UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

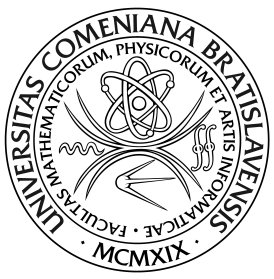
Vizuálno-motorická integrácia v reálnom robotickom prostredí

Diplomová práca

2014

Bc. Tomáš Štibraný

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



Vizuálno-motorická integrácia v reálnom robotickom prostredí

Diplomová práca

Študijný program: Aplikovaná informatika

Študijný odbor: 2511 Aplikovaná informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Školiteľ: doc. Ing. Igor Farkaš, PhD.

Bratislava 2014

Bc. Tomáš Štibraný

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Štibraný
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

Názov: Vizuálno-motorická integrácia v reálnom robotickom prostredí

Cieľ: S využitím robotického ramena (AL5D, Lynxmotion) a stereo vizuálneho systému (SVS, Surveyor) vytvorte robotický systém, aby sa naučil prepojiť vizuálnu informáciu o polohe koncovej časti vlastného ramena s proprioceptívnou informáciou. Učiaci sa model realizujte pomocou obojsmernej neurónovej siete a otestuje ho v reálnom nasadení.

Literatúra: O'Reilly, R.C. (1996). Biologically Plausible Error-driven Learning using Local Activation Differences: The Generalized Recirculation Algorithm. Neural Computation, 8, 895-938.
Farkaš I., Rebrová K. (2013). Bidirectional activation-based neural network learning algorithm. In Proceedings of the International Conference on Artificial Neural Networks (ICANN), Springer. 154-161.

Anotácia: Motiváciou pre túto prácu je využitie biologicky inšpirovaných učiacich mechanizmov pre neurónové siete, konkrétne učenia s učiteľom, ktoré možno využiť pri učení vizuálno-motorického zobrazenia.

Poznámka: Požiadavky: pasívna znalosť angličtiny, záujem o kognitívnu robotiku, relatívna samostatnosť, ochota pracovať priebežne

Kľúčové slová: vizuálny systém, robotické rameno, neurónová sieť, učenie, vizuálno-motorická integrácia

Vedúci: doc. Ing. Igor Farkaš, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 16.12.2011

Dátum schválenia: 06.05.2014

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

.....
študent

.....
vedúci práce

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

……

Za cenné rady a konzultácie ďakujem svojmu vedúcemu diplomovej práce doc. Ing. Igorovi Farkašovi, PhD a za pomoc pri práci s robotickým ramenom a stereo vizuálnym systémom ďakujem Mgr. Pavelovi Petrovičovi, PhD. Taktiež ďakujem svojmu blízkemu okoliu za neustálu podporu.

Abstrakt

V našej práci riešime jeden z problémov kognitívnej robotiky, ktorým je prepojenie medzi proprioceptívnou a vizuálnou informáciou, ktoré možno využiť pri siahaní na objekty. Učiaci model sme realizovali pomocou obojsmernej neurónovej siete s využitím algoritmu BAL, ktorý je založený na obojsmernom šírení aktivácii. Pracovali sme so syntetickou množinou dát, ale aj s reálnou zozbieranou pomocou robotického ramena AL5D a stereo vizuálneho systému Surveyor. V práci prezentujeme výsledky našich experimentov s oboma množinami dát, ale aj spôsoby, akými sme spracovávali reálne dáta z robotického ramena a stereo vizuálneho systému a akými sme generovali syntetické dáta.

Kľúčové slová: siahanie na objekty, robotické rameno, stereo vizuálny systém, kognitívna robotika, BAL

Abstract

In our thesis we are solving one of the problems from cognitive robotics, which is connecting proprioceptive and visual information for usage in object reaching. Learning model is implemented using truly bidirectional neural net trained by BAL learning algorithm, which propagates activations both ways through network. We were working with two data sets: generated one and real one. Real data set was collected using AL5D robotic arm and Surveyor stereo visual system. We present findings and results from our experiments with both sets of data. Also we present how we did process real data from robotic arm and stereo visual system and how we did generate synthetic set of data.

Keywords: object reaching, robotic arm, stereo visual system, cognitive robotics, BAL

Register tabuliek

Register obrázkov

Contents

Kapitola 1

Úvod

Väčšina robotov používaných v praxi v dnešnej dobe nevykazuje inteligentné správanie. Predstavme si napríklad robotické ramená na výrobných továrenských linkách v automobilovom priemysle, alebo aj robotické vysávače. Nevedia sa učiť, nemajú žiadnu náhradu pamäte v takom zmysle, v akom ju poznáme napríklad u človeka. Majú dopredu popísané, ako sa majú hýbať a len "slepo" vykonávajú tieto inštrukcie. Činnosti, ktoré tieto roboty vykonávajú, do nich boli natvrdo vložené programátorom. Takéto roboty často ani nevnímajú svoje okolie, iba vykonávajú jednu konkrétnu činnosť.

Správanie (činnosti) týchto robotov môžu byť komplikované a z pohľadu tretej osoby tak môže správanie robota vyvolávať dojem inteligencie, ale táto inteligencia nie je vlastná robotovi, bola do neho vložená.

Podmnožina robotiky s názvom kognitívna robotika sa snaží o tvorbu robotov, ktorých inteligentné správanie je ich vlastné, nie je do robota vložené programátorom, ale je naučené samotným robotom pri jeho interakcii s prostredím. Programátor do robota vloží len schopnosť učiť sa a robot sa pomocou interakcie so svojím prostredím učí sám. Ak takýto robot vykazuje známky inteligencie, dá sa naozaj hovoriť o inteligentnom robotovi, lebo toto inteligentné správanie si robot osvojil sám.

Jeden z možných prístupov k tvorbe softvéru takýchto robotov je snažiť sa použiť už existujúci model, o ktorom vieme, že funguje, a tým je ľudský mozog. Môžeme použiť poznatky z neurovedy a z kognitívnej vedy a na ich základe zostaviť model učenia sa, ktorý má biologický základ. Takže kognitívna robotika sa môže rozvíjať aj vďaka kognitívnym vedám.

Na tento vzťah kognitívnej robotiky a kognitívnych vied sa však dá pozerať aj obrátene. Nielen kognitívna robotika čerpá z kognitívnych vied, ale aj naopak. Pomocou kognitívnej

robotiky vieme buď overovať, ale aj porovnávať teórie kognitívnej vedy. Po implementácii modelu učenia do robota vieme porovnať správanie robota s tým ľudským (alebo zvieracím) a môžeme tak overiť našu teóriu v praxi [1]. Kognitívna robotika má teda medzi iným aj veľký význam pri spoznávaní vývinu človeka - jedinca.

V tejto práci prezentujeme biologicky inšpirovaný model založený na učiacom algoritme BAL, určený na prepojenie vizuálnej informácie s proprioceptívnou informáciou využívajúc stereo vizuálny systém a robotické rameno a prezentujeme jeho dosiahnuté výsledky.

Kapitola 2

Použité metódy a algoritmy

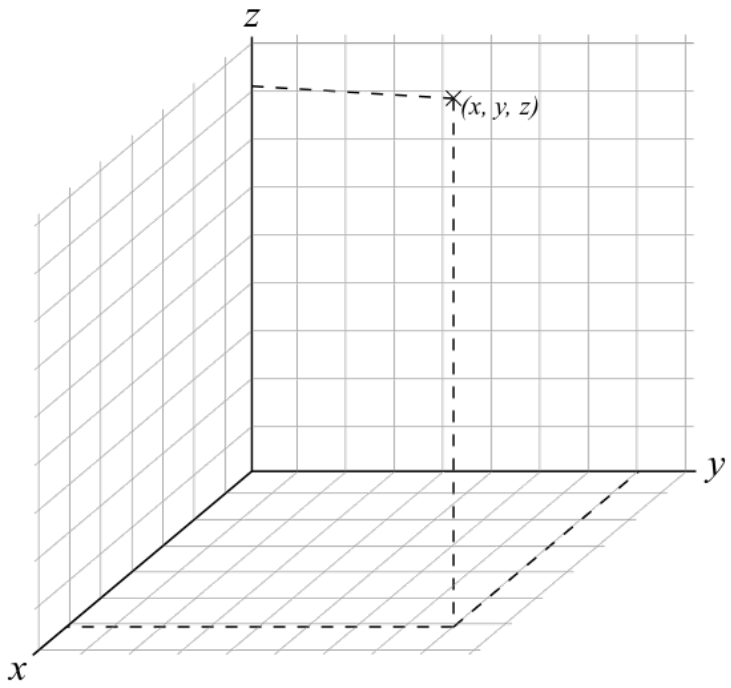
Cieľom našej práce bolo prepojiť robotické rameno s dvoma kamerami pomocou obojsmernej neurónovej siete schopnou naučiť sa súvislosť medzi vizuálnou informáciou z prostredia a proprioceptívnou informáciou robotického ramena (servo kontrolera). Cieľom našej práce bolo aj použiť biologicky prijateľný učiaci algoritmus pre neurónovú sieť. Vizuálnu informáciu sme spracovávali pomocou knižnice Open CV.

2.1 Sférická súradnicová sústava

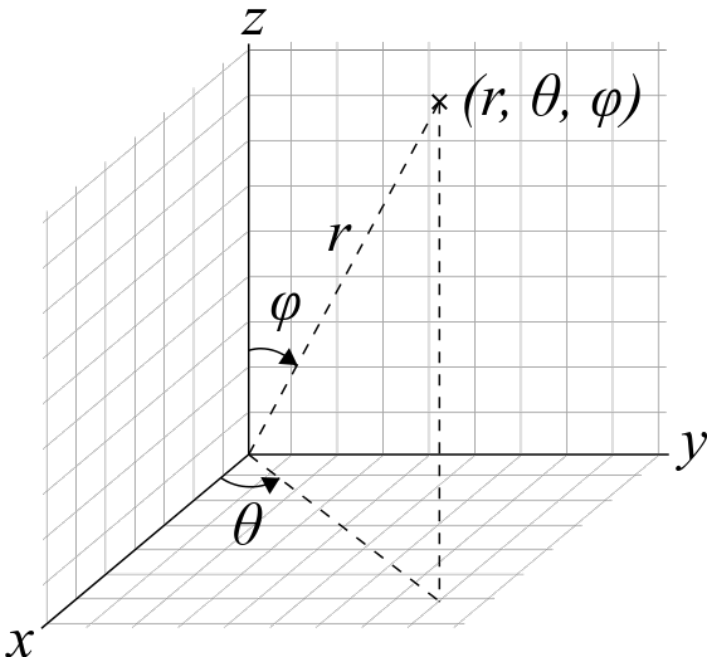
Karteziánska súradnicová sústava je sústava, v ktorej je každý bod jednoznačne špecifikovaný trojicou (dvojicou v 2D) číselných hodnôt, ktoré sú orientované vzdialenosti od tohto bodu k trom (dvom v 2D) fixným, na seba kolmým a orientovaným priamkam. Tieto referenčné priamky sa nazývajú osi. Karteziánska súradnicová sústava je v prírodných vedách najpoužívanejšou súradnicovou sústavou.

Sférická súradnicová sústava je 3D súradnicová sústava, v ktorej je každý bod jednoznačne špecifikovaný trojicou číselných hodnôt[12]:

1. $\mathtt{r}$: radiálnou vzdialenosťou bodu od počiatku súradnicovej sústavy
2. ϕ: polárnym uhlom meraným vzhľadom na fixný zenit
3. θ: orientovaným uhlom medzi zvoleným smerom na referenčnej rovine a čiarou prechádzajúcou ortogonálnou projekciou bodu na referenčnú rovinu a počiatkom súradnicovej sústavy, kde referenčná rovina je rovina, ktorá je kolmá na zenit a prechádza počiatkom súradnicovej sústavy.



Obrázok 2.1: Karteziánska súradnicová sústava.



Obrázok 2.2: Sférická súradnicová sústava.

2.2 Učenie s učiteľom

Existujú tri rôzne typy učenia: učenie s učiteľom, učenie bez učiteľa a učenie s posilňovaním. Pri učení bez učiteľa nemáme možnosť trénovať hypotézu, ale môžeme napríklad klasterizovať vstupy na základe podobnosti, alebo redukovať dimenzionalitu vstupov. Pri učení s posilňovaním máme na trénovanie hypotézy k dispozícii odozvu z prostredia a náš učiaci algoritmus sa snaží maximalizovať svoju odmenu za vykonanú akciu. Odozvu z prostredia nám sprostredkuje učiteľ.

Pri učení s učiteľom rozoznávame dve rôzne úlohy: regresiu a klasifikáciu. Pri klasifikácii sa snažíme učiaci algoritmus natrénovať rozhodovanie o príslušnosti do skupiny na základe príznačných vlastností objektu. Pri regresnej úlohe chceme aby sa učiaci algoritmus naučil odhadovať hodnoty funkcie, ktoré závisia od príznačných vlastností objektu.

Pri učení s učiteľom nám prostredie poskytuje príznačné vlastnosti objektov, a narozdiel od učenia bez učiteľa, nám poskytuje aj skupinu, do ktorej objekt patrí pri klasifikačnej úlohe, respektíve ohodnotenie daného objektu, alebo javu pri regresnej úlohe.

2.3 Algoritmus spätného šírenia chyby

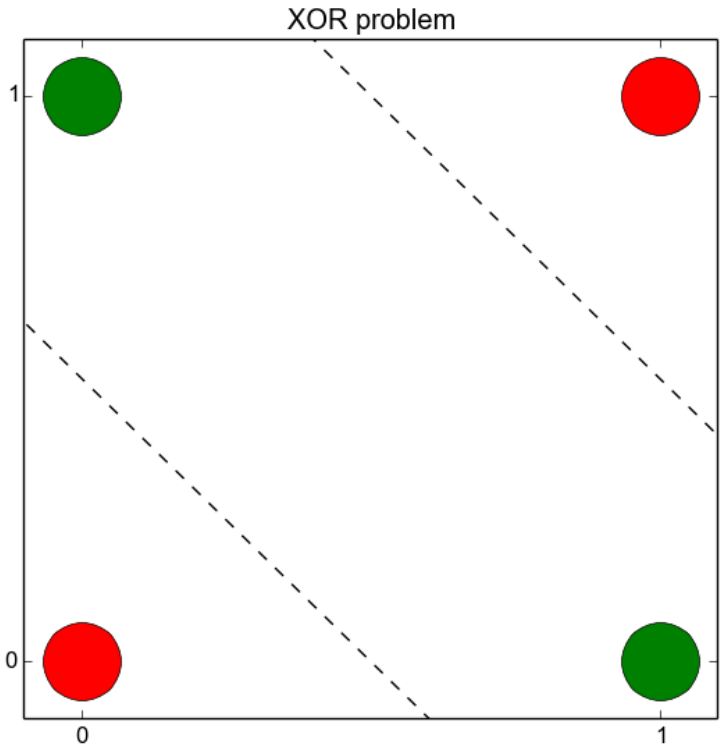
Typickým príkladom algoritmu učenia s učiteľom je algoritmus spätného šírenia chyby (back-propagation), ktorým sa učí dopredná neurónová sieť[1] nazývaná viacvrstvový perceptrón (multilayer perceptron - MLP).

Narozdiel od jednoduchého perceptrónu, ktorý má len vstupnú a výstupnú vrstvu sa pri algoritme spätného šírenia chyby používajú aspoň tri vrstvy: vstupná, výstupná a niekoľko skrytých vrstiev, ktoré sa nachádzajú medzi vstupnou a výstupnou vrstvou. Skryté vrstvy reprezentujú vnútorný stav siete, ktorý sa v jednoduchom perceptróne nenachádza. Perceptrónu musí na trénovanie siete stačiť také kódovanie vlastností objektov, aké dostane na vstupe[10]. Preto je schopný naučiť sa len lineárne separovateľné problémy, ale väčšina problémov z praxe takéto nie sú.

Algoritmus spätného šírenia chyby je vďaka vnútornej reprezentácii (použitie skrytej vrstvy) a zovšeobecneného delta pravidla (schopnosť odvodiť chybu pre všetky vrstvy, nie len výstupnú) schopný naučiť sa aj lineárne neseparovateľné problémy.

Učenie spätným šírením chyby prebieha v dvoch fázach. V prvej fáze sa signál šíri sieťou od

[1] Existujú modifikácie tohto algoritmu, ktorými sa učia rekurentné siete.



Obrázok 2.3: XOR - príklad lineárne neseparovateľného problému

Vrstva	Vážený vstup	Hodnota aktivácie
vstupná (s)	-	s_i = stimul
skrytá (h)	$net_j = \sum_i w_{ij} s_i$	$h_j = f(net_j)$
výstupná (o)	$net_k = \sum_j w_{jk} h_j$	$o_k = f(net_k)$

Tabuľka 2.1: Šírenie signálu v trojvrstvovej sieti učenej pomocou algoritmu spätného šírenia chyby. f(x) vyznačuje zvolenú aktivačnú funkciu [6].

vstupných neurónov k výstupným, hodnoty aktivácii neurónov na výstupnej vrstve porovnáme s očakávanými hodnotami. V druhej fáze vypočítame pre každú váhu jej podiel na chybe výstupnej vrstvy a pomocou tohto podielu vypočítame zmenu váh. Tento podiel sa počíta rozdielne pre váhy smerujúce priamo do výstupnej vrstvy a pre ostatné váhy. Chyba, ktorú spôsobila váha smerujúca do neurónu na výstupnej vrstve sa dá vypočítať priamočiaro ako rozdiel skutočného výstupu a cieleného výstupu. Ostatné váhy vypočítame postupne od váh smerujúcich

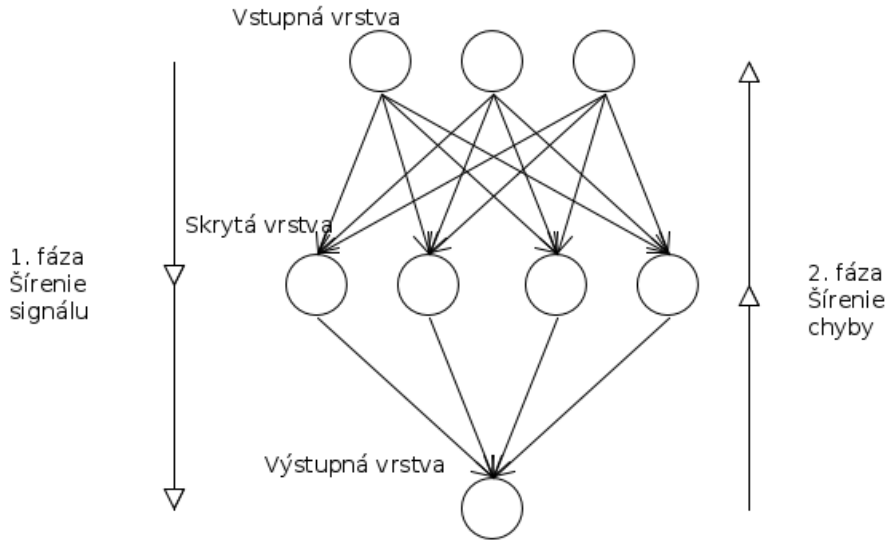
do predposlednej vrstvy až po váhy smerujúce do prvej skrytej vrstvy. Pre tieto váhy vieme určiť ich podiel na chybe podľa podielu chyby na neuróne do ktorého smerujú a sile tejto váhy. Konkrétne vzorce pre výpočet zmeny váh sú nasledovné:

- váhy smerujúce do výstupnej vrstvy

$$\Delta w_{jk} = \alpha * (t_k - o_k) * h_j$$

- váhy smerujúce do skrytej vrstvy

$$\Delta w_{ij} = \alpha * (\sum_k (t_k - o_k) w_{jk}) * \bar{f}(net_j) * s_i$$



Obrázok 2.4: Znázornenie priebehu šírenia signálu a chyby pri učení algoritmom spätného šírenia chyby.

Známym problémom algoritmu spätného šírenia chyby sú jeho biologické "nepresnosti". Tento algoritmus vyžaduje, aby sa chyba na výstupnej vrstve počítala pomocou derivácie aktivačnej funkcie a potom sa táto chyba šíri po neurónoch špeciálnym spôsobom, nie pomocou aktivácii neurónov.[2, 13, 6]

2.4 Generalized Recirculation Algorithm

GeneRec narozdiel od algoritmu spätného šírenia chyby vykonáva svoje učenie len na základe lokálne dostupných informácii (aktivácie neurónov) a tým pádom je biologicky prijateľnejší.

GeneRec vyžaduje schopnosť siete šíriť signál oboma smermi, to znamená zo vstupu na výstup, ale aj obrátene, z výstupu na vstup a symetrickú inicializáciu váh ($w_{pq} = w_{pq}$). Šírenie signálu prebieha v dvoch fázach. Najprv zafixujeme vstupné hodnoty ako aktivácie vstupnej vrstvy a ostatné aktivácie označíme za nulové a potom opakovane šírime signál zo vstupu na výstup a späť, až kým zmeny aktivácii nie sú dostatočne nízke.

Vrstva	**Fáza**	**Vážený vstup**	**Hodnota aktivácie**
vstupná (s)	-	-	$s_i =$ stimul
skrytá (h)	mínus	$net_j^- = \sum_i w_{ij} s_i + \sum_k w_{kj} o_k^-$	$h_j^- = \sigma(net_j^-)$
	plus	$net_j^+ = \sum_i w_{ij} s_i + \sum_k w_{kj} o_k^+$	$h_j^+ = \sigma(net_j^+)$
výstupná (o)	mínus	$net_k^- = \sum_j w_{jk} h_j^-$	$o_k^- = \sigma(net_k^-)$
	plus	-	$o_k^+ = t_k$

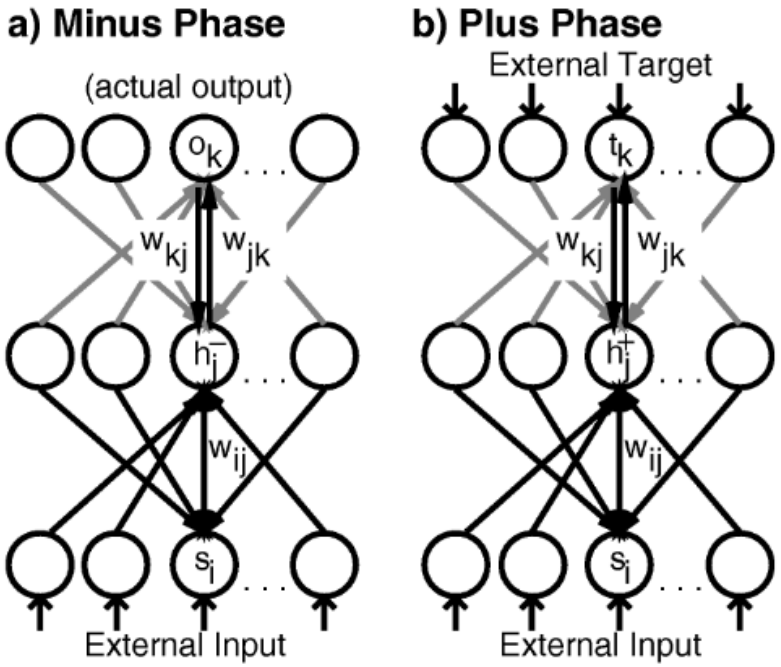
Tabuľka 2.2: Rovnice pre šírenie signálu v trojvrstvovej sieti učenej pomocou GeneRec algoritmu [6].

Učenie prebieha v troch fázach. V mínusovej fáze zafixujeme hodnoty aktivácii vstupných neurónov na stimul a aktivácie ostatných neurónov nastavíme na nulu. V plusovej fáze zafixujeme nie len vstupnú vrstvu na stimul, ale aj výstupnú vrstvu na cieľové hodnoty. V poslednej fáze, pri učení, zmeníme všetky váhy (okrem váh smerujúcich od skrytej do vstupnej vrstvy). Pravidlo pre zmenu váh, ak a p je neurón, ktorý posiela signál a a q je neurón, ktorý prijíma signál:

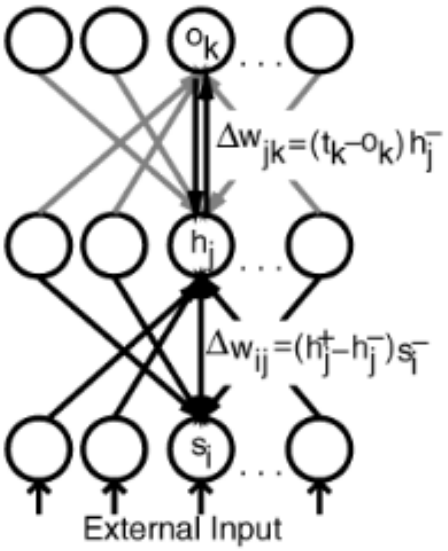
$$\Delta w_{pq} = \alpha * (a_q^+ - a_q^-) * a_p^-$$

Základné GeneRec pravidlo pre zmenu váh sa dá rozdeliť na dve časti. Jednou je výpočet chyby pre konkrétny neurón ($a_q^+ - a_q^-$) a druhou je podiel, akým je konkrétna váha za túto chybu zodpovedná (aktivácia neurónu a_q^-). Modifikovaním výpočtu podielu chyby tak, aby algoritmus bral do úvahy aj očakávanú hodnotu aktivácie neurónu a_p po upravení váh dostaneme midpoint GeneRec algoritmus:

$$\Delta w_{pq} = \alpha * (a_q^+ - a_q^-) * \frac{1}{2} * (a_p^- + a_p^+)$$



Obrázok 2.5: Znázornenie šírenia signálu počas plusovej a mínusovej aktivačnej fázy GeneRec algoritmu[7].



Obrázok 2.6: Znázornenie priebehu učenia GeneRec algoritmu[7].

Ďalšou možnou modifikáciou je upravenie algoritmu, aby zachovával symetrické váhy aj počas učenia. Toto dosiahneme nasledovnou modifikáciou učiaceho pravidla:

$$\Delta w_{pq} = \alpha * (a_p^- * (a_q^+ - a_q^-)) + a_q^- * (a_p^+ - a_p^-))$$

Aplikovaním oboch GeneRec modifikácii naraz dostaneme presne učiace pravidlo pre Contrastive Hebbian Learning algoritmus (CHL):

$$\Delta w_{pq} = \alpha * (a_p^+ a_q^+ - a_p^- a_q^-)$$

2.5 Bidirectional Activation-based Learning algorithm

Bidirectional Activation based Neural Network Learning Algorithm alebo v skratke BAL, je algoritmus na učenie obojsmernej neurónovej siete vychádzajúci z GeneRec-u [3]. BAL, aj keď má aj spätné váhy, narozdiel od GeneRecu nepoužíva dynamické ustálenie signálu. Signál sa šíri v jednom kroku z prvej vrstvy do poslednej vrstvy a takisto v jednom kroku z vrstvy poslednej do vrstvy prvej. Keďže BAL sa učí zobrazovať aj poslednú vrstvu do prvej, nie len „tradične“ prvú do poslednej, zámerne sa pri ňom vyhýbame označeniam vstupná a výstupná vrstva.

Vrstva	Fáza	Vážený vstup	Hodnota aktivácie
vstupná (s)	F	-	$x_i^F =$ stimul
	B	$net_i^B = \sum_j w_{ji} h_j^B$	$x_i^B = \sigma(net_i^B)$
skrytá (h)	F	$net_j^F = \sum_i w_{ij} x_i^F$	$h_j^F = \sigma(net_j^F)$
	B	$net_j^B = \sum_k w_{kj} y_k^B$	$h_j^B = \sigma(net_j^B)$
výstupná (o)	F	$net_k^F = \sum_j w_{jk} h_j^F$	$y_k^F = \sigma(net_k^F)$
	B	-	$y_k^B =$ stimul

Tabuľka 2.3: Rovnice pre šírenie signálu v trojvrstvovej sieti trénovanej učiacim algoritmom BAL [3].

Všetky váhy v sieti sa pri BAL algoritme inicializujú na náhodnú hodnotu s normálnej distribúcie $\mathcal{N}(0, \sqrt{n_I + 1})$, kde n_I symbolizuje dimenziu vstupného vektoru.

Trénovanie prebieha takisto ako pri GeneRec algoritme v troch fázach. V prvej "doprednej" fáze sa signál šíri z prvej vrstvy do poslednej. V druhej "spätnej" fáze sa signál šíri z poslednej vrstvy do prvej. Takto získané aktivácie neurónov sa potom použijú v tretej fáze na výpočet zmeny váh nasledujúcimi učiacimi pravidlami:

- pravidlo pre váhy v smere od prvej vrstvy po poslednú

$$\Delta w_{pq}^{F} = \alpha * a_{p}^{F} * (a_{q}^{B} - a_{q}^{F})$$

- pravidlo pre váhy v smere od poslednej vrstvy po prvú

$$\Delta w_{pq}^{B} = \alpha * a_{p}^{B} * (a_{q}^{F} - a_{q}^{B})$$

2.6 Denavit-Hartenberg transformácie

Častou formou robotického manipulátora je robotické rameno. Skladá sa z pohyblivých kĺbov a z pevných spojov medzi kĺbmi, ktorých postupnosť vytvára takzvané kinematické zreťazenie. Každým kĺbom získava robot niekoľko stupňov voľnosti. Pre jednoduchosť konštrukcie sa v praxi používajú kĺby s jedným stupňom voľnosti a zložitejšie kĺby sa simulujú spojením viacerých kĺbov do jedného zložitejšieho kĺbu (ako je to aj pri použitom ramene AL5D). V 3D priestore poznáme tri druhy rotačných stupňov voľnosti a tri druhy posuvných stupňov voľnosti, t.j. tri stupne voľnosti okolo každej z osí X, Y a Z.

Poloha koncového efektora robota v 3D priestore je určená momentálnym stavom kĺbov, dĺžkami spojov medzi kĺbmi a spôsobmi ich spojenia. Tieto parametre sú v robotike štandardizované Denavit-Hartenberg konvenciou a vieme pomocou nich opísať každý kĺb.

Podľa Denavit-Hartenberg konvencie priraďujeme každému kĺbu svoju vlastnú súradnicovú sústavu, ktorá je určená transformačnou maticou definovanou vlastnosťami spoja $[X]$, transformačnou maticou definovanou natočením kĺbu $[Z]$ a predchádzajúcou súradnicovou sústavou. Ak poznáme transformačné matice všetkých spojov a všetkých kĺbov, tak polohu $[T]$ koncového efektora vypočítame takto:

$$[T] = [Z_1][X_1][Z_2][X_2] \ldots [Z_n][X_n]$$

Súradnicová sústava n-tého kĺbu má takéto vlastnosti:

1. os Z_n určuje smer nasledujúcemu spoju
2. os X_n je kolmá na rovinu určenú osami Z_n a Z_{n+1}
3. os Y_n je definovaná osami X_n a Z_n tak, aby spolu vytvárali pravotočivú súradnicovú sústavu

Denavit-Hartenberg definícia kĺbu zahŕňa štyri parametre, ktoré sú označované ako Denavit-Hartenberg parametre. Sú to uhly α a θ a dĺžky d a r. α je uhol medzi predchádzajúcou a aktuálnou osou Z, θ je otočenie okolo predchádzajúcej osi Z, d je dĺžka pozdĺž predchádzajúcej osi Z a r je vzdialenosť predchádzajúcej a nasledujúcej osi Z na normále $Z_n \times Z_{n+1}$, ktorú spolu definujú.

Transformačné matice spoja a kĺbu su pomocou α, θ, d a r definované takto:

$$[Z_i] = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & 0 \\ \sin(\theta_i) & \cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[X_i] = \begin{bmatrix} 1 & 0 & 0 & r_{i,i+1} \\ 0 & \cos(\alpha_{i,i+1}) & -\sin(\alpha_{i,i+1}) & 0 \\ 0 & \sin(\alpha_{i,i+1}) & \cos(\alpha_{i,i+1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Celková transformácia spôsobená kĺbom a spojom, ktorý do neho vchádza je potom definovaná všetkými Denavit-Hartenberg parametrami takto:

$$[T_i] = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_{i,i+1}) & \sin(\theta_i)\sin(\alpha_{i,i+1}) & r_{i,i+1}\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_{i,i+1}) & -\cos(\theta_i)\sin(\alpha_{i,i+1}) & r_{i,i+1}\sin(\theta_i) \\ 0 & \sin(\alpha_{i,i+1}) & \cos(\alpha_{i,i+1}) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.7 Spracovanie obrazu

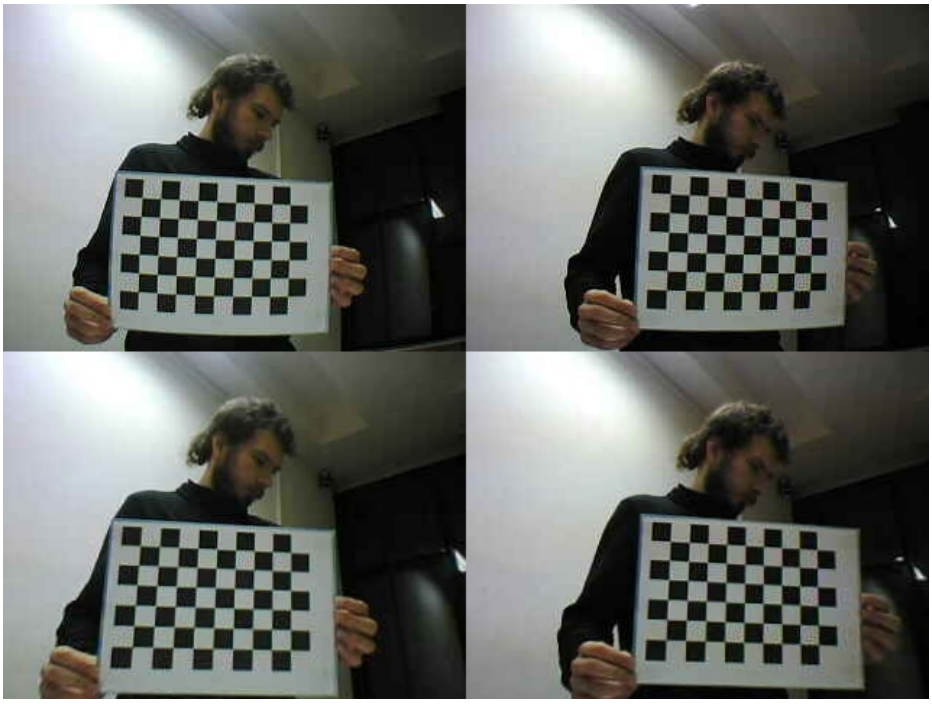
Na spracovanie získaného obrazu sme používali opensource grafickú knižnicu OpenCV, ktorá poskytuje implementáciu algoritmov nielen z oblasti počítačového videnia.

2.7.1 Kalibrácia kamery

Obrázky nasnímané dierkovými kamerami, ktoré používajú šošovky trpia skreslením, ktoré je spôsobené nedokonalosťou šošoviek. Existuje viac druhov skreslení a OpenCV vie obrázky zbavovať radiálneho a tangenciálneho skreslenia pomocou funkcie `undistort`. Táto funkcia však potrebuje vedieť parametre kamery. Tieto sa dajú zistiť (vypočítať) procesom, ktorý

voláme kalibrácia kamery. Kamera sa kalibruje napríklad pomocou obrázku šachovnice, alebo obrázku sústredných kruhov.

Pri kalibrácii kamery pomocou obrázku šachovnice potrebujeme o tomto obrázku vedieť jeho presné rozmery políčok a rozmery celej šachovnice. Samotná kalibrácia používa funkciu `findChessboardCorners` na hľadanie súradníc rohov políčok v obrázkoch, ktoré následne porovnáva s reálnymi súradnicami týchto rohov. Tieto si vieme jednoducho vypočítať z veľkosti šachovnice a veľkosti políčok. Dvojice nájdených a reálnych súradníc rohov políčok spracuje funkcia `calibrateCamera`, ktorá vypočíta kamerovú maticu a koeficienty skreslenia.



Obrázok 2.7: Ukážka odstránenia skreslenia podľa parametrov vypočítaných funkciou calibrateCamera. Vo vrchnom rade sú stereo obrázky pred kalibráciou a v spodnom rade sú obrázky po kalibrácii.

2.7.2 Kalibrácia stereo kamery

Kalibrácia stereo kamery sa veľmi nelíši od kalibrácie jednej kamery. Pri kalibrácii stereo kamery nás nezaujímajú iba matica kamery a koeficienty skreslenia šošovky, ale aj vzťah dvoch kamier v priestore a teda ich vzájomné natočenie a posun medzi nimi. Na stereo kalibráciu sme použili funkciu `stereoCalibrate` z knižnice OpenCV. Algoritmus dosahuje lepšie výsledky,

keď najprv nakalibrujeme obe kamery zvlášť pomocou `calibrateCamera` a až potom pustíme stereo kalibráciu kamery so zapatým flagom `CALIB_FIX_INTRINSIC`.

2.7.3 Hľadanie objektu podľa farby

Pri spracovaný obrazu za účelom hľadania objektu (objektov) na obrázku sa často používa prahovanie (thresholding). Základnou ideou prahovania je pozerať sa na obrázok[2] ako na pole pixlov a odfiltrovať pôvodný obrázok na nový, ktorého pixle budú buď čierne, biele alebo budú mať pôvodnú hodnotu[4, p. 135-141].

2.7.4 Hľadanie kontúr

Hľadanie kontúr je proces pri ktorom sa z binárneho obrázku extrahujú vonkajšie hranice objektov. V OpenCV slúži na tento účel funkcia `findContours`, ktorá vráti pre každý rozoznaný objekt vo vstupnom binárnom obrázku pole bodov, ktoré definujú hranice tohto objektu. Pre hranice objektu vieme pomocou funkcie `contourArea` vypočítať obsah objektu na našom obrázku (počet pixlov, ktorý objekt zaberá) a tiež vieme pomocou funkcie `pointPolygonTest` zistiť, či je arbitrárny bod súčasťou objektu, alebo nie.

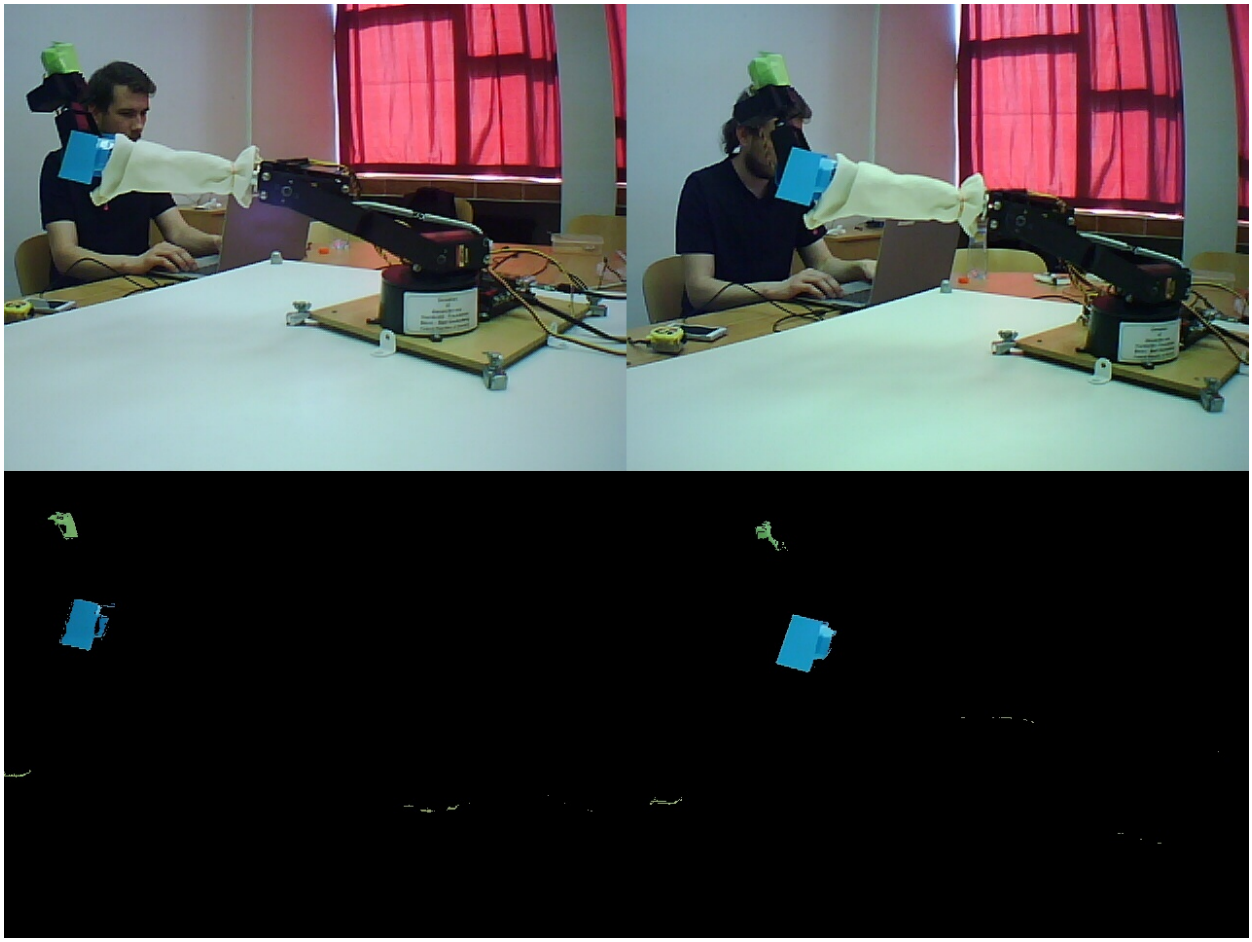
2.7.5 3D rekonštrukcia

3D rekonštrukcia je proces získavania súradníc objektov v reálnom svete (3D súradnice) z viacerých rozdielnych obrázkov toho istého objektu v scéne. Aby sme vedeli počítať súradnice objektov, musíme naviac poznať aj vzťah medzi týmito obrázkami, tj. transformačné matice, ktoré transformujú pohľad na našu scénu v jednom obrázku na pohľad na scénu v obrázku druhom. Samotná 3D rekonštrukcia prebieha v štyroch krokoch[4]:

0. Pomocou stereo kalibrácie získame matice kamier a koeficienty skreslenia pre obe kamery a vzájomnú polohu kamier v priestore.

1. Získame obrázky scény z oboch kamier a pomocou koeficientov skreslenia šošoviek odstránime radiálne a tangenciálne skreslenie z obrázkov.

[2] Prahovanie sa používa na HSV obrázky (Hue - farba, Saturation - sýtosť, Value - jas), preto ak máme obrázok v inom kódovaní, treba ho prekonvertovať do HSV.



Obrázok 2.8: Ukážka výsledku prahovania na jednom stereo obrázku z nášho reálneho datasetu. Existuje viacero druhov prahovania. My sme pre účely tejto ukážky použili orezávacie prahovanie - pri spracovávaní datasetu sme používali binárne prahovanie. Hore pôvodný obrázok, dole obrázok po prahovaní.

2. Transformujeme oba obrázky získané v bode jedna tak, aby boli v tej istej rovine a aby riadky pixlov týchto obrázkou korešpondovali. To znamená, že pre každý pixel v prvom obrázku musí byť jeho korešpondujúci pixel v druhom obrázku v tom istom riadku. Tento proces sa volá rektifikácia.

3. Nájdeme korešpondujúce pixle v oboch obrázkoch, čím dostaneme takzvanú rozdielovú mapu (z anglického disparity map). Je to matica rozdielov x-ových[3] súradníc pre korešpondujúce pixle v dvoch obrázkoch. Tento proces sa volá korešpondencia.

[3]Vďaka rektifikácii majú, v ideálnom prípade, všetky korešpondujúce pixle na ľavom a na pravom obrázku rovnaké y-ové súradnice.

4. Pomocou znalosti vzájomnej polohy kamier vieme triangulációu prepočítať rozdielovú mapu na vzdialenosti od kamery. Tento krok voláme reprojekcia.

Kapitola 3

Hardvér

3.1 Stereovizuálny systém

Na získavanie vizuálnej informácie zo scény sme použili stereo vizuálny systém (SVS) Surveyor s dvomi SRV-1 Blackfin kamerami. SVS sa hneď po nabootovaní skúsi pripojiť na Ad-Hoc WiFi sieť s menom SRV-1. Ak takáto sieť existuje, tak sa nadviaže spojenie a ku obrázkom SRV sa dá dostať cez vstavaný http server. IP adresa SVS je 198.254.0.10. Ľavá kamera je dostupná na porte 10002 a pravá na porte 10001. URL na ktorom nájdeme aktuálnu fotku z ľavej kamery je http://198.254.0.10:10002/robot.jpg. Pre fotku z pravej kamery treba zmeniť port na 10001.

Obrázok 3.1: Fotka stereo vizuálneho systému Surveyor.

Surveyor vizuálny systém je osadený na pan-tilt servo head, pomocou ktorej sa s kamerami dá točiť doľava-doprava a hore-dole.

3.2 Robotické rameno

Ako efektor, pomocou ktorého sme ukotvili priestorovú informáciu zo scény, sme použili robotické rameno od firmy Lynxmotion s kódovým označením AL5D ovládané pomocou SSC 32 servo kontrolera.

Rameno je napodobeninou ľudskej ruky a má šesť stupňov voľnosti. Pomocou servo motorov vieme ovládať natočenie celého ramena a jeho sklon, v lakti sklon a v zápästí vieme ovládať natočenie a sklon. Posledný stupeň voľnosti je v koncovom efektore. Je ním uchopovač (gripper), ktorý je jediným posuvným stupňom voľnosti. Ostatné stupne voľnosti sú rotačné.

Servo motorom sa dá pomocou ssc32 kontrolera nastaviť poloha v rozmedzí od 500 do 2500. Toto sú však teoretické hodnoty pre servo motor a nezodpovedajú reálnym limitáciám nášho použitého ramena, preto v tabuľke 3.1 uvádzame experimentálne zistené limitácie pohybu nášho konkrétneho ramena aj s korešpondujúcimi uhlami.

Servo motor	**Min - poloha**	**Max - poloha**	**Min - uhol**	**Max - uhol**
Natočenie ramena	500	2450	$-101°$	$98°$
Sklon ramena	760	2250	$-71°$	$90°$
Sklon lakťa	780	2250	$-19°$	$135°$
Sklon zápästia	650	2500	$97°$	$100°$
Natočenie zápästia	650	2350	$-70°$	$80°$
Roztvorenie uchopovača	1200	2150	none	none

Tabuľka 3.1: Tabuľka nameraných maximálnych a minimálnych hodnôt polôh servo motorov.

Vysvetlenie uhlov z tabuľky 3.1: vzhľadom na konštrukčné riešenie ramena je zrejmé, že všetky kĺby sa budú vždy nachádzať v tej istej rovine.

- uhol natočenia ramena je uhlom roviny ramena a roviny, ktorá delí servo kontroler napoly a prechádza stredom otočného mechanizmu ramena
- uhol sklonu ramena je uhol medzi podstavou a spojom, ktorý ide z ramena do lakťa

- uhol sklonu lakťa je uhol medzi spojmi od ramena k lakťu a od lakťa k zápästiu
- uhol sklonu zápästia je uhol medzi spojmi od lakťa k zápästiu a od zápästia k uchopovaču
- uhol natočenia zápästia je uhlom medzi ekvidištančnou priamkou od dvoch "prstov" uchopovača a medzi rovinou ramena
- roztvorenie uchopovača je vzdialenosť medzi dvoma "prstami" uchopovača

Obrázok 3.2: Ukážka robotického ramena AL5D spolu s SSC32 servo kontrolerom, ktoré sme použili v tejto práci. Naše rameno je ešte umiestnemé na drevenom podstavci.

3.3 Servo kontroler

Servo motory umiestnené na robotickom ramene sú ovládané pomocou ssc32 servo kontrolera, ktorý dokáže naraz spravovať až 32 servomotorov. Pre potreby nášho ramena stačí servomotorov šesť. Servo kontroler dokáže prijímať inštrukcie cez nainštalovaný serial port. My sme pri našej práci s ramenom použili redukciu serial portu na USB port.

3.3.1 Protokol na ovládanie pripojených servo motorov

Servo kontroler sprostredkuje ovládanie na neho napojených servo motorov pomocou svojho vlastného protokolu. Príkazy prijíma cez serial port ako reťazce ascii znakov ukončených znakom carriage return (<cr>, '\r'). Ovládanie pozície servomotorov sa uskutočňuje pomocou kanálov. Každý hardvérový spoj pre servomotor má svoj vlastný kanál[1] a teda keď chceme zmeniť polohu nejakého serva musíme najprv vedieť na ktorý kanál je toto servo napojené. Pre zmenu pozície serva napojeného na 1. kanál na 1000 musíme poslať servo kontroleru nasledovnú správu:

```
#1 1000<cr>
```

Ak chceme zmeniť polohu viacerých servomotorov naraz, môžeme poslať servo kontroleru správu obsahujúcu viac ako jeden pár "číslo kanálu - nová pozícia". Tieto páry môžu alebo nemusia byť oddelené medzerou:

```
#1 1000 #5 2500#2 2000<cr>
```

Na kontrolu rýchlosti pohybu môžeme použiť dva nepovinné parametre, ktoré uvedieme navyše k obsahu predchádzajúcich správ. Sú nimi rýchlosť pohybu a čas pohybu. Čas pohybu sa uvádza len raz na konci správy a uvádza sa v milisekundách (od predchádzajúcej časti správy môže, ale nemusí byť oddelený medzerou):

```
#1 1100 #3 2500 T3000<cr>
```

Ak uvedieme čas, tak všetky pohyby budú vykonané za tento uvedený čas a teda skončia presne v rovnaký moment. Rýchlosť môžeme uviesť pre každý kanál zvlášť. Uvádza sa kľúčovým písmenom "S". Každý servo motor potom bude meniť svoju rýchlosť podľa svojho vlastného tempa:

```
#2 1800S300 #3 2000#4 2100 S500<cr>
```

Ak uvedieme aj čas aj rýchlosť naraz, tak pohyb bude trvať minimálne požadovaný čas, ale je možné, že bude trvať aj dlhšie, ak si to vyžaduje zadaná rýchlosť.

Ak chceme vedieť v akej pozícii sa práve niektoré servo motory nachádzajú, treba poslať servo kontroleru správu:

[1] Servo kontroler má teda tridsaťdva kanálov.

`QP <arg> <cr>`

Servo kontroler vráti jeden byte pre každý vyžiadaný servo motor. Vrátený byte je potom momentálna pozícia servo motora zaokrúhlená na najbližšiu desiatku a s odstránenou poslednou nulou. Teda napríklad reálna pozícia 1984 by bola servo kontrolerom ohlásená ako 198.

3.4 Robotický stolček

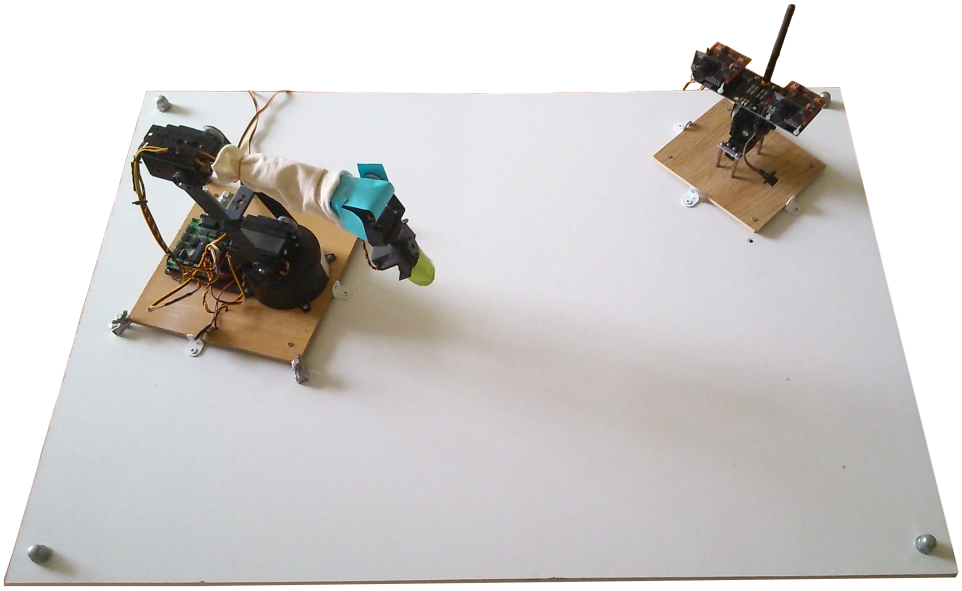
Robotické rameno je pri svojom pohybe veľmi nestabilné. Existuje veľa kombinácii stavov jeho servomotorov, pri ktorých sa rameno prevráži a padne do strany alebo dopredu. Taktiež sa pri prudkom pohybe, pohybe servomotorov, ktorý nie je servo kontrolerom vhodne obmedzený ani rýchlosťou ani časom vykonania pohybu, často stane, že sa rameno posunie zo svojej pôvodnej polohy zotrvačnou silou.

Problém posunutia zo zotrvačnosti pri prudkom pohybe platí aj pre surveyor stereovizuálny systém, ktorý je osadený na pan-tilt head.

Za účelom odstránenia týchto problémov bol navrhnutý a zostrojený robotický stolček, na ktorom sa nachádzajú pevné úchyty aj pre robotické rameno aj pre surveyor stereovizuálny systém. Úchyty boli umiestnené tak, aby sa rameno nemohlo dotknúť stereo kamier a aby tak nedošlo nehodou k poškodeniu hardvéru.

Pevné úchyty plnia aj inú úlohu ako zamedzenie nechcených pádov a pohybu alebo ochrana hardvéru. Slúžia aj na vyznačenie presného miesta a natočenia pre umiestnenie robotického ramena a surveyor stereovizuálneho systému, aby bolo zaručené, že pri budúcich použitiach tejto robotickej platformy budú mať rameno a stereo kamera, resp. ich podstavy, rovnaké relatívne natočenie a posunutie.

Robotický stolček a aj väčšia časť jeho príslušenstva sú kontrastnej bielej farby, aby boli zjednodušené úlohy detekcie objektov na scéne podľa farby.



Obrázok 3.3: Ukážka zostrojeného robotického stolčeka s osadeným robotickým ramenom a stereo vizuálnym systémom, ktoré sme použili v tejto práci na zberanie dát. Uchopovač a predposledný kĺb ramena sú obalené vo farebnom papieri pre ich jednoduchú detekciu na obrázkoch.

Kapitola 4

Zber dát

Pri trénovaní učiacich modelov sme používali dva druhy dát: syntetické, ktoré sme vygenerovali pomocou matematických transformácii a preto neobsahujú žiadny šum a reálne dáta, ktoré sme zberali pomocou robotického ramena a stereo vizuálneho systému a tieto v sebe obsahujú aj šum.

4.1 Syntetické dáta

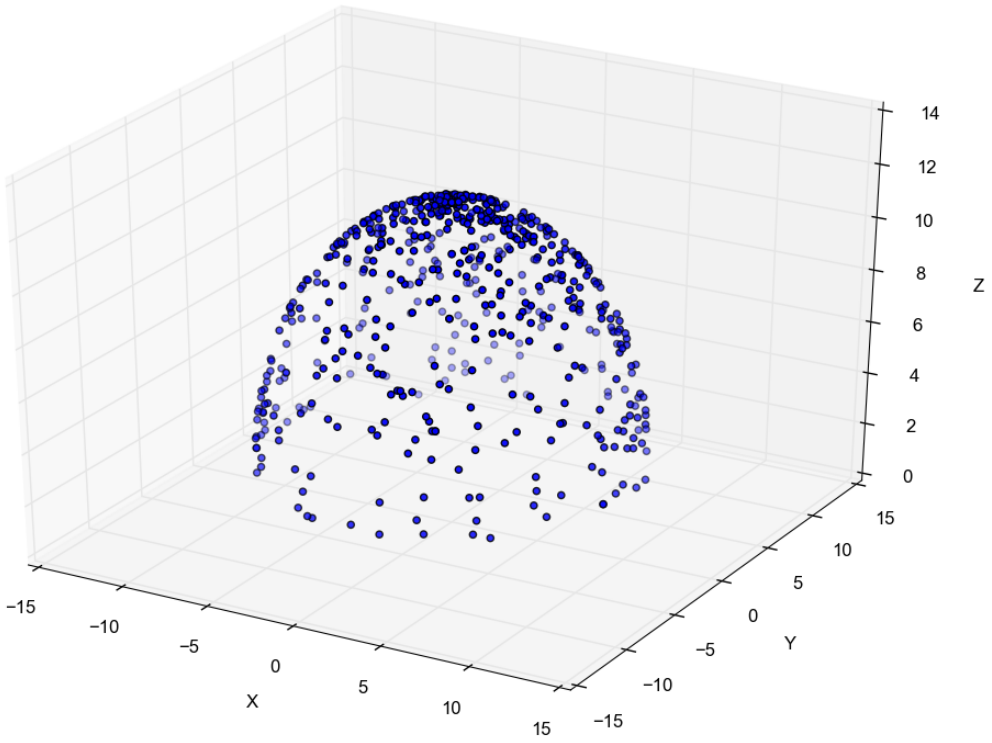
Pri generovaní syntetických dát sme si predstavili, ako vyzerá robotické rameno, pre ktoré sme chceli vygenerovať robotické dáta. Pre servomotory tohto vymysleného ramena sme generovali náhodné pozície a následne sme výsledný stav ramena opísali pomocou Denavit-Hartenberg transformačných matíc. Pomocou týchto transformačných matíc sme počítali potrebné 3D súradnice. Takýmto spôsobom sme generovali dva druhy dát. Podľa toho ako zložitého robota sme si predstavovali sme generovali `jednoduché` a `zložité` syntetické dáta.

Za účelom generovania náhodných syntetických dát sme naprogramovali skript v programovacom jazyku python `robot.py`, ktorý sa nachádza na priloženom CD a je schopný generovať arbitrárny počet vzoriek všetkých druhov syntetických dát, ktoré spomíname v tejto kapitole.

4.1.1 Jednoduché syntetické dáta

Jednoduché syntetické dáta opisujú veľmi jednoduché robotické rameno, ktoré sa skladá z dvoch spojov a má len dva stupne voľnosti. Prvý spoj je dlhý dve jednotky dĺžky a vychádza z centra súradnicovej sústavy smerom od podstavy hore. Druhý spoj je dlhý desať jednotiek dĺžky a vieme ho otáčať okolo osi Z pomocou jedného servo motora a meniť jeho uhol voči pod-

stave (rovina definovaná osami X a Y) pomocou druhého servo motora. Pozícia generovaná pre prvý servo motor teda udáva `zdvih` ramena a pozícia generovaná pre druhý servo motor udáva `natočenie` ramena.



Obrázok 4.1: Jednoduché vygenerované dáta sú umiestnené na povrchu polgule, ktorej stred sa nachádza v bode [0, 0, 2] a jej polomer je 10. Na obrázku vidíme znázornenie šesťsto polôh koncového efektora uvažovaného jednoduchého ramena.

Vzorky vygenerovaného datasetu pre jednoduché robotické rameno teda majú päť prvkov: pozíciu prvého a druhého servo motora a tri súradnice koncového efektora.

Na grafe 4.1 si môžeme všimnúť, že hustota bodov sa zväčšuje so zvyšujúcou sa Z súradnicou a teda niektoré časti povrchu polgule sú ovzorkované viac ako iné. Tento jav je spôsobený procesom generovania vzoriek dát a na výsledný efekt učenia učiaceho algoritmu nemá veľký vplyv.

Časť povrchu polgule, pre ktorú platí, že všetky jej body majú rovnakú Z súradnicu je vlastne kružnica. Pre takéto kružnice na povrchu našej polgule platí aj to, že všetky vzorky, ktoré sa nachádzajú na rovnakej kružnici museli byť vygenerované z konfigurácie ramena s rovnakou pozíciou druhého servo motora. Keďže obe pozície servo motorov sú náhodne vyberané z uniformnej distribúcie tak platí, že na každej takejto kružnici sa nachádza približne rovnaký počet

vzorieka. Tým pádom sú kružnice s menším obvodom ovzorkované hustejšie ako kružnice s väčším obvodom.

4.1.2 Zložité syntetické dáta

Zložité syntetické dáta opisujú o niečo zložitejšie robotické rameno akým je to jednoduché. Zložité rameno sa skladá zo štyroch spojov a má šesť stupňov voľnosti. Dĺžky spojov v poradí v akom sa na seba napájajú sú dve, desať, sedem a tri jednotky dĺžky. Medzi každou susediacou dvojicou spojov sa nachádza kĺb, ktorý poskytuje ramenu dva rotačné stupne voľnosti.

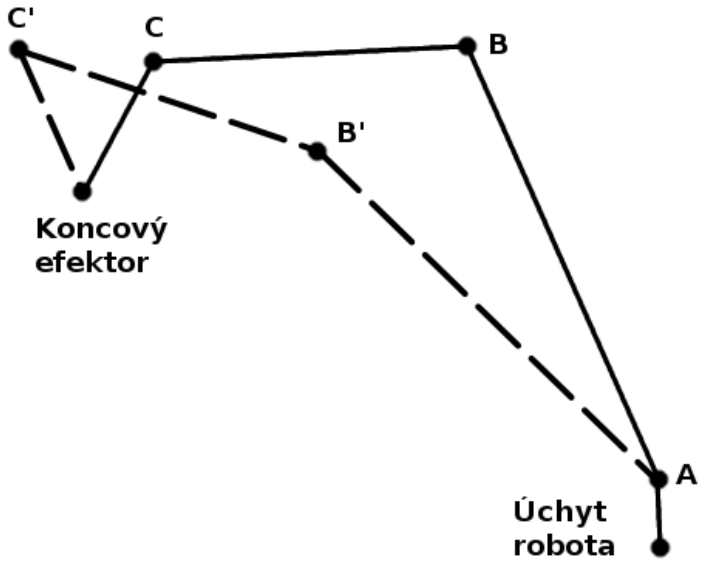
Vzorky vygenerovanej dátovej množiny pre zložité robotické rameno majú desať prvkov, ktorými sú pozície šiestich servo motorov, tri súradnice koncového efektora, tri súradnice pred-posledného kĺbu a tri súradnice pred-predposledného kĺbu.

Na rozdiel od jednoduchej množiny dát majú vzorky v zložitej množine okrem pozícii servo motorov a karteziánskych súradníc koncového efektora aj karteziánske súradnice dvoch kĺbov ramena. Pri jednoduchom ramene iné súradnice, ako súradnice koncového efektora nie je treba, lebo každému bodu na povrchu polgule prislúcha práve jedna kombinácia pozícii servo motorov. Pri zložitom ramene môže jedna pozícia koncového efektora prislúchať viacerým rôznym pozíciám šiestich servo motorov. Preto sme sa rozhodli pridať do tejto množiny dát aj súradnice ďalších dvoch kĺbov a tým vytvoriť medzi konfiguráciou ramena a opisom stavu koncového efektora jedno-jednoznačné zobrazenie.

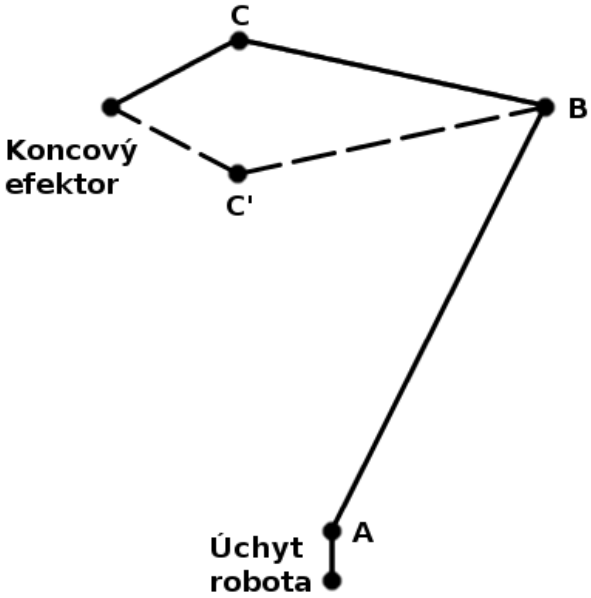
Označme si polohy kĺbov v poradí v akom je robot postavený ako kĺby A, B a C a koncový efektor KF. Najprv si ukážme, prečo treba na jedno-jednoznačné zobrazenie medzi polohami servo motorov a konfiguráciou ramena až tri trojice súradníc.

Predpokladajme, že všetky tri kĺby a aj koncový efektor sa nachádzajú v jednej rovine. Je jednoduché predstaviť si, že body A, B, C a KF sú v rovine umiestnené tak, že vytvárajú všeobecný štvoruholník, ktorého dĺžky hrán a polohu bodov A[1] a KF poznáme. Možných riešení polôh bodov B a C je nekonečne veľa (ilustrované na obrázku 4.2 a), tak sa rozhodneme zafixovať ešte jeden bod. Bez ujmy na všeobecnosti, nech je týmto bodom bod B. Teraz poznáme body A, B a KF a vzdialenosti medzi každou dvojicou susediacich bodov. Body B, C a KF sú teraz vrcholy trojuholníka, ktorého dĺžky strán a body B a KF poznáme. Existujú až dve

[1] Úchyt ramena je v centre súradnicovej sústavy a transformácia definovaná prvým spojom sa vzhľadom na žiadny servo motor nemení a teda nech je konfigurácia servo motorov akákoľvek, poloha bodu A je konštantná a vieme ju vypočítať dopredu.



(a) Príklad takej zmeny zdvihu v kĺboch A, B a C, že poloha koncového efektora ostane nezmenená. V spojitom priestore existuje takýchto zmien zdvihov nekonečne veľa.



(b) Príklad takej zmeny zdvihu v kĺboch B a C, že poloha koncového efektora ostane nezmenená.

Obrázok 4.2: Pri ramene s viacerými stupňami voľnosti nie je poloha koncového efektora postačujúcou informáciou na zistenie konfigurácie ramena. Na obrázkoch vidíme ako zmena zdvihu vo viacerých kĺboch nemusí túto polohu zmeniť.

riešenia súradníc bodu C (obrázok 4.2 b) a teda musíme vybrať aspoň dva body, okrem koncového bodu, aby sme dostali jedno-jednoznačné zobrazenie medzi konfiguráciou servo motorou a

konfiguráciou ramena v priestore. Takže nato, aby sme vedeli zistit presnú konfiguráciu ramena v priestore potrebujeme vedieť okrem súradníc KF aj súradnice bodov C a B.

Keďže úchyt ramena je vždy v počiatku súradnicovej sústavy, súradnicu bodu A si vieme vypočítať a súradnice bodov B, C a KF máme dané, vieme z každej vzorky zložitého datasetu zistiť presnú priestorovú konfiguráciu ramena.

4.2 Reálne dáta

Reálne dáta narozdiel od syntetických neopisujú vymyslené robotické rameno s ideálnymi vlastnosťami, ale konkrétny kus hardvéru, ktorý sme mali k dispozícii.

Zberanie robotických dát sme implementovali tak, aby prebehlo automaticky, bez nutnosti zásahu človeka, okrem samotného spustenia procesu. Od človeka sa požaduje zadať požadovaný počet vzoriek a script bude iteratívne generovať náhodné pozície pre robotické rameno, fotiť scénu a ukladať tieto dáta na disk do súboru formátu csv (Comma-Separated Values).

Predtým ako môžeme začať zberať a vyhodnocovať dáta potrebujeme nakalibrovať našu stereo kameru. Na tento účel sme naprogramovali skript `camera.py` v jazyku python, ktorý v hlavnom cykle fotí a následne ukladá obrázky zo stereo kamery. Týmto skriptom sme urobili niekoľko obrázkov šachovnice, pomocou ktorých sme potom vypočítali potrebné parametre a uložili sme ich pre neskoršie použitie pri spracovaní obrázkov scény[2].

Pre potreby samotného zberania dát nie je potrebné počítať 3D súradnice lakťa, zápästia a koncového efektora [3], na počítanie súradníc koncového efektora a zápästia do datasetu budeme používať obrázky scény, ale tieto súradnice budeme počítať aj tak, aby sme mohli hardvér nášho robota ochrániť pred možným poškodením. Zdrojom možného poškodenia ramena a jeho častí je napríklad taká konfigurácia servo motorov ramena, ktorá vedie k polohe koncového efektora pod úrovňou podstavy. Takéto konfigurácie môžu byť odfiltrované ešte predtým ako ich dáme ramenu vykonať.

Pseudokód 4.1 Idea hlavného cyklu zberu robotických dát

---

[2]Pri rôznych rozlíšeniach fotiek fotených cez tú istú šošovku, sú kalibráciou vypočítané parametre rozdielne. My sme vypočítali parametre pre rozlíšenia 320x240 a 640x480.

[3]Súradnice ramena, meno kĺbu robotického ramena inšpirované ľudskou hornou končatinou, sa pohybom servo motorov nemenia, podobne ako to bolo pri bode A zložitého vymysleného ramena a teda vieme si ich vypočítať na začiatku programu a používať výsledok ako konštantu počas celého behu programu.

```
curent_count = 0
while curent_count < target_count:
    positions = generate_random_position()
    servos.set_positions(positions)
    left_img, right_img = svs.get_images()
    img = join_images(left_img, right_img)
    store_image(img, 'svs%d.jpg' % curent_count)
    update_dataset(positions, 'svs%d.jpg' % curent_count)
    curent_count += 1
```

Po dokončení behu skriptu sa vzorky skladajú zo štyroch pozícii servo motorov a cesty k obrázku scény s ramenom s touto konfiguráciou servo motorov. Do datasetu ukladáme pozície servo motorov, ktoré udávajú natočenie ramena, sklon ramena, sklon lakťa a sklon zápästia. Pozície servo motorov, ktoré udávajú natočenie zápästia a roztvorenie uchopovača zámerne vynechávame, lebo tieto nemajú žiadny vplyv na polohu koncového efektora. Z uloženého obrázka sme zámerne ešte neodstránili šošovkové skreslenie, ani sme ho nijak inak neupravovali. Tento nespracovaný dataset obsahuje aj také vzorky, pre ktoré sa buď na jednom alebo ani na obidvoch obrázkoch nenachádzajú zelený a modrý marker, preto treba tento prvotný dataset odfiltrovať od takých vzoriek, ktoré sú z dôvodu absencie jedného alebo oboch markerov na obrázkoch nepoužiteľné.

Obrázok 4.3: Obrázok robotického ramena, tak ako bol zosnímaný pri zbere reálnych dát - ešte z neho nie je odstránené skreslenie šošoviek. Pôvodný obrázok má rozlíšenie 640x480 pixlov.

Na odstránenie stereo obrázkov, ktoré neobsahujú potrebnú vizuálnu informáciu sme použili prahovanie a hľadanie kontúr. Na oba procesy sme využili knižnicu OpenCV.

Prahovanie sme implementovali aplikovaním funkcie `inRange` na obrázky s reprezentáciou farby pixlov pomocou HSV. Pomocou prahovania získame dve binárne masky stereo obrázka. Jednu pre zelené objekty a jednu pre modré. Konštanty použité pre hľadanie modrej a zelenej farby na obrázku sú uvedené v tabuľke 4.1. Hlavne pri prahovaní zelenej farby sa nám do binárnej masky dostával nechcený šum. Napríklad časti niektorých žltých káblikov alebo časť podstavy robotického ramena (pozri obrázok 2.8).

Farba	**Vlastnosť**	**Od**	**Do**
modrá	Hue	90	110
	Saturation	130	255
	Value	150	255
zelená	Hue	40	70
	Saturation	90	170
	Value	150	200

Tabuľka 4.1: Konštanty použité pri prahovaní.

Na odstránenie tohto šumu sme sa snažili odstrániť čo najviac nepotrebných predmetov, ktoré by mohlo prahovanie vyhlásiť za zelené, zo zorného uhla stereo kamery. Hlavne sme však výsledné binárne masky pre výskyty zelenej a modrej farby, za účelom odstránenia nežiaducich výskytov, ďalej spracovávali hľadaním kontúr.

Výstupom fázy prahovania sú dve binárne masky veľkosti pôvodného obrázku. Pre tieto masky sme našli kontúry pomocou funkcie z knižnice OpenCV `findContours`. Pre každú kontúru sme vypočítali obsah pomocou funkcie `contourArea`. Experimentálne sme si stanovili konštantu - minimálny obsah zelenej alebo modrej farby na plôške - pomocou ktorej sme odfiltrovali nežiaduce farebné plôšky (plochy s veľmi nízkym obsahom pixlov). Zo zvyšných plôch sme vyhlásili za marker plochu s najväčším obsahom pixlov.

Po odfiltrovaní stereo obrázkov, ktoré neobsahujú oba markery sme pristúpili k extrakcii 3D pozícii oboch markerov. Prvým krokom pri extrakcii 3D súradníc pomocou knižnice OpenCV je rektifikácia obrázkov, to znamená zarovnanie aj ľavého a aj pravého obrázku na riadky pre jednoduché párovanie príznakov. Existujú dva prístupy k rektifikácii: prvý používa OpenCV

funkciu `stereoRectifyUncalibrated` a nepotrebuje poznať parametre kamier a druhý používa funkciu `stereoRectify` a potrebuje poznať parametre kamery. Keďže my parametre kamier vďaka stereo kalibrácii poznáme, použili sme druhý spôsob. Jeho výhodou je, že dáva lepšie výsledky pre obrázky s výrazným skreslením[4].

Výstupom stereo rektifikácie sú dva páry rotačných a projekčných matíc. Jedna rotačná a jedna projekčná pre každý z dvoch obrázkov. Pomocou nich si pre každý obrázok zvlášť vypočítame použitím funkcie `initUndistortRectifyMap` dve transformačné matice, ktoré budeme používať ako parametre funkcie `remap`. Táto funkcia nám odstráni skreslenie a zarovná obrázky. Transformačné matice vypočítané funkciou `initUndistortRectifyMap` sa vzťahujú na vlastnosti kamier, nie na vlastnosti konkrétneho páru obrázkov a teda netreba ich počítať pre každý obrázok zvlášť. Výstupom rektifikácie sú dva na riadky zarovnané obrázky. Pre takýto pár obrázkov vieme pomocou `StereoBM` operátora vypočítať takzvanú disparity mapu. Je to jedno kanálový obrázok, ktorého hodnota pixlu hovorí o vzdialenosti tohto pixlu od pixlu reprezentujúceho tú istú črtu na druhom obrázku z toho istého stereo páru.

Z disparity máp sme následne vypočítali pomocou funkcie `reprojectImageTo3D` 3D súradnice pre všetky pixle obrázka. Aj disparity mapa aj 3D mapa zachovávajú súradnice čŕt z prvého (v našom prípade ľavého) obrázku a teda na zistenie 3D súradníc markerov sme použili binárne masky ľavého obrázka. Pre výsledné 3D súradnice sme vypočítali ich priemer a zapísali sme ho do množiny dát.

[4]Naše obrázky výrazné skreslenie majú.

Kapitola 5

Návrh a implementácia

Naším cieľom bolo vytvoriť prepojenie medzi vizuálnou informáciou o polohe koncovej časti robotického ramena a proprioceptívnou informáciou, to je konfigurácia servo motorov, využívajúc biologicky inšpirované algoritmy na učenie obojsmernej neurónovej siete.

Pod prepojením vizuálnej a proprioceptívnej informácie sme chápali dve zobrazenia a to zobrazenie 3D súradníc polohy robotického ramena na konfiguráciu servo motorov a naopak, zobrazenie konfigurácie servo motorov na 3D súradnice polohy koncového efektora robotického ramena.

5.1 BAL

Na implementovanie oboch zobrazení sme si vybrali algoritmus BAL, ktorý vychádza z biologicky motivovaného algoritmu GeneRec a sám je tiež biologicky prijateľnejší ako algoritmus spätného šírenia chyby. Narozdiel od algoritmu spätného šírenia chyby prebieha zmena váh medzi neurónmi aj v GeneRec aj v BAL sieti len na základe lokálne prístupných hodnôt aktivácii neurónov.

Algoritmom BAL sa narozdiel od algoritmu GeneRec navyše neučí iba vstupno - výstupné mapovanie medzi dátami, ale aj spätné výstupno - vstupné. Preto pri učení týmto algoritmom nebudeme hovoriť o vstupnej a výstupnej vrstve a o vstupe a výstupe siete, ale budeme hovoriť o prvej a poslednej vrstve a o vektore X a vektore Y.

5.1.1 Nejednoznačnosť dát

Obojsmerné učenie algoritmu BAL je aj jeho nevýhodou, lebo je pravdepodobné, že práve preňho sa týmto algoritmom nedá natrénovať sieť na binárnu funkciu XOR. Tento fakt je pravdepodobne spôsobený nejednoznačnosťou spätného zobrazenia, kde sa váhy v smere od poslednej vrstvy k prvej, pre povahu problému, a tým pádom aj vzoriek na ktorých je sieť tento problém učená, nevedia ustáliť a preto chyba nikdy dostatočne neklesne.

Zobrazenie 3D súradníc koncového efektora na konfiguráciu servo motorov je tiež nejednoznačné, pozri obrázok 4.2, preto sme sa rozhodli, vzhľadom na spomenutú limitáciu BAL algoritmu, učiť sieť nie len polohu koncového efektora, ale aj polohy niektorých ďalších kĺbov ramena. Rozšírením dimenzie oboru hodnôt, to je pridaním ďalších súradníc, sme dosiahli, že cieľové zobrazenie f má oveľa menej takých priestorových konfigurácii, pre ktoré platí:

$$f(x_1) = f(x_2) \rightarrow x_1 = x_2$$

, kde x_1 a x_2 sú konfigurácie servo motorov. Vďaka tejto úprave dátovej množiny by mal mať BAL lepšie výsledky pri učení tohto problému.

Zobrazenie z konfigurácii servo motorov do priestorovej konfigurácie ramena je jednoznačné, surjektívne, aj bez pridania súradníc niektorých kĺbov a pridanie týchto súradníc to nezmení. Zvýši to však náročnosť úlohy, keďže sieť bude musieť vedieť odhadnúť viac parametrov priestorovej konfigurácie ramena.

5.2 Implementácia učiaceho algoritmu BAL

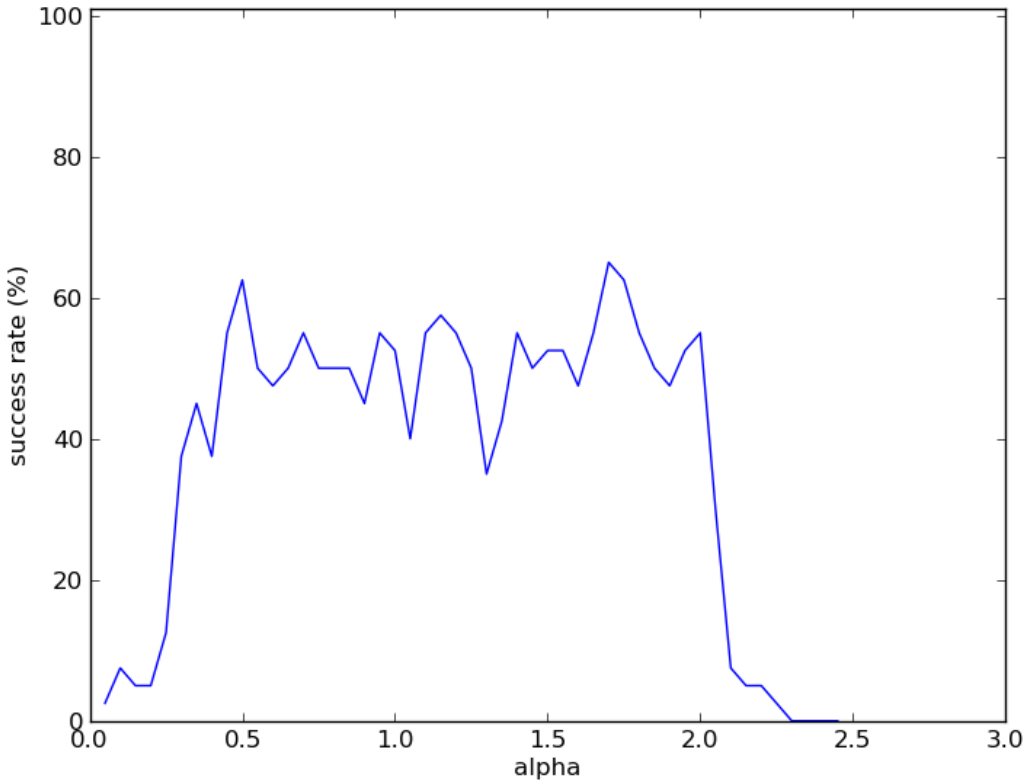
Učiaci algoritmus BAL a aj všetky ostatné programy na zberanie alebo generovanie dát sme implementovali v programovacom jazyku python. Tento sme vybrali pre jeho bohatú ponuku knižníc. Na maticové operácie sme použili knižnicu `numpy`, na kreslenie grafov sme použili knižnicu `matplotlib` [5], na komunikáciu s robotickým ramenom sme použili knižnice `pyserial` a `pyssc32`, na HTTP komunikáciu so stereo vizuálnym systémom sme použili knižnice `requests` a `grequests` (paralelná verzia knižnice `requests`) a na spracovanie stereo obrazu sme použili knižnicu `opencv` s jej python wrapperom.

Implementácia učiaceho algoritmu BAL sa nachádza v súbore `ann.py` spolu s implementáciou neurónovej siete, ktorá má váhy aj s prvej vrstvy k poslednej aj z poslednej vrstvy k prvej.

5.2.1 Testovanie implementácie BALu

Klasifikačná úloha

Našu implementáciu BAL algoritmu sme porovnali s implementáciou pôvodných autorov tohto učiaceho algoritmu na 4-2-4 enkóder úlohe. Je to binárna - klasifikačná úloha v ktorej ukazujeme sieti vzorky o dĺžke štyroch bitov, kde je práve jeden bit nastavený na hodnotu jedna, ostatné sú nastavené na hodnotu nula a od siete požadujeme, aby zobrazila vstupné hodnoty na také isté výstupné hodnoty, a teda aby $\forall i : x_i = y_i$, kde x_i je i-ty vstup a y_i je i-ty výstup siete, pomocou dvoch skrytých neurónov[1]. Pri použití učiaceho algoritmu BAL sme od siete nepožadovali len správne výsledky v smere $x \rightarrow y$, tak ako je zadefinovaný pôvodný problém, ale aj v smere $y \rightarrow x$.



Obrázok 5.1: Graf úspešnosti učiaceho algoritmu BAL na 4-2-4 enkóder úlohe. x-ová os znázorňuje použitú rýchlosť učenia, y-ová os znázorňuje úspešnosť algoritmu v percentách. To je, koľko percent sietí z tridsiatich sa úspešne naučilo klasifikovať x na samé seba a takisto y.

[1] Existujú aj modifikácie tejto úlohy, za všetky uvedieme 8-3-8 a 12-4-12.

Autori pôvodnej BAL implementácia uvádzajú úspešnosť siete na 4-2-4 enkóder úlohe približne 50 až 60% pri učiacej rýchlosti α v rozmedzí od 0.5 do 2.0[3]. Pri použití α mimo tohto intervalu je úspešnosť siete malá až nulová. Ako je vidieť z grafu 5.1 naša implementácia učiaceho algoritmu BAL má pri tejto úlohe identické správanie.

Regresná úloha

Keďže naša cieľová úloha nie je klasifikačná, ale regresná, rozhodli sme sa otestovať, či je algoritmus BAL schopný naučiť sa aj takýto typ úlohy. Za našu testovaciu regresnú úlohu sme vybrali kvôli jednoduchosti vizualizácie výsledkov jednoduchú nelineárnu funkciu $f(x) = x^2$, pre $x \in \langle 0.0, 1.0 \rangle$. V tomto intervale sme zobrali jedenásť rovnomerne rozložených vzoriek a tie sme použili ako trénovaciu množinu. Ako validačnú množinu sme použili päťdesiat jedna rovnomerne rozložených vzoriek z uzavreného intervalu $\langle 0.0, 1.0 \rangle$.

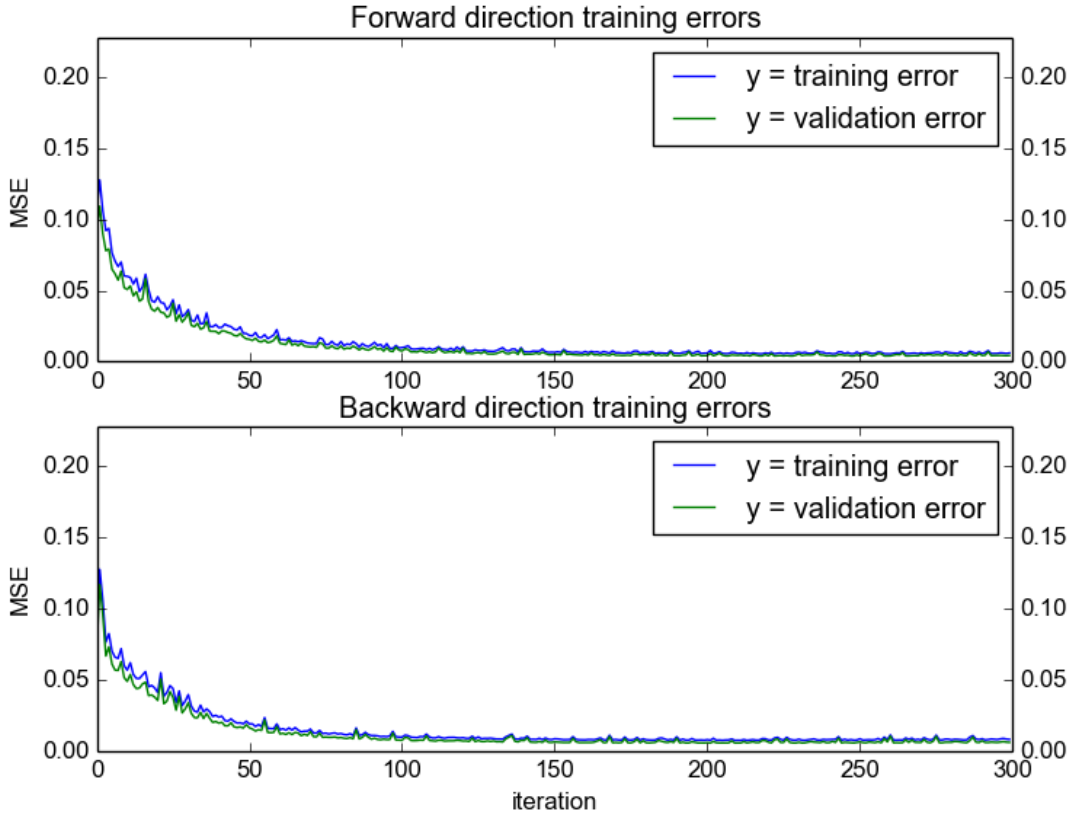
Z povahy algoritmu BAL vyplýva, že kým v doprednom smere sa sieť naučí zadané zobrazenie, v spätnom smere sa naučí inverzné zobrazenie k zadanému. V našom prípade sa spätnom smere sieť má naučiť funkciu $bck(y) = \sqrt{y}$.

Experimentálne sme zistili, že učenie tejto úlohy pomocou BALu dosahuje najlepšie výsledky pri sieti s osemnástimi skrytými neurónmi a s rýchlosťou učenia 0.2. Na obrázkoch 5.2 a 5.3 ilustrujeme priebeh a výsledky učenia takejto siete.

Učenie siete sme zastavili po 300 epochách s konečnou chybou na trénovacej množine 0.006157 v doprednom smere a 0.008041 v spätnom smere. Chyba na validačnej množine bola 0.004399 v doprednom smere a 0.005942 v spätnom smere. Z grafu 5.2 vidíme, že od 150 iterácie sa už chyba výrazne nemenila.

Keďže výstupy aj vstupy nášho problému sa nachádzajú v 1D priestore, maximálna euklidovská vzdialenosť dvoch x a aj dvoch y v priestore je 1. Na vyhodnocovanie chyby sme používali priemernú kvadratickú chybu (z anglického mean square error - MSE).

Keďže problémy, ktoré budeme riešiť majú rozdielne dimenzie prvej a poslednej vrstvy, rozhodli sme sa priemerovať priemernú kvadratickú chybu na priemernú chybu jednej zložky príznakového vektora, aby sme sa vyhli zvyšovaniu chyby pri zvyšovaní dimenzie vstupnej alebo výstupnej vrstve. Pre zobrazenie $f(x) = x^2$ táto modifikácia priemernej kvadratickej chyby nič neovplyvní, lebo pri príznakových vektorov v 1D priestore bude chyba len vydelená jednotkou, ale napríklad pri zobrazeniach medzi konfiguráciou servo motorov a priestorovou konfiguráciou robotického ramena budeme môcť vďaka tejto úprave jednoducho porovnávať

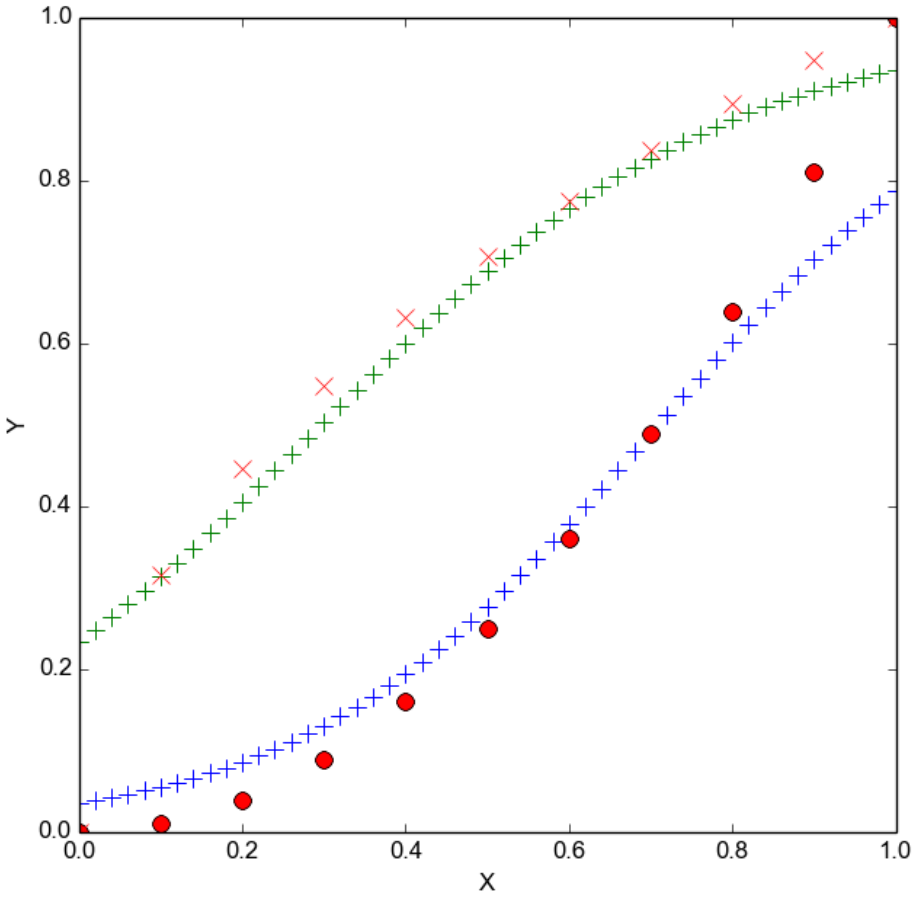


Obrázok 5.2: Dvoj-graf znázorňuje vývoj chyby (priebeh učenia) funkcie $f(x) = x^2$ pomocou učiaceho algoritmu BAL v 300 epochách. Hore vidíme vývoj chyby v doprednom smere a dole vidíme vývoj chyby v spätnom smere učenia. V doprednom smere sa chyba ustálila približne na 0.006157 a v spätnom smere sa ustálila približne na 0.008041. Na vyhodnocovanie chyby používame priemernú kvadratickú chybu.

úspešnosti siete v doprednom a spätnom smere.

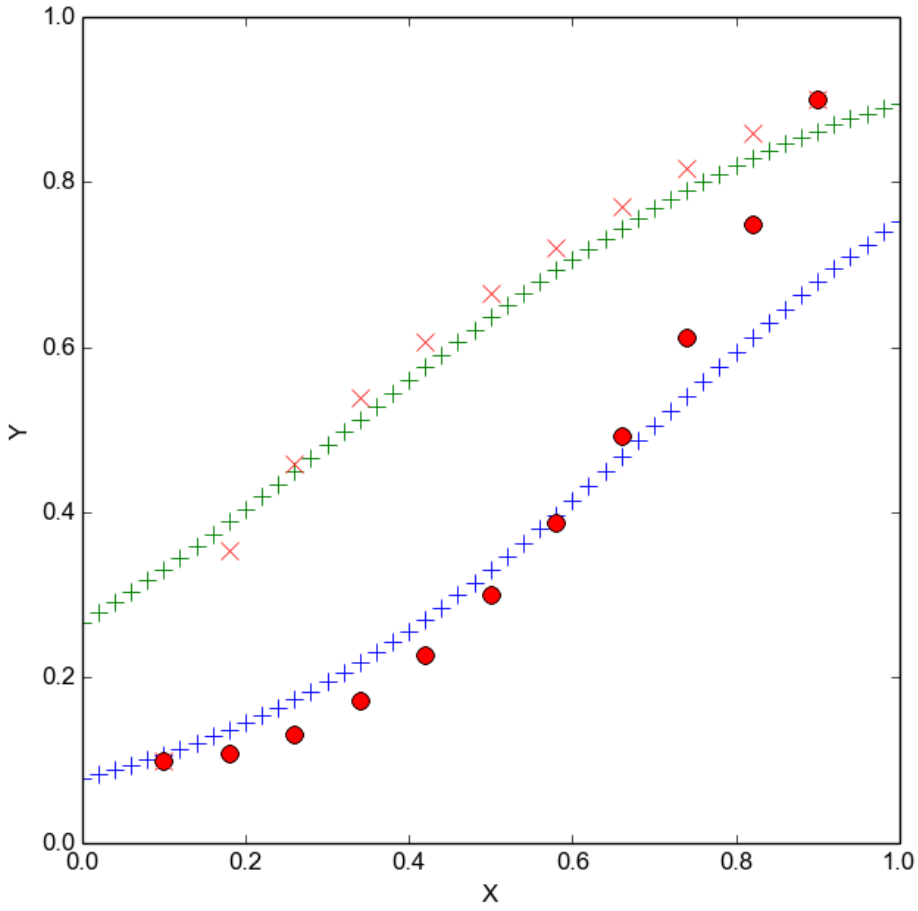
Z výsledkami učenia tejto jednoduchej nelineárnej funkcie sme, čo sa strednej kvadratickej chyby týka, boli spokojný. Vizualizácia naučených zobrazení (obrázok 5.3) odhalila problém siete naučiť sa správne zobrazenie na celom intervale. Sieť nemala problém naučiť sa hodnoty v strede intervalu, ale pri hraničných bodoch (to sú 0.0 a 1.0) sa chyba zvýšila. Skúsili sme teda normalizovať dáta do intervalu $\langle 0.1, 0.9 \rangle$, keďže 0.0 a 1.0 sú krajné hodnoty sigmoidy a pre algoritmus môže byť ťažké dosiahnuť ich.

Vizualizácia naučeného zobrazenia pre dáta normalizované do menšieho intervalu je zobrazená na grafe 5.4. Pri učení normalizovaných dát klesla úspešnosť z takmer 100% na pri-



Obrázok 5.3: Vizualizácia zobrazenia naučeného učiacim algoritmom BAL pri trénovaní funkcie $fwd(x) = x^2$. Z povahy BALu vyplýva, že v smere od poslednej vrstvy k prvej sa naučil inverzné zobrazenie k pôvodnej funkcii a teda $bck(y) = \sqrt{y}$. Červené bodky na grafe znázorňujú trénovaciu množinu a červené krížiky sú len vodiacimi značkami na grafe pre funkciu $bck(y) = \sqrt{y}$ a sieti neboli počas trénovania prístupné. Modré znaky plus znázorňujú výstupy siete v doprednom smere a zelené pluská znázorňujú výstupy siete v spätnom smere.

bližne 25%, navyše naučené zobrazenie aj tak verne nekopírovalo cieľové obrazy krajných bodov intervalu a chyba siete v doprednom smere stúpla aj mimo krajných bodov. Preto sme sa rozhodli normalizovať robotické dáta do intervalu $\langle 0.0, 1.0 \rangle$, nie do menšieho.



Obrázok 5.4: Vizualizácia zobrazenia naučeného učiacim algoritmom BAL pri trénovaní funkcie $fwd(x) = x^2$ na intervale $\langle 0.0, 1.0 \rangle$, pričom trénovacie dáta sme normalizovali do intervalu $\langle 0.1, 0.9 \rangle$. Červené bodky na grafe znázorňujú trénovaciu množinu a červené krížiky sú len vodiacimi značkami na grafe pre funkciu $bck(y) = \sqrt{y}$ a sieti neboli počas trénovania prístupné. Modré znaky plus znázorňujú výstupy siete v doprednom smere a zelené pluská znázorňujú výstupy siete v spätnom smere.

5.3 Ako interpretovať chybu

Pri pokusoch so syntetickými robotickými dátami môžeme predpokladať že použitými jednotkami dĺžky sú centimetre. Ďalej všetky hodnoty, tj. karteziánske súradnice a aj polohy servo motorov sú normované do intervalu $\langle 0.0, 1.0 \rangle$. Na prvej vrstve máme konfiguráciu servo motorov a na poslednej vrstve máme priestorovú konfiguráciu robotického ramena. Ako hod-

notenie siete uvádzame priemer priemernej kvadratickej chyby na jednu zložku príznakového vektora. Táto nám však nepovie veľa o veľkosti chyby vzhľadom na priestor a polohy koncového efektora, preto sme sa rozhodli uvádzať aj iný druh chyby, vďaka ktorému si bude aj človek vedieť predstaviť ako dobre by sieť fungovala v reálnom nasadení.

Na lepšiu predstavu o chybe naučeného zobrazenia budeme v doprednom smere, ale aj v spätnom smere, uvádzať priemernú vzdialenosť v centimetroch medzi vypočítanou a skutočnou polohou koncového efektora. Centimetrovú chybu budeme počítať na validačnej množine dát a budeme ju počítať len jeden krát a to po skončení trénovania. Túto chybu budeme zaokrúhľovať na tri desatinné miesta.

Pri doprednom smere je význam takto hlásenej chyby jasný. Budeme hovoriť o vzdialenosti skutočnej polohy koncového efektora od tej, ktorú nám hlási sieť pre danú konfiguráciu servo motorov. Pri spätnom smere je význam chyby v centimetroch menej zjavný, keďže cieľom siete je naučiť sa priestorovú konfiguráciu servo motorov. V tomto smere bude chyba v centimetroch vyjadrovať vzdialenosť pôvodnej polohy koncového efektora od vypočítanej polohy koncového efektora, kde pôvodná poloha je polohou, ktorú dostala sieť na poslednej vrstve a vypočítaná poloha je poloha, ktorú by koncový efektor mal, ak by robotické rameno malo konfiguráciu servo motorov, ktorú hlási sieť na prvej vrstve.

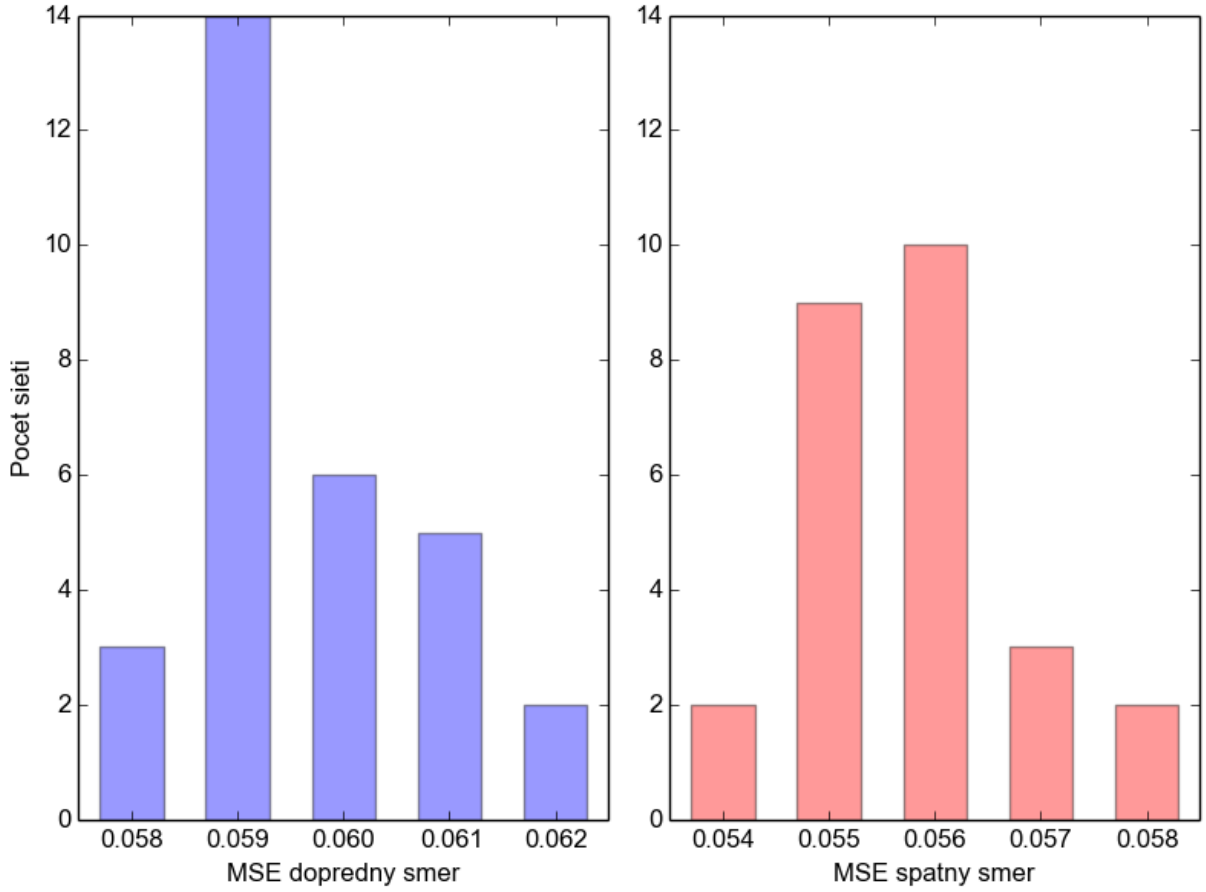
5.4 Pokusy s jednoduchými syntetickými dátami

5.4.1 BAL

Naším prvým experimentom bol pokus natrénovať niekoľko obojsmerných sietí pomocou algoritmu BAL na jednoduché syntetické dáta.

Skúsili sme natrénovať 30 sietí s rýchlosťou učenia 0.5 a 35 skrytými neurónmi. Siete sme trénovali v 1000 epochách. Pre trénované siete si pamätáme najlepšiu a najnovšiu konfiguráciu. Dosiahnuté výsledky algoritmu prezentujeme v grafe 5.5. Ako výsledky algoritmu neprezentujeme najnovšiu (poslednú) konfiguráciu v ktorej sieť ostala po skončení algoritmu, ale najlepšiu konfiguráciu, akú počas učenia dosiahla. K dispozícii sme mali 600 vzoriek jednoduchých syntetických dát, z ktorých sme 200 použili ako trénovaciu množinu a 400 ako validačnú množinu.

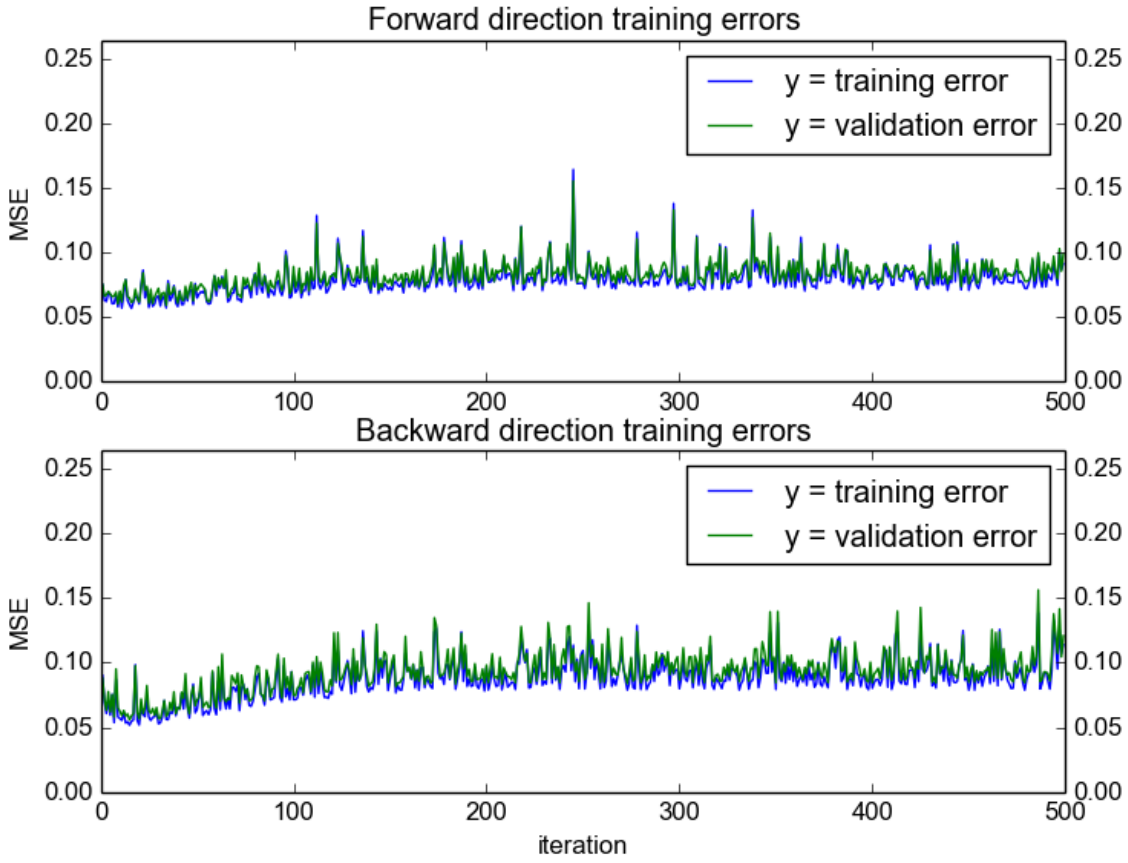
Z natrénovaných tridsiatich sietí sme ručne vybrali jednu, ktorá mala najlepšie výsledky a spočítali sme pre ňu takú chybu v centimetroch, akú sme opísali v predchádzajúcej sekcii. Priemerná chyba dopredného smeru bola 6.025 cm (minimum bolo 3.261 cm a maximum 13.511)



Obrázok 5.5: Naľavo vidíme počty sietí vzhľadom na ich doprednú priemernú kvadratickú chybu zaokrúhlenú na tri desatinné miesta a napravo vidíme počty sietí vzhľadom na ich spätnú priemernú kvadratickú chybu zaokrúhlenú na tri desatinné miesta. Oba grafy sú vygenerované pomocou MSE na validačnej množine. Na grafe spätného smeru sme nezobrazili štyri siete, z ktorých dve mali MSE 0.060, jedna 0.061 a jedna 0.062 pre lepšiu prehľadnosť grafu.

a spätného smeru bola 4.783 (minimum bolo 0.097 cm a maximum 13.983 cm). Takéto výsledky nie sú uspokojivé.

Z grafu 5.5 vidíme, že väčšina sietí dosiahla vo svojej najlepšej konfigurácii doprednú MSE 0.059 a spätnú MSE 0.056. Najlepšie konfigurácie dosahovali siete približne pri dvadsiatej až tridsiatej epoche učenia. Po dosiahnutí najlepšej konfigurácie sieť neostala stabilná a MSE sa v oboch smeroch učenia postupne zhoršovala, až sa neskôr ustálila na horších hodnotách aké mala pôvodná, ešte netrénovaná sieť. Pre ilustráciu uvádzame graf 5.6 učenia jednej z tridsiatich sietí.



Obrázok 5.6: Priebeh učenia jednej z tridsiatich sietí trénovaných pomocou BAL algoritmu s rýchlosťou učenia 0.5. Sieť dosiahla najlepšiu konfiguráciu v 22 epoche, potom sa jej chyba aj v doprednom aj v spätnom smere zhoršovala. Z obrázku je tiež jasne vidieť nestabilnosť konfigurácie siete.

Z tohto grafu je vidieť ako veľmi je sieť nestabilná a preto sme sa rozhodli náš pokus zopakovať, ale tento krát sme zmenšili rýchlosť učenia na 0.01. Vďaka tejto úprave sa sieť stala oveľa stabilnejšou, ale výsledky siete, tak ako sme ich videli na grafe 5.5 sa zhoršili. To znamená, že najnovšia konfigurácia sietí síce bola lepšia ako v pôvodnom experimente, ale najlepšia konfigurácia bola horšia v priemere o pol centimetra.

Pre zlepšenie výsledkov sme skúsili aj zvýšiť počet neurónov na skrytej vrstve na 50 a aj 60 skrytých neurónov, ale výsledky siete sa výrazne nezmenili. Preto sme dospeli k záveru, že pomocou pridávania ďalších neurónov na skrytú vrstvu k zlepšeniu výsledkov siete, ani v jednom smere, nedospejeme.

5.4.2 BAL s rôznymi rýchlosťami učenia

Keďže učiaci algoritmus BAL nedosiahol pre nás postačujúce výsledky, rozhodli sme sa pokračovať v našich experimentoch používajúc modifikovaný BAL algoritmus, v ktorom sa používajú pri učení rozdielne rýchlosti učenia pre jednotlivé vrstvy [1].

Táto modifikácia algoritmu používa presne tri vrstvy neurónov a na zmenu váh w_{IH}^F a w_{OH}^B používa rádovo menšiu rýchlosť ako na zmenu váh w_{HO}^F a w_{HI}^B.

Peter Csiba reportuje najvyššiu úspešnosť tohto učiaceho algoritmu, keď pomer učiacich rýchlostí je rádovo $1 : 10^8$, ale svoje experimenty obmedzuje iba na auto enkóder úlohy, ktoré sú navyše klasifikačné, narozdiel od našej úlohy, ktorá je regresná. Preto sme sa rozhodli, že budeme s pomerom učiacich rýchlostí experimentovať aj my a pokúsime sa nájsť pomer, ktorý bude najlepší pre našu konkrétnu úlohu.

Náš experiment zahŕňal učenie 150 sietí pri 15 rôznych pomeroch rýchlostí učenia a teda pre každý uvažovaný pomer rýchlostí učenia sme trénovali 10 rôznych sietí. Aby sme zaistili stabilitu siete, zvolili sme si za základnú rýchlosť učenia pre váhy zo skrytej vrstvy do výstupných vrstiev 0.1. Táto rýchlosť bola pre každú sieť rovnaká počas celého experimentu. Podľa pomeru sme potom vypočítali rýchlosť učenia pre váhy v smere zo vstupných vrstiev do skrytej vrstvy. Uvažované pomery, ktoré sme v tomto experimente testovali, boli $1 : 10^n$, kde $n \in \{-2, 3, \ldots, 12\}$.

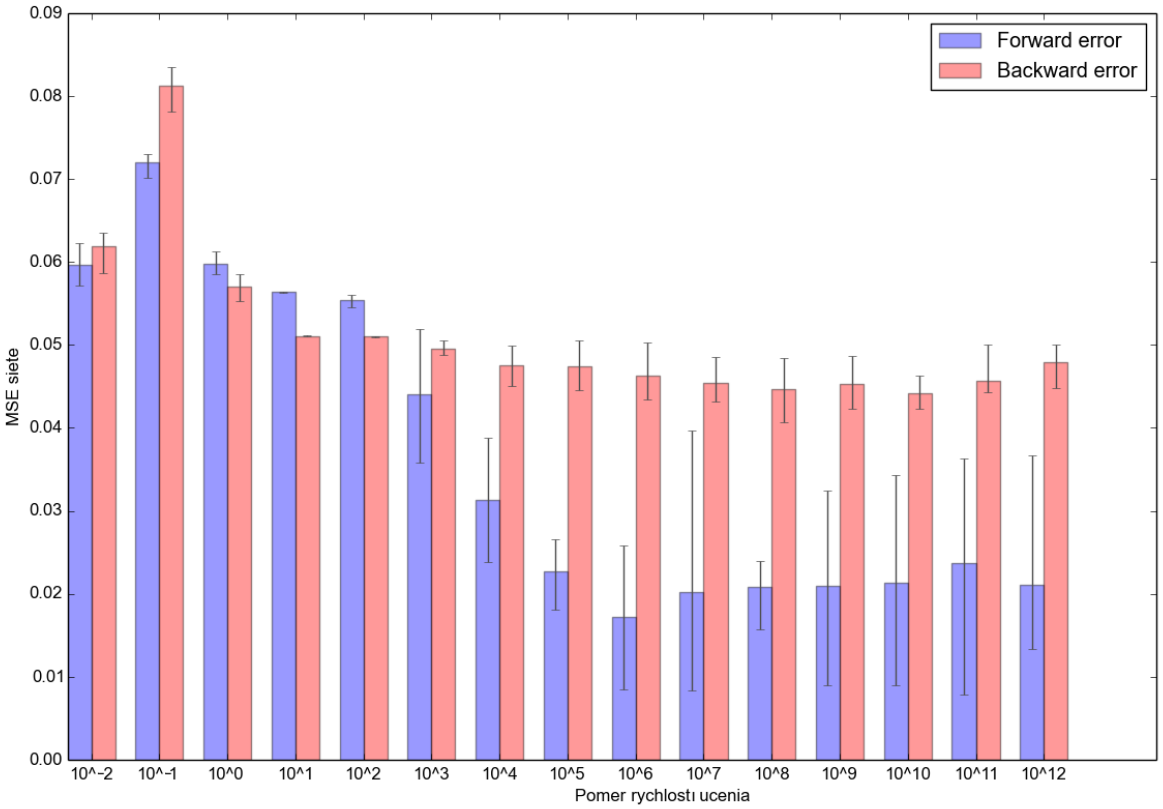
Výsledky tohto experimentu s rozdielnymi pomermi rýchlostí učenia sú znázornené grafom 5.7. Ako je aj z grafu vidieť veľkosť MSE spätného smeru bola rozdielnymi rýchlosťami učenia ovplyvnená iba minimálne. Avšak veľkosť MSE v doprednom smere sa výrazne znížila pri pomere učiacich rýchlostí väčších ako 10^4.

Z grafu je tiež vidieť, že pri nižších pomeroch rýchlostí učenia je oscilácia chyby od priemeru oveľa nižšia ako pri väčších pomeroch. Avšak aj najhoršie siete s vyšším pomerom rýchlostí učenia majú lepší výsledok v doprednom smere ako najlepšie siete s nižším pomerom rýchlostí učenia[2].

Pomocou tohto experimentu sme si vybrali za "zlatý" pomer rýchlostí učenia 10^6, kde priemerná sieť má najmenšiu chybu v doprednom smere a navyše najlepšia sieť všeobecne bola tiež trénovaná týmto pomerom.

[1] Táto idea pochádza z diplomovej práce Bc. Petra Csibu, ktorá v čase písania tohto textu ešte nebola publikovaná.

[2] Neberúc do úvahy "prechodné" pomery $1 : 10^3$ a $1 : 10^4$.

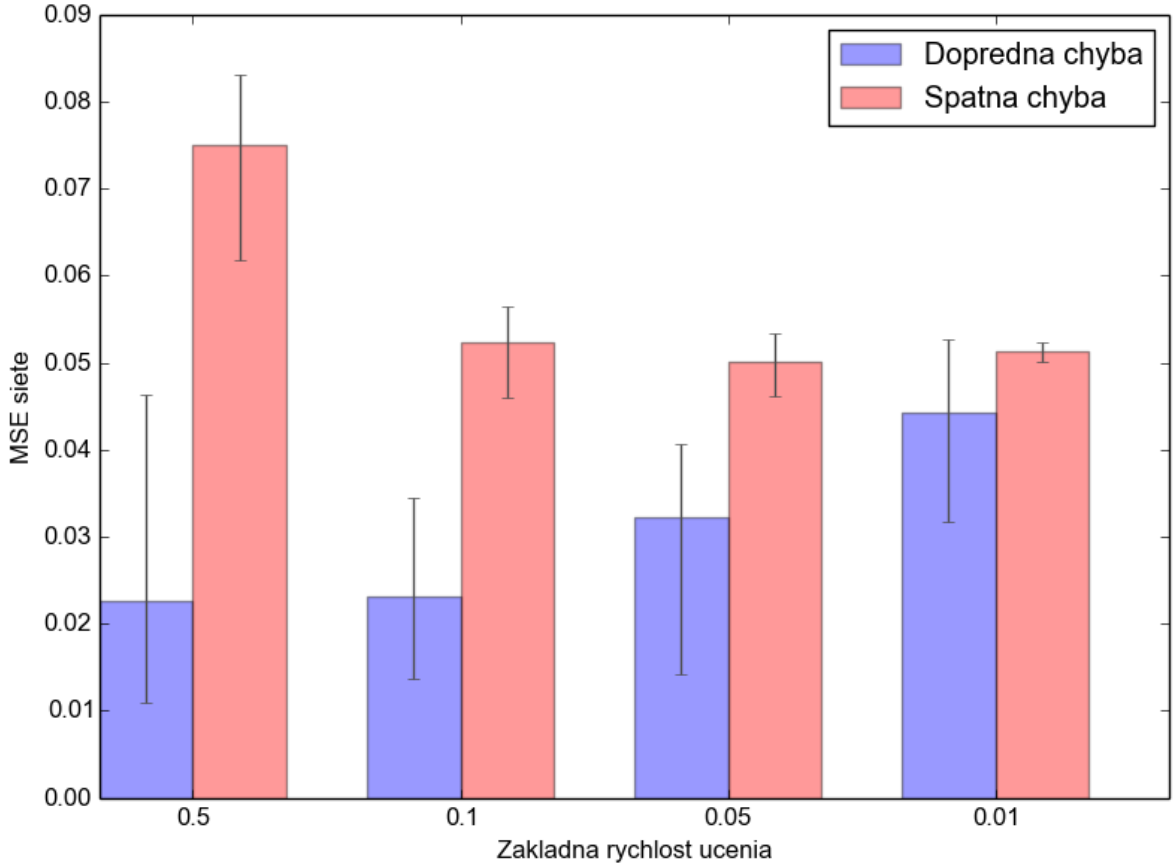


Obrázok 5.7: Graf znázorňuje prehľadávanie stavového priestoru pomerov učiacich rýchlostí pri trénovaní jednoduchých syntetických dát. Každý stĺpec znázorňuje priemernú chybu desiatich sietí. Modrou sú znázornené priemerné dopredné chyby a červenou sú znázornené priemerné spätné chyby. Čierne vodítka na grafoch znázorňujú zaznamenanú minimálnu a maximálnu chybu po epochách z desiatich sietí.

V ďalšom experimente sme sa rozhodli lepšie preskúmať tento zlatý pomer. V tomto experimente sme mali zafixovanú základnú rýchlosť učenia na 0.01 a hľadali sme ich ideálny pomer. V ďalšom experimente sme zafixovali pomer na 10^6 a prehľadávali sme stavový priestor pre základnú alfu.

Správanie siete sme preskúmali pre 4 rôzne rýchlosti učenia, pre každú 30 sieťami. Preskúmané rýchlosti sú 0.5, 0.1, 0.05 a 0.01. Počet skrytých neurónov sme nemenili, a teda zostal, tak ako v predchádzajúcom experimente, 35. V tomto experimente sme ale siete trénovali až v 5500 epochách.

Dosiahnuté úspešnosti sietí sú znázornené na grafe 5.8. Z grafu vidíme, že bez ohľadu na



Obrázok 5.8: Graf znázorňuje prehľadávanie stavového priestoru rôznych základných rýchlostí učenia využívajúc ich fixný zlatý pomer. Každý bar znázorňuje priemernú chybu tridsiatich sietí. Modrou sú znázornené priemerné dopredné chyby a červenou sú znázornené priemerné spätné chyby. Čierne vodítka na grafoch znázorňujú zaznamenanú priemernú minimálnu a priemernú maximálnu chybu zo všetkých sietí, takže niektoré individuálne siete mohli dosiahnuť lepší výsledok, ako ukazuje tento graf. Príkladom takejto siete je napríklad sieť opísaná grafom 5.9.

zvolenú rýchlosť učenia sa sieť nevie dostatočne naučiť zobrazenie v spätnom smere, avšak pri základnej rýchlosti učenia 0.5 sa tento smer naučí horšie ako pri ostatných základných rýchlostiach. V doprednom smere majú podľa tohto grafu siete lepšie výsledky pri vyšších rýchlostiach, avšak toto je spôsobené nedostatočným počtom epoch. Ako dôkaz uvádzame graf priebehu učenia jednej zo sietí učenej rýchlosťou 0.01 (5.9).

Na porovnanie uvádzame aj graf priebehu učenia jednej zo sietí, ktorých základná rýchlosť učenia bola 0.1 (5.10). Z grafu vidieť, že sieť je nestabilná a jej MSE aj v doprednom aj v

spätnom smere oscilujú oveľa viac ako pri sieti učenej nižšou základnou rýchlosťou učenia. Pri tejto sieti však pozorujeme, že v doprednom smere sa sieť po istom počte epoch mierne ustálila. Z grafu závislosti MSE od epochy vieme odhadnúť, že tejto sieti by sa viacerými epochami nepodarilo zlepšiť svoju chybu. Na lepšiu predstavu o natrénovanom zobrazení touto sieťou uvedieme aj chyby v centimetroch. V doprednom smere je priemerná chyba 2.2 centimetrov (minimum je 0.3 centimetrov a maximum 7.0 centimetrov) a v spätnom smere je priemerná chyba 3.9 centimetrov (minimum je 0.0 centimetrov a maximum 13.7 centimetrov).

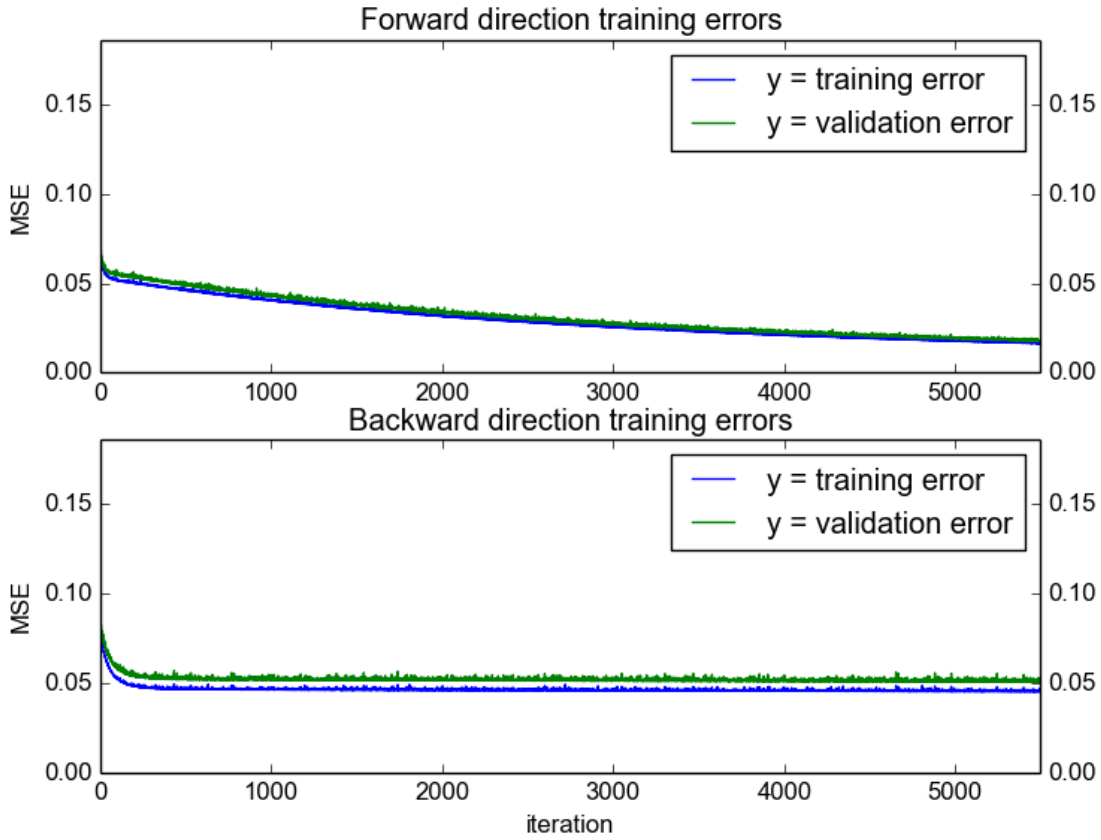
Vzhľadom na pôvodný BAL algoritmus, ktorý používa rovnaké rýchlosti učenia pre zmenu všetkých váh sa nám podarilo dosiahnuť priemerné zlepšenie o 3.83 centimetra v doprednom smere a o 0.911 centimetra v spätnom smere.

5.5 Pokusy z reálnymi robotickými dátami

V predchádzajúcich experimentoch sme našli parametre modifikovanáho učiaceho algoritmu BAL, ktorý používa na zmenu váh rôzne učiace rýchlosti, vďaka ktorým sa sieť bola schopná naučiť dopredné zobrazenie celkom spoľahlivo, no spätnom smere nemala sieť až také dobré výsledky, ale nepodarilo sa nám ich viac zlepšiť (priemerná odchýlka ramena od cielenej polohy bola 2.195 centimetra v doprednom smere a 3.872 centimetra v spätnom smere).

Dátovú množinu, ktorú sme používali v týchto experimentoch bola umelo vygenerovaná na pomocou Denavit-Hartenberg transformácii a teda neobsahovala šum a navyše jej dáta opisovali priestorové konfigurácie a konfigurácie servo motorov robotického ramena len s jediným kĺbom a dvoma stupňami voľnosti. V praxi sa ale používajú roboti s viacerými stupňami voľnosti. V tejto sekcii popíšeme experiment s nájdenými parametrami na zašumených dátach reálneho robotického ramena so štyrmi stupňami voľnosti.

Úloha trénovania siete na reálnych dátach je oveľa zložitejšia ako na našich syntetických dátach. Naše hardvérové rameno má 4 stupne voľnosti, narozdiel od pôvodných 2 vo vygenerovanej množine dát. Ďalej nám už nestačí určovať len polohu koncového efektora, ale sieť musí navyše určiť aj polohu posledného kĺbu. Ďalšou komplikáciou je prítomnosť šumu, ktorého zdroje sú nepresnosť servo motorov a nie ideálny stav niektorých kĺbov nášho robotického ramena a odchýlky pri výpočte 3D pozície efektora a kĺbov. Úlohu navyše robí ťažšou aj skutočnosť, že 3D súradnice sú v tejto množine dát udávané vzhľadom na stereo kameru, nie vzhľadom na rameno.

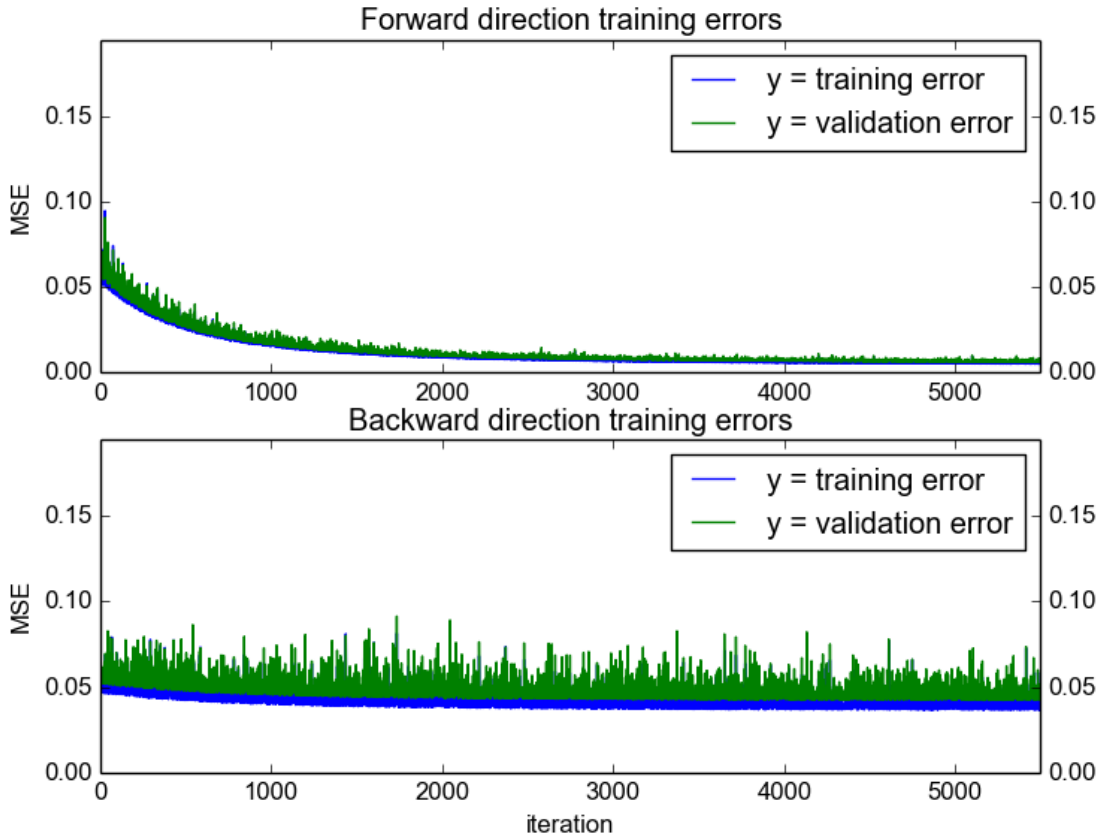


Obrázok 5.9: Graf znázorňuje priebeh učenia jednej zo sietí učenej zlatým pomerom a základnou rýchlosťou učenia 0.01. Sieť bola učená v 5500 epochách a má 35 skrytých neurónov. Z obrázku je vidieť, že ak by trénovanie siete pokračovalo ďalšími epochami, MSE dopredného smeru by sa pravdepodobne ešte znížilo. Ďalšie epochy by už ale pravdepodobne nemali žiadny vplyv na MSE spätného smeru, ktoré sa drží približne od 200. epochy na rovnakej úrovni.

Množina dát, ktoré sme používali na trénovanie a validáciu siete má 277 vzoriek. Z toho sme použili 150 vzoriek na trénovanie siete a 127 na validáciu.

Pri pokusoch s reálnymi dátami nebudeme uvádzať centimetrovú chybu v spätnom smere, nakoľko údaje o 3D pozícii nie sú vzhľadom na robotické rameno, ale vzhľadom na stereo kameru a nepoznáme transformáciu medzi týmito dvoma súradnicovými systémami.

Pre všetky hore spomínané sťaženia úlohy sme sa rozhodli zvýšiť počet neurónov na skrytej vrstve z 35 na 75. Pomer učiacich rýchlostí sme ponechali $1 : 10^6$. Za základnú rýchlosť učenia sme zvolili 0.05, čo je kompromis medzi rýchlim, ale nestabilným priebehom učenia pri základnej rýchlosti 0.1, ktoré sme videli v grafe 5.10, a pomalším, ale stabilnejším priebehom

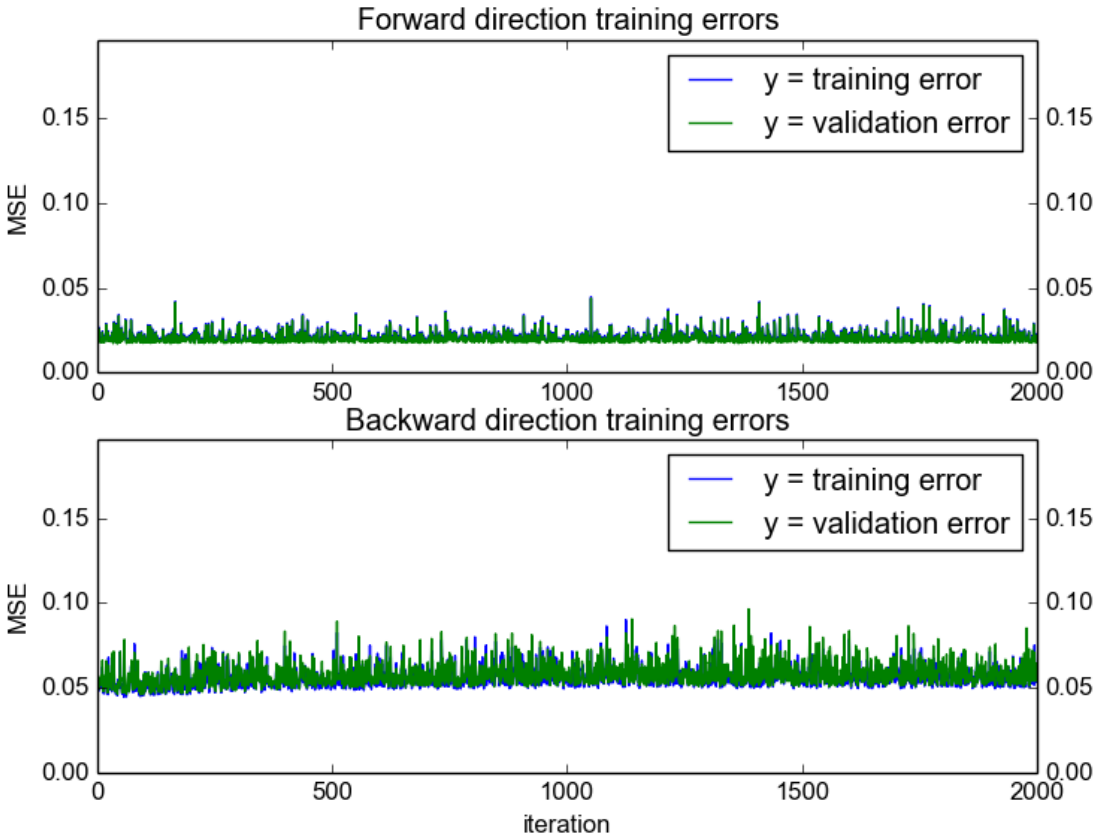


Obrázok 5.10: Graf znázorňuje priebeh učenia jednej zo sietí učenej zlatým pomerom a základnou rýchlosťou učenia 0.1. Sieť bola učená v 5500 epochách a má 35 skrytých neurónov. Z grafu je vidieť, že sieti by sa pokračovaním učenia v ďalších epochách pravdepodobne nepodarilo znížiť MSE ani v jednom smere.

učenia pri základnej rýchlosti 0.01, ktoré sme videli v grafe 5.9.

Trénovali sme 30 sietí na 2000 epochách. Všetky siete vo svojej najlepšej konfigurácii mali výslednú MSE dopredného smeru veľmi blízko pri 0.017. Najlepšia konfigurácia mala dopredné MSE 0.01649, najhoršia 0.01722. V spätnom smere mala väčšina sietí MSE približne 0.046. Najlepšia sieť mala 0.0455 a najhoršia 0.0473. V grafe **??** uvádzame krivku učenia siete s najlepším výsledkom.

Po preštudovaní priebehov učenia ostatných sietí sme zistili, že ani jedna sieť nebola schopná naučiť sa množinu reálnych dát. Priemer reálnej odchýlky od cieľových súradníc pre všetkých 30 trénovaných sietí je 33.036 centimetra a minimum je 28.785 centimetra. Na porovnanie, priemer odchýlky pre 30 netrénovaných sietí je 53.346.



Obrázok 5.11: Graf znázorňuje priebeh učenia siete s najlepším výsledkom pri trénovaní množiny dát reálneho robotického ramena. Ako je vidieť z grafu, sieť nebola schopná naučiť sa toto zobrazenie ani v jednom smere.

Ďalším experimentovaním s počtom skrytých neurónov, pomerom rýchlostí učenia a základnou alfou sa nám nepodarilo tieto výsledky vôbec zlepšiť.

Kapitola 6

Záver

V našej práci sme sa snažili vytvoriť prepojenie medzi vizuálnou informáciou získanou zo stereo kamery a proprioceptívnou informáciou robotického ramena.

Za cieľ sme si dali vytvorenie biologicky prijateľného riešenia pomocou neurónovej siete a podľa toho sme si aj zvolili učiaci algoritmus BAL, ktorý vychádza z algoritmu GeneRec narozdiel od ktorého neposkytuje len jednosmerné zobrazenie, ale poskytuje obojsmerné mapovanie medzi dátami. Počas práce sme sa rozhodli použiť modifikáciu algoritmu BAL, ktorá používa rozdielne učiace rýchlosti pre rozdielne váhy, vďaka ktorej sme boli schopný zvýšiť presnosť naučeného mapovania.

Podarilo sa nám nájsť model na prepojenie priestorovej konfigurácie jednoduchého robotického ramena a konfigurácie jeho servo motorov s odchýlkami pozície koncového efektora od očakávanej pozície o 2.195 centimetra pri hľadaní polohy koncového efektora a o 3.872 centimetra pri hľadaní konfigurácie servo motorov.

Pri overovaní tohto modelu na reálnych dátach sme zistili že model nebol schopný dostatočne sa naučiť zobrazenie medzi priestorovými konfiguráciami reálneho ramena a konfiguráciami servo motorov. Mal lepšiu úspešnosť ako náhodná sieť

Ako produkt práce vznikol skript napísaný v programovacom jazyku `python`, ktorý je schopný generovať nezašumené dáta popisujúce stavy robotického ramena. Taktiež sme zozbierali a spracovali dáta z reálneho prostredia, ktoré opisujú priestorovú konfiguráciu a konfiguráciu servo motorov hardvérového robotického ramena.

Pre účely práce sme tiež zostrojili robotickú platformu určenú hlavne na skúmanie algoritmov určených na učenie uchopovania objektov.

Do budúcnosti vidíme možnosť hlbšie skúmať a analyzovať vplyv rozdielnych učiacich

rýchlostí na výslednú úspešnosť siete učenej modifikovaným algoritmom BAL, poprípade skúmať možnosti alternatívnej inicializácie váh siete a ich vplyv na úspešnosť siete. Ďalej vidíme možnosť pridať do robotickej platformy ďalšie senzory alebo efektory, aby tak vzrástol potenciál zostrojenej robotickej platformy, napríklad pridanie stereo mikrofónu by umožnilo experimentovať s prepojením audio príkazov a správaním robotického ramena.

Dúfame, že táto práca poslúži ako základ na ďalšieho skúmania učiaceho algoritmu BAL a problému siahania na objekty.

Zoznam použitej literatúry

[1] Minoru Asada and KF MacDorman. Cognitive developmental robotics as a new paradigm for the design of humanoid robots. *Robotics and Autonomous Systems*, 37(2-3):185--193, 2001.

[2] F Crick. The recent excitement about neural networks. *Nature*, 337:129--132, 1989.

[3] I Farkaš and K Rebrová. Bidirectional Activation-based Neural Network Learning Algorithm. *Artificial Neural Networks and Machine Learning – ICANN 2013*, pages 154--161, 2013.

[4] B Gary and K Adrian. Learning OpenCV. 2008.

[5] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90--95, 2007.

[6] Randall C. O'Reilly. Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm. *Neural Computation*, 8(5):895--938, 1996.

[7] RC O'Reilly and Y Munakata. *Computational explorations in cognitive neuroscience: Understanding the mind by simulating the brain*. MIT press, 2000.

[8] E Oztop and M Kawato. Models for the control of grasping. *Sensorimotor Control of Grasping: Physiology and Pathophysiology, Cambridge University Press, Cambridge*, pages 110--124, 2009.

[9] K Rebrova, M Pechac, and I Farkas. Towards a robotic model of the mirror neuron system. *Development and Learning and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference*, 5:1--6, 2013.

[10] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. *Learning Internal Representations by Error Propagation*. ICS report. Institute for Cognitive Science, University of California, San Diego, 1985.

[11] Giovani Tessitore. *From Motor To Sensory Processing in Mirror Neuron Models: A Novel Computational Approach*. PhD thesis.

[12] Eric W. Weisstein. Spherical coordinates. From MathWorld---A Wolfram Web Resource. Last visited on 26/4/2014.

[13] D Zipser and R A Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331:679--684, 1988.

Príloha A - CD

Priložené CD obsahuje zdrojové všetky zdrojové kódy, ktoré sme pri práci použili.

- `ann.py` obsahuje implementáciu učiacich algoritmov BAL a GeneRec
- `robot.py` obsahuje funkcionalitu potrebnú na generovanie syntetických dát a implementáciu počítania Denavit-Hartenberg transformácii
- `camera.py` obsahuje implementáciu spracovania obrazu
- `ako_ziskat_realny_dataset.txt` obsahuje popis implementácie získavania 3d súradníc zo stereo obrázku

Na disku sa tiež nachádza elektronická verzia diplomovej práce.