Estimation of hand position with a point cloud

handed in
PRACTICAL COURSE

Bachelor of Science Clarissa Hamann

born on the 15.02.1995
living in:
Guldeinstr. 47
80339 Munich
Tel.: 017645854451

Human-centered Assistive Robotics
Technical University of Munich

Univ.-Prof. Dr.-Ing. Dongheui Lee

Supervisor: Shile Li

In your final hardback copy, replace this page with the signed exercise sheet.

Before modifying this document, READ THE INSTRUCTIONS AND GUIDELINES!

Abstract

This Project is about using a point cloud for hand pose estimation. Point Clouds have two big advantages: full 3D information and not using a unnecessary big amount of memory. The already exciting PointNet gets modified for the this purpose. PointNet is previously successful for segmentation task. In the end the deep learning algorithm can predict the joint locations of a hand with an average error of 2cm.

Contents

Chapter 1

Introduction

In general estimation of a 3D pose of a hand is needed for many human-computer interaction tasks. It is useful to recover the joint location e.g for gesture recognition used for communication such as navigation or driver interaction. [TJCTK14] An other example is to teach a robot grasping capability by human demonstration. For all this the location of the hand and the gestures have to be detected. Therefor it is useful to have an algorithm to estimate the joint locations of hands.

This project is about hand position estimation with using a point cloud as data set for the input. Deep learning algorithms are already existing for this problem. They are using the most of the times a collection of depth images or 3D voxel grids as input, because they have a structured input and these methods are compatible with common convolutional neural networks (CNN). But these two methods have also some big disadvantages. Depth images cannot fully exploit 3D information and the appearance depends on the camera parameters. 3D voxel grids require an unnecessary voluminous amount of data.[QSMG17]
To solve these problem a point cloud of a hand should be used as a direct input to get the full 3D spatial information about the input independent of the used camera. Also additionally transformations should be avoided and prevent the process of using a big memory.
Some difficulties have to be handled for this purpose. A 3D point cloud consists always of an unstructured collection of points. The order of the input is not fixed. So each point has be treated separately. An other problem is that common deep learning elements such as CNN cannot be used for point clouds.
The aim is to find a deep learning algorithm, that can use a point cloud as input and process independent each point of the point cloud. For this task Stanford university has developed the PointNet. PointNet is a deep learning architecture capable to deal with 3D geometric data. It has an unified architecture and is already used successfully for several tasks: object classification, part segmentation, to scene semantic parsing [QSMG17].
The problem of hand pose estimation should be solved by the PointNet. For this

the PointNet is going to be modified, that it is useable for our hand pose estimation task.
The ICVL dataset of the Imperial College London is used for the training and afterwards the testing sequence of the dataset for inference of the program. [ICV]

Deep Learning
Deep Learning is a form of machine learning. Since a few years, supervised deep neural networks are the most effective and successful tools for pattern recognition. [Sch15] After decades it became more and more popular and is nowadays relevant for applications like artificial intelligence.
It is about teaching a computer complicated concepts by building them out of simpler ones. The whole concept is about learning from experience and big datasets of examples. [GBCB16]
The neural networks for deep learning have multiple layers - called multilayer perceptrons. They are responsible for the predicted values.

PointNet of Stanford University
From Charles R. Qi's, Hao Su's Kaichun Mo's and Leonidas J. Guibas's
PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation [QSMG17]:

In this paper [QSMG17] propose a method of object classification, part segmentation, to scene semantic parsing with the data format point cloud. They have developed a deep learning architecture usable for 3D geometric data - the PointNet. This neural network takes directly point clouds as input and treats each point independently. The advantage of this is getting the full information about the three coordinates (x, y, z) and not having an unnecessary voluminous amount of data.
In general Point sets have three properties: unordered, interaction among points and invariance under transformation. Hence, the network has to be invariant of the order of the N input points, capture local structures of close points and invariant to certain transformations, like rotating and translating. The PointNet is a solution for all these problems and is already successfully used for Classification and Segmentation tasks.
The main idea is to treat every point separated. It estimates point-wise features for each point and use max pooling in the end to extract a global feature. The features learn during the training process how to select interesting points for getting a good prediction of the values in the end.[QSMG17]

Further in the main part the whole process is described. It includes the preprocessing of the ICVL dataset, how the PointNet is working with training and inference and in the end the evaluation of the developed program's results.

Chapter 2

Preprocessing

Training requires a set of training data, which consists of a series of input and associated output vectors. [GD98] For this project the data is given by the ICVL dataset.

It is necessary to preprocess the raw data of the ICVL dataset, to get ride of noise and have in the end a normalized point cloud with 16 joint positions in x, y and z coordinates. This preprocessed data is used as input for the PointNet to learn and test the deep learning algorithm. The advantage of converting the coordinates to world coordinates is an independence of camera parameters.

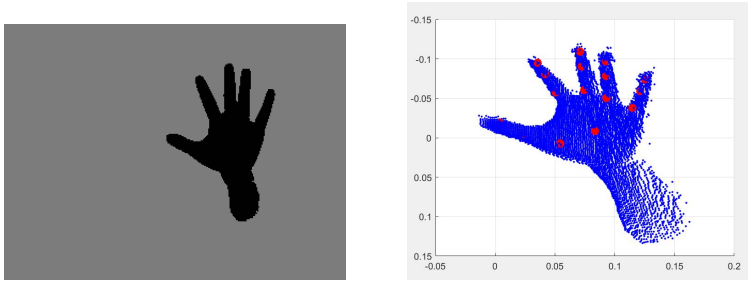


Figure 2.1: Preprocessing: depth image of hand from ICVL dataset (left) and corresponding point cloud (right)

It has to be distinguished between training and testing data. The training ICVL dataset delivers the clean depth image without noise and separate joint positions (target values for the prediction). Whereas the testing data of the ICVL dataset includes noise, which has to be filtered. Also the testing data includes the 16 joint locations of each hand. With these joint locations our results are comparable in the end and the average of the error of the algorithm can be calculated.

2.1 ICVL dataset

The ICVL Hand Posture Dataset of the Imperial College London can be downloaded for free. It contains of training and testing data of depth images of hands and of the corresponding joint locations of each hand saved in a .txt file.
The training set has 31006 files of hands. The testing has two test sequences, one with 702 images and the other one with 894 images (in total 1596 testing examples). 16 x 3 numbers belong to each image. They indicate the depth image coordinates (u,v and d) of 16 joint locations. These locations are centered. The order of the 16 joints is Palm, Thumb root, Thumb mid, Thumb tip, Index root, Index mid, Index tip, Middle root, Middle mid, Middle tip, Ring root, Ring mid, Ring tip, Pinky root, Pinky mid, Pinky tip. The Intel Creative depth sensor is been used for recording this dataset. [ICV]

2.2 Training dataset

Data import
In general it is important to ensure to combine the right depth image with the right joint locations of the .txt file. The .txt file contains the file name for the corresponding image and the Ground Truth data (joint locations). The Ground Truth is saved in an array (331006 x 3 x 16).
Each separate image gets converted to a point cloud (world coordinates: x, y, z).

Depth image to Point Cloud
To get all the relevant data of the hand, image filtering is needed. In this case the background get deleted and the remaining points get converted from depth image coordinates (u, v, d) to x, y and z coordinates (The equation for transformation is in the next section 2.1, 2.2, 2.3).

uvd2xyz
To get a point cloud converting the Ground Truth and all points of the depth image from uvd to xyz coordinates is necessary. Afterwards the hand can be pictured as a point cloud.

$$x = (u - u_0) \cdot \frac{d}{f_x} \tag{2.1}$$

$$y = (v - v_0) \cdot \frac{d}{f_y} \tag{2.2}$$

$$z = d \tag{2.3}$$

with
$u_0 = 160$
$v_0 = 120$

$f_x = 240.99$
$f_y = 240.99$
Calibration Parameters of the Intel Creative Depth Sensor
These parameters u_0, v_0, f_x and f_y are individual for each camera.

Normalization of the Point Cloud This next step is for normalization of the point cloud. The aim is to have all hands in the centre of the coordinate system. To do this the centre of the Ground Truth data $\mathbf{c}$ is subtracted from each i-th element of the point cloud $\mathbf{P}_i$ and from all Ground Truth positions.

$$\mathbf{P}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{2.4}$$

with $\mathbf{P} \in \mathbb{R}^{n \times 3}$

$$\mathbf{c} = \frac{1}{n} \cdot \sum_{i=1}^{n} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{2.5}$$

n: amount of points in point cloud
c: centre of Ground Truth Values (joint locations)

$$\mathbf{P}_c = \mathbf{P}_i - \mathbf{c} \tag{2.6}$$

$\mathbf{P}_c$ is the centred point cloud. It is used for all further calculations and transformations.

write HDF5 file
Hierarchical Data Format (HDF) is a data format used for saving big amounts of data. It stores large numerical arrays of homogeneous type. It can store multidimensional data and tables very efficient.[Col13]

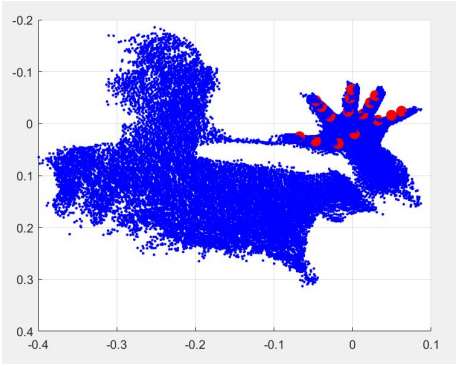
In the end all the data of the point cloud, the 16 joint positions and the center of each hand is saved in a HDF5 file. In one file are 30.000 different point clouds with 4.000 random points of each cloud and the corresponding data (joint locations and centre). All data is stored in x, y, z coordinates.

2.3 Testing dataset

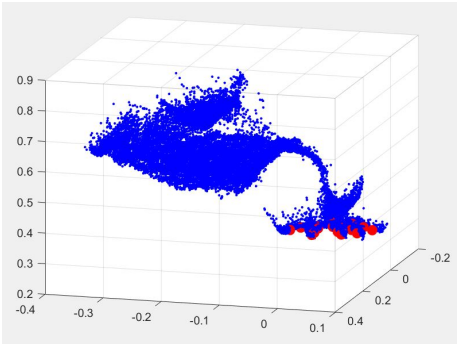
The only difference between the preprocessing of the training data and the test data is the conversion from depth image to a useful point cloud without noise. The test data depth images have not just information about the hand, they also have information about the closest environment (figure 2.2(a) and 2.2(b)). Now it is important to detect and delete the noise and save the clean data of the hand. For this a bounding box is created to eliminate all data more far way than 16 cm of the centre of the hand (equation: 2.5). Some results before and after applying the bounding box are given in figure 2.2(c) and 2.2(d).

Equation for Bounding Box:

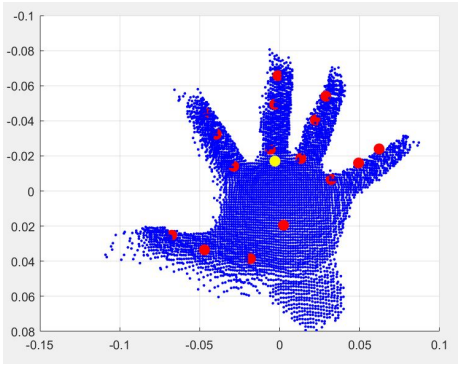
$$||\mathbf{P}_\mathrm{i} - \mathbf{c}|| < 0.16 \tag{2.7}$$



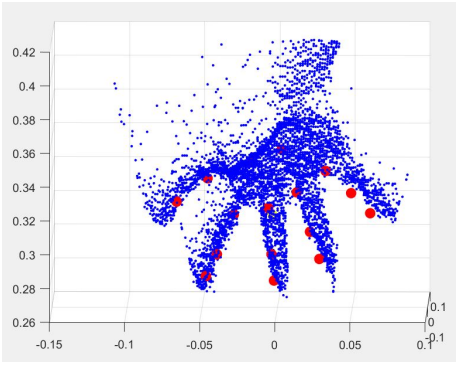
(a) example point cloud before filtering with the bounding box



(b) rotated example of point cloud before filtering with the bounding box



(c) example point cloud before filtering with the bounding box



(d) rotated example of point cloud before filtering with the bounding box

Figure 2.2: Example for point clouds before and after applying a bounding box

Chapter 3

Algorithm for hand pose estimation

The used algorithm for hand pose estimation is described in the following chapter. For the explanation the process is structured in four parts: creating the dataset, training process, calculating the loss and inference.
The framework TensorFlow is used for all deep learning elements. Tensorflow is released as an open-source project and is very popular for machine learning applications. It enables developers to experiment with novel optimization and training algorithms. [ABC+16]
For training and testing data the ICVL dataset is used as mentioned in chapter 2.1.

3.1 Creating the dataset with data augmentation

The given data of the ICVL dataset will be taken for creating a dataset for the training process. To get a more precise result it is better to have more data. In this case augmentation is used to enlarge the dataset of different hand shapes. Augmentation means to create artificially more data with translation, changing the viewpoint or size or modifying the illumination. For deep learning algorithms already a small change of the image is enough to use it as new training data.[Che18]

In this program is worked with scaling, rotation and translation. The rotation is done around camera view (z axis), x axis in a range from -180 to 180 degree and y axis in an area from -5.4 to 5.4 degrees. For scaling and the translation the program works with random values.
A bounding box is created to get rid of useless data for the training and testing data. Of course for the testing data augmentation is unneeded.

3.2 Training - PointNet

PointNet is a neural network for deep learning on point sets [QSMG17]. As mentioned before Standford University uses it for training 3D classification and segmentation tasks. [QSMG17]
The main idea is to modify the PointNet and apply it for hand position estimation. The advantage of this method is getting information about x, y and z coordinates of the hand's position and using a more effective algorithm than over methods.

PointNet Architecture
The classification network takes n points as input. It takes the 4000 saved points of the preprocessed point cloud for the hand pose estimation. ReLU (Rectifies Linear Units) is the non-linear activation function in this network for multi-layer perceptrons (mlp - chapter 3.2). After multiple mlps with different layer size (64, 128, 1024) max pooling follows. This is used for getting the global feature - the maximum value of each row of the calculated n x 1024 matrix. After the dropout (mlp) the network has an output of 48 values. This 48 values are the 16x3 predicted joint locations for the hand.

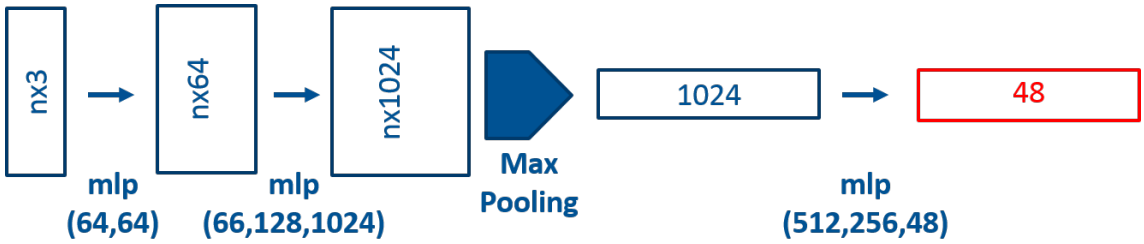


Figure 3.1: PointNet for hand pose estimation

Multilayer perceptron - mlp
Multilayer perceptron is a feed-forward artificial neural network [GBCB16]. It has multiple layers (minimum three) with several nodes (layersize) and applies to each node, except input nodes, an activation function. The input layer is responsible to pass the weighted input vector to the network. MLPs are fully connected, that means each node is connected to every node in the next and previous layer. They have the ability to learn through training with an dataset. For the training the backpropagation algorithm is used. The aim of the algorithm is to get the overall error satisfactorily small. It compared the actual output with the target and adjust due to this error the weights. This has to be repeated until the error is small enough to get in the end an precise result. [GD98]

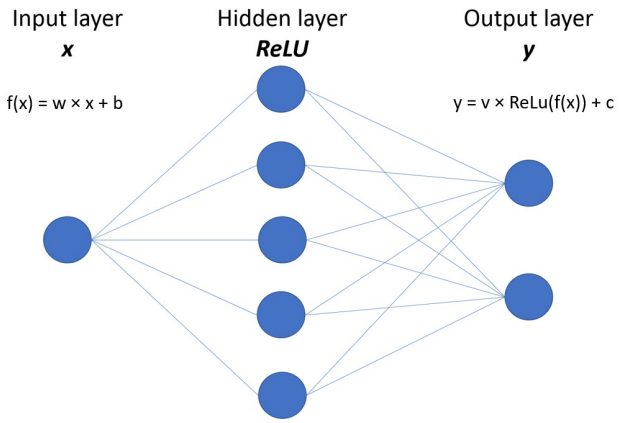


Figure 3.2: Multilayer perceptron with one input, one hidden (with layersize 5) and one output layer

In this project the nonlinear activation function is ReLu (figure 3.2). ReLu is the most popular activation function for deep neural networks.

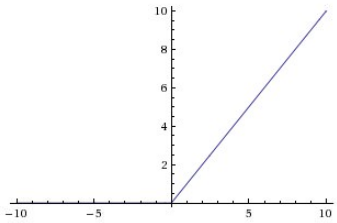


Figure 3.3: activation function: Rectifier Linear Units (ReLu)

At first the mlp applies a linear transformation to the input x. The weights w_j and b_j are for the first data input a random value and afterwards the weights get optimized due to the amount of the calculated error by the loss function. Adjusting the weights happens with the help of backpropagation.
The weights w_j and b_j are vectors of the size 1 x J with J = layer size.

$$\mathbf{f_j}(\mathbf{x}) = w_j^T \cdot x + b_j \tag{3.1}$$

for j= 1,...J
The activation function ReLU maps the weighted inputs to the output. It returns 0 for any negative input, for all positive inputs it returns x.

$$ReLU = max(0, x) \tag{3.2}$$

In the end the outputs of the activation function are weighted again with v_j^k and c_k (3.3). Now it is possible to repeat the activation step and extend the network or

come to the final step: max pooling. Afterwards the output is the final result - in this case the 16 predicted joint locations of the hand.

$$\mathbf{y_k}(\mathbf{x}) = \sum_j ReLU(f_j(x)) \cdot v_j^k + c_k \tag{3.3}$$

j: layer, k: node

3.3 Loss function

In general the loss function (also called cost function) calculates the difference between the predicted output and the ground truth value - calculating the error of the predicted value. It is measured the accuracy of the algorithm with that error value. Of course the aim is to minimize this difference. That means to make it more precise. Concerning to this value of the difference the weights in the neural network are changing (backpropagation). The aim is to optimize the weights so that the estimated values are getting better until an acceptable small error.
During the training with the PointNet the L2 loss is used to optimize the weights.

Square loss $\mathbf{Loss_{L2}}$:

$$e_{\mathrm{L2}} = \frac{1}{2} \cdot \sum_{i=1}^{16} ||J_{\mathrm{i}} - J_{\mathrm{i,\,predicted}}|| \tag{3.4}$$

The diagram shows the square loss e_{L2}. The error is getting smaller with raising number of trained examples, because due to the error the weights get adapted and the prediction optimized. Because of the step by step optimization the graphs shows this fluctuation. The jump arise, due to a change of the dataset in between from a smaller to bigger one with more examples. The aim was getting a better prediction.

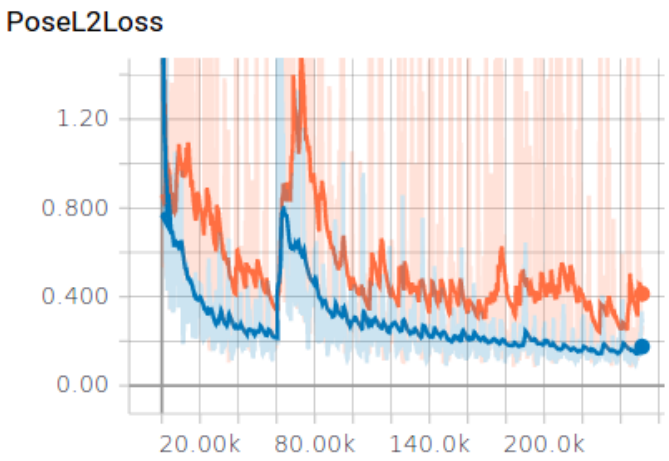


Figure 3.4: Loss Function during Training out of tensorboard

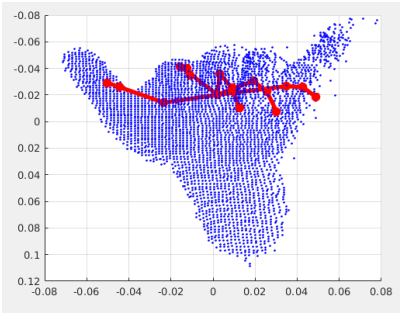
3.4 Inference

With the training process of the hand pose estimation the program learned how to predicted good values for the 16 joint positions. The whole model is saved in the checkpoint file. Now during the inference the model get tested with the preprocessed testing ICVL dataset. It calculates a prediction of the joint locations of unknown point clouds.
Afterwards the results are saved in a .txt file. To see the result the .txt file has to be evaluated and has to be compared with the corresponding hand's point cloud and their target joint locations.

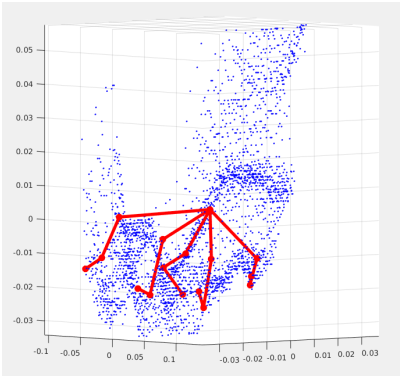
Chapter 4

Results and Evaluation

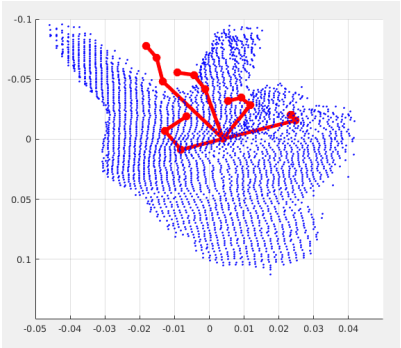
Some example results of the predicted joint locations are shown in 4.1. The input point cloud and the corresponding predicted joints locations are brought together. Some results are better than other ones.



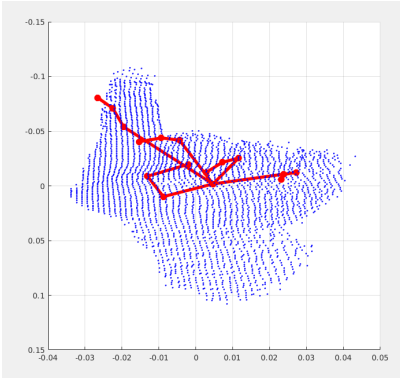
(a) predicted points testing example hand 50



(b) predicted points testing example hand 50



(c) predicted points testing example hand 60

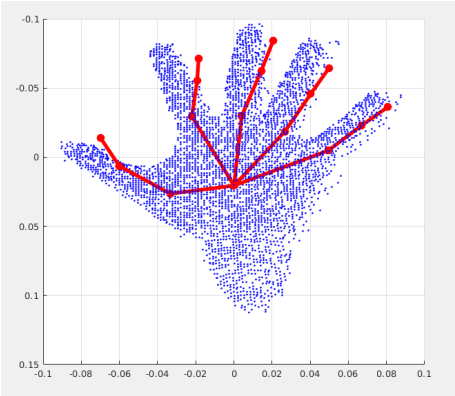


(d) predicted points testing example hand 10

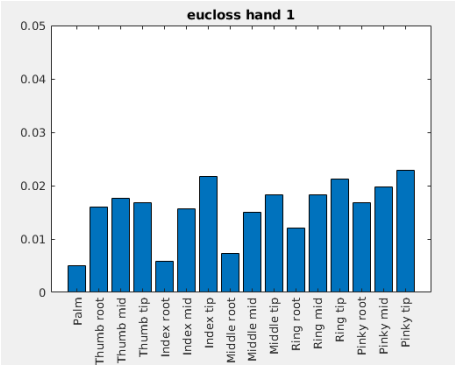
Figure 4.1: Example results of predicted joint positions (red points)

The resulting output is not always the perfect solution. It is remarkable the differences between 4.2(a) and 4.2(c). The both images are a good example to show that it is more difficult for the algorithm to get a precise prediction if the fingers are crossed. The reason for this that information is hidden by other points.
The error of the prediction is calculated with 4.1 and it also shows us the problem with hidden fingers. The error of the joint position 'middle tip' is the highest 4.2(d).

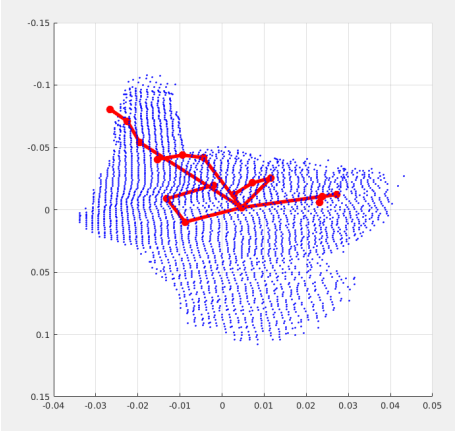
$$e = ||J_{\mathrm{i}} - J_{\mathrm{i,\,predicted}}|| \tag{4.1}$$



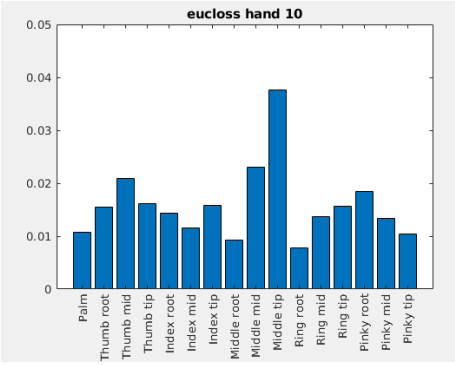
(a) predicted points testing example hand 1



(b) Error of predicted joint locations



(c) predicted points testing example hand 10



(d) Error of predicted joint locations

Figure 4.2: example results of predicted joint positions (red points) and corresponding error (equation: 4.1) of each position (diagram)

Considering the error of equation 4.2 there is a relation between the distance of the joint location to the centre of the hand - the farther the point (tip of the finger),

the larger the error of the prediction (as shown in figure 4.3). To estimate a good value for the tip of any finger is more difficult because of the higher variance of the value. The tip of the finger can move the biggest difference to the original position.

$$e = \frac{1}{F} \cdot \sum_{i=1}^{F} ||J_{\mathrm{i}} - J_{\mathrm{i,\,predicted}}|| \tag{4.2}$$

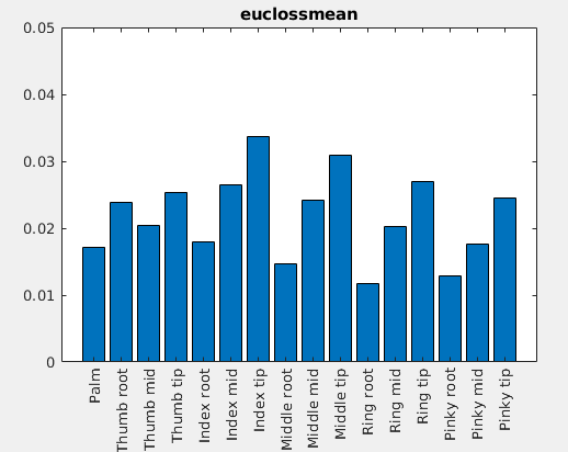


Figure 4.3: Error (4.2) for each separate joint location over all 1596 testing examples

All in all, the average of the error over all examples and predicted joint locations is: **0.0218** (4.3). The prediction diverges around 2 cm to the target values.

$$e = \frac{1}{F} \cdot \frac{1}{16} \cdot \sum_{i=1}^{F} \cdot \sum_{i=1}^{16} ||J_{\mathrm{i}} - J_{\mathrm{i,\,predicted}}|| \tag{4.3}$$

Chapter 5

Conclusion

All in all, it can be said that the hand pose estimation with a point cloud as input worked well with the help of the PointNet. It was possible to modify this deep neural network to give a prediction of the 16 joint locations of each hand. The PointNet could handle the unstructured input and the result benefits from the full usage of 3D spatial information of the hands during our prediction. It is not such a big memory size needed than for 3D voxel grid methods, too.
It is important to mention that these method has limitations, too. It is not possible to get a good prediction of positions hidden by other fingers or anything else. It also has to be kept in mind that the average error is around 2cm and for some applications a more precise result is needed.

List of Figures

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[Che18] Long Cheng. *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13–16, 2018, Proceedings, Part III*. Springer, 2018.

[Col13] Andrew Collette. *Python and HDF5: Unlocking Scientific Data*. " O'Reilly Media, Inc.", 2013.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GD98] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)âa review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[ICV] 3d articulated hand pose estimation with single depth images. `https://labicvl.github.io/hand.html`. Accessed: 2019-01-13.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[TJCTK14] Danhang Tang, Hyung Jin Chang, Alykhan Tejani, and Tae-Kyun Kim. Latent regression forest: Structured estimation of 3d articulated hand posture. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3786–3793, 2014.

License