České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra kybernetiky

Systém vizuomotorické koordinace humanoidního robota Bioloid

Diplomová práce

Autor práce: Bc. Martin Plaček
Studijní program: Biomedicínské inženýrství a informatika (magisterský)
Obor: Biomedicínská informatika

Vedoucí práce: Mgr. Michal Vavrečka, Ph.D.

Praha 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Martin P l a č e k

Studijní program: Biomedicínské inženýrství a informatika (magisterský)

Obor: Biomedicínská informatika

Název tématu: Systém vizuomotorické koordinace humanoidního robota Bioloid

Pokyny pro vypracování:

Cílem práce je implementace architektury založené na neuronových sítích, která umožní humaniodnímu robotu Bioloid řídit pohyb svých rukou na základě obrazu z kamery. Navržený systém vizuomotorické koordinace umožní robotovi ukazovat na objekty v prostoru pomocí naučených propojení mezi vizuálními a motorickými reprezentacemi. Student provede rozbor stávajících metod a popíše možnosti jejich využití v robotickém systému. Následně navrhne systém schopný učit se koordinovat vizuální vstupy a motorické pomocí neuronových sítí. Systém v robotovi otestuje a provede analýzu fungování, včetně efektivity učení.

Seznam odborné literatury:

[1] Vavrečka, M.; Farkaš, I.; Lhotská, L.: Bio-inspired Model of Spatial Cognition. In Lecture Notes in Computer Science 7062 LNCS (PART 1), pp. 443-450, 2011.
[2] Haykin, S.: Neural Networks and Learning Machines (3rd Edition), Prentice Hall, 2009.

Vedoucí diplomové práce: Mgr. Michal Vavrečka, Ph.D.

Platnost zadání: do konce zimního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 7. 2014

Poděkování

Na tomto místě bych rád poděkoval především vedoucímu práce Mgr. Michalu Vavrečkovi, Ph.D. za cenné rady a připomínky v průběhu zpracování této diplomové práce.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne …….. ……..

Podpis autora práce

Anotace

Tato práce se zabývá návrhem a implementací systému založeném na umělých neuronových sítí, který umožní humanoidnímu robotovi Bioloid řídit pohyb svých rukou na základě obrazu z kamery. Systém je založen na samoorganizující mapě (SOM), která vytváří propojení mezi vizuální a motorickou informací. Robot využívá systém k ovládání své ruky, ve které drží stínítko a které se učí umísťovat tak, aby jím zastíňoval objekt, který se před robotem pohybuje. Systém je naimplementovaný v prostředí MATLAB s využitím SOM Toolboxu a Bioloid Toolboxu. S navrženým systémem je robot schopný naučit se umísťovat stínítko s přesností přes 90% po pouhých 30 trénovacích iteracích.

Klíčová slova

samoorganizující mapa, SOM, robot, Bioloid, Matlab, vizuomotorická koordinace

Abstract

This thesis deals with a design and an implementation of a system based on an artificial neural networks. The system gives a Bioloid robot an ability to control its hand and makes the movements based on an image information acquired from a camera. The system is based on an self-organizing map that creates a link between a visual and a motoric informations. The robot uses the system to learn a visuomotor coordination. The robot holds a rod with a small ball at its end. There is an object in front of the robot and the robot learns how to place the ball between the object and the camera in order hide the object. The system is implemented in MATLAB and uses SOM Toolbox and Bioloid Toolbox. The robot was able to place the ball in the correct position with an accuracy over 90% after only 30 learning steps.

Keywords

self-organizing map, SOM, robot, Bioloid, Matlab, visuomotor coordination

Obsah

Seznam obrázků

1 Úvod

Práce se zabývá problematikou ovládání pohybů robota, tak aby robot mohl interagovat s okolním prostředím, o kterém získává informace prostřednictvím kamery. Cílem práce je navrhnou a implementovat systém, který umožní humanoidnímu robotovi Bioloid naučit se ovládat své pohyby na základě vizuálních vstupů.
Navrhovaná architektura učícího systému je založená na vytváření propojení mezi vizuálními a motorickými reprezentacemi skutečného světa. Propojení jsou vytvářena v topologické samoorganizační mapě, jejíž struktura vychází z poznatků studia mozkové kůry lidí a savců obecně.
Jedním z výrazných znaků mozkové kůry savců je topografická organizace senzorické oblasti. Sousedící neurony jsou drážděny stimuly, které pocházejí ze sousedících pozic v prostoru vstupních sensorů a sousedící neurony v jedné části mozku se promítají do sousedících neuronů v další části mozku. Neboli, že propojení zachovává sousedské vztahy a vytváří tak takzvané topografické mapy. V případě somatosensorické kůry se jedná o somatotopickou mapu (povrch těla), v případě sluchové kůry o tonotopickou mapu (spektrum možných zvuků) a v případě zrakové kůry o retinotopickou mapu. Podobná topografická organizace existuje také v motorické oblasti, kde sousedící neurony mají sklon kódovat podobné konfigurace pohybů.
Samoorganizující mapa použitá v systém pro robota Bioloid slučuje senzorickou a morotickou oblast do jedné mapy a vytváří tak jednu topografickou organizaci vizuomotorických vstupů.

Při návrhu vlastního vizuomotorického systému pro robota Bioloid jsem čerpal především z práce *Thomase Martinetze a Klause Schultena* [1], ve které učili uchopovat objekt robotického rameno s pěti stupni volnosti.
Robot Bioloid má, oproti velkému robotickému ramenu, značně omezené možnosti pohybu a proto jsem chytání objektu nahradil umísťováním stínítka, při kterém si robot bez problémů vystačí se třemi stupni volnosti, které má v jedné ruce. Toto zjednodušení také umožnilo postavit systém vizuomotorické koordinace na jedné dvourozměrné samoorganizující mapě.

Pro implementaci architektury vizuomotorické koordinace jsem zvolil programové prostředí MATLAB. Hlavním důvodem byla jistota fungující komunikace s robotem a také množství užitečných toolboxů, které je možné při implementaci využít. Nevýhodou této volby se, v průběhu řešení, ukázaly být velmi omezené možnosti paralelilizace procesů, takže nebylo možné zároveň ovládat robota a zpracovávat obraz z kamery. Bylo tedy nutné pohyb robota vždy rozfázovat na několik menších dílčích pohybů a po každé fázy zpracovat obraz z kamery.

2 Využití samoorganizujících map v robotice

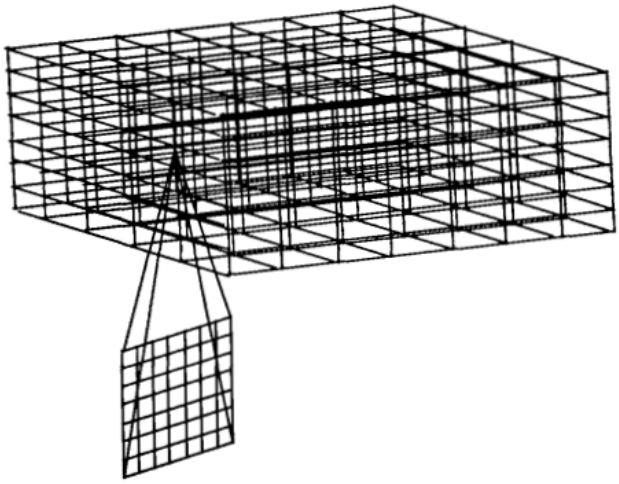
Možnost pohybu a schopnost svoje pohyby ovládat je jednou z nejstarších úloh, které živé organizmy musely vyřešit aby dokázaly přežít. V průběhu evoluce vzniklo mnoho různorodých systémů pro ovládání pohybů a většina z nich překoná současné robotické řídící systémy v mnoha ohledech. Jako příklad jmenujme flexibilitu, rychlost, toleranci k chybám, energetickou úspornost a především schopnost učit se. Máme tedy opodstatněný důvod investovat značné úsilí do hlubšího pochopení biologických systémů a použit získané poznatky při vývoji nových robotických systémů s biologicky inspirovanými principy řízení pohybů.
Význam milníkem využití biologicky inspirovaných principů jsou umělé neuronové sítě, které simulují strukturu a fungování mozku. Právě na mozek je výzkum celkově zaměřený, protože je u obratlovců řídícím centrem a spolu s dalšími oddíly tvoří hlavní orgán centrální nervové soustavy. S tím jak jsou objevovány nové principy fungování mozku, tak také vznikají nové typy umělých neuronových sítí, které nové poznatky prakticky využívají. Jednou z takových sítí jsou samoorganizující mapy, které vychází ze struktury mozkové kůry.
Samoorganizující mapy jsou tradičně používané v datové analýze ke shlukování, ale díky jejich topologickému uspořádání neuronů jsou také vhodné k učení mapování vztahů mezi vizuálními a motorický vstupy a tím vytvoření systému pro řízení robota.

Využití samoorganizujících map pro ovládání robotů není nová myšlenka a na toto téma bylo již publikováno větší množství prací (například [1-7]). Ve většině vytvořených robotických systém nejsou použity samoorganizující mapy ve své základní podobě ale typicky tvoří hierarchickou strukturu z více neuronových sítí. Některé systémy také využívají spiking neurony, mají naimpementovanou autoregulaci učení, nebo mají schopnost plánování trasy. Samoorganizující mapy v robotice jsou stále ve fázy výzkumu a proto existuje a vzniká mnoho návrhů, jak mapy využít nebo vylepšit pro praktické využití při řízení robotů.

V následujím textu jsou představeny některé význačné robotické systémy, které využívají samoorganizující mapy k učení robota ovládat svoje pohyby na základě vizuálních informací.

Prvním je systém navržený dvojcí *T. Martinetz a K. Schulten* [1], ve kterém pro ovládání jejich robota vytvořili neuronovou topografickou strukturu k učení mapování mezi pozicí cylindrického objektu a kloubovými souřadnicemi, které robotické rameno s 5 stupni volnosti nastaví tak, aby dokázalo objekt uchopit. K určování pozice objektu v trojrozměrném prostoru použili dvě kamery. Z obrazu každé kamery je vypočtena pozice objektu v jejím zorném poli a tím jsou získány čtyři čísla. Tato čtveřice je předložena hlavní třírozměrné síti neuronů (Obrázek 2.1), která se učí topologické

mapování mezi čtyřrozměrným vstupem a její třírozměrnou reprezentací. Po naučení každý neuron reprezentuje jednu pozici objektu ve skutečném světě. S každým neuronem je také spjata jedna doplňková dvourozměrná síť neuronů, která vytváří mapování orientace objektu. Ke každému neuronu v hlavní sítí jsou přiřazeny kloubové souřadnice, které umístí rameno k objektu a ke každému neuronu v doplňkové síti jsou přiřazeny kloubové souřadnice, které natočí zápěstí tak, aby bylo možné objekt uchopit.
Po 1000 trénovacích krocích měl systém chybu umísťování 1,2% velikosti pracovního prostoru a chybu natočení zápěstí pod 6,8°. Po 10000 trénovacích krocích se chyba snížila na 0,3% respektive 3,8°.



Obrázek 2.1: Architektura hierarchické neuronové sítě.

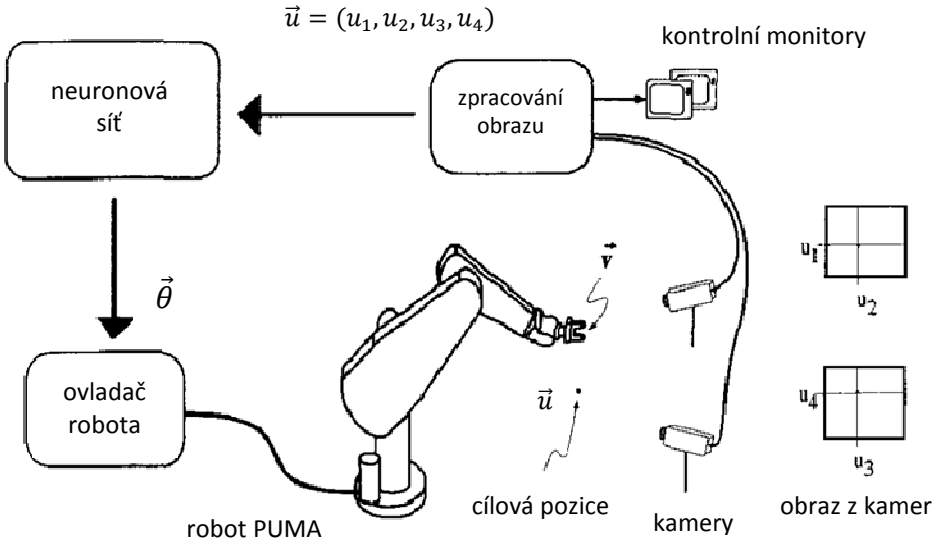
Druhý robotický systém je součástí většího vědeckého projektu, který má za cíl převést nové poznatky z oblasti neurověd do praktického využití v robotice. Práce skupiny *S. V. Adams, T. Wennekers, S. Denham a P. F. Culverhouse* [2] je zaměřená na část, ve které se malý autonomní robot naučí základní vizuomotorickou koordinaci s využitím samoorganizující topologické mapy. Vstupy pro učení je 8 světových směrů (S, SV, V, JV, J, JZ, Z, SZ), každý zakódovaný do 16 rozměrného vektoru.
Učení robota je založené na tradiční samoorganizující mapě, ale s několika klíčovými úpravami. První úpravou je využití spiking neuronů namísto neuronů se spojitou aktivační funkcí. Při učení jsou tak využity jak prostorové, tak i časové vztahy mezi neurony. Díky spiking neuronů lze při učení využít Hebbovský princip, který vychází z poznatků fungování skutečných neuronů. Při Hebbovském učení jsou posilovány vazby mezi neurony, které pálí najednou nebo bezprostředně po sobě.
Druhou úpravou je samoregulující rychlost učení. Mapa díky tomu dokáže bez vnějšího zásahu přizpůsobovat parametry a měnit průběh učení podle předkládaných trénovacích vzorů a stavu neuronů uvnitř sítě. Samoregulující učení má velký potenciál pro využití v autonomních robotech, protože není nutné dělat dopředu přepoklady o prostředí ve kterém se bude robot pohybovat a robotovi tak není nutné dopředu

nastavovat parametry učení. Mapa je také schopna reagovat na nenadálé situace, které můžou v průbehu učení nastat.

Industriální robotické rameno Puma 562 je použito ve třetím systému, který vytvořili *J. A. Walter a K. J. Schulten* [3]. Robot se v něm učí umísťovat konec svého efektoru na základě informací získaných výhradně jen z dvojce kamer. Bez využití externího učitele a bez znalosti své geometrie. Pro učení jsou naimplementovány dva algoritmy. První využívá "neural-gas" síť a druhý upravenou samoorganizující mapu.
Celý systém je schématicky znázorněn na obrázku (Obrázek 2.2). V každé iteraci je náhodně vybrána cílová pozice v pracovním prostoru. Obrazy ze dvou kamer snímajících scénu jsou zpracovány a jsou získány dvourozměrné souřadnice pozice v každém z obrazů. Z obou párů souřadnic je složen jeden čtyřrozměrný vektor $\vec{u} = (u_1, u_2, u_3, u_4)$. Množina všech těchto vektorů tvoří vstupní prostor neuronové sítě.
Aby robot dokázal umísťovat konec efektoru do správné pozice, musí znát transformaci $\Theta(\vec{u})$ z $\vec{u}$ do příslušných kloubových souřadnic $\vec{\theta}$. Základní myšlenkou, jak se robot tuto silně nelineární transformaci učí, je adaptivní diskretizace vstupního prostoru do množiny disjunktních buněk a aproximace $\vec{\theta}$ v každé buňce lineárním mapováním, které se učením postupně zpřesňuje. Každé buňce je přiřazen jeden neuron, který v sobě udržuje tři datové typy - vektor definující střed buňky $\vec{\omega}_\mu$, výstupní vektor kloubových souřadnic $\vec{\theta}_\mu$ a modifikovanou Jakobiho matici A_μ. Provední transformace a získání výsledných kloubových souřadnic je vypočteno podle

$$\vec{\Theta}(\vec{u}) = \vec{\theta}_\mu + A_\mu \cdot (\vec{u} - \vec{\omega}_\mu)$$

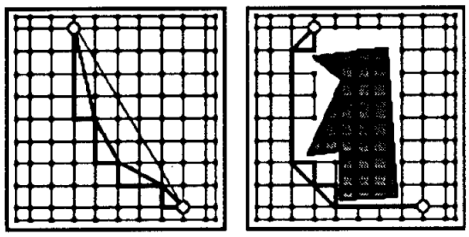
Výběr diskretizace buněk a adaptivní učení středů buněk a Jakobiho matice je zajišťováno "neural-gas" sítí a nebo alternativně samoorganizující mapou.
Oba vytvořené algoritmy mají přibližně srovnatelný výkon. Po 3000 trénovacích iterací mají přesnost umísťování 0,1% (1,3mm) velikosti pracovního prostoru a díky adaptivní zpětné vazbě dokáží naučenou síť přeučovat a tím kompenzovat náhlé změny v robotově geometrii.



Obrázek 2.2: Schématické znázornění robotického systému.

Čtvrtý robotický systém navržený dvojcí *J. B. Saxon a A. Mukerjee* [4] je postavený kolem robotické ruky (Neurobot) se dvěma stupni volnosti. Robot se s využitím samoorganizující neuronové sítě učí vytvářet asociace mezi vizuálními a kloubovými souřadnicemi. Při učení jsou jako vstupy použity hodnoty z obou prostorů a naučená mapa je tedy neurony tvořená nadrovina, ve které každý neuron reprezentuje bod jak ve vizuálním prostoru, tak i v prostoru kloubových souřadnic.

Vytvořené mapování mezi pracovním prostorem a prostorem kloubů je využito k plánování trasy efektoru a vyhýbání se překážkám. Díky tomu, že neurony v mapě jsou topologicky uspořádané a pokrývají celý pracovní prostor, můžeme na ně pohlížet jako na uzly v grafu a na laterální vazby mezi sousedními neurony můžeme pohlížet jako na hrany grafu. Plánování trasy pak probíhá jako hledání nejkratší cesty mezi neuronem, který reprezentuje současnou pozici efektoru a neuronem, který reprezentuje cílovou pozici. V případě, že se v pracovním prostoru vyskytuje překážka, jsou z hledání cesty vynechány všechny neurony, které odpovídají pozici překážky (Obrázek 2.3).



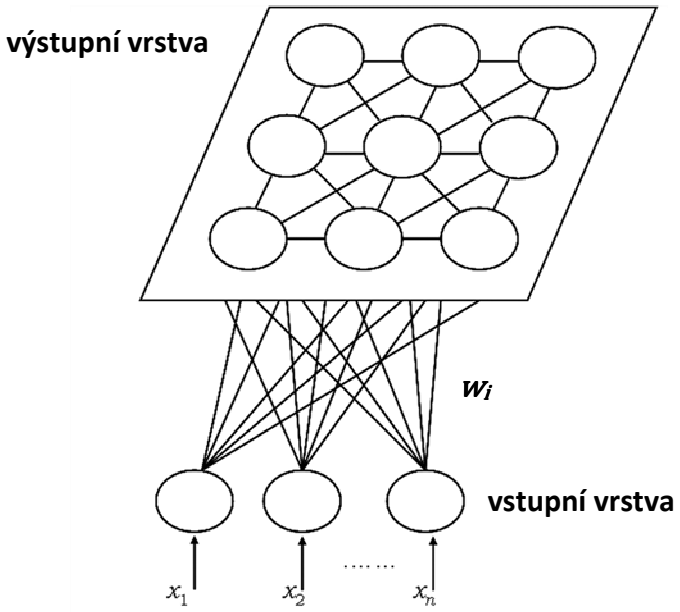
Obrázek 2.3: Naučená neuronová síť se znázorněnou trasou efektoru.

3 Samoorganizující mapy

Samoorganizující mapy (Self-organizing map, SOM) jsou typ umělých neuronových sítí, který využívá učení bez učitele a vytváří diskrétní nízko-dimenzionální reprezentaci trénovacích vzorů. Účení sítě je založeno na strategii kompetitivního učení, tedy po předložení trénovacího vzoru mezi sebou neurony soutěží a zvítězí ten, jehož vzdálenost je od předloženého vzoru nejmenší. Vítěznému neuronu a jeho nejbližším sousedům jsou následně upraveny váhy tak, aby se přiblížily předloženému vzoru. Tomuto postupu učení se také říká samoorganizace, což znamená, že síť je schopná přízpůsobit se trénovacím vzorům a nepotřebuje k tomu znát předpokládané výstupy.

3.1 Topologie

SOM je dvouvrstvá sít s uplným propojením neuronů mezi vstupní a výstupní vrstvou (Obrázek 3.1). Počet neuronů ve vstupní vrstvě je roven dimenzi trénovacích vzorů. Neurony ve výstupní vrstvě jsou uspořádány do pravidelné nízko-dimenzionální struktury s určitým topologickým uspořádáním. Nejčastěji dvourozměrnou mřížkou s neurony uspořádanými do čtvercové nebo šestiúhelníkové struktury. Již méně používané jsou struktury jednorozměrné s neurony uspořadými do řady nebo struktury třírozměrné uspořádané do třírozměrné mřížky. Topologie uspořádání neuronů má vliv na tvar okolí neuronu a na vymezení, které neurony spolu sousedí. Neurony, které jsou na okraji mřížky mají méně sousedů, než neurony, které jsou uvnitř mřížky, což může vést k nevyváženosti a deformaci mřížky do jejího středu. Aby se mřížka vyvážila, používají se topologie, kde krajní neurony mají jako své sousedy také neurony na protější straně mřížky. Z jednorozměrné řady se tak stane prstenec a z dvourozměrné mřížky toroid. Všechny neurony ve výstupní vrstvě jsou také propojeny laterálními vazbami.



Obrázek 3.1: Ukázka samoorganizující mapy se čtvercovou topologií ve výstupní vrstvě.

Síla propojení neuronů mezi vstupní a výstupní vrstvou je pro každý neuron ve výstupní vrstvě definována vektorem vah $\boldsymbol{w}$, který má rozměr stejný jako je počet neuronů ve vstupní vrstvě.

3.2 Funkce sousedství

Funkce sousedství definuje poloměr okolí kolem určitého neuronu a všechny neurony, které se v tomto okolí nacházejí jsou označeny jako jeho sousedé. Funkce může být diskrétní, tedy že neuron do okolí patří nebo nepatří, nebo spojitá, tedy že příslušnost neuronu do okolí je definována určitou funkcí závislou na vzdálenosti od středového neuronu. Často používané funkce jsou Gaussova nebo Mexický klobouk.

3.3 Učení

Algoritmus učení je rozdělen na dvě fáze - soutěžní a spolupracující. V první fázi je nalezen nejpodobnější neuron, neboli vítězný neuron a v druhé fázi jsou upraveny váhy tohoto vítězného neuronu a všech jeho sousedů. Učení probíhá iterativně postupným předkládáním trénovacích vzorů až do vyčerpání trénovací množiny.

3.3.1 Soutěžní fáze

Pro trénovací vzor $\boldsymbol{x}$ je nalezen neuron i^* s nejmenší Eukleidovskou vzdáleností.

$$i^* = \underset{i}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{w_i}\|$$

3.3.2 Spolupracující fáze

V této fázy je zásadní, že váhy neuronů nejsou upravovány nezávisle, ale jsou vždy upravovány váhy vítězného neuronu včetně jeho sousedů. Díky tomu lze dosáhnout vytvoření topografické mapy, ve které reagují sousedící neurony na podobné vzory. Váhy neuronů jsou upravovány tak, aby se vždy přiblížili předloženému vzoru. Velikost změny vah je dána rychlostí učení $\alpha(t)$ a vzdáleností neuronu od vítězného neuronu $h(i^*,i,t)$.

$$\boldsymbol{w_i}(t+1) = \boldsymbol{w_i}(t) + \alpha(t) \cdot h(i^*, i, t) \cdot [\boldsymbol{x}(t) - \boldsymbol{w_i}(t)]$$

Aby se zajistila konvergence k určitému stabilnímu stavu, jsou rychlost učení a velikost okolí s kroky adaptace snižovány. Obvykle mají lineární, po částech lineární nebo exponeciální závislost na kroku adaptace.

3.4 Vybavování

Při vybavování je naučené síti předložen neznámý vzor a je nalezen nejbližší neuron obdobně jako při učení. Tento neuron je odpovědí sítě a udává shluk, do kterého je neznámý vzor přiřazen.

Fáze vybavování může být neadaptační a nebo adaptační. V případě adaptační je neznámý vstupní vzor zařazen do shluku a následně je použit i pro doučení sítě, kdežto při neadaptační je jen zažazen do shluku.

3.5 Vizualizace

U-matrix je často využívaným vizualizačním nástrojem pro prezentaci výsledků získaných samoorganizujícími mapami. V U-matrix jsou Eukleidovské vzdálenosti mezi váhami sousedících neuronů zakódovány do odstínů šedi nebo barev a tím vznikne obrázek, ve kterém je dobře patrné, které neurony jsou si blízké a vytvářejí shluk. Ukázka U-matrix je v obrázku (Obrázek 9.3).

4 Bioloid Toolbox

Některé funkce, které jsou teď součástí Bioloid Toolboxu, začaly vznikat ještě před touto diplomovou prací a sloužily jako rozhraní pro ovládání Bioloid robota z prostředí Matlabu. Jejich hlavní využití bylo v demonstračních úlohách, které na skutečném robotovi předváděly využití umělých neuronových sítí v robotice. Obě dvě úlohy jsou stále funkční a jsou používany při výuce předmětu Neuroinformatika vyučovaného na katedře kybernetiky (ČVUT, FEL).
K původně několika základním funkcím byly postupně přidány další pro ovládání pokročilejších schopností robota. Aby ovládání zůstalo nadále přehledné, byly funkce systematicky sjednoceny a tím vznikl Bioloid Toolbox. Jelikož se jedná o univerzální nástroj s potenciálem pro širší využití, byl toolbox zpřístupněn volně ke stažení na webové stránce [11]. V současnosti má desítky stažení z různých států světa.

Komunikace s robotem Bioloid je založená na obousměrném předávání paketů. Z počítače do robota jsou posílány pakety příkazové, které jsou buď zpracovány přímo v řídící jednotce a nebo jsou po sběrnici Dynamixel přeposlány do servomotorů. V opačném směru, tedy z robota do počítače, jsou posílány pakety stavové. Příkazové pakety obsahují identifikační číslo příjemce (ID), příkaz a jeden nebo více parametrů. Pakety stavové obsahují odezvu na příkaz a jsou odeslány vždy po přijmutí a zpracování příkazového paketu. Stavové pakety tedy fungují i jako potvrzení o příjetí příkazového paketu.
Dynamixel sběrnice slouží pro připojování periferií k centrální jednotce robota. Kromě již zmíněných servomotorů lze také připojit kameru, různé senzory nebo i vlastní zařízení. Přenos dat na sběrnici je založen na komunikačním standardu RS232. Data jsou přenášena po jedné lince a nejsou použity žádné servisní signály. Veškerá komunikace tak probíhá v half-duplexním režimu po jednom vodiči. Pro kódování logických hodnot jsou použity TTL napěťové úrovně. Periferie jsou na sběrnici připojované do série za sebou a vytvářejí tak řetěz. Spolu s datovým kanálem je k periferiím vedeno také napájení a zařízení jsou tedy s centrální jednotkou propojeny třívodičovým kabelem.

Bioloid toolboxu funguje jako zapouzdření komunikace na úrovni posílání paketů a umožňuje pohodlné ovládání robota voláním funkcí s patřičnými parametry. Funkce zajistí správné vytvoření příkazového paketu na základě předaných parametrů a následně i jeho odeslání do robota skrze virtuální sériový port. Taktéž zajistí příjem stavového paketu a v případě potřeby i vyextrahuje z paketu odpověď a předá ji jako návratovou hodnotu z funkce.

Základní komponentou celého toolboxu je instance objektu **serial port**, která je spjata se sériovým portem na kterém je robot připojen k počítači. Tato instance funguje jako komunikační portál pro odesílání a přijímání paketů a proto je jako parametr předávána do naprosté většiny funkcí v toolboxu.

Neoddělitelnou součástí toolboxu je, kromě funkcí v Matlabu, také vlastní firmware pro řídící jednotku robota. Jelikož výchozí firmware nedisponuje potřebnými schopnostmi, musel jsem naprogramovat vlastní firmware, který dokáže přeposílat pakety mezi počítačem připojeným přes USB a servomotory připojenými přes Dynamixel sběrnici. Další naprogramovanou funkcí ve firmware je čtení hodnot z analogových senzorů, které jsou připojeny přímo do řídící jednotky. Firmware je napsaný v jazyce c, ale spolu s toolboxem je distribuován již přeložený do strojového kódu, který stačí jen nahrát do paměti mikrokontroléru uvnitř řídící jednotky. Kvůli odlišným mikrokontrolérům mezi různými typy řídících jednotek je firmware možné použít jen s řídící jednotkou CM530.
Aby toolbox fungoval, je nejprve nutné do mikrokontroléru nahrát příslušný firmware, což lze provés pomocí programu RoboPlus, který je dodáván spolu s robotem. Nahráním firmware pro toolbox je přepsán původní fimware a nebude možné plnohodnotně využít programy od výrobce robota. Pro navrácení původního firmware lze opět použít program RoboPlus.

Ovládání servomotorů a dalších perferií na dynamixel sběrnici je založené na zapisování hodnot do registrů v paměti uvnitř zařízení. O skutečné provedení příkazu se pak stará mikrokontrolér, který dané zařízení řídí. Z tohoto principu ovládání vychází obsah příkazového paketu, který přenáší tři základní údaje - identifikační číslo zařízení, číslo registru a hodnotu.
Identifikační číslo funguje jako adresa a musí být v rámci jedné sběrnice unikátní pro všechna připojená zařízení. **Číslo registru** je adresa registru v paměti zařízení. Pokud zapisovaná hodnota zabírá více bytů, pak se jedná o adresu prvního bytu. **Hodnota** je číslo, které bude zapsáno do registru.

Kromě zápisu hodnot do registru je také možné hodnoty číst a tím získat informace o stavu zařízení. Příkazový paket pro čtení obsahuje taktéž tři základní údaje - identifikační číslo zařízení, číslo registru a počet bytů.
Identifikační číslo a číslo registru mají stejný význam jako u zapisovacího příkazu. **Počet bytů** udává kolik bytů má být přečteno.

Kromě údajů základních obsahuje paket také několik servisních údajů, které slouží pro snažší a spolehlivější přenos celého paketu. Konkrétně se jedná o značku začátku paketu, délku paketu a kontrolní součet.

Typický průběh ovládací funkce a výměny dat mezi počítačem a robotem je možné demonstrovat na následujícím příkladu zavolání funkce **setGoalPosition(s, ID, value**), která otáčí osou servomotoru. Funkce má tři parametry - instanci sériového portu **s**, identifikační číslo motoru **ID** a požadovanou hodnotu natočení **value**. Pro tento konkrétní příklad předpokládejme **ID=11** a **value=700**.
Natočení motoru je vyjádřeno jako 10 bitové číslo s rozsahem 0 až 1023. Komunikace přes sériový port ale probíhá po bytech (8 bitů), které mají rozsah hodnot jen 0 až 255. Proto je nutné požadovanou hodnotu natočení nejprve rozdělit na dva byty - horní a dolní. Označíme li horní byte UB a dolní byte LB, potom

$$UB = \left\lfloor \frac{value}{256} \right\rfloor = 2, \qquad LB = value \bmod 256 = 188$$

Kde mod je operace zbytek po dělení a ⌊ ⌋ je operace dolní celá část.

Následně je vytvořen příkazový paket podle schématu:

[255, 255, ID, N+3, požadavek, registr, hodnota1,... , hodnotaN, CS]

- **2x 255** je značka začátku paketu
- **ID** je identifikační číslo zařízení
- **N** je počet posílaných hodnot
- **požadavek** je v tomto případě zápis (kódovaný číslem 3)
- **registr** je adresa v paměti
- **hodnota1**,... **hodnotaN** jsou posílané hodnoty
- **CS** je kontrolní součet vypočítaný podle

$$CS = 255 - [(ID + N + 3 + požadavek + registr + hodnota1 + \cdots + hodnotaN) \bmod 256]$$

Pro tento příklad bude mít příkazový paket hodnoty

[255, 255, 011, 005, 003, 030, 188, 002, 016]

Tento paket je odeslán a čeká se na přijetí stavového paketu, který je předán jako výstup z funkce. Pokud stavový paket nedorazí ve stanoveném časovém intervalu, tak je vypsáno chybové hlášení a funkce skončí.

Druhý demonstrační příklad výměny dat s robotem je na čtecí funkci **getPresentPosition(s, ID)**, která slouží ke zjištění, na jaké hodnotě je servomotor natočený. Funkce má dva parametry - instanci sériového portu **s** a identifikační číslo motoru **ID**.

Na začátku funkce je vytvořen paket podle podobného schématu jako v předchozím příkladu:

[255, 255, ID, N, požadavek, registr, počet, CS]

- **2x 255** je značka začátku paketu
- **ID** je identifikační číslo zařízení
- **N** je konstanta s hodnotou 4
- **požadavek** je v tomto případě čtení (kódované číslem 2)
- **registr** je adresa v paměti
- **počet** je počet čtených bytů
- **CS** je kontrolní součet vypočítaný podle

$$CS = 255 - [(ID + N + požadavek + registr + počet) \bmod 256]$$

Natočení servomotoru je uloženo v registrech na adrese 36 a 37. Hodnota je 10 bitové číslo a proto zabírá dva byty. Ze servomotoru je možné číst mnoho dalších údajů. Například teplotu, napájecí napětí, nastavené limity rozsahu a rychlosti otáčení. Kompletní seznam všech registrů lze nalézt v dokumentaci k příslušnému zařízení. Konkrétně pro dynamixel servomotory je seznam v uživatelském manuálu [12].

Pro servomotor s **ID=11** bude mít vytvořený paket hodnoty

[255, 255, 011, 004, 002, 036, 002, 200]

Příkazový paket je odeslán do robota a čeká se na přijetí stavového paketu s odpovědí. Pokud paket nedorazí ve stanoveném časovém intervalu, nebo dorazí poškozený, tak je odeslání zopakováno. Nepřijde li paket v pořádku ani po druhém opakování, tak je cílové zařízení považováno za nedostupné a funkce vypíše chybové hlášení a skončí. Dorazí li paket v pořádku, jsou z něj vyextrahovány dva byty s přečtenou hodnotou natočení. Následně jsou oba byty spojeny do jednoho čísla, které je navratovou hodnotou z funkce.

V následujícím soupisu funkcí, které jsem pro Bioloid Toolboxu vytvořil, jsou uvedeny pouze funkce určené pro uživatelské použití. Servisní funkce, které zajišťují komunikaci, zde uvedeny nejsou. Pro snazší vyhledávání jsou funkce seřazeny abecedně a nikoly podle účelu jejich využití. V popisu některých funkcí jsou uvedeny reference na porty na řídící jednotce. Jejich umístění lze nalézt na webové stránce výrobce [14].

getDMS

Vstup: s (serial port) instance sériového portu
Výstup: value (skalár) změřená hodnota ze senzoru
Popis: Přečte hodnotu ze senzoru meření vzdálenosti (DMS - Distance Measure Sensor) připojeného do portu ADC5 [14] v řídící jednotce.

getExternalPort

Vstup: s (serial port) instance sériového portu

Výstup: statusValues (vektor 1x6) stav externích portů v řídící jednotce

Popis: Řídící jednotka má 6 externích portů a na každém portu má dva ovladatelné piny, které lze nastavovat do logické 0 (0 V) nebo 1 (+3,3 V). Funkce přečte jaké hodnoty jsou nastaveny a stav každého portu zakóduje podle:

0: MOT#+=0, MOT#-=0

1: MOT#+=1, MOT#-=0

2: MOT#+=0, MOT#-=1

Znak # zastupuje číslo portu. Hodnoty jsou v poli seřazeny od portu 1 na první pozici až po port 6 na šesté pozici.

getGridingMatrix

Vstup: s (serial port) instance sériového portu

Výstup: colors (matice 24x32) nejčastější barva v bloku
counts (matice 24x32) počet pixelů nejčastější barvy v bloku

Popis: Získá matice barev a počtů vytvořené griding algoritmem v kameře HaViMo2. Griding algoritmus zkomprimuje obraz do dvou matic, ve kterých každá buňka reprezentuje čtverec o velikosti 5x5 pixelů v původním obraze. Matice barev udává jaká z předem definovaných barev byla ve čtverci nejvíce zastoupená a matice počtů udává kolik pixelů ve čtverci tuto barvu mělo.

getGyro

Vstup: s (serial port) instance sériového portu

Výstup: values (vektor 1x2) hodnoty z gyroskopu

Popis: Přečte hodnoty z dvouosého gyroskopu připojeného do portu ADC3 a ADC4 v řídící jednotce.

getIR

Vstup: s (serial port) instance sériového portu

Výstup: values (vektor 1x2) hodnoty z IR senzorů

Popis: Přečte hodnoty ze dvou infračervených senzorů připojených do portu ADC1 a ADC6 v řídící jednotce.

getPresentPosition

Vstup: s (serial port) instance sériového portu
id (vektor) identifikační číslo servomotoru

Výstup: out (skalár) natočení servomotoru

Popis: Získá současné natočení servomotoru.

getPresentPositionMore

Vstup: s (serial port) instance sériového portu
ids (vektor 1xN) identifikační čísla servomotorů

Výstup: out (vektor) natočení servomotorů

Popis: Získá současné natočení několika servomotorů najednou. Hodnoty ve výstupním vektoru mají stejném pořadí jako identifikační čísla ve vstupu.

isMotorMoving

Vstup: s (serial port) instance sériového portu
id (skalár) identifikační číslo servomotoru

Výstup: out (skalár) stav otáčení

Popis: Zjistí zda se servomotor otáčí. Výstup je 1 pokud se otáčí a 0 pokud se neotáčí.

setBaudRate

Vstup: s (serial port) instance sériového portu
id (skalár) identifikační číslo servomotoru
baudRate (skalár) komunikační rychlost

Výstup: statusPacket (vektor 1xN) přijatý stavový paket

Popis: Nastaví servomotoru novou komunikační rychlost. Výpočet rychlosti je v servomotoru prováděn podle vzorce

$$\text{rychlost}[\text{baud/s}] = 2000000/(\text{baudRate} + 1)$$

setBTPort

Vstup: není

Výstup: s (serial port) instance sériového portu

Popis: Vytvoří spojení s robotem přes rozhraní Bluetooth. Jelikož má Matlab určitá omezení pro přístupu k Bluetooth adaptéru, tak je nutné zajistit, aby byl adaptér připojen na COM1. Přenastavení čísla komunikačního portu lze provést ve *Správci zařízení* ve skupině *Porty (COM a LPT)*. Výstupní instance sériového portu slouží jako vstupní parametr do většiny dalších funkcí v toolboxu.

setComplianceMargins

Vstup: s (serial port) instance sériového portu
id (skalár) identifikační číslo servomotoru
margin (skalár) šířka rozsahu volnosti

Výstup: statusPacket (vektor 1xN) přijatý stavový paket

Popis: Nastaví volnost servomotoru kolem cílové pozice. Volnost je symetrická, na každou stranu je šířka nastavena na hodnotu předanou v parametru.

setExternalPort

Vstup: s (serial port) instance sériového portu
id (skalár) číslo externího portu
value (skalár)

Výstup: statusValues (vektor 1x6)

Popis: Nastaví hodnotu pinů externího portu na řídící jednotce. Každý port má dva ovladatelné piny jejichž hodnoty lze nastavit do logické 0 (0 V) nebo 1 (+3,3 V). Požadovaná hodnota pinů musí být zakódována jako číslo 0, 1 nebo 2:
0: MOT#+=0, MOT#-=0
1: MOT#+=1, MOT#-=0
2: MOT#+=0, MOT#-=1

Znak # zastupuje číslo portu. Výstupní pole obsahuje současný stav všech portů, tak jak jsou nastaveny v řídící jednotce. Hodnoty jsou v poli seřazeny od portu 1 na první pozici až po port 6 na šesté pozici.

setGoalPosition

Vstup: s (serial port) instance sériového portu
id (skalár) identifikační číslo servomotoru
value (skalár) hodnota natočení
Výstup: statusPacket (vektor 1xN) přijatý stavový paket
Popis: Pošle příkaz pro natočení servomotoru na požadovanou hodnotu. Úhel je definován v jednotkách servomotoru jako 10 bitové číslo v rozsahu 0°-300° (0°∼0, 300°∼1023).

setID

Vstup: s (serial port) instance sériového portu
id (skalár) identifikační číslo servomotoru
id (skalár) nové identifikační číslo
Výstup: statusPacket (vektor 1xN) přijatý stavový paket
Popis: Nastaví servomotoru nové identifikační číslo.

setMovingSpeed

Vstup: s (serial port) instance sériového portu
id (skalár) identifikační číslo servomotoru
value (skalár) rychlost otáčení
Výstup: statusPacket (vektor 1xN) přijatý stavový paket
Popis: Nastaví servomotoru maximální rychlost otáčení.

setSerialPort

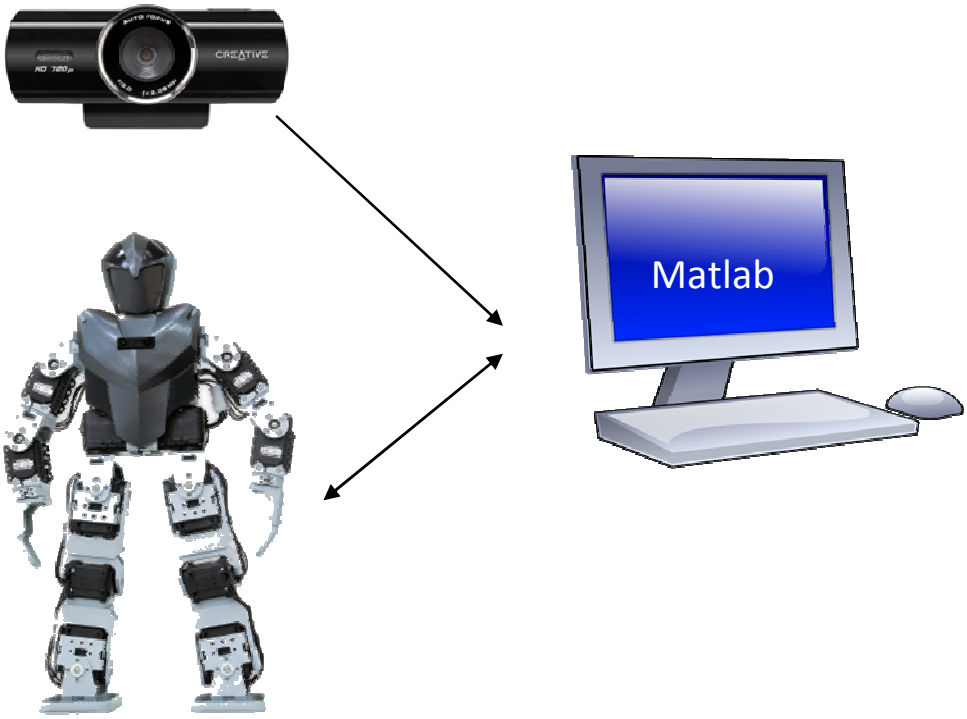
Vstup: varargin (skalár) číslo komunikačního portu
Výstup: s (serial port) instance sériového portu
Popis: Vytvoří instanci sériového portu pro komunikaci s robotem. Číslo komunikačního portu na kterém je robot připojen k počítači lze najít ve *Správci zařízení* ve skupině *Porty (COM a LPT)*. Výstupní instance sériového portu slouží jako vstupní parametr do většiny dalších funkcí v toolboxu.

setTorque

Vstup: s (serial port) instance sériového portu
IDs (vektor 1xN) identifikační čísla servomotorů
value (vektor 1xN) točivý moment
Výstup: není
Popis: Zapne nebo vypne točivý moment v servomotorech. Povolené vstupní hodnoty jsou 1 pro zapnutí a 0 pro vypnutí točivého momentu. Vstupní parametry identifikačních čísel a požadovaných hodnot jsou vektory, aby bylo možné nastavit více servomotorů jedním zavoláním funkce.

5 Popis sestavy

Celá sestava na které jsem navrhoval systém vizuomotorické koordinace se sestává ze tří hlavních komponent (Obrázek 5.1) - počítače, robota a kamery. Základní komponentou je počítač, na kterém běží algoritmus vizuomotorické kooordinace. Motoriku robota zajišťují tři servomotory, díky kterým může robot volně pohybovat svojí levou rukou. Zrak a tedy zdroj vizuální informace je zprostředkován kamerou (Creative Live! Cam Connect HD), kterou má robot umístěnou na hlavě. Robot i kamera jsou k počítači připojeny přes rozhraní USB, které umožnujě dostatečně rychlé a stabilní spojení pro výměnu dat mezi jednotlivými komponentami.



Obrázek 5.1: Schématické zobrazení sestavy.
Šipky vyjadřují spojení přes USB rozhraní a směr komunikace.

5.1 Robot

Pro návrh a testování jsem měl k dispozici robota Bioloid Premium Kit od firmy Robotis. Celý robot je koncipován jako stavebnice ze servo motorů, plastových dílů, senzorů a řídící elektroniky. Základní balení obsahuje vše potřebné pro postavení humanoida se 16 stupni volnosti. Díky konstrukci z univerzálních dílů je ale možné robota bez větších obtíží modifikovat a přizpůsobit konkrétní situaci. Vzhled původní verze robota je na obrázku (Obrázek 5.1) a upravená varianta pro vývoj vizuomotorické koordinace je na obrázku (Obrázek 6.1).

Pro pohon robota jsou použity modulární servomotory AX-12, které v jednom kompaktním pouzdře združují DC motor, převody z ozubených kol, obvody pro řízení a obvody pro digitální komunikaci po sběrnici Dynamixel. Všechna serva a také další periferie jsou připojena k řídící jednotce CM530, která funguje jako digitální centrum robota. Řídící jednotka v sobě obsahuje vlastní mikrokontrolér a pomocné obvody pro komunikaci s periferiemi a počítačem.
V řídící jednotce je použit 32bit mikrokontrolér ARM Cortex M3 s taktovací frekvenci 72 MHz a s flash pamětí o velikosti 500 Kbytes pro uložení programu a dat. Programy lze psát v jazyce embedded C a díky přednahranému bootloaderu je lze snadno do mikrokontroléru nahrát přes USB rozhraní.
Ačkoli je možné vytvářet vlastní programy přímo pro probota, tak jsem této možnosti nevyužil a ponechal jsem v řídící jednotce výchozí program, který funguje jako převodník mezi sběrnicí USB a sběrnicí Dynamixel.

5.2 Počítač

Celá architektura vizuomotorické koordinace je naimplementovaná v programovém prostředí MATLAB, které běží na počítači s operačním systémem Windows. Jiný systém není momentálně možné využít, kvůli použité funkci pro získávání obrazu z kamery. Jedná se o funkci VCAPG2 [5], která pro zachytávání obrazu využívá DirectShow.
Program využívá funkcí dvou toolboxů, které nejsou standardní součástí MATLABu. Jedná se o SOM Toolbox [9], který poskytuje funkce pro práci se samoorganizujícími mapami a o Bioloid Toolbox [10], který zajišťuje komunikaci s robotem.

6 Kinematika robota

Ve fázy učení se robot snaží umísťovat stínítko před kameru tak, aby jím zastíňoval předkládaný objek. Stínítko, které robot drží v ruce, je možné, v rámci pracovního prostoru, umístit do libovolné pozice před robota a tedy i do libovolné vzdálenosti od kamery. Změnou této vzdálenosti stínítko mění velikost zakrývané oblasti a při přílišném přiblížení zakryje celé zorné pole kamery. Do tohoto stavu, kdy kamera vidí pouze stínítko, by se robot při učení mohl snadno dostat a učení mapy by bylo dále bezpředmětné, protože by robot již nemusel stínítkem vůbec pohybovat a přesto by byl objekt vždy zastíněný. Aby se robot do tohoto kritické stavu nedostal, je nutné zajistit, aby si stínítko vždy udrželo alespoň určitou minimální vzdálenost, a nebo jestě lépe, aby si udrželo vzdálenost konstantní.

Pro popis pozice stínítka jsem zavedl světový kartézský souřadnicový systém, který má počátek umístěný ve středu těla robota s osami natočenými tak, že první osa směřuje vzhůru, druhá před robota a třetí tak, aby vznikl pravotočivý souřadný systém (viz. Obrázek 6.1).

Pohyb stínítka je zajišťován robotovou rukou tvořenou třemi klouby, které lze na základě morfologické podobnosti s lidským tělem, pojmenovat rameno, loket a zápěstí. Každý z kloubů má svojí vnitřní proměnnou nazývanou kloubová souřadnice, která určuje vzájemné natočení dvou sousedních částí ruky. Pozice ruky a tedy i stínítka je jednoznačně dána těmito třemi kloubovými souřadnicemi.

S pozicí stínítka se pracuje ve světovém souřadnicovém systému, ale ovládání ruky a nastavování pozice stínítka probíhá v prostoru kloubových souřadnic. Pro práci je tedy nutné znát zobrazení, které promítne pozici vyjádřenou ve světových souřadnicích do souřadnic kloubových. Potřebné zobrazení se nazývá inverzní kinematická úloha a získá se vyřešením inverzní kinematiky ruky. Taktéž existuje zobrazení inverzní nazývané přímá kinematické úloha, které promítá kloubové souřadnice do souřadnic světových.

6.1 Dopředná kinematická úloha

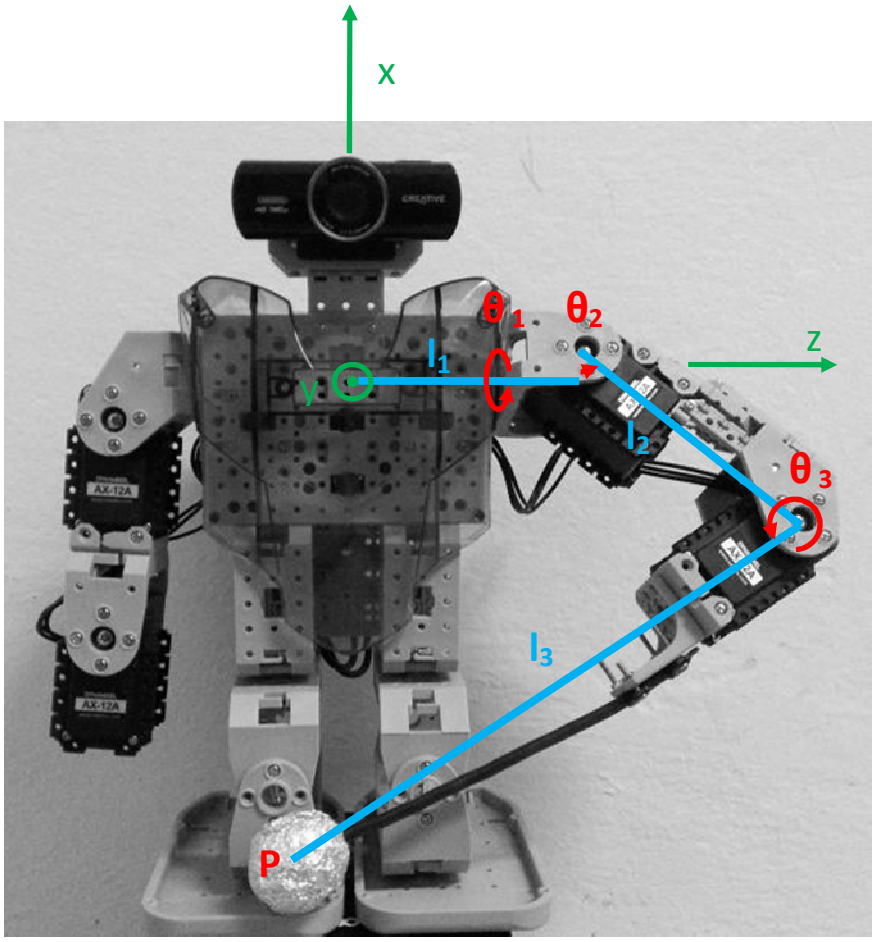
Při řešení dopředné kinematické úlohy jsem využil znalost rozměrů a vzajemných propojení jednotlivých částí robotovi ruky a za pomoci analytické geometrie jsem vytvořil soustavu rovnic, které popisují transformaci ze souřadnic kloubových do souřadnic světových.

Klouby loktu a zápěstí mají rovnoběžné osy rotace a díky tomu je řešení možné rozdělit na dvě fáze. V první fázy je s využitím úhlů θ_2 a θ_3 vypočtena pomocná proměnná d_1 jako vzdálenost stínítka od osy z a druhá pomocná proměnná d_2 jako vzdálenost stínítka od roviny tvořenou osami x a y.

$$d_1 = l_2 \cos(\theta_2) + l_3 \cos(\theta_2 + \theta_3)$$
$$d_2 = l_2 \sin(\theta_2) + l_3 \sin(\theta_2 + \theta_3)$$

Ve fázy druhé je využita vzdálenost d_1 a úhel θ_1 k získání souřadnic x a y a vzdálenost d_2 k získání souřadnice z.

$$x = d_1 \sin(\theta_1)$$
$$y = d_1 \cos(\theta_1)$$
$$z = d_2 + l_1$$



Obrázek 6.1: Popis robota pro dopřednou kinematiku.

Pomocí rovnic lze ze znalosti kloubových souřadnic $\boldsymbol{q}=[\theta_1, \theta_2, \theta_3]$ vypočítat pozici stínítka $\boldsymbol{P}=[x, y, z]$ ve světovém souřadnicovém systému.

6.2 Inverzní kinematická úloha

Řešení inverzní kinematiky je podobné jako řešení kinematiky dopředné. Taktéž jsem využil známé rozměry ruky a analytickou geometrii pro nalezení transformačních rovnic. Řešení je rovněž rozdělené na dvě fáze.

Požadovaná pozice stínítka $\boldsymbol{P}$ je známá a také je známá pozice loketního kloubu, v obrázku (Obrázek 6.2) označeném jako B. Z této znalosti lze dopočítat pozici zápěstí A jako průnik dvou kružnic se středy v B a $\boldsymbol{P}$ a s poloměry l_2 respektive l_3. Obecně má výpočet průniku dvou kružnic jedno, dvě nebo žádné řešení a tomu odpovídá i počet

řešení inverzní kinematické úlohy. V případě, že jsou kružnice od sebe natolik vzdálené, že průnik neexistuje, nemá úloha žádné řešení a požadovaná pozice stínítka je nedosažitelná. Je li vzdálenost středů kružnic rovna součtu jejich poloměrů, pak se kružnice dotknou v jednom bodě a úloha má jedno řešení. Je li vzdálenost menší než součet poloměrů, pak existují dva body průniku a úloha má dvě řešení. V tomto případě je ale nutné brát v úvahu rotační rozsahy kloubů ruky a použitelné řešení může být jen jedno, protože druhé řešení by vyžadovalo nedosažitelné natočení kloubů.
Pro výpočet jsem zavedl dvě nové proměnné r a s, které zjednodušší zápis analytického řešení průniku dvou kružnic.

$$r = \sqrt{x^2 + y^2}$$
$$s = z - l_1$$

Jelikož bude stínítku umísťováno do prostoru před kameru, bude téměř vždy jedno řešení průniku kružnic vycházet do míst, kde má robot tělo a toto řešení tedy bude nepoužitelné. Proto při výpočtu inverzní kinematiky uvažuji pouze jedno řešení $\boldsymbol{A_1}=[f, g]$, uvedené v následujících rovnicích.

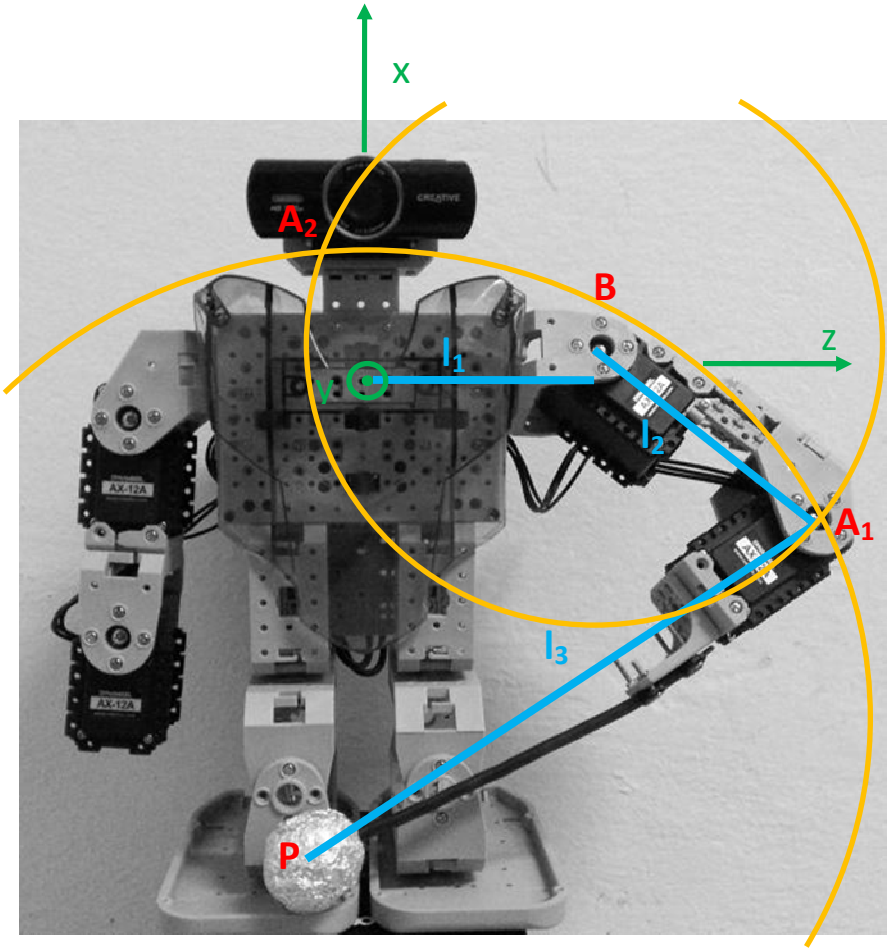
$$Q = r \cdot \sqrt{\left(-r^2 - s^2 + {l_2}^2 + 2 \cdot l_2 \cdot l_3 + {l_3}^2\right) \cdot \left(r^2 + s^2 - {l_2}^2 + 2 \cdot l_2 \cdot l_3 - {l_3}^2\right)} + r^2 \cdot s + s \cdot {l_2}^2 - s \cdot {l_3}^2 + s^3$$

$$f = \frac{r^2 + s^2 + {l_2}^2 - {l_3}^2 - \frac{s \cdot Q}{r^2 + s^2}}{2 \cdot r}$$

$$g = \frac{Q}{2 \cdot (s^2 + r^2)}$$

Ze znalosti pozice kloubu zápěstí $\boldsymbol{A_1}$ lze již dopočítat kloubové souřadnice $\boldsymbol{q}=[\theta_1, \theta_2, \theta_3]$.

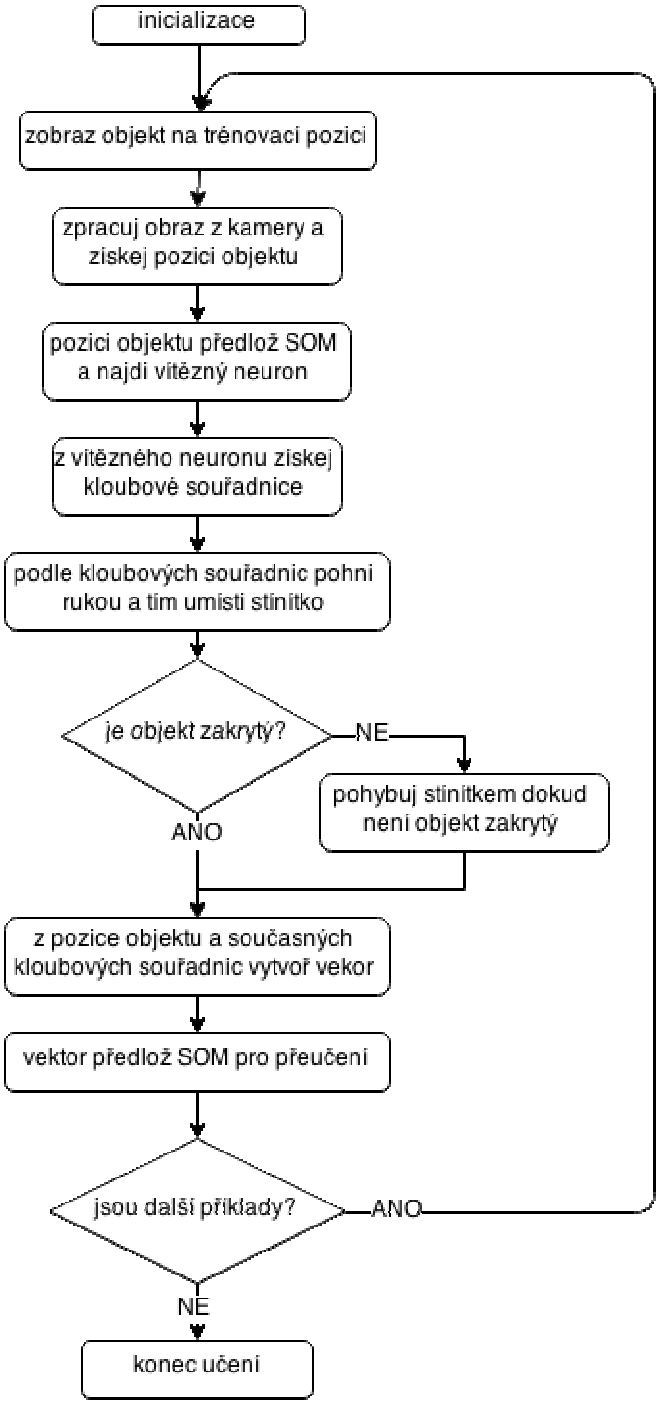
$$\theta_1 = \operatorname{atan2}(x, y)$$
$$\theta_2 = \operatorname{atan2}(-f, g)$$
$$\theta_3 = \operatorname{atan2}(-r + f, s - g)$$



Obrázek 6.2: Popis robota pro inverzní kinematiku.

7 Algoritmus vizuomotorické koordinace

Na začátku programu jsou definovány veškeré potřebné parametry a jsou inicializovány datové struktury. Následně probíhá iterativně proces učení, ve kterém se samoorganizující mapa učí vizuomotorické mapování. V následujících podkapitolách je algorimus učení popsán podrobněji a v obrázku (Obrázek 7.1) je algoritmus pro přehlednost zobrazen v diagramu.



Obrázek 7.1: Diagram algoritmu vizuomotorické koordinace.

7.1 Inicializace

Na základě definovaných parametrů jsou při inicializaci vytvořeny proměnné a datové struktury potřebné pro následný proces učení. Jsou zde definovány parametry samoorganizující mapy, pracovního prostoru ruky, trénovacích dat a počet trénovacích iterací.

Mezi nejdůležitější patří parametry samoorganizující mapy, protože mají zásadní vliv na výsledek učení a při nevhodném nastavení se mapa nemusí být schopna korektně naučit vizuomotorické mapování. Mezi parametry mapy patří velikost a topologie mapy, výchozí hodnoty vah neuronů, rychlost učení a poloměr učení.

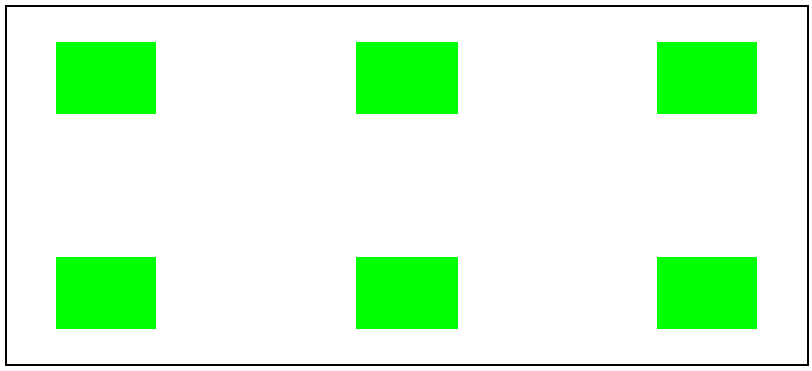
Pracovní prostor ruky je omezen tak, aby se stínítko mohlo pohybovat jen v zorném poli robota a v přibližně konstantní vzdálenosti od kamery. Díky tomuto omezení je zamezeno, aby se robot rukou nebo stínítkem střetl s jinou částí svého těla.

7.2 Trénovací příklady

Jak již bylo zmíněno dříve, trénování probíha tak, že se robot snaží stínítkem zakrývat objekt který se před ním objevuje na různých pozicích. Objekt je detekován na základě jeho výrazné barvy, která se jinde v zorném poli neobjevuje. Díky detekci pouze barvy, nemusí být pro trénování použit skutečný předmět, ale stačí na displeji monitoru zobrazit jen jeho virtuální obraz a detekční algoritmus bude i přesto dávat stejné výsledky. Zobrazování objektu na monitoru umožní výrazně zjednodušit proces učení, protože není nutné objekt po každe iteraci fyzicky přemísťovat, ale stačit jen změnit pozici, kde se objekt na monitoru zobrazí.

Jako objekt k detekci jsem použil zelený list papíru s rozměry 4x5 cm, který má výraznou barvu a je možné ho v obraze snadno rozlišit. Jeho virtuální obraz je zelený obdélník.

Samoroganizující mapa má velkou schopnost zobecňovat a díky tomu není nutné mít trénovací data, která by pokryla celý vstupní prostor, ale stačí trénovat jen na určitých reprezentativních příkladech. Pro trénování jsem definoval šest trénovacích pozic, které jsou přibližně rovnoměrně rozmístěné v zorném poli robota (Obrázek 7.2).



Obrázek 7.2: Trénovací pozice.

7.3 Zpracování obrazových dat

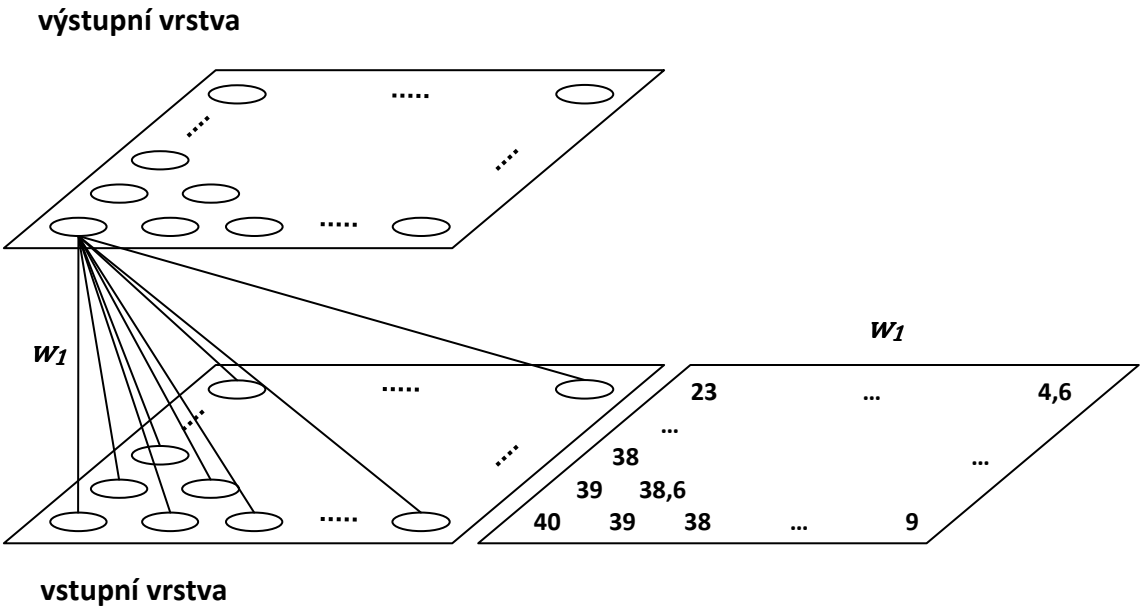
Obraz z kamery je do Matlabu přijímán v HD rozlišení, tedy o velikosti 720x1280 pixelů. Toto rozlišení je ale pro další práci nadbytečné vysoké a proto je nejprve obraz zmenšen na rozlišení 32x18 pixelů. Následně je obraz segmentován na objekt a pozadí a je vytvořen binární obraz, kde pixel s hodnotou 1 je objekt a pixel s hodnotou 0 je pozadí. Segmentace je prováděna na základě RGB složek jednotlivých pixelů $p_i=[R_i, G_i, B_i]$ a definovaného thresholdu Θ. Do dalších fází zpracování pokračuje binární obraz.

$$v_i = 2 \cdot G_i - R_i - B_i$$

Když $v_i > \Theta$ je pixel označen jako objekt, jinak je pixel označen jako pozadí.

Výpočet pozice objektu je prováděn pomocí dvou statických neuronových sítí. V první síti je detekováno těžiště obrázku, tedy střed objektu. A v druhé síti jsou spočteny dvourozměrné souřadnice středu objektu. Tyto sítě jsou nazvané statické, protože váhy neuronů jsou předem definované a v průběhu učení se nemění. Pro výpočet pozice objektu by bylo možné využít i jiné metody, ale jelikož je celý proces učení vytvářen aby využíval biologicky inspirované technologie, jsou použity modifikované neuronové sítě.

Sít pro detekci těžiště obrázku má vstupní i výstupní vrstvu o rozměrech 18x32 neuronů a každý neuron ve výstupní vrstvě je propojen s každým neuronem ve vstupní vrstvě. Díky stejným rozměrům lze obě sítě zarovnat tak, že žádný neuron nebude přesahovat a každý neuron ve vstupní vrstvě bude zarovnán s jedním neuronem ve výstupní vrstvě. Váhy neuronů jsou nastaveny tak, že nejsilnější vazbu má neuron s jeho zarovnaným protějškem a se zvětšující se vzdálenosti mezi neurony se síla vazby lineárně snižuje. Schématické znázornění sítě je v obrázku (Obrázek 7.3). Vazby jsou pro přehlednost znázorněny pouze pro první neuron výstupní vrstvy a pro tento neuron jsou také vypsány hodnoty jeho vah w_1.



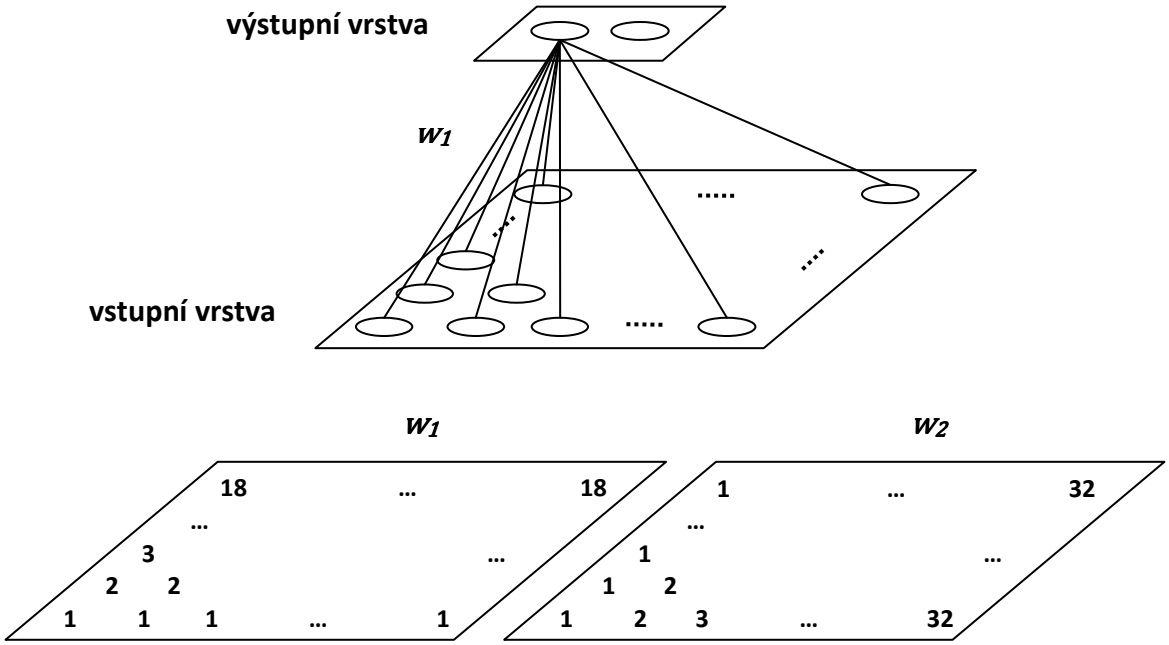
Obrázek 7.3: Statická neuronová síť pro detekci těžistě.

Síť funguje tak, že po předložení vstupního obrazu $\boldsymbol{x}$ jsou vypočteny hodnoty neuronů ve výstupní vrstvě. Neuron i^* s nejsilnější odezvou si svou hodnotu na výstupu ponechá a všem ostatním je nastavena hodnota nula.

$$i^* = \underset{i}{\operatorname{argmax}} \sum_{j=1}^{576} w_{ij} \cdot x_j$$

Výstupem této sítě je dvourozměrné pole, které má nenulovou hodnotu pouze na pozici, kde se v předloženém obraze nachází střed objektu. Velikost této hodnoty vyjadřuje také velikost objektu. Čím větší je hodnota, tím je i objekt v obraze větší.

Síť pro výpočet souřadnic má vstupní vrstvu o velikosti 18x32 neuronů a výstupní vrstvu se dvěma neurony. Každý neuron ve výstupní vrstvě je propojen s každým neuronem ve vstupní vrstvě. Váhy propojení prvního neuronu ve výstupní vrstvě, které vedou do neuronů v prvním řádku vstupní vrstvy mají hodnotu jedna. Propojení, která vedou do druhého řádku mají hodnotu dva a takto obdobně až do posledního řádku, pro který mají propojení hodnotu osmnáct. Druhý neuron ve výstupní vrstvě má váhy propojení podobné, ale místo řádků jsou počítány sloupce. Tady propojení do prvního sloupce vstupní vrstvy mají hodnotu jedna a propojení do posledního sloupce mají hodnotu třicetdva.
Schématické znázornění sítě je v obrázku (Obrázek 7.4), včetně hodnot vah $\boldsymbol{w_i}$ obou neuronů.



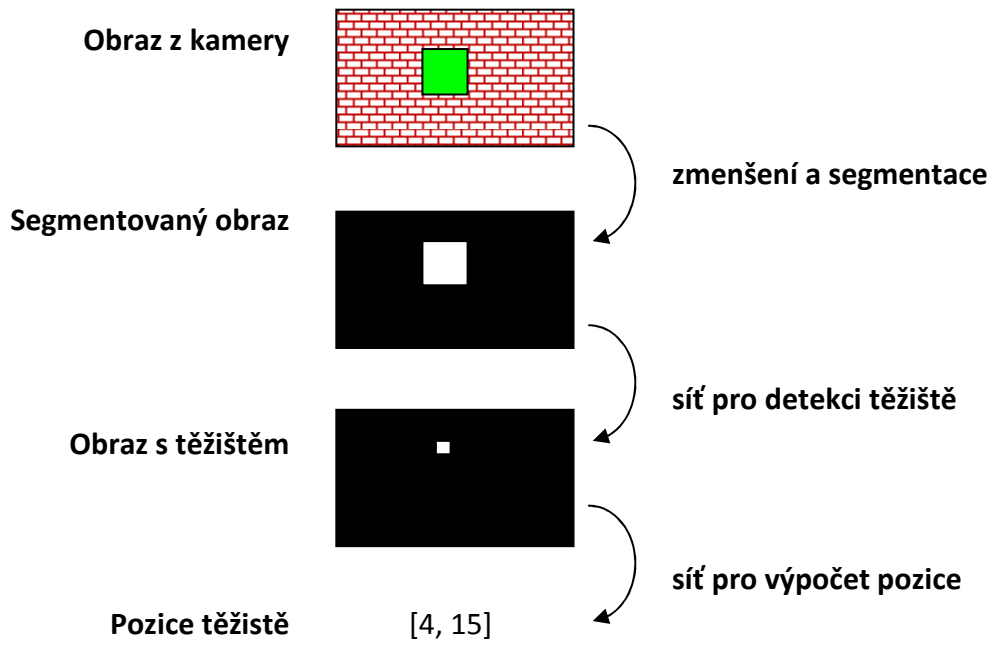
Obrázek 7.4: Statická neuronová síť pro výpočet souřadnic.

Po předložení vstupního vzoru $\boldsymbol{x}$ jsou pro oba výstupní neurony vypočteny hodnoty odezvy y_i, které jsou zároveň i výstupem z této sítě.

$$y_i = \sum_{j=1}^{576} w_{ij} \cdot x_j$$

Do této sítě vstupuje znormované pole získané z předešlé sítě. Pole je znormované tak, že obsahuje nuly a pouze na pozici kde byl detekován střed objektu je jednička.
Výstup z této sítě jsou souřadnice pozice objektu v obraze, které vyjadřují na kolikátém neuronu od kraje je objekt detekován.

Celý postup zpracování obrazu je zobrazen v diagramu (Obrázek 7.5).



Obrázek 7.5: Digram průběhu zpracování obrazu a získání pozice objektu.

7.4 Pohyb ruky

Po zpracování obrazu jsou získané souřadnice y_1 a y_2 předloženy samoorganizující mapě a je nalezen vítězný neuron. Jelikož má mapa pět vstupních neuronů, jsou ke dvěma souřadnicím objektu přidány tři hodnoty "NaN", které vyjadřují, že tyto hodnoty nejsou pro výpočet k dispozici a mapa je interpretuje tak, že při výpočtu vítězného neuronu tyto vstupní neurony ignoruje. Vítězný neuron je tedy počítán jen ze dvou vstupních neuronů. Vektor předkládaný mapě je ve tvaru

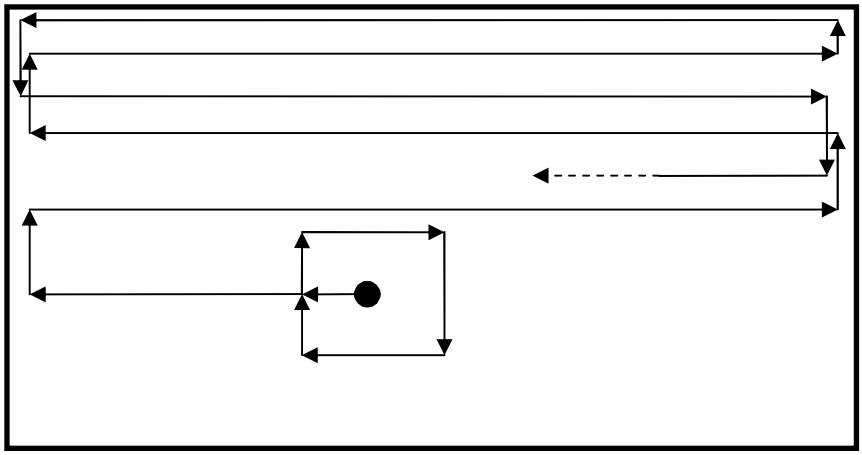
$$\boldsymbol{x} = [y_1, y_2, NaN, NaN, NaN]$$

Z vítězného neuronu je získán vektor jeho vah $\boldsymbol{w^*}$. První dvě souřadnice vektoru odpovídají pozici objektu a zbylé tři souřadnice odpovídají úhlům kloubů, které natočí stínítko tak, aby zakrylo objekt. Při přeučení je mapě předkládán úplný vektor, který obsahuje jak pozici objektu, tak také správné kloubové souřadnice a proto je možné z vítězného neuronu vyextrahovat kloubové souřadnice ačkoli pro určení vítězného neuronu byly použity jen souřadnice objektu.

$$\boldsymbol{w}^* = [x, y, \theta_1, \theta_2, \theta_3]$$

Takto ideální váhy neuronů, které vytvářejí vizuomotorické propojení, získává síť postupně až v průběhu učení. Na začátku jsou váhy všech neuronů inicializovány na stejné hodnoty a uložená pozice objektu tedy neodpovídá správnému natočení kloubů. Z vektoru vah vítězného neuronu jsou použity tři kloubové souřadnice θ_1, θ_2 a θ_3 k pohybu ruky a umístění stínítky do místa, kde by mělo zastínit objekt. V této fázy velmi záleží, jak dobře je mapa naučená. V prvních iteracích učení je stínítko umísťováno zcela mimo, ale postupně, jak se mapa učí vizuomotorické mapování, je stínítko umísťováno blíže k pozici, kde objekt zastíní. Umísťování stínítka se v průběhu učení zlepšuje díky tomu, že mapa posiluje vazbu mezi pozicí stínítka a správným natočením kloubů a potlačuje vazby pro natočení které objekt nezastíní.
V případě, že je stínítko umístěno do správné pozice a objekt je zakrytý, přejde algoritmus do fáze přeučování mapy. Pokud je ale stínítko mimo, je nutné nalézt jeho správnou pozici, kde objekt zastíní.

Hledání správné pozice probíhá pseudo-náhodným pohybem stínítka v zorném poli kamery tak dlouho, dokud nedojde k zastínění objektu. Celý pohyb probíhá ve dvou fázích. Nejprve je obkroužena současná pozice stínítka, protože je pravděpodobné, že správná pozice je někde poblíž. Tato pravděpodobnost se zvyšuje s tím jak se zlepšuje naučení mapy. Obkroužení má tvar čtverce se středem v současné pozici stínítka a je provedeno algoritmem, který je stále stejný a v průběhu učení se nemění. Poloměr obkroužení má stejnou hodnotu jako krok pohybu stínítka, který je nastaven na 1,3 cm. Pokud stále není správná pozice nalezena, přejde pohyb do druhé fáze. Stínítko započne náhodně pohyb do jednoho ze čtyř směrů (nahoru, doprava, dolů, nebo doleva) a v tomto směru se pohybuje tak dlouho, dokud nenarazí na okraj pracovního prostoru. Následně se kousek posune podel okraje a začne se vracet proti původnímu směru. Při dalších nárazech do okraje se situace identicky opakuje a tím stínítko postupně prochází zorné pole kamery a hledá správnou pozici. Znázornění trajektorie možného pohybu stínítka je na obrázku (Obrázek 7.6).
Tento způsob hledání zajišťuje, že bude prohledáno celé zorné pole a bude nalezena správná pozice. Zároveň je do vyhledávání zavedena náhodná složka tím, že může pohyb probíhat vertikálně shora dolů nebo zdola nahoru a nebo horizontálně zleva doprava nebo zprava doleva. Oproti zcela náhodnému pohybu má výhodu v rychlosti nalezení správné pozice, protože plocha zorného pole je prohledávána přibližně rovnoměrně.



Obrázek 7.6: Ukázka možné trajektorie stínítka v zorném poli kamery.

Během hledání správné pozice je také zpracováván obraz z kamery, aby bylo možné detekovat, kdy dojde k zastínění objektu. Zpracování obrazu je podobné jak bylo popsáno v předchozí kapitole. Nejdříve je obraz zmenšen a segmentován na objekt a pozadí. Následně je segmentovaný obraz předložen neuronové síti pro detekci těžiště, jejíž výstup obsahuje informaci o tom, zda se objekt v obraze nachází, a nebo nenachází. Pokud se objekt v obraze nenachází, pak musí být zakrytý stínítkem a správná pozice stínítka byla právě nalezena.

7.5 Přeučení mapy

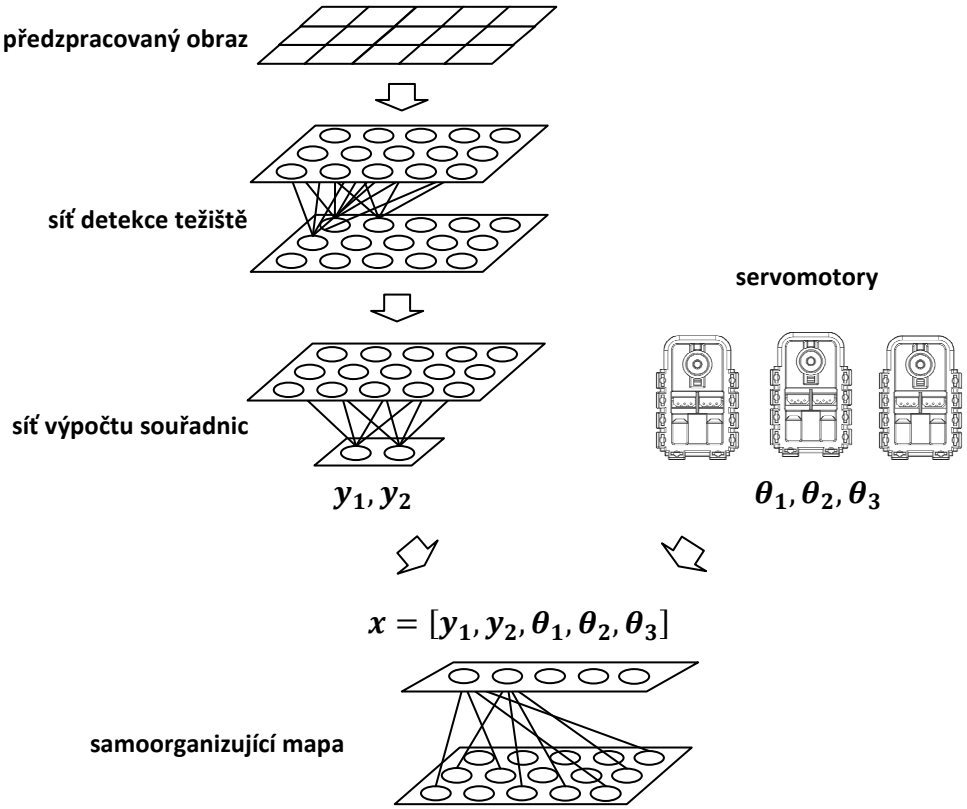
Je-li stínítko umístěné tak, že zakrývá objekt, může algoritmus přejít do fáze přeučování samoorganizující mapy.

Vektor $\boldsymbol{x}$ pro přeučení mapy se skládá z pozice objektu v obraze a kloubových souřadnic ruky.

$$\boldsymbol{x} = [y_1, y_2, \theta_1, \theta_2, \theta_3]$$

Pozice objektu v obraze $[y_1, y_2]$ je již známá z předchozích fází, jelikož se objekt v průběhu jedné iterace nepohybuje. Kloubové souřadnice $[\theta_1, \theta_2, \theta_3]$ jsou takové, na které jsou servomory právě natočeny, protože objekt je momentálně stínítkem zakrytý. Schématické znázornění vzniku vektoru pro přeučení je zachyceno v obrázku (Obrázek 7.7).

Vytvořeným vektorem je mapa přeučena a tím je posílen vztah mezi pozicí objektu a kloubovými souřadnicemi které má ruka když stínítkem objekt zakrývá.
Samotný algoritmus pro přeučování je součástí SOM Toolboxu a v mém programu pouze volám funkci s příslušnými parametry.



Obrázek 7.7: Schématické znázornění hierarchie neuronových sítí a proces vytvoření vektoru pro přeučení mapy.

7.6 Testování

Pro testování naučené mapy a její schopnosti ovládat pohyby robotovi ruky jsem naimplmementoval dva testovací algoritmy, které jsou modifikovanými verzemi učícího algoritmu.

První testovací algoritmus využívá virtuální objekt zobrazovaný na monitoru a je schopný získat přesnou úspěšnost zakrývání objektu.
Testování probíhá iterativně. V každé iteraci je zobrazen objekt na náhodně vygenerované pozici v zorném poli robota. Následně je získán a zpracován obraz z kamery a je vypočtena pozice objektu. Pozice je předložena mapě, je nalezen vítězný neuron a z jeho vah jsou získány kloubové souřadnice, které jsou použity k pohybu ruky. Po dokončení pohybu je opět získán a zpracován obraz z kamery a je zjištěno, zda je objekt zastíněný a nebo nezastíněný. Pokud je objekt zastíněný, jedná se o úspěšný případ vizuomotorického mapování.
Na konci testování je vypočtena celková úspěšnost jako podíl uspěšných případů k celkovému počtu iterací.

V druhém testovacím algoritmu je místo virtuálního objektu využit objekt skutečný a o umísťování a pohyb s objektem se tedy musí postarat uživatel, který robota testuje. Úspěšnost se zde hodnotí pouze empiricky pozorováním pohybů roboty ruky.
Pro tento testovací algoritmus není definovaný počet iterací a algoritmus běží v nekonečné smyčce dokud není program ukončen. Průběh není řízený vnějšími událostmi a algoritmus provádí jeden cyklus za druhým. Rychlost s jakou jednotlivé cykly probíhají záleží především na výkonu počítače a pohybuje řádově v jednotkách až desítkách cyklů za sekundu.
V každém cyklu je nejprve získán a zpracován obraz z kamery. Pokud není v obraze detekován objekt, tak je objekt buď zakrytý a nebo se v zorném poli vůbec nevyskytuje. V takovém případě není potřeba cokoli dělat a algoritmus skočí na začátek nového cyklu. Pokud je objekt v obraze detekován, tak je vypočtena jeho pozice. Ta je předložena mapě, je získán vítězný neuron a z něho kloubové souřadnice, které jsou použity k pohybu ruky. Po skončení pokybu ruky je konce cyklu a algoritmus skočí na začátek dalšího cyklu.

8 Popis implementace

Celá programová část systému vizuomotorické koordinace je rozdělená na tři skripty a jedenáct funkcí. Základní komponentou je skript **VMC_learning**, který obsahuje příkazy pro navázání komunikace s robotem, inicializaci datových struktur a v neposlední řadě také naučení samoorganizující mapy. Další důležitou komponentou jsou skripty **VMC_testing_PC** a **VMC_testing_real**, které provádějí testování kvality naučené mapy pomocí virtuálního respektive skutečného objektu. Funkce jsou volány ze scriptů a slouží převážně pro zpřehlednění celého programu a pro snazší ladění a další úpravy.
V následujícím textu je soupis všech částí programu. U skriptů jsou uvedeny klíčové části zdrojového kódu včetně podrobného popisu, aby se případní další uživatelé dokázali v programu snadněji zorientovat a přispůsobit ho vlastním potřebám. U funkcí jsou popsány jejich vstupní parametry, výstupní hodnoty a činnost jakou funkce vykonává.

VMC_learning
Celý skript je rozdělen na pět částí. Každá část je tvořena jednou buňkou (část kódu mezi **%%**) a je spustitelná samostatně, aniž by bylo nutné spouštět celý script. Spouštění po částech je velmi důležité, protože některé části je nutné spustit pouze jednou na začátku práce s robotem, zatímco jiné jsou určené pro opakované spouštění.

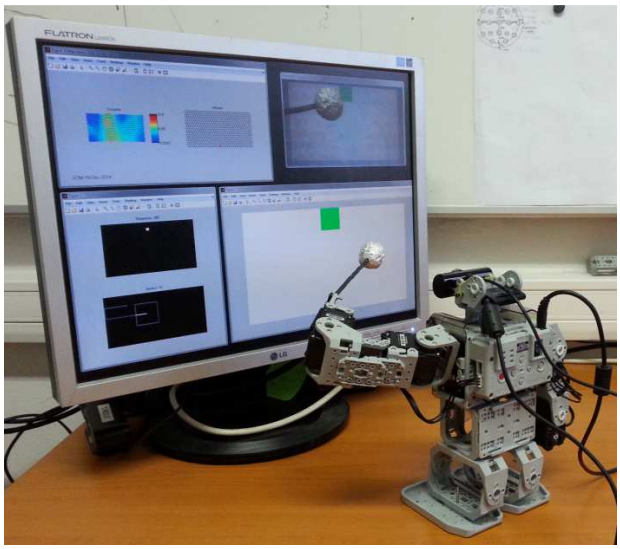
První část je navázání spojení s robotem a kamerou a uvedení robota do výchozího stavu. Nejprve je do vyhledávacích cest Matlabu přidána složka s Bioloid Toolboxem. Cesta se bude lišit v závislosti na skutečném umístění na konkrétním počítači. Následně je nastaven a otevřen virtuální seriový port, přes který probíhá komunikace s robotem. Parametrem funkce setSerialPort() je číslo COM portu, které bylo robotovi přiděleno operačním systémem. Přidělené číslo se mezi počítači může lišit a lze ho najít ve *Správci zařízení* ve skupině *Porty (COM a LPT)*. Proměnná **s** je instance sériového portu a je nezbytná pro fungování Bioloid Toolboxu, jelikož veškerá výměna dat s robotem probíhá právě skrze tuto proměnnou. V případě jejího smazání nebo přepsání nebude možné s robotem dále komunikovat a také nebude možné s robotem navázat nové spojení. Při otevření sériového portu jsou přístupová práva vyhrazena pouze pro instanci, která ho otevřela a pro všechny ostatní je nepřístupný. Proto v případě ztráty proměnné, která na instanci ukazuje, nebude možné port řádně uzavřít a zůstane trvale nepřístupný. Opětovné uvolnění portu lze provést restartováním Matlabu, při kterém je port uzavřen a uvolněn.
Zavoláním funkce **vcapg2()** se otevře okno se živým obrazem z kamery. Parametrem funkce je číslo kamery, ze které požadujeme brát obraz. V případě, že je k počítači připojeno více kamer, tak se parametr může lišit.

Následně jsou nastaveny parametry servomotorů v robotově ruce a ruka je umístěna do výchozí pozice, tak aby stínítko bylo ve středu zorného pole kamery. Kamera není k robotovi přidělána napevno a má určitou volnost v pohybu. Proto je vhodné pozici kamery upravit tak, aby se stínítko dostalo přibližně do středu zorného pole.

```
addpath('C:\Users\Martin\DIP\Bioloid_Toolbox')
s=setSerialPort(16);
fopen(s);
vcapg2(0);
initRobot(s);
moveHand(s,55,160,0);
```

Druhou částí je příprava oken pro zobrazování obrázků a grafů. Po spustění se otevřou tři okna. Dvě prázdné a jedno s obrázkem na kterém je zobrazen virtuální objekt. Právě v tomto okně se bude objekt zobrazovat i při učení a testování a proto je nutné robota umístit tak, aby zorné pole kamery zabíralo akorát celý obrázek. Ve zbylých dvou oknech se začnou zobrazovat údaje až začne proces učení. Okno označené jako **Figure 2** bude horizontálně rozdělené na dvě části. V horní části bude zobrazován graf závislosti chyby mapy na uběhlých iteracích. Chyba je počítána jako vzdálenost mezi pozicí stínítka, do které bylo umístěné mapou a pozicí stínítka ve které objekt zastíní. Ve spodní části okna bude zobrazován obrázek trajektorie, po které se stínítko pohybovalo při hledání správné pozice. V okně označené jako **Figure 3** bude zobrazována u-matrix učené mapy a zvýrazněný vítězný neuron. Možné rozmístění oken a robota je vidět na obrázku (Obrázek 8.1).

```
[ img, label, position ] = createPicture([150 150],[2 3],0,1);
figure(1);
h1=imshow(img);
figure(2)
figure(3)
```



Obrázek 8.1: Pohled na robota a trénovací prostředí.

Ve třetí části jsou inicializovány parametry a vytvořeny proměnné a další datové struktury. V následujícím textu jsou popsány pouze parametry, které jsou určené k nastavování na uživatelské úrovni.

Učení probíhá po trénovacích sériích. V každé sérii je objekt zobrazen v náhodném pořadí právě jednou na každé ze šesti trénovacích pozicích. Parametr **N** udává počet těchto sérii (celkový počet trénovacích iterací je **N**·6).

Parametr **siz** je pole se dvěma hodnotami, které udávají rozlišení zpracovaného obrazu, velikost statických neuronových sítí a velikost SOM.

Parametrem **response_threshold** se nastavuje tolerance při určování, zda je objekt zakrytý nebo nezakrytý. Nulová hodnota znamená, že objekt musí být zcela zakrytý. Se zvyšující se hodnotou stoupá i tolerance k nedokonalému zakrytí a objekt může zpoza stínítka částečně vykukovat a i přesto zůstane označen jako zakrytý. Hodnota 100 odpovídá přibližne 90% zakrytí.

Struktura **hand** združuje parametry robotovi ruky do jedné proměnné. Struktura slouží především pro plánování pohybů ruky při vyhledávání pozice, kde stínítko zastíní objekt. Prvky **constraints** a **z** vymezují pracovní prostor, ve kterém se stínítko může pohybovat. První sloupec v **constraints** definuje vertikální rozsah, druhý sloupec horizontální rozsah a **z** definuje vzdálenost od kamery. Prvek **step** udává velikost kroku při vyhledávání pozice. Zpracování obrazu a pohyb ruky neprobíhá paralelně, ale po fázích kde vždy po malém pohybu následuje zpracování obrazu. Hodnota **step** je velikost tohoto malého pohybu. Prvek **constraints_angles** omezuje rozsahy kloubů, tak aby nedošlo ke kolizi ruky s robotovým tělem a nebo okolním vybavením. Každý řádek definuje rozsah pro jeden kloub v pořadí rameno, loket, zápěstí. Hodnoty v prvcích **contraints**, **z** a **step** jsou měřeny v milimetrech.

Struktura **som** združuje parametry samoorganizující mapy. Prvek **size** definuje rozměr výstupní vrstvy mapy jako počet neuronů v řádcích a ve sloupcích. Topologii mapy definuje prvek **shape**. Na výběr jsou tři možnosti ('sheet', 'cyl', 'toroid'), které se liší

ve způsobu zakončení krajních neuronů. 'Sheet' má krajní neurony volně, 'cyl' má v jednom rozměru krajní neurony spojené a vytváří tak válec, 'toroid' má krajní neurony spojené v obou rozměrech a vytváří tak toroid. Prvek **alpha** je vektor, který udává míru úpravi vah (rychlost učení) neuronů v průběhu učení mapy a prvek **radius** je vektor, který udává poloměr okolí kolem vítězného neuronu. Oba vektory mají velikost stejnou, jako je počet trénovacích iterací a pro každou iteraci tak připadá vždy jedna hodnota.

```
N=10;
siz=[18 32];
response_threshold=100;

hand.constraints=[25 60; 85 -60];
hand.z=160;
hand.step=13;
hand.constraints_angles=[480 410; 530 770; 770 900];

som.size=siz;
som.shape='sheet';
som.alpha=linspace(1,0.1,num);
som.radius=linspace(10,1,num);
```

Čtvrtá část scriptu je již samotné učení, které probíhá tak, jak je popsané v kapitole 7 Algoritmus vizuomotorické koordinace.

Pátá část ukládá na disk počítače data získaná v průběhu učení. Z každé iterace je získán jedna sada údajů a ukládaná data tak mají stejnou velikost jako je počet trénovacích iterací. Vektor **distances** obsahuje vzdálenosti mezi pozicí stínítka získanou z mapy a správnou pozicí. Jedná se o stejné hodnoty jaké jsou zobrazovány v grafu ve **Figure 2**. Vektor **times** je záznam, jak dlouho každá iterace trvala. Vektor **labels** obsahuje označení pozic, na kterých byl objekt zobrazen. Matice **data** má v každém řádku vektor, který byl použit pro přeučení mapy. Parametr **data_num** je číslo, které je přidáno na konec jména ukládaného souboru, aby bylo možné rozlišit mezi údaji z různých trénovacích průběhů.

```
data_num=30;
saveData( data_num, distances, times, labels, data )
```

VMC_testing_PC

Script slouží pro otestování kvality naučené samoorganizující mapy. Aby tento skript fungoval, je nutné nejprve zpustit skript VMC_learning, ve kterém jsou vytvořeny veškeré proměnné, potřebné jak pro trénování tak i pro testování. Jediný parametr, který se zde nastavuje je počet testovacích iterací **num**.

Virtuální objekt je při testování zobrazován do stejného okna jako při učení a proto není potřeba s robotem manipulovat a po proběhnutí učícího skriptu je možné hned spustit skript testovací.

VMC_testing_real

Tento script slouží taktéž pro otestování kvality naučené samoorganizující mapy. Skript nemá žádné parametry a po spuštení skočí do nekonečné smyčky, ve které se robot snaží zakrývat objekt za pomoci naučené SOM. Pro ukončení testování je nutné použít klávesovou kombinaci Ctrl+C, která Matlab donutí ukončit prováděný skript.

V následujícím seznamu jsou uvedeny všechny vytvořené funkce. Pro snadné vyhledávání jsou funkce abecedně seřazeny.

computeCentroidNNResponse

Vstup: code (matice MxNx[M·N]) váhy neuronů statické neuronové sítě
image (matice MxN) obraz pro zpracování

Výstup: response (matice MxN) odezvy všech neuronů sítě
value (skalár) hodnota vítězného neuronu

Popis: Vypočítá odezvu statické neuronové sítě pro detekci těžiště. Vstupní obraz musí mít stejnou velikost jako je velikost sítě ve vstupní vrstvě. Výstup je binární matice s hodnotou 1 na pozici těžiště obrazu. Velikost matice je stejná jako velikost vstupního obrazu.

computeXYNNResponse

Vstup: code (matice MxNx2) váhy neuronů statické neuronové sítě
image (matice MxN) výstupní matice z funkce *computeCentroidNNResponse*

Výstup: response (vektor 1x2) odezva neuronové sítě

Popis: Vypočítá odezvu statické neuronové sítě pro výpočet souřadnic. Vstupní obraz musí mít stejnou velikost jako je velikost sítě ve vstupní vrstvě. Výstup je dvourozměrný vektor se souřadnicemi těžiště.

createNeuronWeights

Vstup: siz (vektor 1x2) požadovaná velikost statických neuronových sítí
r (skalár) poloměr efektu neronů v síti pro detekci těžiště

Výstup: nw_center (matice MxNx[M·N]) váhy neuronů sítě pro detekci těžiště
nw_xy (matice MxNx2) váhy neuronů sítě pro výpočet souřadnic

Popis: Vypočítá hodnoty vah neuronů ve statických neuronových sítích. Požadovaná velikost sítě musí být stejná jako velikost předzpracovaného obrazu.

createPicture

Vstup: obj_size (vektor 1x2) velikost objektu v pixelech
division (vektor 1x2) rozdělení obrazu do mřížky
rotate (boolean) jestli objekt může rotovat
select (skalár) vybraná buňka v mřížce

Výstup: img (matice 720x1280x3) obraz s umístěným objektem
label (string) označení pozice objektu
position (vektor 1x2) pozice objektu v obraze

Popis: Vytvoří obraz s objektem umístěným na zadané pozici. Obraz je rozdělen do mřížky a objekt je vykreslen do vybrané buňky. Vybraná buňka je zadána jako její lineární index tedy pořadí ve sloupcích seřazených za sebou do jednoho vektoru. Pokud je povolena rotace, tak se objekt může otočit o náhodný úhel kolem svého středu. Vykreslený objekt je zelený RGB=(0, 255, 0).

createPictureRandom

Vstup: obj_size (vektor 1x2) velikost objektu v pixelech

Výstup: img (matice 720x1280x3) obraz s umístěným objektem

Popis: Vytvoří obraz s objektem umístěným na náhodné pozici. Vykreslený objekt je zelený RGB=(0, 255, 0).

initRobotvstup:

Vstup: s (serial port) instance sériového portu

Výstup: není

Popis: Nastaví rychlost a volnost servomotorů v robotově ruce. Při odpojení robota od napájení se nastavené parametry vrátí do jejich výchozích hodnot a je nutné je znovu nastavit.

moveHand

Vstup: s (serial port) instance sériového portu
x (skalár) souřadnice cílové pozice - vertikálně
y (skalár) souřadnice cílové pozice - horizontálně
z (skalár) souřadnice cílové pozice - vzdálenost

Výstup: není

Popis: Pro požadovanou cílovou pozici stínítka spočítá inverzní kinematiku a na získané úhly natočí servomotory v robotově ruce.

preprocessImage

Vstup: img_in (matice PxQ) vstupní obraz
size_out (vektor 1x2) požadovaná velikost výstupního obrazu

Výstup: img_out (matice MxN) zpracovaný obraz

Popis: Obraz z kamery předpřipraví pro další zpracování. Obraz zmenší na požadované rozlišení a na základě RGB složek pixelů provede segmentaci na objekt a pozadí.

saveData

Vstup: num (skalár) pořadové číslo ukládaných dat
distances (vektor Mx1) chyba mapy v průběhu učení
times (vektor Mx1) trvání trénovacích iterací
labels (cell Mx1) label trénovacích iterací
data (matice Mx5) data trénovacích iterací

Výstup: není

Popis: Uloží data na disk počítače. Slouží pro ukládání hodnot získaných v průběhu učení.

scanMoveHand

Vstup: q (struktura) struktura udržující parametry

Výstup: q (struktura) upravená struktura s parametry ruky)

Popis: Na základě současného stavu stínítka vypočítá kam se má stínítko umístit při následujícím pohybu.

whereIsHand

Vstup: s (serial port) instance sériového portu

Výstup: x (skalár) souřadnice pozice stínítka - vertikálně
y (skalár) souřadnice pozice stínítka - horizontálně
z (skalár) souřadnice pozice stínítka - vzdálenost

Popis: Získá natočení servomotorů v robotově ruce a pomocí dopředné kinematiky vypočítá umístění stínítka v kartézském souřadnicovém systému.

9 Výsledky

Při závěrečném testování fungování vytvořeného systému vizuomotorické koordinace jsem využil samoorganizující mapu s paramery nastavenými na následující hodnoty:

- Obdélníková topologie s velikostí stran 18x32 neuronů. Naurony jsou uspořádány do šestiúhelníkové mřížky, která je na krajích ukončena a nenavazuje tedy na neurony na protější straně.
- Rychlost učení je lineární funkce, která klesá z počáteční hodnoty 1 v první iteraci učení na konečnou hodnotu 0,1 v poslední iteraci.
- Poloměr učení je taktéž lineární funkce, která klesá z hodnoty 10 na hodnotu 1.
- Jak rychlost učení tak poloměr učení mají stejný průběh bez ohledu na počet trénovacích iterací. Pro menší počet iterací tedy klesají rychleji a pro větší počet klesají pomaleji.
- Výchozí váhy neuronů jsou pro všechny neurony nastaveny na stejné hodnoty.

K těmto hodnotám nastavení jsem se dostal na základě testování a postupného ladění parametrů. Robota jsem nechal učit se s určitou sadou parametrů a sledoval jsem vývoj samoorganizující mapy a pohyby ruky robota. V případě, že se robot dokázal naučil svojí ruku ovládat, tak jsem na závěr jestě spustil testování, abych mohl porovnávat úspěšnosti mezi různými sadami parametrů. Při ladění parametrů jsem se zaměřil jen na rychlost učení, poloměr učení a na tvar sítě (list, válec, cylindr). Ostatní parametry jsem nechal na výchozích hodnotách nastavených SOM Toolboxem a nebo jsem je nastavil na jednu hodnotu, která zůstala konstantní po celou dobu práce s robotem.
Při prvotním odhadu a následném ladění parametrů mapy jsem využil praktické zkušenosti získané při předchozí práci na výukových úlohách demonstrujících využití umělých neuronových sítí v robotice. V jedné z úloh se robot Bioloid pomocí samoorganizující mapy učil ukazovat na objekt, který se předním v jednom rozměru pohybuje. Zkušenosti s nastavováním parametrů v této menší a jednodušší mapě se ukázali být přenosné i na mapu použitou v této diplomové práci.

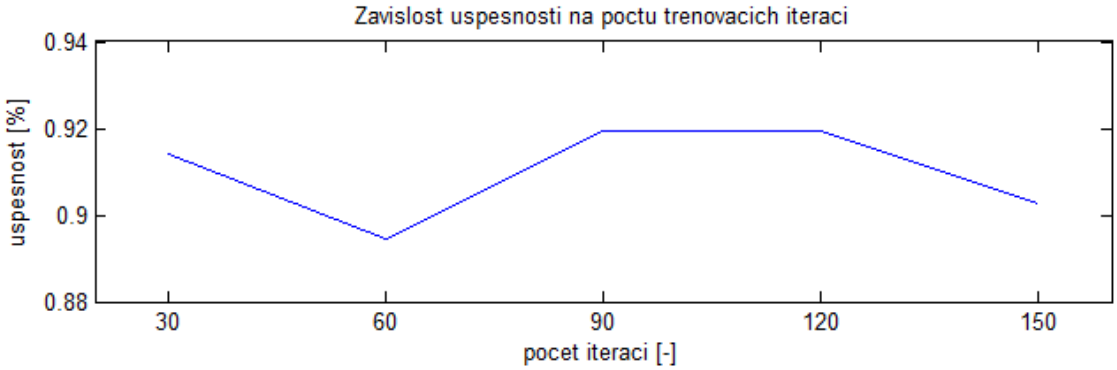
Při ladění parametrů jsem velmi vyžíval u-matrix, která byla zobrazována po každé trénovací iteraci, takže bylo možné v reálném čase sledovat jak se mapa vyvýjí. U-matrix poskytuje sice jen přibližný odhad stavu neuronů uvnitř mapy, ale z průběhu vývoje mapy bylo možné odhadnout výsledek ještě před dokončením všech trénovacích iterací. Pokud v mapě v průběhu učení začalo vznikat šest shluků, které měli přibližně stejné rozestavení jako šest trénovacích pozic, tak to byl indikátor správně nastavených parametrů. Tyto shluky jsou v u-matrix zobrazeny jako tmavě modré oblasti a reprezentují trénovací vzory (vektory), které jsou si podobné. Jelikož trénovací vektory jsou složené z pozice objektu a kloubových souřadnic ruky při kterých je objekt zakrytý, tak pro každou ze šesti trénovacích pozic existují vektory,

které jsou si velmi podobné a to má za důsledek vznik šesti shluků. Pokud v mapě začalo vznikat více shluků nebo měli výrazně odlišné rozestavní oproti trénovacím pozicím, tak bylo možné učení předčasně ukončit a zkusit jiné nastavení parametrů.
Průběh úspěšného vývoje mapy je zachycen na obrázku (Obrázek 9.3). Počet trénovacích iterací byl 120 a parametry mapy byly nastaveny tak, jak je uvedeno v úvodu této kapitoly. Kvůli úspoře místa a větší přehlednosti nejsou zobrazny u-matrix po každé proběhlé iteraci, ale jen po každých patnácti iteracích. V obrázku je patrné, že již přibližně v polovině učení začalo vznikat šest dobře rozestavených shluků, což byl příslib úspěšného naučení vizuomotorického mapování.

Nalezené parametry mapy, které jsem použil pro závěrečné testování, nemusí být nutně optimální. Jelikož jsem ladění prováděl iterativně na základě praktických experimentů, tak je možné, že parametry jsou pouze suboptimální. Nicméně výsledky dosažené při testování jsou velmi povzbuzující a nepředpokládám, že pouhým doladění parametrů by mohla být úspěšnost výrazně navýšena.

V průběhu testování se také ukázalo, že nastavení parametrů rychlost a poloměr učení mapy není kritické. Hodnota obou parametrů může být změněna až o přibližne 20% a mapa bude stále schopná korektně se naučit vizuomotorické mapování. Výsledná úspěšnost se může řádově o několik procent lišit, ale mapa bude stále schopná robota ovládat.

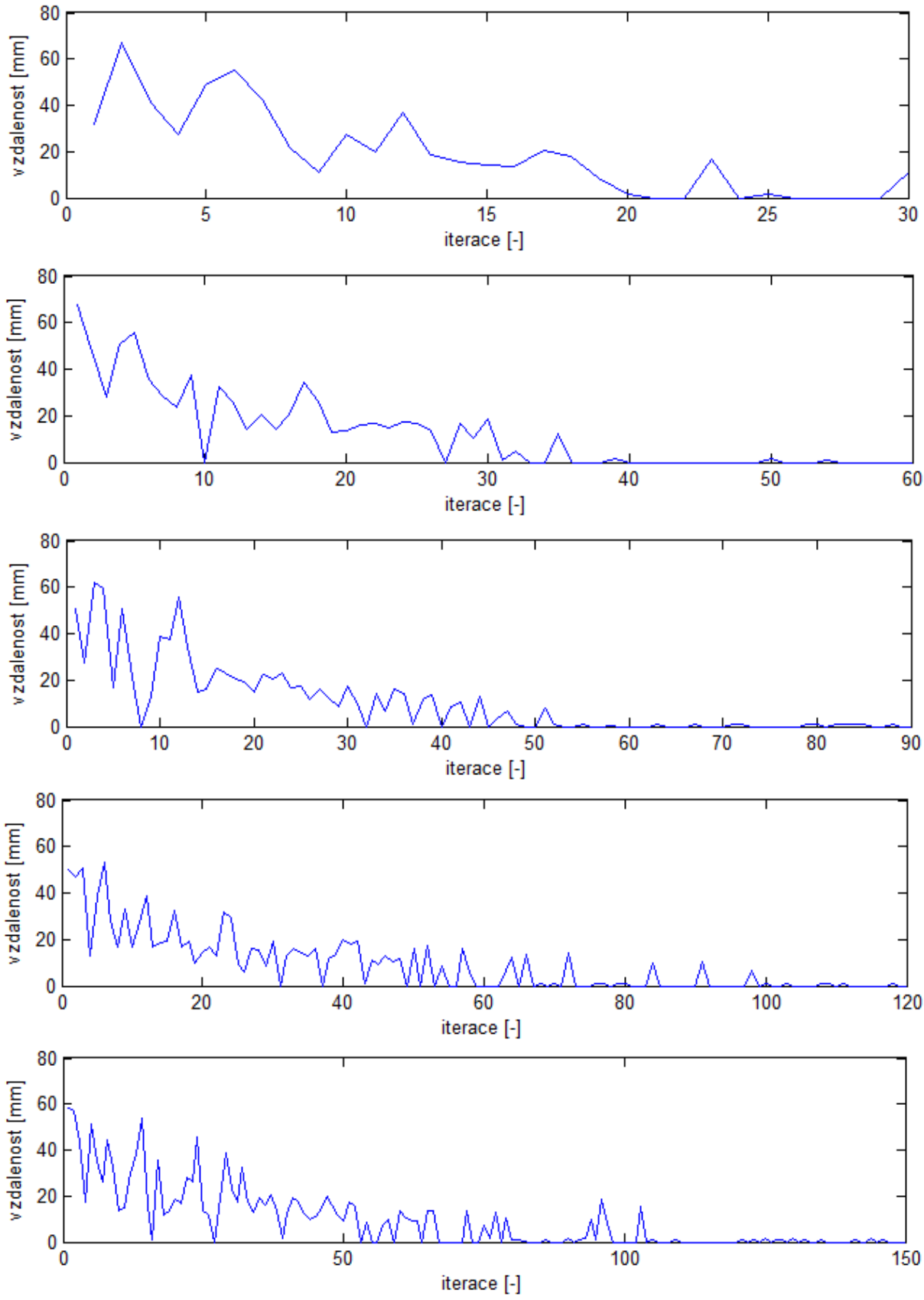
Při závěrečném testování jsem, kromě sledování celkové funkčnosti systému, také provědel průzkum závislosti úspěšnosti zakrývání na počtu trénovacích iterací.
Dohromady jsem provedl pět experimentů s počtem trénovacích iteracích 30, 60, 90, 120 a 150. V každém experimentu jsem nechal samoorganizují mapu naučit se vizuomotorické mapování na daném počtu trénovacích iterací a následně jsem naučenou mapu otestoval na 360 testovacích příkladech. Parametry mapy byly pro všech pět experimentů stejné a výsledné úspěšnosti jsou tedy závislé jen na délce učení.
Získané úspěšnosti zakrývání jsou pro přehlednost uvedeny ve spojnicovém grafu (Obrázek 9.1) a vytváří tak průběh závislosti úspěšnosti na počtu trénovacích iterací. Naměřené výsledky jsou velmi neočekávané, jelikož úspěšnost je přibližně konstantní a nezávisí na délce trénování. V podobných případech strojového učení je obvyklé, že úspěšnost se zlepšuje s tím jak se zvyšuje počet příkladů použitých při učení. V tomto případě má ale robot úspěšnost kolem 91% již po 30 trénovacích iterací a s více iteracemi se úspěšnost výrazně nemění.



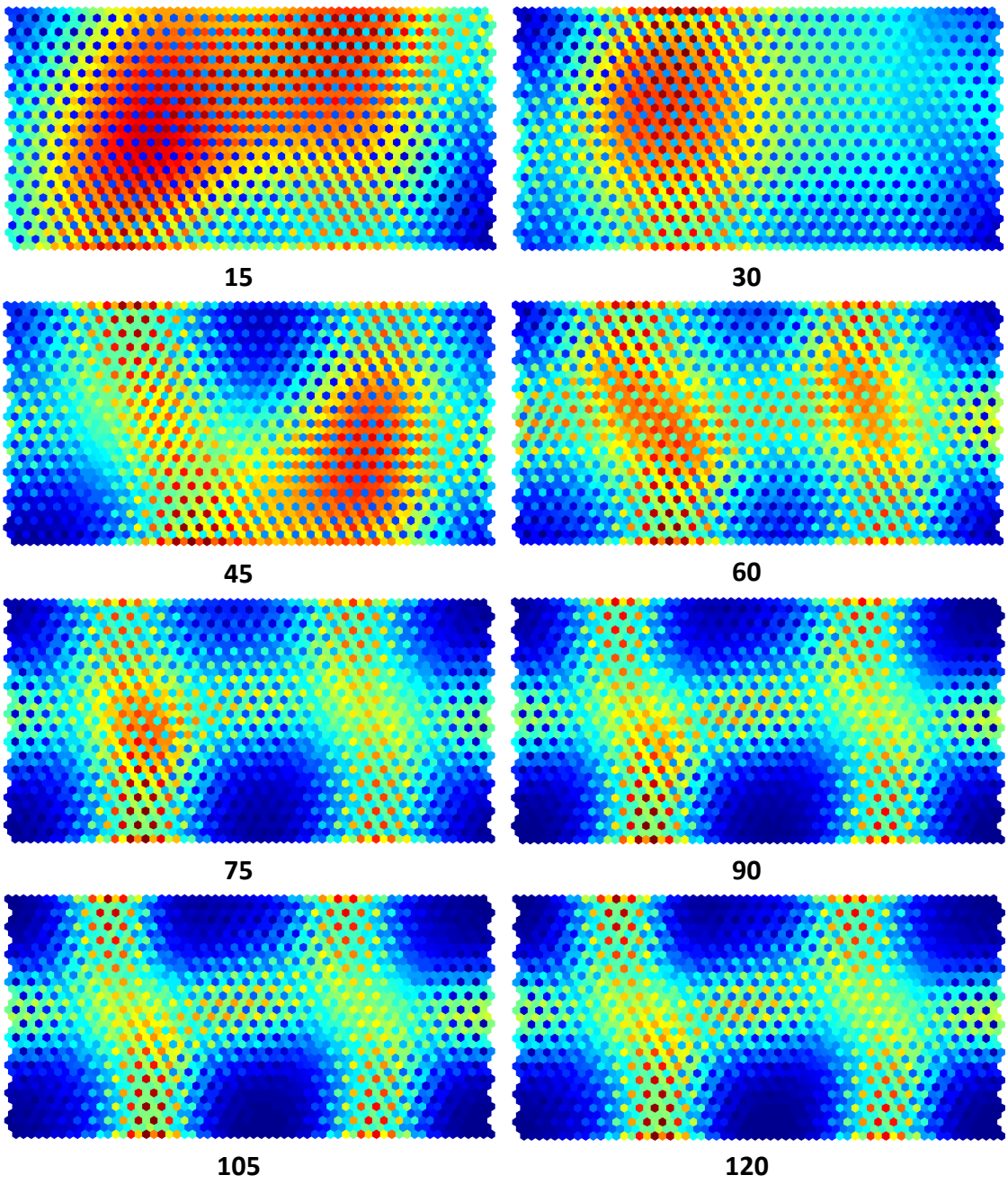
Obrázek 9.1: Závislost úspěšnosti zakrývání objektu na počtu trénovacích iteracích. $\mathbf{30}\sim\mathbf{91},\mathbf{4}\%$, $\mathbf{60}\sim\mathbf{89},\mathbf{4}\%$, $\mathbf{90}\sim\mathbf{91},\mathbf{9}\%$, $\mathbf{120}\sim\mathbf{91},\mathbf{9}\%$, $\mathbf{150}\sim\mathbf{90},\mathbf{3}\%$

Při všech pěti experimentech jsem v průběhu učení mapy také zaznamenával její chybu při umísťování stínítka. Tato chyba je počítána jako vzdálenost mezi pozicí stínítka, do které bylo umístěné mapou a pozicí stínítka ve které objekt zastíní. Zaznamenané průběhy jsou v grafech na obrázku (Obrázek 9.2). Výsledky jsou podle očekávání. Na začátku učení má mapa slabé vazby mezi pozicí objektu a správnými kloubovými souřadnicemi a proto je chyba výrazně větší. Tím jak se mapa učí a vazby posiluje, tak také snižuje chybu v umísťování.

Průběh velikosti chyby má přibližně exponenciální charakter pro všech pět experimentů. Průběh pro 30 trénovacích iterací a částečně i pro 60 vypadají odlišně, což je ale způsobené tím, že jsou zobrazeny „natažené“ na stejnou délku jako ostatní, ačkoli mají výrazně menší počet naměřených hodnot



Obrázek 9.2: Závislost chyby mapy na počtu proběhlých iterací. Pro velikost trénovacích množin v pořadí zhora 30, 60, 90, 120 a 150.



Obrázek 9.3: Vývoj samoorganizující mapy v průběhu učení na 120 trénovacích iteracích. Mapa je zobrazena v podobě u-matrix po každých 15 proběhlých iteracích.

10 Závěr

Vytvořil jsem systém vizuomotorické koordinace pro humanoidního robota Bioloid. Systém je založený na samoorganizující mapě, která vytváří propojení mezi vizuální a motorickou informací. Vizuální informací je pozice objektu, který robot před sebou vidí a motorickou informací jsou kloubové souřadnice robotovi ruky, které umístí robotem držené stínítko do pozice, kde zakryje viděný objekt.
Pro výpočet pozice objektu v obraze je použito segmentování obrazu na objekt a pozadí na základě barevných složek jednotlivých pixelů. Následně jsou pomocí dvou modifikovaných neuronových sití nalezeny souřadnice těžiště segmentovaného obrazu, které také představují pozici viděného objektu.
Kromě programu, který dokáže robota naučit zakrývat stínítkem objekt, jsem také vytvořil programy pro testování, kterými je možné určit kvalitu naučení a úspěšnost při zakrývání.

Systém vizuomotorické kordinace jsem naimplementoval v programovém prostředí MATLAB s pomocí SOM Toolboxu a Bioloid Toolboxu, které poskytují funkce pro pohodlnou práci se samoorganizujícími mapami a robotem Bioloid. Prostředí MATLAB běží na počítači, do kterého jsou připojeny robot a web kamera, která funguje jako robotův zrak.

Úspěšnost robota v zakrývání objektu je již po 30 trénovacích iteracích 91,4%. Zajímavým výsledkem je, že tato úspěšnost se s narůstajícím počtem trénovacích iterací výrazně nemění a po 150 trénovacích iterací má robot úspěšnost 90,3%.
Při testování byly robotovi předkládány objekty na náhodných pozicích a tedy i na pozicích, které se při trénování neobjevili. Tím byla ověřena schopnost mapy generalizovat a úspěšně se vypořádat i s do té doby neznámými vstupy.

Úspěšnost v zakrývání by bylo možné zvýšit několika způsoby. Například použít mapu s více neurony a tím zvýšit rozlišení nastavované pozice stínítka, nebo předkládat robotovi při učení objekt na více než šesti pozicích a tím rovnoměrněji pokrýt vstupní prostor samoorganizující mapy. Také by bylo možné aplikovat adaptivní způsob vybavování mapy. Tedy, že by se robot dokázal v omezené míře přeučovat i v průběhu testování a tím se adaptovat na nové podmínky. Navržená architektura má potenciál pro další růst a zvyšování kvality naučené vizuomotorické koordinace, který ale zatím zůstává jen jako téma pro další práci.

11 Seznam použité literatury a odkazy

[1] *Thomas M. Martinetz & Klaus J Schulten*, Hierarchical Neural Net for Learning Control of a Robot's Arm and Gripper, International Joint Conference on Neural Networks, Vol. 2, pp. 757-752, 1990

[2] *Samantha V. Adams, Thomas Wennekers, Sue Denham, Phil F. Culverhouse*, Adaptive training of cortical feature maps for a robot sensorimotor controller, Neural Networks 44, pp. 6-21, 2013

[3] *Jörg A. Walter and Klaus J. Schulten*, Implementation of Self-Organizing Neural Networks for Visuo-Motor Control of an Industrial Robot, IEEE Transaction on Neural Networks, Vol. 4, No. 1, 1996

[4] *James Bennett Saxon & Amitabha Mukerjee*, Learning the Motion Map of a Robot Arm with Neural Networks, Neural Networks, IJCNN International Joint Conference, Vol. 2, pp. 777-782, 1990

[5] *Craig Sayers*, Self Organizing Feature Maps and their Applications to Robotics, University of Pennsylvania, Technical Reports, 1991

[6] *Thomas Martinetz & Klaus Schulten*, A Neural Network for Robot Control: Cooperation between Neural Units as a Requirement for Learning, Computer Elect. Engng, Vol. 19, No. 4, pp. 315-332, 1993

[7] *Guilherme de A. Barreto, Aluizio F. R. Araújo, Helge J. Ritter*, Self-Organizing Feature Maps for Modeling and Control of Robotics Manipulator, Journal of Intelligent and Robotic Systems, Vol. 36, pp. 407-450, 2003

[8] *Viktor Žáček*, Kohonenova Samoorganizační mapa, VUT v Brně, Diplomová práce, 2012

[9] *Kazuyuki Kobayashi*, funkce VCAPG2 pro MATLAB, 2003
http://www.mathworks.com/matlabcentral/fileexchange/2939-vcapg2

[10] SOM Toolbox pro MATLAB, 2005
http://www.cis.hut.fi/somtoolbox/

[11] Bioloid Toolbox pro MATLAB, 2014
http://incognite.felk.cvut.cz/index.php?page=projects/bioloid&cat=projects

[12] *Robotis*, User's Manual Dynamixel AX-12, 2006

[13] *Juha Vesanto, Johan Himberg, Esa Alhoniemi, Juha Parhankangas*, SOM Toolbox for Matlab 5, Helsinky University of Technology, 2000

[14] Rozmístění portů na řídící jednotce CM530,
http://support.robotis.com/en/software/embeded_c/cm530/programming/hardware_port_map_cm530.htm

[15] *Joel M. Esposito*, Tutorial: Serial Communication in Matlab, 2009

Příloha A – Obsah přiloženého disku

Přiložené CD obsahuje zdrojové kódy vytvořeného systému vizuomotorické koordinace, text této diplomové práce a Bioloid Toolbox.
SOM Toolbox a funkci VCAPG2 jsem na CD nepřiložil, jelikož nejsem jejich autorem. Oba programové nástroje lze volně stáhnou na adrese [9] respektive [10].