	x	y	z
od	-0.2	0.55	0.2
do	0.05	0.75	0.3

Na trénovanie pomocou CACLA sa používa základná implementácia algoritmu z vlastnej knižnice pre strojové učenie. Aktér vygeneruje 4-rozmerný vektor akcie, škálované cieľové zmeny uhlov pre 1. až 4. stupeň voľnosti ramena. Ten sa následne prenásobí hodnotou (< 1), ktorá závisí od vzdialenosti k cieľu. Čím je cieľ bližšie tým, nižšia škálovacia hodnota a tým nižšie výsledné zmeny uhlov. Tie sa následne ešte orezávajú v prípade, že hodnota vyskočí z intervalu $\langle -1, 1 \rangle$.

Zdrojový kód 4.4: Ukážka metódy applyAction pre vykonanie generovanej akcie

```
1  State Environment::applyAction(Action action) {
2      // Vypocet skalovacieho faktora v zavislosti od vzdialenosti
3      double scaleFactor = 0.3 + 0.3 * (this->distance() / Environment::MAX_DISTANCE);
4
5      // Upravime jednotlive uhly o zmeny z akcie
6      for (int i = 0; i<4; i++) {
7          state.values[i] += action.values[i] * scaleFactor;
8          state.values[i] = max(min(state.values[i], 1.0), -1.0);
9      }
10
11     // Preskalujeme spat do realnych uhlov pre iCuba
12     command[0] = ICubSim::reScale(-95, 90, state.values[0]);
13     command[1] = ICubSim::reScale(0, 161, state.values[1]);
14     command[2] = ICubSim::reScale(-37, 100, state.values[2]);
15     command[3] = ICubSim::reScale(6, 106, state.values[3]);
16
17     // Posleme pokyn na nastavenie ramena
18     this->setICubPosition(command.data());
19
20     return this->state;
21 }
```

Stav sa následne vyhodnotí funkciou odmeny a aktualizujú sa váhy kritika a vzhľadom na splnenie podmienky aj váhy aktéra. V každej novej epizóde (ktorá trvá určitý počet krokov) sa na začiatku stav prostredia resetne. Úvodné uhly štyroch používaných stupňov voľnosťi sa nastavia náhodne. Keďže aktér generuje len zmeny uhlov, kroky epizódy vlastne vytvárajú celú trajektóriu pohybu pre dosiahnutie cieľa.

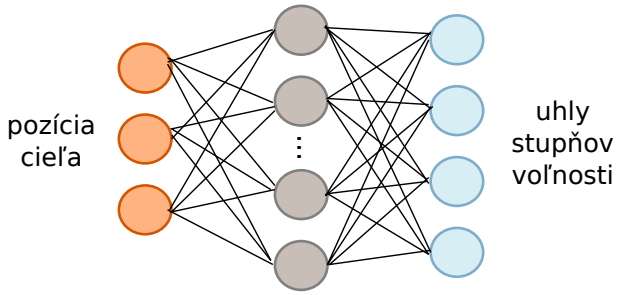


Obr. 4.2: Zjednodušený model tried pre siahanie

4.2.1 Ovládač založený na viacvrstvovom perceptróne

V článku Tikhanoff et al. (2011) použili na siahanie pozície iCuba jednoduchý viacvrstvový perceptrón. Keďže ten sme už mali implementovaný v knižnici zodpovednej za neurónové siete, rozhodli sme sa doplniť modul o alternatívny ovládač ramena zostrojený podľa špecifikácie v článku talianskych vedcov. Chceli sme porovnať výsledky našej implementácie viacvrstvového perceptrónu pre tento problém s výsledkami v článku a takisto ich porovnať s výsledkami nášho CACLA modelu.

Trénovacie príklady sme generovali z už vytvoreného CACLA modelu, ktorému sme výrazne zvýšili exploračný faktor, aby robot ramenom čo najviac exploroval svoj pracovný priestor. Jednotlivé pozície a korešpondujúce uhly stupňov voľnosti sme zaznamenávali do súboru. Z celkových 14900 trénovacích príkladov sme po odstránení duplikátov získali 9914 príkladov. Pred trénovaním sme náhodne zvolili 5000 trénovacích príkladov a polovicu z nich sme používali na trénovanie, druhú polovicu na testovanie. V závislosti od architekúry siete sa chyba RMSE na testovacích dátach pohybovala v intervale $\langle 0.11, 0.22 \rangle$ čo korešponduje so zverejnenou chybou 0.156 Tikhanoffa et al v ich spomínanom článku. Dospeli sme k záveru, že kvalita naučenia aproximovať priestor je silno závislá od kvality trénovacích dát (najmä rovnomerné rozmiestnenie cieľových bodov v pracovnom priestore robota). Testovali sme rôzny počet skrytých neurónov v intervale $\langle 10, 40 \rangle$, rôzne aktivačné funkcie (sigmoida aj hyperbolický tangent) a rôzne rýchlosti učenia $\langle 0.1, 0.001 \rangle$.



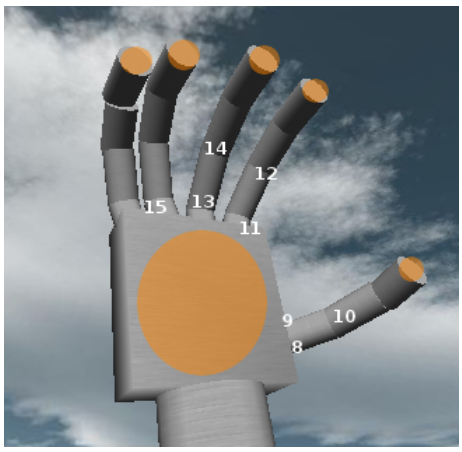
Obr. 4.3: Architektúra viacvrstvového perceptrónu pre siahanie

Nami implementovaný ovládač ramena pre siahanie implementuje rovnaké rozhranie ako náš CACLA model, preto je možné v príprave pri uchopovaní použiť ľubovoľný z nich. Z kognitívneho hľadiska je však vierohodnejšie učenie s posilňovaním než učenie s učiteľom použitom pre trénovanie perceptrónu.

4.3 Modul pre uchopovanie objektov

Najdôležitejšou a najnáročnejšou časťou diplomovej práce bola implementácia modelu pre uchopovanie objektov. Triedy zodpovedné za uchopovanie tvoria samostatný modul, ktorý je možné využívať v ďalších častiach aplikácie. Implementácia opäť vychádza z abstraktných tried knižnice pre strojové učenie.

Funkciu odmeny sme popísali v kapitole Návrh a jej implementácia je o niečo náročnejšia než odmena pre siahanie. Odmena je pri učení s posilňovaním asi najdôležitejší aspekt, ktorý ovplyvňuje učenie. V našom prípade zahŕňa funkcia odmeny zložku euklidovskej vzdialenosti od cieľového objektu a potom nemenej dôležitú dotykovú zložku. Vzdialenosť je implementovaná totožným spôsobom ako pri siahaní v priestore, preto sa ňou ďalej nebudeme zaoberať. V našej obálke nad simulátorom sme implementovali metódy a dátovú štruktúru pre spracovanie informácie z dotykových senzorov.



Obr. 4.4: Dotykové senzory na ruke iCuba a stupne voľnosti na prstoch

Robot iCub má na každej ruke k dispozícii šesť dotykových/tlakových senzorov. Päť senzorov je umiestnených na prstoch a jeden na dlani. Každý z týchto senzorov sa skladá z 12 menších senzorov, čo ešte viac zvyšuje rozlišovaciu schopnosť pri dotyku. V prípade simulátoru sa však tieto senzory správajú ako jeden a všetky sú nastavené na rovnakú hodnotu. Voľba dotyk/tlak sa nastavuje prepínačom pressure v konfiguračnom súbore iCuba *app/simConfig/conf/iCub_parts_activation.ini*. Dotykové senzory vracajú hodnotu 255 v prípade dotyku, 0 v opačnom prípade. Tlakové senzory vracajú hodnotu z intervalu $\langle 0, 255 \rangle$ podľa intenzity dotyku.

V našom modeli pracujeme aj s tlakovými senzormi, keďže je pre nás podstatná aj sila dotyku s objektom. Kvôli zjednodušeniu sme sa rozhodli pracovať so 4 prstami (mimo palca) ako s jediným a preto hodnoty z ich tlakových senzorov priemerujeme.

Tabuľka 4.1: Popis stupňov voľnosti využívaných v modeli pre uchopovanie

	popis	povolený rozsah	využívaný rozsah
0	ramenný kĺb - 1. stupeň voľnosti	$\langle -95, 90 \rangle$	$\langle 0, 90 \rangle$
2	ramenný kĺb - 3. stupeň voľnosti	$\langle -37, 100 \rangle$	$\langle -20, 60 \rangle$
4	zápästie - otáčanie okolo osi	$\langle -90, 90 \rangle$	$\langle -90, 90 \rangle$
5	zápästie - nakláňanie nahor/nadol	$\langle -90, 10 \rangle$	$\langle -30, 0 \rangle$
6	zápästie - posúvanie doprava/doľava	$\langle -20, 40 \rangle$	$\langle -20, 40 \rangle$
8	palec - veľký kĺb	$\langle -15, 105 \rangle$	$\langle -15, 105 \rangle$
10	palec - kĺb v strede prstu	$\langle 0, 90 \rangle$	$\langle 0, 90 \rangle$
11	ukazovák - veľký kĺb	$\langle 0, 90 \rangle$	$\langle 0, 90 \rangle$
12	ukazovák - kĺby medzi článkami	$\langle 0, 90 \rangle$	$\langle 0, 90 \rangle$
13	prostredník - veľký kĺb	$\langle 0, 90 \rangle$	$\langle 0, 90 \rangle$
14	prostredník - kĺby medzi článkami	$\langle 0, 90 \rangle$	$\langle 0, 90 \rangle$
15	prstenník a malíček	$\langle 0, 115 \rangle$	$\langle 0, 90 \rangle$

Všetky hodnoty škálujeme na interval $\langle 0, 1 \rangle$ a tak využívame vo funkcii odmeny.

Pri uchopovaní využívame až 12 z celkovo 16 stupňov voľnosti. Na obrázku 4.4 je zobrazených 8 stupňov voľnosti prstov[2]. V tabuľke 4.1 uvádzame popis a pracovný rozsah jednotlivých stupňov voľnosti, ktoré sme využili v našom modeli. Z dôvodu obmedzenia nechcených extrémne zlých pohybov nevyužívame v modeli celý pracovný rozsah u väčšiny stupňov voľnosti. Taktiež sme na ovládanie niektorých stupňov voľnosti použili jediný neurón, ktorý ich ovláda ako skupinu. Toto obmedzenie sme použili na ovládanie štyroch prstov, ktoré sa preto pohybujú spolu.

[2]prstenník a malíček zdieľajú z pohľadu ovládania jediný stupeň voľnosti

Kapitola 5

Výsledky

5.1 Učenie sa siahať v priestore

Problém aproximácie motorického priestoru robotickým ramenom je známy už desiatky rokov a riešením boli spočiatku algoritmy založené na matematike, konkrétne metódy inverznej kinematiky. Úlohou inverznej kinematiky je uhádnuť (vypočítať) správne nastavenia stupňov voľnosti ramena, aby efektor (koncová časť) dosiahol určitú pozíciu v priestore. V jednoduchých aplikáciách tejto metódy je možné vypočítať priamo cieľové uhly už z iniciálneho stavu, väčšinou sa však využíva metóda klesajúceho gradientu, na základe ktorej sa iteratívne vypočítavajú uhly, pričom výpočet vychádza vždy z aktuálneho stavu ramena. I keď je inverzná kinematika v robotike zrejme najpoužívanejšou metódou pre siahanie v priestore, s umelou inteligenciou nemá veľa spoločného. Z pohľadu kognitívnej robotiky nemá pri skúmaní učenia sa uchopovať žiadne opodstatnenie, pretože ani dieťa sa nerodí s vrodeným, na matematike založeným algoritmom na siahanie v priestore.

V práci Tikhanoffa použili na siahanie v priestore jednoduchú neurónovú sieť s učením s učiteľom. Podľa nášho názoru takáto sieť len hľadá mapovanie medzi bodmi z priestoru a správnym nastavením uhlov ramena, teda aproximuje rovnicu, ktorá sa využíva pri inverznej kinematike. Navyše táto architektúra negeneruje trajektóriu pohybu z iniciálnej do cieľovej pozície, ale len vygeneruje cieľové uhly. Ich architektúru sme sa rozhodli verifikovať a porovnať s našou aplikáciou CACLA učenia na tento problém.

5.1.1 Využitie existujúcich modelov

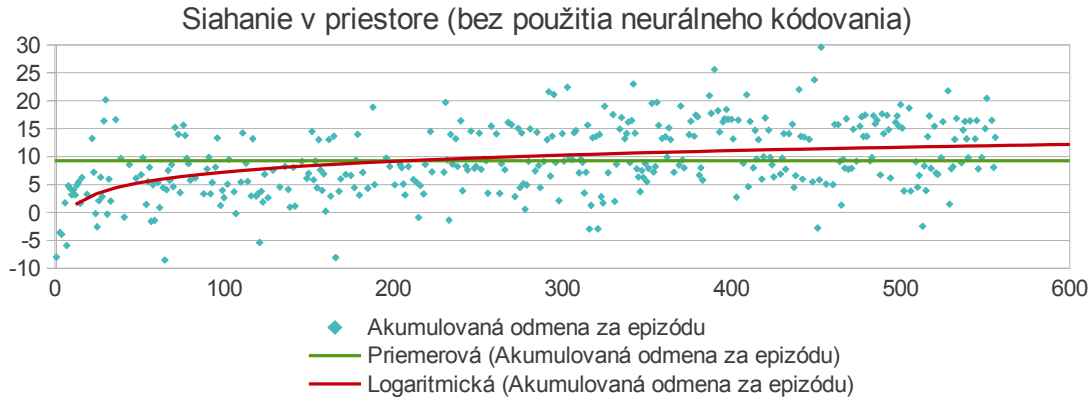
V našom modeli sme spočiatku nevyužívali škálovanie, ako na vstupe, tak ani na výstupe. Navyše na vstupe sme zadávali len súradnice cieľovej pozície. Lineárna aktivačná funkcia sa však ukázala ako absolútne nevyhovujúca na tento typ úlohy a preto sme využili tangenciálnu aktivačnú funkciu ako na skrytej vrstve, tak aj na výstupnej vrstve. Samozrejme výstup siete bol už teda v intervale $(0, 1)$ a tieto hodnoty sme škálovali späť na konkrétne uhly podľa rozsahu príslušného stupňa voľnosti. Implementovali sme rovnakú architektúru pre aktéra a kritika ako vo svojej diplomovej práci navrhol Korenčiak (2010), avšak pri použití so simulátorom humanoidného robota sa ukázala ako ťažko použiteľná, nakoľko autor uvádza, že na natrénovanie jeho model potrebuje vyše 100 000 iterácií, čo by v simulátore trvalo netriviálny čas. Inšpirovali sme sa preto architektúrou pre CACLA, ktorú navrhol vo svojej diplomovej práci Malík (2011) a tá navyše využívala informáciu o aktuálnom stave uhlov a na výstupe generovala zmeny uhlov a nie cieľové hodnoty uhlov. Autor však CACLA využíval na zjednodušené siahanie v priestore, nakoľko sa robot učil siahať len na tri diskrétne pozície zadané do siete formou lokalistického (one-hot) kódovania.

5.1.2 Priebeh učenia

Zhrnieme výsledky učenia rôznych „dobrých“ variantov nášho modelu pri rôznych architektúrach sietí. Na vizualizáciu učenia využívame odmenu akumulovanú za epizódu. Keďže každá epizóda sa skladá z niekoľkých krokov (u nás spravidla 50) a za každý krok dostane robot z prostredia odmenu, sčítaním týchto odmien získame meranú akumulovanú odmenu.

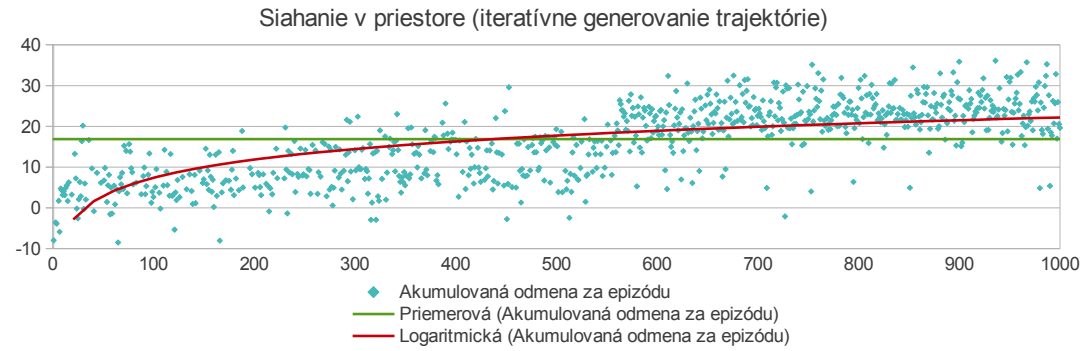
Na začiatku sme uvažovali aj o modeli, kde by aktér generoval na výstupe priamo cieľové hodnoty uhlov a teda iCub by pohyb vykonával na jeden krát. Na obrázku 5.1 vidíme, že model sa síce učil, avšak v porovnaní s neskoršími experimentami nedosiahol vysokú úspešnosť. Priermerná akumulovaná odmena bola pod 10 a len v málo epizódach získal iCub v súčte odmenu viac ako 15. Prisudzujeme to najmä tomu, že po úplnom pregenerovaní uhlov v každom kroku ramena, toto príliš nekontrolovane lietalo v priestore.

Neskôr sme sa rozhodli prerobiť model tak, aby sa pri siahaní generovala trajektória a teda aktér generoval zmeny uhlov. Graf na obrázku 5.2 vykazuje vyššiu úspešnosť použitého modelu. Prvých 500 epizód bol exploračný faktor ešte pomerne vysoký a iCub sa venoval najmä prieskumu (explorácii) prostredia. Neskôr, keď explorácia klesla pod



Obr. 5.1: Priebeh akumulovanej odmeny po epizódach

hodnotu 0.1 je vidieť, že väčšina pokusov skončila s nadpriemernou odmenou.

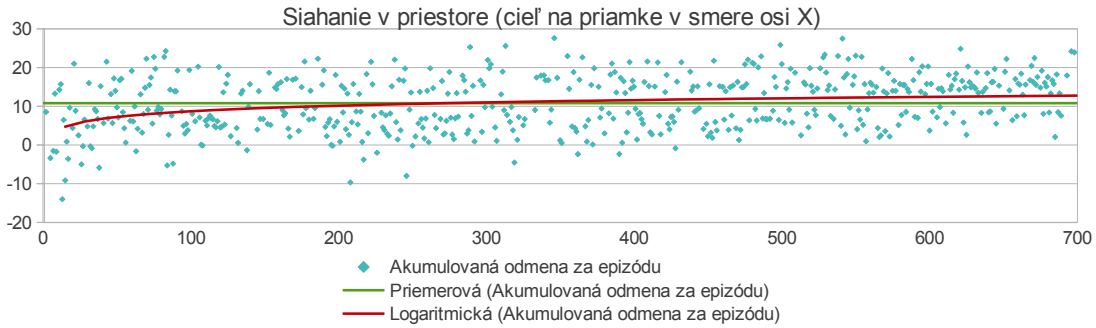


Obr. 5.2: Priebeh akumulovanej odmeny po epizódach - generovanie trajektórie

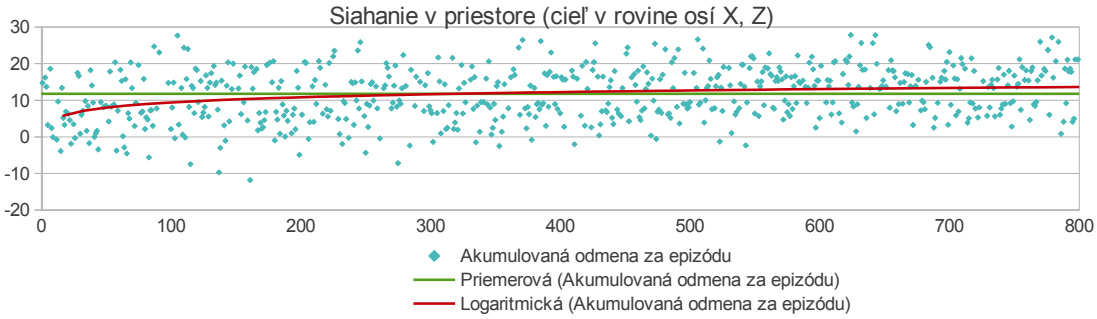
Medzi jeden z problémov, ktoré podľa nás spôsobovali ešte stále nie dokonalé naučenie, bola reprezentácia cieľovej pozície (3 súradnice pomocou 3 vstupných neurónov). Preto sme sa rozhodli využiť populačné kódovanie, kde sme kódovali 1 súradnicu pomocou 9 neurónov. Nasledujúci graf na obrázku 5.3 znázorňuje priebeh učenia po tejto úprave v prípade, že cieľ bol generovaný pozdĺž úsečky rovnobežnej s osou X (súradnica na osi Y aj na osi Z boli dané natvrdo). Musíme však poznamenať, že pri týchto experimentoch sme používali 30 krokov na 1 epizódu.

Následne sme zťažili úlohu pridaním ďalšieho rozmeru do generovania cieľových pozícií, tie už mohli byť z obdĺžnika (ak uvažujeme napr. siahanie k objektom na stole, nevadí nám natvrdo zadaná Y-ová súradnica). S výsledkami takéhoto modelu sme boli spokojní a dosiahol porovnateľnú úspešnosť s predchádzajúcim modelom, kde sme sa v cieli obmedzovali len na pohyb po úsečke.

Postupne sme získali veľa skúseností s nastavovaním príslušných parametrov učenia,



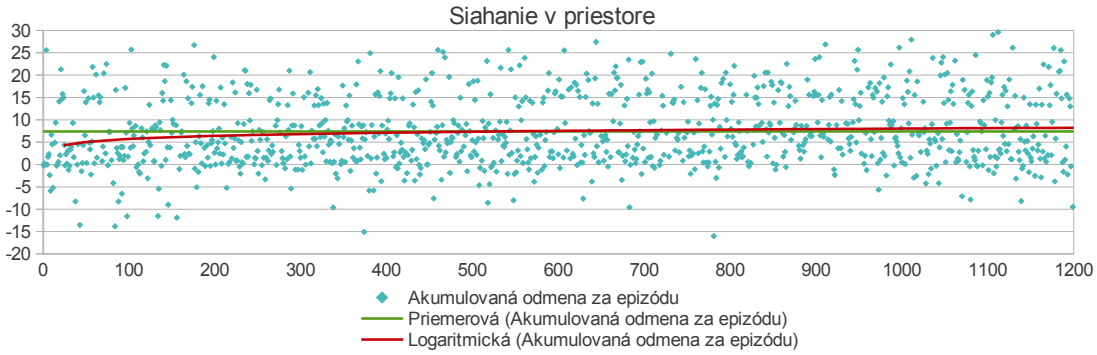
Obr. 5.3: Priebeh akumulovanej odmeny po epizódach - os X



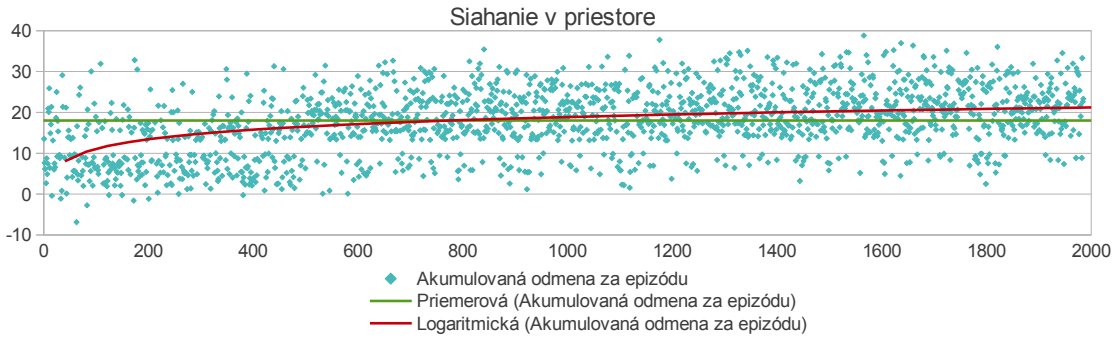
Obr. 5.4: Priebeh akumulovanej odmeny po epizódach - osi X a Z

a preto sme sa rozhodli vyskúšať model opäť na klasickom siahaní v 3-rozmernom priestore a cieľové pozície sme generovali z kvádra o rozmeroch 30cm × 20cm × 15cm. Obrázok 5.5 vyjadruje úspešnosť modelu, ktorá však i tak nebola pre použitie v uchopovaní dostatočne použiteľná, nakoľko aktér stále pomerne slabo rozoznával cieľové pozície a jeho pohyb smeroval takmer vždy cez ťažisko kvádra.

Po implementácii viacvrstvového perceptrónu, ktorý dokázal vykonávať úlohu siahania v priestore pomerne dobre a s jednoduchšou architektúrou než sme doposiaľ používali my, sme sa rozhodli vyskúšať zjednodušenú architektúru sietí aj v našom modeli. Počet neurónov v skrytej vrstve sme znížili na 10 a rýchlosť učenia sme u aktéra zvýšili na 0.01. Výsledkom, ktorý je vizualizovaný na obrázku 5.6, bol lepší priebeh učenia než v predchádzajúcich prípadoch. Počet krokov na 1 epizódu sme opäť zvýšili na 50.



Obr. 5.5: Priebeh akumulovanej odmeny po epizódach - osi X, Y a Z

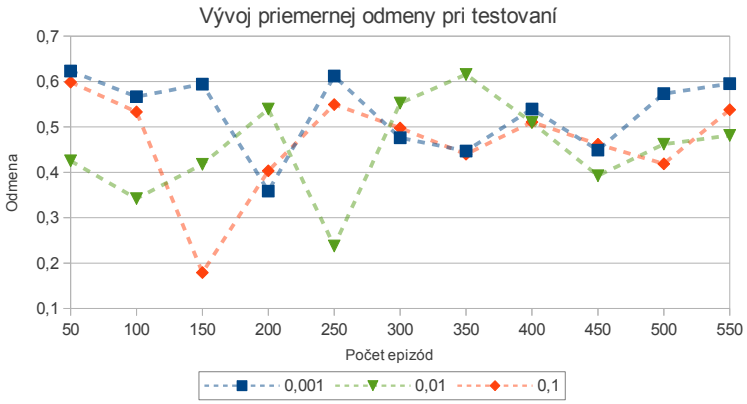


Obr. 5.6: Priebeh akumulovanej odmeny po epizódach - výsledný model

5.1.3 Testovanie modelu

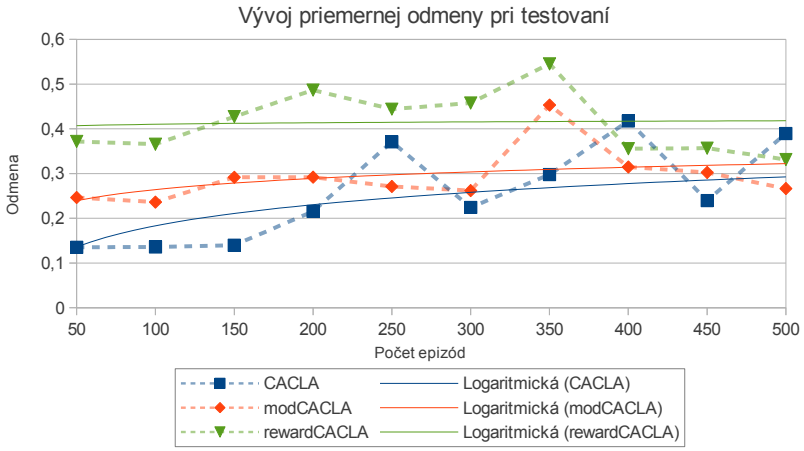
Testovanie prebiehalo po každých 50 epizódach učenia a robot mal postupne dosiahnuť 10 rôznych náhodne vygenerovaných pozícii vo vnútri kvádra trénovacieho priestoru. Graf na obrázku 5.7 znázorňuje vývoj testovacej chyby pri použití siete s 10 neurónmi na skrytej vrstve a rôznych rýchlostiach učenia. Explorační faktor učenia CACLA bol na začiatku nastavený na 0.2 a robot s pravdepobnosťou 40% vykonal v kroku náhodnú zmenu (ignorujúc akciu aktéra). Neskôr sa explorační faktor znížil na 0.1 a pravdepodobnosť náhodného pohybu na 20%. Vidíme, že rýchlosť učenia vo všeobecnosti nemá až taký veľký vplyv na kvalitu učenia. Zároveň je vidieť, že čím vyššia rýchlosť učenia, tým nižšie lokálne minimá výkonnosti boli pri učení dosiahnuté.

Keďže pri učení sme využívali rôzne modifikácie CACLA, analyzovali sme aj ich vplyv na učenie. Pri nasledujúcich experimentoch sme používali 10 neurónov na skrytej vrstve, rýchlosť učenia 0.001, explorační faktor 0.1 a pravdepodobnosť generovania náhodnej akcie 0.2. Graf 5.8 znázorňuje vývoj priemernej odmeny získanej pri testovaní v priebehu 500 trénovacích epizód pre 3 rôzne druhy učenia. Vidíme, že klasická



Obr. 5.7: Testovacia odmena pri rôznych rýchlostiach učenia

CACLA mala zo začiatku problémy, nakoľko kritik nebol ešte dostatočne natrénovaný a robil zlé rozhodnutia o učení aktéra. Následne po 250 epizódach začala výkonnosť siete kolísať avšak so stúpajúcim trendom. Modifikovaný CACLA algoritmus mal lepšiu výkonnosť zo začiatku, po 350 epizódach prudko vzrástla, avšak následne začala opäť klesať a pohybovať sa v blízkosti trendu. Keďže pri učení rewardCACLA (pomenovanie autora) vynechávame kritika, aktéra učíme výhradne pokiaľ akcia viedla k hodnotnejšiemu stavu. Je očakávané, že zo začiatku bude výkonnosť tohto učenia najvyššia. V našom modeli si takéto učenie môžeme dovoliť, pretože dostávame odmenu priebežne po každom kroku a je úmerná úspešnosti každého stavu. V prípade, že by sme si zvolili model, kde by priebežná odmena bola záporná a vysokú odmenu by agent získal až v blízkosti cieľa, nemohli by sme takýto typ učenia použiť. Je vidieť pomerne silnú koreláciu medzi učením rewardCACLA a modCACLA.



Obr. 5.8: Testovacia odmena pri rôznych druhoch CACLA učenia

5.1.4 Porovnanie a využitie modelu

V našej práci sme spomenuli, že sme implementovali aj viacvrstvový perceptrón pre siahanie v priestore, ktorý bol učený s učiteľom na vygenerovanej testovacej množine. Overovali sme tak modul pre siahanie použitý v práci Tikhanoff et al. (2011). Takáto sieť bola schopná siahať na pozície v priestore avšak s pomerne vysokou chybou RMSE, ktorá bola v priemere 0.23. Reálne sme na simulátore odsledovali, že táto sieť zvládala dosiahnuť určité pozície pomerne presne, v iných zas ruka minula cieľovú pozíciu aj o viac ako 30 cm.

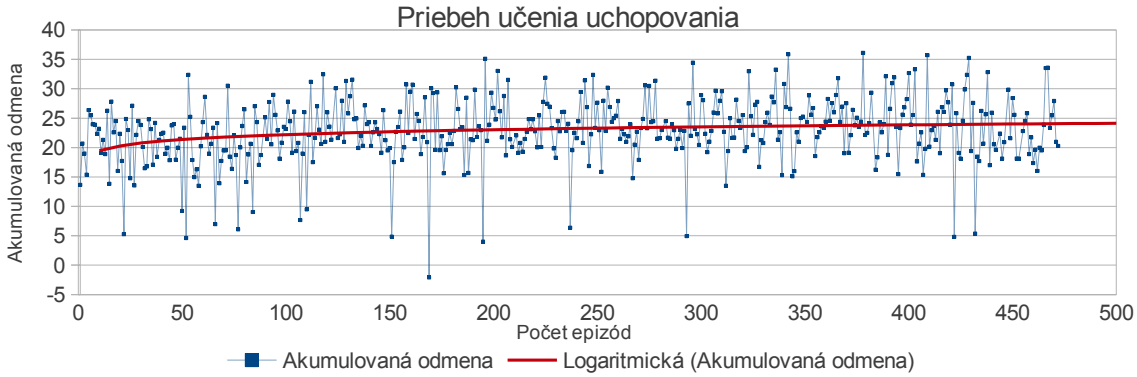
Náš model dosiahol podobné výsledky ako spomínaný viacvrstvový perceptrón, avšak s omnoho nižším počtom trénovacích epizód a s biologicky vierohodným učením s posilňovaním. Pre viacvrstvový perceptrón sa trénovacie príklady generovali náhodným pohybom ramena po priestore a zaznamenávaním informácií o trajektóriach pohybov. Je to podobné explorácii prostredia agentom učiaceho sa učením s posilňovaním. To je podľa nás dôvodom veľmi podobných výsledkov týchto modelov.

Veríme, že po ďalšom vylepšení nášho modelu by ten mohol slúžiť ako motorický modul pre vyššie moduly v rôznych aplikáciách a výskumných prácach, kde bude potrebné pohybovať ramenom v priestore.

5.2 Učenie sa uchopovať objekty

Cieľom modelu pre uchopovanie je v prvej fáze priblížiť ruku k objektu z pozície, ktorú dosiahol model siahania v priestore. Zároveň model musí natočiť ruku tak, aby bolo možné objekt skutočne uchopiť. V druhej fáze potom dochádza k samotnému zvieraniu prstov a uchopeniu objektov, o čom má robot informáciu v podobe haptickej spätnej väzby.

Aj v prípade tohto modelu sme uplatnili sledovanie akumulovanej odmeny. Využili sme architektúru s 10 neurónmi na skrytej vrstve a aktivačnou funkciou hyperbolický tangent. Na grafe 5.9 je vidieť vývoj dosahovanej akumulovanej odmeny pri trénovaní. 1 epizóda bola podobne ako pri siahaní rozdelená na 50 krokov.



Obr. 5.9: Akumulovaná odmena počas trénovania

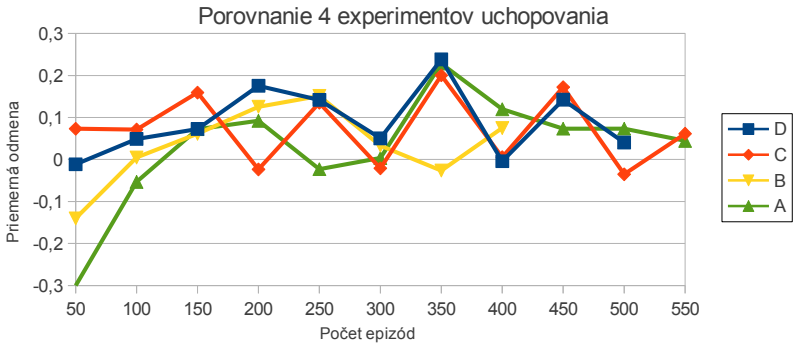
Z našich experimentov bolo ďalej badateľné, že váhovaním tlakových zložiek (palec, dlaň, prsty) sa dal docieliť jav, že danú časť ruky s vyššou váhou (napr. dlaň) pritlačil robot k objektu najskôr. Zaujímavé bolo aj zistenie, že modul pre siahanie nemusí byť nutne úplne bezchybný, pretože model pre uchopovanie dokázal ruku zo vzdialenosti asi 10 cm pritiahnuť k objektu. Myslíme si, že je to dôsledok zakomponovania vzdialenosti od objektu do funkcie odmeny. Pri testoch sme využívali exploračný faktor v intervale $\langle 0.1, 0.25 \rangle$ a diskontný faktor s hodnotami 0, 0.5 a 1.0. Diskontný faktor však ani v tomto modeli nespôsoboval rozdiely.

S modelom uchopovania sme spravili taktiež rôzne experimenty, aby sme sledovali priebeh učenia pri rôznych nastavaniach parametrov. V tabuľke 5.1 uvádzame nastavenia, ktoré sme použili pri 4 experimentoch.

Na obrázku 5.10 vidíme porovnanie vývoja učenia pri opisovaných experimentoch. Zaujímavé je porovnanie experimentov C a D, ktoré sa líšia len v spôsobe učenia.

Tabuľka 5.1: Popis 4 experimentov uchopovania s rôznymi nastaveniami

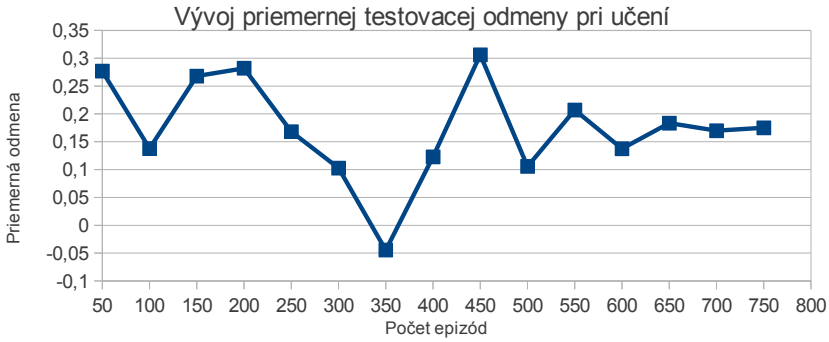
experiment	σ	Π	γ	α_{akter}	α_{kritik}	učenie
A	0.4	0.4	0	0.001	0.01	CACLA
B	0.25	0.2	0	0.01	0.01	CACLA
C	0.2	0.2	1	0.01	0.01	len odmenou
D	0.2	0.2	1	0.01	0.01	CACLA



Obr. 5.10: Vývoj učenia v 4 experimentoch uchopovania

Môžeme sledovať podobný vývoj ako pri modeli siahania v priestore. Učenie výhradne odmenou vykazovalo dobré výsledky už na začiatku učenia, kým učenie bežným algoritmom CACLA dosahovalo spočiatku slabú priemernú testovaciu odmenu (vidíme to ešte viac pri experimentoch A a B, ktoré mali navyše vyšší exploračný faktor).

Niekoľko experimentov sme skúsili trénovať omnoho dlhší počet epizód, aby sme sledovali či pri zníženom exploračnom faktore bude sieť ešte výraznejšie meniť svoju výkonnosť.

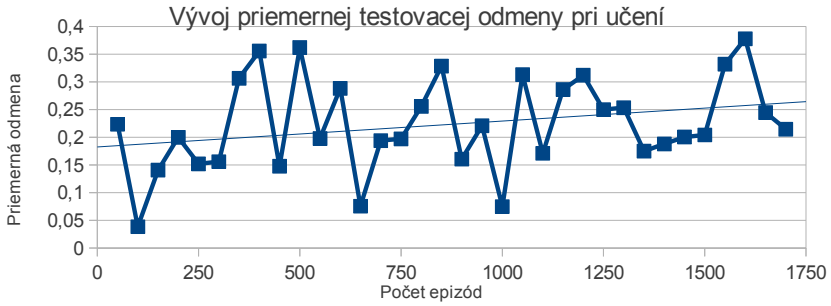


Obr. 5.11: Vývoj naučenia modelu počas 750 epizód

Na obrázku 5.11 je najlepšie vidieť dopad postupného znižovania exploračného fak-

tora. Kým spočiatku učenia má výsledková krivka vysoké výkyvy, neskôr (po asi 500 epizódach) sa model kvôli nízkemu exploračnému faktoru už len dolaďuje. V tomto experimente sme použili klasické učenie CACLA s exploračným faktorom 0.15, pravdepodobnosťou generovania náhodnej akcie 0.15, rýchlosťami učenia 0.01 a diskontným faktorom 0.

Pre ďalší experiment sme sa rozhodli vyskúšať diskontný faktor 0.5 a zvýšiť počiatočný exploračný faktor na 0.4 a taktiež pravdepodobnosť generovania náhodnej akcie na 0.2. Vývoj učenia môžeme vidieť na obrázku 5.12.



Obr. 5.12: Vývoj naučenia modelu počas 1750 epizód

Simulácie potvrdili predpoklady, že robot sa najskôr naučí uchopovať väčší objekt a až neskôr aj menší objekt. Ako prvý sa vyvíja tzv. silový úchop (angl. power grasp) a po dlhšej dobe aj bočný úchop (angl. side grasp). Malé objekty iCub pomocou nášho modelu dokázal uchopiť, avšak nešlo o typický precízny úchop (angl. precision grasp), pretože pri ňom využíval takmer všetky prsty.

Záver

Cieľom našej práce bolo neurálne modelovanie uchopovania objektov v robotickom simulátore iCub. Podarilo sa nám navrhnúť a úspešne implementovať model pre siahanie ramenom v priestore a model uchopovania objektov, ktoré spolupracujú pre úspešné vykonanie akcie. Náš model je navyše v porovnaní s existujúcimi modelmi uchopovania učený biologicky vierohodnou metódou. Využili sme algoritmus učenia s posilňovaním v spojitom priestore stavov a akcií (CACLA), u ktorého sme vyskúšali aj rôzne modifikácie a analyzovali ich dopad na učenie.

Model siahania v priestore generuje zmeny uhlov 4 stupňov voľnosti ramena pre stav zadaný ako vektor absolútnych hodnôt týchto 4 stupňov voľnosti a cieľovú pozíciu v 3D priestore prepočítanú populačným kódovaním. Odmena z prostredia je navrhnutá ako funkcia vzdialenosti ťažiska dlane robota od cieľovej pozície. Do budúcna vidíme priestor na rozšírenie funkcie o odmeny o zložku, ktorá by ohodnocovala natočenie ruky, čo by mohlo viesť k vyššej úspešnosti uchopovania atypických objektov.

Model uchopovania má omnoho zložitejšie definovaný stav. Ten odráža aktuálnu informáciu z tlakových senzorov na dlani a prstoch, informácie o objekte (veľkosť a orientácia) a o vzťahu medzi rukou a objektom (vzdialenosť a natočenie). Aktér generuje vektor absolútnych hodnôt pre použité stupne voľnosti. Funkcia odmeny je opäť zložitejšia ako pri siahaní a skladá sa z 2 zložiek (vzdialenosti od objektu a informácií z dotykových senzorov). Robot sa úspešne naučil uchopovať 3 rôzne druhy objektov (guľa, kocka, kváder-tyč) v rôznych veľkostiach. V práci uvádzame výsledky oboch modelov z mnohých experimentov, kde sme porovnávali použitie rôznych hodnôt parametrov a rôzne typy učenia.

Pri implementácii sme narazili na rôzne problémy so simulátorom iCuba. Tieto sme vyriešili úpravami zdrojového kódu simulátora, ktoré sme v práci dôkladne okomentovali.

Veríme, že práca bude základom budúceho neurálneho motorického modulu pre simulátor iCuba, ktorý bude využívaný v ďalších prácach z oblasti kognitívnej robotiky.

Zoznam použitej literatúry

Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., a Yoshida, C. (2009). Cognitive developmental robotics: A survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):12–34.

Baldi, P. a Hornik, K. (1989). Neural networks and principal component analysis: learning from examples without local minima. *Neural Netw.*, 2(1):53–58.

Bonaiuto, J., Rosta, E., a Arbib, M. (2007). Extending the mirror neuron system model, I. audible actions and invisible grasps. *Biological Cybernetics*, 96(1):9–38.

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Computation*, 12:219–245.

Fagg, A. H. a Arbib, M. A. (1998). Modeling parietal-premotor interactions in primate control of grasping. *Neural Networks*, 11:1277–1303.

Farkaš, I., Malík, T., a Rebrová, K. (2012). Grounding the meanings in sensorimotor behavior using reinforcement learning. *Frontiers in Neurorobotics*, 6. doi:10.3389/fnbot.2012.00001.

Fleischer, F., Casile, A., a Giese, M. (2009). View-independent recognition of grasping actions with a cortex-inspired model. *9th IEEE-RAS International Conference on Humanoid Robots*, pp 514–519.

Griffith, S., Sinapov, J., Miller, M., a Stoytchev, A. (2009). Toward interactive learning of object categories by a robot: A case study with container and non-container objects. In *IEEE 8th International Conference on Development and Learning*, pp 1–6.

Hasselt, H. (2012). Reinforcement learning in continuous state and action spaces. In Wiering, M. a Otterlo, M., editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pp 207–251. Springer Berlin Heidelberg.

Atil I., Dag N., Kalkan S. a Sahin E. (2010). Affordances and emergence of concepts. *10th International Conference on Epigenetic Robotics*.

Kawato, M. a Oztop, E. (2009). *Models for the Control of Grasping*. Cambridge University Press.

Kawato, M. a Samejima, K. (2007). Efficient reinforcement learning: computational theories, neuroscience and robotics. *Current Opinion in Neurobiology*, 17(2):205–212.

Korenčiak, R. (2010). Aproximácia motorického priestoru ramena simulovaného robota. Diplomová práca, FMFI UK, Bratislava.

Kraft, D., Detry, R., Pugeault, N., Basandeski, E., Guerin, F., Piater, J., a Kruandger, N. (2010). Development of object and grasping knowledge by robot exploration. *IEEE Transactions on Autonomous Mental Development*, 2(4):368–383.

Malík, T. (2011). Neurálny model integrácie jazykových a motorických znalostí simulovaného agenta. Diplomová práca, FMFI UK, Bratislava.

McClelland, J. L. a Rumelhart, D. E., editors (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT Press, Cambridge, MA.

Montesano, L. a Lopes, M. (2009). Learning grasping affordances from local visual descriptors. In *Proceedings of the 2009 IEEE 8th International Conference on Development and Learning*, DEVLRN '09, pp 1–6, Washington, DC, USA. IEEE Computer Society.

Oztop, E. a Arbib, M. A. (2002). Schema design and implementation of the grasp-related mirror neuron system. *Biological Cybernetics*, 82(2):116–140.

Oztop, E. a Arbib, M. A. (2004). Infant grasp learning: a computational model. *Experimental Brain Research*, pp 480–503.

Pouget, A., Dyan, T., and Zemel, R. (2000). Information processing with population codes. *Nature Reviews Neuroscience*, 1:125–132.

RobotCub (2012). icub manual - wiki for robotcub and friends. `http://eris.liralab.it/wiki/Manual`.

Schenck, W., Hoffmann, H., a Möller, R. (2011). Grasping of extrafoveal targets: A robotic model. *New Ideas in Psychology*, 29(3):235–259.

Smith, R. (2006). Open dynamics engine user guide 0.5. `http://www.ode.org/ode-latest-userguide.html#sec_5_1_0`.

Sutton, R. a Barto, A. (1998). *Reinforcement Learning: an Introduction*. Adaptive computation and machine learning. MIT Press.

Thomaz, A. L. a Cakmak, M. (2009). Learning about objects with human teachers. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction*, HRI '09, pp 15–22, New York, NY, USA. ACM.

Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L., a Nori, F. (2008). An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pp 57–61, New York, NY, USA. ACM.

Tikhanoff, V., Cangelosi, A., a Metta, G. (2011). Integration of speech and action in humanoid robots: icub simulation experiments. *IEEE Transactions on Autonomous Mental Development*, 3(1):17–29.

Príloha A - CD

K diplomovej práci prikladáme CD médium so všetkými zdrojovými kódmi vrátane dokumentácie. Dokumentácia bola vygenerovaná priamo z okomentovaného zdrojového kódu pomocou programu Doxygen a je dostupná vo verzii HTML a PDF. Ďalej sa na disku nachádza elektronická verzia diplomovej práce.