UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

AKTÍVNE ROZPOZNÁVANIE OBJEKTOV V ROBOTICKOM SIMULÁTORE ICUB

Bakalárska práca

2013 **Michal Rataj**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

AKTÍVNE ROZPOZNÁVANIE OBJEKTOV V ROBOTICKOM SIMULÁTORE iCUB

Bakalárska práca

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

Bratislava, 2013 **Michal Rataj**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Michal Rataj
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	9.2.9. aplikovaná informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	slovenský

Názov:	Aktívne rozpoznávanie objektov v robotickom simulátore iCub
Cieľ:	1. Nainštalujte softvér potrebný pre simulátor humanoidného robota iCub. 2. Implementujte metódu, s využitím knižnice OpenCV, ktorá umožní vytvoriť reprezentáciu pozorovaného objektu. 3. Implementujte jednoduchý model neurónovej siete, ktorá umožní aktívne rozpoznávanie objektov na základe ich manipulácie.
Literatúra:	http://opencv.org http://eris.liralab.it/wiki/Main_Page
Anotácia:	Účelom zadania je efektívna funkcionalita robota, ktorá mu umožní pracovať s objektami. Ťažisko spočíva v naštudovaní si problematiky spracovania obrazov, so zameraním sa na extrakciu príznakov a v implementácii doprednej neurónovej siete, ktorá napomôže robotovi vhodne manipulovať s objektom s cieľom rozpoznať ho.
Poznámka:	Očakávam pravidelné konzultácie a systematickú prácu.
Kľúčové slová:	iCub, rozpoznávanie objektov, extrakcia príznakov, neurónová sieť

Vedúci:	doc. Ing. Igor Farkaš, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	doc. PhDr. Ján Rybár, PhD.
Dátum zadania:	17.10.2012

Dátum schválenia: 17.10.2012

doc. RNDr. Mária Markošová, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

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

.....

Michal Rataj

Ďakujem môjmu vedúcemu doc. Ing. Igorovi Farkašovi, PhD. za jeho čas a cenné rady počas písania práce. Tiež ďakujem svojej rodine a priateľom za neustálu podporu.

Abstrakt

V našej práci riešime jeden z problémov robotiky, a to identifikáciu uchopeného objektu. Náš model je navrhnutý na aktívne rozpoznávanie uchopeného objektu pomocou jeho manipulácie. Je založený na dopredných neurónových sieťach a trénovaní z učiteľom. Pri práci nepoužívame skutočného robota iCub, ale jeho simulátor. Identifikáciu sme si rozdelili na niekoľko menších častí. Jeden modul je trénovaný na rozpoznávanie farby objektu, druhý na rozpoznávanie tvaru objektu a tretí má za účel naučiť sa akcie, ktoré má robot vykonať pre lepší odhad tvaru uchopeného objektu. V kapitole o implementácii uvádzame návrh a parametre týchto modulov. V časti o výsledkoch uvádzame ich výslednú úspešnosť pri rozpoznávaní farby a tvaru, ako aj porovnanie pasívneho a aktívneho prístupu k rozpoznávaniu objektov.

Kľúčové slová: aktívne rozpoznávanie objektov, iCub simulátor, neurónové siete, učenie s učiteľom, extrakcia príznakov, OpenCV, FANN.

Abstract

In our work we are dealing with one of the problems of robotics – identification of grasped object. Our module is designed for an active object recognition of grasped object by manipulation. It is based on feed-forward neural networks and supervised learning. However instead of using real robot iCub we decided for a simulator. We have divided identification into several smaller parts. One module is trained to recognize colour of the object, second to recognize shape of the object and third is aimed at learning actions, which should help the robot to better assess the shape of the object. In a chapter concerning the implementation we present the design and parameters of these modules. In the next chapter we evaluate positive results in recognizing colours and shape, as well as a comparison between passive and active approach to object recognition.

Key words: active object recognition, iCub simulator, neural networks, supervised learning, features extraction, OpenCV, FANN.

Obsah

Úvod 1

Teória 2

1.1 Umelá neurónová sieť 2

1.2 Perceptrón 2

1.3 Aktivačné funkcie 3

1.4 Učenie 4

1.5 Viacvrstvový perceptrón (Multilayer perceptron) 6

1.6 Algoritmus backpropagation (spätné šírenie chyby) 7

1.7 SOM 9

1.8 Učenie SOM 10

1.9 Algoritmus SOM 11

Prostredie 15

2.1 ICub 15

2.2 Simulátor iCub 16

2.3 YARP 17

2.4 Scéna 17

2.5 FANN 18

Implementácia 19

3.1 Komunikácia so simulátorom 19

3.2 Spracovanie obrazu 20

3.2.1 Obraz z iCuba 20

3.2.2 Thresholding 21

3.2.3 Rozostrenie 21

3.2.4 Kontúry a ohraničenia 22

3.3 Extrakcia príznakov 23

3.3.1 Extrakcia príznakov farby 24

3.3.2 Extrakcia príznakov tvaru 25

3.3.3 Vyhodnotenie 26

3.4 Vizualizácia dát 26

3.4.1 Matlab 27

3.4.2 Výsledky pre farbu 27

3.4.3 Výsledky pre tvar 29

3.4.4 Výsledky pre kontúry 30
3.4.5 Výsledky pre opísané útvary 31
3.4.6 Vyhodnotenie 33
3.5 Moduly pre rozpoznávanie farby a tvaru objektu 34
3.6 Modul pre aktívnu manipuláciu objektov 35
Výsledky 36
Záver 40
Literatúra 41
Príloha A - CD 42

Úvod

Žijeme v ére stále sa rozvíjajúcich digitálnych technológii, ktoré nás ovplyvňujú každý deň. Pomáhajú nám v práci aj v bežnom živote a možno si ich prítomnosť až tak neuvedomujeme, pokiaľ nám nejaká z nich nezačne výraznejšie chýbať. Vývoj digitálnych technológii stále napreduje a s ním sa nám vynárajú stále nové problémy a výzvy. Niektoré úlohy sú však už také náročné, že ich nezvládajú robiť ani ľudia. Preto ich riešenie dali do rúk strojom – robotom. Väčšina z nich sú iba priemyselné automatizované prístroje, stvorené na monotónnu činnosť, ktorú vykonávajú neúnavne a stále dokola. Niektoré vlastne ani nevykonávajú nejakú pre človeka neriešiteľnú úlohy, máme ich jednoducho preto lebo nám zjednodušujú život. Človeka však odjakživa fascinovala predstava stvoriť svoju kópiu – po stránke vzhľadovej aj rozumovej. Po fyzickej, vzhľadovej stránke robia stále pokroky a výrazne napredujú. Humanoidným robotom sa v dnešnej vedeckej a akademickej výskumnej činnosti prikladá čoraz väčší význam. Konštruktéri sa stále zdokonaľujú pri ich konštrukcii a možnostiach pohybu. Čo nás však ešte stále oddeľuje od robotov a robí nás ľuďmi je náš mozog, ktorý zatiaľ nedokážeme simulovať. Je to komplexný, nelineárny a paralelný systém. Mnohé schopnosti ako rozpoznávanie, vnímanie a motorická kontrola, schopnosť učiť sa, schopnosť pamätať si a generalizovať, podnietili vedcov ku konštrukcii modelu biologického neurónového systému – známeho ako umelá neurónová sieť.

V našej práci sme sa rozhodli navrhnúť a implementovať model neurónovej siete na aktívne rozpoznávanie uchopeného objektu (farba aj tvar). Boli sme inšpirovaní článkom od Browatzkeho a kol., 2012, v ktorom autori predstavujú výhody aktívneho prístupu k rozpoznávaniu objektov oproti pasívnemu prístupu, kde sú volené náhodné alebo dopredu zvolené sekvencie pohybov robota.

Náš model budeme testovať v robotickom simulátore iCub. Ten bol zostrojený ako simulátor skutočného robota iCub, ktorý má parametre približne 2,5 ročného dieťaťa. Simulátor zostrojili pre vedecké a akademické tímy, ktoré si nemôžu skutočného robota dovoliť, alebo aby sa odladili algoritmy, ktoré môžu poškodiť okolie alebo samotného robota.

Na záver práce uvedieme kapitolu výsledkov, v ktorej zhrnieme testovanie siete na rôznych dátach a uvedieme úspešnosť sietí pri rozpoznávaní jednotlivých objekto

Kapitola 1

Teória

1.1 Umelá neurónová sieť

Umelé neurónové siete boli inšpirované štúdiom ľudského mozgu. Úsilie o vymodelovanie neurónovej siete sa nám vráti v podobe schopnosti riešiť komplexné problémy s dobrou efektivitou. Medzi najbežnejšie oblasti využitia ANN patria: klasifikácia – cieľom je určiť triedu vstupného vektora, rozpoznávanie vzorov – nájsť vzor, ktorý sa najviac podobá zadanému vektoru alebo doplniť jeho chýbajúcu časť, optimalizácia – nájsť optimálne hodnoty parametrov, kontrola – navrhnúť primerané opatrenia podľa zadaného vektora, aproximátor rôznych funkcii. Umelá neurónová sieť je vlastne nelineárne mapovanie z R^I do R^K

$$f_{NN} = R^I \rightarrow R^K$$

kde I a K sú rozmery vstupného a cieľového (požadovaný výstup) priestoru. Funkcia f_{NN} je zvyčajne komplexná funkcia skupiny nelineárnych funkcii (každému neurónu v sieti prislúcha jedna funkcia) (Engelbrecht, 2007).

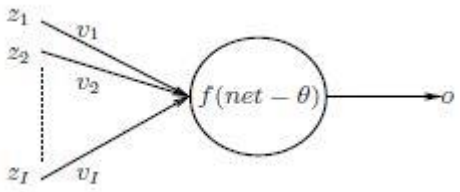
1.2 Perceptrón

Umelé neuróny (AN – artificial neuron) tvoria základ umelých neurónových sietí. Perceptrón je najjednoduchší model siete pozostávajúci len z jednej vstupnej a jednej výstupnej vrstvy. Implementuje nelineárne mapovanie z R^I zvyčajne do množiny [0, 1] alebo [-1, 1] v závislosti od použitej aktivačnej funkcie (obrázok 1.1). Na vstupe dostane I-rozmerný vektor vstupných signálov:

$$z = (z_1, z_2, \dots, z_I)$$

Tento môže dostať z prostredia alebo priamo od iného neurónu. Ku každému vstupu z_I sa priradí váha v_I, ktorá má za úlohu zosilniť alebo zoslabiť vstupný signál. Neurón vypočíta hodnotu *net* vstupného signálu a následne za pomoci aktivačnej funkcie f_{AN} vypočíta hodnotu výstupného signálu o. Sila výstupného signálu je ďalej ovplyvnená hraničnou hodnotou θ, ktorá sa bežne nazýva *bias*. Hodnota *net* sa zvyčajne počíta ako váhovaná suma všetkých vstupných signálov (Engelbrecht, 2007):

$$net = \sum_{i=1}^{I} z_i v_i$$



Obr. 1.1: Umelý neurón (Engelbrecht, 2007)

Hlavná myšlienka trénovacej procedúry perceptrónu je takáto: najskôr zaznamenajme odpoveď každého formálneho neurónu na daný podnet. Ak je odpoveď správna, nemodifikujme váhy. Ak je odpoveď daného neurónu nesprávna, potom modifikujme váhy všetkých aktivovaných vstupných synáps, a to nasledovným spôsobom: ak má byť neurón aktívny a nie je, zväčšime ich, a naopak, ak má byť na výstupe neurónu 0 a nie je, zmenšime ich (Kvasnička a kol., 1997).

Ukázalo sa však, že tieto siete vôbec nie sú výpočtovo univerzálne a nedokážu riešiť všetky triedy problémov. Hlavne však išlo o to, že perceptróny nedokážu riešiť tzv. lineárne neseparovateľné problémy. Klasickým a najjednoduchším príkladom zlyhania je logická funkcia XOR (vylučujúce alebo) (Kvasnička a kol., 1997).

1.3 Aktivačné funkcie

Funkcia f_{AN} dostane vstupný signál siete *net* a hodnotu *bias* a podľa toho sa rozhodne pre výstup neurónu. Táto funkcia sa tiež nazýva ako aktivačná funkcia. Môžeme využiť viaceré druhy aktivačných funkcii. Vo všeobecnosti sú aktivačné funkcie monotónne rastúce funkcie, kde platí (okrem lineárnej funkcie):

$$f_{AN}(-\infty) = 0 \text{ alebo } f_{AN}(-\infty) = -1 \text{ a zároveň } f_{AN}(\infty) = 1$$

Bežne používané aktivačné funkcie, ktoré sa využívajú aj v praxi:

1. **Lineárna funkcia**:

$$f_{AN}(net) = \lambda(net)$$

kde λ je sklon funkcie. Lineárna funkcia vypočíta lineárne modifikovaný výstup, kde λ je konštanta.

2. **Sigmoidná funkcia**:

$$f_{AN}(net) = \frac{1}{1 + e^{-\lambda(net)}}$$

kde parameter λ určuje strmosť funkcie, zvyčajne $\lambda = 1$. $f_{AN}(net) \in (0, 1)$.

3. **Hyperbolický tangens**:

$$f_{AN}(net) = \frac{e^{\lambda(net)} - e^{-\lambda(net)}}{e^{\lambda(net)} + e^{-\lambda(net)}}$$

po aproximácii:

$$f_{AN}(net) = \frac{2}{1 + e^{-\lambda(net)}} - 1$$

Výstup hyperbolického tangensu je v intervale $f_{AN}(net) \in (-1, 1)$.

Medzi ďalšie používané aktivačné funkcie patria napríklad: skoková funkcia, lineárne rastúca funkcia a gaussova funkcia (Engelbrecht, 2007).

1.4 Učenie

Otázka, ktorú zostáva zodpovedať je, či existuje nejaký automatizovaný prístup pre stanovenie hodnôt v_I a prahovej hodnoty θ? Pre jednoduché problémy sa dajú tieto hodnoty ľahko vypočítať. Ale predpokladajme, že nemáme žiadne predošlé vedomosti o funkcii (okrem dát), ako môžeme potom vypočítať hodnoty v_I a θ? Odpoveď je pomocou učenia. Umelé siete sa učia najlepšie hodnoty pre v_I a θ zo vstupných dát. Učenie pozostáva z upravovania váh v_I a prahovej hodnoty θ pokiaľ nie je splnená určitá podmienka (alebo niekoľko podmienok zároveň). Toto učenie prebieha podľa istých pravidiel. Tieto pravidlá ovplyvňujú hodnoty v sieti a jej budúce správanie (Engelbrecht, 2007).

Poznáme tri hlavné paradigmy učenia:

Supervised learning (učenie s učiteľom). Po technickej stránke idea spočíva v tom, že keď poznáme pre zadaný vstup korešpondujúci výstup, zadáme ho sieti a jej úlohou je nájsť mapovanie zo vstupnej hodnoty na výstupnú. V aplikáciách používajúcich trénovanie s učiteľom musia byť známe vstupné a k nim korešpondujúce výstupné hodnoty a musia byť poskytnuté trénovaciemu algoritmu (tieto dáta nazývame tréningová množina). Úlohou siete je nájsť správne mapovanie. Váhy v sieti sa menia v závislosti od veľkosti chyby na výstupnej vrstve. Čím je chyba väčšia – rozdiel medzi skutočným výstupom siete a správnou hodnotou výstupu (požadovaný výstup) pre zadaný vstup – tým viac sa váhy upravia. Voľne by sa dalo hovoriť o učení, povedzme malého dieťaťa, s učiteľom, ktorý mu pomáha, vedie ho. Takáto definícia je však pomerne nejasná a ťažko sa prekladá do podoby konkrétneho algoritmu pre neurónové siete (Pfeifer, 2010).

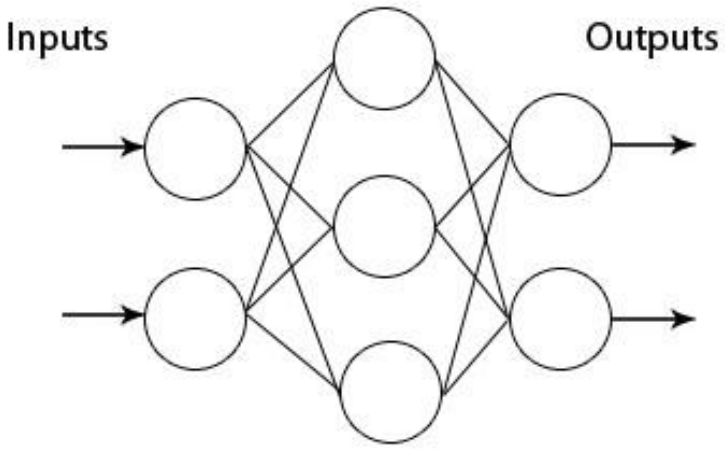
Reinforcement learning (učenie pomocou odmeny a trestu, alebo tiež učenie posilňovaním). Cieľom je odmeniť neurón (alebo časť neurónovej siete) za dobrý výkon a penalizovať neurón za zlý výkon. Idea spočíva v tom, že učiteľ povie študentovi iba či je odpoveď (výstup) správna alebo nie. Úloha zistiť prečo je odpoveď správna alebo nesprávna prenecháva na študentovi. Používa sa napríklad keď chceme dosiahnuť určité správanie. Typicky robot dostane pozitívnu odozvu (odmenu) ak výsledok bol dobrý a žiadnu alebo negatívnu odozvu (trest) keď bol výsledok akcie zlý. Napríklad keď robot zdvihne objekt, nájde cestu z bludiska alebo odpáli loptičku do bránky, dostane pozitívnu odmenu. Vo všeobecnosti tento algoritmus nie je spätý len s neurónovými sieťami. V ideálnom prípade sa agent snaží maximalizovať množstvo odmien, ktoré dostane, interakciou s komplexným prostredím okolo seba (Pfeifer, 2010) (Engelbrecht, 2007).

Unsupervised learning (učenie bez učiteľa). Cieľom je nájsť vzor alebo príznak vo vstupných dátach bez akejkoľvek pomoci z externého zdroja. Tieto algoritmy prevažne vykonávajú klasterizáciu (zhlukovanie) vstupných dát (tréningových vzoriek). Zástupcami sú napríklad: hebbovské učenie a Kohonenove mapy. Idea Hebbovho algoritmu spočíva v tom, že ak sú dva neuróny aktívne súčasne (alebo v nejakom časovom okne), je väzba medzi nimi posilnená. Tento prístup sa stal populárny preto, lebo mu stačia len lokálne informácie a je aj biologicky hodnoverný. V priemyselnej oblasti sa však nepoužíva.

Kohonenove mapy (tiež SOM – samo-organizujúce sa mapy) sa používajú na nájdenie zhlukov vo vstupnej trénovacej množine. Tiež majú istú podobnosť s biologickým modelom. Navyše získali široké uplatnenie v praxi. Detailnejšie si ich predstavíme neskôr (Pfeifer, 2010) (Engelbrecht, 2007).

1.5 Viacvrstvový perceptrón (Multilayer perceptron)

Jednoduchý perceptrón opísaný v časti 1.2 mal jedno vážne obmedzenie. A to, že nedokázal riešiť problémy, ktoré neboli lineárne separovateľné. Toto obmedzenie už ale neplatí pre viacvrstvovú doprednú sieť alebo inak viacvrstvový perceptrón (obrázok 1.2). Ten ako už z názvu vyplýva obsahuje jednu vstupnú vrstvu neurónov, jednu alebo viac „skrytých“ vrstiev neurónov a jednu výstupnú vrstvu neurónov. Bude obsahovať aj viac ako jednu aktivačnú funkciu, pričom nemusia byť všetky rovnaké. Aj keď môže obsahovať aj viac skrytých vrstiev, bolo dokázané, že už pomocou jednej vrstvy skrytých neurónov môžeme reprezentovať ľubovoľnú booleovskú funkciu (aj spomínanú funkciu XOR). Hoci sa o ich funkčnosti vedelo už dávnejšie, – najmä vďaka tomu, že sa používali ako funkčný univerzálny aproximátor funkcii – výraznejšiu pozornosť si vyslúžili až s príchodom backpropagation algoritmu (Pfeifer, 2010).



Obr. 1.2: Viacvrstvová neurónová sieť. Medzi vstupnou a výstupnou vrstvou je jedna alebo viac skrytých vrstiev. Neuróny v každej vrstve majú nejaké aktivačnú funkciu (nemusia byť všetky rovnaké).

1.6 Algoritmus backpropagation (spätné šírenie chyby)

Algoritmus spätného šírenia chyby je nosnou časťou v mnohých prácach s témou neurónových sietí. Možno ho rozdeliť na dve fázy: predný prechod – vypočítame výstup siete, spätné šírenie – postupujeme od výstupnej vrstvy a podľa veľkosti chyby upravujeme príslušné váhy.

Označme si vstupnú vzorku μ, vstup k potom bude nastavený na ξ_k^μ, keď sa prezentuje vzorka μ. ξ_k^μ môže nadobúdať hodnoty buď z intervalu (0, 1) alebo reálnych čísel. N je počet vstupných neurónov, p je počet vstupných vzoriek ($\mu = 1, 2, \dots, p$).
Pre vzorku μ je vstup do neurónu j v skrytej vrstve (vrstva V):

$$h_j^\mu = \sum_k w_{jk}\xi_k^\mu$$

aktivácia neurónu v skrytej vrstve V_j^μ potom bude:

$$V_j^\mu = g\left(h_j^\mu\right) = g(\sum_k w_{jk}\xi_k^\mu)$$

kde g je aktivačná funkcia (napríklad sigmoidná). Výstup neurónu i vo výstupnej vrstve O:

$$h_i^\mu = \sum_j w_{ij}V_j^\mu = \sum_j w_{ij}g(\sum_k w_{jk}\xi_k^\mu)$$

a prechodom aktivačnej funkcie g dostávame výstup:

$$O_i^\mu = g\left(h_i^\mu\right) = g\left(\sum_j w_{ij}V_j^\mu\right) = g(\sum_j w_{ij}\, g(\sum_k w_{jk}\xi_k^\mu))$$

Hodnotu prahovej premennej, bias θ, sme vo výpočte vynechali. Môžeme ho zahrnúť pridaním jedného extra neurónu, ktorý bude spojený so všetkými jednotkami v siete a bude nastavený na hodnotu (-1). Váhy z tohto neurónu reprezentujú prahovú hodnotu každej jednotky.

Chybová funkcia je opäť definovaná pre všetky výstupné jednotky a pre všetky vzorky (hodnota ζ je požadovaná hodnota výstupu):

$$E(\underline{w}) = \frac{1}{2}\sum_{\mu,i}\left[O_i^\mu - \zeta_i^\mu\right]^2 = \frac{1}{2}\sum_{\mu,i}\left[g(\sum_j w_{ij}V_j^\mu) - \zeta_i^\mu\right]^2 = \frac{1}{2}\sum_{\mu,i}\left[g(\sum_j w_{ij}\, g(\sum_k w_{jk}\xi_k^\mu)) - \zeta_i^\mu\right]^2$$

Zodpovedajúce váhy môžeme následne upraviť:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \sum_{\mu} \left[o_i^{\mu} - \zeta_i^{\mu}\right] g'(h_i^{\mu}) V_j^{\mu} = -\eta \sum_{\mu} \delta_i^{\mu} V_j^{\mu}$$

$$\text{pričom } \delta_i^{\mu} = \left[o_i^{\mu} - \zeta_i^{\mu}\right] g'(h_i^{\mu})$$

V prípade sigmoidnej funkcie je derivácia rovná:

$$g'\left(h_i^{\mu}\right) = O_i^{\mu}(1 - O_i^{\mu})$$

$$\delta_i^{\mu} = (o_i^{\mu} - \zeta_i^{\mu}) O_i^{\mu}(1 - O_i^{\mu})$$

Váhy zo skrytej vrstvy do výstupnej vrstvy upravíme nasledovne (sigmoidná funkcia):

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta \sum_{\mu} \delta_i^{\mu} V_j^{\mu} = -\eta \sum_{\mu} (o_i^{\mu} - \zeta_i^{\mu}) O_i^{\mu}(1 - O_i^{\mu}) V_j^{\mu}$$

Na zmenu váh zo vstupnej do skrytej vrstvy treba použiť pri derivovaní reťazové pravidlo, kde po pár krokoch dostávame (Pfeifer, 2010):

$$\Delta w_{jk} = -\eta \sum_{\mu} \delta_j^{\mu} \xi_k^{\mu}$$

$$\text{pričom } \delta_j^{\mu} = \left(\sum_i w_{ij} \delta_i^{\mu}\right) g'\left(h_j^{\mu}\right) = \left(\sum_i w_{ij} \delta_i^{\mu}\right) V_j^{\mu}(1 - V_j^{\mu})$$

Algoritmus 1.1 Algoritmus spätného šírenia chyby (pseudokód)

---

m: index vrstvy; M: počet vrstiev; m = 0 : vstupná vrstva; $V_i^0 = \xi_i$; váha w_{ij}^m : z V_j^{m-1} do V_i^m; ζ_i^{μ}: požadovaný výstup;

(1.) Inicializuj váhy na malé náhodné čísla.

(2.) Vyber vzorku ξ_k^{μ} z tréningovej množiny, použi ju na vstupnú vrstvu (m = 0)
$V_k^0 = \xi_k^{\mu}$ pre všetky k.

(3.) Postupuj s aktiváciou cez celú sieť:
$V_i^m = g(h_i^m) = g(\sum_j w_{ij}^m V_j^{m-1})$
pre všetky hodnotu *i* a *m* pokiaľ nie sú vypočítané všetky hodnoty V_i^M (V_i^M = aktivácia neurónov na výstupne vrstve).

(4.) Vypočítaj hodnotu delta pre výstupnú vrstvu *M*:
$\delta_i^M = g'(h_i^M)\left[\zeta_i^{\mu} - V_i^M\right]$, pre sigmoidu: $\delta_i^M = V_i^M(1 - V_i^M)\left[\zeta_i^{\mu} - V_i^M\right]$

(5.) Vypočítaj delta pre predošlé vrstvy pomocou postupného spätného šírenia chyby:
$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m$, pre $m = M, M-1, M-2, \ldots, 2$ pokým nie je delta vypočítaná pre každú jednotku.

(6.) Použi:
$\Delta w_{ij}^m = \eta\, \delta_i^m V_j^{m-1}$ na úpravu všetkých spojení pomocou: $w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}$

(7.) Vráť sa na krok 2. a postup opakuj pre ďalšiu vzorku.

(Pfeifer, 2010)

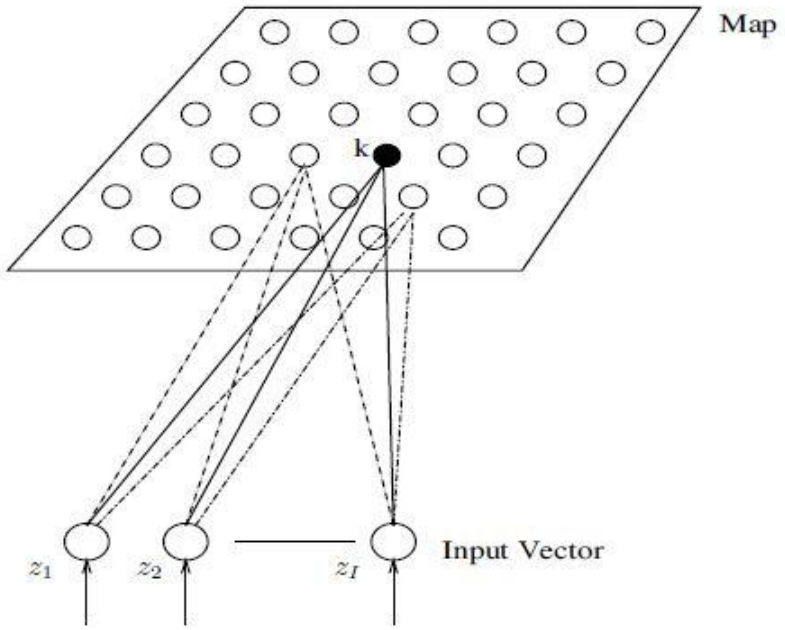
1.7 SOM

Self-organizing map (SOM) alebo tiež self-organizing features map (SOFM) – samoorganizujúca sa mapa – je typ umelej neurónovej siete, ktorá znázorňuje nízko-rozmernú (typicky dvoj-rozmernú), diskrétnu reprezentáciu zo vstupov trénovacej množiny, zvanú **mapa**. Tento model bol ako umelá neurónová sieť po prvý raz opísaný fínskym profesorom Teuvo Kohonenom, preto sa niekedy nazýva aj ako Kohonenova mapa alebo Kohonenova sieť (Kohonen, 1982).

Kohonen vyvinul samoorganizujúce sa mapy motivovaný charakteristickou samo organizujúcou sa schopnosťou ľudskej mozgovej kôry. Štúdie ľudskej mozgovej kôry preukázali, že motorická kôra, somatosenzorická kôra, vizuálna ako aj zvuková kôra sú reprezentované ako topologicky usporiadané mapy. Tieto topologické mapy sa formujú ako reprezentácia vstupných vnemov získaných zmyslovými senzormi (Engelbrecht, 2007).

Samoorganizujúca sa mapa je teda multirozmerná škálovacia metóda ako projektovať I-rozmerný vstupný priestor na diskrétny výstupný priestor, efektívne využívajúc kompresiu vstupného priestoru na sadu kódovaných vektorov. Výstupný priestor je zvyčajne dvoj-rozmerná mriežka. SOM využíva mriežku na aproximáciu funkcie hustoty pravdepodobnosti vstupného priestoru, pritom však zachováva topologickú štruktúru vstupného priestoru. To znamená, že ak sú dva vektory blízko seba vo vstupnom priestore, je tomu tak i v mapovej reprezentácii. Dôležité je to, že susedné neuróny majú tiež vplyv na rozloženie neurónov v mape.

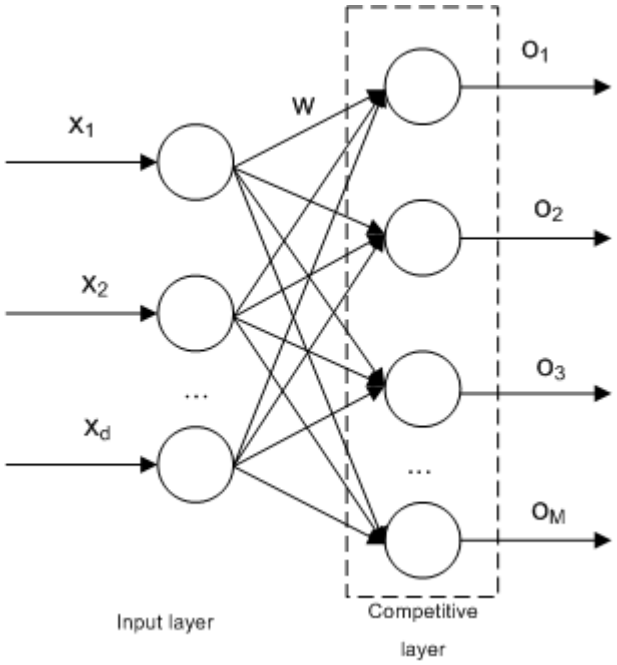
SOM sieť sa skladá zo vstupnej vrstvy a vrstvy obsahujúcej „súťažné“ neuróny, ktoré sú usporiadané v dvojrozmernej mriežke (obrázok 1.5). Každý z týchto vektorov je opísaný vektorom váh $w_i = (w_{i1}, \dots, w_{id})^T$. Keď je vstupný vektor $x = (x_1, \dots, x_d)$ prezentovaný sieti, neuróny na mriežke súťažia a vybratý je víťaz m, ktorého váhový vektor w_m je najviac podobný x (Kohonen, 2001).



Obr. 1.5: Samo organizujúca sa mapa (Engelbrecht, 2007).

1.8 Učenie SOM

Samoorganizujúce sa mapy implementujú učenie bez učiteľa (unsupervised learning), technika, ktorú využívajú sa dá nazvať „súťaženie“ lebo neuróny na výstupe medzi sebou súťažia (obrázok 1.3). Každý neurón i je opísaný vektorom váh $w_i = (w_{i1}, \dots, w_{id})^T$, $i = 1, \dots, M$ a vypočíta sa podobnosť medzi vektorom vstupných dát $x^n = (x_{n1}, \dots, x_{nd})^T \in R^d$ a váhovým vektorom w_i. Pre každý vstupný vektor, neuróny „súťažia“ medzi sebou o víťazný neurón, ktorého váhový vektor sa najviac podobá na tento konkrétny vstupný vektor. Víťazný neurón m nastaví svoj vstup ako $o_i = 1$ a všetky ostatné neuróny nastavia svoje výstupy ako $o_i = 0, i = 1, \dots, M, i \neq m$. Zvyčajne sa na výpočet podobnosti využíva funkcia inverznej Euklidovskej vzdialenosti $||x - w_i||$ medzi vstupným vektorom x^n a váhovým vektorom w_i (Kvasnička a kol., 1997). Hlavnými predstaviteľmi takéhoto učenia sú tzv. Learning vector quantization (LVQ) a Self-organizing map (SOM).



Obr. 1.3: Architektúra modelu neurónovej siete bez učiteľa. Neuróny na výstupnej vrstve (na obr. competitive layer) súťažia o to, ktorý je najbližšie zadanému vstupu (http://jsalatas.ictpro.gr).

1.9 Algoritmus SOM

Algoritmus začína inicializovaním váhových vektorov $w_i = (w_{i1}, \dots, w_{id})^T$ na malé náhodné hodnoty produkované generátorom náhodných čísel. Po tomto kroku nastávajú tri hlavné úrovne, ktoré môžeme zosumarizovať takto (Haykin, 2008):

- *Súťaž*: Pre každú trénovaciu vzorku x^n neuróny na mriežke vypočítajú hodnotu funkcie podobnosti. Neurón s väčšou hodnotou funkcie podobnosti sa stane víťazom. Ako funkcia podobnosti sa zvyčajne používa Euklidovská vzdialenosť medzi vstupným vektorom $x_n = (x_1, \dots, x_d)^T$ a váhovým vektorom $w_i = (w_{i1}, \dots, w_{id})^T$ každého jedného neurónu .
- *Spolupráca*: Víťazný neurón definuje v mriežke topologických suseda. Neuróny, ktoré prislúchajú tomuto susedovi, všetky upravia hodnoty svojich váhových vektorov pre konkrétny, zadaný vstup. Základným problémom je definícia topologického suseda. Zadefinujme si $h_{j,i}$ topologického suseda s centrom vo víťaznom neuróne. Topologický sused pozostáva zo sady neurónov a j je náhodný neurón z tejto sady. Tiež si definujme $d_{j,i}$ ako vzdialenosť medzi víťazným neurónom i a neurónom j. Môžeme predpokladať, že topologický sused $h_{j,i}$ je funkciou $d_{j,i}$ spĺňajúcou nasledujúce podmienky:

- Je symetrická k bodu, kde nadobúda najväčšiu hodnotu a kde $d_{j,i} = 0$, alebo inými slovami k bodu víťazného neurónu.
- Amplitúda funkcie klesá monotónne, ako sa zvyšuje vzdialenosť $d_{j.i}$ od víťazného neurónu, približujúc sa k nule, keď vzdialenosť $d_{j,i}$ ide do nekonečna.

Funkcia, ktorá spĺňa obe tieto kritéria je Gaussova funkcia:

$$h_{j,i}(x) = \exp(-\frac{d_{j,i}^2}{2\sigma^2}) \qquad (2)$$

kde σ je efektívna šírka topologického suseda, ktorá určuje rozsah, do ktorého sa každý neurón v topologickom susedstve zapojí do procesu učenia. Tento parameter klesá exponenciálne v každej epoche *n* v závislosti od nasledujúceho vzorca:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right), n = 0, 1, 2, \ldots \qquad (3)$$

kde σ_0 je inicializovaná hodnota efektívnej šírky a τ_1 je konštanta zvolená autorom (dizajnérom) siete.

- *Prispôsobenie synapsií*: V poslednom kroku sa koná úprava váhových vektorov pre neuróny v mriežke podľa nasledujúceho vzťahu:

$$\Delta w_j = \eta h_{j,i(x)}(x - w_j), \begin{cases} i: & víťazný\ neurón \\ j: & neurón\ v\ susedstve\ i \end{cases} \qquad (4)$$

Konečne, vzhľadom na váhové vektory $w_j(n)$ v epoche n, môžeme ľahko vypočítať vektor pre epochu $n + 1$ za pomoci vzťahu:

$$w_j(n+1) = w_j(n) + \eta(n) h_{j,i(x)}(n) \left(x(n) - w_j(n)\right) \qquad (5)$$

Ako môžeme vidieť v poslednom vzorci, premenná $\eta(n)$ (learning rate – rýchlosť učenia) je tiež časovo (od epoch) závislá. Mala by sa inicializovať na hodnotu η_0 a klesať exponenciálne s narastajúcim počítadlom času (epoch) n:

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), n = 0, 1, 2, \ldots \qquad (6)$$

kde τ_2 je ďalšia konštanta zvolená dizajnérom siete.

Procedúry opísané vyššie môžu byť rozdelené do dvoch fáz:

- Fáza usporiadania, je prvá fáza, počas ktorej sa koná topologické preusporiadanie váhových vektorov neurónov v konkurenčnej vrstve. V tejto fáze by mala premenná $\eta(n)$ začínať na hodnote blízkej 0,1 a postupne klesať na hodnotu 0,01.

Tieto hodnoty môžeme dosiahnuť, keď do vzťahu (6) dosadíme nasledujúce hodnoty (Haykin, 2008):

$$\eta(0) = 0{,}1, \tau_2 = 1000 \qquad (7)$$

Ďalej by funkcia na nájdenie topologických susedov $h_{j,i}$ mala na začiatku obsahovať skoro všetky neuróny v mriežke a postupne sa zmenšovať aby obsahovala iba zopár neurónov alebo dokonca len samotného víťaza. V prípade dvojrozmernej mriežky môžeme nastaviť počiatočnú hodnotu efektívnej šírky σ_0 na hodnotu „polomeru“ mriežky a tiež nastaviť hodnotu τ_1 vo vzťahu (3) rovnú (Haykin, 2008):

$$\tau_1 = \frac{1000}{log\sigma_0} \qquad (8)$$

- Fáza konvergencie, v ktorej váhy dostávajú svoje finálne hodnoty prostredníctvom ďalších úprav vstupných dát. V tejto fáze počet opakovaní (epoch) závisí od rozmerov vstupných vektorov a spravidla by mala byť 500 násobkom počtu súťažných neurónov. Parameter $\eta(n)$ by mal mať hodnotu blízku 0,01 a, konečne, funkcia na nájdenie topologických susedov $h_{j,i}(n)$ by mala obsahovať iba neuróny najbližšie k víťaznému neurónu, ale môže skončiť aj tak, že bude obsahovať iba samotný víťazný neurón (Haykin, 2008).

Keď zosumarizujeme vyššie uvedený popis, učiaci algoritmus SOM je nasledovný:

Algoritmus 1.3: Trénovací algoritmus SOM

1. Vypočítame počet neurónov v konkurenčnej vrstve M.
2. Inicializujeme M ťažísk $w_i(0), i = 1, \dots, M$.
3. Inicializujeme parameter $\eta(0)$, parameter $\sigma(0)$, počítadlo epoch $k = 0$ a počítadlo opakovaní $\kappa = 0$.
4. Pre každú epochu k urobíme nasledujúce kroky pre $n = 1, \dots, N$
   - Nastav vektor x^n ako vstup neurónovej siete.
   - Vyber víťazný neurón m.
   - Uprav váhové vektory všetkých neurónov v susedstve víťazného neurónu:

$$h_{m,i}(\kappa) = \exp\left(-\frac{||r_m - r_i||^2}{2\sigma^2(\kappa)}\right), i = 1, \dots, M \quad (9)$$

$$w_{i,j}(\kappa + 1) = w_{i,j}(\kappa) + \eta(\kappa) h_{m,i}(\kappa)\left(x_{n,j} - w_{i,j}(\kappa)\right), j = 1, \dots, d \quad (10)$$

   - $\kappa = \kappa + 1$
5. Postupne znižujeme hodnotu $\eta(k)$.

6. Postupne znižujeme hodnotu efektívnej šírky $\sigma(k)$.
7. Skontroluj podmienky na ukončenie. Ak neplatia nastav $k = k + 1$ a vráť sa na krok 4.

Trénovacia fáza je iteratívna a opakuje sa dovtedy, pokým sa nenájde dostatočne „dobrá“ mapa. Zvyčajne sa ako indikátor presnosti mapy používa kvantizačná chyba, definovaná ako suma Euklidovských vzdialeností všetkých vzoriek od víťazného neurónu. Tréning sa zastaví, keď ε_T je dostatočne malá hodnota (Engelbrecht, 2007).

$$\varepsilon_T = \sum_{p=1}^{P_T} ||x_p - w_{m,i}(t)||_2^2$$

P^T je počet vzoriek.

Kapitola 2

Prostredie

V našej práci pracujeme v robotickom simulátore iCub. To simulátor reálneho humanoidného robota, ktorý sa podobá človeku výzorom aj pohyblivosťou. Kód píšem v prostredí Microsoft Visual Studio 2010 v jazyku C++. Neurónové siete sme naprogramovali pomocou voľne dostupnej knižnice FANN. Na počítači som mal nainštalovaný operačný systém Windows 7 (64 bit).

2.1 ICub

ICub je meno humanoidného robota vyvinutého ako výsledok projektu konzorcia RobotCup (obrázok 2.1). Dôraz sa kladie na otvorenosť celého projektu ako po softvérovej, tak aj po hardvérovej stránke. Zdrojové kódy a náčrty jednotlivých dielov sú voľne dostupné a ďalší vedci ich môžu použiť. Je to jeden z najpoužívanejších humanoidných robotov na výskumné a akademické účely v Európe. Robot svojimi rozmermi a fyzickými schopnosťami verne kopíruje dva a pol ročné dieťa. Je vysoký 90 centimetrov a váži necelých 23 kilogramov. Na humanoidných robotov sú kladené požiadavky na čo najviac stupňov voľnosti a množstvo senzorov. ICub má 53 stupňov voľnosti rozdelených nasledovne: sedem pre pohyb ramena a ďalších deväť pre ruku, šesť na každej nohe a po tri pre pás, krk a pohyb očí. O každom stupni voľnosti vieme zistiť jeho presnú pozíciu. ICub disponuje aj množstvom senzorov ako tlakové a dotykové senzory (ruka robota môže byť rozšírená o špeciálnu kožu) a mnohé ďalšie. V každom oku má kameru, ktorá zachytáva farebný obraz. Na komunikáciu sa využíva rozhranie YARP – komunikuje sa prostredníctvom portov. Keďže však nie každý vedecký alebo akademický tím si môže dovoliť zaobstarať takéhoto robota, vyvinuli vedci (Tikhanoff a kol., 2008) vernú kópiu robota ako open-source simulátor. (Malík, 2011) (Zdechovan, 2012)

Obr. 2.1: Robot iCub. (robotcup.org)

2.2 Simulátor iCub

Simulácie majú vo výskume dôležitú úlohu. Virtuálne prostredie umožňuje rýchlejší vývoj algoritmov a bezpečnejšie ladenie nepredvídaných situácii. Nehrozí totiž poškodenie reálneho robota alebo jeho okolia. Preto aj vedci vyvinuli simulátor iCub-a (obrázok 2.2), ktorý sa intenzívne využíva v rôznych projektoch. Simulátor je zostrojený pomocou fyzikálnej knižnice ODE a grafickej knižnice OpenGL a je vyvíjaný ako open-source projekt (projekt s otvoreným zdrojovým kódom). Simulovaný iCub bol navrhnutý na základe špecifikácie jeho reálnej predlohy, so zachovaním podstatných parametrov (rozmery, rozloženie hmotnosti, stupne voľnosti, pohyblivosť atď.). Simulátor sa skladá s 3D modelu iCub-a a okolitého sveta, kde je robot umiestnený. Vizualizácia prebieha oddelene od fyzikálnej časti. Na komunikáciu so simulátorom sa využíva rozhranie YARP, také isté ako pri reálnom robotovi (Malík, 2011) (Zdechovan, 2012).

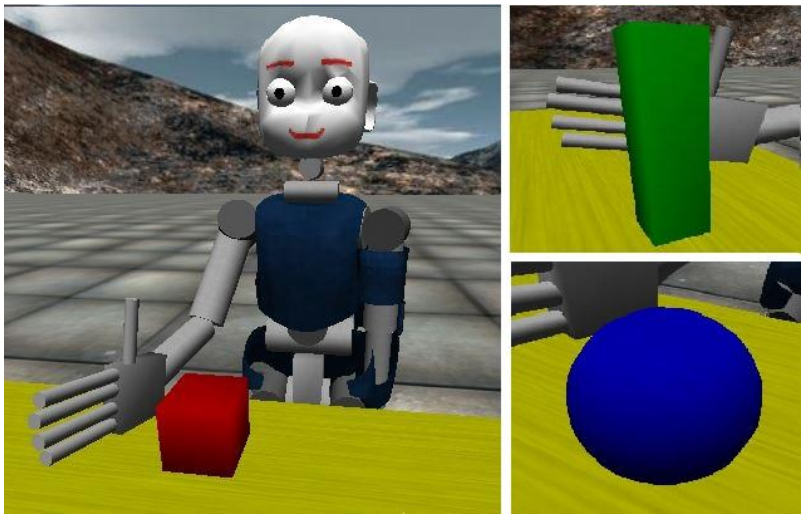
Obr. 2.2: Robotický simulátor iCub-a.

2.3 YARP

Na celú komunikáciu s robotom sa využíva rozhranie YARP. YARP je skratka odvodená od anglického názvu Yet Another Robot Platform. Cieľom projektu je vytvoriť univerzálne rozhranie (s otvoreným zdrojovým kódom) pre komunikáciu rôznych robotických zariadení. Vďaka tomu sa vedci a výskumníci môžu sústrediť na prácu. YARP je naprogramovaný ako multiplatformový nástroj v množstve programovacích jazykov (pôvodný je C++, ostatné napríklad: Java, Perl, Python, Matlab, C#, ...) (Yarp, 2011).

2.4 Scéna

V simulátore robota iCub budeme pracovať s rôznymi objektmi (obrázok 2.3). Pred robotom sa vytvorí pevná doska („stôl“) a na ňom bude vždy len jeden objekt. Ten bude v dosahu pravej ruky robota. Ako objekty používame tri zabudované objekty v simulátore – kocku, kváder a guľu (simulátor poskytuje aj možnosť importovania si vlastných modelov). Keďže rozoznávame objekty na základe farby, rozhodli sme odstrániť modrý kryt na pravej ruke iCub-a aby nám neskresľoval výsledky.



Obr. 2.3: Scéna v simulátore. V dosahu pravej ruky robota je umiestnení vždy jeden z objektov (kocka, kváder alebo guľa), ktorý má jednu z troch farieb (červená, zelená alebo modrá).

2.5 FANN

Na programovanie neurónových sietí sme použili voľne dostupnú knižnicu FANN – Fast Artificial Neural Network Library. Táto knižnica má otvorený zdrojový kód a implementuje viacvrstvové umelé neurónové siete (v jazyku C) s podporou plne prepojených alebo čiastočne prepojených sietí. Podporuje multiplatformový výpočet s fixnou aj pohyblivou desatinnou čiarkou. Obsahuje funkcie pre ľahké spracovanie tréningových sád. Je funkčná s viac ako pätnástimi jazykmi. Je jednoduchá na použitie, dobre a zrozumiteľne zdokumentovaná, univerzálna a rýchla. Obsahuje úvodný článok s inštrukciami aj referenčný manuál. Ponúka aj viaceré grafické rozhrania. (FANN, 2013)

Kapitola 3

Implementácia

3.1 Komunikácia so simulátorom

Robot iCub je postavený na otvorenej robotickej platforme YARP (Yet Another Robot Platform), ktorú vytvorilo konzorcium RobotCub. Ide o implementáciu strednej vrstvy pre komunikáciu jednotlivých častí humanoidných robotov. Samostatné časti robota (ruka, hlava, kamery, dotykové senzory...) vystupujú z architektonického pohľadu ako klienti, ktorý sa cez unikátne názvy portov registrujú v YARP serveri. Ten zabezpečuje ich vzájomnú komunikáciu. Jednotlivé časti robota si teda medzi sebou posielajú správy (napr. príkaz na určité natočenie pravej ruky), alebo čítajú prichádzajúce dáta (napr. z dotykových senzorov na pravej dlani). Aby sme vedeli robota ovládať z aplikácie, musíme sa cez YARP pripojiť na porty, na ktorých sú registrované požadované časti robota (Zdechovan, 2012).

Komunikácia s portami robota je možná viacerými spôsobmi. Jedna možnosť je použitie príkazového riadku na vzdialené volanie funkcii (remote procedure call - RPC). Natočenie ruky potom môže vyzerať nasledovne:

yarp rpc /icubSim/right_arm/rpc:i

set pos 0 45

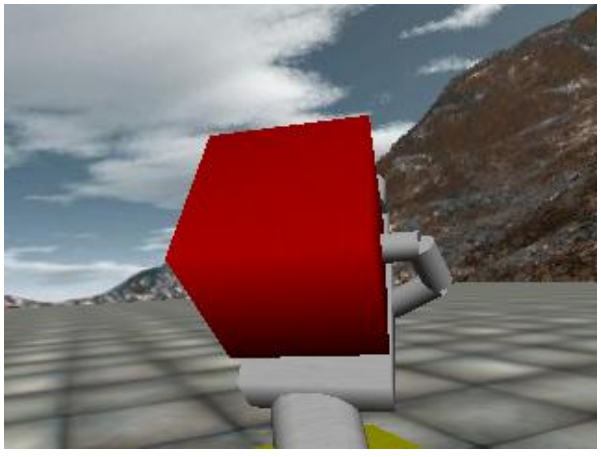
Prvý príkaz spustí program yarp a pripojí sa na port pravej ruky. Nasleduje názov procedúry a jej parametre (v tomto prípade nastavenie polohy prvého kĺbu ruky na hodnotu 45 stupňov). Toto je ale dosť nepraktický spôsob vzhľadom na potrebu meniť polohu viacerých stupňov voľnosti naraz. Preto je vhodné mať možnosť komunikovať so simulátorom s prostredia programovacieho jazyka. YARP poskytuje aj túto možnosť, ktorú sme využili a pracovali sme v programovacom jazyku C++. Peknú príručku na inštaláciu a ovládanie simulátoru vypracoval vo svojej diplomovej práci Tomáš Malík (Malík, 2011). Sú tam aj konkrétne algoritmy ako sa napojiť na porty robota a vykonávať rôzne akcie (napríklad zachytiť obraz z kamery robota) (Malík, 2011).

3.2 Spracovanie obrazu

Spracovaním obrazu sa zaoberajú samostatné, rozsiahle časti informatiky ako počítačové videnie alebo spracovanie obrazu. Tie počas svojej existencie vyvinuli, vylepšili a implementovali nepreberné množstvo algoritmov na spracovanie obrazu na počítači. Medzi ne patria napríklad rôzne algoritmy na zisťovanie hrán objektov, zostrenie a rozostrenie obrazu, vyhľadanie konkrétnej farby na obraze, konverzia do iných farebných spektier a mnohé iné. V našej práci sme podstatnú časť informácii získavali práve spracovaním obrazu, preto je táto oblasť pre nás dôležitá. My sme sa v našej práci rozhodli uprednostniť pred vlastnou implementáciu použitie už existujúcej knižnice OpenCV, v ktorej sú implementované všetky pre našu prácu potrebné funkcie a procedúry. Táto knižnica je open-source. Je teda možné skúmať a upravovať jej zdrojový kód, čo patrí určite medzi jej výhody. Funkcie si môžeme doprogramovať do požadovanej podoby alebo, keď sa vyskytne nejaká chyba vieme ju vystopovať, prípadne aj opraviť chybný kód. Ďalším faktorom prečo sme sa rozhodli použiť túto knižnicu je ten, že je implementovaná v C/C++, takže ju môžeme priamo použiť v našom pracovnom prostredí bez toho, aby sme museli niečo zložito kompilovať (Zdechovan, 2012).

3.2.1 Obraz z iCuba

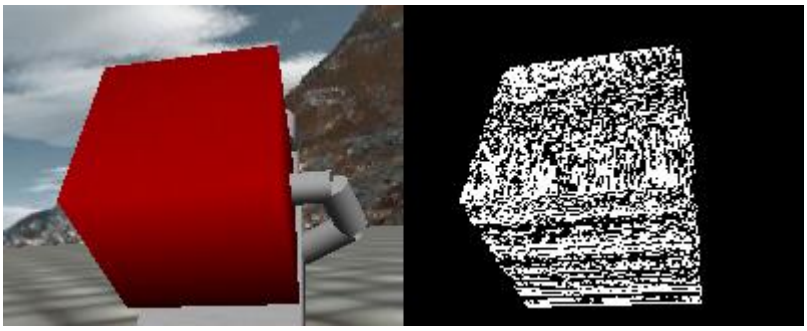
Na to aby sme nejaký obraz mohli upravovať potrebujeme ho najprv získať. V našej práci využívame simulátor ICub, ktorý má v pravom aj v ľavom oku po jednej kamere. My využívame obraz iba z pravej kamery. Na tieto kamery je možné pripojiť sa cez YARP prostredníctvom portov. Cez porty načítame trojkanálový obraz o veľkosti 320x240 pixlov (obrázok 3.1).



Obr. 3.1: Obraz z pravej kamery v simulátore ICub.

3.2.2 Thresholding

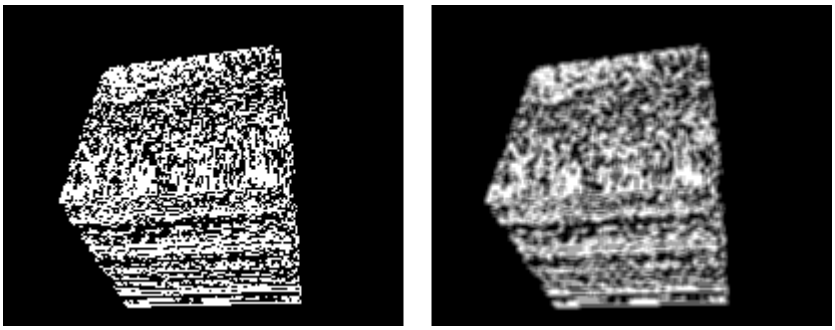
Na thresholding (prahovanie) je v knižnici OpenCV naprogramovaných niekoľko funkcií. Z verzie knižnice, ktorú sme použili v našej práci sme sa rozhodli pre funkciu *inRange*. Okrem zdrojového a cieľového obrázku sú parametrami funkcie aj dva vektory reprezentujúce farebný interval v HSV farebnom modely. Táto funkcia vytvorí zo zdrojového obrázku čierno-biely obraz tak, že bielou farbou zafarbí pixle, ktoré spadajú do zadaného farebného intervalu a naopak čiernou tie, ktoré tam nepatria (obrázok 3.2). Vznikne nám tak teda čierno-biely obraz, na ktorom sa zvýraznia objekty požadovanej farby. Keďže našu scénu v simulátore ICub sme si zvolili tak, že každý objekt ma jedinečnú farbu (červenú / zelenú / modrú) vieme tieto objekty jasne identifikovať.



Obr. 3.2: Vľavo je farebný obraz, vpravo je obraz po prahovaní na červenú farbu.

3.2.3 Rozostrenie

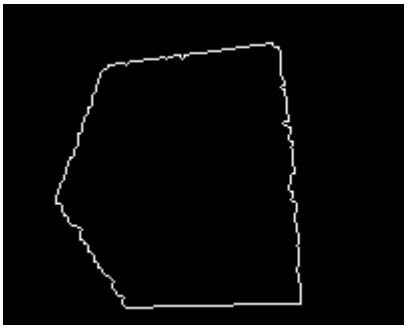
Rozostrenie alebo tiež blurring sa používa po prahovaní ešte pred ďalšou úpravou obrazu. Často sa totiž po prahovaní stane, že je biela plocha plná množstva menších, či väčších čiernych dier. Tie sa primeraným rozostrením zaplnia (obrázok 3.3). Robíme to preto aby sme na obraze jednoduchšie vyhľadali kontúry objektov, keďže nebudú pôsobiť ako veľa malých objektov na čiernej ploche, ale ako jeden veľký. Použili sme na to funkciu *GausianBlur*, ktorá, ako už vyplýva z názvu, používa gaussovský filter pre hodnoty jednotlivých pixlov.



Obr. 3.3: Vľavo je prahovaný obraz a vpravo obraz po rozostrení.

3.2.4 Kontúry a ohraničenia

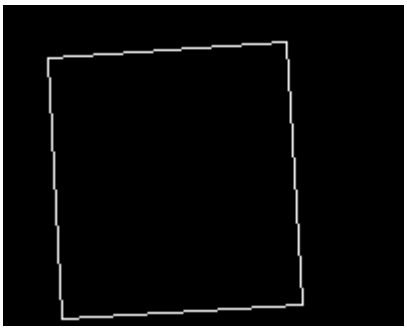
Po prahovaní a rozostrení vyhľadávame v obraze kontúry objektu, ktorý chceme identifikovať. Používame na to funkciu FindContours, ktorá nájde všetky kontúry na obrázku a to tak, že nájde pole postupnosti bodov – každá postupnosť tvorí jednu kontúru. Výsledné kontúry si zakreslíme do cieľového obrázka pomocou funkcie DrawContours, do ktorej musím poslať kontúry vyhľadané predošlou funkciou (obrázok 3.4). Preto je výhodné podrobiť obraz najskôr rozostrenie aby sme vo výsledku dostali iba jednu kontúru opisujúcu objekt (aj keď nie dokonale presne). Užitočná je aj funkcia ContourArea, ktorá vypočíta pre zadanú kontúru jej obsah. Využiť sa to dá napr. keď chceme vykresliť len objekt, ktorý zaberá najväčšiu plochu. Niekedy to môže uľahčiť prácu aj nám, najmä keď aj po rozostrení sa obrázok javí ako viac objektov (pritom čierne škvrny, ktoré to spôsobia zaberajú len zlomok veľkosti cieľového objektu).



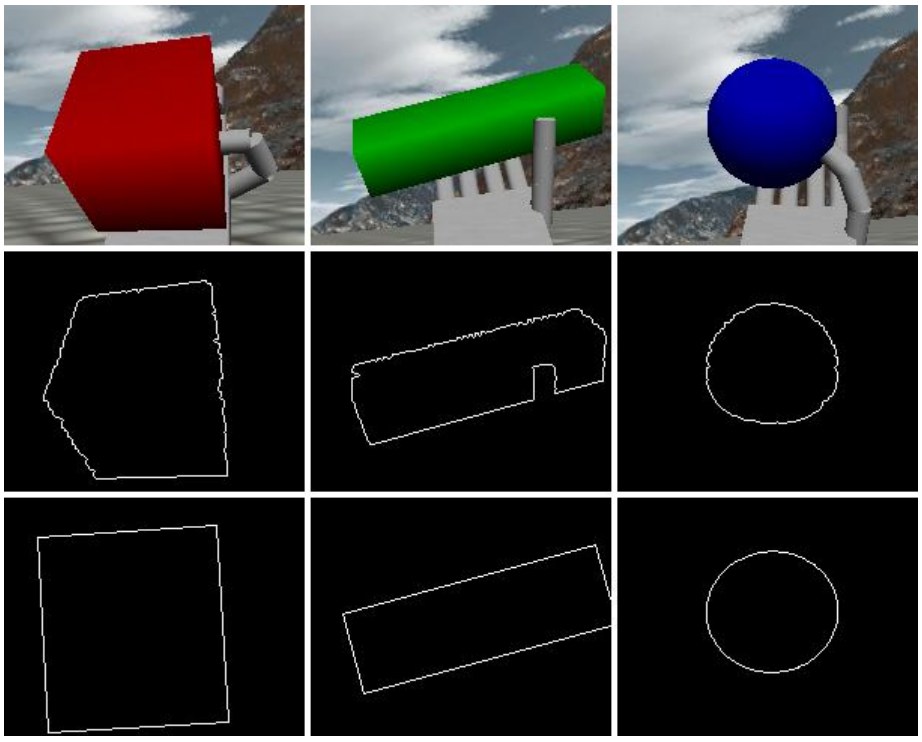
Obr. 3.4: Vykreslené kontúry objektu.

Ďalším krok je ohraničenie kontúry minimálnym obdĺžnikom resp. minimálnou elipsou. Využívame na to funkcie minAreaRect a fitEllipse, ktoré tiež pracujú s kontúrami nájdenými v predchádzajúcom kroku a každej opíšu buď obdĺžnik alebo elipsu. Tie si opäť môžeme zakresliť do výsledného obrázka (obrázok 3.5). Môžeme aj obe naraz, ale to by bolo dosť zbytočné, nakoľko napr. v rozpoznávaní gule by spôsobilo určitý zmätok kebyže

ju reprezentujeme minimálnym opísaným obdĺžnikom. Ohraničenie kontúry minimálnym obdĺžnikom alebo elipsou je ale pre nás značné zjednodušenie a zovšeobecnenie (obrázok 6), keďže v našej práci sa zameriavame výhradne na identifikáciu objektu z hľadiska tvaru a nepoužívame tak ďalšie údaje ako pomer strán, veľkosť či orientácia objektu. Preto sme sa rozhodli pracovať s pôvodnými kontúrami objektov a tieto dáta použiť iba na porovnanie úspešnosti natrénovaných neurónových sietí. Výsledky testovania opíšeme kapitole Výsledky.



Obr. 3.5: Minimálny obdĺžnik opísaný objektu z obrázka 3.4.



Obr. 3.6: V prvom riadku sú pôvodné obrázky, v druhom po hľadaní kontúr a na treťom po opísaní minimálnych útvarov.

3.3 Extrakcia príznakov

Aby sme si nemuseli pamätať kompletnú informáciu o spracovanom obrázku z predošlej časti iba vo forme každého pixlu (v našom prípade po orezaní mal obrázok veľkosť 200 x 160 pixlov), je vhodné sústrediť sa na charakteristické rysy – príznaky – ktoré dobre vystihujú daný obrázok. Vyextrahované príznaky najlepšie charakterizujú konkrétny obrázok, pritom však majú omnoho menej premenných a tak aj menšiu veľkosť. Tá je nesporne výhodou pri ďalšom spracovaní údajov, kde môžeme potom narábať aj s rádovo omnoho menším počtom premenných pre jeden obrázok. Výsledky bežných funkcii na extrakciu rôznych príznakov sú rádovo v stovkách nanajvýš v tisíckach (pri dôraze na najvyššie detaily), čo je stále menej ako keby sme si pamätali hodnotu každého pixla na obrázku (pre nás by to predstavovalo 200 x 160 = 32 000).

3.3.1 Extrakcia príznakov farby

Farbu objektu, ktorý drží iCub v ruke sme zisťovali tak, že sme si vyextrahovali farebné príznaky z obrázka, na ktorom iCub drží konkrétny predmet v ruke. Môžeme si to dovoliť nakoľko vieme, že na scéne sa nevyskytuje ďalší objekt rovnakej farby (vytvorené objekty sme mali v troch farbách – červená, zelená a modrá) a po orezaní pozadia, nám ani to neskresľuje výsledné príznaky.

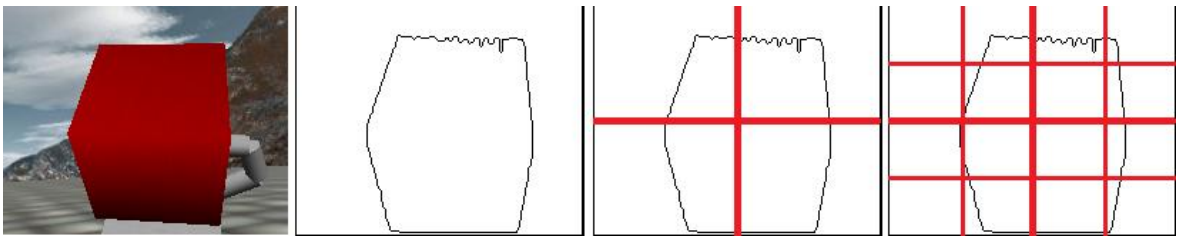
Na extrakciu príznakov farby sme použili funkciu *extractColor*, ktorá je dostupná aj v priečinku s iCub-om, presnejšie v priečinku *poeticon* v súbore *FeatureExtractor.cpp*. Je implementovaná v C++, takže sa presne hodí do nášho projektu. Funkcia extrahuje príznaky z obrázka vo farebnom spektre *L*a*b** – *L* je svetlosť (lightness), *a** je hodnota medzi červenou a zelenou (záporná hodnota pre zelenú, kladná pre červenú), *b** je hodnota medzi modrou a žltou (záporná pre modrú, kladná pre žltú). Na vstupe dostane zdrojový obrázok a jej výstupom je vlastne histogram farieb – stĺpcový graf, kde na osi x sú intervaly, v našom prípade farebné, a na osi y je ich početnosť – na zdrojovom obrázku. Ten je prevedený do podoby jednorozmerného vektora a charakterizuje zadaný obrázok. Funkcia si najprv prevedie vstupný obrázok do formátu Lab a rozloží ho na farebné roviny. Pôvodný obrázok použije ako masku na odstránenie čiernych pixlov a vypočíta výsledný histogram. Základné nastavenie pre rovinu *a* aj rovinu *b* je 15 intervalov, čo znamená, že výsledný histogram aj výsledný vektor majú veľkosť 225.

3.3.2 Extrakcia príznakov tvaru

Na extrakciu príznakov o tvare objektu sme použili funkciu *extractPHOG*, ktorej jedna z implementácii je taktiež v priečinku *poeticon* v súbore *FeatureExtractor.cpp* v domovskom priečinku iCub-a. *PHOG* je skratka pre Pyramid of Histograms of Orientation Gradients. Deskriptor PHOG predstavili vo svojej práci *Representing shape with a spatial pyramid kernel* A. Bosh, A. Zisserman a X. Munoz v roku 2007. Ich idea spočívala v tom, že spojili výhody dovtedy známych techník: tvar a priestorové rozloženie tvaru sa získavalo pomocou detekcie hrán, ale výsledná reprezentácia bola vo forme vektora. Funkcia vytvorí histogramy orientovaných gradientov pre každú časť obrázka v každom levely – odtiaľ názov Pyramid of Histograms of Orientation Gradients. Výsledkom je potom konkatenácia týchto histogramov od najvyššieho levelu. Každý level znamená rozdelenie predošlého regiónu obrázka na dve v každom smere – región obsahuje štyri menšie podoblasti (obrázok 3.7). Orientované gradienty hrán sa počítajú v každej podoblasti tak, že sa rozdelí na určitý počet úsekov (každý takýto úsek zaberá rozpätie v intervale 0 až 360 stupňov), spočítajú sa histogramy a hodnota celého regiónu je vlastne suma štyroch podoblastí, ktoré ho tvoria. Level 0 je reprezentovaný K-rozmerným vektorom zodpovedajúcim K úsekom, na ktoré je rozdelený (na osi x v histograme), level 1 je reprezentovaný $4K$-rozmerným vektorom atď. Veľkosť PHOG deskriptora celého obrázka je daná vzťahom:

$$K \sum_{l \in L} 4^l$$

kde L je počet levelov a K počet úsekov, na ktoré sa delia jednotlivé podoblasti. Napríklad pre $L = 1$ a $K = 20$ to bude vektor zo 100 stĺpcami. Viac informácii a porovnanie s ostatnými metódami extrakcie príznakov o tvare sa môžete dozvedieť v spomínanej práci (Bosh, 2007).



Obr. 3.7: Porovnanie rôznych levelov použitých v descriptore PHOG. Prvý je pôvodný obrázok, na ostatných sú nájdené hrany (invertované farby). Prvý je level 0, druhý je level 1 a tretí je level 2.

My sme v našej práci použili hodnoty $L = 1$ a $K = 20$ (každý úsek zodpovedá 360 / 20 = 18 stupňov). Konkrétne implementácia v priečinku iCub-a, ktorú používame, dostane vstupný obrázok, prevedie ho do šedej škály a následne vyhľadá hrany. Ďalej vypočíta orientované gradienty v smere osi x a v smere osi y. Na záver vypočíta histogramy pre všetky levely „pyramídy“. V našom prípade je teda výsledkom vektor s jedným riadkom a 100 stĺpcami.

3.3.3 Vyhodnotenie

Namiesto aby sme si pamätali údaje o celom obrázku, využili sme funkcie na extrakciu príznakov. Pre údaje o farbe na obrázku je to vlastne histogram farebného zastúpenia farieb a pre údaje o tvare sme využili deskriptor PHOG, ktorý vypočíta orientované gradienty hrán pre každý level a hodnoty spojí do výsledného vektora. Vektor farby má 1 riadok a 225 stĺpcov a vektor opisujúci tvar má 1 riadok a 100 stĺpcov. Obe funkcie sú implementované v súbore *FeatureExtractor.cpp*, ktorý je v priečinku s iCub-om. Pre lepšiu názornosť sme sa rozhodli výsledné vektory ešte pred konečným trénovaním a testovaním sietí vizualizovať pomocou štruktúry SOM – samo-organizujúca sa mapa. To nám pomôže predstaviť si ako vyzerá rozloženie vektorov, a či sa dajú jednoznačne rozdeliť na tri triedy (červená, zelená, modrá resp. kocka, kváder, guľa).

3.4 Vizualizácia dát

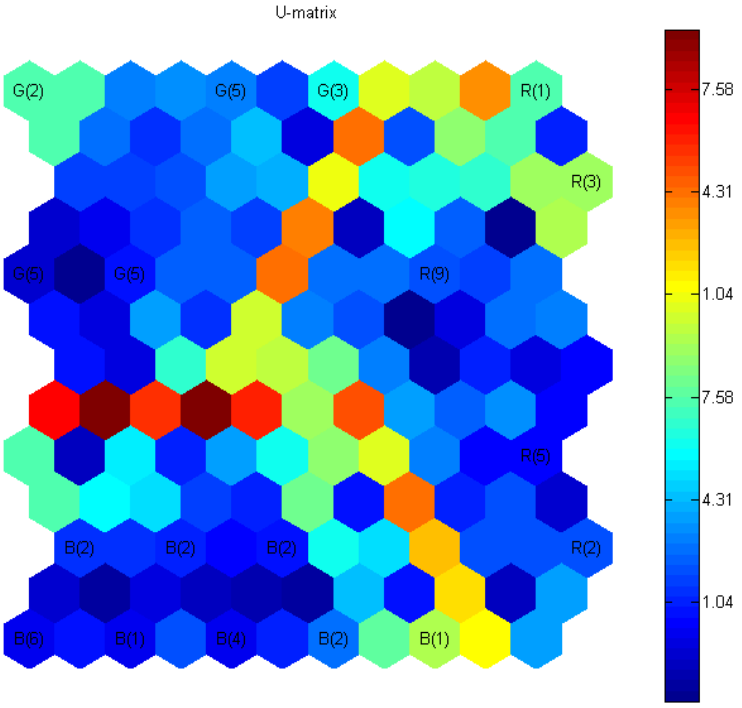
Dáta získané v predošlej časti našej práce sú v podobe číselných vektorov. Na to aby bolo trénovanie neurónovej siete úspešné potrebujeme aby sa dáta dali ľahko a jednoznačne rozlíšiť. Ešte pred trénovaním samotných sietí sme sa preto rozhodli vizualizovať naše získané dáta pomocou štruktúry SOM – Samo-organizujúca sa mapa. Je to tiež jeden z druhov neurónových sietí, ktorý slúži najmä na vizualizáciu viacrozmerných dát do prijateľnej (zvyčajne 2D) podoby. Obvyklým výsledkom je dvojrozmerná štvorcová (môže byť aj obdĺžniková) mriežka, na ktorej sú rovnaké alebo podobné vstupy znázornené väčšinou rovnakou alebo podobnou farbou a z hľadiska rozmiestnenia sú blízko seba. Proces tvorby mapy prebieha vo viacerých krokoch. Najprv sa určí „víťaz“ , neurón, ktorý sa najviac podobá zadanému vstupu. Následne sa upravia váhové vektory všetkých jeho topografických susedov. Vektory, ktoré mali podobné hodnoty, budú aj na mape vykreslené podobnou farbou a blízko pri sebe (Kohonen, 2001) (Farkaš, 1997).

3.4.1 Matlab

Na vizualizáciu sme použili program Matlab s doinštalovaným modulom na SOM – somtoolbox. Najprv treba načítať požadované dáta. Funkcia *som_read_data* načíta dáta zo súboru v podporovanom formáte – prvý riadok je počet položiek jedného vektora, ďalej nasledujú riadky vstupných vektorov. Za každým vektorom môže byť „meno“ , podľa ktorých potom môžeme na výslednú mapu umiestniť menovky. Keď máme načítané hodnoty (vhodné je ich aj normalizovať pomocou funkcie *som_normalize*) dá sa z nich vytvoriť SOM mapa. My sme na to využili funkciu *som_make*, ktorá zaobaluje viacero funkcií do jednej. Dostane ako parameter vstupné dáta a vráti hotovú mapu. Najprv treba zistiť počet neurónov, ak nie je určené inak, funkcia použije heuristický vzorec $munits = 5 * dlen^{0,54321}$ na zistenie ich počtu. Môžeme ich aj slovne zadať (napríklad „big“ znamená počet pôvodne zistených neurónov krát štyri). Keď už máme počet neurónov, ďalším krok je určiť veľkosť mapy. Jednoducho povedané, vypočítajú sa dve najväčšie hodnoty zo vstupných dát a pomer strán mapy je nastavený na hodnotu pomeru týchto hodnôt. Výsledný pomer je potom upravený aby čo najviac vyhovoval počtu neurónov. Mapa sa inicializuje. ako prvá sa skúša lineárna inicializácia pomocou dvoch najväčších vektorov, ak je táto neúspešná (vektory nemôže byť vypočítané), použije sa náhodná inicializácia. Po inicializácii SOM je mapa trénovaná v dvoch fázach: najprv hrubý tréning a potom doladenie. Ak je hodnota argumentu „sledovanie“ väčšia ako nula, na konci sa vypočíta priemerná kvantizačná chyba a priemerná topografická chyba finálnej mapy. Všetky hodnoty si môže samozrejme užívateľ meniť podľa seba, podrobný popis nájde v dokumentácii modulu (SOM Toolbox, 2003).

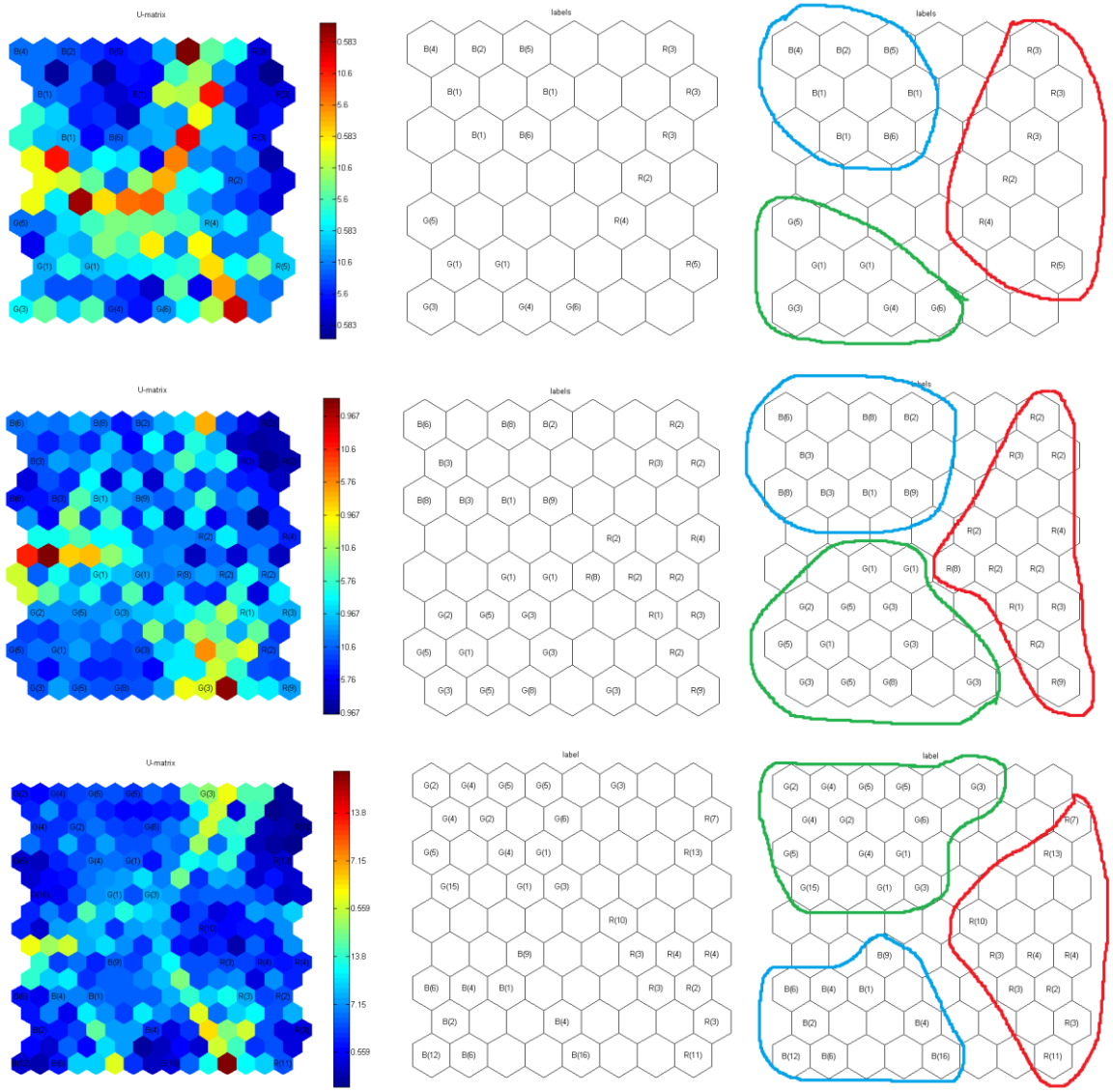
3.4.2 Výsledky pre farbu

Najprv sme sa zamerali na vizualizáciu vektorov farby. Na obrázky z kamery ICub-a sme zavolali funkciu *extractColor* (bližšie popísaná v predošlej časti), ktorej výsledkom je vektor s 225 položkami. Očakávania boli, že s klasterizáciou farby by nemal byť žiadny problém. To sa aj potvrdilo ako je vidieť na obrázku 3.8. Môžeme tam vidieť tri klastery (zhluky), pre každú farbu jeden (zafarbené podobným odtieňom modrej farby), medzi sebou oddelené odtieňmi žltej a červenej farby. Ako metódu umiestnenia menoviek vo funkcii *som_autolabel* sme zvolili metódu „*freq*“, ktorá pre každý neurón vypíše, pre aký vstup sa tento neurón stal víťazom (menovky spomínané v úvode tejto časti) a do zátvorky uvedie ich počet.



Obr. 3.8: Samo-oraganizujúca sa mapa pre vektory farby. R – červená, G – zelená, B – modrá. Na stupnici sú znázornené vzdialenosti medzi váhovými vektormi.

Testovali sme rôzne počty vstupných vektorov – 20, 40 a 60 pre jednu farbu. Vždy sme použili funkciu *som_make* a jej parametre sme nijako neupravovali. Celkový prehľad je uvedený na obrázku 3.9 (vo väčšom rozlíšení v prílohe) a v tabuľke 3.1 sú uvedené kvantizačná a topografická chyba. Kvantizačná chyba je rozdiel vstupnej vzorky a neurónu, ktorý bol označený ako víťazný, topografická chyba predstavuje podiel všetkých vektorov, pre ktoré nebol prvý a druhý víťazný neurón vedľa seba. Na prvom obrázku pre dvadsať vzoriek je pekne vidno tri klastery oddelené oranžovou až červenou farbou. Pri štyridsiatich vzorkách sa zelený klaster priblížil ku červenému, ale je stále oddelený. Modrý je pekne oddelený. Pri šesťdesiatich vzorkách už ku žiadnemu splynutiu neprichádza a klastery sú pekne rozdelené svetlou farbou. Potvrdilo sa nám teda, že klasterizácia farby nie je problém, preto predpokladáme aj úspešné trénovanie a následné testovanie v neurónovej sieti.



Obr. 3.9: Porovnanie počtu vzoriek vektorov farby na vplyv SOM. Prvý riadok je 20 vzoriek, druhý 40 a tretí 60. V prvom stĺpci je pôvodná mapa SOM, v druhom sú iba menovky na prázdnej mriežke pre lepšiu orientáciu a v treťom sú zvýraznené tri farebné zhluky (po jednom pre červené, zelenú a modrú farbu).

Chyba/počet vzoriek	20	40	60
kvantizačná	2,424	2,969	2,115
topografická	0,050	0,017	0,006

Tab. 3.1: Popis kvantizačnej a topografickej chyby pre rôzne počty vzoriek.

3.4.3 Výsledky pre tvar

Vektory získané funkciou *extractPHOG* (bližšie v časti 3.3) ako reprezentácia tvaru, sme sa tiež rozhodli najprv vizualizovať. Rovnako, ako pri vizualizovaní vektorov popisujúcich farbu, sme použili funkciu *som_make* bezo zmeny parametrov. Ako sme spomenuli v časti o spracovaní obrázkov z ICub-a, rozhodli sme sa porovnať dva spôsoby

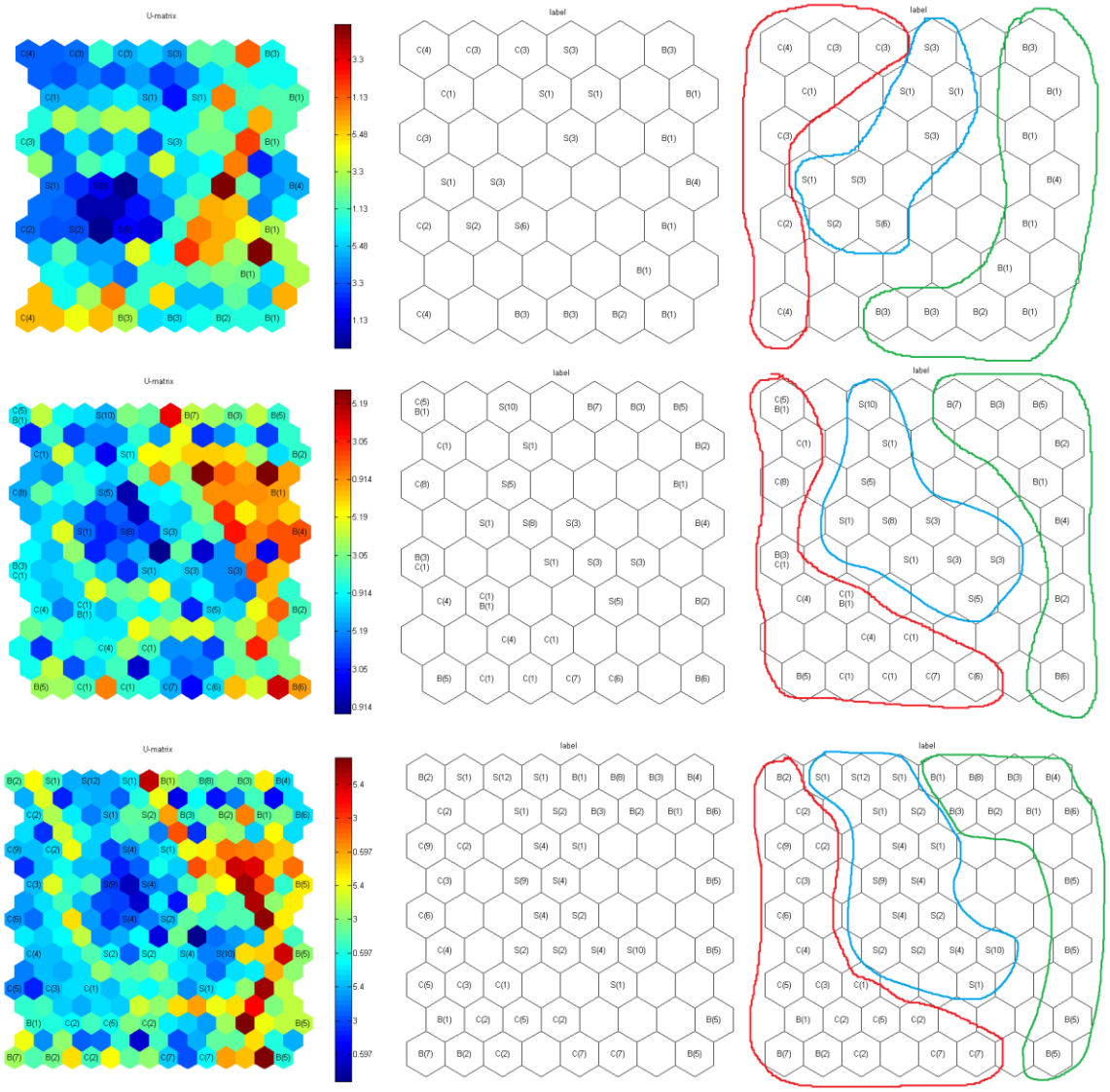
reprezentácie objektov. Jeden je pomocou kontúry objektu, ten druhý, všeobecnejší, spočíva v tom, že objektu opíšeme minimálny útvar. V prípade kocky a kvádra to je štvorec resp. obdĺžnik, v prípade gule je to elipsa. Naším prvým predpokladom je, že vzhľadom na jednoznačnosť objektov by s klasterizáciou nemal byť väčší problém. Je tu ale aj možnosť, že pri niektorých špecifických pozíciách ramena robota sa budú objekty javiť rovnako. Toto sa týka najmä kocky a kvádra (napríklad sa budeme pozerať na vrchnú stenu kvádra a nebudeme vidieť jeho výšku).

3.4.4 Výsledky pre kontúry

Ako prvé sme teda vizualizovali vektory, ktoré reprezentovali obrázky s kontúrami našich objektov. Ako je vidieť na obrázku 3.10 (vo väčšom rozlíšení v prílohe), pri dvadsiatich vzorkách sa jasne vyprofilovala guľa (tmavomodrou farbou a blízko seba), od kocky je oddelená svetlou farbou a od kvádru tmavo-oranžovou. Pri štyridsiatich vzorkách sa guľa ešte viac vzdialila od kocky a kvádra. Tie sa sústreďujú vo vlastných oblastiach oddelených žltou až tmavo oranžovou farbou. Ojedinele je tu vidieť aj vektory kvádra namapované do oblasti kde prevláda kocka. Pri počte vzoriek šesťdesiat sa guľa ešte viac osamostatnila v strede mriežky. Kocka je oddelená svetlou farbou na ľavej strane, zatiaľ čo kváder je na pravej strane za tmavo oranžovou farbou. Aj tu sa vyskytli niektoré vzorky kvádra, ktoré sa javili ako kocka. V oblasti, kde prevláda kváder je vidieť aj veľa neurónov zafarbených tmavo oranžovou až červenou farbou, čo indikuje veľké vzdialenosti aj v rámci tohto klastera (celkové chyby sú zachytené v tabuľke 3.2). Inak sme ale nijaké problémy, či prekvapenia nezaregistrovali.

Chyba/počet vzoriek	20	40	60
kvantizačná	4,446	4,499	4,441
topografická	0,017	0,033	0,006

Tab. 3.2: Popis kvantizačnej a topografickej chyby pre rôzne počty vzoriek.

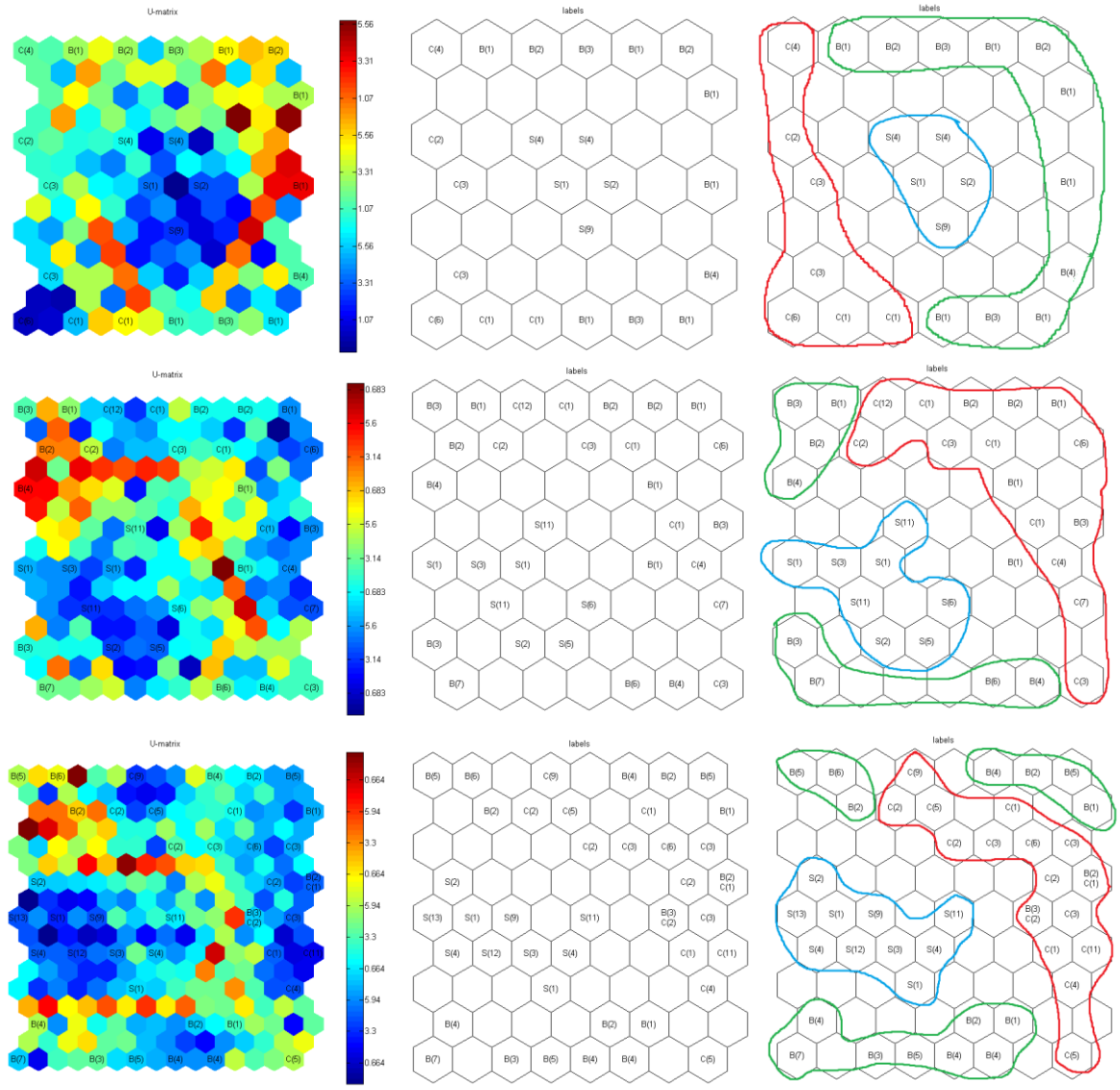


Obr. 3.10: Porovnanie počtu vzoriek vektorov kontúr na vplyv SOM. Prvý riadok je 20 vzoriek, druhý 40 a tretí 60. V prvom stĺpci je pôvodná mapa SOM, v druhom sú menovky a v treťom sú zvýraznené tri farebné klastery (červená – kocka, zelená – kváder, modrá - guľa).

3.4.5 Výsledky pre opísané útvary

Na záver vizualizácie nám ostali vektory reprezentujúce objekty pomocou opísaných útvarov – obdĺžnika (resp. štvorca) a elipsy. Rovnako ako v predchádzajúcich prípadoch, aj tu sme porovnali rozdielne počty vzoriek pre jednotlivé tvary a to 20, 40 a 60. Výsledky sú znázornené na obrázku 3.11 (vo väčšom rozlíšení v prílohe), príslušné chyby potom v tabuľke 3.3. Ako je vidieť pre dvadsať vzoriek sa nevyskytli žiadne problémy, guľa je namapovaná do stredu mriežky a pekne oddelená od okolia žltou a oranžovou farbou. Po stranách sú oblasti kde prevládajú kocka a kváder. Pri štyridsiatich a šesťdesiatich vzorkách už ale pozorujeme, že vektory nie sú mapované do troch

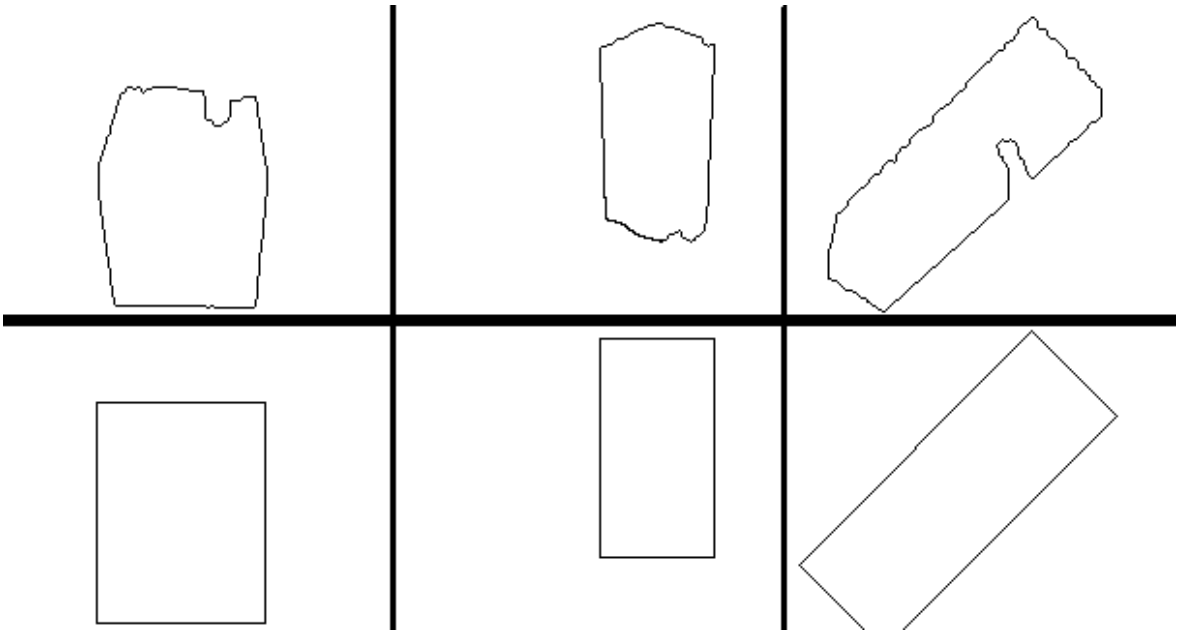
prevládajúcich klasterov. Guľa je stále pekne oddelená od ostatných žltou až červenou farbou a jej neuróny sú zafarbené podobným odtieňom modrej, čo indikuje malé vzdialenosti medzi nimi. To už ale neplatí pre ostatné dva útvary. Je to spôsobené nastavením mriežky v tavare toroidu (vrchná a spodná časť ako aj pravá a ľavá časť sa berú ako susedné). Pre lepšiu názornosť sme na obrázku 3.12 porovnali jednoznačnosť útvarov.



Obr. 3.11: Porovnanie počtu vzoriek vektorov opísaných útvarov a príslušné siete SOM. Prvý riadok je 20 vzoriek, druhý 40 a tretí 60. V prvom stĺpci je pôvodná mapa SOM, v druhom sú menovky a v treťom sú zvýraznené tri farebné klastery (červená – kocka, zelená – kváder, modrá - guľa).

Chyba/počet vzoriek	20	40	60
kvantizačná	3,146	3,480	3,026
topografická	0,017	0,067	0,033

Tab. 3.3: Popis kvantizačnej a topografickej chyby pre rôzne počty vzoriek.



Obr. 3.12: Porovnanie jednoznačnosti objektov (kocka, kváder, kváder), (invertované farby). V prvom riadku sú objekty reprezentované pomocou kontúr, v druhom im je opísaný útvar. Vidíme, že s identifikáciou kocky na prvom a kvádra na treťom obrázku nemáme väčšie ťažkosti. Na druhom obrázku je kváder. Podľa kontúry by sme ho zaradili určite ako kváder, ale podľa druhého obrázka to až také jednoznačné nie je. To spôsobuje, že aj na mape SOM sa nám niektoré reprezentácie kvádrov namapovali aj opticky bližšie ku kocke .

3.4.6 Vyhodnotenie

V tejto časti sme sa venovali vizualizácii vektorov, ktoré sme získali v predošlej fáze našej práce ako reprezentáciu objektu pomocou jeho farby a tvaru. Robili sme to pre lepšiu predstavu, čo môžeme očakávať od ďalšej – hlavnej – časti našej práce, a to implementovanie neurónových sietí na rozpoznávanie objektov. Predpoklad pre dobré fungovanie siete je, že dáta sú dostatočne dobre rozlíšiteľné aby mohla fungovať jednoznačná klasifikácia – rozdelenie dát na triedy (v našom prípade tri triedy pre farbu a tri triedy pre tvar). V prípade farby sa nám potvrdilo, že s klasifikáciou nebude žiaden problém a dáta sú pekne rozdelené na tri časti. Čo sa týka tvaru, aj tu sa nám potvrdili prvotné predpoklady. Teda, že pri určitej polohe objektu sa napríklad kváder môže viac javiť ako kocka než kváder. Ale potvrdilo sa aj, že v ďalšej fáze by sme nemali mať výraznejšie problémy, nakoľko neurónovú sieť nezaujíma topografická poloha vektorov v mriežke, ale ich hodnota. To znamená, že aj keď máme na obrázku 4 (pre šesťdesiat vzoriek) až tri oblasti pre kváder, naša sieť sa ich všetky naučí klasifikovať ako kváder. A to vďaka tomu, že od ostatných sú v dostatočnej vzdialenosti na to, aby sa javili ako

jednoznačné. Ako hlavnú však budeme používať sieť natrénovanú a testovanú pomocou kontúr.

3.5 Moduly pre rozpoznávanie farby a tvaru objektu

Na prácu s neurónovými sieťami používame voľne dostupnú knižnicu FANN. Tá nám poskytuje možnosť vytvoriť neurónovú sieť so zadaným počtom vstupných, skrytých a výstupných neurónov. Umožňuje nám načítať naše vzorky zo súboru v správnom formáte (prvý riadok obsahuje počet vzoriek, veľkosť jednej vzorky a počet výstupných neurónov, ďalšie riadky sú vstup a pod ním požadovaný výstup). Môžeme si zvoliť aktivačné funkcie aj ostatné parametre. Môžeme zadať kedy má skončiť trénovanie (buď po istom počte trénovacích epoch alebo keď chyba siete bude dostatočne malá). Všetky tieto hodnoty si užívateľ môže ľubovoľne meniť a sledovať rôzne úspešnosti natrénovanie siete (FANN, 2013).

Prvý modul je klasická dopredná viacvrstvová neurónová sieť, s jednou vstupnou vrstvou, jednou skrytou vrstvou a jednou výstupnou vrstvou. Je trénovaná a testovaná na vstupných vektoroch reprezentujúcich farbu objektu získaných z obrázka pomocou úprav a následnej extrakcie príznakov (bližšie popísané v časti 3.3). V našej práci používame tri farby – červenú, zelenú a modrú. Veľkosť vektorov, ktoré ich reprezentujú je 225 a výstupy sme zakódovali pomocou aktivácie troch neurónov na hodnoty: 1 0 0 – červená, 0 1 0 – zelená, 0 0 1 – modrá. V strednej, skrytej vrstve sme použili 50 skrytých neurónov (počet neurónov v sieti: 225 – 50 – 3) a na vstupe aj na výstupe sme použili funkciu sigmoid.

Druhý modul je tiež normálna dopredná viacvrstvová neurónová sieť. Taktiež nám stačí jedna skrytá vrstva neurónov. V našej práci používame tri základné typy objektov, ktoré chceme aby sa iCub naučil rozpoznávať – kocka, kváder a guľa. Preto aj tu budeme mať tri výstupné neuróny s nasledujúcimi aktiváciami: 1 0 0 – kocka, 0 1 0 – kváder, 0 0 1 – guľa. Vektory, ktoré popisujú ich príznaky zo vstupných obrázkov majú v našom prípade veľkosť 100 (pozri časť 3.3). Na skrytej vrstve sme použili 25 neurónov. Sieť má teda počty neurónov: 100 – 25 – 3. Na vstupe a výstupe sme takisto použili funkciu sigmoid. Výsledky úspešnosti oboch modulov popíšeme v kapitole Výsledky.

3.6 Modul pre aktívnu manipuláciu objektov

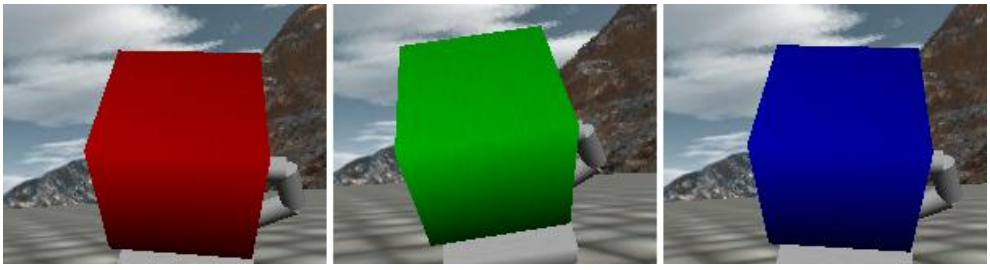
Záverečnou časťou našej práce bol návrh modulu, ktorý by umožnil simulátoru iCub-a rozhodnúť sa ako manipulovať s objektom, ktorý drží v ruke. Dostali by sme sa teda k tomu, že rozpoznávanie objektov by nebolo už iba pasívne, ale iCub by sa doňho aj aktívne zapájal. Predošlé dva moduly boli natrénované na rozpoznávanie objektov z trénovacích vzoriek, tie však iCub získaval buď náhodným, alebo vopred predpísaným postupným natáčaním dlane a ramena. Tento modul sme navrhli tak aby povedal robotovi ďalšiu pozíciu, do ktorej sa má dostať jeho rameno. Pracuje iba z reprezentáciou tvaru, keďže farba objektu sa natočením nemení.

Tento modul je tiež klasická dopredná neurónová sieť s jednou skrytou vrstvou neurónov. Ako vstup jej ide vektorová reprezentácia objektu (časť 3.3), za ktorou nasledujú hodnoty stupňov voľnosti 4, 5 a 6 (tie sú v našom prípade najviac zodpovedné za natočenie objektu – polohu prstov zmeniť nemôžeme lebo by predmet vypadol a polohu celého ramena nemeníme z dôvodu aby sa nám objekt nestratil z dohľadu kamery). Tieto hodnoty sú preškálované na interval (-1, 1). Ich počet sme vynásobili piatimi – to znamená, že každým takýmto neurónov nasledujú štyri s rovnakou hodnotou. Urobili sme tak preto lebo vektor opisujúci tvar je veľkosti 100 a hodnoty troch neurónov by tak zanikli. Vstupný vektor tak má nasledovnú veľkosť: 100 (tvar) + 5 (st. voľ. 4) + 5 (st. voľ. 5) + 5 (st. voľ. 6). Ako cieľ má tri hodnoty. Sú to hodnoty práve týchto troch stupňov voľnosti. Ich hodnoty sme však nezadali náhodne. Získali sme ich vo fáze testovania predošlého modulu a to tak, že sme si zaznačili novú hodnotu natočenia kĺbov iba vtedy ak aj odhad tvaru bol lepší. To znamená, že sme si pamätali starú polohu kĺbov a starú hodnotu pravdepodobnosti odhadu tvaru a keď sa zmenila poloha natočenia, skontrolovali sme, či je nová hodnota odhadu tvaru lepšia ako stará. Ak tomu tak bolo, do súboru sme si zaznačili starú hodnotu tvaru, za ňou staré hodnoty natočenia kĺbov (preškálované a každú päť krát) a ako cieľ sme si poznačili nové hodnoty natočenia (preškálované). Takto vytvorené testovacie vzorky nám po natrénovaní poslúžia na to, že robot si bude natáčať objekt tak, aby lepšie rozpoznal aký objekt drží. Na rozpoznávanie tvaru naďalej používame natrénovanú sieť z predošlej fázy. Tento modul nám slúži na určenie natočenia kĺbov robota. Výsledky uvedieme v kapitole Výsledky.

Kapitola 4

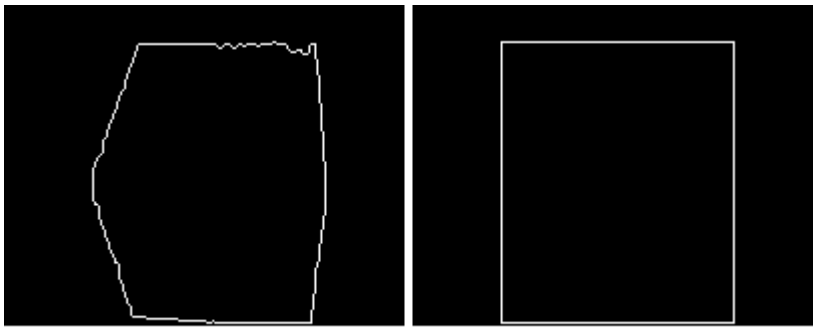
Výsledky

Cieľom našej práce bolo navrhnúť a implementovať modul neurónovej siete, ktorý by robotovi umožnil aktívne rozpoznávanie uchopeného objektu. Prácu sme si rozdelili na niekoľko menších častí. Najprv sme za pomoci voľne dostupnej knižnice FANN implementovali modul na rozpoznávanie farby. Objekt na scéne má jednu z troch farieb – červenú, zelenú alebo modrú (obrázok 4.1). Ako vzorky pre doprednú neurónovú sieť s jednou skrytou vrstvou poslúžili vektory opisujúce farbu na obrázku (získané v časti 3.3). Ako výstupné boli hodnoty: 1 0 0 – červená, 0 1 0 – zelená a 0 0 1 – modrá. Sieť bola trénovaná na šesťdesiatich vzorkách pre každú farbu. Jej úspešnosť pri rozpoznávaní tréningových vzoriek bola stopercentná, rovnako tak aj úspešnosť rozpoznávania neznámych vzoriek (obrázok 4.1). Je to spôsobené tým, že používame jasné farby a každý objekt má v jednom momente len jednu farbu a tiež tým, že natočením sa farba objektu nemení a do polohy, kde by bol objekt celý zakrytý sa nedostávame.



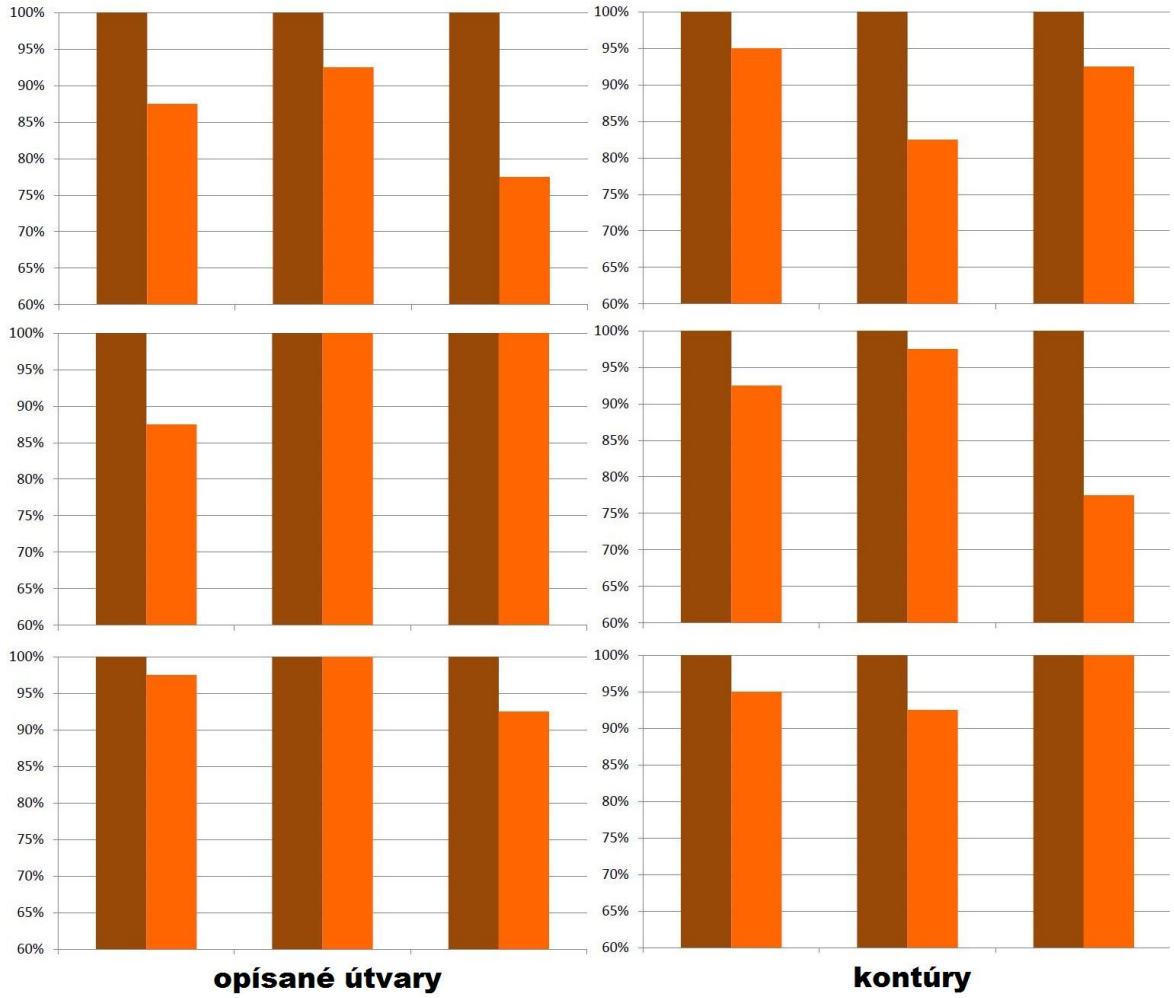
Obr. 4.1: Farby objektov použité v našej práci.

Ako ďalší sme implementovali modul na rozpoznávanie tvaru uchopeného objektu. Použili sme pri tom takisto knižnicu FANN na prácu s neurónovými sieťami. Ide o klasickú doprednú sieť s jednou skrytou vrstvou. V našej práci používame tri druhy objektov – kocku, kváder a guľu. Ako vstup pre túto sieť poslúžili vektory opisujúce tvar objektu (časť 3.3). Výstupné aktivácie neurónov boli: 1 0 0 – kocka, 0 1 0 kváder, 0 0 1 – guľa. Takúto sieť sme vytvorili dvakrát. Jedna bola trénovaná na rozpoznávanie objektu pomocou obrysov a druhá pomocou opísaných útvarov (obrázok 4.2).



Obr. 4.2: Obrys tvaru (a) a jemu opísaný útvar (b).

Sieť sme trénovali na šesťdesiatich vzorkách z každého tvaru. Pri rozpoznávaní tréningových vzoriek boli siete neomylné. Každý objekt v každej farbe bol reprezentovaný dvadsiatimi vzorkami a pokus sme robili dvakrát, pre každú sieť jeden (40 pokusov pre červenú kocku, 40 pre zelenú kocku, atď.). Najhoršiu úspešnosť sme zaznamenali 70 % pri prvom pokuse o rozpoznanie kvádra. Na obrázku 4.3 uvádzame hodnoty spolu za prvý aj druhý pokus. Výraznejší rozdiel medzi sieťami sme zaznamenali iba v prípade rozpoznania kvádra, kde bola menšia úspešnosť rozpoznania v prípade reprezentácie pomocou kontúr.



Obr. 4.3: Graf úspešnosti neurónových sietí na dátach. Naľavo je graf siete trénovanej pre opísané útvary, napravo pre kontúry. Sieť pre rozpoznávanie farby je rovnaká v oboch prípadoch. Prvý riadok je pre kocku, druhý pre kváder a tretí pre guľu. Prvý stĺpec je červená farba, druhý zelená a tretí modrá. Hnedá v grafe reprezentuje farbu a oranžová tvar. Ako je vidieť farbu sieť určila presne vždy. Výraznejší rozdiel pri tvare je badať len pri kvádri. 100 % sa rovná 40/40 pokusov.

Poslednou časťou našej práce bola implementácia modulu, ktorý by bol zodpovedný za aktívnu manipuláciu s objektom za účelom lepšieho rozpoznania. Urobili sme to pomocou doprednej neurónovej siete, ktorej vstup tvorila vektorová reprezentácia tvaru plus staré hodnoty natočenia kĺbov ramena robota (preškálované na interval (-1, 1)) a jej výstup boli nové hodnoty natočenia kĺbov. Tieto tréningové dvojice boli vytvorené vo fáze testovania predošlého modulu a do pomocného súboru sa ukladali iba vtedy, keď bol objekt rozpoznaní „lepšie“, to znamená keď bola aktivácia neurónu väčšia ako pri predošlom natočení kĺbov. Výsledok testovania takejto siete bol, že robot už priemerne po treťom natočení kĺbov vedel naisto aký objekt drží a svoju pozíciu už nemenil – nemal už lepšiu pozíciu, z ktorej by získal lepšie hodnoty rozpoznaného tvaru (obrázok 4.4). Úspešnosť

rozpoznania objektu tak neklesla pod 85 %. Zlepšili sme tak pasívne rozpoznávanie objektu z náhodných otočení ramena robota aktívnym výberom ďalšieho stavu natočenia.

Obr. 4.4: Postupné aktívne natočenie kĺbov ruky. Na prvom obrázku je počiatočný stav, na druhom po prvom natočení a na treťom je po druhom natočení kĺbov. Táto poloha sa už ďalej nemenila.

Záver

Cieľ bakalárskej práce sa nám podarilo splniť. Hlavným cieľom bakalárskej práce bolo navrhnúť a implementovať model neurónovej siete na aktívne rozpoznávanie uchopených objektov. Na to sme museli mať nainštalovaný simulátor robota iCub. Ďalej sme použili voľne dostupnú knižnicu OpenCV a jej funkcie na reprezentáciu pozorovaného objektu na základe vizuálneho vstupu z pravej kamery robota. Následne sme pomocou knižnice FANN implementovali niekoľko modulov na rozpoznávanie objektov. Ku každému z nich sme uviedli ich návrh a parametre, ako aj výslednú úspešnosť.

Prvý modul, ktorý sme implementovali, sa naučil spoľahlivo rozpoznávať farbu objektu na scéne. Bola to klasická dopredná neurónová sieť s jednou skrytou vrstvou. Pri správnom natrénovaní a voľbe parametrov sa nám podarila dosiahnuť 100% úspešnosť nie len na tréningových, ale aj na testovacích vzorkách. To znamená, že sieť presne rozpoznala farbu objektu z obrázkov, ktoré nikdy pred tým nevidela.

Ďalej sme implementovali dva moduly na rozpoznávanie tvaru objektu. Jeden bol trénovaný a testovaný na reprezentácii objektu pomocou obrysov. Pri druhom bol pozorovaný objekt reprezentovaný ako opísaný útvar (štvorec, obdĺžnik alebo elipsa). Výrazné rozdiely sme pri ich testovaní neobjavili. Najnižšia úspešnosť bola 70% pri prvom module a 75% pri druhom module.

Ako tretí sme implementovali modul na aktívne rozpoznávanie objektov pomocou ich manipulácie za účelom lepšieho odhadu tvaru uchopeného objektu. Bola to dopredná neurónová sieť s jednou skrytou vrstvou. Pomocou nej sa robot vedel rozhodnúť ako má natočiť svoje kĺby aby lepšie odhadol tvar uchopeného objektu. Podarilo sa nám ju implementovať tak, že robot v priemere už po treťom natočení polohu svojich kĺbov nemenil. Dosiahli sme tak, že úspešnosť testovania pri 20 vzorkách neklesla pod 85 %.

Priestor pre ďalšie potenciálne pokračovanie vidíme vo využití iných ako nami spomenutých objektov – kocka, kváder, guľa. Simulátor za týmto účelom umožňuje importovať si vlastné, vytvorené objekty. Ďalším návrhom môžu byť viacfarebné objekty, napr. susedné strany kvádra budú mať špecifické farby. Aktívne rozpoznávanie by sa dalo zlepšiť predikciou akcií pred tým, ako ich robot vykoná.

Literatúra

Bosh, A., Zisserman, A., Munoz, X. (2007), Representing shape with a spatial pyramid kernel. International conference on image and video retrieval, strany 401-408, ACM.

Engelbrecht, A. P. (2007), Computational Intelligence: An Introduction, Second Edition, John Wiley & Sons, University of Pretoria, South Africa.

FANN (2013), Fast Artificial Neural Network Library, http://leenissen.dk/fann/wp/

Farkaš, I. (1997), SOM. Z knihy Kvasnička V., Beňušková Ľ., Pospíchal J., Farkaš I., Tiňo P. a Kráľ A. (1997), Úvod do teórie neurónových sietí, Iris, Bratislava.

Haykin, S. (2008), Neural Networks and Learning Machines, tretia edícia, Pearson Education.

Kohonen, T. (1982), Self-Organized Formation of Topologically Correct Feature Maps, Biological Cybernetics 43 (1): 59–69.

Kohonen, T. (1989), Self-Organization and Associative memory, tretie vydanie, Springer

Kohonen, T. (2001), Self-Organizing Maps, tretie vydanie, Springer.

Kvasnička, V., Beňušková Ľ., Pospíchal J., Farkaš I., Tiňo P. a Kráľ A. (1997). Úvod do teórie neurónových sietí, Iris, Bratislava.

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

Pfeifer, R., Damian D., Füchslin R. (2010), Neural Networks, University of Zurich.

SOM Toolbox, 2003, Online documentation, http://www.cis.hut.fi/projects/somtoolbox/package/docs2/somtoolbox.html

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 IEEE Workshop on Performance Metrics for Intelligent Systems* (PerMIS08).

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

Zdechovan, L. (2012), Modelovanie uchopovania objektov pomocou neurónových sietí v robotickom simulátore iCub, Diplomová práca, FMFI UK, Bratislava

Príloha A - CD

Na priloženom disku CD sa nachádza text tejto bakalárskej práce v elektronickej podobe spolu s obrázkami vo väčšom rozlíšení spomenutými na príslušných miestach v texte. Ďalej prikladáme zdrojový kód výslednej aplikácie spolu s vygenerovanou dokumentáciou a zdrojový kód pomocného programu na trénovanie neurónových sietí.