Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Learning neural proprioceptive-tactile arm representations in a humanoid robot

Diploma thesis

Jana Harvanová 2021

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Learning neural proprioceptive-tactile arm representations in a humanoid robot

Diploma thesis

Study program:	Cognitive Science
Field of study:	2508 Informatics
Department:	Department of Applied Informatics
Supervisor:	prof. Ing. Igor Farkaš, Dr.

Bratislava, 2021
Jana Harvanová

13016281

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Jana Harvanová
Study programme:	Cognitive Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Learning neural proprioceptive-tactile arm representations in a humanoid robot

Annotation: People are able to reach so-called somatosensory goals specified by proprioceptive (joint angles) and tactile information, without reliance on vision. Self-touch represents an important developmental process, allowing autonomous construction of a complex relationship between these two modalities, as parts of the body schema.

Aim: Buiding on the existing work (Pecen, 2019), improve and test a neural network model that learns to associate proprioceptive representations and tactile representations in a simulated iCub robot. Generate a larger set of inputs from robotic simulator where the robot explores its own body by touching itself. Analyze the model behavior.

Literature:
Hoffmann M. & Bednárová N. (2016). The encoding of proprioceptive inputs in the brain: knowns and unknowns from a robotic perspective. In Kognice a umělý život XVI, pp. 55-66.
Hoffmann, M.; Straka, Z.; Farkaš, I.; Vavrečka, M. & Metta, G. (2018). Robotic homunculus: Learning of artificial skin representation in a humanoid robot motivated by primary somatosensory cortex. IEEE Transactions on Cognitive and Developmental Systems 10 (2), 163-176
Pecen M. (2019). Akvizícia proprioceptívno-dotykových reprezentácií tela u humanoidného robota. Diplomová práca, FMFI UK v Bratislave.

Supervisor:	prof. Ing. Igor Farkaš, Dr.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	06.01.2020

Approved: 06.02.2020 prof. Ing. Igor Farkaš, Dr.
Guarantor of Study Programme

.....
Student

.....
Supervisor

13016281

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jana Harvanová
Študijný program: kognitívna veda (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Learning neural proprioceptive-tactile arm representations in a humanoid robot
Učenie neurálnych proprioceptívno-dotykových reprezentácií rúk u humanoidného robota

Anotácia: Ľudia sú schopní dosahovať tzv. somatosenzorické ciele určené proprioceptívnymi (uhly kĺbov) a hmatovými informáciami bez toho, aby sa spoliehali na videnie. Dotyk samého seba predstavuje dôležitý vývinový proces, ktorý umožňuje autonómne budovanie komplexného vzťahu medzi týmito dvoma modalitami ako súčasti schémy tela.

Cieľ: Vychádzajúc z práce Pecen (2019), vylepšite a otestujte model neurónovej siete, ktorý sa učí spájať proprioceptívne a hmatové reprezentácie v simulovanom robotovi iCub. Vygenerujte väčšie množstvo trénovacích dát z robotického simulátora, kde robot skúma svoje vlastné telo pomocou dotykov svojich rúk. Analyzujte správanie modelu.

Literatúra:
Hoffmann M. & Bednárová N. (2016). The encoding of proprioceptive inputs in the brain: knowns and unknowns from a robotic perspective. In Kognice a umělý život XVI, pp. 55-66.
Hoffmann, M.; Straka, Z.; Farkaš, I.; Vavrečka, M. & Metta, G. (2018). Robotic homunculus: Learning of artificial skin representation in a humanoid robot motivated by primary somatosensory cortex. IEEE Transactions on Cognitive and Developmental Systems 10 (2), 163-176
Pecen M. (2019). Akvizícia proprioceptívno-dotykových reprezentácií tela u humanoidného robota. Diplomová práca, FMFI UK v Bratislave.

Vedúci: prof. Ing. Igor Farkaš, Dr.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 06.01.2020

Dátum schválenia: 06.02.2020 prof. Ing. Igor Farkaš, Dr.
garant študijného programu

.....
študent

.....
vedúci práce

Acknowledgements

I wish to express my sincere thanks to my supervisor prof. Ing. Igor Farkaš, Dr. for his invaluable advice, guidance and patience while I was programming and writing this thesis. I would also like to thank Mgr. Matěj Hoffmann, PhD. for his advice and help with iCub simulator and insight into data collection, and Dr. Lúčny for his approval of using some parts of his code. Finally, I am grateful for the emotional support I received from my family and partner.

Abstract

People are able to reach so-called somatosensory goals specified by proprioceptive (joint angles) and tactile information, without reliance on vision. Self-touch represents an important developmental process, allowing autonomous construction of a complex relationship between these two modalities, as parts of the body schema. Vision is not required for this process, although it does get involved in later stages of development. In this master thesis, we built upon an existing thesis by Martin Pecen in which he has implemented a biologically-inspired neural network model for this purpose. We concentrate on one of the proposed models BAL. Before associating the two modalities, both sets of input signals are topographically preprocessed using self-organizing maps. The main contribution of this work was expanding the data set and executing different experiments not done in the original work. The final data set consists of hundreds of samples, fully auto-generated, using the simulator of iCub by babbling its arms resulting in self-touch. The model achieved decent performance and generalization during both training and testing on both touch and non-touch data.

Keywords: proprioception, iCub, robotics, neural networks, bidirectional associative learning

Abstrakt

Ľudia sú schopní dosahovať tzv. somatosenzorické ciele určené proprioceptívnymi (uhly kĺbov) a hmatovými informáciami bez toho, aby sa spoliehali na videnie. Dotyk samého seba predstavuje dôležitý vývinový proces, ktorý umožňuje autonómne budovanie komplexného vzťahu medzi týmito dvoma modalitami ako súčasti schémy tela. Zrak nie je nevyhnutná súčasť tohto procesu, hoci v nejskoršom štádiu vývinu je aplikovaný tiež. V tejto diplomovej práci nadväzujeme na existujúcu diplomovú prácu od Martina Pecena, v ktorej implementoval biologicky inšpirovanú komplexnú neurónovú sieť. V našej práci sa sústredíme na jeden z navrhnutých modelov BAL. Pred asociovaním týchto dvoch modalít , obe vstupné množiny dát sú topograficky predspracované použitím samoorganizujúcich sa máp.Hlavný cieľ práce je otestovať model na väčšom data sete a ďalších experimentoch, ktoré neboli v pôvodnej práci vyskúšané. Finálny dataset obsahuje stovky vzorov, je plne autogenerovaný pomocou navrhnutého algoritmu pomocou simulátora iCub. Simulátor hýbal rukami iCuba ktoré sa navzájom dotýkali. Model dosiahol relatívne dobrú presnosť a schopnosť generalizácie, na dotykových aj nedotykových dátach.

Kľúčové slová: propriocepcia, iCub, robotika, neurónové siete, asociatívne učenie

Contents

List of Figures

List of Tables

Chapter 1

Introduction

It is increasingly more common for researchers and the automation industry to have an interest in robotics, or more specifically, how to create robots which can function reliably and independently when performing their respective tasks in our increasingly technological world.

One of the crucial concepts some of these robots will need is the awareness of their surroundings, so they will not collide with the environment, people, animals, other robots etc. causing harm to themselves or others. An important part of this concept is the robots' own body representation, for them to have an understanding of where they are oriented in space and knowing how different movements of their body affects this position in the environment. This is useful for grasping and interacting with objects, but also in knowing which movements would cause them to collide with their own body, and if so, where.

In humans, the ability to perceive their body's position in space (proprioception) is gained in infancy. This stage of human life is characterized by the fact that eyesight is not yet fully developed, and therefore the baby explores its body mostly without using its eyes. According to research, the children at this stage perceive their body by moving their limbs around, during which they sometime collide, and this stimuli is the information needed for learning the body schema.

Understanding how humans have this knowledge represented in the brain and being able to model it by using neural networks could be useful for both robotics, and for further research into how the process develops in humans.

This diploma thesis consists of two main parts: the first part is an algorithm for acquisition of proprioceptive data with their corresponding touch inputs from a humanoid robot named iCub. The second part is a complex neural network model, which will

use the acquired data to learn the associations of which proprioceptive configurations correspond to which (if any) touches occurring, and where. In this first chapter, we will discuss the current state-of-the-art of the research field regarding proprioception, body schema acquisition/representation using humanoid robotics. In Chapter 2, we will introduce the technical aspects of the thesis and talk about iCub, YARP, and other technologies and approaches used. Chapter 3 will introduce the aspects of data acquisition, multiple approaches to automating this process we've tried, what was used in the end, and why. In Chapter 4, specific neural networks used as a part of the complex model will be described. Chapter 5 contains the experimental results.

This chapter introduces how something quite complex as self-body representation is thought to be organized in the brain, and diving into deeper definitions and understanding of some terms which are crucial for this work.

1.1 Biological motivation

Research suggests that an implicit sense of self is developing from birth, long before children begin to manifest explicit (conceptual) self-knowledge by the second year (Rochat and Striano, 2000).

Most of the processes involved in self-perception regarding the body seem to occur in infancy, before the child is even one year old. Some part of this time-frame, in the early months, sight isn't fully developed which means the learning of the body schema takes place (in part) without visual perception (Kandel and Schwartz, 2000).

At that stage, the small child acquires this body representation by babbling - flailing his or her feet and arms around, moving so that sometimes they collide with physical objects in their environment, but more importantly, making contact with their own body at different places. This is a sensory stimuli the child feels on their skin and, in a very abstract manner, an association happens in the brain of what position of their body causes which stimulus - if any.

The outcome of this representation is the ability to understand where one's body is positioned in space, and recognizing that by executing some movement, a touch on on some specific body part happens (or not), even when closing our eyes.

1.2 Proprioception

In the acquisition of self-body schema, there is one crucial ability involved and that is perception. People have the ability to perceive their surroundings and reacting to incoming stimuli from the environment. These stimuli are processed through the receptors of the respective sensory organ, and this signals travel all the way to the central nervous system. In the nervous system, they are sent further and ultimately into the brain, which produces neural responses to the stimuli.

1.2.1 Terminology

The term proprioception was passed down to us by Sherrington (Sherrington, 1907). He stated, "In muscular receptivity we see the body itself acting as a stimulus to its own receptors — the proprioceptors." Traditionally, the term proprioceptor has been restricted to receptors concerned with conscious sensations, and these include the senses of limb position and movement. (Proske and Gandevia, 2012)

We could therefore say that proprioception is responsible for perceiving one's body position in space, as well as coordinating the movement of limbs and torso.

More so, even when we are not directly looking at our limbs, we are still able to reasonably predict where they are located, as well as whether they are static or moving.

1.2.2 Role of proprioception in motor skills acquisition

A more expansive question is in what role does proprioception play in the acquisition and development of motor skills in children. There has to be a regular re-calibration of the body schema, because the body of children undergoes a lot of changes in relatively short periods of time.

Pioneering observations by Laszlo and Bairstow (Laszlo and Bairstow, 1980) have led to the view that the ability of children to use proprioceptive feedback has matured by the age of 7 years. Other reports suggest that there is ongoing learning of movement control which continues through adolescence (Goble et al., 2005), and even to adulthood. (Hearn M, 1989).

So, development of proprioceptively controlled movements continues beyond the period required for acquisition of motor skills, thought to be complete by the age of 10 years (Seefeldt and Haubenstricker, 1982) (Proske and Gandevia, 2012)

If we were to look on proprioception from the bio-physical perspective, proprioception can be understood as the body's ability to collect and process information by using skin receptors, various movements and actions coming from the muscles and bones, and transferring them to a part of the brain called the somatosensory cortex, which will be more closely introduced in the next subchapter.

Sensory neurons carrying these nerve impulses from sensory stimuli (the skin, musles etc.) toward the central nervous system and brain, are called afferent neurons. Afferent neurons carry signals to the brain and spinal cord as sensory data. Then in the gray matter, these signals are further processed, and an output which describes the current state of the body is produced. (Proske and Gandevia, 2012)

1.2.3 Somatosensory cortex

An area of the brain that receives and processes sensory information from the entire body is called the somatosensory cortex Penfield and Rasmussen (1950). This area of the brain processes the received sensations of touch, pain, and vibration from the entire body. It is divided into two parts:

1. Primary somatosensory area or cortex (S1)

2. Secondary somatosensory area or cortex (S2)

The reason for this division is that a distinct and separate spatial orientation of different parts of the body is found in the two areas (Ronthal, 2004) , but most activations seem to happen in the Primary somatosensory area or cortex part.

Its location is in the forebrain, which is present in the parietal lobe (Ronthal, 2004).

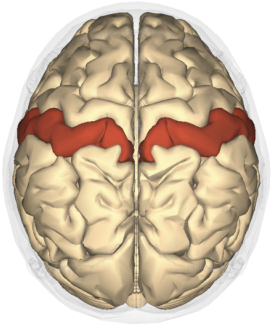


Fig. 1.1 Somatosensory cortex location

Since the somatosensory areas react to the stimuli and sensations of the whole body, specific representations for different body parts is present.

What is interesting is that the proportion of the representation differs - the proportion of the cortex representation for an individual body part depends not on its size, but on its functionality or frequency of usage. In fact, the area occupied by a particular body part is proportional to the number of sensory receptors present in it.

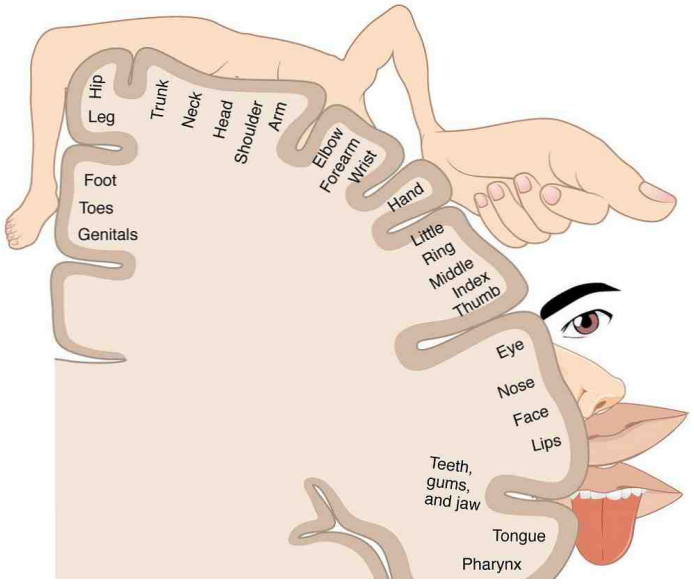


Fig. 1.2 Somatosensory homunculus depicting separate body parts, topologically organized. Exact areas of separate body representations are not universally fixed, there can be differences among individuals. As mentioned above, note the differences in size in relation to frequency of usage.

Another interesting finding is from a very recent research regarding the primary somatosensory cortex actively encoding information also with imagined movement only, with absence of sensory information.

Specifically, the research shows that single units in human primary somatosensory cortex encode imagined reaches in a cognitive motor task (but not other sensory–motor variables such as movement plans or imagined arm position). (Jafari et al., 2020) with quite good temporal and spatial precision of the single unit activations.

It seems to hint to the fact that the (imagined) arm movements and positions (proprioception) as well as (imagined) reaching towards objects have a common representation. Additionally, in some cases also an imagined stimuli can activate neurons in these areas, but for others, only physical stimuli are reactive.

1.3 Current state of the study field

This subsection will briefly summarize the state of the research field concerning humanoid robots, proprioception or self-body representation so far.

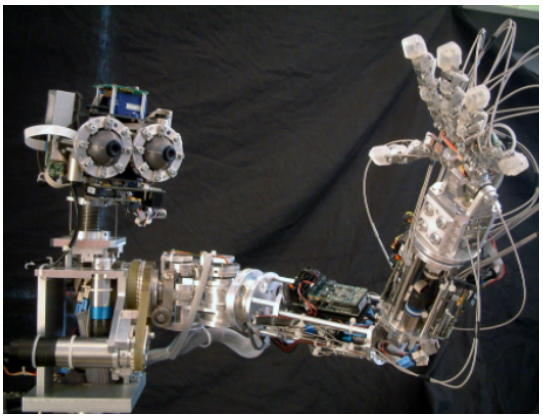


Fig. 1.3 The humanoid robot James (Jamone et al., 2006)

One research conducted in University of Genoa, Italy, has designed a humanoid robot called James, which they equipped with moving eyes and neck, an arm and a highly anthropomorphic hand. The design of this robot has been guided by the concept of embodiment, material compliance and embodied interaction (Jamone et al., 2006).

The main point of the research was to design experiments to see the relationship of tactile sensing and proprioception in getting information about grasped objects of similar shape, but different softness and elasticity, and correctly classifying them.

For this, two Self Organizing Maps (SOM) were used to cluster and visualize data - one had been trained only with proprioceptive information, the other one on both tactile and proprioceptive information. The results have shown that small changes in proprioception of the hand with similar objects wasn't enough to consistently predict the correct object. With both proprioceprive and tactile information, James was notably more accurate with classifying the objects he grasped.

Another study done by Hoffman et. al, they presented work studying how a humanoid robot iCub with sensitive skin could learn a topographic representation of its body surface from experience, by receiving tactile stimulations all over its artificial skin (Hoffmann et al., 2018).
In this case, it had been also SOMs used for the topological representation of the robot's skin, however, instead of the classic SOM algorithm, they proposed a modification of it (MRF-SOM) that allows to prespecify certain, partially overlapping, receptive fields of the output layer neurons. This modification was biologically inspired, as we have already mentioned above when we introduced the somatosensory cortex a.k.a somatosensory homunculus.

Another study, which was also done using iCub, proposed a biologically inspired approach to model a concept of peripersonal space - region of space immediately surrounding our bodies - for robots. Guided by the present understanding of the neurophysiology of the fronto-parietal system, they developed a computational model inspired by the receptive fields of polymodal neurons identified in brain areas such as F4 or VIP (Roncone et al., 2016)

The conducted experiments had shown that iCub was able to learn its peripersonal space, in real-time, by a relatively easy way in simple interaction with the robot. It also lead to the generations of behaviours, such as avoiding an unknown object approaching the robot, or reaching for an object presented to the robot in the peripersonal space.

Chapter 2

Technology and tools

This chapter will introduce the tools and technologies used. Before going into further details of YARP and iCub which will make up most of this chapter, the only programming language used for implementation of the whole model and all associated scripts was Python 3.7, with everything running on a Windows 10 x64 machine.

2.1 YARP

YARP (Yet Another Robotic Platform) supports building a robot control system as a collection of programs which communicate in a peer-to-peer way, using ports.[1] It is very flexible in that it supports multiple connection types such as TCP, UDP, Muliticast/Local and more.

Originally, YARP was written for C++, but using other technologies such as SWIG. makes it possible to use it also from other, higher-level languages such as Python, C#, Ruby and others.

SWIG is an interface compiler that connects programs written in C and C++ with scripting languages. It works by taking the declarations found in C/C++ header files and using them to generate the wrapper code that scripting languages need to access the underlying C/C++ code. Since our experience with C++ is limited, we had tried this approach with SWIG to create YARP bindings for python. Unfortunately it turned out that compiling the bindings was quite problematic on windows, but in the end, with the permission from Dr. Andrej Lúčny, we used YARP bindings which had already been compiled and used by him in his course Introduction to robotics.

[1] `https://www.yarp.it/latest/`

2.1.1 Usage

As already mentioned, the communication between YARP processes is implemented through ports. It is possible to open ports, connects ports to other ports, and create a data stream which can be read for another port. The messages are sent through the network adapter, which means that it is not required for all processes to run on the same computer.

Some of the most useful commands possible to write in the command line are:

- `yarpserver` - starts a yarp server instance.
- `yarp read /port_name` - opens a port for reading.
- `yarp write /port_name` - creates a port where data will be streamed (and accessible from other ports for reading this data)
- `yarp connect /port1 /port2` - creates a connection between two ports. This makes it possible for them to exchange data.
- `yarp disconnect /port_that_is_writing /port_that_is_reading` - removes the connection between the ports specified

The same functionality is accessible through the library method calls, so for example in our case, YARP functionality was accessible from Python directly.

2.2 Robot iCub

iCub is a humanoid robot widely used for research purposes with regards to human cognition and artificial intelligence. The whole project is open-source, all hardware, software and documentation related is released under the GPL license.

This robot is approximately 1 meter tall and was designed by the RobotCub Consortium, with collaboration of several universities in Europe and ultimately constructed by the Italian Institute of Technology icub-humanoid-robot. The naming of iCub is part acronym, where 'cub' stands for Cognitive Universal Body.

The motivation behind the creation of a robot so closely resembling a human is the Embodied cognition hypothesis.

Embodied cognition holds that cognitive processes are deeply rooted in the body's interactions with the physical world. embodied-cognition. A baby learns a lot of cognitive

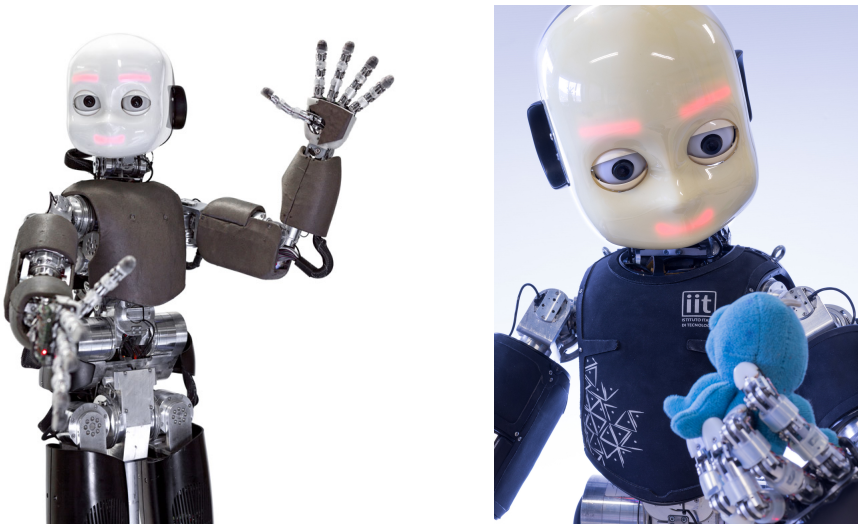


Fig. 2.1 Well-known pose of waving iCub [left],
iCub grasping a plush toy and looking at it [right]

abilities by directly interacting with the environment it exists in by using its senses and limbs. Consequently, its internal model of the world is in no small part determined by the form of the human body - since, should the body be different, then the experienced sensations through the hypothetical other-body would be different as well.

This robot was designed with this goal in mind - to try to accurately re-create the perceptual system of a small child, so that researchers could create scenarios which simulate cognitive learning similar to what a small child would experience. Of course, it is also possible to use iCub for various other experiments in robotics, AI, and cognitive science where a humanoid robot is appropriate.

2.2.1 Physical specification

iCub is approximately 1 meter tall, weights 25 kilograms and has 53 motors that enable the movement of the head, individual parts of hands, forearms, legs, and torso. It is equipped with sensors which make it possible for iCub to hear and see (Metta et al., 2010) as well as artificial skin - pressure sensors that can detect touching. It is capable of crawling, grasping objects, interacting with people. Additionally, it has a semi-limited ability to display different facial expressions thanks to eyebrows and mouth, simulated as red LED pixels and lines on its head.

2.2.2 Proprioception of the robot

iCub's proprioceptive system consists of 53 motors, which control individual limb joints. Every joint can be rotated , some in one direction, others at most in three directions.

Each of these motors moving the joints is restricted in a way which would resemble real, human joint movements as much as possible.

The term used in robotics in general for describing this movement range of one joint is called the *degree of freedom*, expressed in degrees and angles. For iCub, the range of all degrees of freedom is between 44° and 270°. Also, not every joint has the same range of movement, as is seen in real-life people. Additionally, individual degrees of freedom are not symmetrical around the value 0 (Metta et al., 2010), and every degree of freedom has a default, non-zero value set.

The internal proprioceptive system is fully observable by the robot at any point in time, meaning that at any time it knows all degrees of freedom of all the joints in its body. This can be understood as a very simplified analogy of proprioceptive information in the robot's 'brain'. For living beings, these information consist of the sensations perceived by their skin, muscles and bone movements, rotations and positions. From the robot's perspective, it is simply the numeric values of angles for each motor joint at a given time.

Further in the thesis, we will refer to limb positions consisting of n joints expressed by each joint's current degree, as *limb configurations*.

2.2.3 Artificial skin, sensing touch

As briefly mentioned above, iCub's hands, forearms and the torso are covered by artificial skin. Physically, this skin is composed of multiple triangular chips, with each containing 10 individual touch sensors - taxels, which are capable of detecting pressure.

In practice, multiple of these triangular chips are connected together and form a surface of a body part, as we can see illustrated below.

2.3 iCub Simulator

Beside the physical iCub robot, an open source simulator `iCub_SIM` was developed. The simulator tries to replicate the behaviours and properties of the physical robot as accurately as possible.

It is not trivial to obtain a physical iCub model, since they are quite expensive. Furthermore, it definitely is more practical to do research and initial testing of a model

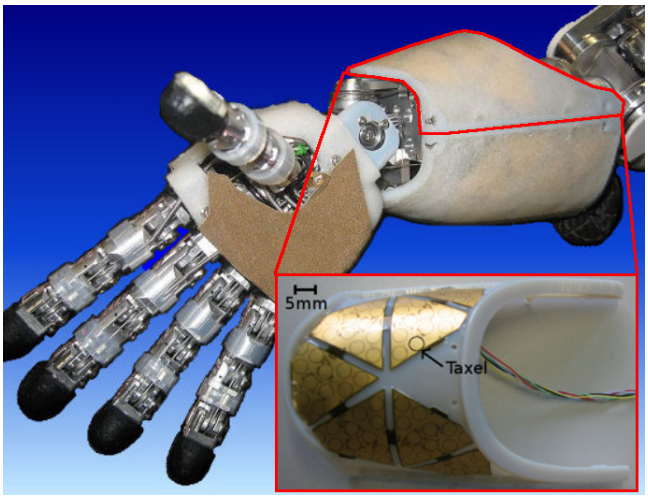


Fig. 2.2 Right arm of iCub, with skin mounted on the forearm, palm and fingertips. In the red box, a detailed view of the forearm cover and the placement of the sensors (Del Prete, 2013).

or solution in a simulator in the early stages of a project, since it is a safe environment where mistakes do not pose unforeseen, irreversible damage to the hardware, for example.

2.3.1 iCub_SIM modules

In order to have a working iCub_SIM environment, we first need to install YARP and create a `yarpserver` instance. After the server is running, we can run the command `iCub_SIM` in the command line, which will recognize the yarpserver running, sets up all the required ports, and initalizes a 3D graphical interface with the simulated iCub.

By default, after installing `iCub_SIM`, the collision and skin emulation are turned off. In order to enable them, we need to manually change the settings in one of the configuarion files called `iCub_parts_activation.ini` (the concrete steps for this process are specified on the official iCub wiki) before running the simulator instance.

If they would stay disabled, collision wouldn't occur when changing the limb positions, potentially causing the limbs to overlap with themselves, or other body parts. If the skin emulation would stay turned off, we wouldn't get any tactile information from the ports.

Other than the main simulator instance, there are additional optional modules used for visualisation or additional control of the robot, some of which we will briefly describe.

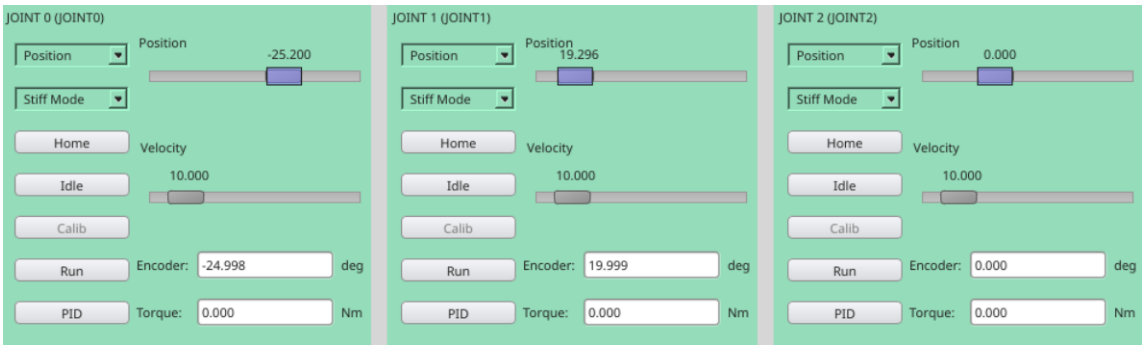


Fig. 2.3 An example of how the user interface of `MotorGui` looks. For the sake of clarity, only the first three joints are shown.

2.3.2 MotorGui

`MotorGui` is a graphical application which can be run alongside the simulator. It automatically detects which ports the simulator is using, and can be used to change individual joint positions of individual limbs, head, or torso.

We have used this module mostly in the initial stages of development, when we designed a simple, automated babbling algorithm for data acquisition. More details will be mentioned later in chapter 3.

2.3.3 iCubSkinGui

`iCubSkinGui` is a module for skin visualisation. When executed, it shows a 'map' of a specified body part, and visualizes a real-time activation of its tactile sensors. Note that one `iCubSkinGui` instance can visualize a single body part only, so for multiple different body parts, multiple instances need to be started.

Also, with contrast to `MotorGui`, the ports need to be connected manually. What that means is that after running the base `iCub_SIM` instance, and the skin emulation is enabled, you connect the iCub's skin port to a virtual port like

```
yarp connect /icubSim/skin/left_hand_comp /skinGui/left_hand:i
```

which tells YARP to stream the data from the port `/icubSim/skin/left_hand_comp` to the `/skinGui` port - which is the one read by the `iCubSkinGui` module.

And then, the `iCubSkinGui` instance can be run like:

```
iCubSkinGui --from [path_to_skin_config_file_name.ini]
```

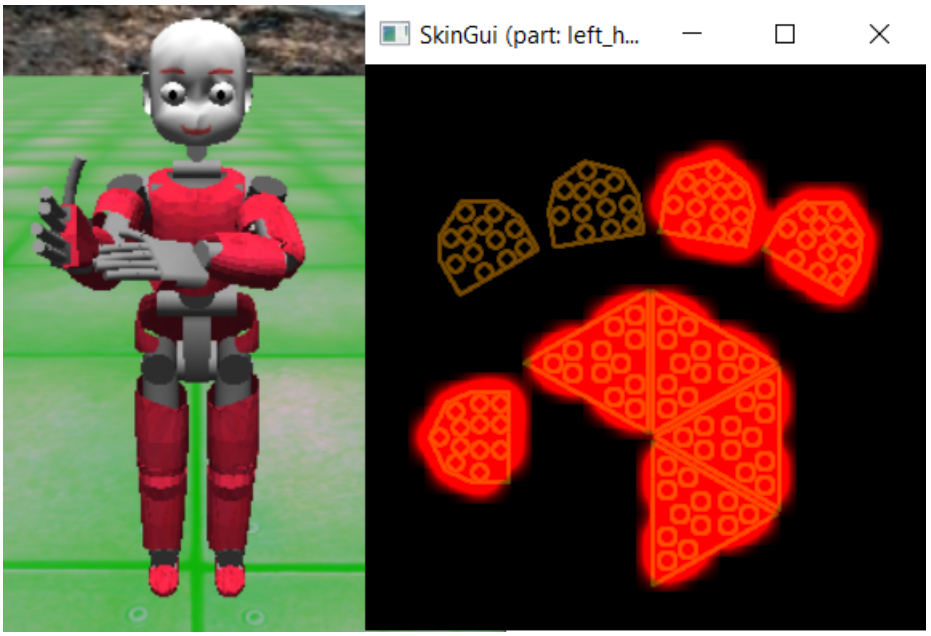


Fig. 2.4 Simulated iCub standing, his left hand touching his right forearm. One `iCubSkinGui` instance showing the skin emulation of left hand, the index and middle finger sensors are active.

Then, the instance shows the state of skin emulation for the specified body part (in this example, the left hand.)

Chapter 3

Data acquisition

In this chapter, we will describe our approaches to data generation and data collection on which our neural network model was later trained, and the problems we had encountered on each of them. The idea was to have an automatized program which the user could run for a parametrized amount of time, during which the iCub's arms would move to different positions, while simultaneously tracking the state of iCub's skin - essentially collecting pairs of proprioceptive data to touch data, at a given point in time.

As we've mentioned in the previous chapter, despite the fact that `YARP` was originally developed in C++, we have acquired compiled Python bindings for YARP from professor Lúčny. This enabled us to access YARP modules directly from Python. We've also used a pre-coded interface of `iCubLimb` written in Python, also by prof. Lúčny. The interface, after creating an instance of it in code, can represent one whole arm. It sets up all the YARP connections to correct ports, had a Get method for getting the current joint positions, and a Set method for changing the joint position to specified values.

We have enhanced the `iCubLimb` interface with implemented methods for babbling, so from the outside main script, we could babble either one or the second arm by a simple method call.

3.1 Simulator data pre-processing

When collecting the data from iCub, we noticed that the number of all taxels and number of all values for one touch data sample is different. The underlying reason for this discrepancy was that iCub is sending in total 192 values for every group of triangular chips, even though some groups were incomplete and consisted of less than 16 triangles.

Hand	min	max	Forearm	min	max
fingers - out/in spread	0	60	shoulder - vertical	-90	10
thumb - towards palm	10	90	shoulder - horizontal	0	160
thumb - towards fingers	0	90	shoulder - rotating	-35	80
thumb - cur/uncurl	0	180	elbow	15	105
index finger - lower	0	90	elbow - rotating	-90	90
index finger - upper	0	180	wrist - horizontal	-90	0
middle finger - lower	0	90	wrist - vertical	-20	40
middle finger - upper	0	180			
ring and pinkie fingers	0	270			

Tab. 3.1 The ranges of degrees of freedom, for all 16 individual joints

One triangular chip consists of 12 values, in which 2 are always 0 (specifically, the values on the fourth and seventh position serve as potential memory and is reserved for future, set to 0 by default).

Additionally, we noticed that for forearms on ports `/icub/skin/left_forearm_comp` and `/icub/skin/right_forearm_comp`, iCub was sending 384 values instead of 276. There were multiple zero-values reserved by the system for other operations, and their valeus never changed from 0. The specific ranges of the always-zero values in the touch vector were 193–204, 217–252, 265–288, 325–336, and 361–384.

After deleting these obsolete values, we had the complete touch information of $108 + 276 = 384$ values, for one limb. Important thing to note is the simulator works in a way that, if at least one taxel was signaled as a touch, then the whole triangular chip is considered as a touch and the taxel's remaining values are signaled as well - therefore, we could reduce the 384-length vector to just 32 values, without any information loss.

For proprioception, one limb has 16 joints in total - 5 are for manipulation of shoulder and elbow, the remaining 11 for hand, wrist and fingers. Total proprioception information consisted of two vectors of length 16, and later, since individual joints have different degrees of freedom (as can be seen in Table 3.1), we have normalized the values to fall into [0, 1] as:

$$val_i = \frac{val_i - min_i}{max_i - min_i}$$

3.2 Data representation

Irrespective of which babbling algorithm was used for data generation, the data representation format was the same.

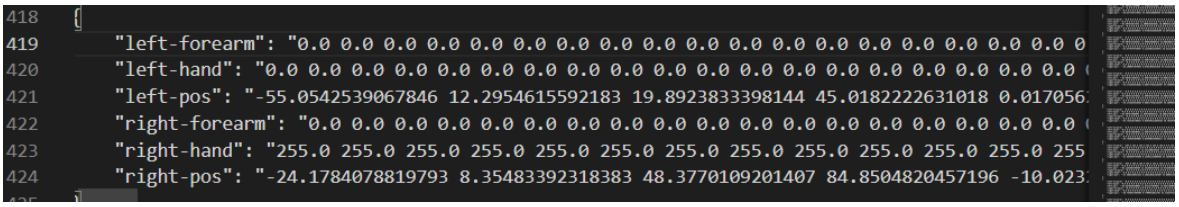


Fig. 3.1 Example of one batch of values for proprioception ('left-pos' and 'right-pos') and touch data (other entries, either 0.0 or 255.0 values shown).

The first data format was a simple JSON file , with raw data values of both limb joints and touch values from individual ports. The file format was introduced by Martin Pecen in his diploma thesis for data collection, we've only re-created the format and script in python (originally was in C++).

The entries for 'left-pos' and 'right'pos' are proprioception data. Only first 4 out of all 16 values are displayed, for illustration purposes. Other entries are touch data, in this case either zero values, or value 255 (signalling touch).

After each data collection session, this file was created.

From this file, another JSON file was created. This file had the data normalized, and the touch vectors reduced, as described in the previous section. Additionally, all touch vectors were changed to show either a zero (on place where no touch was detected) or 1 (if touch occured), instead of the original scale of [0, 255].
The reason for the larger [0,255] range coming from the simulator is that there is an existing skin pressure functionality - which was not needed for our purposes and was turned off, but the simulator still represents a touch as a 255 value.

3.3 Data validation

For data validation, we've created a script that was able to 'replay' the process of data collection. It needed to have iCub_SIM running, and then simply loaded the first JSON file with collected simulator raw data, read the data entry batches, and set the proprioceptive values form the batch to iCub_SIM limb joints. This was useful for looking at the data as well as sanity checks of the collected configurations - for instance, as we were setting the proprioceptive configurations to the simulator's joints, we could compare the recorded touch data from the batch with the values read from the simulator's touch ports and check if they match. Other than that, we could check for touch / no touch

configurations visually, potentially detect a mistake during data collection (i.e proprioceptive configurations would result in iCub having its limbs far apart, but the touch vectors would signal a touch occuring, which would mean the batch was incorrect.)

3.4 Babbling algorithm

The first approach we tried was a replication of an open source babbling module implemented in C++. The module was developed to issue pseudo random (sinusoids) commands to the iCub, which could be done either with several joints, or an individual, specified joint.

The original source code can be found on robotology' github[1].

The main idea behind the algorithm was to generate a new value for joint i by an equation:

$$x_i = start_i + asin(ft2\pi)$$

where a is amplitude, f is frequency, and t stands for time.

$small_i$ is the starting value of joint i. These initial values were specified during module initialization, before calling the babbling method.

The amplitude and frequency were parametrized by the user. The defaults were 5 for amplitude, 0.2 for frequency.

The final, new joint value has been calculated as:

$$value_i = 10 * (x_i - encoder_i)$$

where $encoder_i$ was the i th joint before the change value x_i was computed.

Using this algorithm, we could generate different limb configurations.

Depending on the selected start positions, and the values set to a and f, the degree of one babbling change were more or less pronounced. The main limitation in this approach was that, since the underlying joint changes were always calculated based on the sinusoid and other parameters in time, the babbling was not truly random, the movements looked artificial and were repeated frequently.

Additionally, since we truly wanted to create an approach that would be more biologically plausible, we thought that generating joint changes based on mathematical functions didn't serve this goal very well.

[1] `https://robotology.github.io/icub-hri/doxygen/doc/html/babbling_8cpp_source.html`

What could potentially increase the variation of limb configuration would be to change the 'initial' limb positions, originally set only at the beginning of babbling, also during the babbling process.

3.5 A simple Motor babbling algorithm

This approach was tried after the previous implementation above. Since the original babbling was using a mathematical function, we wanted to create a more biologically motivated approach, which would be more randomized.

For this, we have designed a simple method of babbling. We defined an list of integers k describing the potential values by each joint could be moved.

$$k = [5, 8, 10, 12, 15, 20, 30]$$

For every joint, we had decided on a minimum and maximum values possible to generate. This was decided by empirical observations during observations of the babbling.

One arm was fixated during the babbling phase, as the second arm was babbled. The babbling took a parametrized amount of time, during which:

- For every joint, a random value n from k was taken as a reference by which the joint should change.
- For every joint, after determining the value of n, direction d was chosen randomly, $d \in \{1, -1\}$.
- New joint value j_{t+1} was calculated as

  $$j_{t+1} = j_t + (kd)$$

  where j_{t+1} was also verified that it wouldn't fall out of bounds of specified minimum and maximum values of joint j. If it did, then the minimum/maximum was set instead.

We didn't do any specific checks for ensuring that the babbled limb made contact or not - after running the babbling for some time, both non-touch and touch limb configurations emerged.

As can be observed in table 3.2, it was mostly the shoulder/elbow joints which were babbled, with slight movement of wrists and fingers. The main reason was to keep it

Joint description	min	max
Shoulder - vertical	-35	0
Shoulder - horizontal	0	60
Shoulder - rotating	85	105
Elbow - rotating	-10	30
Wrist - vertical	-30	0
Wrist - horizontal	-15	15
Fingers - spreading in/out	40	58

Tab. 3.2 The final ranges of all limb joints which were changed during the motor babbling algorithm.

simple, as moving the fingers too much before could sometimes lead to the limbs getting stuck. Then, further code-behind setting of the joint positions from the babbling script would sometimes try to move the arms where they couldn't physically be moved - because of enabled collision - and, in worst case scenario, caused iCub_SIM to crash.

We identified several limitations of the method. One problem was that, for ensuring that a reasonable amount of touch configurations would occur in some time frame, the space which is covered by the limited joint ranges is only a subset of all possible DoF the simulator is capable of achieving. This would limit the limb movements to (mostly) in front of iCub, with some area on both sides of his body.

There is no exploration of arms moving i.e above the head, reaching towards the back, or down. It could be argued that the area in front of iCub is also the area a child is probably exploring the most, since in infancy, the baby is mostly lying down on its back. Nevertheless, it needs to be mentioned and acknowledged.

With this fact is another associated issue arises, which is that after some time (2 hours of babbling), some limb configurations started to repeat, which lead to only around 75-80% of the data being unique. This was somewhat expected, as the ranges of some joint movements aren't as large. Originally, we've had the joint ranges larger, but that caused a significant drop in touch configurations. Nevertheless, it wasn't that big of an issue, since overall on the longer babbling sessions, the generated amount of different touch configurations amounted to 200-400. Non-touch data count was even higher.

Lastly, having one arm fixated could also be considered a limitation. We could move both arms at the same time, though again, it would be harder to get as many touch configurations. This is more of a drawback addressed towards biological plausibility - in itself, the configurations are not incorrect, since symmetrically, we would get the same limb configurations if the arms were reversed.

Because of this, we've also generated the symmetrical complements to the collected data, after the data was collected.

3.6 Final data set and data collection

In the end, we decided to use the second approach of data generation, since it had more varied amount of limb configurations.

For data collection, we've done so as a part of the data generation algorithm. After every limb joint change, the program waited for a short time (2seconds) for the changes to truly take place in `iCub_SIM`, all motors finish moving and the simulator's joints have stabilized. Then, the touch ports were queried and their values stored in memory during the program was running. When babbling finished, all the collected proprioceptive/touch configurations were dumped into a JSON file described at the start of this chapter.

However, later when training the model, we couldn't achieve a reasonable training or testing accuracy. Other metrics for describing the trained model were also either insufficient, or oscillating during the whole time the training took place.

Later, when validating the collected data and visualizing them in `iCub_SIM`, we have found out that some configurations were desynchronized - meaning that some proprioceptive limb configurations were assigned an incorrect touch data, and thus were incorrectly classified as a touch or not. This was evident when we restricted the visualization to display only limb configurations in which touch has occurred, but the visualisation had shown the limbs not touching at all.

We've tried multiple approaches to eliminate this problem such as synchronizing threads during the babbling process, waiting for longer between every joints babbling, to no end.

In the end, we concluded that it was probably an issue with how YARP was accessed and operated through the Python bindings in our case, and as the probable delays causing the desynchronization were internal, we couldn't address them. The solution was using a separate module for data collection called the YARP data dumper.

3.7 YARP Data dumper

YARP data dumper acquires and stores data from a YARP port. When launched, the service monitors the presence of incoming data and stores it within a folder called with the same name as the service port.

Another module called `yarpmanager` can be used to manage individual `yarpdatadumper` services. It exists both as a console version and a version with user interface.

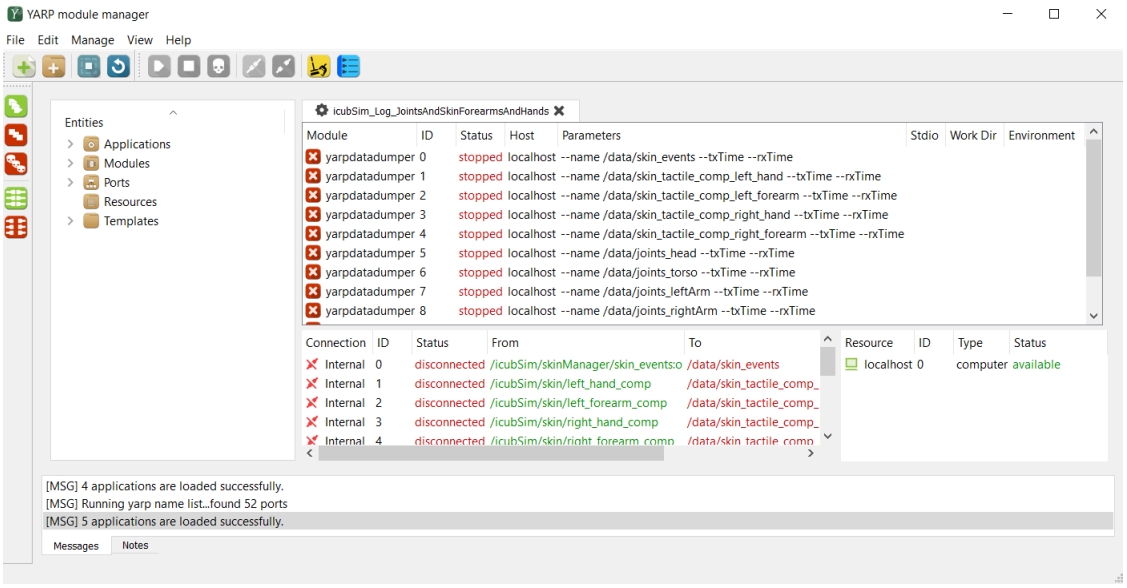


Fig. 3.2 An example of a running `yarpmanager` instance, with a loaded configuration file of multiple `yarpdatadumper` modules.

For our purposes, we have used the version with a user interface. There is a possibility to use a config file, which specifies individual modules - `yarpdatadumper` instances - each specifying a port from which the data should be read, and the location where the data should be dumped.

There was only one problem which needed to solve, and that was the data synchronization out of all 6 `yarpdatadumper` services - 2 for proprioceptive left/right arm, and 4 for touch info - forearms and hands, of both arms.

It is possible to select multiple rows in YARP manager and start multiple services at the same time. Despite that, there were small offsets in the timestamps across the dumped files, since the sampling of proprioceptive data is done every 10-30ms, and for touch/skin data the sampling is 30-70ms.

In order to have complete data batches - have all 6 values under one timestamp - we made an optimisation parser for the dumped files. We will briefly describe the process of parsing one file, the process was the same for the remaining five.

- For every line in the dumped file, round the timestamp to three decimal places (the timestamps were in UNIX format).
- Then, based on this timestamp, there were two options: either this same timestamp was already encountered (possible because of the rounding) , or it was new.

1. If it was new, it was remembered, and all the dumped data were assigned to this timestamp
2. If the timestamp already existed, then the values were averaged with the remembered values

After finishing process for all 6 dumped files, we ordered all the rows of data by the rounded timestamps.

For every such row, if it had all 6 values, we remembered it as a full data batch. If it had at least 3 values, then we looked one row up and down to check if the missing parts could be filled from there. If yes, then the batch was remembered, if not, the row was ignored and the program continued to the next row.

This process worked quite well, and after validating the data, there were no longer any error configurations produced.

Chapter 4

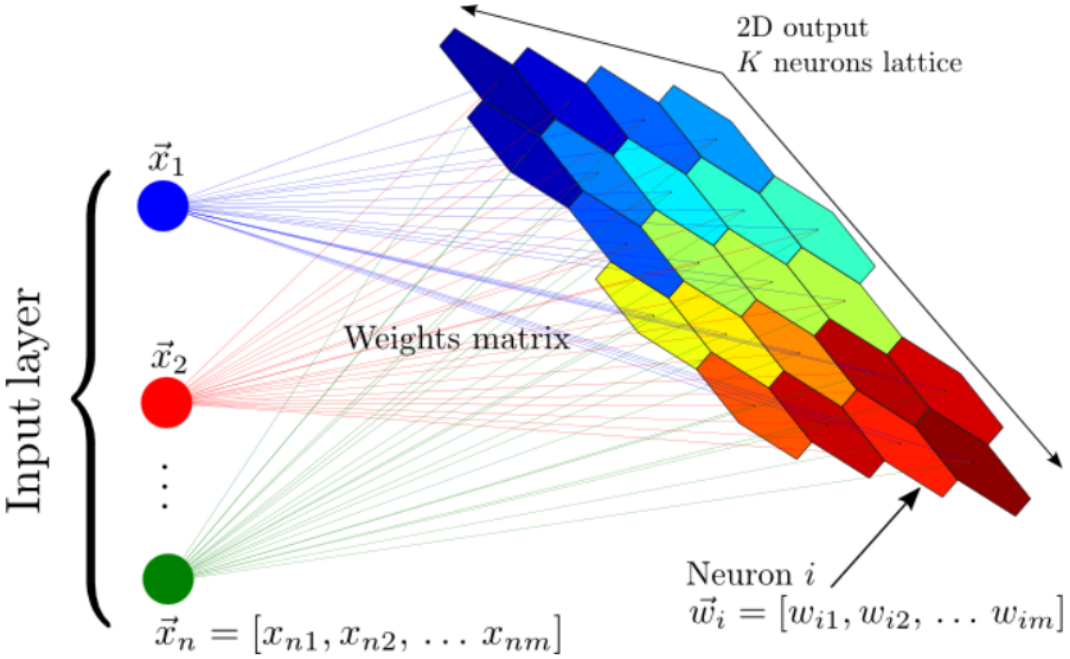
Neural network models

Here we will introduce the neural network models used in our approach to learning associations between the proprioceptive limb configurations and correct touch predictions. First, we will shortly summarize the well-know artificial neural network trained using unsupervised learning, self-organized map (SOM) (Kohonen, 1982).
Aside from SOM, we will also talk about a modification of it proposed in (Hoffmann et al., 2018) which, compared to traditional SOM, uses maximum receptive fields (MRF-SOM) for clustering touch information, and we will be explain how this approach works. From supervised learning, we will say a few sentences about multi-layer perceptron, and lastly, the model of Bidirectional Associative Learning (BAL) (Farkas and Rebrová, 2013).

4.1 Self-organizing maps

Self-organizing maps (SOMs) are widely used and popular artificial neural network models. They are readily simple and highly visual, effective for clustering and reducing the dimensions of the input data for the purpose of data analysis. They are used to represent high-dimensional input data in a form that has lower amount of dimensions - for example, a projection to a 2-dimensional grid of neurons.

SOMs differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space (Wang and Zhang, 2020). For this purpose, they are positioned in a grid-like, usually hexagonal or rectangular shape.



SOM

Fig. 4.1 An Overview of SOM neural network. Source: (Lan, 2018)

When positioned this way, they represent the topological space of the neural network where the *distance* between two neurons is the Euclidean distance between the input-to-hidden layer weights of each neuron.

As can be seen in Fig. 4.1, the network has only one layer of neurons, with the input layer fully connected to the 'competitive' layer, so every neuron has information about the value of all input data.

The training of the neural network is the competing among the neurons regarding which one of them will be adapted to the training vector the most. More specifically, which neuron will be the 'winner', and thus its position in the grid will be shifted to represent the training vector the most, and by how much much other neighboring neurons of this winner shift as well. The degree of how much the neighboring neurons shift is determined by the neighborhood function of the network.

Individual steps of training the SOM (Kohonen, 1982)) are following:

1. The initialization of neuron weights:

   The weights of neurons are which represent their position in the grid, their change being the individual neuron's shifting position depending on the training data. At the start of the training process, the weights are initialized to small values.

If we do know the training data, then positions from the training set can be chosen randomly. The learning rate α should be set to $\alpha < 1$, and a number representing the epochs for training e_{max} is chosen.

2. The selection of vector from the training set:

   In each epoch, it is important to iterate over all the vectors that make up the training set. Therefore, in every epoch, every training vector from the training set should be used exactly once. The selection of which one should be used is chosen randomly.

3. The selection of the winner neuron for the training vector:

   The winner neuron for the specific training vector is the one which will have the shortest distance to the training vector. As the metric for determining the distance, we use Euclidean distance:

$$i* = argmin_i||x - w_i|| \tag{4.1}$$

   where x is is the training vector, and w_i is the position of the i-th neuron in the space of the competing layer.

4. The adaptation of weights in the competing layer:

   The winner neuron is shifted towards the training vector, and its neighbors are shifted with him. The farther away the neighboring neuron is, the smaller the shift of its position.

$$w_i(t+1) = w_i(t) + \alpha(t) \cdot h(i^*, i) \cdot [x(t) - w_i(t)] \tag{4.2}$$

   where *$w_i(t)$* is the weight (position) of the i-th neuron, *$h(i^*,i)$* is the function calculating the distance between the neighbor and the winner neuron. This is calculated as a discrete:

$$h(i^*, i) = \begin{cases} 1 & \text{if } i \in N_{i^*}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

   or a continuous Gaussian neighborhood:

$$h(i^*, i) = \exp(-\frac{d^2(i^*, i)}{\lambda^2(t)}) \tag{4.4}$$

   where $d(i^*, i)$ is the distance of neurons i^* and i in the grid.

   The shifting of neuron weights is done every time after a winner neuron is evaluated. Based on the equation above, the weights are recalculated for of *all neurons*

in a defined area of N_{i*}. During the training process, the value of the learning rate is continuously reduced, as well as the area surrounding the winner neuron which is done by the parameter λ.

The training of the neural network can be though of as consisting of two phases. The first phase is doing large shifts of neuron weights, it serves to 'map' the whole network to the training data.

In the second phase, the shifting is smaller, because the goal is to fit the neurons to represent the data as accurately as possible.

The parameters of α and λ are determined as follows:

$$\alpha(t) = \alpha_s(\frac{\alpha_f}{\alpha_s})^{(\frac{e-1}{e_{max}})} \tag{4.5}$$

$$\lambda(t) = \lambda_s(\frac{\lambda_f}{\lambda_s})^{(\frac{e-1}{e_{max}})} \tag{4.6}$$

5. The length of the training - selecting the amount of epochs:

   The neural network will be trained for e_max epochs. Selecting this value of this parameters depends on the size of the training data. Simple rule would be to increase the value of e_max proportionally to the size of the training set - the larger the set, the larger the amount of epochs, in order to give the network time to 'fit' the data set.

   In the beginning, all weights of neurons are set to random values taken from the training set. Setting the learning rate (α) has a significant influence of both the speed and final accuracy of the trained neural network, initially its value is set close to 1. The parameter defining the neighborhood size λ is also set, and both λ and α are decreased over the process of the training.

4.2 MRF-SOM Model

For representing the areas of possible touches occuring on the forearms or hands, we used the modification of the classic SOM model.

MRF-SOM uses maximum receptive fields, which are biologically inspired since they are based on the observations of the somatosensory area of the brain in primates (Leyton and Sherrington).

These observations have attracted attention of other researchers, since this information was significant for better understanding of how the body is represented in the brain.

The brain as well as the individual areas of the body representations had been already discussed in the first chapter, along with the illustration of the "somatosensory homunculus" (Fig. 1.2). There we introduced how different body parts are represented in different areas of the somatosensory homunculus, as well as the topological organization properties of said brain area.

Since this is topological organization is known to occur in the physical brain of humans, primates and possibly in other animals, the question is how to transfer this structure to a neural network model for simulation and research purposes. That is what lead to the invention of MRF-SOM maps in (Hoffmann et al., 2018) which was designed to represent the skin sensations and stimuli of different body parts. In general, there were two opinions regarding the origins of these representation in the brain. One stated that, the fact that the topographic maps develop is native to the neural system in a broader sense, in which there are no specific neural activations required.

The second one has taken an inverse approach to the former, where the neural activations occurring during the development of the somsatosensory neural area are seen as being crucial to the process. This idea of two opposing view was written by Crair in 1999 (Crair, 1999).

The MRF-SOM model was designed with the second approach in mind - that the structure of neural activity is important. The outcome of the paper was to invent, create and train these modified SOMs, and test them against different parts of the training vectors to validate that different neurons are more reactive to different parts of the information. To achieve the outcome that neurons are reactive to only specific parts of the training vectors, a mask is introduced.

The mask in MRF-SOM is a binary vector of the same dimensions as the training vector. Every neuron in MRF-SOM will have this mask specified, which will define which part(s) of the input vector the neuron should ignore. These masks are not assigned to the neurons randomly - they are assigned to each neuron individually and specifically, based on what part of the input the neuron should react to.

We can see the division of the input space in Fig. 4.2. The division is also not entirely discrete, but the masks slightly overlap. The reason for this is that, should the areas be discrete, the map would become deformed. This enables a very slight changes of the SOM from the other areas, but still retains the activations to be grossly localized.

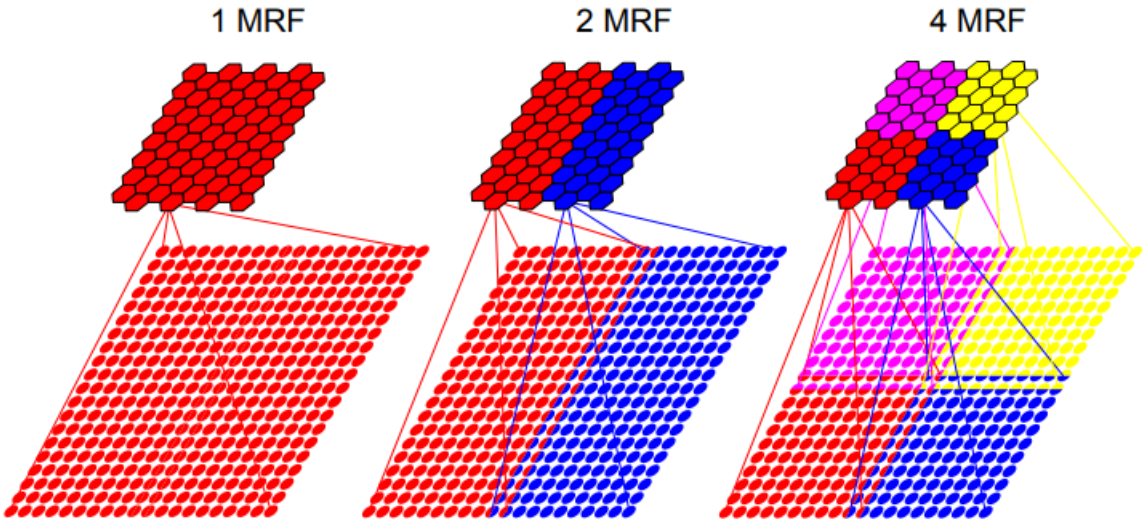


Fig. 4.2 MRF masks differentiated by color, separate the input space. In this example, there are 8x8 neurons at the top, 20x20 inputs (simulated taxels) at the bottom. The color code and the span of weight vectors mark the maximum receptive field size of every output neuron area. Taxels with multiple colors mark the overlap of maximum receptive fields (Hoffmann et al., 2018)

4.2.1 MRF-SOM training process

The training of MRF-SOMs is very similiar to the training process of SOM. However, in contrast to classical SOMs, MRF-SOMs are more specific - since they were invented to represent touch information - and thus there are some changes. The concrete training steps for training an MRF-SOM are following:

1. Neuron weights initialization:

   The weights of all neurons are initialized to small values. If the training data are known, then we can choose random training vectors. We set the learning rate $\alpha < 1$. The masks are initialized for each neuron, and the number of epochs from training e_{max}.

2. The selection of vector from the training set:

   In each epoch, all vectors from the training set are iterated through exactly once. Which one is selected during the iteration is random every time.

3. The selection of the winner neuron for the training vector:

   In MRF-SOM, the winner neuron is not found by calculating Euclidean distance as is the case in SOM, but the winner i^* is calculated by the dot product:

$$i^* = \arg\max(w_i^T \cdot x(t)) \tag{4.7}$$

The reason for this change compared to SOM are the masks, and the data.

MRF masks serve to zero the values of the input vector which fall outside of the range the neuron should be sensitive to, consequently making these values not influential in the selection of the winner.

4. The adaptation of weights in the competing layer:

$$w_i(t+1) = \frac{w_i(t) + \alpha h(i^*, i)x(t)}{||w_i(t) + \alpha h(i^*, i)x(t)||} \tag{4.8}$$

The entire process of weight adaptation can be summed in three steps:

(a) $full\_vector = w_i(t) + \alpha h(i^*, i)x(t)$

(b) $masked\_vector = full\_vector * mask_i$
where * means element-wise product of the mask and vector - which results in the 'adjusted' vector specific to the receptive field it belongs to. In the end, the change can be written also as:

(c) $w_i(t+1) = \frac{masked\_vector}{||masked\_vector||}$

4.3 Multilayer perceptron

Multilayer perceptron is a well-known type of neural network and is considered an universal approximator thanks to its nonlinear properties. This feedforward neural network uses supervised learning for its training. It is one of the most widely used network models.

A perceptron consists of multiple fully-connected layers of neurons, and each synaptic connection has its own weight. A multilayer perceptron consists of an input layer, at least one hidden layer and an output layer. In case a multilayer perceptron has multiple hidden layers, it is also known as a deep neural network.

Compared to a simple perceptron, Multilayer perceptron has improved computational power because of its multiple hidden layers and activation functions, which enables it to solve non-linear problems.

Forward pass of a perceptron is calculated in the following way:

$$h_k(n+1) = f\left(\sum_{j=0}^{n-1} w_{kj} h_j(n)\right) \tag{4.9}$$

$h_k(n+1)$ is the k-th neuron of the $(n+1)$th layer, f is a differentiable (activation) function, w_{kj} is the weight of a specific synapse and $h_j(n)$ is the activation value of j-th neuron in the n-th layer.

There are multiple activation functions that can be used, some of the most commonly used are for instance the sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.10}$$

or hyperbolic tangens. Before training, we assign small random values to the weights and set the learning rate $1 > \alpha > 0$. Perceptron is trained by supervised learning, which means that its output is compared to desired output. The calculated difference between the actual output y and the desired output d is the error, $(d-y)$. Weight adjustment of the output layer is calculated in the following way:

$$w_{ik}(t+1) = w_{ik}(t) + \alpha(d_i - y_i)f_i' h_k \tag{4.11}$$

where $w_{ik}(t+1)$ is the synapse weight in epoch $t+1$, α is the learning rate, d_i is the desired output, y_i is the calculated output and h_k is the activation of k-th neuron.
After adjusting the weights of output layer we need to train the other layers.

To train the input and hidden layers we use the error back-propagation algorithm.

$$\delta_k = \sum_{i=0}^{n-1}(w_{ik}\delta_i)f_k' \tag{4.12}$$

where δ_k is error of k-th neuron of a particular layer, δ_i are errors of the next layer propagated back. Afterwards, we update the weights as

$$w_{kj}(t+1) = w_{kj}(t) + \alpha\delta_k x_j \tag{4.13}$$

4.4 Bidirectional associative learning

Bidirectional Associative Learning (BAL) (Farkas and Rebrová, 2013) is a bidirectional neural network model that is biologically plausible in its design. The architecture is similar to a perceptron with one hidden layer, but the association of the two modalities happens simultaneously both forward and backwards. This is in contrast to classic Backpropagation, which is not a biologically plausible approach (O'Reilly, 1996)

Compared to a perceptron with one hidden layer, in BAl every synaptic connection between neurons has not one, but two weights. One of the weights is updated only

during a forward pass through the network, and the other only through a backward pass.

Since the architecture is bidirectional, it is possible to do a backwards association, naturally with the input and output vectors (and their corresponding dimensions) swapped. In our case this means not only predicting the touches from a proprioceptive configuration, but also potentially predicting the limb positions from a touch vector. This, however, is a one-to-many association which presents multiple challenges further described in Chapter 5.

In contrast to perceptron, BAL is learning based on local information. When we have two modalities between which we want to associate, then the model should learn how to represent this relationship as a shared information. The place for encoding these information is the hidden layer.

For BAL, one epoch of training also means randomly iterating over all data samples from the training set, and every data sample is passed both froward and backward through BAL. During the backward pass, the prediction is done based on the target vector from the data sample, not the one predicted from BAL. All activation values are remembered, and then the weights are updated. For every layer - the hidden and output layer in the forward, and the hidden and input layer in the backward pass - the weights are updated in the following way:

$$w_{ij}^F = w_{ij}^F + \alpha b_i^F (a_j^B - a_j^F) \tag{4.14}$$

where w_{ij}^F is the weight of a neuron in the forward direction, α is the learning rate, b_i^F is the pre-synaptic activation in the forward pass a_j^B is the post-synaptic activation in the backwards pass, and a_j^F is the post-synaptic activation in the forward pass.

The update of weights in the backward pass is done analogically (the 'F' and 'B' are exchanged).

Chapter 5

Proprioceptive–tactile association

5.1 Experiments

In chapter 5, we will describe all the experiments we've tried. First, we will describe and show the Self Organizing Map models for forearms and hands, the MRF-SOM models used for touch representation, and the results of trained BAL associators with different parameters.

Then, we will describe the final, complex neural network model which learns to represent which proprioceptive limb configurations lead to self-touch, and which do not. The last two experiments was examining the Hidden layer of BAL associators, where we observed and visualized the weights, and we tried to predict proprioceptive vector from a touch vector using the trained BAL associator.

Additionally, we also tried to create a simple Hebbian associator, with no hidden layer. Unfortunately, we couldn't get the model to associate the two modalities due to some problem with stabilizing the weights. After some time of not being able to fix the problem, we've abandoned the experiment due to time constraints, and so no reportable results were obtained.

5.2 Training and visualizing SOM models

For both right and left limb, we've trained two SOM models, separately for the hand and forearm. All were generated with a rectangular grid of neurons. For forearms, we've decided to represent the space as an area of 9 x 12 neurons, and for hands an area of 8 x 8 was chosen.

The models were trained on the dataset which we've described in chapter 3, with a 80/20 split of training and testing data, respectively. The accuracy of the model was evaluated by calculating a Quantization error for every SOM, for both training and testing data. Final training was done on 500 Epochs, with a starting $\alpha = 0.5$, which produced the best results with relation to the data and the size of the data set.

For visualizing the model, we've created an automatized script which can translate the SOM neurons (weights) to a corresponding proprioceptive configuration, load this configuration to iCub, and take a screenshot. Due to the larger size of neurons, only every second neuron was evaluated this way. The final outcome is a grid of screenshots topologically representing the trained SOM model. The visualisation can show which neurons were sensitive to which proprioceptive configuration of the specific limb part.

Fig. 5.1 Visualization a 6 x 5 neurons of SOM representing the left forearm positions the network learned. Topological organisation of the weights can be observed.

SOM	Quantization Error - training	Quantization error - testing
Left hand	0.0068	0.0112
Left forearm	0.02156	0.02177
Right hand	0.00683	0.01128
Right forearm	0.02174	0.02130

Tab. 5.1 The quantization error values of Final SOM models.

5.3 Training and visualizing MRF-SOM models

For representing the touch information, we have used the MRF-SOM architecture described in Chapter 4. For forearms, we chose 12 x 9 neurons, and for the hand 7 x 7 neurons to represent the touch area. For training the MRF-SOMs, we didn't use the data collected from iCub, as was the case for proprioceptive SOM models. The reason for this decision was to adequately represent all possible touches occuring on the hands and forearms - not just a subset of possible touch configurations collected from iCub.

For the sake of visualisation, we've re-used Martin's python script for the forearms. In case of hands, we've modified our own implementation of the visualisation, where the fingers are rectangular shapes instead of small dots, as was the case in the original thesis.

5.4 Data normalization

As was already described in Martin's thesis, he tried training the models on collected data from iCub, normalized to interval of [0, 1] for all joint values on the proprioceptive side, and the touch data transformed to vectors consisting of 0 or 1s instead of the ranges [0, 255] the simulator uses. In our case, we also approached data normalisation this way.

What we tried additionally were multiple different data scaling approaches on the proprioceptive side, and consequent training of the SOMs and BAL associators. In the simulator, iCub's limb joints have different degrees of freedom, as was already mentioned in Chapter 3. What we wanted to see is how different data scalings would influence the accuracy of the models, and which normalisation would be the closer to optimal.

We tried scaling the data in the following ways:

1. A simple constant (100) by which every join value in the proprioceptive data were multiplied. This didn't bring promising results, since both SOMs and BAL associators were not rescaled. Some joints have larger degrees of freedom, some smaller, and simply multiplying by a constant didn't change this fact.

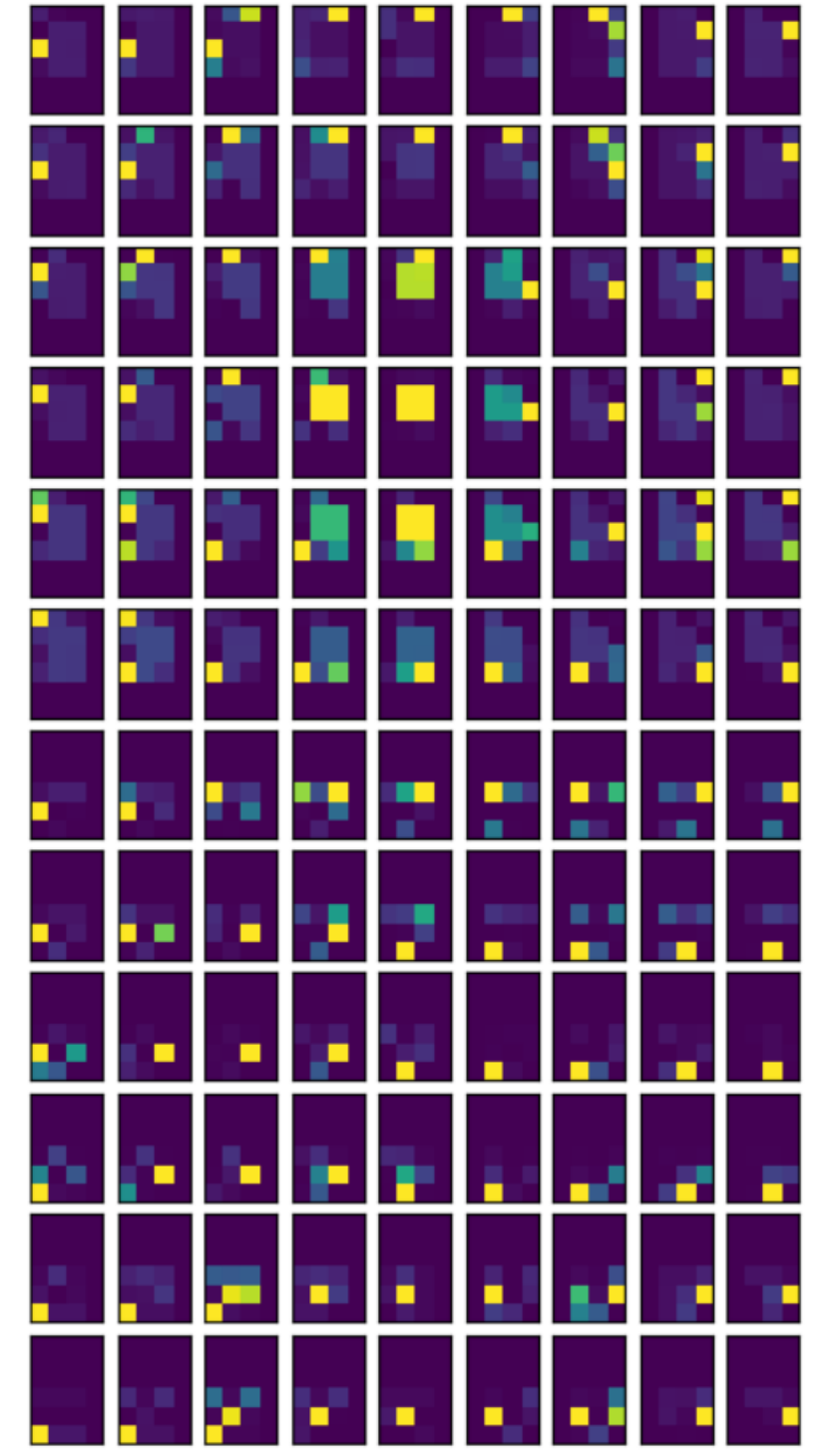


Fig. 5.2 Visualisation of touch representations of MRF-SOM , describing left forearm. Every rectangle correspond to a forearm consisting of 23 taxels in the simulator. Every point is differentiated by color, the brighter the color, the higher the intensity of touch. Yellow represents touch. The blue-green represents a possible touch, and the blue/purple represents non-touch. Topological properties can be observed. The MRF mask for forearms separates the area into four quadrants.

2. Normalizing all joint values to fall between [0, 1] by taking the global minimum

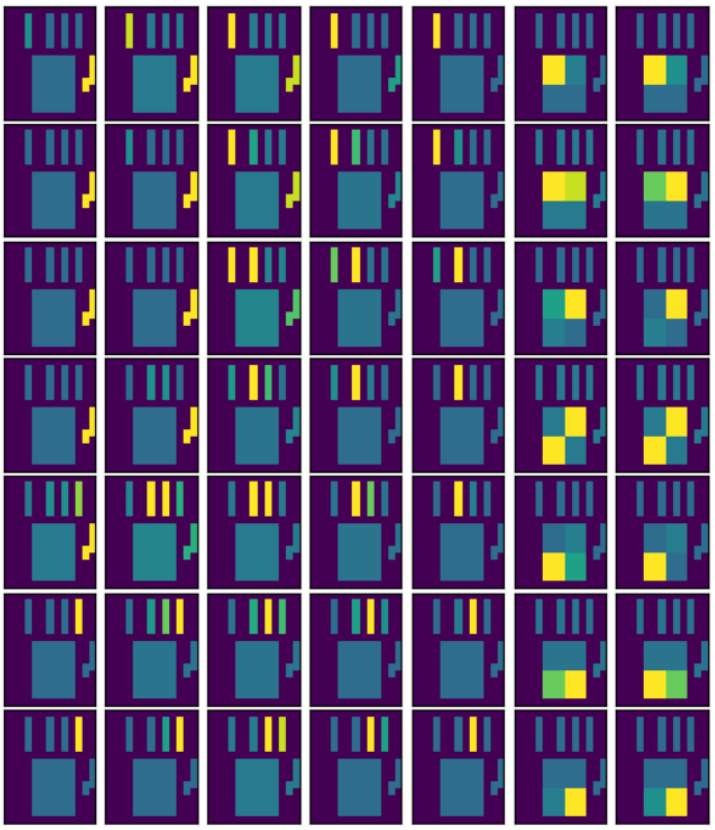


Fig. 5.3 The learned touch representations of a hand MRF-SOM, showing right hand. Every image shows a palm with fingers, consisting of 9 taxels total - 4 for palm, 5 for fingers. Every point is differentiated by color, the brighter the color, the higher the intensity of touch. Yellow represents touch. The blue-green represents a possible touch, and the blue/purple represents non-touch. Topological properties can be observed here as well. The MRF mask for hands separates map into two areas - the fingers and palm

and maximum possible value out of all joints. This ensured that all joints were represented by values [0, 1], but also that some joints would be represented on a smaller interval compared to other joints with more DoF.

3. Normalizing all joint values to fall between [0, 1] , but every joint was normalized independently. This preserved the ranges of all individual joints while simultaneously ensuring that all joints are represented in the space of [0, 1] values. This approach was significantly more successful, and was the one we used for all final trainings reported in this chapter.

5.5 Data representation

After the data were normalized, we will describe how both proprioceptive and touch data were further represented for training the BAL associators by using SOMs and

MRF-SOMs for prediction.

For every data sample from the normalized data set:

1. The sample was divided into two parts: the proprioceptive and touch information.
2. For the proprioceptive part, a winner vector was selected from each corresponding SOM, and was returned as one-hot-encoded vector. All 4 vectors were concatenated into a single vector containing exactly 4 ones.
3. For the touch part, similarly a winner of each MRF-SOM was retrieved and one-hot encoded. However, the touch vector always contained exactly 2 ones - depending on where the touch occurred (i.e right hand touched left forearm) for any data sample.

In other words, the data set was systematically separated into 4 different sets depending on where the touch occurred. Implicitly, we were training BAL associators only on touch data, since in Martin's work, he concluded that the model was significantly more successful when trained on touch data only.

This way, the normalized data was pre-processed into 4 different files used to train all the individual BAL associators, for each limb part.

5.6 BAL associators

We've trained 4 individual BAL associators, similarly to how they were trained in the original master thesis. We experimented with different parameters to see how their values influence the success of the model.

In general, the associators were trained on around 200-400 data samples. Also, for every associator, we measured the model accuracy by calculating the mean square error of the predicted output:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(d_i - y_i)^2 \tag{5.1}$$

where n is the dimension of the output, d_i is the expected output, and y_i is the target value on ith component. The mean square error will be evaluated in both forward and backward direction. Aside from that, we also state the classification accuracy, expressed in percentage.

5.6.1 Setting different thresholds on prediction

After each epoch, we evaluated the classification error of the model at that point in time. The prediction by making a forward pass has given us the predicted touch vector, with values ranging between [0, 1] . Since the target vectors are one-hot encoded, we needed to choose a treshold to decide which values should be 0 and which 1 when evaluating the predicted vector.

At first, the threshold was set to 0.5 which lead to BAL associators predicting a significant amount of errors where the prediction was evaluated as a zero vector (no touch occuring), even though the target was a touch vector. Afterwards, we raised the threshold to 0.6, which produced nearly no change.

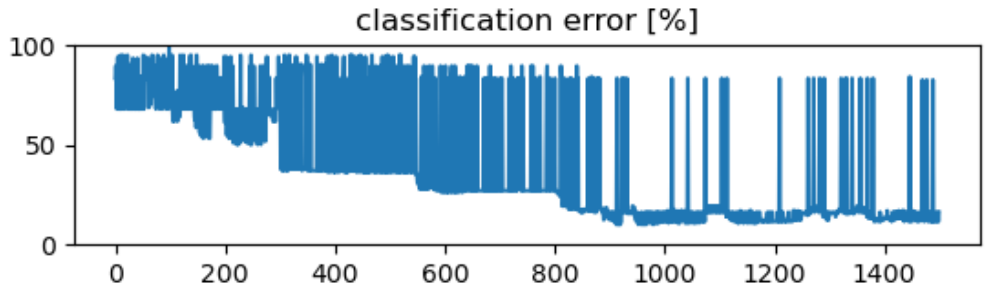


Fig. 5.4 The classification errors produced when predictions were classified as no touches instead of touches, threshold=0.6. Showing BAL Associator for right hand.

When observing the predicted vectors more closely, we noticed that those which should be evaluated as ones were often in the range between 0.4 and 0.5. This was most likely due to the vectors being sparse, and so the values were lower than average as well. The

threshold set to 0.4 turned out to be the most optimal parameter, as 0.3 led to the opposite problem of incorrectly predicted touches where none existed.

5.6.2 Weight enhancements during training

Additionally to changing the threshold when predicting the touch vectors, we tried enhancements to weight changes during training. This was applied to both the forward and backward pass during training.

The basic case was leaving this threshold equal to 0.5, which had similar problems as we've already discussed above.

Another approaches were flooring and enhancing. Flooring was setting the value to either 0 or 1 if a certain threshold was reached: 0.75 for 1, and 0.35 to 0. When running the training with this modification, it proved to be too aggressive; the steep change would introduce a larger error in some cases, which lead to larger corrections, and caused the training to oscillate.

Enhancing was essentially a milder flooring: instead of flooring the values to 1 or 0 directly, the values were either increased by a constant σ if a value was at least 0.65, or decreased by σ if the value was lower than 0.35. Naturally, we prevented the values from falling outside of the interval [0, 1] when enhancing.

The conclusion was that with a smaller σ, the model was trained somewhat slower, but no oscillation occurred, while the accuracy has stayed the same or slightly increased. We used the value $\sigma = 0.15$ for final training of the associators.

5.6.3 Model training and final accuracy

Initially, we tried to replicate the results that were achieved in the previous master thesis, but with our generated data set. The model was not learning however, both metrics (mean square error and classification error) showing either oscillating or not converging to stable, low values.

What we realized was that due to the size of our data set being approximately three to four times larger than the data set the associators were trained on originally, the size of the hidden layer was insufficient and thus was incapable of capturing all the proprioceptive and touch associations.

Original count of neurons present on the hidden layers was set to 110. We tried 150, 200, 250, 300 and 350. Out of all these, 300-350 seemed to produce the best result as further increase has not improved the final training/testing accuracy.

By increasing the number of neurons present on the hidden layer and tweaking other parameters mentioned in the previous subsection, the model looked to be trained very well, the training accuracy between 88-95% on all 4 associators similarly.

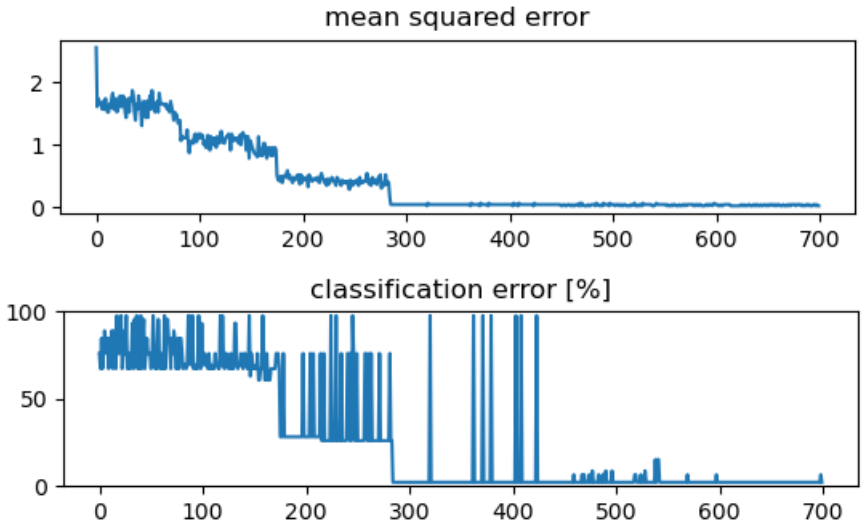


Fig. 5.5 Graphs presenting the mean square error and classification error decreasing during training.

Unfortunately, while the training accuracy in Fig. 5.5 was quite high, when evaluating the testing accuracy we observed it to be very low (between 10-40% on average). This meant that the the model was prone to overfitting on the training data, and would not generalize well.

We addressed this issue by introducing a condition to the training: During the training, after each epoch we evaluated the accuracy the model was attaining on the testing set at that point in training. If the testing accuracy was at least 80% and the training has reached at least half the amount of total epochs specified, the training of the model would be stopped at this stage.

This has introduced more balance to the model regarding its generalization capabilities, but also simultaneously reduced the training accuracy.

The main problem with not reaching a more precise accuracy can in our case be explained by the data. Since everything was automatically generated by our babbling algorithm, there were some proprioceptive configurations which were almost touching, but were not touching in reality. As we can see the accuracy was higher for both training and testing data for the hands in comparison to the forearms. This is most likely due to

Associator	average training accuracy (%)	average testing accuracy (%)
Left hand	86.42	80.28
Left forearm	76.27	80.31
Right forearm	77.17	80.49
Right hand	84.12	81.54

Tab. 5.2 Table displaying the average of all training/testing accuracies, over 10x model training, with 350 neurons on the hidden layer, 800 epochs

higher variation in proprioceptive configurations for forearms. As described in chapter 3, the babbling was not changing the degrees of freedom for hands nearly as much as it did for the forearms. This in combination with the many to one relationship between proprioceptive configurations and touch occurrence results in lower success rate.

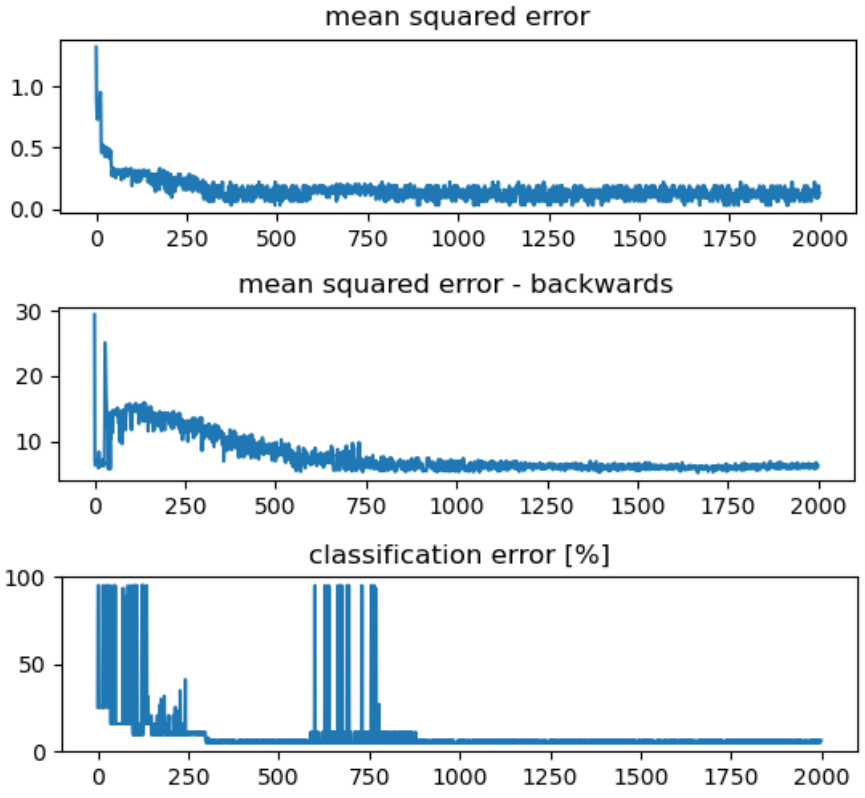


Fig. 5.6 The training progress of BAL associator for right hand.

As we can see in Fig.5.6 the training condition we introduced influenced the results. Classification error oscillates around 10-20%, however, the corresponding testing accuracy was around 80%.

5.7 Investigating hidden layers

This experiment was focused on the properties of the hidden layer in BAL associators. The aim was to investigate whether there are some neurons which are more active for some inputs compared to others, and whether the differences are pronounced - for instance, some neurons very close to 1, and others almost 0. Evaluating this for a data sample in both forward and backward pass, we wanted to see whether we could observe similar activations. This result was represented as a heat map, each pixel on the X axis representing a neuron, and the Y axis representing a data sample which was passed through the layer.

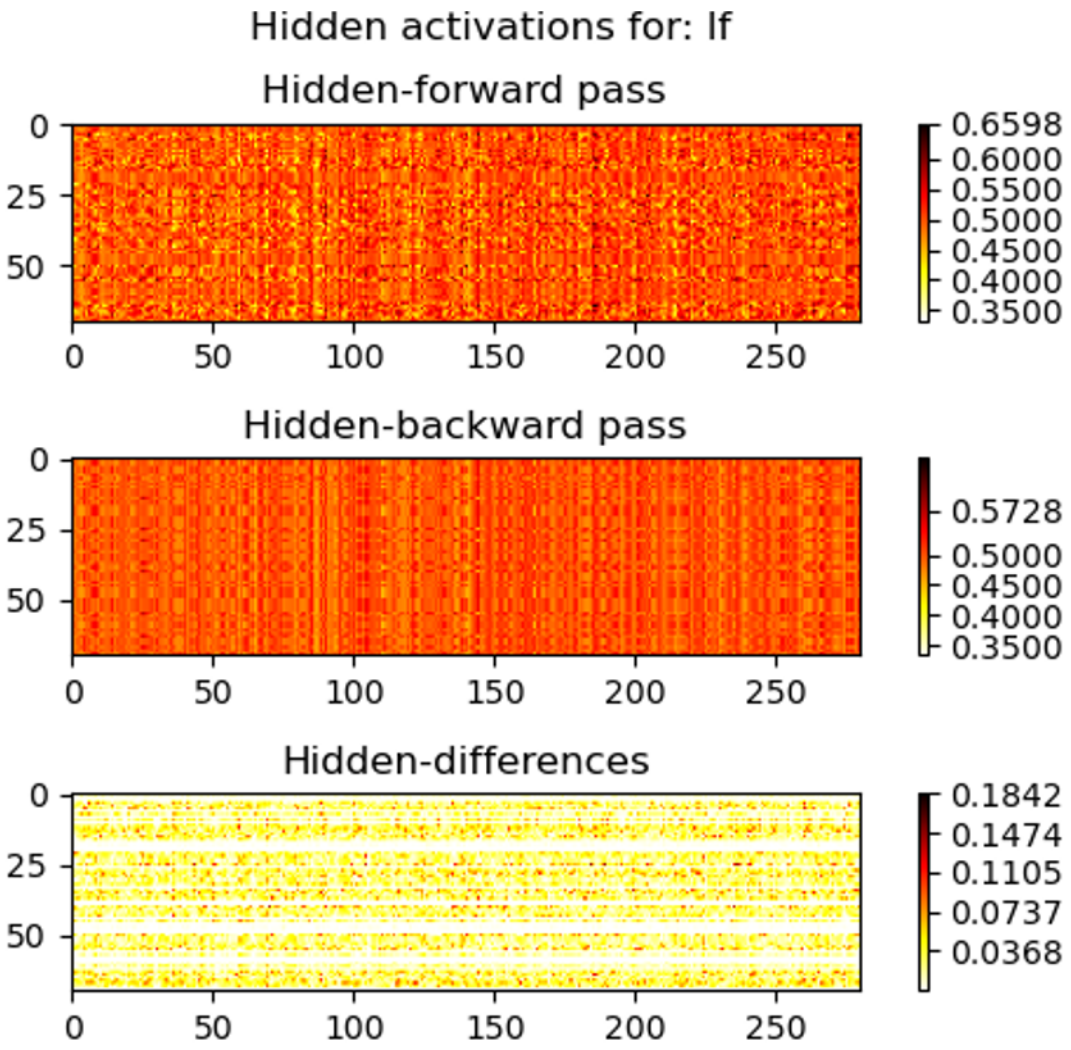


Fig. 5.7 The activations on the hidden layer of the left forearm. The heat map shows 70 random data samples. The Y coordinate are the data samples, while the X coordinate represent individual neurons, of complete 300, which was the size of the hidden layer.

Additionally, we wanted to see how responsive each neuron was to the input, both through a forward pass, and a backward pass. How responsive in this case refers to the neuron's activation/value. This was expressed as a graph of points, each point

representing the mean value of the activation, with a vertical line intersecting it. The vertical line represents the minimal and maximal value the neuron produced. Therefore, the shorter the line, the less varied the neuron's activation was, and vice versa.

As can be observed, the proprioceptive activations have more variations compared to the touch neurons. This can be explained by the fact that, the touch is a more concrete information - it either occurs or not. Furthemore, since there can be multiple proprioceptive configurations associated with the same touch, the variation will always be greater for proprioceptive information.

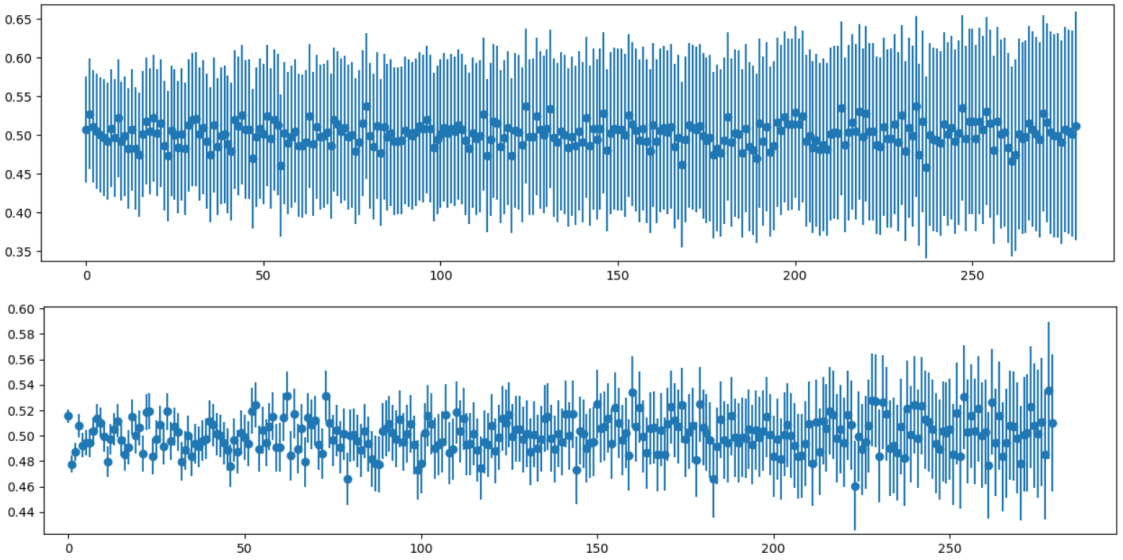


Fig. 5.8 The activations of all 300 neurons on the hidden layer, for the left forearm, sorted by the least to most varying neurons/activations. The top graph represents the activations for the proprioceptive data, the bottom graph for touch data.

5.8 Predicting proprioceptive configurations from a touch vector

One property of the model we wanted to test was its ability to predict the input vector from target vector. In other words doing the backwards prediction from one hot encoded touch vector to its proprioceptive representation. Naturally, this problem is non-deterministic one touch configuration can be represented by multiple proprioceptive configurations of the limbs. It was found that in such cases BAL should predict the mean value as the target (Farkas and Rebrová, 2013).

For this experiment we tested it the following way:

1. We randomly selected a touch configuration where left hand was touching the right forearm.
2. Then we ensured that we could retrieve MRF-SOM winners for this touch configuration.
3. We made a backward pass through BAL using concatenated winner vectors as input.
4. This way we received a one hot encoded proprioceptive configuration as the output. Then we used SOMs to find the winners for forearms and hands. This way we received a specific proprioceptive configuration as the output.

Fig. 5.9 Comparison of the original proprioceptive configuration (upper image) with predicted configuration (lower image) using touch data.

As expected the predicted proprioceptive configuration rarely matched the original limb positions. Additionally, in some cases the predicted position of the limbs resulted in a slightly different touch, as can be seen on Fig. 5.9.

Chapter 6

The complete model

This chapter introduces the final, complex model consisting of all the neural network models introduced in this work. The goal of this experiment is to see how reliably the model can predict whether a proprioceptive configuration ends in a touch, and if it does, where.

The final model is quite complex, and is visualized in Fig.6.1. The main difference compared to the complex BAL model in Pecen (2019) is that we use four SOMs for proprioceptive representations instead of only two.

6.1 Model scheme

The model consists of three parts. The first part is the filter. The second part transforms the proprioceptive and tactile inputs to one hot encoded vectors via SOMs and MRF-SOMs, respectively. The third part associates transformed data using BAL model.

BAL always associates full proprioceptive information(both arms) with one MRF-SOM (touch on one part of the limbs) in this complex model.

Since we've already explained how we trained SOMs and MRF-SOMs in Chapter 5, we will only introduce the filter.

6.2 The filter

The one additional neural network model used in the complex schema is a filter for detecting touch configurations. The purpose of this filter is to pass the current data sample

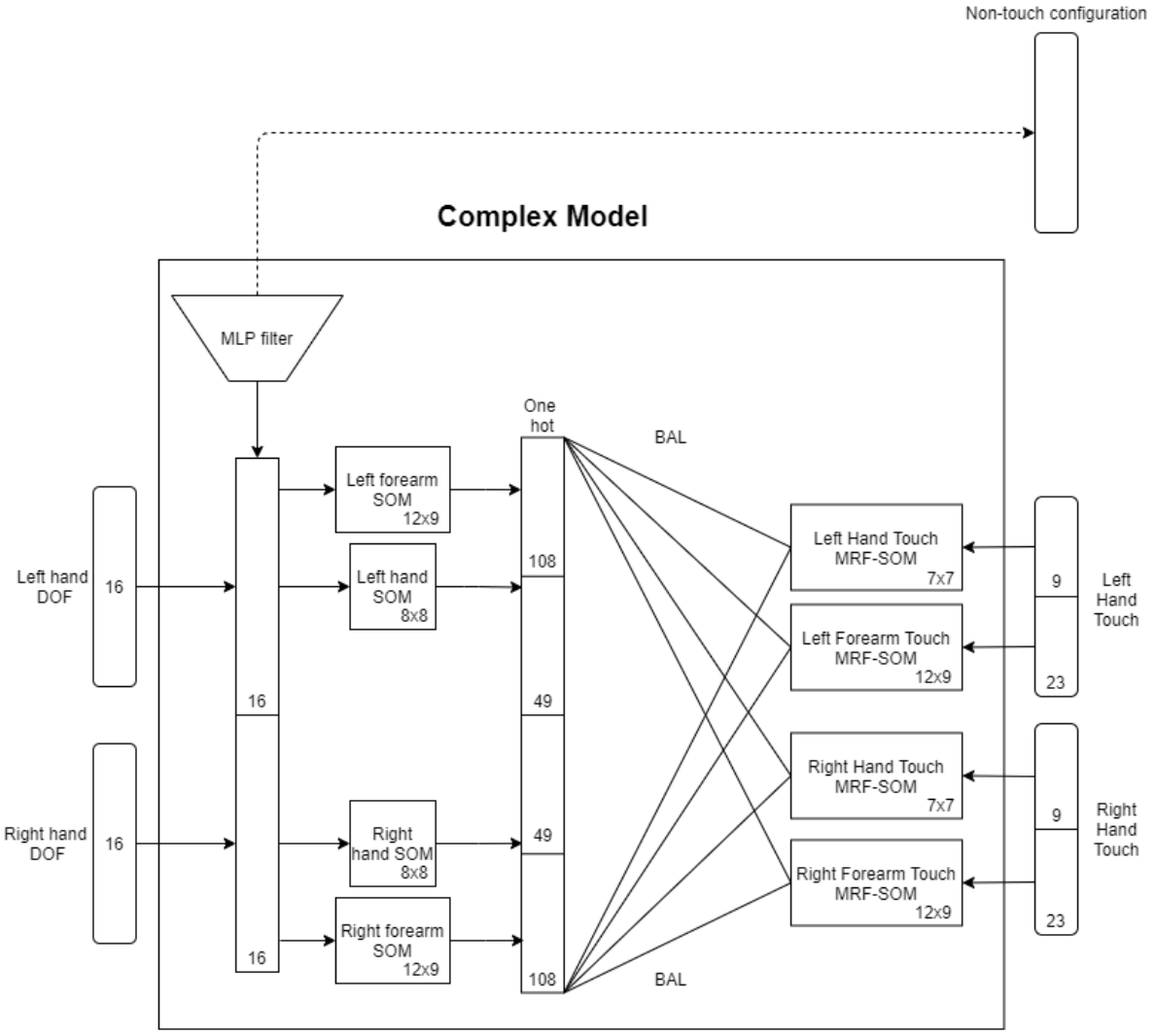


Fig. 6.1 Schema showing the complex model used for association of proprioceptive and tactile inputs.

further through the complex model only when the data sample is a touch configuration. If not, it should be filtered away, and no further predictions are necessary.

The filter was implemented using TensorFlow and Keras Python libraries. The architecture of the filter is a multi-layer perceptron with 2 hidden layers. The dimensions of layers are 32, 64, 70 and 1, respectively. We used 80:20 split between training and testing data. The touch information was simplified into 0 or 1 instead of the full touch vector. We trained the perceptron over course of 250 epochs. The training accuracy was around 98-99% and the testing accuracy was around 90%.

6.3 Testing the overall model

For the final testing of the whole complex model, we load all the SOM, MRF-SOM and BAL associator models, and the filter. We've tried testing the model on various data sets.

First, we used a small set consisting of only non-touch data, the same one used in Pecen (2019). This was mostly to check whether the filter would correctly exclude these configurations. These data were created manually, using iCub and the MotorGui interface. The results were a 100% success - the multilayer perceptron has correctly identified all 60 configurations as non-touches.

However, it is still important to report the filter accuracy on the larger, generated data set. Especially for configurations which do not end in touch but are very close to touching, and vice versa.

Data set	Success	Correct	Ghost	Missed	Other errors
notouch.data	100%	60	0	0	0
T0-dumped.data	80%	4261	823	84	133
smaller.data	74%	197	43	7	20

Tab. 6.1 The results of different test runs. The first data set contained only non-touch data. The second was generated by the babbling, and dumped through yarpdatadumper

In Table 6.1 , we report results of multiple test runs. The explanation for the terms in the headers are following:

- Correct - amount of data samples for which every part of the complex model predicted the correct result.
- Ghost - cases where MLP filter has incorrectly evaluated non-touch configurations as touch ones
- Missed - opposite to Ghost touches. Configuration resulted in a touch, but it was incorrectly filtered out.
- Other errors - configurations in which at least one incorrect prediction was made, either by an MRF-SOM or the BAL associator.

6.4 Winner neighbors counted as correct

We tried to re-run the tests with one change. In table 6.2, the change made is in evaluating the validity of touch predictions.

Originally, the one-hot vectors predicted by BAL needed to equal MRF-SOM outputed vector. In this case, we also look at the neighbors of the MRF winner, and should the prediction from BAL match either one of the neighbors, we evaluate it as a correct prediction.

Data set	Success	Correct	Ghost	Missed	Other errors
test-notouch.data	100%	60	0	0	0
test-T0-dumped.data	83%	4383	823	84	11
test-smaller.data	80.2%	214	43	7	3

Tab. 6.2 The results of different test runs. MRF-SOM neighbors of the winner count as a correct prediction.

As can be observed in 6.2, this has improved the results, but only slightly, by 2-3% on average. Ultimately, it is evident that the limitations in the final model lay in the BAL associators, which were shown to attain the accuracy around 80-85% during training and testing. The most significant but complicated problem lies in recognizing very similar proprioceptive configurations in which some are touch, but some are not.

Conclusion

The outcome of the thesis was a complex neural network model used for learning and representation of proprioceptive information of iCub. The model consists of three parts: four SOM models representing proprioceptive information of iCub's arms, a filter which determines whether the proprioceptive configuration occurs in a touch on one of the arms, and lastly the association of proprioceptive information with touch information represented by MRF-SOM maps. The mapping of these modalities is achieved via four BAL associators. The individual models were trained separately before combining them to the final complex model.

For input data collection, used for iCub training, we have tried two different babbling algorithms. The first approach was a modification from an existing babbling algorithm, which was changing individual joints in relation to time to a sinusoid function. The second was designed by us and was ultimately used to generate the final dataset, which consisted of thousands of data samples.

Additionally, we performed additional experiments that investigated the properties of BAL hidden layer, the activations of hidden layer's neurons and their variation, and observing the backwards prediction capabilities of trained BAL.

Ultimately, we succeeded in implementing a way to acquiring many data, but there is room for improvements regarding the classification capabilities of BAL, as we couldn't get over 90% in the final testing of the complex model.

Bibliography

Crair, M. (1999). Neuronal activity during development: permissive or instructive? Current opinion in neurobiology, 9(1):88—93.

Del Prete, A. (2013). Control of Contact Forces using Whole-Body Force and Tactile Sensors: Theory and Implementation on the iCub Humanoid Robot. PhD thesis.

Farkas, I. and Rebrová, K. (2013). Bidirectional activation-based neural network learning algorithm. In TODO.

Goble, D., Lewis, C., Hurvitz, E., and Brown, S. (2005). Development of upper limb proprioceptive accuracy in children and adolescents. Human movement science, 24:155–70.

Hearn M, Crowe A, K. W. (1989). Influence of age on proprioceptive accuracy in two dimensions. Percept Mot Skills, pages 811–818.

Hoffmann, M., Straka, Z., Farkaš, I., Vavrečka, M., and Metta, G. (2018). Robotic homunculus: Learning of artificial skin representation in a humanoid robot motivated by primary somatosensory cortex. IEEE Transactions on Cognitive and Developmental Systems, 10(2):163–176.

Jafari, M., Aflalo, T., Chivukula, S., Kellis, S. S., Salas, M. A., Norman, S. L., Pejsa, K., Liu, C. Y., and Andersen, R. A. (2020). The human primary somatosensory cortex encodes imagined movement in the absence of sensory information. Communications Biology, 3(1):757.

Jamone, L., Metta, G., Nori, F., and Sandini, G. (2006). James: A humanoid robot acting over an unstructured world. In 6th IEEE-RAS International Conference on Humanoid Robots.

Kandel, E. and Schwartz, H. (2000). Principles of Neural Science. 4 edition.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. Biological Cybernetics, 43:59–60.

Lan, H. (2018). Analyzing climate patterns with self-organizing maps (soms).

Laszlo, J. I. and Bairstow, P. J. (1980). The measurement of kinaesthetic sensitivity in children and adults. developmental medicine and child neurology. Developmental medicine and child neurology, (22(4)):454–464.

Leyton, A. and Sherrington, C. Observations on the excitable cortex of the chimpanzee, orang-utan, and gorilla. Experimental Physiology, 11:135–222.

Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., and Montesano, L. (2010). The icub humanoid robot: An open-systems platform for research in cognitive development. Neural Networks, 23:1125–34.

O'Reilly, R. C. (1996). Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. Neural Computation, 8(5):895–938.

Pecen, M. (2019). Akvizícia proprioceptívno-dotykových reprezentácií tela u humanoidného robota. Master's thesis, Comenius University in Bratislava.

Penfield, W. and Rasmussen, T. (1950). The Cerebral Cortex of Man: A Clinical Study of Localization of Function. MacMillan.

Proske, U. and Gandevia, S. C. (2012). The proprioceptive senses: Their roles in signaling body shape, body position and movement, and muscle force. Physiological Reviews, 92(4):1651–1697.

Rochat, P. and Striano, T. (2000). Perceived self in infancy. Infant Behavior and Development, 23:513–530.

Roncone, A., Hoffmann, M., Pattacini, U., Fadiga, L., and Metta, G. (2016). Peripersonal space and margin of safety around the body: Learning visuo-tactile associations in a humanoid robot with artificial skin. PLOS ONE, 11(10):1–32.

Ronthal, M. (2004). Textbook of Clinical Neurology. Wolters Kluwer Health, 2 edition.

Seefeldt, V. and Haubenstricker, J. (1982). Patterns, phase, or stages: An analytical model for the study of developmental movement. The development of movement control and coordination, pages 309–318.

Sherrington, C. S. (1907). On the proprio-ceptive system, especially in its reflex aspect. Brain, 29(4):467–482.

Wang, S. and Zhang, X. (2020). Analysis of self-organizing maps (som) methods for cell clustering with high-dimensional oam collected data. 2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA), pages 229–233.