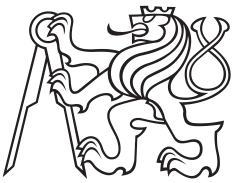
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Artificial Skin Calibration for the Nao Humanoid Robot Using "Self-touch"

Lukáš Rustler

Supervisor: Mgr. Karla Štěpánová, Ph.D.
Supervisor–specialist: Mgr. Matěj Hoffmann, Ph.D.
Field of study: Cybernetics and Robotics
May 2019

Acknowledgements

I would like to thank everyone who helped me with this work. Firstly, to Karla Štěpánová, who supervised this thesis and helped me with my problems and questions, and also spent a lot of time with the correction of my mistakes. I also want to thank Matěj Hoffmann, who gave me the opportunity to work on this project, answered my questions and helped me with the formal side of this thesis.

I would also like to thank my family for being continuous support through all the years and to my friends for their encouragement. And finally, I thank CTU in Prague for being such a good *alma matter*.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of the information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

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

V Praze dne 24. května 2019

.....
Lukáš Rustler

Abstract

This work deals with the problem of automatic calibration of the positions of arrays of tactile sensors placed on a robot body. In this case, the artificial skin is composed of triangular modules and mounted on the Nao humanoid robot. Accurate information about the actual placement of the tactile system on the robot is not available. This work presents a framework for calibration which exploits "self-touch" configurations, when we assume we know the kinematic structure of the robot and have approximate coordinates of the skin available. We collected a dataset with about 1000 activations for each of the self-touch configurations and we use the pressure centres of these activation to calibrate the skin. Through a series of experiments we found that the best approach is to calibrate every part of the artificial skin individually and that sequential calibration is superior to simultaneous calibration of all skin parts. We accomplished the relative error under 3 mm in the position of individual sensors.

Keywords: artificial skin, calibration, tactile sensing, self-touch, robot kinematics, Nao, humanoid robots, kinematic parameters, body representation, robot self-touch, forward kinematics, robot sensing systems

Supervisor: Mgr. Karla Štěpánová, Ph.D.

Abstrakt

Tato práce je zaměřena na problém automatické kalibrace polohy systému dotykových senzorů na těle robota. V našem případě je kůže tvořena trojúhelníkovými moduly, které jsou nasazeny na humanoidním robotovi Nao. Nemáme k dispozici informaci o přesné poloze snímačů na robotovi. Práce představuje framework pro kalibraci, který využívá sebedotykových konfigurací za předpokladu, že známe kinematickou strukturu robota a alespoň přibližné koordináty kůže. Nasbírali jsme dataset, který obsahuje okolo 1000 doteků pro každou sebedotykovou konfiguraci a pro kalibraci využíváme středy těchto aktivací. Pomocí serie experimentů jsem objevili, že nejlepším přístupem je kalibrovat zvlášť každou část umělé kůže, a že sekvenční kalibrace má lepší výsledky než současná kalibrace více částí kůže. Ve výsledku jsme získali relativní chybu v poloze sensorů menší než 3mm.

Klíčová slova: umělá kůže, kalibrace, snímání doteků, sebedotyk, kinematika robota, Nao, humanoidní roboti, kinematické parametry, reprezentace těla, robotický sebedotyk, dopředná kinematika, robotické systémy pro vnímání

Překlad názvu: Kalibrace robotické kůže Nao robota pomocí "sebedotykových" konfigurací

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

As the number of humanoid robots rises, the demand for their collaboration and interaction with humans is growing. Together with visual perception, tactile sensing is widely used as a useful source of robot's feedback.

Survey about artificial skins through time can be read in [1], but the most relevant for our work are tactile arrays for humanoid robots. On of them was introduced in 2006 by Ohmura et al. [2]. Another type of tactile sensors were created by Mittendorfer and Cheng in 2011 [3] and the skin constructed by the same authors with the use of their hexagonal modules was described in 2012 [4]. Triangular modules were developed by Cannata et al. in 2008 [5]. The another design of the triangles was presented in 2013 [6]. Skin formed by these modules is used on the iCub as well as on the Nao robot used in this work.

The majority of the artificial skins is created by combining small modules into bigger patches. The advantage of this approach is cheap, fast, and simple construction of these modules and the possibility of deployment on almost any robot. Even robots which were originally designed without tactile sensing can be additionally equipped with artificial skins. Examples of electronic skin integration into existing robots is in [7].

Except for a few examples like the iCub robot from version v2.0 [8], the skin is being added in the aftermath. But adding the skin *a posteriori* also poses many challenges. Maybe the most challenging and at the same time the most common problem is estimation of the position of the sensors on the robot body. To make use of the signals of tactile sensors, precise information about the arrangement of the sensors on the robot's body is needed. To account for manufacturing variability and changes of the skin during the robot's operation, automatic (re-) calibration methods are essential. As we have experienced by ourselves, the malfunction of the modules can occur and they must be changed and it is nearly impossible to place the new ones on the exact same location. Adding the skin to the robot is not just a matter of scientific research, but it is used in the industry as well. Dean-Leon et al. [9] presented a system for rapidly deployable robot with the use of auto-calibrated multi-modal robot skin.

1.2 Goals

The goals of this thesis were:

- Analyze skin attachment and create a hierarchical model of the robot

- Create a method/framework for unsupervised self-calibration using touch information
- Create tools for visualization of the configurations, etc.
- Find out what is the best approach to the calibration

1.3 Related work

The basic principle of robot kinematic calibration is the *open-loop* method when the end-effector configuration is observed by some external measurement (typically visual system). An overview of these methods can be found, for example, in [10]. There are several cases, where laser pointer was used as external measurement ([11], [12]), but more often on humanoid robots, the camera of the robot is used as the additional source of information. Online methods to calibrate humanoid torso with visual feedback were presented in [13] or [14]. If these external sensors are modeled as an additional joint, the kinematic chain can be closed. In this case, closed-loop calibration (overview again in [10]) is performed.

With development of artificial skins, new methods become available. Mittendorfer and Cheng [15] presented an approach that uses accelerometers distributed along the robot's body on the skin they developed [3], to calibrate DH parameters of the robot. Accelerometers are frequent tools for calibrations, as other approaches with the use of this sensor were presented by Guedelha et al. in [16] or Yamane in [17]. Accelerometers were also used by Mittendorfer and Cheng in [18], where the authors presented an approach to reconstruct the 3D surface of the robotic body equipped with the artificial skin.

A relevant robotic platform for the development of calibration is the iCub robot [19], which is equipped with stereo cameras, inertial sensors, force/torque sensors, and artificial skin. The Nao robot used in this work was uniquely equipped with the same skin technology [6]. Most of the scenarios mentioned above required at least some prior knowledge of the robot's environment, which is not very practical, which motivates automatic self-contained calibration. One of the approaches was described by Roncone et al. [20]. With motivation in biology, the paper presents a way where the kinematic structure can be calibrated by touching itself (so-called *self-touch*). This is another way how to "close" the loop and our work is also based on this principle, even though it is not as autonomous. The feedback from the camera and tactile sensors can be combined, which opens more accurate ways of calibration. Comparison of using different combinations of chains was described in [21].

The artificial skin is surely beneficial feedback, but it needs to be calibrated as well to be a suitable source of information. The biggest inaccuracies in terms of position come from manual placement of the skin arrays on the robot. One approach was described by Cannata et al. [22], where the calibration problem is formulated as the maximum likelihood mapping problem in 6D space. Another paper which deals with this problem was written by Del Prete et al. in [23]. This paper use force/torque measurement to estimate the position of the tactile sensors. Mittendorfer et al. used calibrated monocular camera to estimate the homogeneous transformations between multiple skin patches. Albini et al. [24] proposed a method that combines RGB-D camera with self-touch of end-effector and artificial skin. We will compare with the last three mentioned papers in the Result section of this work.

1.4 Contribution

Motivated by a real engineering problem of calibrating the artificial skin in unknown positions on a robot, this work investigated the methods for automatic calibration of such skin arrays through self-touch configurations. To this end, a number of components were needed first. Communication and data collection methods for the robot (NAOqi) and the artificial skin (YARP) had to be set up. Datasets of approximately 1000 touches for each of the selected configurations (hands touching the torso and hands touching the head). We also developed pipeline to parse the dataset, pair activations on different skin parts, transform them into the base frame and analyze statistics. Real-time visualization of gathering of the dataset was created.

For the calibration itself, we created a first version of a multirobot calibration framework, which could allow to simplify the process of calibration for nearly any robot. Part of this framework is a pipeline for automatic calibration of the pose of the artificial skin mounted on the Nao robot. The pipeline allows to calibrate each part of the skin on its own or all skin parts simultaneously. The pipeline can combine the gathered datasets and use them for more precise calibration.

Through a series of experiments we found that the best approach is to calibrate every part of the artificial skin individually and that sequential calibration is superior to simultaneous calibration of all skin parts. We accomplished the relative error under 3 mm in the position of individual sensors.

The works [23, 25, 24] are most related to this work, as they deal with the same artificial skin placed on humanoid robots. Roncone et al. [25] and Albini et al. [24] employ self-touch. However, Roncone et al. [25] deal with the opposite problem: using artificial skin to calibrate robot kinematics. Conversely, Del Prete et al. [23] and Albini et al. [24] employ other knowledge (robot dynamics model and force/torque measurements [23]; kinematics and self-touch [24]) to estimated the pose of the skin. In this work, self-touch is not achieved autonomously by the robot (like in [25, 24]) but robot is animated manually. Our focus is instead on the calibration of large areas of the skin and how best accuracy can be achieved by combining prior knowledge and constraints arising from the self-touch configurations.

Chapter 2

Materials and Methods

All scripts and directories mentioned in this and all other sections are located relatively to the root of the main *gitlab* repository [26], unless otherwise stated.

For our work, we have used a Nao robot covered by artificial skin on forearms, torso and head. The detailed description of the skin and its placement, as well as robot kinematics, is described in following subsections.

2.1 Robot platform

2.1.1 NAO robot

For our work, we have chosen the NAO humanoid robot from Softbank Robotics (formerly Aldebaran Robotics). This robot is worldwide known as an excellent tool for education, research as well as for entertainment. Fans of robotic soccer could find him at famous robotics competition RoboCup, where a team of NAO robots plays football.

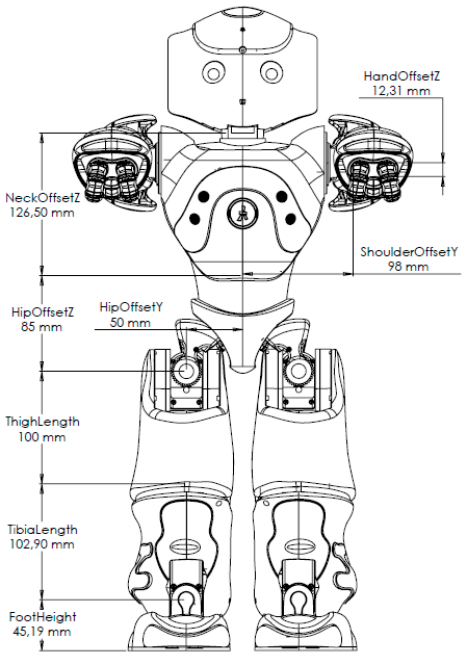


Figure 2.1: NAO robot dimensions from Aldebaran website [27].

Our NAO is the H25 version, which differs from the previous version in some dimensions. Its official proportions can be seen in Figure 2.1. The robot used in this work is also unique because of artificial skin equipped on his upper body, specifically on the head, the torso and on both hands (from the wrist to the fingers).

2.1.2 Artificial skin overview

The skin is a tactile system described in [1], which consists of triangles, which are in fact Flexible Printed Circuit Board (FPCB). Each triangle is formed of 12 sensors (taxels), from which two are thermal and are used to compensate the thermal drift of the other ten taxels. These two are embedded into the FPCB, which secures that pressure does not affect these. This implantation inside the triangles also creates a bigger space for the remaining ten pressure taxels.

The size of the pressure taxels is 15.20 mm^2 (radius is 2.2mm) and the taxels are equally distributed over the triangles with a fixed distance from each other (distance between two neighbor taxels is about 2mm). Example of one triangle is in Figure 2.2. These triangles are associated into *patches*, in which they are physically connected, and all read by one microcontroller. One patch consists typically of 16 triangles, but not always the whole patch is used.

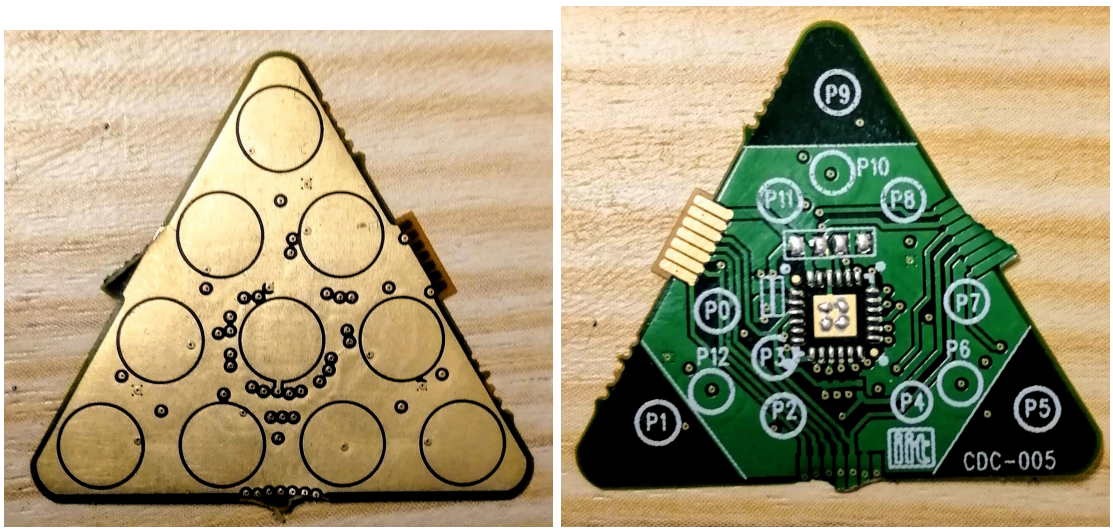


Figure 2.2: Example of front side (Left) and back side (Right) of the triangles.

Our Nao robot is covered by 97 triangles in total, which makes 970 pressure-sensitive taxels over the robot's body. More accurately, the head and both hands are each covered by 24 triangles (in two patches), and the torso provides us information from 25 triangles (in two patches). The schematics of the skin can be seen in Figure 2.3.

The top layers of the skin—black fabric—can be seen in Figure 2.4. This layer actually consists of three layers, which are 3D air mesh fabric at the bottom, Lycra in the middle, which works as a common ground for all the triangles under it, and protective fabric as the

last layer, which improves mechanical property of sensors [6].

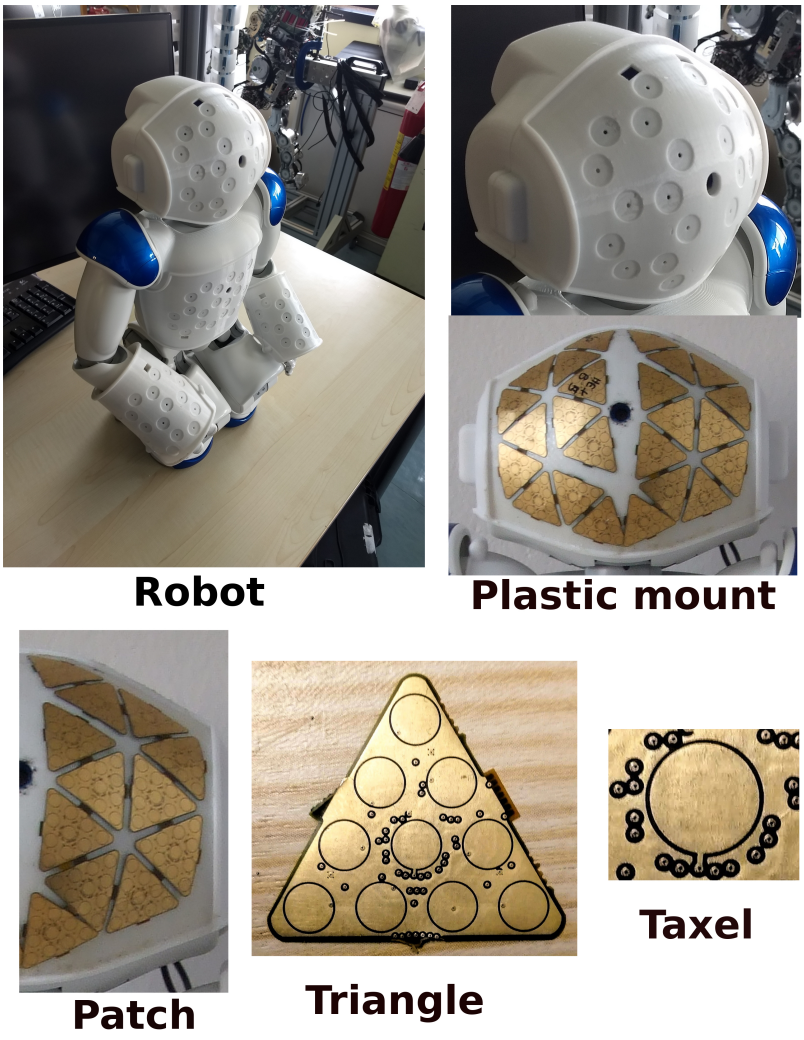


Figure 2.3: Schematic of our setup from the robot to a taxel.

2.1.3 Custom parts to hold the skin

The artificial skin is mounted on custom plastic parts, which changed the robot's dimensions and its operation space and joint limits. Because of the larger body and loss of shoulder pads, some original kinematic parameters cannot be used and we needed to update them. Without their update robot could move further than it was initially intended and it could lead to several damages on the robot. This work was done by Adam Rojik [28]. The output of this work is `Constraints/safeMotion.py` script, which can check whether given joints coordinates are safe for the robot.

Figure 2.4: NAO robot with plastic mounts (top left), backpack for communications with the computer (top right), artificial skin (bottom left) and protective Lycra layers (bottom right).

2.1.4 Forward kinematics

As Spong [29] says: "The forward kinematics problem is to determine the position and orientation of the end-effector, given the values for the joint variables of the robot. The joint variables are the angles between the links in the case of revolute or rotational joints, and the link extension in the case of prismatic or sliding joints."

The position is created by a vector $\mathbf{L}$, which consists of coordinates in x, y and z-axis

$$\mathbf{L} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}.$$

Then, the orientation is described with 3×3 rotation matrix $\mathbf{R}$, which describes a rotation around a given axis. By composing these two matrices we get a 4×4 homogeneous

transformation matrix

$$\mathbf{T} = \begin{pmatrix} & \mathbf{R} & & \mathbf{L} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Creation of this matrix can be simplified with the use of the standard *Denavit-Hartenberg notation*.

Denavit-Hartenberg parameters

Denavit-Hartenberg (DH) notation introduced by Jacques Denavit and Richard Hartenberg [30] in 1955 is used to standardize the choice of the coordinate frames in robotics. The convention determines four parameters:

- **d** - offset along previous z axis to common normal
- θ - angle about previous z axis, from old x axis to the new one
- **a** - radius about previous z axis
- α - angle about common normal, from old z axis to the new one

With the use of these parameters, the transformation to previous coordinate system can be described by the following rotation-translation matrix:

$$\mathbf{T}_i^{i-1} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.1}$$

where $[a_i, d_i, \theta_i, \alpha_i]$ are DH parameters for i^{th} link and $\mathbf{T}_i^{i-1}$ indicates transformation from i^{th} to $(i-1)^{th}$ coordinate system.

DH parameters tables

We did not know any DH parameters for our customized robot, so we had to derive them first. With the official NAO dimensions (for H25 version of the robot) from Aldebaran website, which were used for **a,d** parameters, all derived DH parameters are listed in Table 2.1.

a[m]	d[m]	α[rad]	offset[rad]
0	0.1	-$\pi/2$	0
0	0.098	$\pi/2$	0
0	0	$\pi/2$	$\pi/2$
0	0.105	-$\pi/2$	0
0	0	$\pi/2$	0
0	0	-$\pi/2$	π

(a) : DH for left hand.

a[m]	d[m]	α[rad]	offset[rad]
0	0.1	-$\pi/2$	0
0	-0.098	$\pi/2$	0
0	0	$\pi/2$	$\pi/2$
0	0.105	-$\pi/2$	0
0	0	$\pi/2$	0
0	0	-$\pi/2$	π

(b) : DH for right hand.

a[m]	d[m]	α[rad]	offset[rad]
0	0.1265	0	0
0	0	-$\pi/2$	0
0	0	-$\pi/2$	-$\pi/2$

(c) : DH for head.

Table 2.1: Tables with DH parameters.

Calculation of point in the base frame

Calculation of the position of any point in the base frame, with knowledge of position in its local frame, is done with the following equation

$$\mathbf{P}_{new} = \mathbf{T}_i^0 \cdot \begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix}, \tag{2.2}$$

where $\begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix}$ is a point $\mathbf{P}$ in homogeneous coordinates in its local frame and $\mathbf{T}_i^0$ is homogeneous transformation matrix from i^{th} frame to the base coordinates frame.

Code for forward kinematics

Mathematical principles shown in Equations 2.1 and 2.2 must be implemented as scripts. In Matlab part of work, we use utilities in `MultiRobot/VirtualModel/utils` directory, initially created by Alessandro Roncone [25]. For the Python part, the code has been rewritten into `ForwardKinematics/FwdKin.py`, which is used in the optimization part of this work. The possibility to transform between coordinate systems allows us to visualize robot in Matlab and mainly to compare the relative position of the skin on different body parts during touch configurations.

2.2 Robot model and skin in Matlab

2.2.1 Taxel 3D coordinates

From the manufacturer of the artificial skin, we have only 2D coordinates of the triangles and taxels on the robot's body. From this fact logically implies, that we needed to transform these coordinates into 3D. Without prior knowledge of the shape of each plastic holders (we only knew normal of centre of each triangle from CAD model which differed from the actual placement of the skin patches), we just approximated the position of each triangle.

Scripts in `matlabcodes/fullBodySkin_baseFrame` directory[31] are used to generate the 3D coordinates from the supplied 2D coordinates, when the scripts are called by `matlabcodes/fullBodySkin_baseFrame/plots_main.m` [31]. They were first created by Hassan Saeed [32], and then we edited them because there were some mistakes with triangle indexes. Input to the scripts are 3D coordinates of the centres of each triangle (with normals) extracted by Maksym Shcherban [33] and 2D coordinates from *Excel* files within the directory.

At first, the angle between the normal of the centre of each triangle and the normal of the axis is computed and then each taxel is rotated by these angles and then translated from the centre of the triangle with fixed translation. Output can be seen in Figure 2.5.

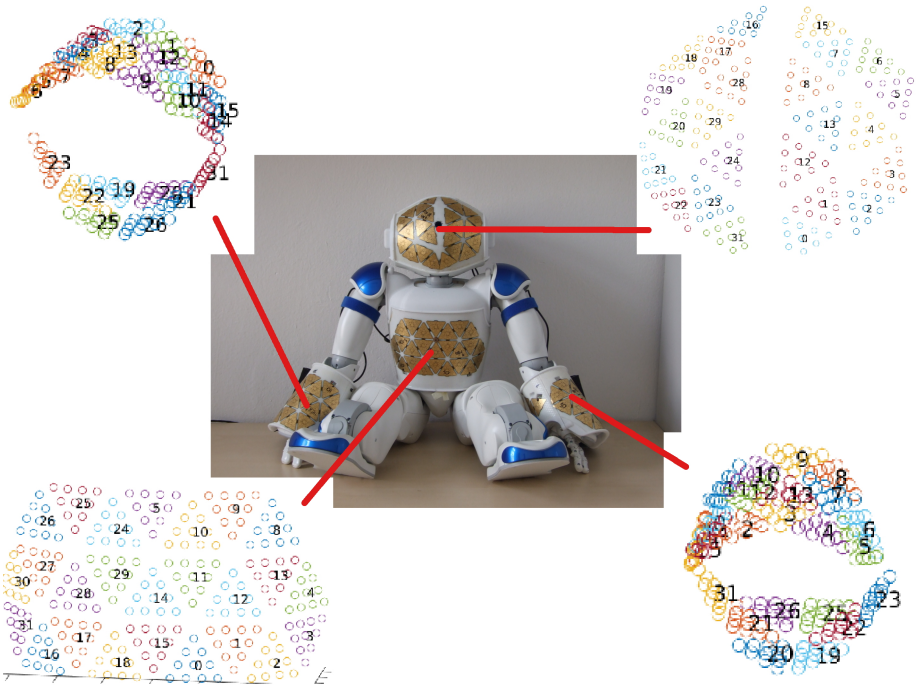


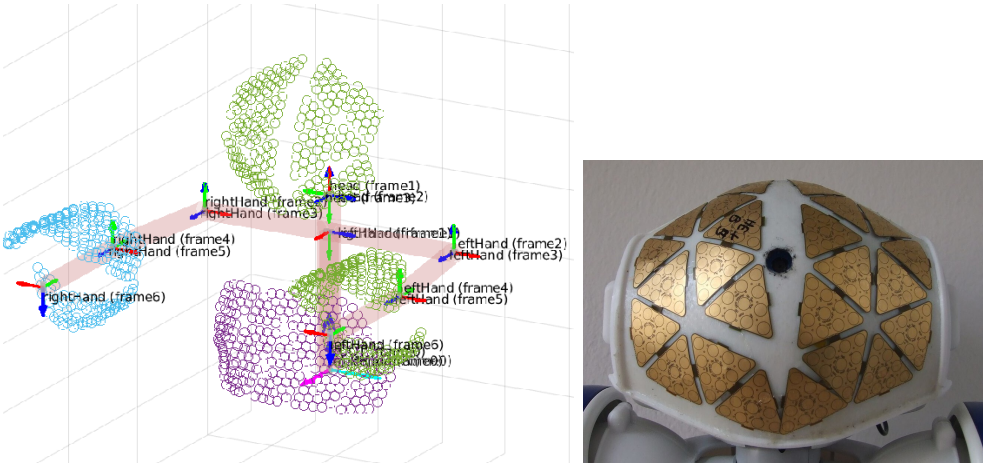
Figure 2.5: Nao with exposed skin and corresponding 3D skin visualized in Matlab. The red lines connects the given skin part with the skin on the real robot.

Coordinates export

Script `matlabcodes/fullBodySkin_baseFrame/plots_main.m` also saves all these coordinates together with the normals and the indexes on each skin part as *.mat* files. These

files are then used by `matlabcodes/fullBodySkin_baseFrame/Output/iniCreator.py`, which creates *.txt files* for reading and evaluating the skin data. Structure of these files is taken from the original ones for *iCub* repository. In these files *-1* means not used triangle from a patch, *-2* implies heat sensor. A line full of zeros stands for taxels from the not used triangles or the thermal sensors.

These coordinates are just approximate because on the real robot not all of the taxels have the same normal as the triangle centre, which causes that our triangles are 'flat' and not bent over the plastic body. Also because of the shape of the head, few triangles had to be put over themselves, as can be seen in Figure 2.6b.



(a) : Example of Matlab model with displayed skin.

(b) : Triangles covering each other on the head of the robot.

Figure 2.6: Triangles on the head and model in Matlab.

2.2.2 Robot model

With 3D positions of the skin, we were able to visualize the robot in Matlab with the use of the scripts in `MultiRobot/VirtualModel` directory. The scripts were originally created for the iCub robot and we supplied the DH parameters from Table 2.1 to them.

The option to display skin was added. So now we can pass joint angles into the function and visualize current position of all chains in the upper body. It is very helpful to reveal possible errors in DH parameters and also this model is used to visualize dataset as described in Section 3.3.2. Example of this model is in Figure 2.6a.

2.3 Optimization problem formulation

The problem can be expressed as estimation of the vector of parameters

$$\phi = \{[a_1, \ldots, a_n], [d_1, \ldots, d_n], [\alpha_1, \ldots, \alpha_n], [\theta_1, \ldots, \theta_n]\},$$

where $i \in N$ and $N = \{1, \ldots, n\}$ is a set of indices identifying individual links. The parameters a, d, α and θ indicate the four DH parameters. We do not optimize DH parameters of the robot itself, just the skin parts, and the number of parameters n for each is skin part is 35 (one for the skin mounts, two for the patches and 32 for the triangles).

It is possible to optimize only a subset of these parameters, assuming that the others are known. The subset can be a subset of links $N' \subset N$ (e.g. calibration of the pose of one triangle only) or a subset of the parameters (e.g. calibration of the parameter a only).

The estimation of the parameter vector $\boldsymbol{\phi}$ is in general done by optimizing a given objective function:

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{m=1}^{M} ||\boldsymbol{p}_m^r - \boldsymbol{p}_m^e(\boldsymbol{\phi}, \boldsymbol{\Theta}_m)||^2, \tag{2.3}$$

where M is the number of configurations in given dataset. The vector $\boldsymbol{p}_m^r$ is the real position of the end-effector (taxel in our case) and $\boldsymbol{p}_m^e$ is the estimated position of the end-effector computed using forward kinematics for a given set of parameters $\boldsymbol{\phi}$ and joint angles $\boldsymbol{\Theta}_m$.

We do not have any ground truth and real position of the taxels (as was used for example in [24] to estimate the absolute error), we can only estimate the parameters from closing the kinematic chain through self-touch. As there are multiple activations on both self-touching parts, it is difficult to find the corresponding taxels to enable kinematic chain closure. One of the possibility is to compute a centre of pressure (COP) of the activated taxels and then optimize the Euclidean distance between the corresponding COPs. Equation 2.3 then changes to:

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{m=1}^{M} \left(\underset{\boldsymbol{\phi}}{\operatorname{argmin}} ||\boldsymbol{c}_p^r - \boldsymbol{c}_q^e(\boldsymbol{\phi}, \boldsymbol{\Theta}_m)||^2 \right), \tag{2.4}$$

where $P = \{1, \ldots, p\}$ and $Q = \{1, \ldots, q\}$ are numbers of taxels on first and second skin part, $\boldsymbol{c}^r$ is vector of *COPs* on one skin part and $\boldsymbol{c}^e$ on the other skin part. This optimization is performed under the assumption that if two body parts are touching, they are at the same position. But this is not exactly true, because the triangles have a Lycra layer on them (see Section 2.1.2), which has at least 1mm on each skin part.

Second option is to substitute vectors $\boldsymbol{p}$ with vector of taxels on different skin parts, but pair just a subset of them (we use five taxels, or less when there are not five activations on both skin parts) as we again assumes the ones with lowest distance are the same. Equation 2.4 then changes to:

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{m=1}^{M} \left(\underset{\boldsymbol{\phi}}{\operatorname{argmin}} \sum_{l=1}^{L} ||\boldsymbol{t}_p^r - \boldsymbol{t}_q^e(\boldsymbol{\phi}, \boldsymbol{\Theta}_m)||^2 \right), \tag{2.5}$$

where $L \in \max(5, \min(P, Q))$, when $P = \{1, \ldots, p\}$ and $Q = \{1, \ldots, q\}$ are numbers of taxels on first and second skin part. And vector $\boldsymbol{c}^r$ is vector of *COPs* on one skin part and $\boldsymbol{c}^e$ on the other skin part. This can be easily extended for more than two chains.

These equations apply in case, where one of the chains in the configuration is taken as reference. If we decide to optimize all skin parts, the vector $\boldsymbol{p}^r$ (respectively $\boldsymbol{c}^r$ and $\boldsymbol{t}^r$) would also depend on set of parameters $\boldsymbol{\phi}$ and joint angles $\boldsymbol{\Theta}_m$.

2.4 Multirobot kinematic optimization framework – Matlab

The *robot* class was created to simplify the process of the optimization. The class contains a set of inner methods and utilities, from which the important for *Nao* are:

- `MultiRobot/Utils/loadNAO.m` - serves to choose optimized chains and calibration settings.
- `Opt/Utils/callPythonParsing.m` - calls `DataParsing/dataParse.py` (parsing function, more can be read in Section 3.2.1) with correct arguments.
- `MultiRobot/@robot/runOptimization.m` - for given number of repetitions calls the `Opt/LayersFunc/optAll.m` function (main function for optimization, described in Section 2.5.3) and `Opt/Utils/callPythonParsing.m` method.
- `MultiRobot/@robot/visualizeConf.m` - shows the *Matlab* model with skin for given configuration.
- `MultiRobot/@robot/visualizeNAO.m` - shows graphs and statistics for the dataset based on the arguments:
  - *statistics* - print the statistics about the given configuration (average distance of the *COPs*, number of activated taxels and triangles and more).
  - *distances* - shows graphs containing distances between taxels.
  - *activations* - shows 3D body parts with activated taxels.

  The function accepts another argument, which is name with which the graphs will be saved.
- `MultiRobot/@robot/splitDataset.m` - splits dataset into training and testing part.

The instance of the class can be created for example with command *r=robot('nao')* and the then the methods are called with the instance as the first argument, e.g. *visualizeNAO(r)*.

This class is the first attempt to create the multirobot framework, which would allow to optimize and visualize different types of robots with just minimal effort. Right now it works well just with the Nao, but it was created with the idea of extensibility to another robots. The core of the framework already works. For every new robot its structure can be written into configuration file and the class will create a model. The structure is written in format{ {joint$_1$},...,{joint$_n$}}, where joint is structured as follows:

{jointName, jointType, parent, DHparameters, toOptimize,
whatOptimize, perturbation, isEnd-effector },

where:

- jointName - name of the joint
- jointTyope - type of the joint. For Nao it is: base, joint, mount, patch and triangle. Every new robot can add it own types.

- parent - index in the structure to parent joint. Used for iterating over the joints.
- DH parameters - four DH parameters from previous joint to this joint
- toOptimize - whether the part is optimized or not
- whatOptimize - 1x4 array with true/false for every DH parameters if it will be optimize or not
- parturbation - 1x4 array with perturbations for the DH parameters
- isEnd-effector - true/false, whether the joint is end-effector

From this information, the structure of the robot is made and all joint and end-effector can be accessed with *r.joints* (*r.endEffectors*). Except methods mentioned above, few else are implemented:

- `MultiRobot/@robot/findJoint.m` - returns cell array of pointers to the instance of searched joint, which can be changed etc. It does not have to be full name of the joint, every joint with given string in its name will be returned.
- `MultiRobot/@robot/findJointByType.m` - return cell array of pointers to the instance of searched joint by its type, which can be changed etc. It does not have to be full type of the joint, every joint with given string in its type will be returned.
- `MultiRobot/@robot/changeJoint.m` - the method accepts arguments:
  - type - search by name or by type
  - name - name/type of joint to be changed
  - parameter - which parameter to change (DH parameters, parent etc.)
  - newValue - new values of the parameter. If cell array with same length as array of found joints if passed, the joints will get the values on corresponding index
- `MultiRobot/@robot/print.m` - prints all joints/end-effectors as string "jointName, index"
- `MultiRobot/@robot/showModel.m` - shows Matlab model (described in Section 2.2.2) from passed DH parameters and joint angles

The class uses *optProperties Matlab* struct which includes all kind of settings. Description and default values can be found in Appendix A.

2.5 Nao robot skin optimization

Our framework allows us to optimize three main parts of the artificial skin: the positions of plastic mounts due to its body parts, the positions of each patch in their local frame and also positions of every triangle (schematics of the setup can be seen in Figure 2.3 in Section 2.1.2 about the skin). Right now, there is no possibility to optimize the DH parameters of the robot itself.

The user is able to choose between these types of optimization, and it is possible to optimize them repeatedly and in any order. This is important because of the fact, that

each principle changing with the given configuration, and sometimes it is preferable to optimize patches and then triangles, but sometimes vice versa. But we can also optimize all parameters in the same time. Also we can perturbate or bound each of these parameters. It is also possible to forbid in configuration file optimization of any of DH parameters - i.e. when redundant links.

There are two possible scenarios:

- **one chain is taken as a reference** - we set one chain as the reference and optimize the other one from the mutual touches. For example we can optimize the skin parts on the hands with the use of the torso, or the head with the hands.
- **simultaneous calibration** - none of the chains is reference and the parameters of both chains are optimized in the same time.

Plastic mounts

The plastic mounts are mounted on the original robot, and the skin is mounted on them (schematics of the setup can be seen in Figure 2.3 in Section 2.1.2 about the skin). The initial optimization of plastic mounts position and orientation was necessary because:

- We had the positions of centres of the triangles in the plastic holders local frames, but in the CAD model, they were not mounted on the robot, so we do not know their exact position.
- In the CAD model, they were rotated differently than they are rotated in the default 'home' position of the robot. This resulted in wrong orientation after transformation with forward kinematics to the base frame.

Visualization of plastic mounts on right hand before/after optimization can be seen in Figure 2.7.

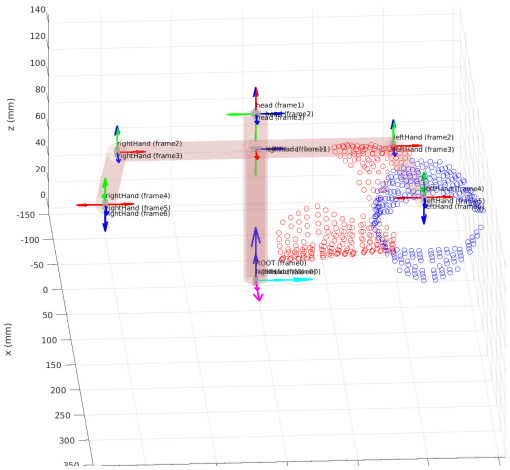


Figure 2.7: Example caused by wrongly set rotation in CAD. Intended rotation is in blue and result of forward kinematics in red. Robot is in his 'home' position, which means zero radians as value of all of his joint angles.

Patches and triangles

The patches and the triangles can be also optimized (schematics of the setup can be again seen in Figure 2.3). It is better, when the optimization of the plastic mounts is done before, because they should just adjust the overall pose by changing their position by few millimeters. The minimum number of data poses in dataset to estimate DH pars of patches is twice as much as for plastic mounts. For triangles we optimize 128 parameters for each chain which requires a bigger dataset.

COP

Important term is *COP - centre of pressure*. As comparing directly the postions of the taxels is not easy as discussed in Section 2.3, we use the *COPs* – 3D position of the pressure center computed from the positions of the activated taxels and weighted by the force on each taxel. During one activation, there could be more *COPs* on one skin part. This depends on parameters used during calculation of the *COP*. The *COPs* can be computed while collecting the data from the robot (see Section 3.1.1) or during the parsing (see Section 3.2.1)

2.5.1 Matlab implementation of nonlinear least-squares

We have decided to chose nonlinear least-squares optimization as a solution to our optimization problem. Matlab already offers their implementation of a solver for this type of problems in *Optimization toolbox*. We decided that it will be better to use official implementation because it is well tested and provides a lot of options to be set, so we can adjust its output. The *lsqnonlin* function provides a possibility to switch between *Levenberg-Marquardt* and *trust region reflective* algorithm. It also supports bounds for optimized values and iterations limit. Some of these parameters used by us are stated in Appendix A.

It also expects custom functions for computing of the criterion values. It is very useful, because each type of optimization requires a bit of different attitude. Functions are then passed to *lsqnonlin* as Matlab anonymous functions with optimized values as parameters.

2.5.2 Training and testing sets for optimization

Every kind of optimization has one thing in common, and that's the distribution of the dataset into a training and a testing section. The datasets have few minor differences, which will be described later, but the main idea above the splitting is the same.

We take the total number of entries in a dataset and create vector composed of random permutation from one to n, where n is the size of the dataset. The last 30% is dedicated to the testing segment. The rest 70% is the training segment.

We took an example from the training of the neural networks and the training segments can be lately divided into random *mini batches* (mini batches are just small portions of the training segment chosen at random from the overall number of poses.), because how Masters and Luschi [34] proved, it is much better to train over small batches with more repetitions. It is less demanding on memory and computing power of the computer, and

also it converges faster. This method is connected with the Stochastic gradient descent, but it is well applicable to our problem because Levenberg-Marquardt method is from the same family of algorithms.

2.5.3 The calibration pipeline

The core of the pipeline is the `Opt/LayersFunc/optAll.m` function, which calls other functions. Input to this function is the instance of the *robot* class, described in Section 2.4.

Optimization principle

Our approach is based on pairing two closest *COPs*. The closest means two *COPs* on different skin parts with lowest Euclidean distance between them (this is described in Section 3.2.1 and included in the dataset described in Section 3.5.2). With the *COPs* we call the `Opt/Utils/prepareData.m` function, which finds two taxels with lowest distance between them (one taxel on each skin part, described in Section 3.2.2) and in given radius around the *COPs*. From these taxels new dataset is created (described in Section 3.5.3).

This dataset is split into training and testing part, as described in Section 2.5.2, and the training part is used for the optimization with the use of the `Opt/OptimizationFunc/optFunction.m` function. In this function, the pairs of taxels mentioned in paragraph above are transformed into the base frame and the output of the function is vector of Euclidean distances between those taxels. This vector is used by *lsqnonlin* method to optimize the parameters.

The output parameters are then tested on the training dataset and the parameters with the best results are taken and saved into *.txt* files in the `Opt/OptimizedParameters` folder. For the plastic mount it is four parameters (a, d, α, θ), each on a new line. For the patches it is two lines for each skin part and for the triangles it is 32 lines. Each run of optimization is divided by an empty line.

We do not optimize directly the input positions and rotations of the taxels, but we added a set of new links: a link between the last joint and the plastic holder, between plastic holder and patches and between patches and individual triangles and we optimize *DH* parameters of these links. Thanks to this, original DH parameters of the robot can be kept uncalibrated and only calibration of the new virtual links corresponding to individual skin parts is performed.

Calculation of a new point in a local frame

Adding new links leads to the fact that we need to recompute COPs. For that we need to know the new position of every taxel in its local frame. This can be achieved with the knowledge of the forward kinematics and matrix multiplication. In Equation 2.2 we can do a matrix inversion of $\mathbf{T}_i^0$, and thus in general we get

$$\begin{pmatrix} \mathbf{P} \\ 1 \end{pmatrix} = \left(\mathbf{T}_i^0\right)^{-1} \cdot \mathbf{P}_{new} = \mathbf{T}_0^i \cdot \mathbf{P}_{new}, \tag{2.6}$$

where $\begin{pmatrix}\mathbf{P}\\1\end{pmatrix}$ is a point in homogeneous coordinates in a local frame and $\mathbf{T}_0^i$ is a homogeneous transformation matrix from the base frame to the point's local frame.

In our particular case, we applied a transformation from a local frame of the plastic mounts to the base frame, then added a link from a new position of the plastic mount to the plastic mount, a new link from each patch to a new position of the plastic mount and a link from each triangles to their parent patch. The diagram of the connections can be seen in Figure 2.8 That can be mathematically expressed as

$$\begin{aligned}\mathbf{P}_0 = \mathbf{T}^0_{plastic} \cdot \mathbf{T}^{plastic}_{new\_plastic} \cdot \mathbf{T}^{new\_plastic}_{patch_i} \cdot \mathbf{T}^{patch_i}_{triangle_j} \cdot \mathbf{P}_{local} = \\ = \mathbf{M} \cdot \mathbf{T}^{new\_plastic}_{patch_i} \cdot \mathbf{T}^{patch_i}_{triangle_j} \cdot \mathbf{P}_{local},\end{aligned} \tag{2.7}$$

where we substituted transformation from the new position of the plastic holder to the base frame by matrix $\mathbf{M}$. It is just for clarity because we know $\mathbf{P}_0$ of every taxel in the base frame and we want to know its position in its local frame, but we take the frame of the plastic holders as a part of the original chain. It can be achieved as follows

$$\mathbf{P}_{new\_local} = \mathbf{M}^{-1} \cdot \mathbf{P}_0, \tag{2.8}$$

where the inverse matrix is computed as:

$$\mathbf{M}^{-1} = \begin{pmatrix}\mathbf{R} & \mathbf{t}\\ \mathbf{0} & 1\end{pmatrix}^{-1} = \begin{pmatrix}\mathbf{R}^T & -\mathbf{R}^T\mathbf{t}\\ \mathbf{0} & 1\end{pmatrix}, \tag{2.9}$$

where $\mathbf{R}$ is rotation and $\mathbf{t}$ is translation component of the matrix $\mathbf{M}$. This is done in `DataParsing/dataParse.py` script described in Section 3.2.1.

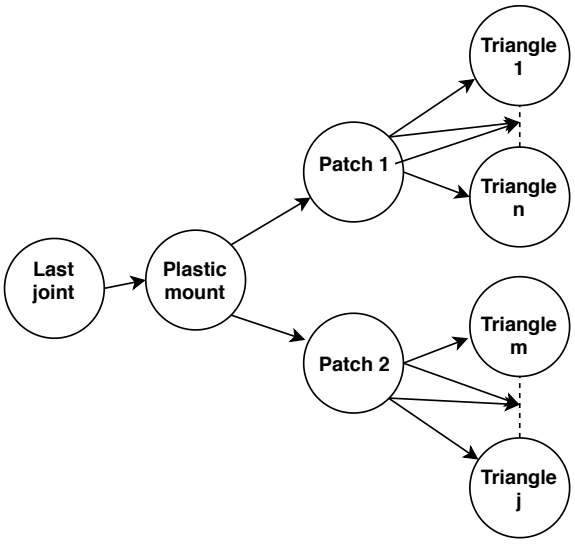


Figure 2.8: Diagram of connections between the frames

Implementation

The principle described above is implemented in `Opt/LayersFunc/optAll.m`, shown in Pseudocode 1.

Algorithm 1: Pseudocode of the `Opt/LayersFunc/optAll.m` function.

```
1  call Opt/Utils/prepareData.m to get dataset;
2  for each triangle do
3  |   create another dataset by assigning the taxels to the triangles;
   |   // the output dataset is described in Section 3.5.3
   |   /* if this dataset contains more items than given threshold (10 by default) this
   |      dataset is added to big dataset (this helps to not optimize triangles, which
   |      have only few activations and the optimization would not be accurate)       */
4  end
5  precompute the homogeneous transformation matrices from the last joint to the base
     frame;
   /* to accelerate the optimization, because these matrices are the same for the whole
      time                                                                            */
6  call MultiRobot/@robot/splitDataset.m to split dataset into training and testing
     part;
7  if three chains are used then
   |   // e.g.  the right hand, the left hand and the torso
8  |   repeat lines 2-6 for second configuration;
9  end
10 select the right parameters based on what is optimized;
   // only plastic mounts/only patches/everything etc.
11 set the bounds for each parameter;
12 for number of repetitions do
13 |   optimize the parameters on the training dataset with
   |    Opt/OptimizationFunc/optFunction.m;
   |   // with the use of the lsqnonlin
14 end
15 for number of repetitions do
16 |   test the parameters on the testing part of the dataset;
17 end
18 save the best parameters into text file;
   // the parameters with lowest error on the testing part of the dataset
   // parameters are saved in Opt/OptimizedParameters folder
```

The same can be seen in Figure 2.9.

And pseudocode of `Opt/OptimizationFunc/optFunction.m` is described in Pseudocode 2.

Algorithm 2: Pseudocode of the `Opt/OptimizationFunc/optFunction.m` function.

```
1  for each item in dataset do
2  |   select the right parameters for each optimized link;
   |   // if the link is not optimized, the values saved before are used
3  |   compute the homogeneous transformation matrix from the given triangle to the
   |    last joint;
4  |   multiply it with the precomputed matrices to the base frame;
5  |   multiply it with a taxel in its local frame (on the first skin part) to get the first
   |    point;
   |   // taxel_base = T^base_plastic · T^plastic_new_plastic · T^new_plastic_patch_i · T^patch_i_triangle_j · taxel_local
6  |   if not simultaneous calibration then
7  |   |   the second point already in dataset;
8  |   else
9  |   |   compute the second point with repeating lines 3-5 for the second skin part;
10 |   end
11 |   compute Euclidean distance between the two points;
12 end
13 if three chains are used then
14 |   repeat lines 1-11 for second dataset;
15 end
16 return n × 1 vector of the Euclidean distances, where n is size of the dataset;
   // size of both datasets if three chains are used
```

2.5.4 Sequential optimization principle

The entire optimization is based on pairing two of the *COPs* as it is more beneficial than pairing all of the taxels. But on the other hand, it brings several inconveniences from which the most unpleasant is pairing of "wrong" taxels. We assume, that the *COPs* with the lowest Euclidean distance between themselves are the one which should have the same position expressed in the base frame. But in some cases (e.g. first run of calibration when offsets of plastic holders are totally wrong; situation after perturbation etc.) the selected *COPs* are not the ones, that should be compared.

For this reason it is necessary to run the optimization sequentially, which means to estimate parameters, parse new data (pair new taxels etc.) and run optimization again. Sometimes one or two runs are enough, but sometimes more than 10 runs is needed (e.g. after perturbation). The pseudocode of general calibration can be seen in Pseudocode 3

Algorithm 3: Structure of the sequential optimization

```
load settings; load parameters;
parse the data with the parameters;
while Error > wantedError do
    estimate new parameters;
    parse the data with new parameters (select new pair of COPs);
    check error;
end
```

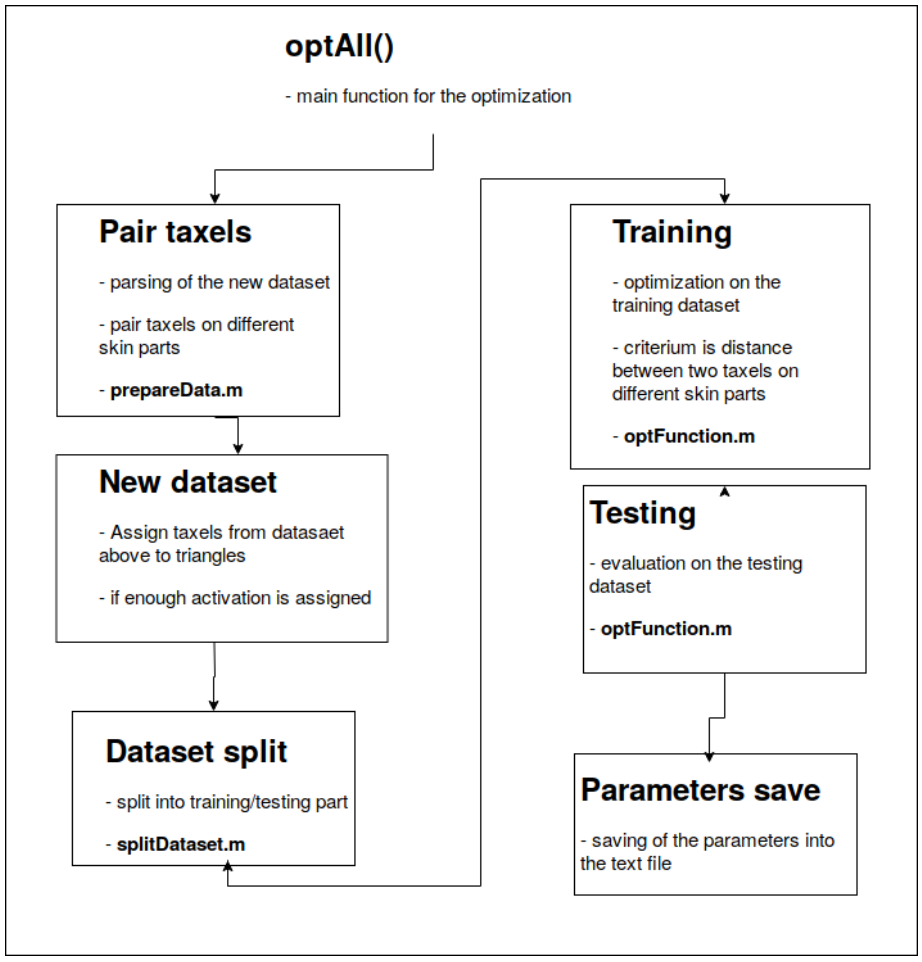


Figure 2.9: Schematics of the optimization framework.

Chapter 3

Data collection

3.1 Reading of the skin data

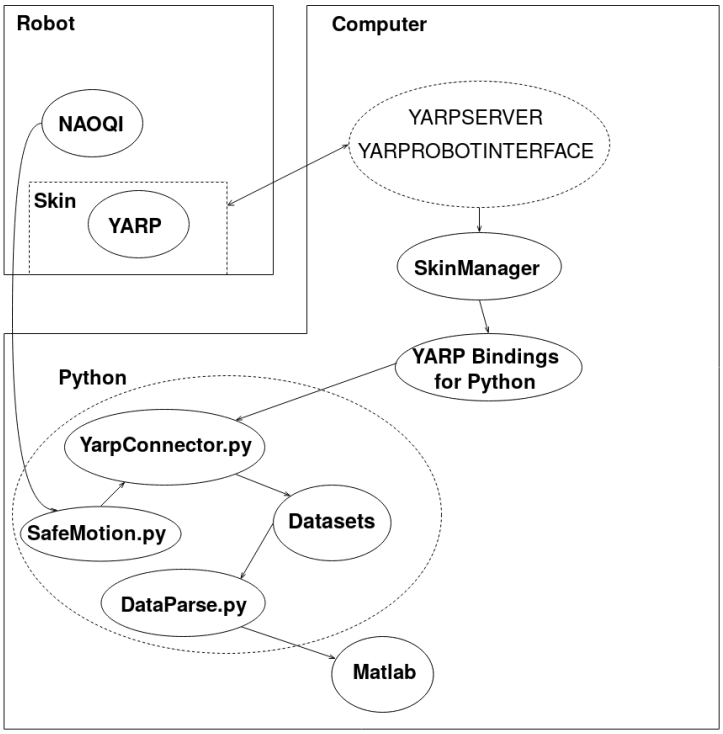


Figure 3.1: Schematics of the robot and computer connection, with drawn scripts and programs and their connections. All terms have been already mentioned, or will be described in the following chapter.

We have to read skin activations and joint angles of the robot. For this purpose, we use the `Constraints/safeMotion.py` script mentioned in Section 2.1.3. It communicates with *NAOqi* framework and reads the robot's data. *NAOqi* is the name of the main software that runs on the robot and controls it, and with the *NAOqi* framework we can program the robot [35].

To read the skin, the board situated on the robot's back is used. This board sends data to *TCP* port which we can read and later process the data. The board reads data from the patches, where each patch is connected to own bus. This procedure is provided by *YARP*

system. "Backpack" mounted on the NAO can be seen in Figure 2.4.

YARP

YARP stands for "Yet Another Robot Platform." It is not an operating system, but a middleware between a robot and a computer. Metta *et al.* [36] say: *"The goal of YARP is to minimize the effort devoted to infrastructure-level software development by facilitating code reuse, modularity and so maximize research-level development and collaboration."*

Important utilities from this platform are:

- `yarpserver` - a *YARP* name server, translates IP address and port of given TCP port to names. For example: *"http://10.0.1.104:10001"* translates to *"/nao/skin/left_hand"*.
- `yarprobotinterface` - starts all the devices required by a robot.
- `yarpmanager` - manager for the original GUI. Simplifies visualization and connection of the skin, based on a provided configuration file.

3.1.1 Types of sensor data

As have been mentioned above, *YARP* sends data to TCP ports, from where we read them. One possibility is to read raw data directly from ports. It is quick, but not efficient. The data are in a form of an array with *384* values (one for each taxel) from *0* to *255*, where *0* stands for maximal activation. The raw data from the sensors have to be later processed/compensated for temperature effects.

Second option is to use `skinManager` from the iCub library. This tool do thermal compensation and other things for us.

skinManager

Official description from the iCub website [37] is: "*This module reads the raw tactile sensor values, compensates for the (thermal) drift of the sensors (basically it is a high pass filter) and writes the compensated values on output ports.*"

`SkinManager` is configured using the configuration file *ReadSkin/conf/skinManAllNao.ini* (based on original iCub configuration file). Most important settings are input/output ports and other parameters:

- *binarization* - if true, data are not in *0-255* format, but it only returns *0* for no activation and *100* for activation.
- *maxNeighborDist* - sets maximum distance between activated taxels in metres for which `skinManager` returns one common centre of pressure.
- *zeroUpRawData* - if true, raw data are in range 255-0 (reversed from the original).
- *taxelPositionFiles* - list of files in the same order as input ports, from which the module take location of taxels and compute information.

- *skinParts* - returns index of the skin part, where an activation was detected.

To enable reading data from the new head skin part (not available for iCub) in the `skinManager`, we set the head to a non-used index – we choose skin on the right forearm.

It is possible to read from the module two types of data. One option is to read the "raw" compensated data which have the same shape as the original data directly read from the robot. By setting the above mentioned parameters we get data which are not dependent on the skin temperature, etc.

The second option is to read *skinContactList* [38], which is a list of *skinContacts* created by `skinManager`. Each *skinContact* contains post–processed information about touch data. For example:

- *center of pressure (COP)* - 3D position of the pressure center computed from the positions of the activated taxels and weighted by the force on each taxel. In one touch, there could be more *COPs*, based on *maxNeighbourDist* parameter.
- *taxelList* - a list of the indexes of the activated taxels. Used to find position of taxels.
- *skinPart* - an index of the activated skin part. Based on the *skinParts* parameter in `skinManager` configuration file.
- *pressure* - an average output of the activated taxels.

To allow *skinManager* compute this statistics, it needs to have text files with position of each taxel in relative path *position/*.txt* to the *skinNaoAll.ini*. These files are generated by `matlabcodes/fullBodySkin_baseFrame/Output/iniCreator.py` [31]. It takes data which were computed along with generating the 3D positions of the skin and transform them into a format, which is needed by the `skinManager`.

3.1.2 Pipeline for reading skin inputs

The actual reading is done in `YarpConnector/yarpConnector.py`. This script accepts name of the dataset and name of two chains which will be touching. Then it starts three parallel processes. One for collecting *skinContactList* and two for collecting data from compensated ports (one for each chain).

The script uses the *YARP* bindings for *Python*. With the use of this library, we are able to open own ports to which we forward ports from *skinManager*. These ports then serve as readers of the data. Together with reading, the visualization of gathering can be displayed (more can be seen in Section 3.3.3).

Three main functions are implemented:

- `readSkinManager` - this function connects to a port at which are *skinContactList* sent. It reads *skinContacts* one by one and parses them into dictionary, which is then saved into the dataset directory. It also saves raw, unparsed vector of *skinContacs*.

- `readRaw` - this function connects to compensated port for a given chain. Reads the data and saves them into a file.
- `printOutput` - prints data parsed in `readSkinManager` in human readable form.

Both functions also use the `Constraints/safeMotion.py` class mentioned in Section 2.1.3. It serves as middleware for reading of the robot's joint angles. These angles are saved together with the other information because we need to reconstruct the pose of the robot for the given configuration. Output of the reading are four files for each configuration in format:

- *chain1chain2.txt* - dictionary with parsed *skinContactList*.
- *chain1chain2_raw.txt* - unparsed *skinContactList*.
- *chain1.txt,chain2.txt* - compensated raw data for a given chain.

Content of all files is a dictionary dumped into string. This format was chosen because it is well human readable and in *Python* it is very easy to get the dictionary back from the text file.

3.2 Parsing

3.2.1 Python parsing

The main part of the parsing is implemented in Python. Script `DataParsing/dataParse.py` was created for this reason. It contains class *Parser*, which loads files with positions of the taxels in the local frames, files with optimized parameters and parse data from the robot into datasets for Matlab. Parameters can be set in the file itself or can be obtained from a command line. That is used while this script is called from Matlab in function `Opt/Utils/callPythonParsing.m`.

Inside the class following methods are implemented:

- `computeFwdMat` - this method computes the homogeneous transformation matrix from a local frame to the base frame. Without any new link added during optimization. It uses the `FwdKin.py` script for forward kinematics, mentioned in Section 2.1.4.
- `computeKinematics` - this method transforms point from local frames to the base frame. Specifically it computes matrix $\mathbf{M}$ from Equation 2.5.3 and multiplies it with points passed into the method.
- `computeCOP` - this method computes new *COP* from the taxels.
  - It accepts *maxNeighborDistance* argument, which allows us to change distance between activated taxels beyond which new *COP* is recognized – helps to reduce the area from which the *COP* is calculated and isolate the touches of the skin parts.
  - The calculation is based on calculation of distances between two activated taxels and comparison of the distance with *maxNeighborDistance*

- Right now the *COPs* are computed as a mean from the positions of the activated taxels (they are not weighted as the *COPs* from `skinManager` described in Section 3.1.1)

- `parseRaw` - this method parses data from the compensated raw ports. It pairs activations based on similar timestamps and connects pairs of taxels with the lowest Euclidean distance.

- `parseSkinManager` - this method loads data from the `skinManager` and parse them to *.mat* files for Matlab. The implementation can be seen in Pseudocode 4.

Algorithm 4: Pseudocode of the `parseSkinManager` method.

```
delete activations where only one skin part was activated;
for each activation do
    calculate number of activated taxels and triangles;
    // Later used for calculation of dataset statistics (e.g. percentage of
       activated triangles)
    if any optimized parameters are used then
        compute new position of the taxels in the local frames;
        // Necessary when we added new links in optimization.
        // Implementation of mathematical principle described in Section 2.5.3 in the
           optimization framework.
    end
    call computeCOP method;
    find the two closest COPs;
    /* The closest COPs represent pair of COPs, each on different skin part, which
       have the lowest Euclidean distance between them. The distance is calculated
       by well known formula

                    d = sqrt((A_x - B_x)^2 + (A_y - B_y)^2 + (A_z - B_z)^2),        (3.1)

       where A, B are the COPs on different skin parts.                          */
    call computeKinematics method;
    save .mat file with activation info;
end
```

3.2.2 Parsing for the optimization functions

As optimization function does not use directly *COPs* the dataset described in Section 3.5.1 is needed to be parsed. This is implemented in `Opt/Utils/prepareData` and implementation can be seen in Pseudocode 5.

Algorithm 5: Pseudocode of the `Opt/Utils/prepareData` function.

```
1 for each each activation do
2     find taxel with the lowest Euclidean distance to the COP (on the first skin part);
3     find taxel on the second skin part in given radius from the COP on the second
        skin part and with the lowest Euclidean distance to the taxel found on line 2;
      // we want to find the nearest taxels, but both of them need to be close to COPs
         on their skin part
4     save data into Matlab struct;
      // distance between taxels, joint angles and the position of the taxels
      // see dataset structure in Section 3.5.3
5 end
```

3.3 Visualization

There are multiple types of visualizations throughout this work (e.g. to visualize data collection, robot model, skin parts, etc.). Some of them are official from the manufacturers of the skin and some were created by us.

3.3.1 Official iCub GUI

One way to visualize is using *the iCub skin GUI* running with configurations for NAO. This GUI is also recorded while a dataset is being collected for a backward check of how the configurations looked. It can be seen in Figure 3.2.

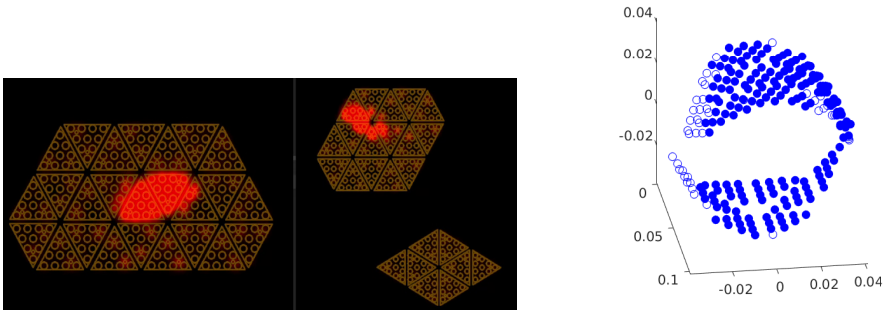


Figure 3.2: (Left) Example of the iCub skin GUI with activation on the torso and the right hand. (Right) Nao right hand during dataset collection.

3.3.2 Robot model in Matlab

Another way is based on the *Matlab* model mentioned in Section 2.2.2. In the script `Visualization/getFigs.m` we use this model and the dataset from the robot to visualize robot model with skin activations in the given configuration. For each activation, the current configuration based on joints angles is shown, together with activation number and further information of activated taxels. Activations can be selected with *left* and *right* arrows on the keyboard. Example of the visualization can be seen in Figure 3.3.

3.3.3 Visualization of dataset collection

For full calibration it is important to activate all the taxels on the robotic skin. Therefore we visualize already activated taxels on the skin part during the dataset collection. Right now, it does not distinguishes how many times the taxel was activated, but still it is much more comfortable to collect the dataset with this tool. The output during the collection process can be seen on the right side in Figure 3.2.

3.3.4 Dataset information

Also minor scripts for evaluating the dataset were created. The scripts are called by a method `MultiRobot/@robot/visualizeNAO.m`, described in Section 2.4. These graphs are used in the Results chapter and the possibilities available are:

- *statistics* - prints the statistics about the given configuration (average distance of the *COPs*, number of activated taxels and triangles and more).
- *distances* - shows graphs containing distances between taxels.
- *activations* - shows 3D body parts with activated taxels.

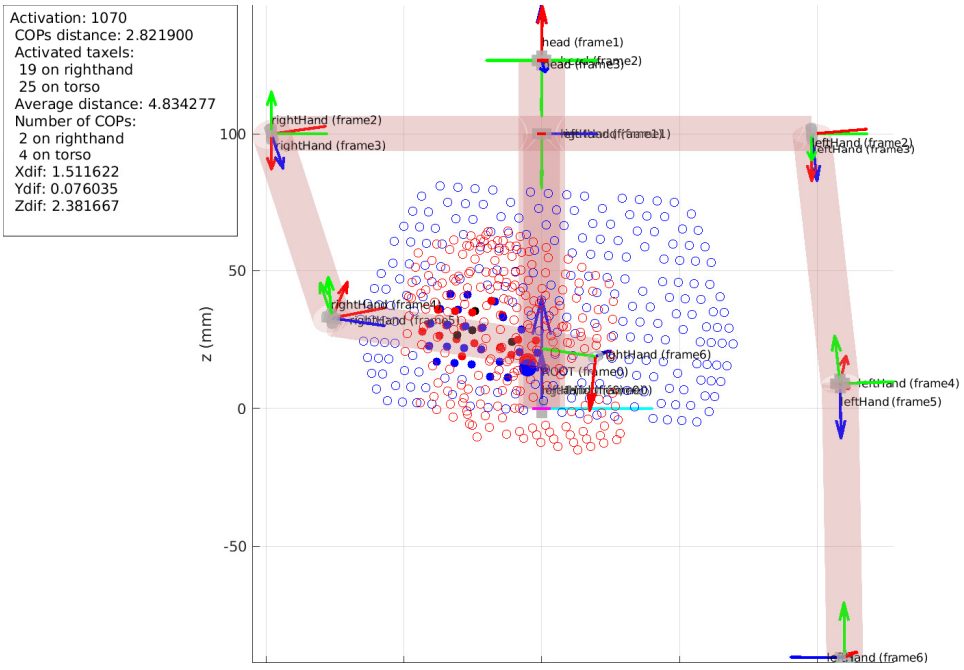
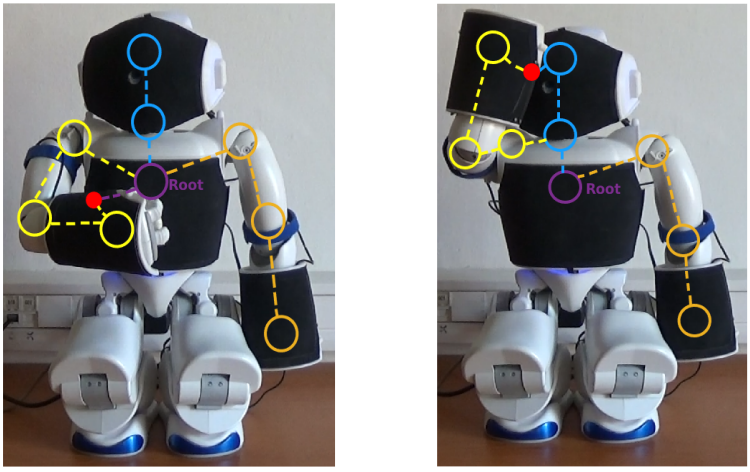


Figure 3.3: Example of the visualization with the use of the *Matlab* model.

3.4 Robot in self-touch configurations

The datasets are created by manually moving the robot into the touch configurations and not autonomously by the robot. Every configuration consists from two chains getting in contact with each other. Four different configurations were recorded: right hand – torso, left hand – torso, right hand – head, left hand – head. Two of the configurations are

demonstrated in Figure 3.4. The poses are chosen in accordance with the feedback of activations (with visualization mention in Section 3.3.3) to ensure that the majority of the taxels is activated. Some examples from dataset collection can be seen in Figure 3.2 and videos recorded with the dataset can be found in *Videos* folder for the most of the datasets (datasets are located in the *DataParsing/Datasets* folder) .



(a) : Right hand - torso. **(b) :** Right hand - head.

Figure 3.4: Figures show chains of the robot in self-touch configurations – torso (purple), right arm (yellow), left arm (green) and head (blue). The detected *COP* is marked as the red point.

3.5 Datasets

In the thesis, several types of dataset were used. They differ in format, file type and content.

3.5.1 Data from the robot

The datasets recorded from the robot are saved in the *DataParsing/Datasets* folder, where the folders of each dataset are named as *Data**, where * indicates the date of creation. Names of files and their content is described in Section 3.1.2. These datasets are source for all other datasets.

Compensated raw data

Each file with compensated raw data contains Python dictionary with timestamps as keys. For each key there is an array with two members. On the first index it holds array with 384 numbers representing taxels on the given skin part. Each number indicates the force of the touch. Values differ for every dataset, based on the settings of *skinManager*. They can be zero (non-activated) or 100 (activated), if the binarization parameter is turned on. Second possibility is that they are from zero (non-activated) and rising with force, in most cases up to the value of five.

On the second index, there is another dictionary with joint names as keys and their angles in radians as values. This is common for all files from this type of dataset. The general structure can be seen below.

{"TimeStamp":[[taxel$_1$,...,taxel$_n$],{"Joint$_1$":angle,...,"Joint$_n$":angle}]}

Unparsed *skinContactList*

These files again contain Python dictionary with timestamps as keys. Value for the timestamps is an array with unparsed *skinContactList* on the first index and the dictionary with joint angles on the second index. The structure is following:

{"TimeStamp":["((contactId bodyPartId linkNumber skinPart) (centerOfPressure$_x$ COP$_y$ COP$_z$) (force$_x$ f$_y$ f$_z$) (moment$_x$ m$_y$ m$_z$) (geometricCenter$_x$ GC$_y$ GC$_z$) (surfaceNormalDirection$_x$ SND$_y$ SND$_z$) (activatedTaxelId1 ATId2 ... ATIdN) average_pressure)",{"Joint$_1$":angle, ...,"Joint$_n$":
angle}]}

Parsed *skinContactList*

This is the most complex dataset, which is also the most used one. Again it is a dictionary with timestamps as keys, where the value of each of them is an array with *skinContacts* dictionary on the first index and dictionary with joint angles on the second one. General structure is simple:

{"TimeStamp":[skinContacts,{"Joint$_1$":angle,...,"Joint$_n$":angle}]}

The *skinContacts* dictionary itself has *contactId* as key and values are another dictionaries with key/value pairs in Table 3.1.

Key	Value
"cop"	1 × 3 float array
"force"	1 × 3 float array
"normal"	1 × 3 float array
"skinPart"	string
"linkNumber"	int
"moment"	1 × 3 float array
"averagePressure"	float
"geoCenter"	1 × 3 float array
"bodyPartId"	string
"taxelInfo"	dictionary

Table 3.1: Structure of the *skinContacts* dictionary.

The *taxelInfo* dictionary is in a simple format {"taxelId": 1 × 6 float array}, where the first three numbers are the position of the taxel on the skin part and the following three form the normal of the taxel.

3.5.2 Data parsed for Matlab

Datasets from Section 3.5.1 are parsed for Matlab. We have chosen to save each configuration into a separate *.mat* file with names in the format: *chain1chain2_coords**{i}**.mat*, where **{i}** indicates the number of the configuration.

The *.mat* file contains 2 variables – one for each chain. Each variable is Matlab struct including these fields:

- cop - n×3 array with *n centres of pressure* detected on the given skin part. Transformed into the base frame.
- selectedCop - 1×3 array with *COP*, which has been chosen on the skin part to be compared with *COP* on the second skin part. Transformed into the base frame.
- copNoT - 1×3 array with *selectedCop* expressed in a local frame.
- NoCops - integer. Number of detected *COPs*.
- distance - float. Distance between the *selectedCop* on both skin parts.
- difs - struct with fields *x, y, z* with distance in individual axis.
- activated - n×4 array with *n* taxels, which were activated. First three numbers on each line is position of the taxel in the base frame and the last one is its index on given skin part.
- activatedTaxels - integer. Number of activated taxels.
- taxelsOriginal - 384×3 array with all taxels of given skin part expressed in a local frame.
- taxelsNew - 384×3 array with new position of all taxels after optimization expressed in the local frame.
- angles - struct with all joints and corresponding joint angles.

This form of dataset is the essential dataset used for all kinds of visualization and also for the optimization.

3.5.3 Other datasets

Dataset from `prepareData.m`

This dataset is created by calling the `Opt/Utils/prepareData.m` function and contains information collected by comparing two skin parts and serves as an input into the optimization. Fields in the struct are named after taxels on the first skin part (the optimized one).

For example, if the taxel selected on the first skin was number 50, the output looks like this:

- taxelStruct.s50 - struct for the given taxel
  - .data - $1 \times n$ array with struct containing mainly the joint angles

- .newTaxels - $1 \times n$ vector of taxels of the second skin part expressed in the base frame (indexes of the taxels on the second skin part, if the simultaneous calibration is enabled)
- .distances - $1 \times n$ vector of Euclidean distances

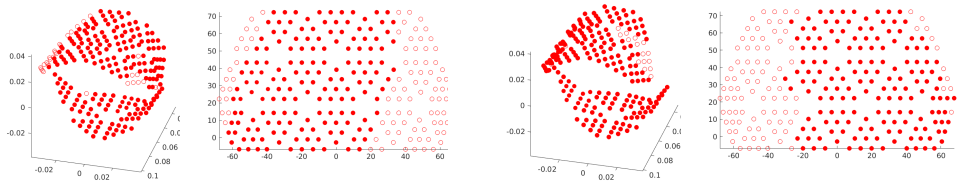
Datasets derived from *taxelStruct*

These datasets are derived from dataset described in Section 3.5.3. They are used for optimizing the pose of the patches and triangles.

- dataset - struct
  - .data - $1 \times n$ array with struct containing mainly the joint angles
  - .chain1 - $1 \times n$ vector containing taxel on the optimized skin part in its local frame
  - .chain2 - $1 \times n$ vector containing taxel on second skin part expressed in the base frame (in the local frame if simultaneous calibration is enabled)
  - .triangles - $1 \times n$ vector of triangle indexes corresponding to taxels in the *.chain1* field.
  - .triangles2 - $1 \times n$ vector of triangle indexes corresponding to taxels in the *.chain2* field.
  - .patches - $1 \times n$ vector of patch indexes corresponding to taxels in the *.chain1* field.
  - .patches2 - $1 \times n$ vector of patch indexes corresponding to taxels in the *.chain2* field.

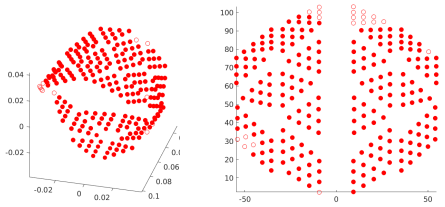
3.5.4 Gathered datasets

Activations on each skin part can be seen in Figure 3.5. When we use configurations where both hands touch the torso or the hand, datasets are combined. In Figure 3.6 the Nao with exposed skin can be seen for better idea about the position of the visualized skin parts on the real robot.

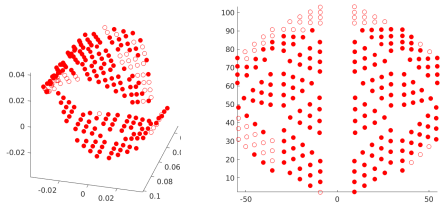


(a) : Right hand – torso.

(b) : Left hand – torso.



(c) : Right hand – head.



(d) : Left hand – head.

Figure 3.5: Activations on the given skin parts for different dataset. Red taxel means taxels activated at least one time in the dataset.

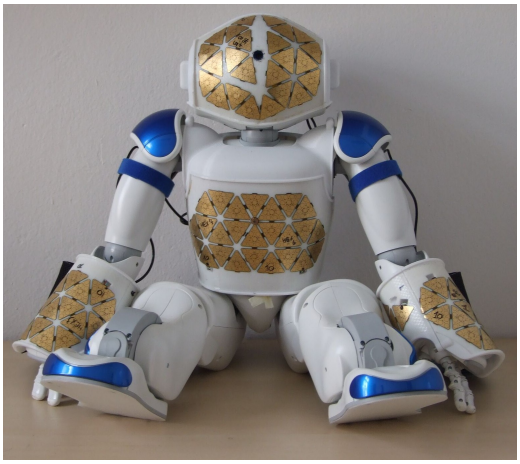


Figure 3.6: Nao with exposed skin.

Chapter 4

Results

In this section, we will discuss experiments using the framework to compare different approaches to calibration. We want to find which one is the superior approach. All of the experiments are evaluated on the training dataset (about 1000 touches for each self-touch configuration) and independently gathered testing dataset (about 50 touches for each self-touch configuration) .

4.1 Single skin part calibration using torso

In this section, we show results of calibration of the right and left hands assuming the torso pose is correct.

4.1.1 Right hand

The first optimized part is the right hand. We decided to try the following configurations:

- Sequential calibration of the plastic mounts, patches and triangles in this order
- Optimization of the plastic mounts, patches and triangles all at once, with prior optimization of plastic holders
- Optimization of the plastic mounts, patches and triangles all at once, with no prior optimization and no bounds
- Optimization of the plastic mounts, patches and triangles all at once, with no prior optimization and bounds for patches and triangles
  - With prior knowledge of the shape of the skin, we decided to bound DH parameters of the triangles in the following order: a[mm], d[mm], alpha[rad] and theta[rad] with ranges [-1 to 1], [-1 to 1], [-pi/20 to pi/20] and [-pi/20 to pi/20], respectively.
  - The patches were bounded with similar values in the following order: a[mm], d[mm], alpha[rad] and theta[rad] with ranges [-1 to 1], [-inf to inf], [-pi/20 to pi/20] and [-pi/20 to pi/20].

Figure 4.1 shows box plots of all of the configurations, where the boxes include values from 25th to 75th percentile. We can see, that all of the approaches resulted in a very similar training error, but the testing errors show differences, mainly the one configuration without bounds (**e)** in the figure) shows higher error. The best testing error was achieved for the calibration of plastic, patches and triangles in sequence (**b)** in the figure) with

median of 4.88mm. Visualization of corresponding results can be seen in Figure 4.3 .The highest number of outliers can be seen for the configuration with bounds but no prior calibration (**d)** in the figure), which is much higher than for the other configurations.

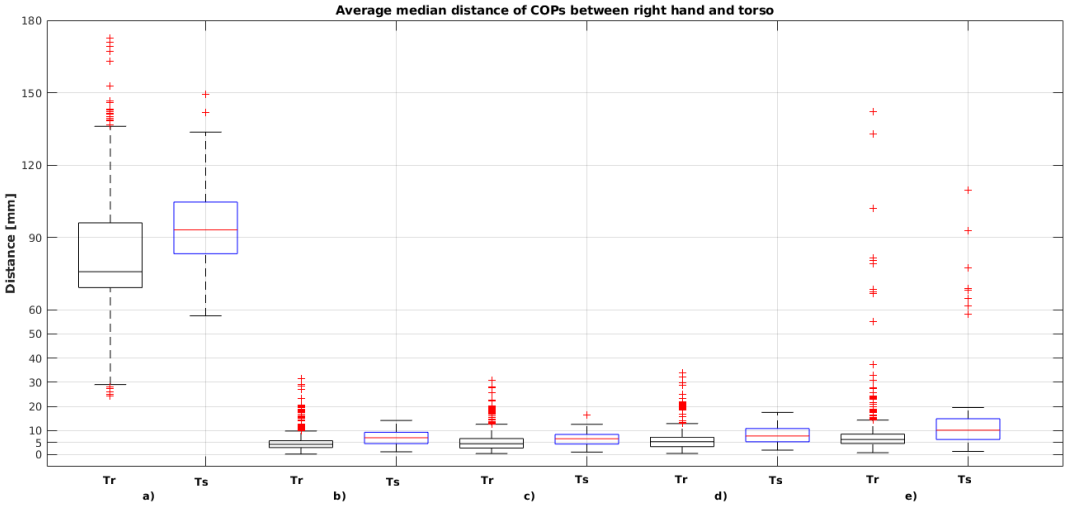
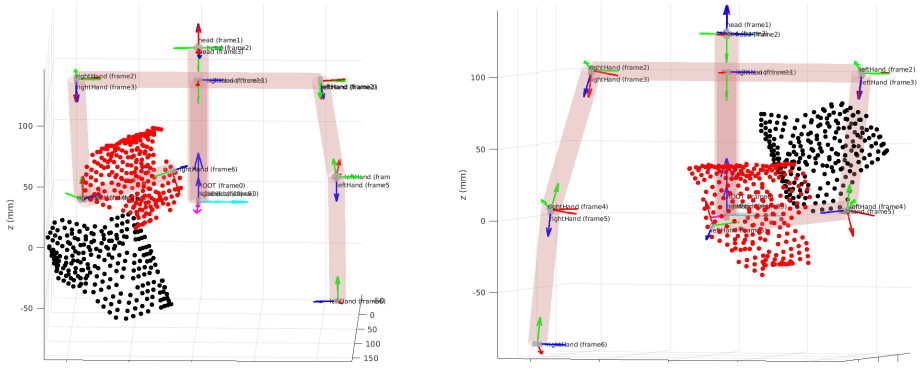


Figure 4.1: **a)** Before optimization, **b)** Optimization of plastic, patches and triangles in sequence **c)** After optimizing plastic, patches and triangles all at once, with prior optimization of the plastic **d)** Same as c) but with no prior optimization and bounds for patches and triangles, **e)** same as d) but with no bounds. **Tr** means error over training and **Ts** error over testing dataset.

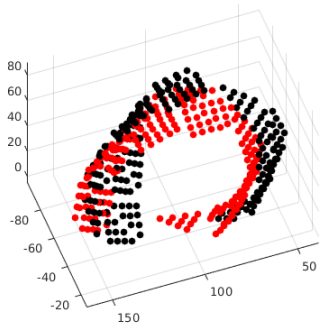


(a) : Right hand. **(b) :** Left hand.

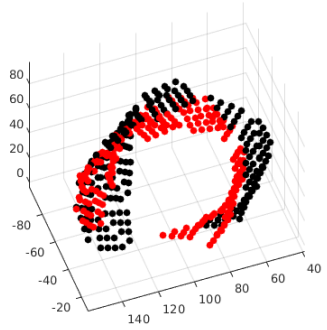
Figure 4.2: Difference between non-optimized (black) and optimized(red) plastic mount.

As can be seen in Figure 4.2 the original pose of the skin part is very distant from the real one (this is caused by bad rotation of the skin part in *CAD* model, which was mentioned in Section 2.5). So we decided to rotate the original values to get an estimated pose of the skin part for better visual comparison of the calibrations. These comparisons are in Figure 4.3. The figure visually proves the behaviour observed in the box plots. Configurations **a)** and **b)** converged into similar pose just with slight deviation to the estimated model. But

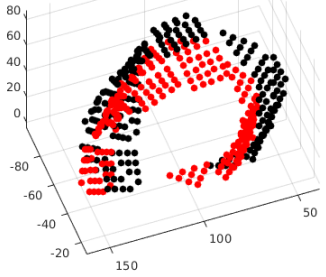
configuration **c)** is more divergent and **d)** is noticeably different as it lost the shape of the skin part.



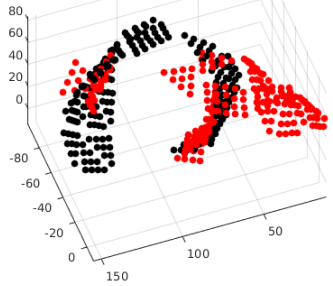
(a) : After optimization of plastic, patches and triangles in sequence.



(b) : After optimization of plastic, patches and triangles all at once with prior calibration of plastic.



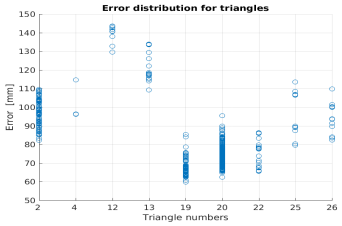
(c) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, bounds for patches and triangles.



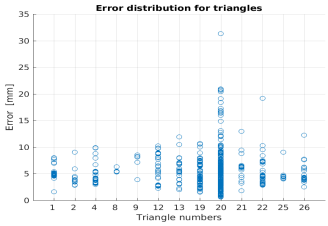
(d) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, no bounds.

Figure 4.3: Comparison of the right hand skin part after given optimization configuration (red) and estimated position (black).

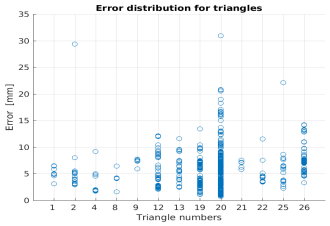
Figures 4.4 shows distributions of the paired taxels during parsing for the calibration (described in Section 3.2.2). It can be noticed, that configurations **d)**, **e)** (respectively **i)**, **j)** for distribution over taxels) again show bigger errors than the other configurations. These graphs will be shown only for the right hand.



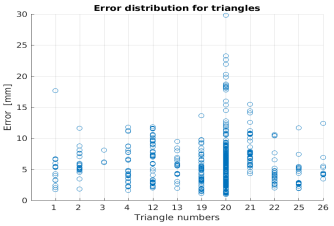
(a) : Before optimization.



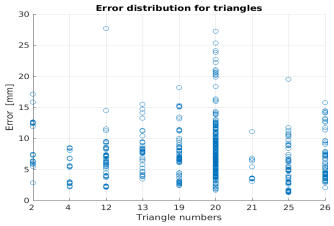
(b) : After optimization of plastic, patches and triangles in sequence.



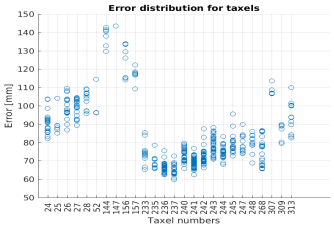
(c) : After optimization of plastic, patches and triangles all at once with prior calibration of plastic.



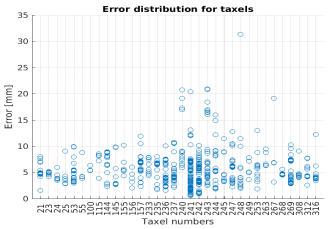
(d) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, bounds for patches and triangles.



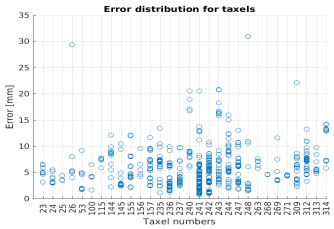
(e) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, no bounds.



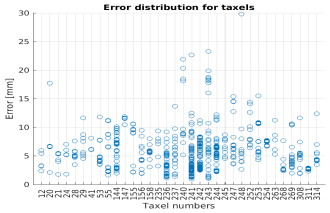
(f) : Before optimization.



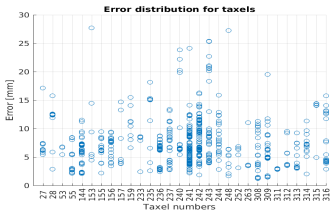
(g) : After optimization of plastic, patches and triangles in sequence.



(h) : After optimization of plastic, patches and triangles all at once with prior calibration of plastic.



(i) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, bounds for patches and triangles.



(j) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, no bounds.

Figure 4.4: Distribution of distances between paired taxels. From **a)** to **e)** sorted by triangles and from **f)** to **j)** for taxels by the optimized skin part.

4.1.2 Left hand

For the left hand we chose the same configurations as for the right hand (see 4.1.1). The Graphs 4.5 and 4.6 show the same trend as the corresponding figures for the right hand skin part. All of the configurations have similar median distances over the training set, but the not bounded configuration has bigger testing error. The visualization in Figure 4.6 **d)** shows that the shape of the skin part was as well as for the right arm not respected.

The difference against the right hand is that now the configuration with prior calibration of plastic mount pose demonstrates worse performance on the testing dataset than the one without any prior calibration (**c)** vs **d)** in the box plots). Also we had to bound even the sequential calibration (with the same bounds mentioned in 4.1.1), because the lower patch tends to shift too much. This resulted in decision, that we started to bound all other calibrations.

The left hand also has worse overall error (4.88 mm vs 5.79 mm). It can be caused by different size of the dataset (1072 poses for the right hand, 810 for the left hand) or by more "complicated" poses in the dataset. While collecting the dataset we focused on activating as many taxels as possible, but that sometimes resulted in complicated poses of the robot, which could produce error in the optimization process.

Based on the results of the calibration of the skin on the right and left arm we concluded that for the successful optimization prior calibration of the plastic mounts is crucial. Therefore, for all further optimization, prior calibration of plastic mounts will be used.

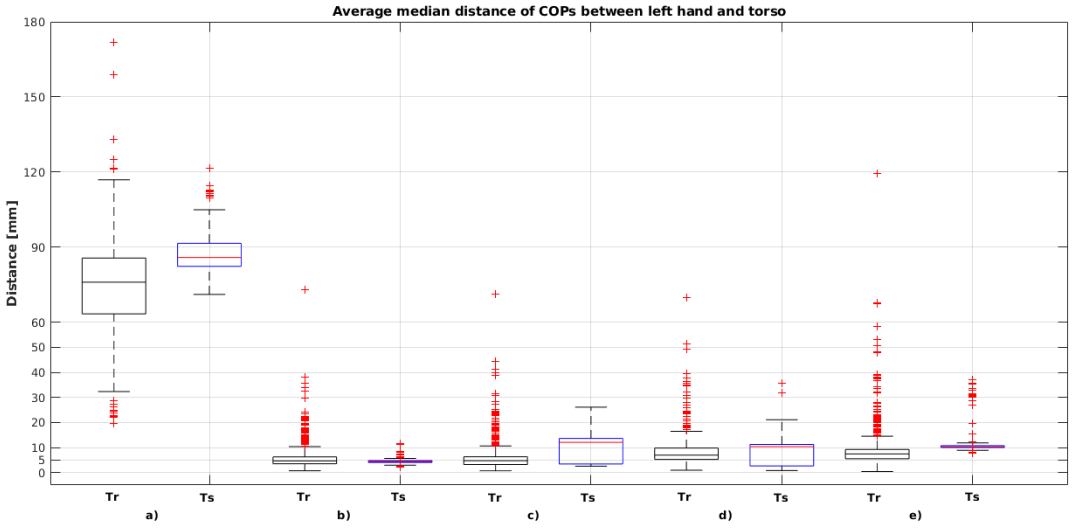
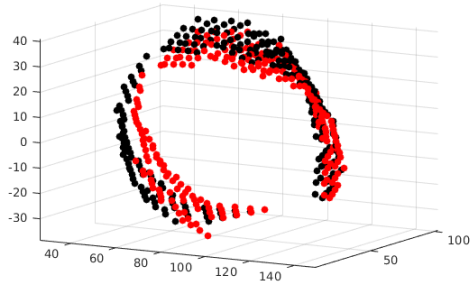
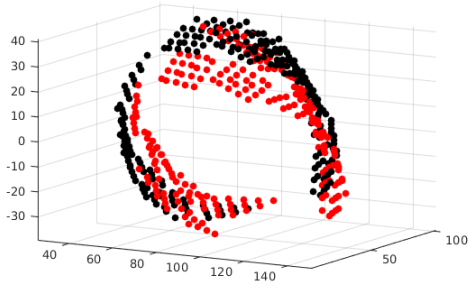


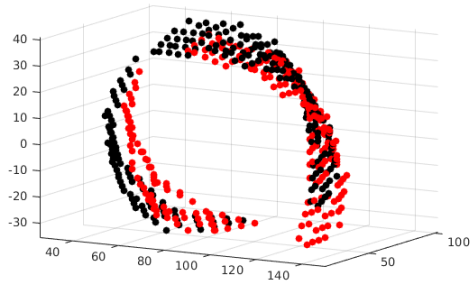
Figure 4.5: **a)** Before optimization, **b)** Optimization of plastic, patches and triangles in sequence **c)** After optimizing plastic, patches and triangles all at once, with prior optimization of the plastic **d)** Same as c) but with no prior optimization and bounds for patches and triangles, **e)** same as d) but with no bounds. **Tr** means error over training and **Ts** error over testing dataset.



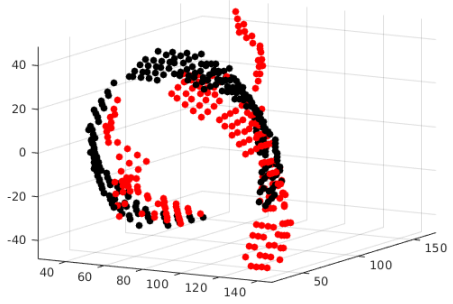
(a) : After optimization of plastic, patches and triangles in sequence.



(b) : After optimization of plastic, patches and triangles all at once with prior calibration of plastic.



(c) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, bounds for patches and triangles.



(d) : After optimization of plastic, patches and triangles all at once without prior calibration of plastic, no bounds.

Figure 4.6: Comparison of the left hand skin part after given optimization configuration (red) and before optimization with rotation made by hand (black).

4.2 Calibration of multiple skin parts simultaneously

In this section, we will discuss the case where both touching skin parts are calibrated simultaneously (right or left hand with torso). We split the section again in two parts to compare the results on both hands. As we already mentioned, we now use bounds for all configurations unless otherwise stated. Also we do not calibrate the triangles while performing the simultaneous optimization, because it would mean estimating at least 256 parameters, which is not feasible with our size of the dataset. Datasets include 1000 activations at max and some of the taxels were activated only one or two times (some of them never) which does not allow to calibrate all of the triangles.

4.2.1 Torso and right hand

For this experiment the following configurations were selected:

- Calibration of the plastic mounts and patches in sequence with full prior calibration of the hand
- Calibration of the plastic mounts and patches in sequence with prior optimization of the plastic mounts of the hand
- Calibration of the plastic mounts and patches at one time with prior optimization of the plastic mount of the hand
- Calibration of the plastic mounts and patches at one time with prior optimization of the plastic mount of the hand and without bounds

The box plots are shown in Figure 4.7. The results indicate, that all of those calibration approaches achieve nearly the same outcome (differences are in range of $\pm$ 0.5 mm). Anyway, the configuration with the full prior calibration of the hand (**b)** in the figure) achieves the best training error, smaller box of of 25th to 75th percentile and the least number of outliers. The second best results are gained from the sequential calibration with prior calibration of the plastic mounts (**c)** in the figure). This confirms results from the previous section as the sequential calibration is the best approach.

On the other hand, the non-bounded configuration shows the worst statistics over both training and testing datasets. This affirms that well set bounds are essential.

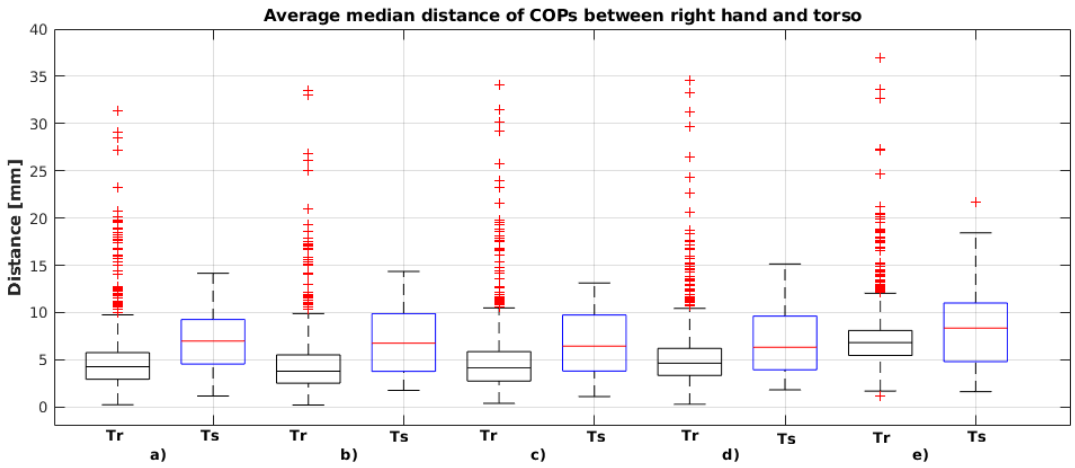
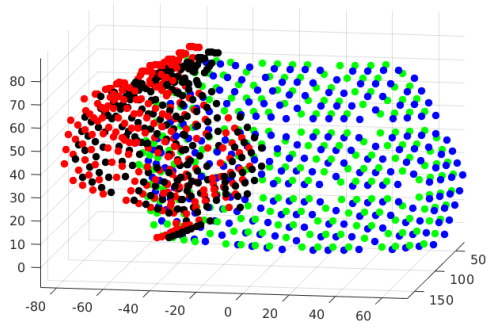
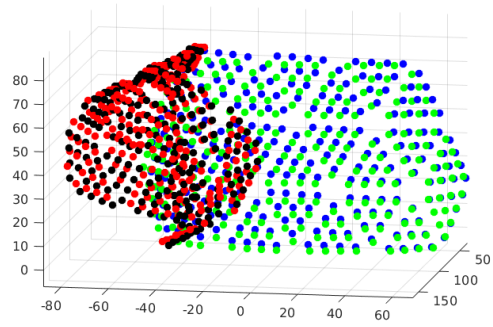


Figure 4.7: **a)** Optimization of plastic, patches and triangles of the right hand in sequence , **b)** optimization of the plastic mounts and patches in sequence with full prior calibration of the hand, **c)** optimization of the plastic mounts and patches in sequence with prior calibration of the plastic holder, **d)** optimization of the plastic mounts and patches at once with prior calibration of the plastic with bounds, **e)** same as d) but with no bounds. **Tr** means error over training and **Ts** error over testing dataset.

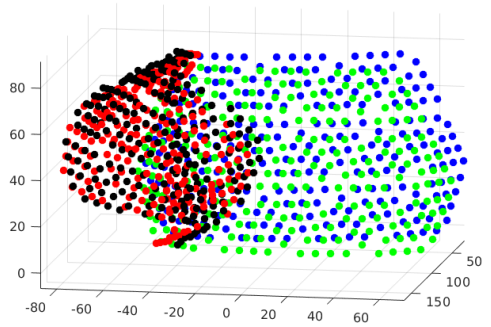
Visualizations in Figure 4.8 justify our assessment. As we assume that the torso was in approximately right position even before the calibration, the configuration which shows the best performance in the box plots had changed the pose of the torso and the right hand just in millimeters (about $\pm$ 2mm). The other two configurations shifted the torso more (up to 10mm).



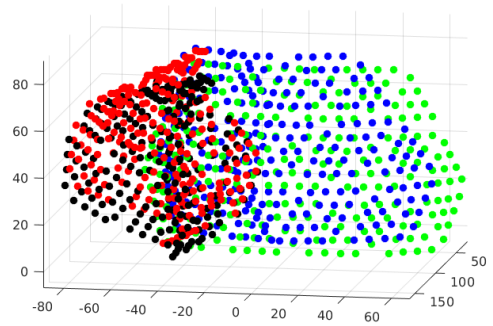
(a) : optimization of the plastic mounts and patches in sequence with full prior calibration of the hand.



(b) : optimization of the plastic mounts and patches in sequence with prior calibration of the plastic holder.



(c) : optimization of the plastic mounts and patches in sequence with prior calibration of the plastic with bounds.



(d) : optimization of the plastic mounts and patches at once with prior calibration of the plastic with no bounds.

Figure 4.8: Comparison of the torso before (green) and after(blue) calibration and the right hand before(red) and after(black) calibration.

4.2.2 Torso and left hand

The box plots in Figure 4.9 show similar situation as the box plots for the right hand. The only difference is that now the sequential optimization (**c)**) is worse than optimizing the plastic mounts and patches at one time (**d)**). But it can be just coincidence in choosing testing dataset which fits well for this configuration, because if we look in Figure 4.10, where Subfigure **c)** shows that the torso moved and rotated far from the original position which is probably not right with respect to the situation in Subfigure **a)**, which includes the configuration with the lowest error on both datasets.

Interesting situation is displayed in Subfigure **d)**, where lack of activations on the right patch of the torso caused its shift totally out of the torso. This definitely confirms the importance of the bounds.

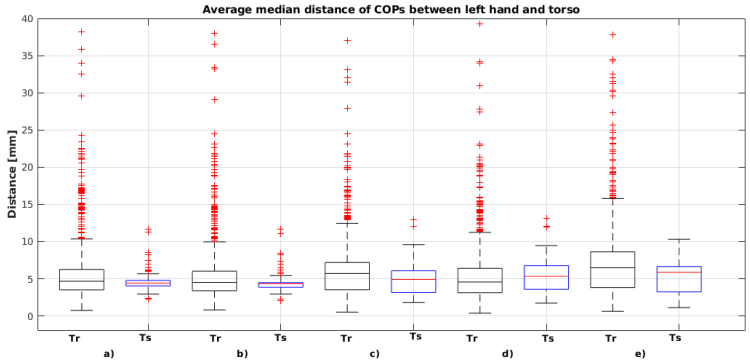
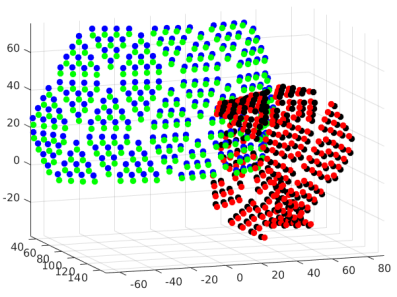
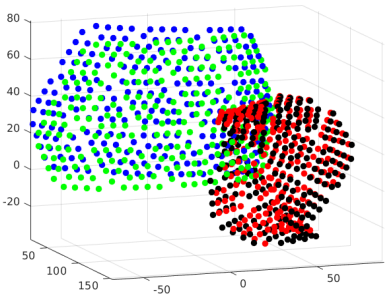


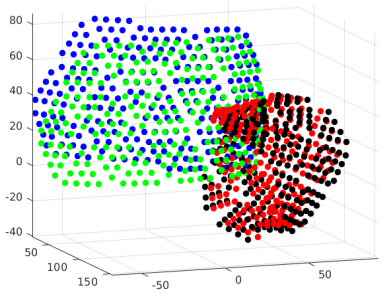
Figure 4.9: **a)** Optimization of plastic, patches and triangles of the left hand in sequence, **b)** optimization of the plastic mounts and patches in sequence with full prior calibration of the hand, **c)** optimization of the plastic mounts and patches in sequence with prior calibration of the plastic holder, **d)** optimization of the plastic mounts and patches at once with prior calibration of the plastic with bounds, **e)** same as d) but with no bounds. And **Tr** means error over training and **Ts** error over testing dataset.



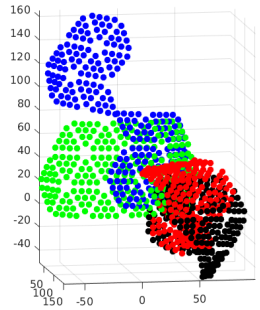
(a) : optimization of the plastic mounts and patches in sequence with full prior calibration of the hand.



(b) : optimization of the plastic mounts and patches in sequence with prior calibration of the plastic holder.



(c) : optimization of the plastic mounts and patches in sequence with prior calibration of the plastic with bounds.



(d) : optimization of the plastic mounts and patches at once with prior calibration of the plastic with no bounds.

Figure 4.10: Comparison of the torso before (green) and after(blue) calibration and the right hand before(red) and after(black) calibration.

4.3 Multiple chains combinations

In this section, we will look at configurations, where both of the hands were touching the third skin part (the torso or the head).

4.3.1 Torso, right hand, left hand

We want to compare the two approaches discussed above (one skin part as a reference vs. simultaneous touch). We examined the configuration, where torso was touched by both hands. In view of previous result the tested configurations are these:

- Calibration of the plastic, patches and triangles of the torso in sequence, where the hands are taken as reference and both have the plastics, patches and triangles optimized
- Simultaneous calibration of the plastic, patches of the torso and the hand in sequence, where both of the hands have the plastics, patches and triangles optimized

Figure 4.11 is divided into three parts - error between the left hand and the torso, error between the right hand and the torso and combined error (mean from the previous two). As can be seen, the non-simultaneous calibration is better in all three cases and over both of the datasets. The visualizations in Figure 4.12 are not so clear this time as both of the configuration changed the pose of the torso just in millimeters.

But if we also consider that the simultaneous calibration is much more computationally demanding, then the second option is more preferable.

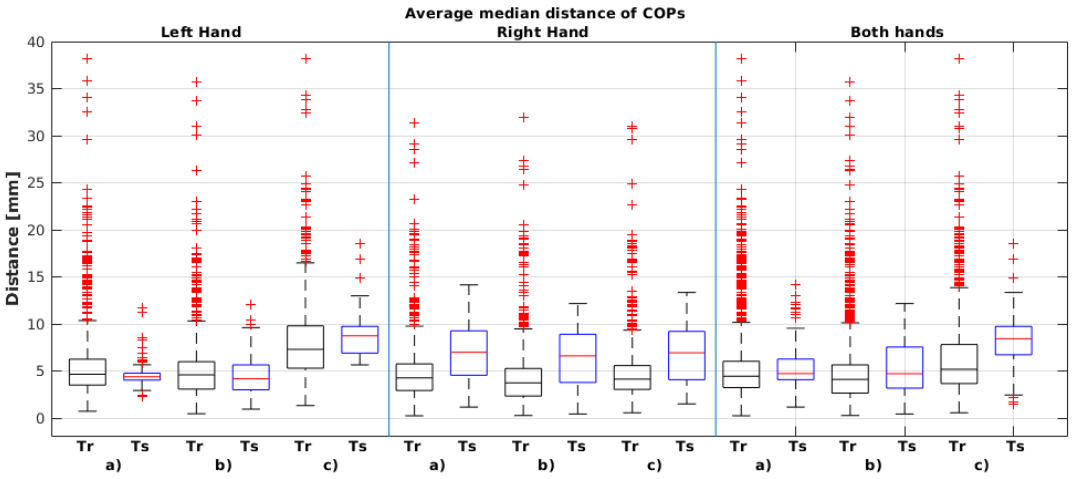
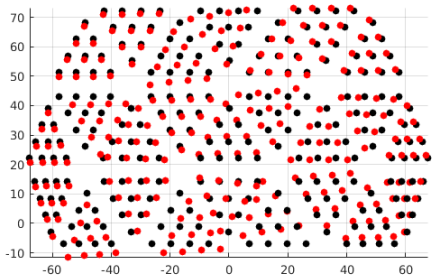
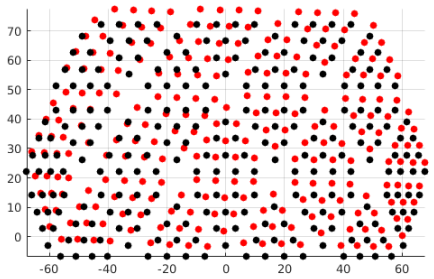


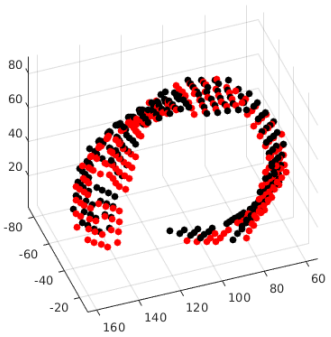
Figure 4.11: **a)** Optimization of plastic, patches and triangles of the hands in sequence ,**b)** Calibration of the plastic, patches and triangles of the torso in sequence, when the hands are taken as reference and both have the plastics, patches and triangles optimized, **c)** Simultaneous calibration of the plastic, patches of the torso and the hand in sequence, when both of the hands have the plastics, patches and triangles optimized. **Tr** means error over training and **Ts** error over testing dataset.



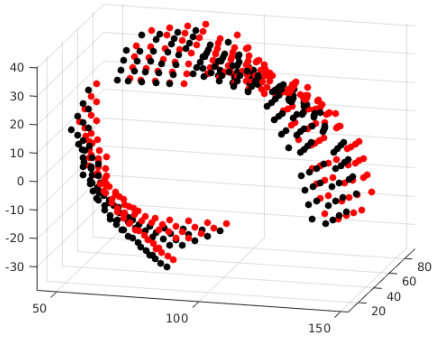
(a) : Torso from calibration of the plastic, patches and triangles of the torso in sequence, when the hands are taken as reference.



(b) : Torso from calibration, where the torso and the hands are optimized simultaneously.



(c) : Right hand from calibration, where the torso and the hands are optimized simultaneously.



(d) : Left from calibration, where the torso and the hands are optimized simultaneously.

Figure 4.12: Comparison of the torso before (black) and after (red) calibration. And comparison of the hands for the configuration of simultaneous calibration of the torso and the hands.

4.3.2 Head, right hand, left hand

As we observed, that it is better to take one skin part as a reference, this section does not include experiment with simultaneous touch. But it helps to prove statements from before, whether is it more beneficial to calibrate sequentially or all at once.

Figure 4.13 is again divided into three parts. We can see for all cases that we can calibrate parameters of the skin even in the case of higher original offset which is also visible in Figure 4.14a. The next conclusion is that the sequential calibration is better than calibration all parts at one time. It has lower errors on both dataset and also lower number of outliers in all cases. This data are visualized in Figure 4.15 and we can seen, that the optimization of all parts at one time (**d)**) did not preserve the shape of the head.

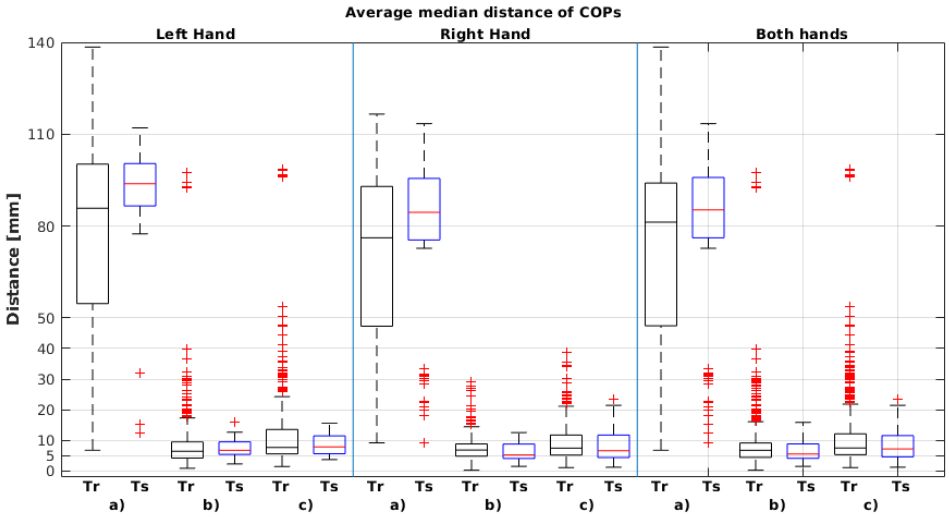
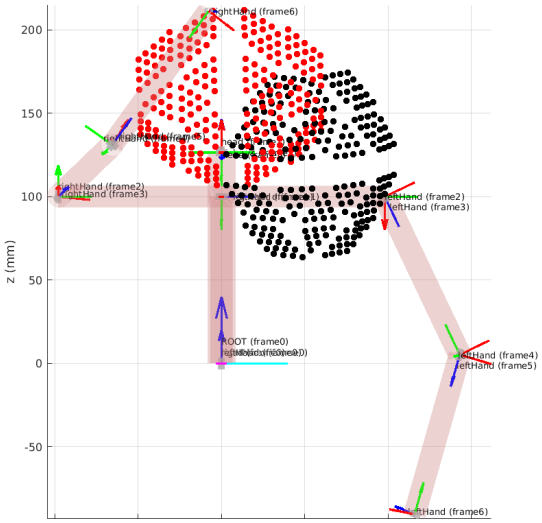


Figure 4.13: **a)** Before optimization, **b)** optimization of the plastic, patches and triangles of the head in sequence with full prior calibration of both hands, **c)** optimization of the plastic, patches and triangles of the head all at once with full prior calibration of both hands. **Tr** means error over training and **Ts** error over testing dataset.

With respect to other results we can state, that the sequential calibration is better. The first three Subfigures in Figure 4.15 show how the pose of the head changed with calibration. The result after calibration of the plastic mounts, patches and triangles (**c)**) has the bottom triangles covered over each other. In Figure 4.14b we can see the real robot, where the triangles are also over each other. It is great confirmation of our results.



(a) : Comparison of the head before (black) and after calibration of the plastic mounts (red).

(b) : Triangles covering each other on the head of the robot.

Figure 4.14: Visualized head before and after calibration and head of the real robot.

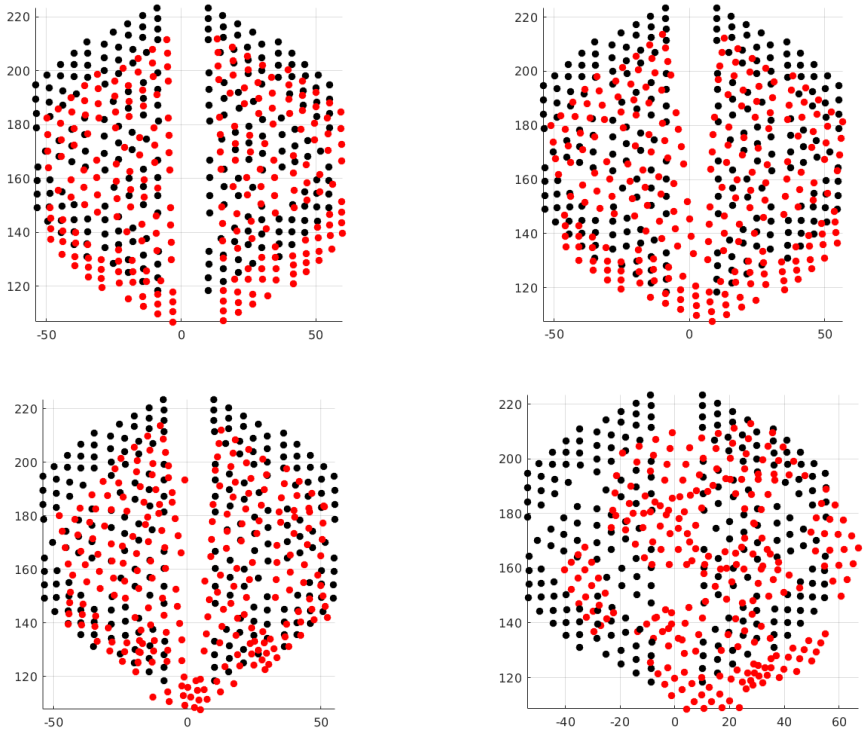


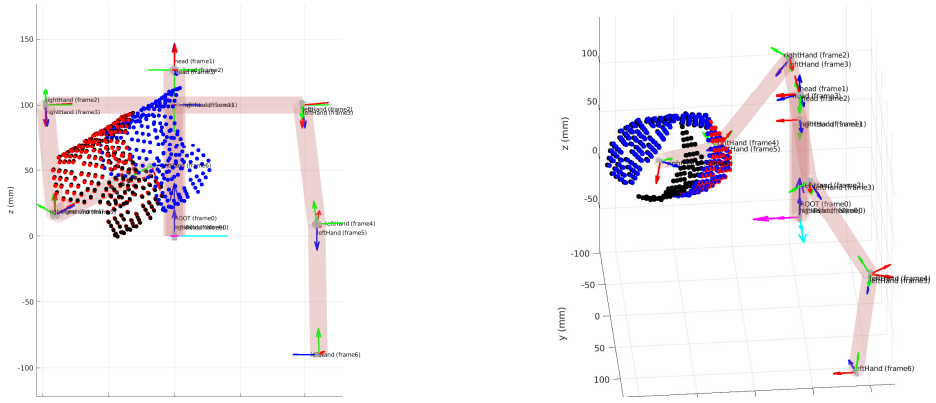
Figure 4.15: Comparison of the pose of the head before (black) and after(red) optimization.

4.4 Initial parameter perturbations

To prove that the optimization is working, we can perturb the initial position and run the optimization again. We have chosen to try two configuration, where we shifted the whole plastic 5 cm in the direction of the axis x, which is the direction of the last link. We did the same for the lower patch (we did not perturbed the triangles). In Figure 4.16 we can see visualization of this configuration and output after the optimization.

As can be seen, in Figures 4.16 the skin part gets to the same position as it was before perturbation for both of the configurations. But it requires more runs of the optimization, because with every change of position different two *COPs* are paired. In our case, it took nine runs (more about this is mentioned in Section 2.5.4). Table 4.1a shows changes in parameters in individual runs for the perturbation of the whole plastic. The return to the start position is also confirmed by difference between the first and the last values of parameter a, which in this case represent translation in direction of axis x. The difference is 49,5mm, so the error from perturbed value (50 mm) is just 0.5mm and this distance would decrease with more runs. The difference in other parameters is negligible.

In Table 4.1b, the same is shown for the perturbation of the lower patch. As can be seen the difference between first and last value of parameter a is about 5mm, which is higher than for the whole plastic. We did not continue in the calibration, because the difference between each other run was small and it would take large amount of time.



(a) : Perturbation of the whole plastic.

(b) : Perturbation of the lower patch.

Figure 4.16: Visualized situation of perturbation. **a)** Black shows situation before and blue after perturbations, the red shows situation after calibration. **b)** Blue shows situation before and black after perturbations, the red shows situation after calibration.

Run	a [mm]	d [mm]	α [rad]	θ [rad]
0	14.389	-4.495	2.777	-1.584
1	-3.323	-4.113	2.727	-1.575
2	-11.234	-3.935	2.681	-1.559
3	-23.409	-3.921	2.714	-1.551
4	-26.877	-3.367	2.716	-1.554
5	-29.903	-4.028	2.714	-1.569
6	-32.337	-4.278	2.735	-1.576
7	-33.658	-4.252	2.744	-1.577
8	-34.760	-4.417	2.754	-1.577
9	-35.117	-4.272	2.754	-1.583

(a) : Table of runs of the optimization after the perturbation of the right hand plastic mount

Run	a [mm]	d [mm]	α [rad]	θ [rad]
0	0.658	2.396	-0.020	0.019
1	-35.556	0.412	-0.043	0.019
2	-40.899	1.168	0.015	-0.385
3	-42.248	1.507	0.024	-0.035
4	-43.475	1.692	0.044	-0.052
5	-44.097	1.745	0.0438	-0.045
6	-44.7000	1.931	0.051	-0.037
7	-44.885	2.116	0.035	-0.003
8	-44.958	2.268	0.040	0.001

(b) : Table of runs of the optimization after the perturbation of the right hand lower patch

Table 4.1: Table of changes in parameters during calibration after perturbation.

Figure 4.17 shows distribution of distances between paired taxels for the perturbation of the lower patch. The middle Subfigures displays how all distances on the lower patch increased and then on the third Subfigures got back to initial position.

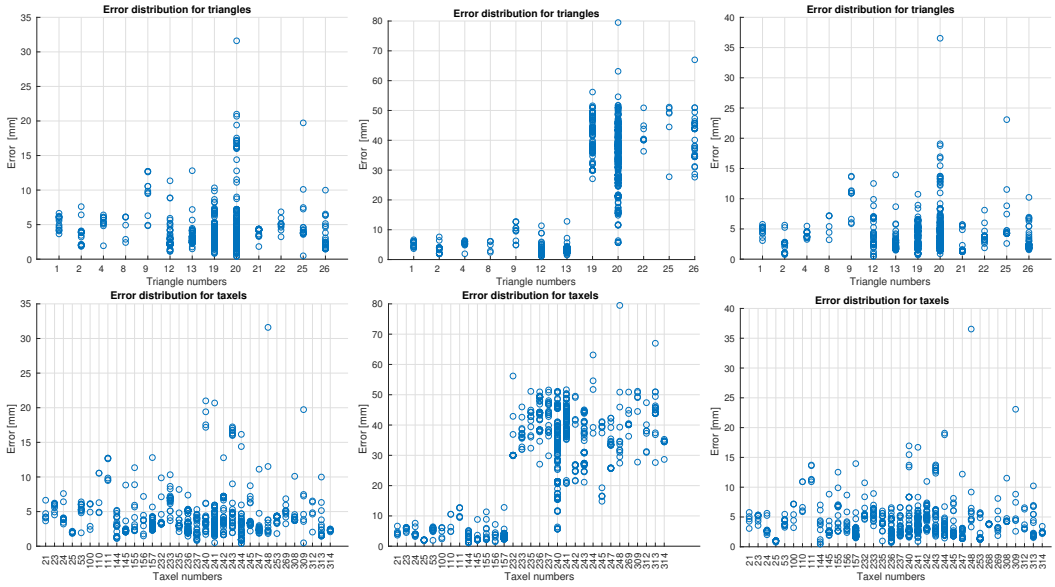


Figure 4.17: Euclidean distances in mm for the right hand and the torso between compared taxels for each triangle, which has *COP* detected on it, without perturbation (upper left), with perturbation before (upper middle) and after (upper right) the optimization. In the lower figures the same configuration is shown with distances for taxels closest to the *COPs*.

4.5 Summary

In Figure 4.18 we can see all experiments with right hand. From the box plots we can see that the best performance are for the configurations **e)** and **i)**, when slightly better is configuration **i)**. So, as we already mentioned, the best approach is to optimize the plastics, patches and triangles in sequence with torso taken as reference and then adjust the error by calibrating the torso from touches of both hands, where the hands are taken as reference. This is consistent with result in section about calibration of the head 4.3.2.

Configuration	Training poses over *COPs* [mm]	Testing poses over *COPs* [mm]	Training poses over taxels [mm]	Testing poses over taxels [mm]
Right hand - torso	4.32	6.42	2.25	2.94
Left hand - torso	5.49	4.83	2.67	1.65
Both hands torso	4.91	5.62	2.3	2.46
Right hand - head	7.18	6.07	2.83	2.80
Left hand - head	8.32	7.43	4.91	3.41
Both hands - head	7.76	6.75	3.87	3.15

Table 4.2: Table of results for all datasets with the sequential approach.

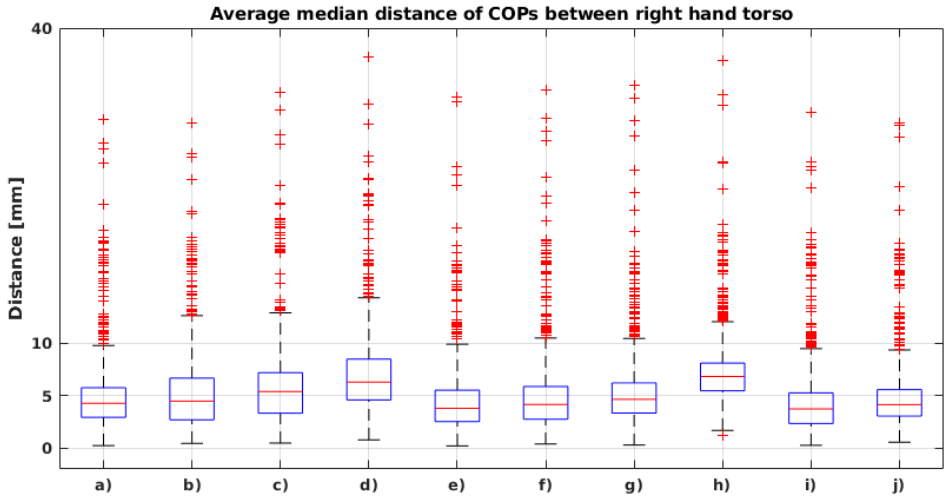


Figure 4.18: Training errors of all tested configuration for the right hand.
a) Optimization of plastic, patches and triangles of the hand in sequence **b)** After optimizing plastic,patches and triangles of the hand all at once, with prior optimization of the plastic **c)** Same as b) but with no prior optimization and with bounds for patches and triangles, **d)** same as c) but with no bounds, **e)** optimization of the plastic mounts and patches of the hand and the torso in sequence with full prior calibration of the hand, **f)** optimization of the plastic mounts and patches of the hand and the torso in sequence with prior calibration of the plastic holder, **g)** optimization of the plastic mounts and patches of the hand and the torso at once with prior calibration of the plastic with bounds, **h)** same as g) but with no bounds, **i)** Calibration of the plastic, patches and triangles of the torso in sequence, when the hands are taken as reference and both have the plastics, patches and triangles optimized, **j)** Simultaneous calibration of the plastic, patches of the torso and the hand in sequence, when both of the hands have the plastics, patches and triangles optimized.

In Table 4.2, we can see errors for all datasets with the sequential approach, which was selected as the best. We can take values from dataset with both hands with head and torso and make average from them and add our results to table of relative errors created by Albini et al. [24] and compare with other authors in Table 4.3. As can be seen, we achieved very similar results to the experiment performed in [24] on the Baxter robot.

Approach	Error [mm]
Del Prete et al. I [23]	7.2
Del Prete et al. II [23]	6.6
Mittendorfer et al. [39]	$\leq$10
Albini et al. [24]	$\leq$2.9
Our approach - training dataset	3.1
Our approach - testing dataset	2.8

Table 4.3: Comparison of different approaches

In Table 4.4 we can see changes in the four DH parameters of the new links. As we can see the biggest change was in the position of the plastic mounts of the hands. The torso

moved just a little bit, which is caused by the fact, we took it as reference frame in the start.

Segment	a [mm]	d [mm]	α [rad]	θ [rad]
Plastic mount	14.75	-4.43	2.747	-1.602
Upper patch	-0.57	-6.19	-0.079	-0.063
Lower patch	0.25	2.37	-0.016	0.030
Triangles - mean	-0.08	-0.04	0.008	-0.004
Triangles - std	0.34	0.37	0.053	0.043

(a) : Right hand.

Segment	a [mm]	d [mm]	α [rad]	θ [rad]
Plastic mount	15.69	-5.40	-2.975	-1.594
Upper patch	0.48	-4.82	-0.066	-0.047
Lower patch	-0.57	1.53	0.005	-0.005
Triangles - mean	-0.01	0.05	-0.004	0.015
Triangles - std	0.40	0.36	0.082	0.069

(b) : Left hand.

Segment	a [mm]	d [mm]	α [rad]	θ [rad]
Plastic mount	0.02	0.11	0.009	-0.003
Upper patch	-0.40	0.49	-0.003	-0.014
Lower patch	0.11	-0.73	0.005	0.015
Triangles - mean	-0.06	-0.11	0.012	0.008
Triangles - std	0.59	0.68	0.061	0.036

(c) : Torso.

Segment	a [mm]	d [mm]	α [rad]	θ [rad]
Plastic mount	-1.14	5.79	1.595	0.234
Upper patch	0.87	1.41	-0.022	-0.138
Lower patch	-2.69	2.79	0.066	0.112
Triangles - mean	-0.11	0.01	0.003	-0.003
Triangles - std	0.59	0.52	0.057	0.057

(d) : Head.

Table 4.4: Table with changes of DH parameters of each link in the sequential approach.

In Figure 4.19 can be seen the comparison of the real robot and the model with the artificial skin. The black skin shows the non-optimized position of the skin (not fully non-optimized, but with the rotation made by hand to estimate the pose) and the red skin shows differences after the calibration.

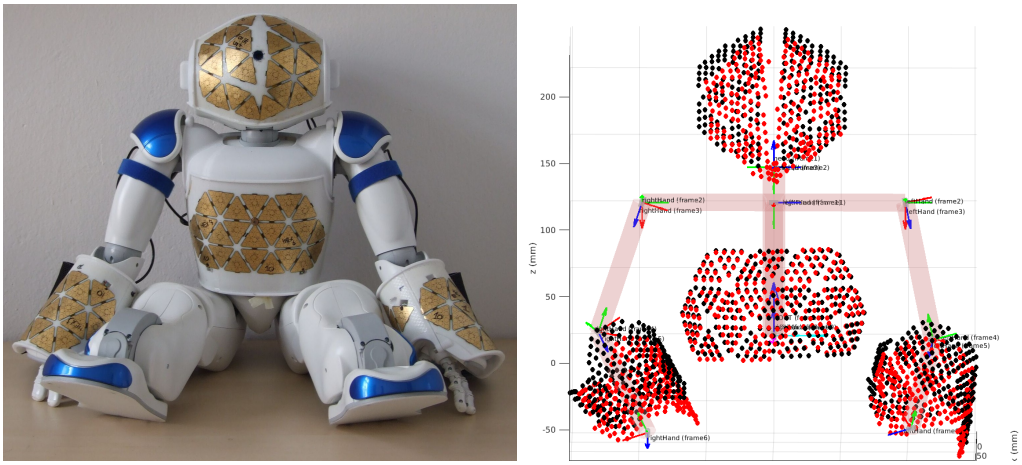


Figure 4.19: Comparison of the real robot and the model, which has visualized original position of the skin (black) and skin after calibration (red) using the sequential method.

Chapter 5

Discussion, conclusion and future work

We developed a system to gather datasets, communicating with the robot to read its joint angles together with the tactile information during self-touch configurations. From the tactile sensors, we read raw data as well as preprocessed information that clusters the stimulations. These data are saved for each touch and form a dataset. We gathered dataset for four configurations (right hand – torso, left hand – torso, right hand – head, left hand – head), where the dataset can be combined. The dataset with hands touching torso includes about 1000 touches for each hand and the dataset with the head about 600 touches. In the future, it would be beneficial to gather an even bigger dataset, because we were not able to optimize the pose of the triangles with the simultaneous calibration of the hands and torso, because it is not feasible to optimize all of the parameters. We created a GUI which visualizes activated taxels during a collection of the dataset, but this visualization should be improved by adding some kind of heat map because right now it only shows whether the taxels were or were not activated.

Besides the gathering GUI, we created a set of different visualization tools that help to evaluate the dataset and the results of the optimization. The ability to reconstruct every configuration from the dataset on the virtual model (in Matlab) helped to discover many problems during the work (e.g., errors in DH notation and forward kinematics were difficult to detect in the code, but the visualization showed that the arms of the robot are not moving the way they should have). The possibility to show the skin on this model was crucial to compare changes in the poses of the skin parts because as we could see through the entire Results section, it is always better to visualize the configuration than only looking at the numbers.

We created a first version of the multirobot framework. Right now it was tested only for the Nao robot. For other robots, only supplemental utilities are working at the moment. Everyone is capable to create the structure of their robot and change settings of every joint or visualize the robot with the Matlab model just with providing the joint angles. The work on this framework has just started, but it could be very promising later. The calibration part was tested on our Nao robot, and we proved that it is working. The calibration can estimate the DH parameters even if pose of the skin part is very different from the real one (see Figure 4.2 for comparison of non-optimized and optimized skin part or Figure 4.16 for visual representation of perturbations). In Figures 4.14b and 4.15 we can see that after the calibration the bottom triangles of the head skin part are covering each other as it is on the real robot. We think this is a solid argument to state the framework performs well.

We observed a set of configurations for each dataset to find the best approach and calibration sequence. In Sections 4.1 and 4.2, we found that it is better to bound the

patches and triangles to constrain the shape of the skin parts and also that it is better to optimize each segment of the skin part (plastic mounts, patches and triangles) in sequence rather than calibrating all at once. Then in Section 4.3.1 we performed experiments to determine if it is preferable to optimize when one skin part is taken as a reference or when all skin parts are optimized at one time, and we concluded that the simultaneous calibration performs worse. Finally, in Section 4.3.2 we confirmed that the sequential approach is better and the same outcome is visible in Figure 4.18 where all configurations for the right hand are shown. We can now declare that the superior way of calibration is to optimize both hands from the position of the torso, where each segment of the skin part is calibrated in sequence (in this order: plastic mounts, patches and triangles) and then adjust the pose of the torso and the head from the position of the hands. In the future, we could perform more experiments with different combinations of the chains to further prove our statements.

From Table 4.2, we can see that the error over the training dataset is a bit higher for configurations including the left hand. This can be caused by a number of reasons. We think one of the reasons is the lower size of the dataset (1741 for the right hand and 1309 for the left hand). Another problem could be noise in the data collection – activations in which the skin parts were not activated by self-touch but by touch with the person gathering the dataset or some other object around the robot. In future work, the kinematic constraints could be employed to verify the feasibility of particular self-touch configurations in the data. The performance could also improve if we perform more runs of the optimization for each configuration, because as we mention in Section 2.5.4, the *COPs* marked as the closest are changing when we use new DH parameters, but in our experiments we only run every optimization for each configuration two times.

Finally, we compared our results with other approaches in Table 4.3. We compared relative error over the taxels with other authors and achieved competitive results. It should also be noted that we did not touch with a small end-effector but with big skin parts, so we can not compute the distance between taxels directly: we compute the error over taxels for each activation as error over five (or less if activation does not have five activated taxels on both skin parts) taxels with the lowest Euclidean distance between them on different skin parts. In our approach, the self-touch was not autonomous like in [24, 25] but the robot was brought manually to the self-touch configurations. Taking into consideration that the triangles are covered by the fabric which has at least 1mm thickness on each of the skin parts in contact, the error of 3mm is a strong result and seems to approach the lower bound on the possible error. It should also be noted that all results are w.r.t. self-touch dataset (albeit split into training and testing) and no ground truth information was available.

Except for the already mentioned ideas, in the future, we could use also visual feedback as another source of data: looking at the skin on its hand using its own camera, the robot could also use self-observation for calibration (see e.g., [21]). Another improvement could be the implementation of some end-effector which would touch the other parts instead of touching with the second skin part. It could make the contact more point-like (like [24]) and with the right length, it could improve the range of touches to activate more taxels (from Figure 3.5 we can see that not all of the taxels were activated because it was not feasible to reach them). Both of these suggested improvements could also help to get us closer to an autonomous collection of the dataset by the robot.

Bibliography

[1] R. S. Dahiya, G. Metta, M. Valle, and G. Sandini. Tactile sensing—from humans to humanoids. *IEEE Transactions on Robotics*, 26(1):1–20, Feb 2010.

[2] Y. Ohmura, Y. Kuniyoshi, and A. Nagakubo. Conformable and scalable tactile sensor skin for curved surfaces. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1348–1353, May 2006.

[3] P. Mittendorfer and G. Cheng. Humanoid multimodal tactile-sensing modules. *IEEE Transactions on Robotics*, 27(3):401–410, June 2011.

[4] P. Mittendorfer and G. Cheng. Uniform cellular design of artificial robotic skin. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–5, May 2012.

[5] G. Cannata, M. Maggiali, G. Metta, and G. Sandini. An embedded artificial skin for humanoid robots. In *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 434–438, Aug 2008.

[6] P. Maiolino, M. Maggiali, G. Cannata, G. Metta, and L. Natale. A flexible and robust large scale capacitive tactile system for robots. *IEEE Sensors Journal*, 13(10):3910–3917, Oct 2013.

[7] A. Schmitz, P. Maiolino, M. Maggiali, L. Natale, G. Cannata, and G. Metta. Methods and technologies for the implementation of large-scale robot tactile sensors. *IEEE Transactions on Robotics*, 27(3):389–400, June 2011.

[8] `http://wiki.icub.org/wiki/ICub_versions`. [Online; accessed May-2019].

[9] E. Dean-Leon, K. Ramirez-Amaro, F. Bergner, I. Dianov, and G. Cheng. Integration of robotic technologies for rapidly deployable robots. *IEEE Transactions on Industrial Informatics*, 14(4):1691–1700, April 2018.

[10] John Hollerbach, Wisama Khalil, and Maxime Gautier. Handbook of robotics. In *Springer Handbook of Robotics*, pages 321–341. Springer, 2008.

[11] J. Hu, J. Wang, and Y. Chang. Kinematic calibration of manipulator using single laser pointer. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 426–430, Oct 2012.

[12] C. S. Gatla, R. Lumia, J. Wood, and G. Starr. An automated method to calibrate industrial robots using a virtual closed kinematic chain. *IEEE Transactions on Robotics*, 23(6):1105–1116, Dec 2007.

[13] M. Hersch, E. Sauser, and A. Billard. Online learning of the body schema. *International Journal of Humanoid Robotics*, 5:161–181, 2008.

[14] R. Martinez-Cantin, M. Lopes, and L. Montesano. Body schema acquisition through active learning. In *Proc. Int. Conf. on Robotics and Automation (ICRA)*, 2010.

[15] P. Mittendorfer and G. Cheng. Open-loop self-calibration of articulated robots with artificial skins. In *2012 IEEE International Conference on Robotics and Automation*, pages 4539–4545, May 2012.

[16] N. Guedelha, N. Kuppuswamy, S. Traversaro, and F. Nori. Self-calibration of joint offsets for humanoid robots using accelerometer measurements. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 1233–1238, Nov 2016.

[17] K. Yamane. Practical kinematic and dynamic calibration methods for force-controlled humanoid robots. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 269–275, Oct 2011.

[18] P. Mittendorfer and G. Cheng. 3d surface reconstruction for robotic body parts with artificial skins. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4505–4510, Oct 2012.

[19] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: An open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, pages 50–56, New York, NY, USA, 2008. ACM.

[20] A. Roncone, M. Hoffmann, U. Pattacini, and G. Metta. Automatic kinematic chain calibration using artificial skin: Self-touch in the icub humanoid robot. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2305–2312, May 2014.

[21] K. Stepanova, T. Pajdla, and M. Hoffmann. Robot self-calibration using multiple kinematic chains—a simulation study on the icub humanoid robot. *IEEE Robotics and Automation Letters*, 4(2):1900–1907, April 2019.

[22] G. Cannata, S. Denei, and F. Mastrogiovanni. Towards automated self-calibration of robot skin. In *2010 IEEE International Conference on Robotics and Automation*, pages 4849–4854, May 2010.

[23] A. Del Prete, S. Denei, L. Natale, F. Mastrogiovanni, F. Nori, G. Cannata, and G. Metta. Skin spatial calibration using force/torque measurements. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3694–3700, Sep. 2011.

[24] A. Albini, S. Denei, and G. Cannata. Towards autonomous robotic skin spatial calibration: A framework based on vision and self-touch. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 153–159, Sep. 2017.

[25] Alessandro Roncone. Visualization of kinematic chains in matlab. https://github.com/alecive/kinematics-visualization-matlab. [Online; accessed May-2019].

[26] Project repository. https://gitlab.fel.cvut.cz/body-schema/code-nao-skin-control/tree/master. [Online; accessed May-2019].

[27] Aldebaran. Nao documenttation. http://doc.aldebaran.com/2-1/family/index.html. [Online; accessed May-2019].

[28] Adam Rojik. Joint constraints for naoprague. https://docs.google.com/document/d/14eYPeTlPOEelmroKRqpS_ajDNbR8v7G-dnKC0xnRGJ0/edit#heading=h.vldrhxxpsuhd.

[29] Mark W. Spong. *Robot Dynamics and Control*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1989.

[30] Richard Scheunemann Denavit, Jacques; Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans ASME J. Appl. Mech. 23*, page 215–221, 1955.

[31] Skin repository. https://gitlab.fel.cvut.cz/body-schema/code-nao-skin. [Online; accessed May-2019].

[32] Hassan Saeed. Estimation of taxels positions. https://docs.google.com/document/d/14eYPeTlPOEelmroKRqpS_ajDNbR8v7G-dnKC0xnRGJ0/edit#heading=h.wf20ry7gzq8d. [Online; accessed May-2019].

[33] Maksym Shcherban. Skin coordinates generation. https://gitlab.fel.cvut.cz/body-schema/code-nao-simulation/tree/master/gazebo9/skin-generation/coordinates. [Online; accessed May-2019].

[34] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018.

[35] Aldebaran. Naoqi description. http://doc.aldebaran.com/1-14/dev/naoqi/index.html. [Online; accessed May-2019].

[36] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. Yarp: Yet another robot platform. *International Journal of Advanced Robotic Systems*, 3(1):8, 2006.

[37] skinmanager description. http://www.icub.org/doc/icub-main/group__icub_skinManager.html. [Online; accessed May-2019].

[38] Wiki for iCub and friends. Tactile sensors (aka skin). http://wiki.icub.org/wiki/Tactile_sensors_(aka_Skin). [Online; accessed May-2019].

[39] P. Mittendorfer, E. Dean, and G. Cheng. 3d spatial self-organization of a modular artificial skin. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3969–3974, Sep. 2014.

Appendix A

Default settings of the framework for Nao

All possible properties of the multirobot framework for the Nao can be seen here:

- `options` - Struct for the *lsqnonlin* solver
  - Algorithm - algorithm used for optimization.
    Default value: 'levenberg-marquardt'
  - MaxIter - maximum iterations of one run.
    Default value: 50
  - InitDamping - value of the parameter μ.
    Default value: 1000
  - MaxFunctionEvaluations - maximal number of function call.
    Default value: 99999
- part1 - name of the first chain used for given configuration.
  Default value: 'right_hand'
- part2 - name of the second chain.
  Default value: 'torso'
- part3 - name of the third chain.
  Default value: 'left_hand'
- name - name of the used dataset. Value is automatically created from selected chains.
- name - name of the second used dataset. Value is automatically created from selected chains.
- filename* - path to the files with input parameters for optimization. The paths are determined by the chosen chains.
- loadOptimized* - index of which input parameters from the files to use. For example value *0* means non-optimized, and *-1* means the last saved value.
  Default values: '-1'
- writeMode - determines whether to save parameters as a new entry ('a') or to replace the loaded ones ('w').
  Default values: 'w'
- optSecond - if true, the second chain is optimized.
  Default values: false
- structure - structure of the robot, used by *joint* class.

- modelStructure - structure of chains for the visualization.
- selftouch - determines whether simultaneous touch is used.
  Default value: false
- twoChain - if true, more than two chains are used.
  Default values: false
- offsets - settings for optimization of the plastic holders.
  - optimize - determines whether to start this optimization or not.
    Default value: true
  - repetitions - a number of repetitions for each optimization run.
    Default value: 20
  - optRepetitions - a number of optimization runs.
    Default value: 1
  - batchSize - size of data used every repetition from the training dataset.
    Default value: 1
  - DH - 4x1 array with true/false values, which says if to optimize/not to optimize given DH parameter.
    Default value: [true,true,true,true]
  - bound - determines whether to use bounds for optimization. If true, the algorithm has to change to 'trust-region-reflective', because 'levenberg-marquardt' does not support bounds.
    Default value: false
  - lowerBounds - lower bounds, which limit optimization by the minimal values of DH parameters.
    Default value: [-inf,-inf,-pi, -pi]
  - upperBounds - maximal values.
    Default value: [inf,inf,pi,pi]
- patches - settings for optimization of the whole patches
  - optimize - determines whether to start this optimization or not.
    Default value: true
  - repetitions - a number of repetitions for each optimization run.
    Default value: 1
  - optRepetitions - a number of optimization runs.
    Default value: 1
  - batchSize - size of data used every repetition from the training dataset.
    Default value: 1
  - bound - determines whether to use bounds for optimization. If true, algorithm has to change to 'trust-region-reflective', because 'levenberg-marquart' does not support bounds.
    Default value: false
  - lowerBounds - the lower bounds, which limit optimization by the minimal values of DH parameters.
    Default value: [-1,-inf,-pi/20, -pi/20]

- upperBounds - maximal values.
  Default value: [1,inf,pi/20,pi/20]
- hands_upper, hands_lower, head_left, head_right, torso_left, torso_right - arrays with triangles included in these patches.
- patch*.triangles - assigned triangles to new variables, used in optimization, based on the used chains.
- selftouch - determines whether simultaneous touch is used.
  Default value: false

- triangles - settings for optimization of the triangles
  - optimize - determines, whether to start this optimization or not.
    Default value: true
  - repetitions - number of repetitions for each optimization run.
    Default value: 1
  - optRepetitions - number of optimization runs.
    Default value: 1
  - batchSize - size of data used every repetition from the training dataset.
    Default value: 1
  - bound - determines whether to use bounds for optimization. If true, algorithm has to change to 'trust-region-reflective', because 'levenberg-marquart' does not support bounds.
    Default value: false
  - lowerBounds - lower bounds, which limit optimization by the minimal values of DH parameters.
    Default value: [-1,-1,-pi/20, -pi/20]
  - upperBounds - maximal values.
    Default value: [1,1,pi/20,pi/20]
  - selftouch - determines whether simultaneous touch is used.
    Default value: false
  - withTr - if true, x,y,z offsets are optimized.
    Default value: false
  - withDH - if true, DH parameters are optimized. Computationally more demanding.
    Default value: true

Appendix B

Extended table of results

The tables below show all results from the experiments. Value *X* means that the value was not computed because it is not relevant for the other results (e.g. not a final value for given configuration).

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Before calibration	83.34	X	73.26	X
After calibration of plastic	6.49	X	X	X
After calibration of patches	5.17	X	X	X
After calibration of triangles	4.88	7.08	3.12	3.31
All at once, prior calibration of plastic	5.18	6.48	3.11	3.17
All at once, no prior calibration, bounded	5.79	8.12	3.48	3.16
All at once, no prior calibration, not bounded	7.81	20.61	7.80	14.69

Table B.1: Right hand - torso.

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Before calibration	76.61	X	68.86	X
After calibration of plastic	7.44	X	X	X
After calibration of patches	6.91	X	X	X
After calibration of triangles	5.79	4.66	4.02	2.00
All at once, prior calibration of plastic	5.99	9.53	3.08	4.25
All at once, no prior calibration, bounded	8.22	8.19	3.59	3.92
All at once, no prior calibration, not bounded	8.98	14.21	4.00	6.57

Table B.2: Left hand - torso.

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Torso plastic	5.37	X	X	X
Torso patches	5.34	X	X	X
Torso triangles	4.91	5.62	2.3	2.46

Table B.3: Both hands - torso.

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Plastics	6.13	X	X	X
Plastics, prior plastic of right hand	6.01	X	X	X
Patches, prior plastic of right hand	4.86	6.76	2.23	3.11
Plastic + patches, prior plastic of right hand, no bounds	7.17	8.46	2.50	3.49
Plastic + patches, prior plastic of right hand, bounds	5.11	6.61	2.39	2.75
Plastics + patches, prior full calibration of right hand	4.46	6.85	3.09	2.94

Table B.4: Right hand - torso (simultaneously).

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Plastics	8.02	X	X	X
Plastics, prior plastic of right hand	6.56	X	X	X
Patches, prior plastic of right hand	6.25	5.02	2.84	2.74
Plastic + patches, prior plastic of right hand, no bounds	7.68	5.17	3.75	2.70
Plastic + patches, prior plastic of right hand, bounds	5.70	5.38	2.87	2.73
Plastics + patches, prior full calibration of right hand	5.73	4.52	3.98	2.15

Table B.5: Left hand - torso (simultaneously).

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Plastics	6.47	X	X	X
Patches	6.53	7.87	3.50	4.88

Table B.6: Both hands - torso (simultaneously).

Configuration	Training poses, *COPs* [mm]	Testing poses, *COPs* [mm]	Training poses, taxels [mm]	Testing poses, taxels [mm]
Plastic	9.11	X	X	X
Patches	8.18	X	X	X
Triangles	7.76	6.75	3.87	3.15
All at once, bounded	9.95	8.48	4.06	5.10

Table B.7: Head.

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Rustler Lukáš** Personal ID number: **465972**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Artificial Skin Calibration for the Nao Humanoid Robot Using "Self-touch"

Bachelor's thesis title in Czech:

Kalibrace robotické kůže Nao robota pomocí "sebedotykových" konfigurací

Guidelines:

1. Data collection on Nao robot with artificial skin in self-touch configurations (joint angles and skin activations).
2. Matching of corresponding contact locations on different body parts and assessment of the possibility of "multi-touches" for data collection and subsequent calibration.
3. Robot and skin model in Matlab environment.
4. Hierarchical and modular optimization framework allowing to calibrate different parameters (skin part pose, skin triangle pose, individual taxel poses, DH parameters) using non-linear least squares methods.
5. Evaluation of self-calibration w.r.t. different datasets, parameterizations, number of unknowns, prior knowledge about 2D skin structure and 3D robot structure, and sequential vs. simultaneous calibration of different robot parts.
6. Facultatively, incorporation of additional sensory modalities - vision or inertial sensing on the robot body parts.

Bibliography / sources:

[1] Albini, A., Denei, S., & Cannata, G. - Towards autonomous robotic skin spatial calibration: A framework based on vision and self-touch - In Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on (pp. 153-159). IEEE, 2017, September.
[2] Del Prete, A., Denei, S., Natale, L., Mastrogiovanni, F., Nori, F., Cannata, G., & Metta, G. - Skin spatial calibration using force/torque measurements - In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on (pp. 3694-3700). IEEE, 2011.
[3] Maiolino, P.; Maggiali, M.; Cannata, G.; Metta, G. & Natale, L. - 'A flexible and robust large scale capacitive tactile system for robots' - Sensors Journal, IEEE 13(10), 3910—3917, 2013.
[4] Mittendorfer, P. & Cheng, G. - 3D surface reconstruction for robotic body parts with artificial skins - In 'Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)', 2012.
[5] Roncone, A.; Hoffmann, M.; Pattacini, U. & Metta, G. - Automatic kinematic chain calibration using artificial skin: self-touch in the iCub humanoid robot - In 'Robotics and Automation (ICRA), 2014 IEEE International Conference on', pp. 2305-2312, 2014.

Name and workplace of bachelor's thesis supervisor:

Mgr. Karla Štěpánová, Ph.D., Robotic Perception, CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Mgr. Matěj Hoffmann, Ph.D., Vision for Robotics and Autonomous Systems, FEE

Date of bachelor's thesis assignment: **24.01.2019** Deadline for bachelor thesis submission: **24.05.2019**

Assignment valid until: **20.09.2020**

Mgr. Karla Štěpánová, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature