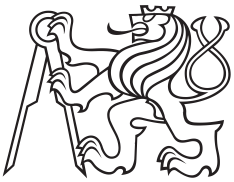
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Automating Spatial Calibration of Whole-Body Artificial Robot Skin Using 3D Reconstruction

Bc. Bohumila Potočná

Supervisor: Doc. Mgr. Matěj Hoffmann, Ph.D.
Supervisor–specialist: Ing. Lukáš Rustler
Study program: Cybernetics and Robotics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Potočná Bohumila** Personal ID number: **465812**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Automating Spatial Calibration of Whole-Body Artificial Robot Skin Using 3D Reconstruction

Master’s thesis title in Czech:

Automatizace prostorové kalibrace umělé kůže na robotovi pomocí 3D rekonstrukce

Guidelines:

With the advent of sensitive electronic skins covering large areas of robot bodies, their spatial calibration - position of individual sensors with respect to the robot kinematic model - is gaining importance. In [3,4], CAD-based calibration, self-contact, and 3D reconstruction was compared. 3D reconstruction was found to be most accurate but the procedure rather laborious, involving manual labeling of individual tactile sensors in images. In this work, we focus on making this procedure more automated by exploring different ways of automatically identifying tactile sensors in images.
Instructions:
1. Familiarize yourself with the existing skin spatial calibration for the iCub robot.
2. Collect pictures of robot skin from multiple viewpoints. Explore the use of a single camera, stereo camera, or RGB-D camera.
3. Perform 3D reconstruction of individual skin parts.
4. Develop automatic recognition of taxels in the images and assess the possibility of automatic assignment of their IDs.
5. Tie the 3D positions of taxels per body part to the robot kinematic model.
6. Motion capture system can be used to provide reference measurements.
7. Provide the taxel positions files for the skin on the iCub legs (not available at [5]). For the other skin parts, compare the performance of your solution with existing calibration [5].

Bibliography / sources:

[1] Albini, A., Denei, S., & Cannata, G. (2017, September). 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) (pp. 153-159). IEEE.
[2] Del Prete, A., Denei, S., Natale, L., Mastrogiovanni, F., Nori, F., Cannata, G., & Metta, G. (2011, September). Skin spatial calibration using force/torque measurements. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 3694-3700). IEEE.
[3] Potočná, B. (2020), 'Artificial Skin Calibration for a Humanoid Robot: Comparing or Combining “Self-Touch” and 3D Reconstruction from Images', Bachelor’s thesis, Faculty of Electrical Engineering, Czech Technical University in Prague.
[4] Rustler, L., Potocna, B., Polic, M., Stepanova, K., & Hoffmann, M. (2021, July). Spatial calibration of whole-body artificial skin on a humanoid robot: comparing self-contact, 3D reconstruction, and CAD-based calibration. In 2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids) (pp. 445-452). IEEE.
[5] https://github.com/robotology/icub-main/tree/master/app/skinGui/conf/positions
https://github.com/robotology-playground/icub-taxel-positions (earlier version)

Name and workplace of master's thesis supervisor:

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

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

Ing. Lukáš Rustler Vision for Robotics and Autonomous Systems FEE

Date of master's thesis assignment: **15.02.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Mgr. Matěj Hoffmann, Ph.D. Supervisor's signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

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

.	
Date of assignment receipt	Student's signature

Acknowledgements

Firstly I would like to thank my supervisor, Matěj Hoffmann for providing guidance and valuable feedback throughout this project. My gratitude also goes to Lukáš Rustler for helping me navigate through the relevant information and data from previous works on iCub, for correcting mistakes in my text and suggesting solutions when I was stuck. I would also like to thank Michal Polic for providing his professional insight into the 3D data processing problem.

I thank my family and friends for being a continuous support through my studies. Without them, I would not make it to the end of this journey. And finally I thank CTU in Prague for being such a good *alma mater*.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of 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 vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských prací.

V Praze dne 26. května 2023

.....
Bohumila Potočná

Abstract

This work proposes a method of calibrating artificial robot skin by 3D reconstruction from an RGB-D camera image. The calibration is performed on the iCub robot platform, equipped with a set of capacitive tactile sensors (around 4000 on the complete body surface). The goal is to automate the process of localizing the tactile sensors in the 3D scene and aligning them with the robot's kinematic structure, which was previously done manually. The tactile sensors are detected by a custom-made neural network model. We have developed a script, which takes a picture with an RGB-D camera, searches for tactile sensors in the RGB image, retrieves their coordinates in the 3D point cloud and returns them as an array. This process takes just a few minutes and requires only one input from the user: selection of the coordinate system origin. Another valuable contribution of this work is retrieving labeled skin models from configuration files and their visualization. Some errors were found in the reference files, they have been reported to be resolved. Furthermore we provided spatial calibration for some skin parts of the iCub's legs, where none was available yet. Due to the errors in the reference files and distortions in the reconstructed point cloud (possibly caused by reflections of the headlight used for lighting up the scene) taxel identification and alignment with the kinematic model is left unresolved, to be tried later with corrected reference and better light.

Keywords: artificial skin, calibration, 3D surface reconstruction, iCub, humanoid robots, deep learning, object detection

Supervisor: Doc. Mgr. Matěj Hoffmann, Ph.D.

Abstrakt

Tato práce přichází s metodou kalibrace umělé kůže robota pomocí 3D rekonstrukce ze snímků z RGB-D kamery. Kalibrace se provádí na robotovi iCub, který je vybaven kůží z kapacitních dotekových senzorů (celkem přibližně 4000). Cílem práce je zautomatizovat lokalizaci dotekových senzorů ve 3D prostoru a jejich vztažení ke kinematickému modelu robota, které se dříve prováděly ručně. Dotekové senzory jsou ve snímku detekovány na míru vytvořeným modelem neuronové sítě. Byl vyvinut program, který udělá snímek RGB-D kamerou, vyhledá v něm dotekové senzory, získá jejich souřadnice ve 3D a vrací je jako pole. Tento proces trvá jen několik minut a vyžaduje od uživatele pouze jeden vstup: výběr počátku souřadného systému. Dalším přínosem této práce je získání modelů kůže z konfiguračních souborů a jejich vizualizace. V referenčních souborech bylo nalezeno několik chyb, jež byly nahlášeny k opravení. Dále byla získána prostorová kalibrace některých částí kůže na nohou iCuba, pro které dosud žádná neexistovala. Kvůli chybám v referenčních souborech a zkreslení 3D modelu získaného rekonstrukcí (pravděpodobně způsobeného odrazy světla použitého k osvětlení) identifikace dotekových senzorů a napojení na kinematický model robota nebyly zatím provedeny. Tento proces bude realizován později s lepším osvětlením, až budou referenční modely opraveny.

Klíčová slova: umělá kůže, kalibrace, 3D rekonstrukce povrchu, iCub, humanoidní roboti, hluboké učení, detekce objektů

Překlad názvu: Automatizace prostorové kalibrace umělé kůže na robotovi pomocí 3D rekonstrukce

Contents

Chapter 1

Introduction

1.1 Motivation

Many collaborative robots are equipped with artificial skin in order to perceive the environment by the sense of touch, which allows them to interact properly with their surroundings. The skin usually consists of a set of tactile sensors mounted on the robot's body—further referred to as taxels. Nevertheless, in most cases the exact positions of individual taxels are initially unknown and need to be calibrated. This also applies to the iCub robot which we will use in this work.

Currently the calibration data for iCub's skin is incomplete—either not very accurate, or totally missing for some parts. This work will improve the existing calibrations and provide calibration for parts, where it is not yet available.

3D reconstruction has shown promising results in previous work on the same skin mounted on a Nao robot [1,2]. The taxel position data extracted from a 3D point cloud reconstructed from photos achieved better accuracy than the alternative approach based on self-touch configurations. However in the methods used a lot had to be done by hand, such as marking the taxels in the point cloud and assigning their IDs, which is very time consuming and therefore rather inconvenient to be used repeatedly or applied on various platforms.

Apart from contributing to calibration of iCub's skin, this work also aims to come with a method to automatically recognize artificial skin elements in 3D image data and assign their IDs based on a provided 2D layout. This will allow to use the 3D reconstruction calibration technique on other platforms as well without much manual effort.

Due to continuously increasing availability of cameras with depth sensors, decrease of their costs, and improvement of their accuracy, we have new options to approach the skin surface reconstruction. A single image from a depth camera (or up to 5, if the surface is more curved) could be enough to obtain a reasonably precise 3D model of the robot skin.

1.2 Goals

Goals of this thesis:

- Perform 3D reconstruction of individual iCub skin parts
- Develop automatic recognition of taxels in the images and assess the possibility of automatic assignment of their IDs
- Tie the 3D positions of taxels per body part to the robot kinematic model
- Provide the taxel position files for the skin on the iCub legs (none exist yet). For the other skin parts, compare the result with the existing calibration

Chapter 2

Related Work

2.1 Previous work on iCub skin calibration

A calibration of some parts of iCub's skin was performed by Del Prete *et al.* [3] in 2011 using force/torque sensors located in the robot's arms. Using a thin metal tool, they activated each taxel under several angles and computed the taxel's position as approximated intersection of the force axes (provided by the force/torque measurements). There were two estimations made: with and without the knowledge of the robot surface. Their mean errors were 7.2 mm and 6.6 mm respectively. The method was tested on iCub's right forearm and later also applied to the left arm and forearm.

Another iCub skin calibration has been provided by Traversaro *et al.* [4], where the skin triangle centers were extracted from CAD models of the robot and other taxels were computed based on their 2D layout in the triangle, neglecting the curvature of the surface. This work provided calibrations of iCub's arms, forearms, hands and torso.

This skin calibration was used by Roncone *et al.* [5] in 2014 to calibrate iCub's kinematic chain (Denavit-Hartenberg parameters of the joints) in the upper body by a self-touch method. In that work only the joint parameters were calibrated (while the skin positions were assumed to be correct), but this approach can just as well be used for calibrating the skin.

In 2017 Kangro *et al.* [6] came up with a time-efficient technique to calibrate the skin surface normals using vacuum bags. When a skin part is wrapped in a plastic bag and the air pressure inside is lowered by a pump, the bag applies a known uniformly distributed force on the whole skin surface, which provides the information for calibrating each taxel simultaneously. Based on this approach, they later developed a calibration device [7].

2.2 Using 3D reconstruction for robot skin calibration

In 2017 Albini *et al.* [8] came with a calibration method that combined 3D reconstruction and self-touch. The robot used an image from RGB-D camera mounted on its head and knowledge of its kinematic structure to reconstruct the shape of its body part (arm) in the field of vision. This provided the possible set of points in 3D space that could be touched (assuming the whole body surface is covered in skin and all the points are reachable). From this set, it chose one random point and navigated the end effector on its other arm to apply pressure to it along its normal. The contact centroid (end effector position) was then

associated with the activated taxel. This process was repeated until enough measurements were collected for each taxel. The test of this method on the Baxter robot resulted in an average position error less than 2 mm for a skin patch placed on a table and 2.9 mm for a skin patch mounted on the robot body.

In 2020 a 3D reconstruction-based calibration was performed on the same model of skin mounted on Nao robot [1]. First, a 3D mesh model of the robot with exposed skin was reconstructed from a set of photos from different angles. Then the taxels' centers were manually selected in the model and their 3D coordinates saved as a point cloud with each taxel marked with its ID. It was not possible to associate the taxels' positions to the reference point in Nao's body part by this technique only. Nevertheless, it provided very accurate 3D layout of the taxels—mean position error was below 2 mm. In the work by Rustler *et al.* [2], this 3D reconstruction approach was compared and combined with calibration by self-touch configurations and calibration obtained from CAD models of Nao's body parts with the knowledge of 2D skin layout. Positions from CAD models were a useful input for the self-touch calibration, since it already needed some position data to start with. Thanks to the data from 3D reconstruction-based calibration, the mutual positions of the taxels could be fixed, therefore there was only one position left to calibrate (the position of one of the taxels w.r.t. the reference point in each robot body part) instead of hundreds.

2.3 3D segmentation and object detection

Getting and processing depth information has been an important part of robotics for a long time. Many robots use Time of Flight or LiDAR sensors to navigate in their surroundings. Though in the last couple of years more variety of technologies for 3D data measurement have become available (e.g. smaller and more affordable stereo cameras) and at the same time new demands for their use have arisen, such as autonomous vehicles, collaborative robots or augmented reality. Because of this, Computer Vision is expanding its field of action from 2D data to 3D as well. This also applies to Deep Learning (DL).

3D object recognition will help robots to better comprehend their surroundings and interact with them. In 2021 Birri *et al.* have introduced a software for a healthcare robot to recognize and localize small objects in a hospital environment, using an RGB-D Kinect camera [9]. The input from camera – RGB image + depth map – is converted to a point cloud, which goes through several processes, including RANSAC segmentation, Euclidean cluster, feature extraction using a Viewpoint Feature Histogram descriptor and finally Neural Network classification, returning a visualization with bounding boxes around the detected objects and their labels. The experiment was done on 9 classes representing some common objects the robot is meant to interact with, such as plate, bowl or glass. The system achieved 90.77% accuracy on multi-object scenes and 98.73% on a single object. Another good 3D DL example is the work by Yeom *et al.* coming with a system to perform an indoor space segmentation with HybridNet [10]. The purpose was to help indoor moving gadgets perceive the structure of their environment and better navigate the space. Their network consisted of a 2D network and a 3D network in parallel, whose features are combined in the end by a fusion network. The input is a 3D point cloud, which goes through voxelization for 3D feature extraction and spherical projection for 2D feature extraction. The 2D features

are then reprojected to 3D space and together with the 3D network output they go through a 3D convolution to make a segmentation prediction. The considered classes are several basic parts of indoor spaces like floor, ceiling, window, table, etc. The results have shown better accuracy than using each of the networks (2D and 3D) separately.

O'Mahony *et al.* provided a review of state-of-the-art 3D deep learning techniques in 2018 [11], focusing on convolutional neural networks (CNNs), which have been proven to be one of the most effective DL architectures. They analyze the commonly used approaches and introduce their challenges. For example, one of the biggest challenges in 3D semantic segmentation is the accurate annotation of 3D points belonging to a class when creating a dataset. The more complex the shape, the harder it is to determine the exact boundary between the points belonging to the object and the background. Another significant issue is memory requirements, which are much higher for 3D datasets than for 2D. Also, both of the above mentioned works (Birri *et al.*, Yeom *et al.*) show that object recognition in 3D data is not at all a straightforward process. Some preprocessing like voxelization, projection or VFH feature extraction is required before passing the input to the network.

Chapter 3

Materials and Methods

3.1 iCub robot

The robot platform used in this work is iCub 2.7. iCub is an open-source humanoid robot platform developed by a collective from several European universities and the Italian Institute of Technology for the purpose of research in cognitive robotics [12]. The robot is about 1 m tall and it is unique by its sensor equipment and flexibility (53 degrees of freedom), making it a suitable platform to study a huge variety of functionalities, including walking, vision, touch, object manipulation and human interaction. Figure 3.1 shows a photo of one of the iCub models.

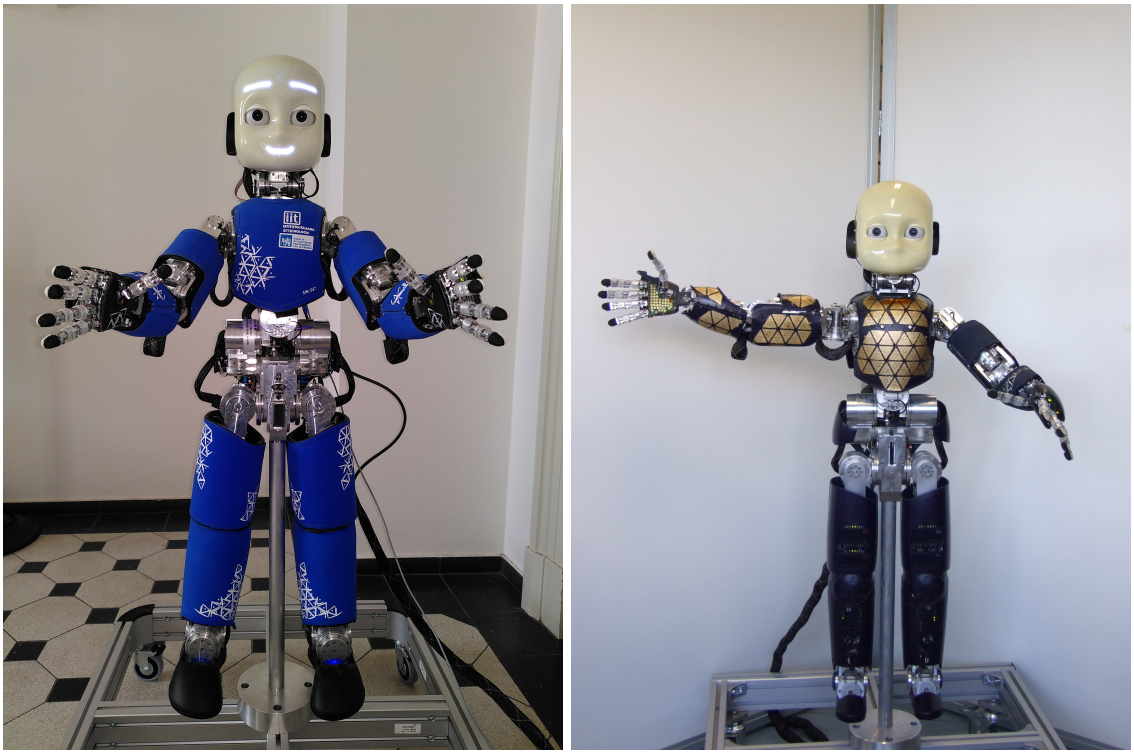


Figure 3.1: iCub robot: covered (left) and with exposed skin (right).

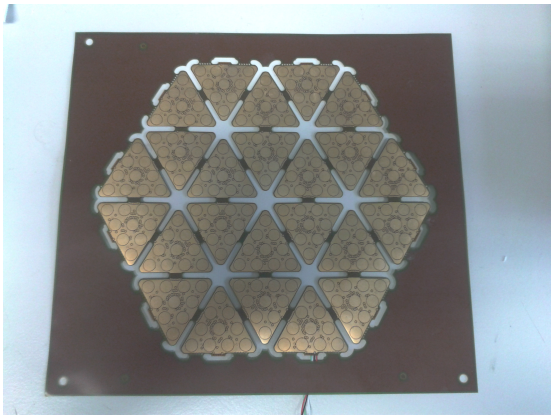


Figure 3.2: A single skin patch before being attached to the robot.

3.2 Artificial skin

The robot is equipped with an artificial skin made of capacitive tactile sensors developed by Maiolino *et al.* [13], capable of measuring contact position and intensity. In total, iCub 2.7 is equipped with over 4000 taxels. Besides from a few exceptions (fingers and palms) the taxels are organized into patches composed of triangular modules containing 10 taxels each (+ 2 thermal sensors). A single patch is shown in Figure 3.2. The skin patches are attached to custom-made plastic holders mounted on iCub's body. On the top, there is a removable protective fabric—this was one of the improvements to the original version of the skin. Making the protective layer removable, instead of glued to the sensors, to allow replacement when needed and temporary removal for checking the sensors below [13]. This feature is also essential for our calibration method, since the individual sensors have to be visible on camera during the reconstruction.

In this work, we focus on calibrating the skin on the torso and legs. For the torso there are already previous calibrations available, which can be used as reference for testing our method, whereas the leg skin was added later than the other parts, and thus no calibration exists yet.

3.3 RGBD camera

For capturing the skin, we used the Intel RealSense depth camera D435 [14]. Its maximum resolution is 1920x1080p for RGB and 1280x720 for depth. The ideal operating range is 0.3 to 3 meters and depth accuracy $\pm 2\%$ at 2m distance. The depth error decreases with distance. Based on a depth error testing performed in an instance segmentation project described in [15] the error should be below 1 cm for distances up to 0.5 m.

Our iCub model has one RealSense camera mounted on its forehead, which could be used for calibrating the forearms, but its position does not permit to capture the torso and legs. Therefore, we used another camera externally. When capturing a skin part, the camera was always situated opposite its surface at about 0.5m away. In some of the experiments, we additionally used a ring-shaped light stand to stabilize the light conditions of the scene.

The installation of the camera with the light stand is shown in Figure 3.3.

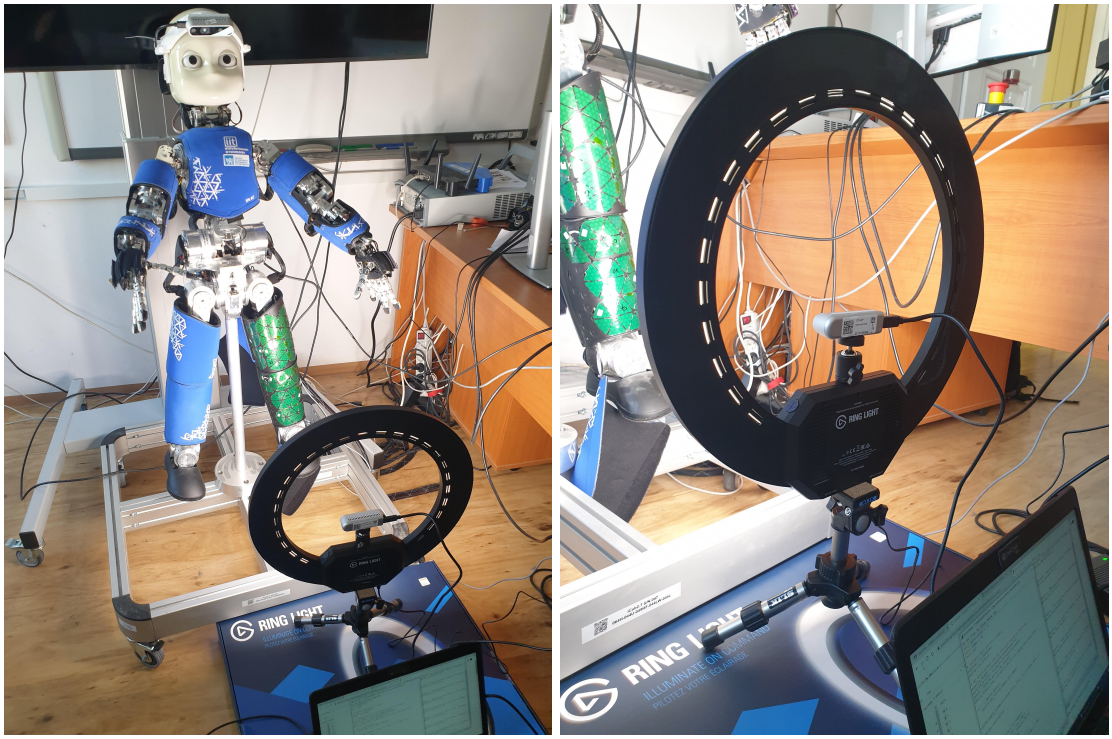


Figure 3.3: Installation to capture the skin on iCub's left leg using a light stand (left) and a detail of the light stand (right).

3.4 Software

The taxel detection and 3D reconstruction processes are implemented in Python. Intel provides a library for operating the camera from Python called `pyrealsense2`, which contains the needed functions and tools for setting the streaming parameters, capturing images or videos and various post-processing options. They also share several examples for using some standard functions (RGB and depth frame streaming, point cloud extraction, background removal, etc.) in Intel RealSense Documentation [16].

Another important library for the project was `opencv`—a computer vision library, which provides a wide range of tools for image and video processing, from basic picture editing (applying filters, stitching, etc.) to object recognition, camera calibration or movement tracking. Documentation is available at [17].

Finally, the taxel detector was implemented in PyTorch, a tensor library for deep learning. It provides a wide range of tools to build a custom neural network of any sort (image/audio/text processing and more), including an option to easily make one's own dataset. Documentation is available at [18].

3.5 Taxel detection

In an earlier work on Nao skin calibration [1], the positions of taxels in 3D point cloud were selected and identified manually. In order to automate this process, the taxels have to be recognized and localized by the program. This requires image classification or segmentation. The output we need is a 3D point cloud, where each point represents a taxel's center (+ its normal, if possible). The input we have is an image from RGBD camera: 2D RGB image + depth map.

Functions needed for point cloud creation are provided in `pyrealsense2` library. In the documentation, Intel also shares a code example, which lets the user construct and display a point cloud from the camera stream in real time, similarly as in RealSense Viewer app. The created point cloud can also be saved (for example as .ply file). By default, this creates a point cloud from the whole scene the camera sees. What we need to do is to filter out redundant parts of the scene and only save the points which are taxels. Theoretically there are two possible ways of performing the selection:

1. Run the classifier on 2D image (with optional usage of depth map as additional information) and create the 3D point cloud from the filtered set of 2D points.
2. Run the classifier on the created 3D point cloud.

As we see in the works mentioned in Chapter 2, running a classification on 3D data requires a lot of preprocessing and making a dataset is very complicated. Another potential problem is that taxels would come just as a piece of flat surface in space and not a 3D object. Therefore we chose the more straight-forward first option—find the taxels in 2D RGB picture and pick their corresponding 3D points from the point cloud. The following subsections (3.5.1-3.5.7) describe the strategies and tools used for building and training the classifier.

3.5.1 Neural network

Neural network (NN) (also referred to as artificial neural network — ANN) is a machine learning algorithm inspired by the biological learning, i.e., the learning process as it is happening in human and animal brains. [19] They normally consist of a set of nodes which are interconnected (like neurons in a biological brain), and a set of weights which are being learned. There are multiple NN models for solving different kinds of problems. The type of NN we use is based on linear or logistic regression. It contains a set of weights w, by which the input X is multiplied to get the predicted output $\hat{y}$. During the training of NN weights w are numerically optimized, usually by a gradient descent method. Figure 3.4 shows a scheme of a linear neuron—the basic building unit of neural networks. Forward propagation of input x through the neuron gives us the output based on the following equation:

$$\hat{y} = \mathbf{x}\mathbf{w}^T \tag{3.1}$$

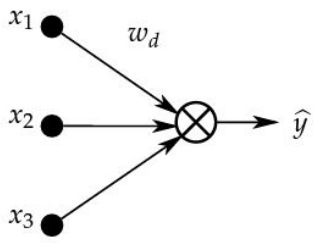


Figure 3.4: Scheme of a linear neuron. [20]

For logistic regression, which is more commonly used, we use a non-linear activation function on the output (e.g. sigmoid or ReLU; more options mentioned later), according to the following equation:

$$\hat{y} = g(\mathbf{x}\mathbf{w}^T) \tag{3.2}$$

For training a neural network we need a dataset $T = (\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ are the input vectors and y their true labels. We also need to determine a loss function $J(\mathbf{w})$, which will measure the difference between the predictions $\hat{y}$ and true labels y and which we want to optimize. There are several options, e.g. squared error or cross-entropy loss. Then in each step of the training the weights w are changed so that $J(w)$ decreases, *i.e.*, they are updated in the opposite direction to the gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(w) \tag{3.3}$$

where η is the learning rate (step size) [20].

These neurons are stacked into layers and more complex neural networks consist of more of these layers. All the layers between the input vector and the final layer are called hidden layers. Figure 3.5 shows an example of a fully connected neural network (also referred to as MLP—multilayer perceptron) with one hidden layer. The layers of neural networks can be further modified in various ways to solve a certain type of problem. One of these modifications are convolutional layers, which we will use in this work for the taxel recognition.

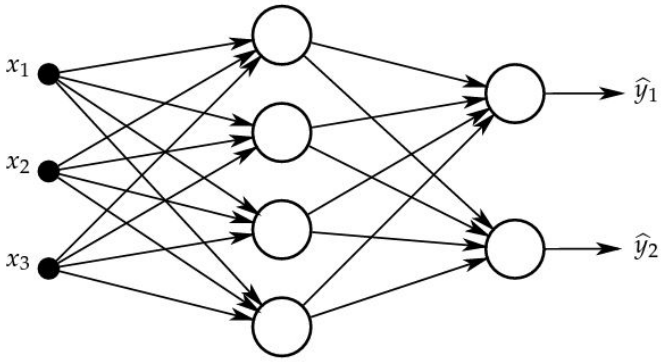


Figure 3.5: An example of multilayer perceptron with one hidden layer. (Each empty circle represents a neuron.) [20]

3.5.2 Convolutional neural network

A convolutional neural network (CNN) is a special case of a neural network, which besides from fully connected layers contains also convolutional layers, which have a smaller number of parameters and are used to detect some specific features. CNNs are typically used in image processing. They have been proven especially successful in object recognition.

A convolutional layer scans through the image with a convolutional window (kernel) to search for patterns, which make its features. The required arguments for a convolutional layer are:

- number of input features—for the first layer it is the number of color channels, i.e., 1 for grayscale, 3 for RGB image, for others it is the number of output features of the previous layer
- number of output features—number of features the layer learns
- kernel size—usually a square kernel with odd size

The output of a convolutional layer is called an activation map. Its dimensions are $F \times M \times M$, where F is the number of output features, and $M = N - K + 1$, where N is the size of input image and K is the kernel size. So, for example, if we apply a convolutional layer with 6 output features and kernel size 5 on a grayscale image with dimensions 32×32, we will get a feature map with dimensions $6 \times 28 \times 28$.

A very common use of CNN classifiers is for detecting objects like chair, person, car, dog, etc. Many datasets and networks with pretrained weights are available for this problem by now. Some of the most frequently used datasets are ImageNet [21], CIFAR [22] and MNIST. As for network architectures, here are a few honorable mentions:

- LeNet-5 (1998) [23]
- AlexNet (2012) [24]
- VGG (2014) [25]
- Inception (2015) [26]
- ResNet (2016) [27]

For a better idea what a complex CNN architecture looks like see the scheme of VGG-16 in Figure 3.6. The input is a 224×224 RGB picture. The network then consists of 13 convolution layers and 3 fully connected layers with ReLU activation interlaced with max pooling layers. The output has 1000 channels, because the network differentiates between 1000 classes. Thanks to softmax function at the end, it is returned in the form of probability percentage for each class.

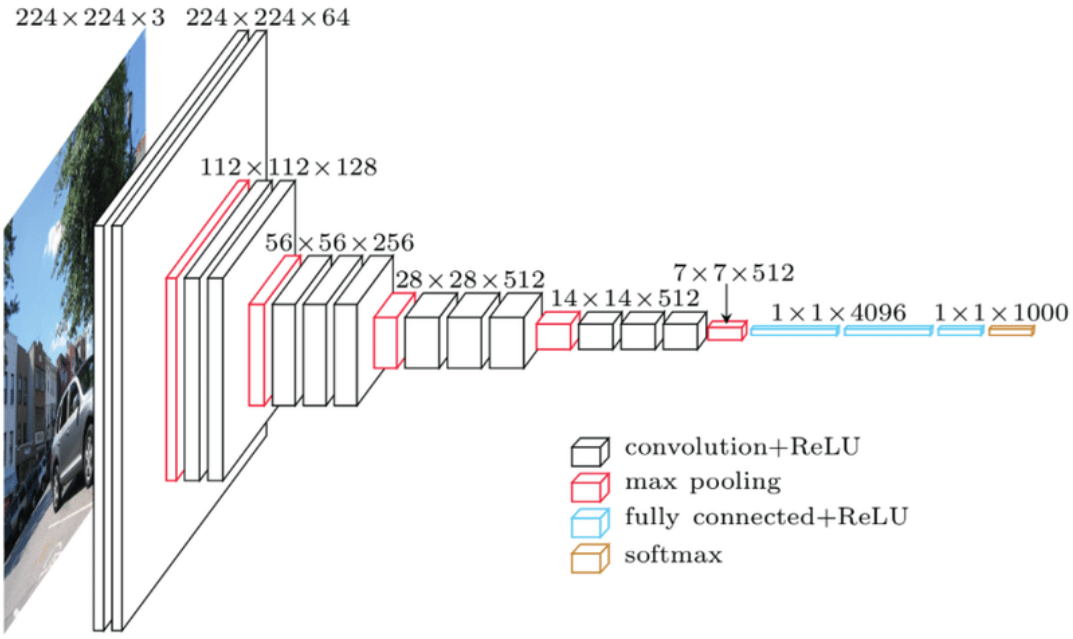


Figure 3.6: VGG-16 architecture scheme [28].

In this work we are differentiating between only two classes: the tactile sensors and the background, i.e., we are making a binary classifier. The appropriate architecture needs to be found. Some relevant qualities which should be taken into account are dataset size, image size and the complexity of visual properties (features) of the classes.

Here is a list of CNN layers we can use and parameters we can tune and how to choose their values:

- Kernel size — size of the weight matrices to detect features, normally odd size. Smaller filters — 3x3, 5x5 — detect small local features, whereas bigger filters — 9x9, 11x11 — collect more complex features from larger areas.
- Number of channels — on the input it is equal to the number of color channels (3 for RGB, 1 for grayscale), later it determines the number of used filters, i.e., number of features learned. Too high value causes over-fitting.
- Padding — a layer which adds extra columns or rows of zeros to maintain the sizes after a convolution. It may improve performance.
- Stride — number of pixels to skip while traversing the input. We can choose not to convolve the kernel with every square in the input, but only with every 2nd, 3rd, etc., which makes the output image of the layer considerably smaller. It increases speed of the network, but some information is lost.
- Pooling — serves for down-sampling the picture. The most common type is Max pooling, which takes the maximum value from every square of pixels. Same as with stride it increases the speed for the cost of loosing some information.

- Number of layers — too many layers may cause overfitting and vanishing/exploding gradient. Too few on the other hand can cause high bias and low accuracy. Depends a lot on the dataset size.
- Hidden units per layer — number of weights to tune in each layer. Similar influence as number of layers.
- Activation function — ReLU, LeakyReLU, Sigmoid or Tanh. Sigmoid and Tanh perform well for shallow networks, otherwise ReLU and LeakyReLU are the reliable options.
- Optimizer — the algorithm to update weights. Adam is the most frequent choice. Also SGD (Stochastic Gradient Descent) is used a lot.
- Learning rate — typical values are powers of 10, normally 0.001 to 1. Since there are not so many options, it can be chosen experimentally (try each option and see which performs best). Too high value tends to prevent convergence and too little makes the learning process too slow.
- Batch size — number of images used as input for one learning iteration. Typically powers of 2, the most common are 16, 32 and 64. Too low value causes chaotic weights update resulting in a poor model and too high slows down the learning process.
- Number of epochs — too high may cause over-fitting or vanishing/exploding gradient, too low limits the network's potential. Same as learning rate, it can be chosen experimentally.

When building a network, it is advised to start by choosing the layers. Shashank Ramesh in his guide to building and tuning CNNs [29] suggests to add convolutional layers gradually until we reach overfitting, starting with small kernel size (3x3) and smaller number of output channels and gradually increase both when adding more layers, or keep the kernel size the same, but do not decrease it. Some existing models may serve as good inspiration. Once the network is built (i.e., we have determined all the layers), comes the hyper-parameter tuning. Both of these processes are done experimentally by training various NN models with different hyper-parameters and observing their learning process and testing performance. For evaluating the model's learning ability, we use the Learning curve and Confusion matrix (described below).

3.5.3 Learning curve

For monitoring the behavior of the model during training, we use the learning curve. Learning curve is a plot depicting the learning performance of a machine learning model over time: epoch on x-axis and loss or accuracy on y-axis. The plot consists of two curves: training loss and validation loss. The shape of the learning curve tells us a lot about the learning ability of the model and about the quality of the dataset. For example, we can tell whether the model underfits (too few epochs) or overfits (too many epochs), whether it is too shallow for the task or if the dataset is imbalanced [30]. Some examples of a learning curve are shown in Figure 3.7.

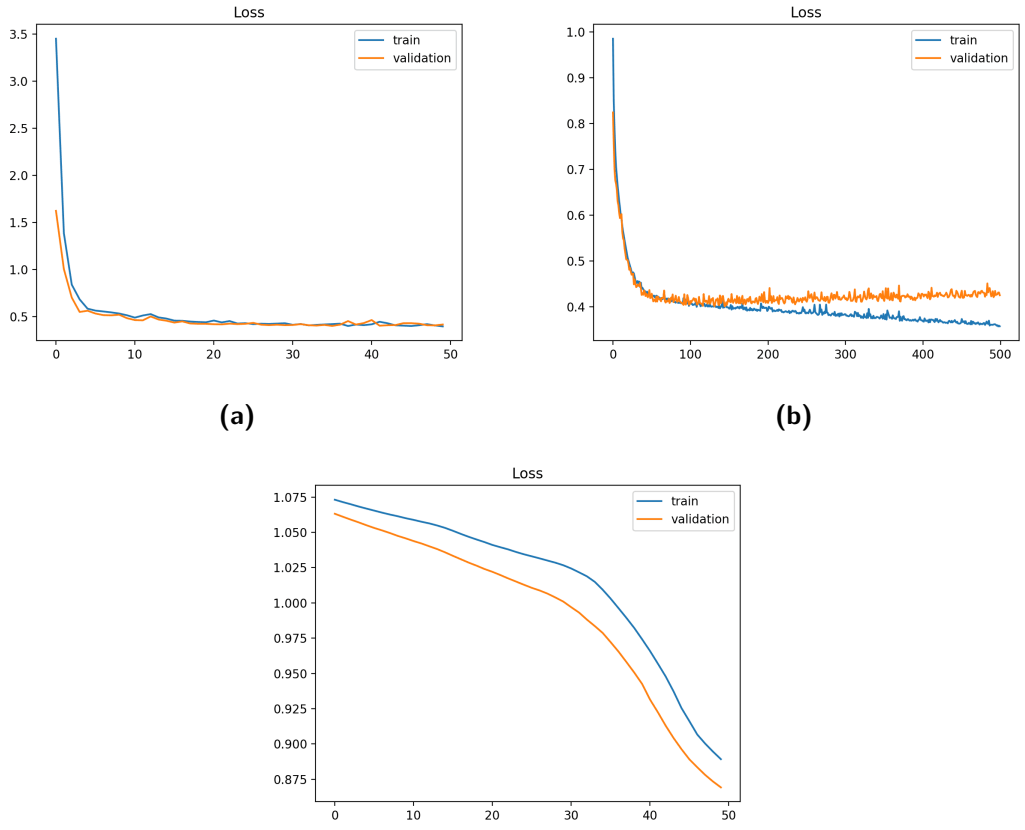


Figure 3.7: Examples of a learning curve: a) well fit — both training and validation error settled down b) overfit — the model accommodates too much to the training data and does not generalize well, validation error starts to increase c) underfit — both errors are still decreasing, the model has more learning potential. [30]

3.5.4 Confusion matrix

A good metric for a binary classifier's accuracy is the confusion matrix, which depicts the number of correctly classified classes and incorrectly classified classes. Its equation looks as following:

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{3.4}$$

where TP means true positive, FP false positive, FN false negative and TN true positive.

3.5.5 Sliding window detector

A sliding window detector does not perform a segmentation on the whole image at once, but uses a window to iterate through the image and search for instances of the given class(es). The sliding window is usually combined with a so-called pyramid, which repeats the iterations with different window sizes. But since the camera's distance from the skin

Figure 3.8: Dataset photos examples: torso (top), legs (bottom).

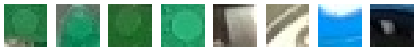


Figure 3.9: Cutout dataset examples: taxels (first four) and non-taxels (second four).

will not differ much (30-50cm) and taxels have a perfectly regular shape of a circle (or ellipse when viewed from side), the pyramid is not needed, and we can choose a constant window size.

3.5.6 Making the dataset

For training and validation of the classifier, we took a set of photos of iCub's torso and legs with exposed skin. The following sets were taken:

1. 81 photos of torso from about 6 different angles with varying light conditions (light on/off, windows shaded/unshaded, adding flashlight), size 640x480
2. 20 photos of iCub's legs, each leg from about 4 different angles, using a ring-shaped headlight for illumination, size 1280x720

For both sets, the distance between the camera and the skin was 30-50 cm, in order to achieve the highest precision possible. Examples of the photos from both datasets are shown in Figure 3.8.

For the sliding window detector, we needed to obtain square cutouts with labels: 1 = taxel, 0 = non-taxel. This procedure was done manually with the help of a custom-made Python program, which opens and shows each of the given photos and each time the user clicks on a point in the picture, it saves a square cutout of a certain size around that point with a label determined in advance (each n-th click is 1, otherwise 0). A few examples from the cutout dataset are shown in Figure 3.9.

This cutout dataset was made using the bootstrapping technique: starting with some primary cutout set to train and validate the CNN and gradually adding false positives to the dataset until (almost) only taxels are detected. The effect is shown in Figure 3.10. More details about each of the cutout datasets and the networks' performance after adding them are described in Chapter 4.

Figure 3.10: Visualization of a taxel recognition performance before (left) and after (right) bootstrapping.

3.5.7 Cascade classifier

In order to speed up the detection, we run the input through some weaker filters first before applying the complex CNN, forming a so-called cascade classifier. First filter we use is a depth filter. Based on the depth map provided by the RGBD camera we remove every pixel that is further than 0.5m, which in good conditions leaves only the robot body in the scene, reducing the input for CNN by more than half. Another very simple filter is a color filter, which compares the color of each individual pixel to the taxel color. Since our iCub's skin has a very specific color—a distinct shade of green, which very rarely occurs in the background or non-skin body areas, a filter like this for preprocessing is definitely worth a try.

After applying these two filters, the detection mask is significantly reduced and the image can be passed to the CNN, which goes only through the pixels, which are marked in the mask.

3.6 Taxel identification

After getting a 3D point cloud of detected taxels the next step is to assign them their corresponding ID. Each skin part is assembled from a set of patches, whose layout follows a given order. The complete sets of triangles with their IDs, 2D positions and orientations are available in the form of configuration (.ini) files at Robotology GitHub repository [31], as well as MS Excel sheets they were generated from. Order of taxels in a single triangle is

assigned as shown in Figure 3.11. There are 12 sensors in each triangle, though 2 of them are thermal sensors (number 6 and 10), which we do not work with. Then each taxel's unique ID can be calculated from its triangle's ID by the following formula:

$$taxel\_ID = triangle\_ID \cdot 12 \cdot i$$

where i is the taxel's index in the triangle.

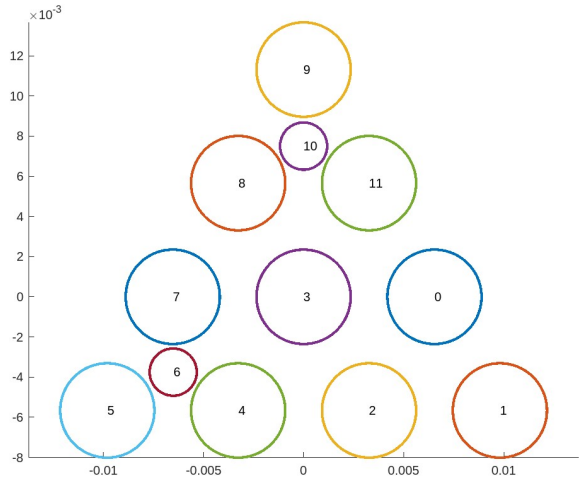


Figure 3.11: Taxels' numbers within single triangle.

The configuration files with 2D layouts are available for all iCub's skin parts. Additionally, for some skin parts there are also CAD models of their mounts available, which provide the information about triangle mounts placement, i.e., precise 3D location of triangle centers. These have been used in the previously mentioned work by Traversaro *et al.* [4], providing a 3D torso and arm skin model with precise locations of center taxels (taxel number 3) and other taxels approximated as flat triangles.

Torso skin consists of 44 triangles (440 tactile sensors), whereas leg skin is divided into upper and lower part consisting of 68 and 38 triangles, respectively.

3.6.1 Iterative closest points

Iterative closest points (ICP) is an algorithm for so-called point cloud registration problem — given two (or more) sets of point clouds in different coordinate systems, the task is to align them together. In other words, we are searching for a transformation, which minimizes the difference between them. The ICP algorithm is based on alternating between two steps: Correspondence step and Alignment step. Correspondence step finds the closest point in the second set to each point in the first set, based on the current transformation. Alignment step updates the transformation by minimizing the distance between the corresponding points. For more details, see Section 3 of the work by Zhang *et al.* [32]. This algorithm could help with the taxel identification by aligning a labeled skin model with a point cloud retrieved from the detector.

Chapter 4

Experiments and Results

All the scripts used in this work are available in *iCub skin calibration* repository [33].

4.1 Taxel detection

4.1.1 Datasets

All the dataset pictures are uploaded in folder `icub-skin-calibration/dataset` [34]. First photo dataset consists of 81 photos of torso skin with changing light conditions, with dimensions 640×480. Second photo dataset contains 20 photos of leg skin lit by a ring-shaped headlight, dimensions 1280×720.

From these two photo sets, 5 sets of cutouts with labels were made, using a custom-made Python script in `taxel_dataset_collector.py`. Information about each of them is shown in Table 4.1. Datasets 1 and 4 were primary datasets for the torso photo set and leg photo set, respectively, and were focused on differentiating the skin from the background. Datasets 2, 3 and 5 were made as bootstrapping additions to their primary dataset and their purpose was to differentiate the taxels from other parts of the skin. The Python class for representing the datasets for a pytorch neural network is defined in `torch_nn_dataset.py`.

folder	labels file	number of cutouts	cut from
`taxel_dataset1`	`taxel_labels1.csv`	972	torso skin photos
`taxel_dataset2`	`taxel_labels2.csv`	972	torso skin photos
`taxel_dataset3`	`taxel_labels3.csv`	1200	torso skin photos
`taxel_dataset4`	`taxel_labels4.csv`	800	leg skin photos
`taxel_dataset5`	`taxel_labels5.csv`	800	leg skin photos

Table 4.1: Cutout datasets info.

4.1.2 Color filter

The color filter is used during the preprocessing of a camera snapshot before it is passed to the CNN. It serves to quickly filter out the pixels which clearly does not correspond to the skin color. An important criterion here is that the color interval has to be set so that no part of the skin is removed from the selection, i.e., no false negatives.

The color interval was chosen based on statistical results from picking points in dataset photos, using a custom code that opened the selected photo and saved the RGB values of

each clicked pixel to a .csv file. We collected a set of skin pixels and a set of non-skin pixels, 1000 samples each. The histograms of red, green and blue values are shown in Figures 4.1 and 4.2. We see that RGB values of the skin are approximately in the following intervals:

- R: <0,80>
- G: <52, 180>
- B: <14, 130>

RGB values of the background have approximately a uniform distribution. Therefore, we can expect some false positives, but a part of the background will be filtered. The filter is set to keep only the pixels which have all the RGB values within the intervals as stated above. The result of using this filter only on a picture from the dataset is shown in Figure 4.3. There are indeed some false positives, but all the pixels with skin are kept and the process takes only a few seconds, unlike running a sliding window with CNN on the whole picture, which takes several minutes.

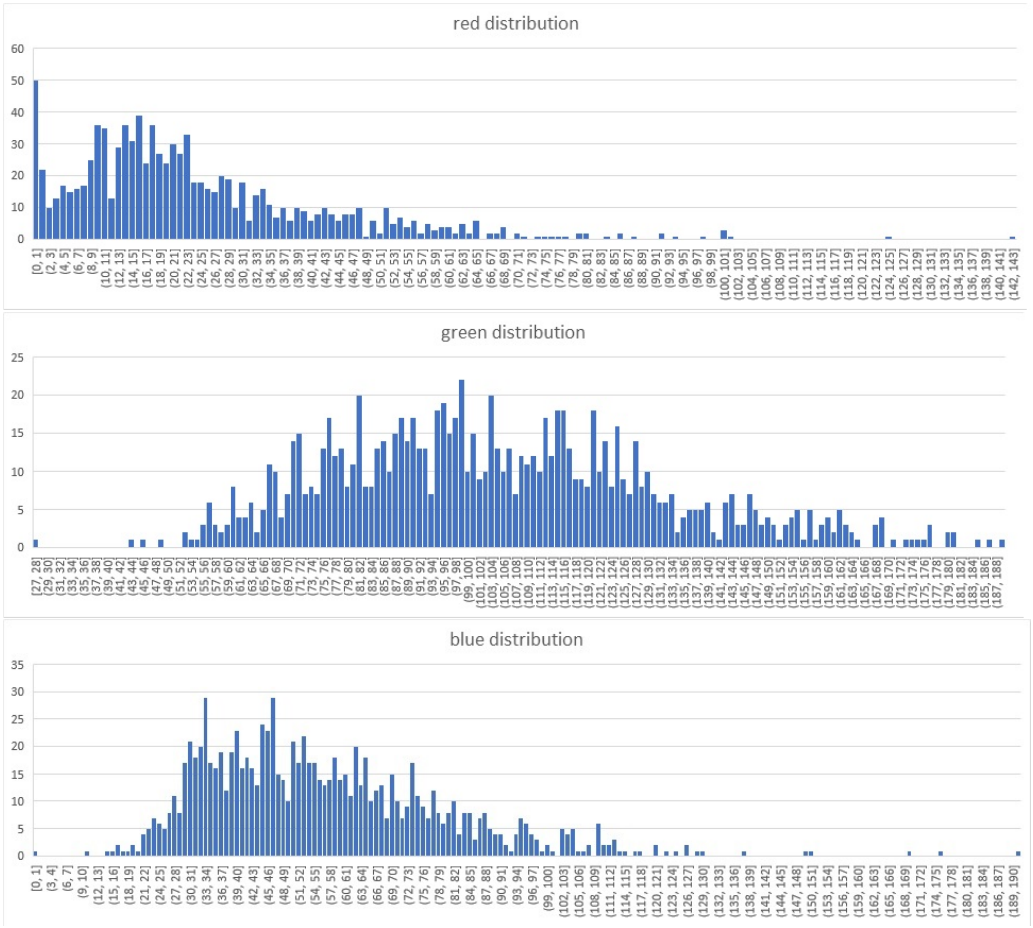


Figure 4.1: Histograms of RGB values for skin pixels.

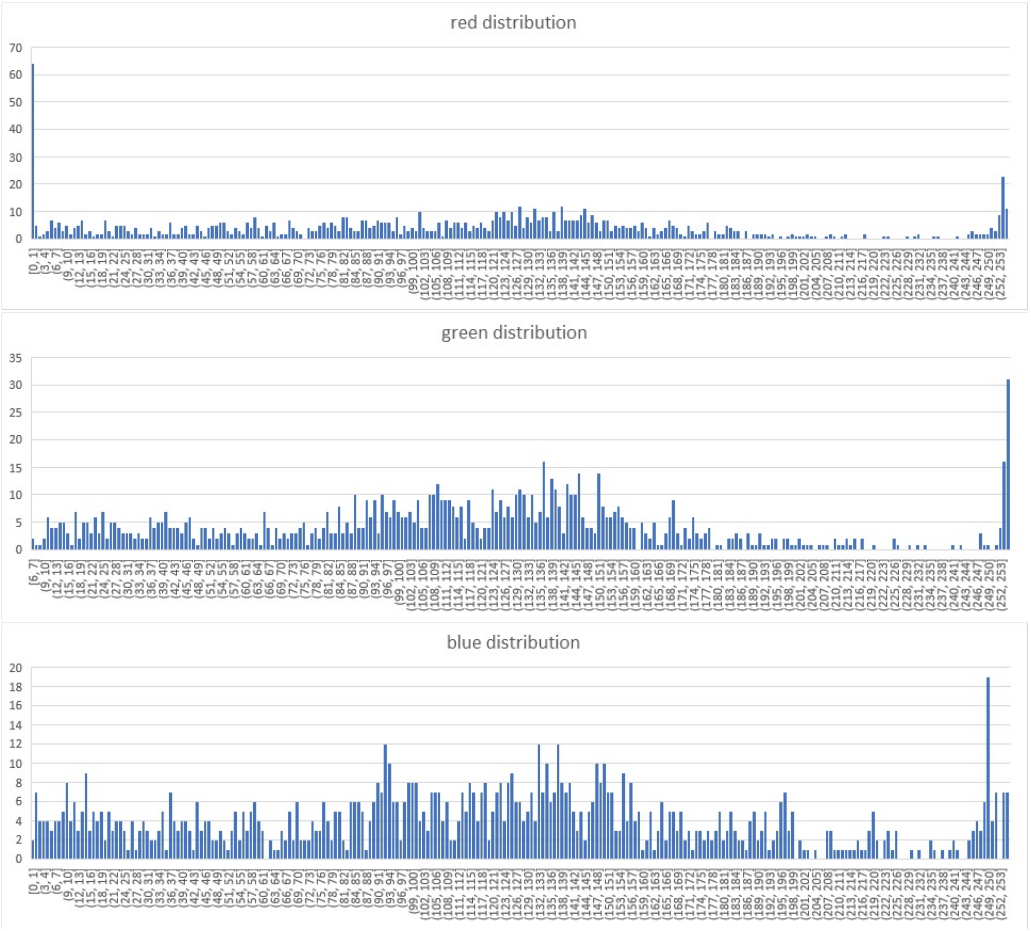


Figure 4.2: Histograms of RGB values for non-skin pixels.

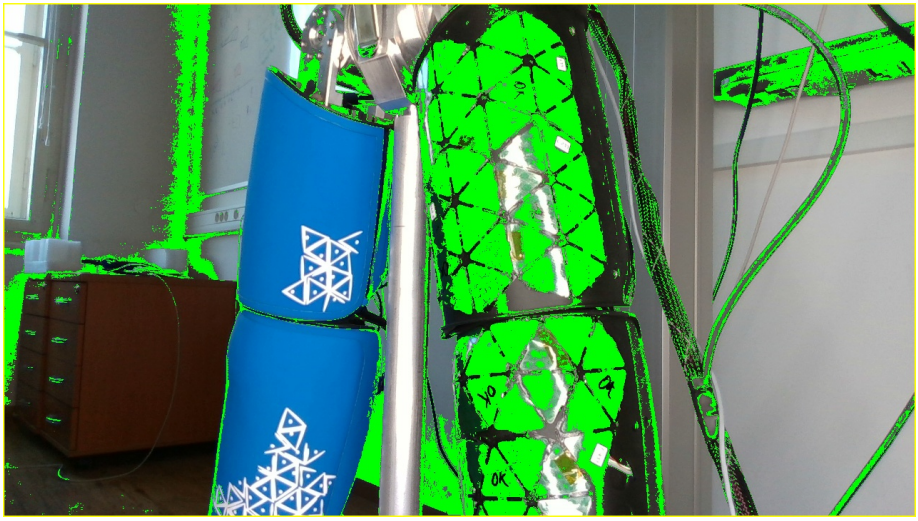


Figure 4.3: Result of using only pixel color filter on a photo from a dataset.

4.1.3 Neural network classifiers

This section presents the used classifier architectures and their performance. The first task was to find a suitable network architecture. We started with a simple model with 3 fully connected layers and proceeded by adding convolutional layers, starting with small kernel size and few output features and gradually increasing. Since the input image is only 16x16 in size, there is not much space for too many layers. For the same reason, adding max pooling layers or increasing the stride is not necessary.

In total, the tests were run on 6 different network architectures. An overview with brief info about each can be found in table 4.2. Individual network classes are defined in `object_detection/torch_nn_classes.py` and the learning processes were run and evaluated by `torch_model_training.py` script. The default hyperparameters were set to:

- activation function: ReLU (for all layers)
- optimizer: SGD
- learning rate: 0.001
- loss function: Binary Cross Entropy with logits
- number of epochs: 80
- batch size: 32

network	class name	architecture
MLP	mlp	3 fully connected layers
CNN1	cnn1	1 convolutional, 3 fully connected layers
CNN2	cnn2	2 convolutional, 3 fully connected layers (inspired by LeNet-5 [23])
CNN3	cnn3	2 convolutional, 3 fully connected layers (different kernel sizes and channels from CNN2)
CNN4	cnn4	3 convolutional, 3 fully connected layers
CNN5	cnn5	4 convolutional, 3 fully connected layers

Table 4.2: Info to used neural network architectures.

Table 4.3 shows the performance of each network in the form of validation loss on two cutout dataset combinations, and confusion matrix on their validation parts. Figures 4.4 and 4.5 show their learning curves. The best results were achieved by CNN4 with 3 convolutional layers. The scheme of this network's architecture is shown in Figure 4.6. With adding the 4th layer the validation loss got higher. We can also see that the amount of false negatives is minimum even for weaker models. What does decrease with network's complexity are false positives. Although in the case of the last model, CNN5, more false negatives start to occur and false positives disappear. The results from datasets 4 and 5 show the behavior more clearly. They were made from the set of photos with constant light conditions and, therefore, have less variety in samples, which makes them easier to learn.

network	datasets 1+2+3 validation loss	datasets 1+2+3 confusion matrix	datasets 4+5 validation loss	datasets 4+5 confusion matrix
MLP	0.085	$\begin{bmatrix} 315 & 9 \\ 0 & 96 \end{bmatrix}$	0.199	$\begin{bmatrix} 158 & 11 \\ 2 & 149 \end{bmatrix}$
CNN1	0.058	$\begin{bmatrix} 315 & 9 \\ 4 & 104 \end{bmatrix}$	0.243	$\begin{bmatrix} 159 & 8 \\ 1 & 152 \end{bmatrix}$
CNN2	0.059	$\begin{bmatrix} 314 & 6 \\ 1 & 99 \end{bmatrix}$	0.109	$\begin{bmatrix} 160 & 5 \\ 0 & 155 \end{bmatrix}$
CNN3	0.243	$\begin{bmatrix} 296 & 12 \\ 19 & 93 \end{bmatrix}$	0.070	$\begin{bmatrix} 158 & 5 \\ 2 & 155 \end{bmatrix}$
CNN4	0.057	$\begin{bmatrix} 315 & 10 \\ 0 & 95 \end{bmatrix}$	0.017	$\begin{bmatrix} 159 & 1 \\ 1 & 159 \end{bmatrix}$
CNN5	0.062	$\begin{bmatrix} 315 & 9 \\ 0 & 96 \end{bmatrix}$	0.051	$\begin{bmatrix} 156 & 0 \\ 4 & 160 \end{bmatrix}$

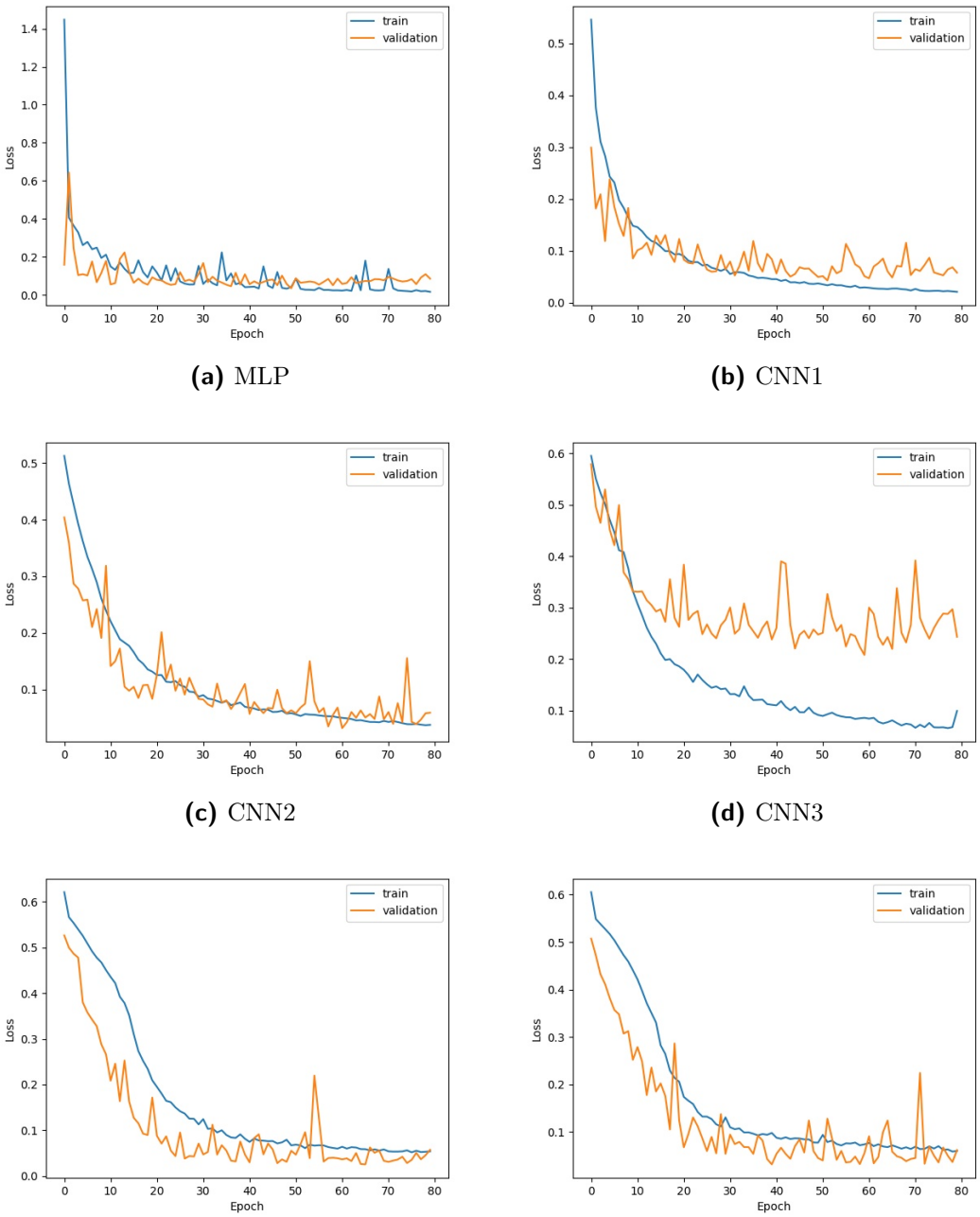
Table 4.3: Results of each network's learning process.

The next step is to choose the hyperparameters. For these experiments we used network CNN4, which had the best results. Since trying all the possible options would be very time consuming and the classifier already has quite a good performance with the initial values, we chose only a relevant subset for each parameter:

- activation function: ReLU (fixed)
- optimizer: SGD, Adam
- learning rate: 0.01, 0.001
- loss function: Binary Cross Entropy with logits (fixed)
- number of epochs: 50, 80, 100
- batch size: 16, 32, 64

A series of learning processes with different hyperparameter combinations were run. In the end, none of the results exceeded the default combination — SGD optimizer, learning rate 0.001, 80 epochs and batch size 32 — therefore it was kept. Also, since datasets 4 and 5 (leg skin with headlight) are more reliable, we decided to use the weights trained on these datasets.

As a final check, a sliding window classifier with CNN4 and its trained weights was run on several photos from the leg skin dataset. One of the results is shown in Figure 4.7. We can see that all the taxels except for those which are covered by glare from the headlight have been detected and just around 10-20 false positives (approximately 6% of the total detections), from which most are in the background and would be eliminated by the depth filter.

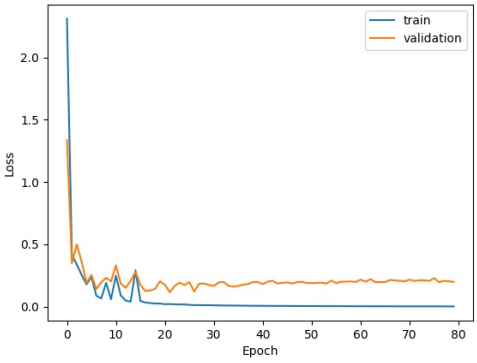


(a) MLP **(b)** CNN1

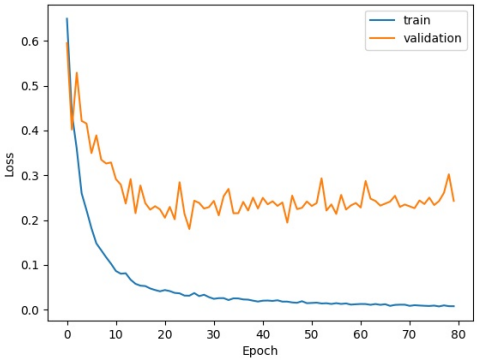
(c) CNN2 **(d)** CNN3

(e) CNN4 **(f)** CNN5

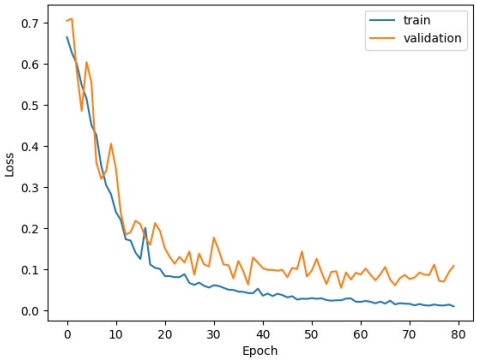
Figure 4.4: Learning curves of each network on datasets 1+2+3.



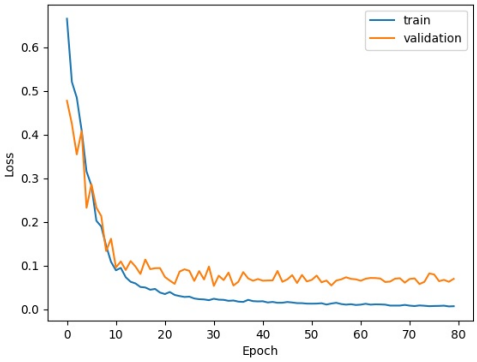
(a) MLP



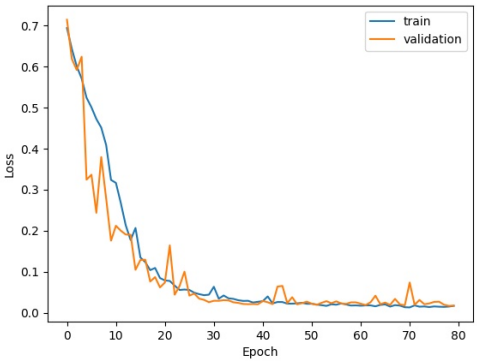
(b) CNN1



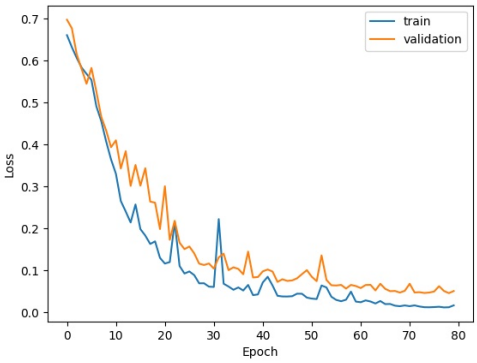
(c) CNN2



(d) CNN3



(e) CNN4



(f) CNN5

Figure 4.5: Learning curves of each network on datasets 4+5.

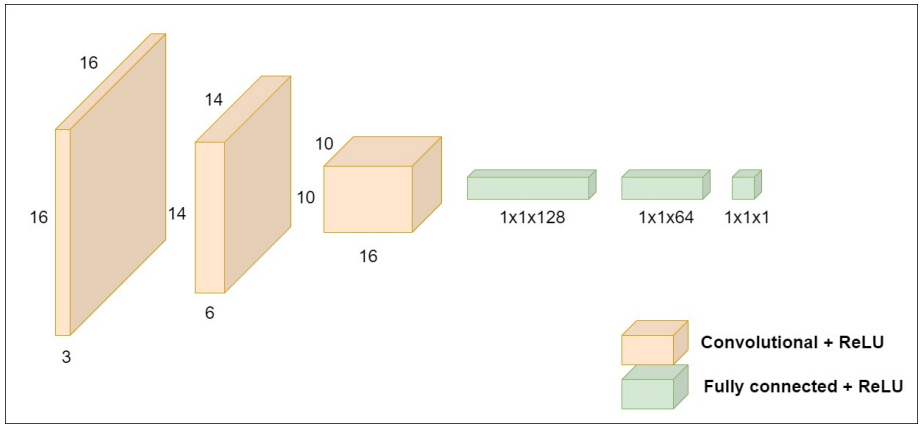


Figure 4.6: Scheme of CNN4 architecture.

Figure 4.7: Result of sliding window classifier with a trained CNN4 model: raw detections (top) and contour centroids (bottom).

4.2 Cascade classifier and point-cloud extraction

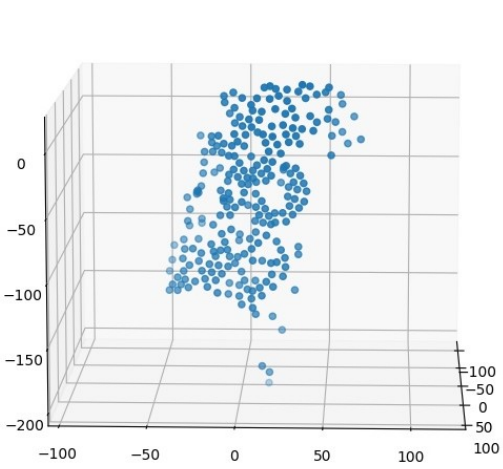
The process of taxel classification and point cloud extraction is implemented in `torch_cascade.py`. It needs an RGB-D camera connected to take pictures. Here is a brief schedule of the tasks in this script:

1. start the camera stream
2. take a snapshot (RGB picture + depth map)
3. apply the depth filter + make a point cloud (to be used later)
4. apply the color filter (optional)
5. run a sliding window CNN classifier on the remaining pixels
6. get contours in the detection mask and calculate their centroids
7. using the pixel coordinates of the centroids, find their 3D coordinates in the point cloud
8. display the 2D image with found centroids and ask the user to pick a center taxel of one of the triangles
9. transform the 3D points into a coordinate system with origin in that chosen taxel
10. save the 3D points as numpy array
11. visualize the result

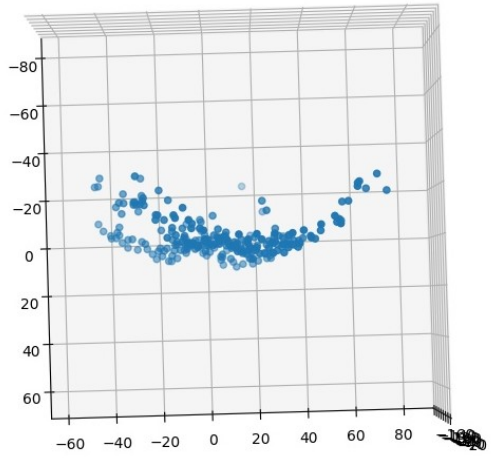
This whole process takes up to 10 minutes on a computer with 16GB of RAM. The output is a 3D point cloud of detected taxels + some extra points (false positives) in a coordinate system with origin in one of the triangle centers. The origin is moved there in order to be able to align this point cloud with the retrieved skin models. For some of the skin parts the 3D transformations to the triangle centers are known, therefore after the alignment we can use the inverse of this transformation to move the point cloud to the required coordinate system. Apart from moving the origin, the transformation also rotates the system to roughly match the orientation of the skin model (assuming the robot and camera are in upright position, otherwise it needs to be edited).

Figure 4.8 shows one of the results of running the script capturing the skin on lower right leg. From the 2D visualization it is visible that the classifier is not as accurate as on the dataset photo. There are more false positives present. The 3D plot also shows that some of the points are distorted. This could be caused by glare or a limited precision of the camera.

(a) 2D visualization



(b) 3D point cloud from front



(c) 3D point cloud from top

Figure 4.8: Result of capturing and classifying the skin on lower right leg.

4.3 Taxel ID matching

The following files and information from *robotology* GitHub repository [35] were used for retrieving information about the skin:

- configuration (.ini) files with 2D positions and orientations of the triangles [31]
- taxel placement in individual triangle
- 3D triangle positions for torso skin from `torsoTrianglesCAD.py`
- 3D triangle positions and rotations for lower right leg skin from `icub-models/iCub/robots/iCubGazeboV2_7/model.urdf`

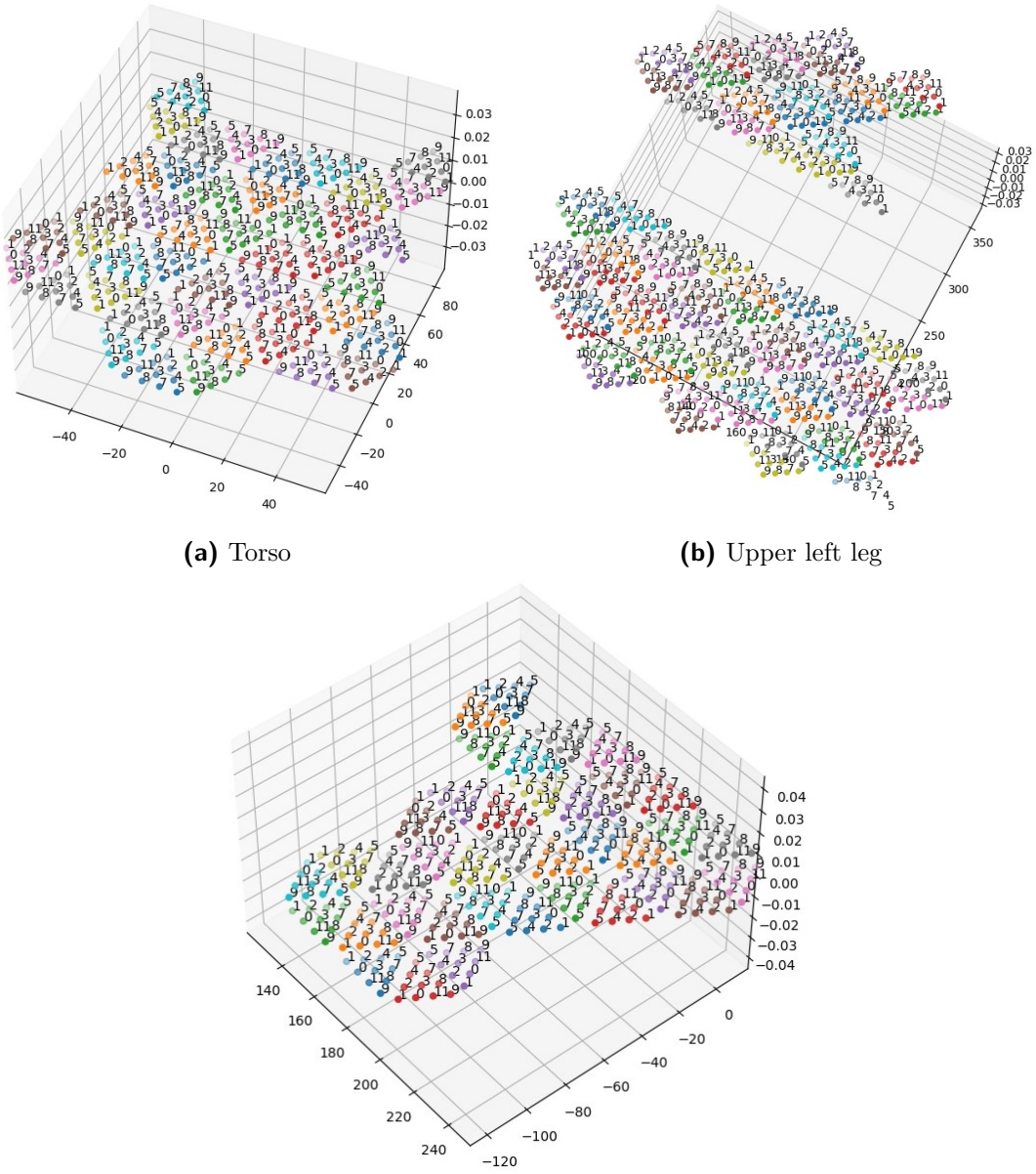
The Python codes used for extracting the information and converting them to numpy arrays are saved in `icub-skin-calibration/skin_models`. Functions implemented in `read_ini.py` extract triangles' 2D positions and orientations from an .ini file and use the given taxel placement in individual triangle to make a flat model of the skin, either in 2D or 3D with Z=0. Plots of the outputs from this script are shown in Figure 4.9.

The script `modeling3D.py` uses the provided triangle center 3D positions from `torsoTrianglesCAD.py` and 2D triangle orientations extracted from `torso.ini` to make an approximated 3D model of torso skin, where the triangle center positions are correct, but their 3D rotation wrong and taxels have a flat layout. See the output in Figure 4.10. The script in `skin_coords_from_urdf.py` extracts 3D positions **and rotations** of triangles in the lower part of right leg from `model.urdf` and (again with using 2D triangle orientations from .ini file) makes and approximated 3D model of lower right leg skin with correct 3D positions of the triangle centers and triangle rotations, and just the other (non-center) taxels left to calibrate. The result is plotted in Figure 4.11.

Unfortunately the .urdf model does not contain information about the other leg skin parts (right upper, left upper, left lower), so for these three we have only the 2D layout from .ini files. Table 4.4 shows a recapitulation of the reference information we have for each involved skin part.

skin part	2D layout	3D triangle center position	3D triangle orientation
torso	✓	✓	✗
upper right leg	✓	✗	✗
upper left leg	✓	✗	✗
lower right leg	✓	✓	✓
lower left leg	✓	✗	✗

Table 4.4: Available information to individual skin parts.



(a) Torso

(b) Upper left leg

(c) Lower left leg

Figure 4.9: Plots of flat skin models extracted from .ini files. (Also right leg skin models were extracted, they look similar to left leg skin.)

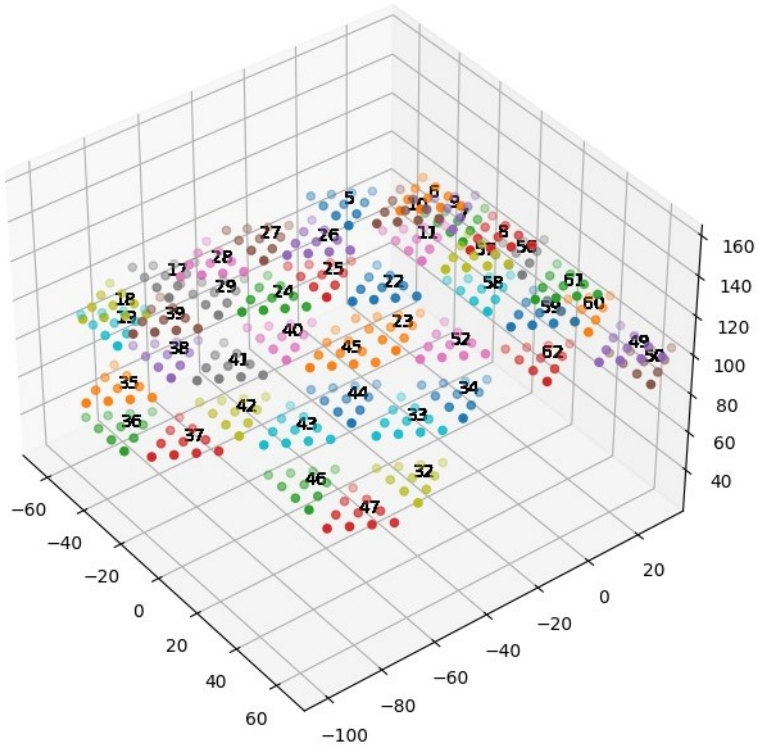


Figure 4.10: Approximated 3D torso skin model.

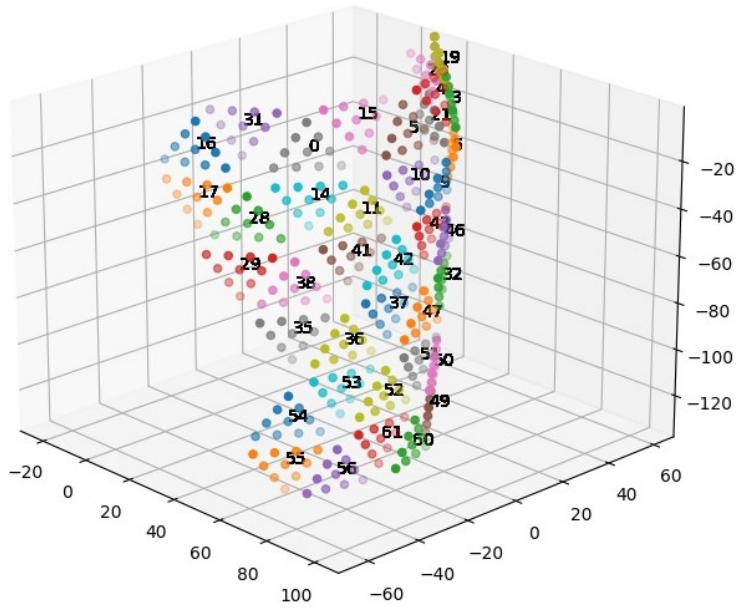


Figure 4.11: Approximated 3D lower right leg skin model.

4.3.1 Revealed errors in reference files

In the configuration file to lower right leg skin `right_leg_lower.ini` the position and orientation of triangles 19, 20 and 21 are wrong. Difference of the model from reality is depicted in Figure 4.12. This problem was reported as an issue in the respective GitHub repo [36]. Also, triangle 19 is duplicated (lines 106 and 109), though this did not cause any serious issue with modeling.

In the iCub kinematic model in `icub-models/iCub/robots/iCubGazeboV2_7/model.urdf` triangles 43 and 46 are swapped. Comparison with 2D model extracted from the configuration file `right_leg_lower.ini` is shown in Figure 4.13. In our code we deal with this issue by swapping the IDs additionally. This was also reported a GitHub issue [37].

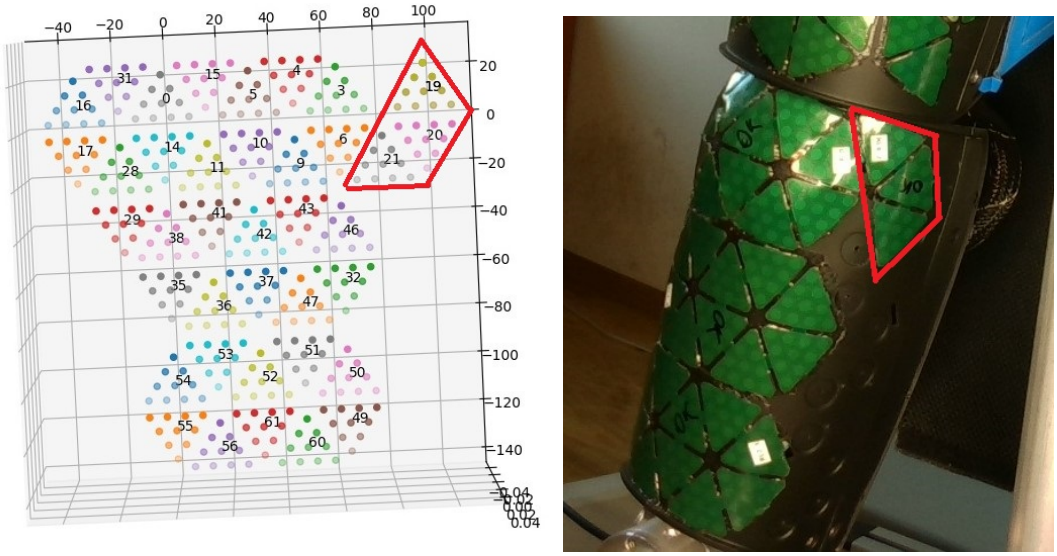


Figure 4.12: Comparison of lower right leg 2D model and reality.

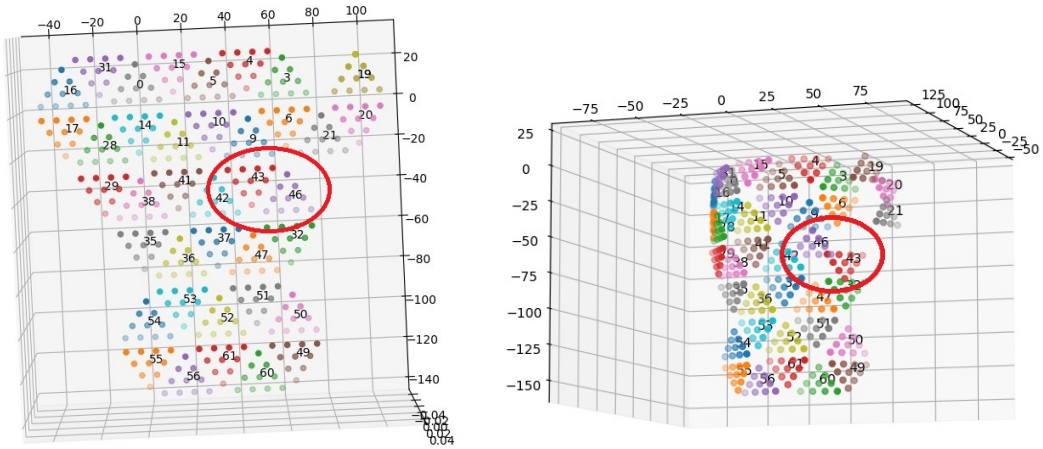


Figure 4.13: Comparison of the layout of triangles 43 and 46 in 2D data from .ini file and 3D data from .urdf file.

4.3.2 Model construction for lower leg skin

The model of lower right leg skin created by `skin_coords_from_urdf.py` script provides a calibration with reasonable accuracy by itself, since it contains precise 3D positions of triangle centers and their rotations in the coordinate frame of right lower leg joint.

Furthermore, assuming the left leg has the same kinematic structure as right leg (i.e., the coordinate frame has the same relative position to the skin), a model of lower left leg skin can be made by mirroring the model of lower right leg skin and assigning the correct IDs and orientations from lower left leg ini file. The script for modeling and visualizing the lower left leg skin is implemented together with the scripts for lower right leg skin in `skin_coords_from_urdf.py`. The only triangles that could not be obtained by mirroring are triangles 16, 17 and 28 (19, 20 and 21 in right leg), since they are placed differently. Their incorrect placement in the lower right leg ini file unfortunately affects both of the models: for right leg they have a wrong orientation and for left leg they have a wrong position. The comparison of 2D and 3D models with reality is shown in Figure 4.15.

Result text files with calibrations to both right and left lower leg skins are saved in `skin_models/calibrations`, together with `read_calib.py` script which will plot their taxels. Note that triangles 16, 17 and 28 of the right leg and 19, 20 and 21 of the left leg are affected by the above mentioned issue and therefore will have a larger error. Plots of the calibrations are shown in Figure 4.14.

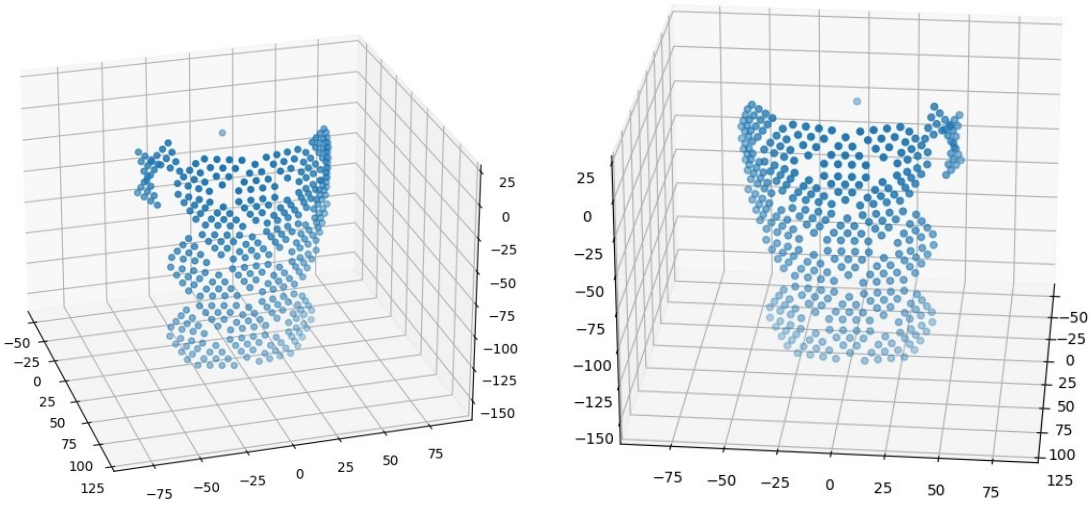
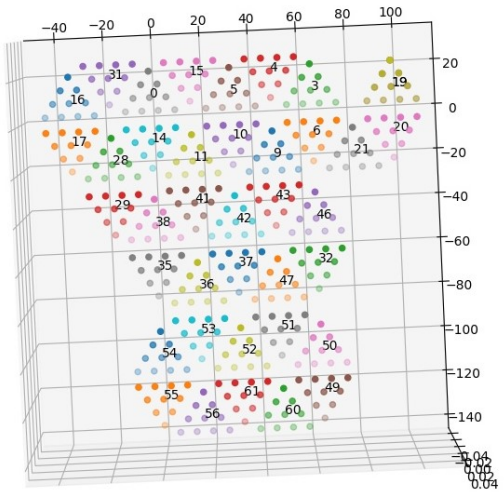
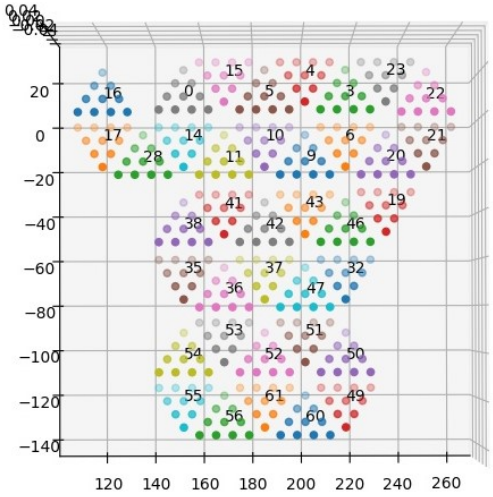


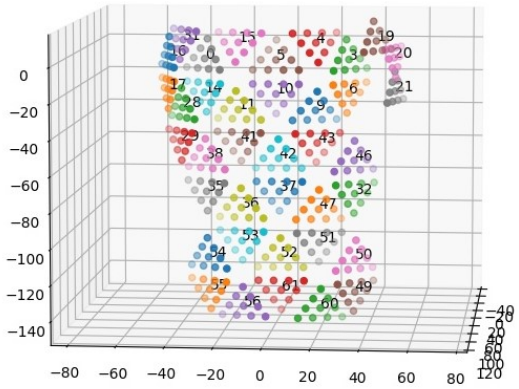
Figure 4.14: Lower leg skin calibration plots: left (left) and right (right).



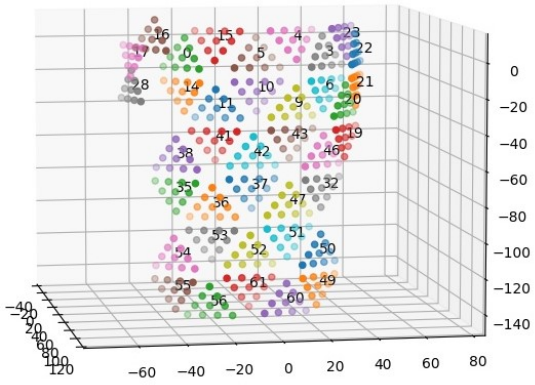
(a) Right 2D from .ini



(b) Left 2D from .ini



(c) Right 3D from .urdf



(d) Left 3D from mirroring right 3D

(e) Right reality

(f) Left reality

Figure 4.15: Lower leg model issue: differences between 2D models, 3D models and reality. Notice triangles 19, 20 and 21 on right leg and 16, 17 and 28 on left leg.

4.3.3 Summary

Figure 4.16 shows a summary of the processes implemented in this work and how their outputs were planned to be connected to form a calibration. Pipeline on the left shows the taxel recognition and 3D reconstruction process, whose output is an unlabeled 3D point cloud of detected taxels. The pipeline on the right depicts the process of making skin models out of the provided configuration files and iCub's kinematic model. The resulting point clouds from these two processes were planned to be aligned by a point cloud registration algorithm. Due to distortions in 3D point cloud from 3D reconstruction and several errors in configuration files this process was left to be performed later.

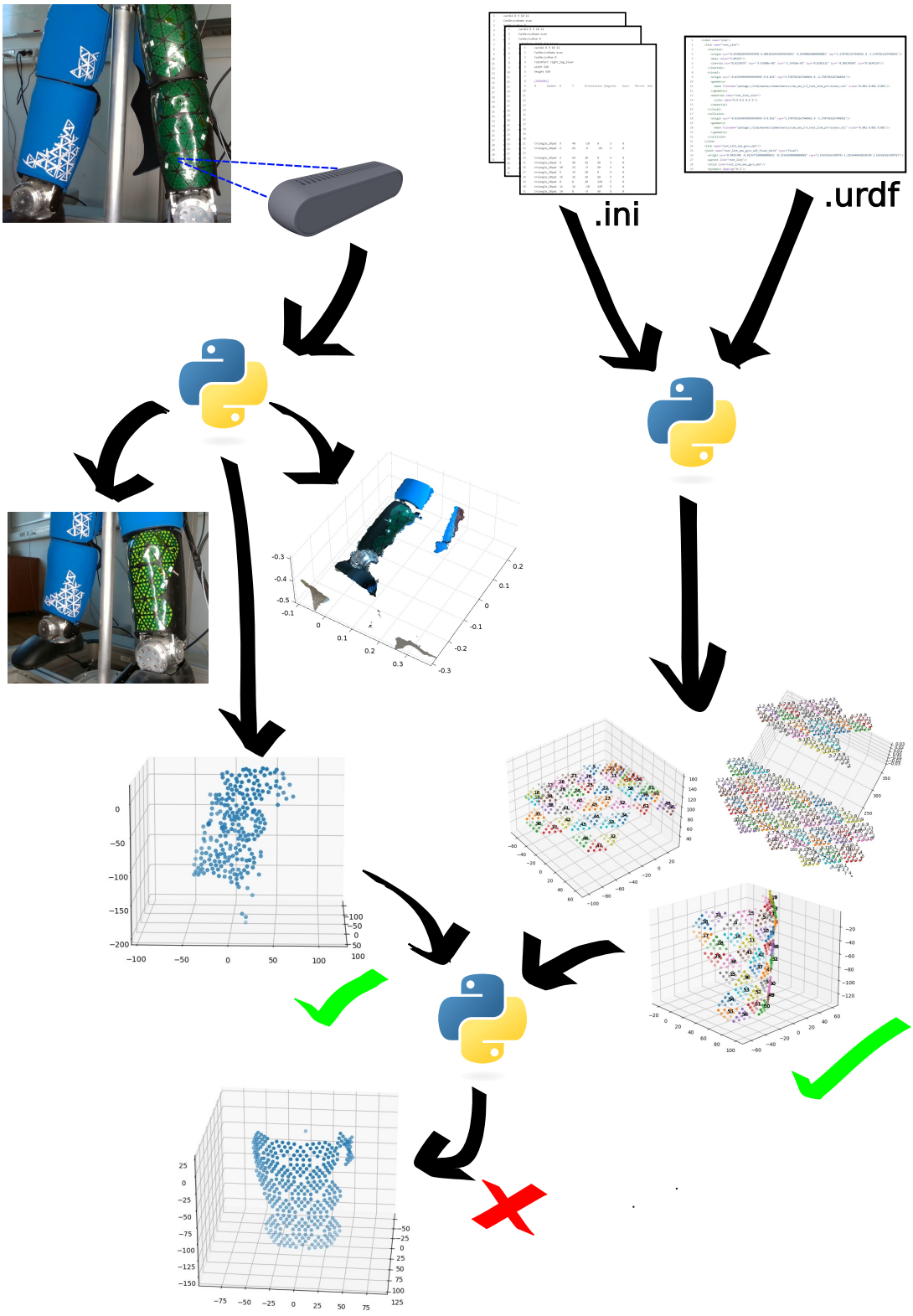


Figure 4.16: Pipeline of the implemented processes.

Chapter 5

Discussion, Conclusion and Future Work

We have developed and tuned a CNN classifier for taxel recognition in an RGB image. In good light conditions it is able to detect vast majority of the visible taxels in the scene with a reasonable amount of false positives. The best discovered model had a 99% accuracy on validation dataset. Using this model, combined with a depth filter and color filter, we extract a 3D point cloud of detected taxels from an RGB-D camera input. This process is almost fully automatic, it takes just a few minutes and requires attention of the user only once: at the end to pick a point (a triangle center) to move the coordinate system origin into.

The resulting 3D models are a bit distorted, possibly due to the camera's limited precision. Another problem was with the headlight that was used for lighting up the scene. The light was not diffused enough and it was reflecting from the skin, making some of the taxels unrecognizable. This issue could be fixed by using a more diffused light. Afterwards if there are still some deviations, a depth camera with better precision could help.

An alternative approach, which would most likely maximize the accuracy, would be making several shots for each skin part and making a 3D reconstruction using another program, just like it was done in previous work on Nao skin [1], but with less photos and additional depth information instead. A suitable program for this would be for example Colmap. In addition to each photo and depth map we would also have the coordinates of detected taxels, which Colmap could also localize in the final model and align them together. This procedure would require a bit more work from the user, but it would deliver a calibration with much better accuracy, if a point cloud from one depth camera shot is not enough.

From the available iCub skin configuration files we have extracted labeled 2D models of torso and leg skin parts, with both triangle IDs and taxel IDs assigned. Furthermore for torso skin and lower right leg skin we constructed an approximated 3D model based on additional information about triangle center positions. The lower right leg model is reasonably precise even without further calibration, since both positions and orientations of the triangles were available and 2D taxel layout within a triangle as well, so the only inaccuracy comes from neglecting the curvature of the surface and assuming triangles are flat. Since no calibration existed for the leg skin yet, this model is a useful contribution.

Based on the provided configuration files we assumed the model of lower left leg skin could be retrieved by mirroring the model of lower right leg skin (+ editing IDs). In reality there are 3 triangles placed differently on the left side. The model of lower left skin was

made nonetheless, with the remaining 35 triangles placed correctly.

During the extraction of the models several errors were revealed in the configuration files and the kinematic model. Due to these errors 3 (of the total 38) triangles in the lower leg skin models on both sides are placed incorrectly. Due to this issue and the 3D point clouds from the taxel detector being too distorted the point cloud alignment was not performed. This can be done in the future, when the configuration files are corrected and the reference models are complete.

One more extension could be possible to retrieve a skin model for the parts, where only a 2D layout is available. The flat skin model could be represented as a kinematic structure with each triangle being a link. Then, using an iterative optimization it could be roughly fit to match the 3D shape of the skin retrieved from the detection and 3D reconstruction process. Afterwards the ICP algorithm could have a higher chance of success.

Bibliography

[1] B. Potočná, "Artificial skin calibration for a humanoid robot: Comparing or combining "self-touch" and 3d reconstruction from images," Bachelor's Thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2020.

[2] L. Rustler, B. Potocna, M. Polic, K. Stepanova, and M. Hoffmann, "Spatial calibration of whole-body artificial skin on a humanoid robot: comparing self-contact, 3d reconstruction, and cad-based calibration," pp. 445–452, 2021.

[3] A. Del Prete, S. Denei, L. Natale, F. M., F. Nori, G. Cannata, and G. Metta, "Skin spatial calibration using force/torque measurements," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2011, pp. 3694 –3700.

[4] "iCub calibration repository," https://github.com/robotology/icub-main/tree/master/app/skinGui/conf/positions, [Online, accessed February 2023].

[5] 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)*, 2014, pp. 2305–2312.

[6] J. Kangro, S. Traversaro, D. Pucci, and F. Nori, "Skin normal force calibration using vacuum bags," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 401–406.

[7] J. Kangro, A. V. Sureshbabu, S. Traversaro, D. Pucci, and F. Nori, "A plenum-based calibration device for tactile sensor arrays," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3937–3943, 2018.

[8] A. Albini, S. Denei, and G. Cannata, "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.* IEEE, 2017, pp. 153–159.

[9] I. Birri, B. S. B. Dewantara, and D. Pramadihanto, "3d object detection and recognition based on rgbd images for healthcare robot," pp. 173–178, 2021.

[10] S. Yeom and J. Ha, "Hybridnet: Indoor segmentation with range and voxel fusion," pp. 1509–1515, 2021.

[11] N. O'Mahony, S. Campbell, L. Krpalkova, A. Carvalho, G. A. Velasco-Hernández, D. Riordan, and J. Walsh, "Convolutional neural networks for 3d vision system data: A review," in *2018 12th International Conference on Sensing Technology (ICST)*, 2018, pp. 160–165.

[12] G. Metta, L. Natale, F. Nori, and G. Sandini, "The icub project: An open source platform for research in embodied cognition," in *Advanced Robotics and its Social Impacts*, 2011, pp. 24–26.

[13] 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*, vol. 13, no. 10, pp. 3910–3917, 2013.

[14] "Depth camera D435 - product webpage," https://www.intelrealsense.com/depth-camera-d435/, [Online, accessed April 2023].

[15] "Instance segmentation with realsense," https://github.com/bowu1004/instance_segmentation_RealSense, [Online, accessed April 2023].

[16] "Intel realsense documentation," https://dev.intelrealsense.com/docs, [Online, accessed April 2023].

[17] "opencv documentation," https://docs.opencv.org/4.7.0/, [Online, accessed April 2023].

[18] "Pytorch documentation," https://pytorch.org/docs/stable/index.html, [Online, accessed April 2023].

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, Massachusets 02142: Massachusetts Institute of Technology, 2016.

[20] P. Pošík, "Neural networks," https://cw.fel.cvut.cz/old/_media/courses/be3m33ui/b12nn-handouts.pdf, [Lecture handouts from Artificial Intelligence].

[21] "Imagenet," https://image-net.org/index.php, [Online, accessed May 2023].

[22] "Cifar," https://www.cs.toronto.edu/~kriz/cifar.html, [Online, accessed May 2023].

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[26] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[28] "Vgg-16 scheme," https://www.researchgate.net/profile/Timea-Bezdan/publication/333242381/figure/fig2/AS:760979981860866@1558443174380/VGGNet-architecture-19.ppm, [Online, acessed May 2023].

[29] S. Ramesh, "A guide to an efficient way to build neural network architectures-part ii: Hyper-parameter selection and tuning for convolutional neural networks using hyperas on fashion-mnist," https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7, [Online, acessed May2023].

[30] "How to use learning curves to diagnose machine learning model performance," https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/, [Online, accessed May 2023].

[31] "iCub skin configuration files," https://github.com/robotology/icub-main/tree/master/app/skinGui/conf/skinGui, [Online accessed May 2023].

[32] J. Zhang, Y. Yao, and B. Deng, "Fast and robust iterative closest point," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3450–3466, 2022.

[33] "iCub skin calibration repository," https://gitlab.fel.cvut.cz/body-schema/icub/icub-skin-calibration.

[34] "Dataset picture folder," https://drive.google.com/drive/folders/1RiEyTiTRvUaopRxY8E2Dle_psDnBdUJW?usp=sharing.

[35] "Robotology GitHub repository," https://github.com/robotology.

[36] "GitHub issue on lower right leg skin configuration file," https://github.com/robotology/icub-main/issues/880.

[37] "GitHub issue on icub gazebo 2.7 model," https://github.com/robotology/icub-models/issues/200.