Master thesis

Temporal Interpolation of human point clouds using neural networks and body part segmentation

Author: Ignacio Reimat Corbella

Supervisors: Irene Viola, Pablo Cesar

Tutor: Antonio Chica

A thesis submitted in fulfillment of the requirements for the Master in Innovation and Research in Informatics – Computer Graphics and Virtual Reality from:

Research conducted at Centrum Wiskunde & Informatica in Amsterdam

Abstract

In the context of social VR, one of the media formats that is gaining popularity is that of a point cloud. Point clouds are unstructured volumetric representations of individual points that represent a 3D shape. They are easy to render but are voluminous in size, and thus they require high bandwidth to be transmitted, so concessions have to be made either in spatial or temporal resolution.

In this thesis we explore the state-of-the-art solutions for temporal interpolation of dynamic point clouds, with a focus on human bodies. We see that the current solutions work well predicting rigid motions but not deformations, which is the case of the human bodies. We hypothesize that the performance of these architectures can be boosted by segmenting the body in different body parts and predicting the interpolation for each body part individually.

Due to the lack of dynamic human point clouds datasets, we generate our own point cloud dataset based on a publicly available image dataset, being that the first contribution of this thesis. It consists of a total of 248.080 point cloud frames representing 40 avatars (20 males and 20 females) performing 70 actions each.

We adapt a current state of the art neural network architecture to fit our data, changing the loss function, tuning some parameters from its feature's extraction layers, and adding an extra layer to obtain the desired output. We obtain an architecture capable of performing temporal interpolation, which is the second contribution of this thesis.

We design a set of experiments in order to validate our hypothesis. These consist on a series of models trained to interpolate individual body parts, and one model trained to interpolate the full body. We observe performance gains in all the models trained with individual body parts, so we conclude with the hypothesis that applying body part segmentation and predicting the interpolation of individual body parts can improve the accuracy of point cloud temporal interpolation systems.

Table of Contents

List of Figures

List of tables

List of equations

1. Introduction

In the current technology revolution, Cross Reality (XR) Technologies (term that includes Augmented Reality, Virtual Reality and Mixed Reality) are gaining popularity. Because of this, new software, hardware (head-mounted-displays, sensors, 3D scanners, 360 cameras…), and media formats are constantly appearing to satisfy the demand of new immersive contents.

Social VR is the term used to designate the concept of getting together in a simulated world using Virtual Reality. Some examples are Facebook Horizon [1], vTime [2] and Mozilla hubs [3]. Figure 1 shows visual examples of these applications.

Figure 1. Screenshots from Facebook Spaces (left) and vTime(right)

In most of these systems, participants are represented as avatars, virtual identities chosen by the user to represent himself. One of the benefits of these representations is that you only need to send the model once to the other users, reducing the amount of data to transmit for each frame. The model is sent at the beginning of the connection, and later the server just needs to transmit the movement information or the new positions of the joints of the skeleton.

Other applications instead want to use a photorealistic representation of the user in real-time. This can be achieved with 3D scanners or a combination of depth cameras. This is the case of the H2020 European project VRTogether [4] that aims to develop an end-to-end system for the production and delivery of photorealistic social immersive virtual reality experiences. The goal of the project is to improve the immersive experience by innovating in how media formats are used (i.e., how audio, video and graphics are captured, delivered and rendered at users' homes) demonstrating a significant improvement of the feeling of being there together and the photorealistic quality of the content. The system combines different types of media (conventional 2D video with chroma, 360 video, Computer-Generated Imagery (CGI) objects, Time Variant Meshes (TVM) and dynamic point clouds) in a shared virtual space.

A recent output from this project is the paper from J. Jansen et al. [5], where they present the design and development of an architecture intended for volumetric videoconferencing. The system provides a highly realistic 3D representation of the participants, and it is based on point clouds.

Figure 2. Scenes from VRTogether [4]. From left to right: recorded 2D video presenter using chroma inserted in a CGI scenario, point cloud self-representation and point cloud representation of the partner.

One of the media formats that is becoming popular for volumetric representation is the point cloud. A point cloud is a collection of individual points that represent a 3D shape. Each point has at least a set of {x,y,z} coordinates denoting its position, and can also contain additional attributes such as color {R,G,B}, intensity or normal vectors. Unlike meshes, where points are grouped in polygonal faces, point clouds are sets of unordered and unrelated points. That means that no 3D reconstruction is needed to be rendered, making them suitable for real-time delivery systems.

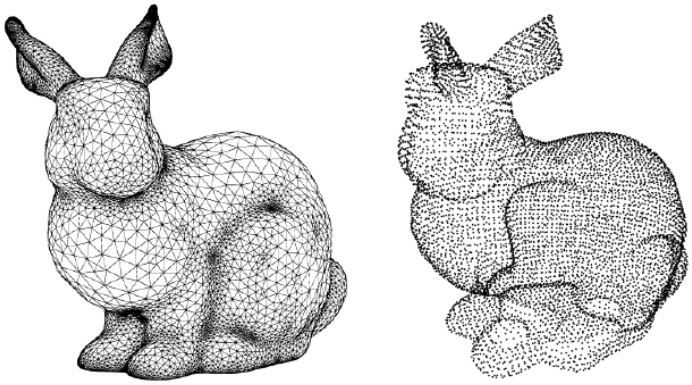


Figure 3. Comparison between a polygon mesh (left) and point cloud (right)

To use point clouds in social VR applications, it is necessary to transmit big amounts of data through the network, so there is a need of reducing the bandwidth. That can be accomplished in multiple ways: applying compression algorithms, using spatial interpolation (downsampling), or temporal interpolation (dropping frames and predicting them in the client side). So, the main research question of this master thesis is going to be focus on the latter.

Research question*: How to perform temporal interpolation of dynamic point clouds?*

Another technology which is becoming popular is Artificial Intelligence (AI), a term that has multiple definitions, as it currently covers a wide variety of subfields. Ranging from general areas as learning and perception, to more specific ones such as chess, mathematical theorems, poetry writing and disease diagnosis. AI synthesizes and automates intellectual tasks, and is, therefore, potentially relevant to any field of human intellectual activity.

A subfield of AI that is attracting a special interest is Machine Learning, a set of techniques and algorithms that have the ability to acquire their own knowledge, by extracting patterns from raw data. Deep Learning is a subgroup of Machine Learning that tries to simulate the behavior of human neural networks. Some of its best-known architectures are deep neural networks (DNN), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). These architectures have been applied to multiple research fields such as computer vision, speech recognition, natural language processing, audio recognition, social network filtering, bioinformatics, and medical image analysis.

Machine learning algorithms, and especially CNN have been applied in the field of video coding and have become the current state of the art in temporal interpolation in 2D video. Therefore, exploring how CNNs can be used to achieve temporal interpolation of point cloud contents might lead to promising results. An example of that is Flownet3D [6]. This network architecture is capable to estimate the scene flow from a pair of input point clouds. Scene flow is the same concept to the optical flow on 2D video but adapted to the 3D space.

The paper Temporal Interpolation of Dynamic Digital Humans using Convolutional Neural Networks [7] presents a novel architecture for temporal interpolation of point clouds using CNN's. It takes as input two point clouds and computes the scene flow between them, and uses the scene flow to predict the point cloud in between. The results are promising, improving accuracy over current state-of-the-art algorithms, but it can be improved. We work with the assumption that local deformation of human bodies is more difficult to predict than rigid transformations (translations, rotations) of objects. That is why we hypothesize that applying a similar architecture as the one presented on this paper, independently, to each part of the body, can improve the results. One of our research sub questions is thus:

> **Sub question:** *How pose estimation/segmentation can be used to improve temporal interpolation of human bodies?*

One of the problems nowadays is the lack of point cloud datasets, and specifically dynamic point clouds datasets with segmentation annotations. This is because the process of manually annotating 3D data, is complicated and time consuming. To solve this issue, we decide to use the 3D People dataset [8]. This synthetic image dataset contains 2 million images of 40 males and 40 females performing 70 actions, captured from 4 cameras. The information provided is: the camera matrices, the rgb frames, depth maps, normals, 3D skeleton joints, body part and cloth segmentation and optical flow. Our idea is to use the 3D people images dataset to generate a dataset of dynamic point clouds with body part segmentation annotations.

Figure 4. Some models from the 3D People dataset [8] performing different actions

By answering our research questions, the contributions of this thesis will be two: 1) a dynamic point clouds dataset; 2) an architecture for point cloud interpolation based on body part segmentation. In order to prove our results, we are going to use existing point cloud error metrics. We decided to train the network using the dropped point cloud frames (the one that has to be predicted) as the ground truth and using the point to point distance metric as loss function.

We designed a set of experiments in order to validate our hypothesis. These consist on a series of models trained to interpolate individual body parts, and one model trained to interpolate the full body. We observe performance gains in all the models trained with individual body parts. We also noticed a drop in performance for the interpolation of the hands, which is not surprising, because the hand itself has several articulations, and therefore, it does not behave as a rigid body. The extended experiments and results exposed in this thesis conclude that, the use of body part segmentation to solve the problem of temporal interpolation of dynamic point clouds, improves the accuracy of the interpolation systems.

In summary, in this thesis we focus on the temporal interpolation of point clouds representing humans, using neural networks. We study how body part segmentation can increase the accuracy of such algorithms. In order to do so, we generate a dataset of dynamic point clouds with body part segmentation annotations, and we use it to train our neural networks and perform some experiments to validate our hypothesis.

The thesis follows the following structure: In chapter 2, 2. Related Work, we provide some background about video interpolation, learning on point clouds, scene flow and temporal interpolation of point clouds. On chapter 3. Datasets, we provide an overview of the existing datasets and go through the process of generating our own dynamic point clouds dataset. On chapter 4. System Architecture, we describe the different modules that compose the selected architecture. On chapter 5. Evaluation and Results, we report the experiments we have carried out in order to validate our hypothesis we discuss the obtained results. Finally, in chapter 6. Conclusions and Future Work, we summarize the full process of completing the

thesis, discussing our system limitations, problems encountered, lessons learned, and highlighting the remaining challenges and the future work.

2. Related Work

In this chapter we provide a brief review of the related work that motivated this master thesis. We will start in Section 2.1 Temporal interpolation on video, explaining current state of the art of temporal interpolation on video (2D), to have a baseline in order to jump to the temporal interpolation in 3D data. In Section 2.2 Learning on point clouds, we will review different ways to use deep learning techniques to learn on point clouds. Some of them converting the point cloud to an intermediate format, and others directly learning on point clouds. In Section 2.3 Scene flow and temporal interpolation of point clouds, we will review Flownet3D, a neural network that predicts the scene flow between to input point clouds, and the paper [7], a first approach to perform interpolation in point clouds using CNN. Finally, in Section 2.4 Segmentation and Pose estimation, we will see some techniques to segment and extract pose estimation from point clouds.

2.1 Temporal interpolation on video

Temporal interpolation has been successfully used in video coding to increase the frame rate of video sequences. This technique works by estimating the optical flow between input frames and synthesizing intermediate frames based on this optical flow. Optical flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene. It describes a sparse or dense vector field, where a displacement vector is assigned to a certain pixel position, that points to where that pixel can be found in another image.

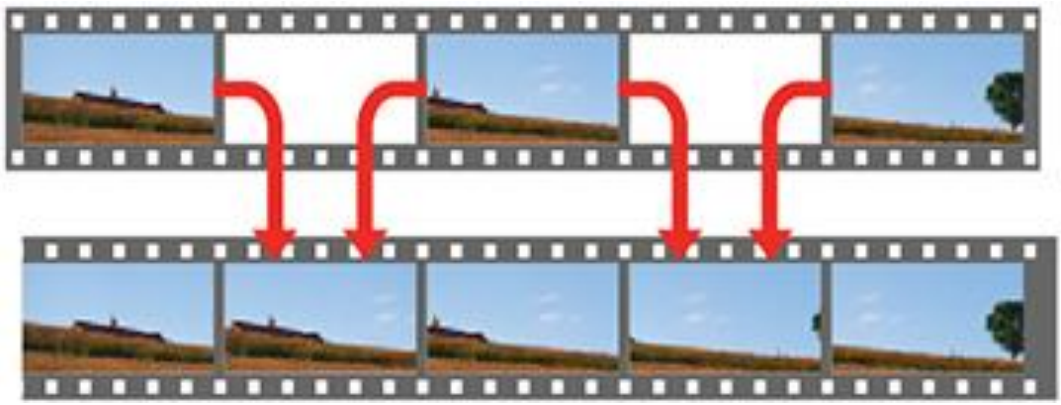


Figure 5. Example of temporal interpolation of frames in a video sequence [9]

Over the last years with the increasing interest in artificial intelligence, neural network-based approaches have been introduced also in this field [10], [11], and now represent the state-of-the-art in video frame interpolation [12].

2.2 Learning on point clouds

The idea would be to apply the same concepts used in 2D video, for temporal interpolation on point clouds. However, traditional neural network approaches, like convolutional neural networks, are not directly applicable to this media format, given that point clouds do not follow a regular grid structure. Also, 3D convolutional neural networks are expensive. We should then focus in other strategies to deal with learning on point clouds. The most well-

known are view-based methods, volumetric methods and direct learning on point clouds. We are going to briefly explain them in the next subsections.

2.2.1 View-based methods

These methods work by transforming the irregular structure into an intermediate format where traditional learning techniques can be applied. An example of these methods is the one that generates a 2D view of the point cloud by projecting the point cloud onto a surface and then performs the processing based on 2D images.

Multi-View CNN (MVCNN) [13] is an example of these methods. MVCNN system takes as input a point cloud and performs 2D captures of the point cloud from 20 different viewpoints. For each view four rotations are taken (0, 90, 180 and 270 degrees). The 80 images are passed through a special CNN that merges all the information in a global shape descriptor to predict the final classification of the object.

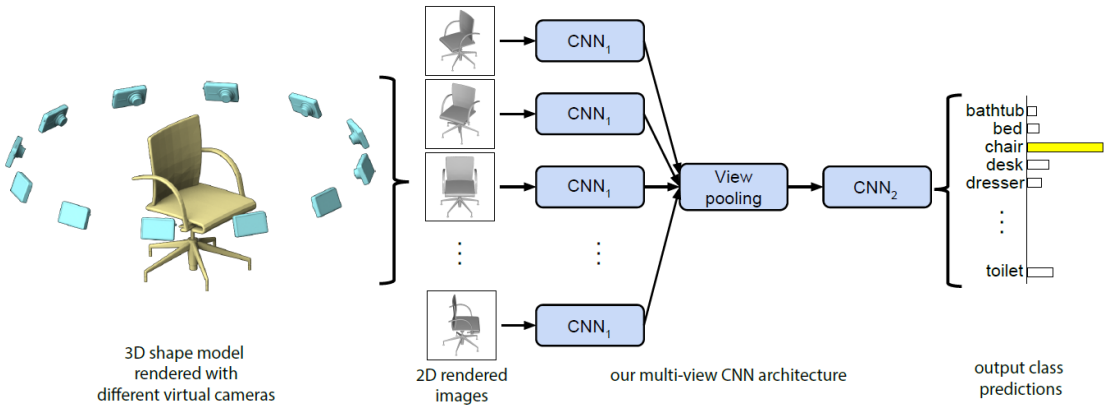


Figure 6. Visual example of view-based methods for classification of 3D objects [13]

An advantage of view-based methods is that 2D learning techniques can be used, but it is still a challenge to decide the number of views to generate, and some viewpoints may have difficulties dealing with occlusions or ambiguities. In the context of point cloud temporal interpolation, the output of the network must be converted again into a point cloud, which adds extra computation and complexity. For this reason, we decide against using view-based methods in this thesis.

2.2.2 Volumetric methods

Similar to view-based methods, volumetric methods transform the point clouds to an intermediate format to apply 2D learning techniques. In this case the intermediate format is a volumetric representation, like occupancy grids (or voxels) [14], [15] and KD-trees [16].

The advantages and shortcomings of volumetric methods are then similar to the view-based methods. Transforming the point cloud to an intermediate format allows applying 2D techniques for learning on the data, but the output needs to be converted back to a point cloud if we want to use it in an interpolation context. Additionally, volumetric methods resolution tends to scale poorly, so in the task of temporal interpolation, where fine-grained

motions play an important role, it can cause some troubles. Given the mentioned shortcomings, we choose not to use this kind of methods.

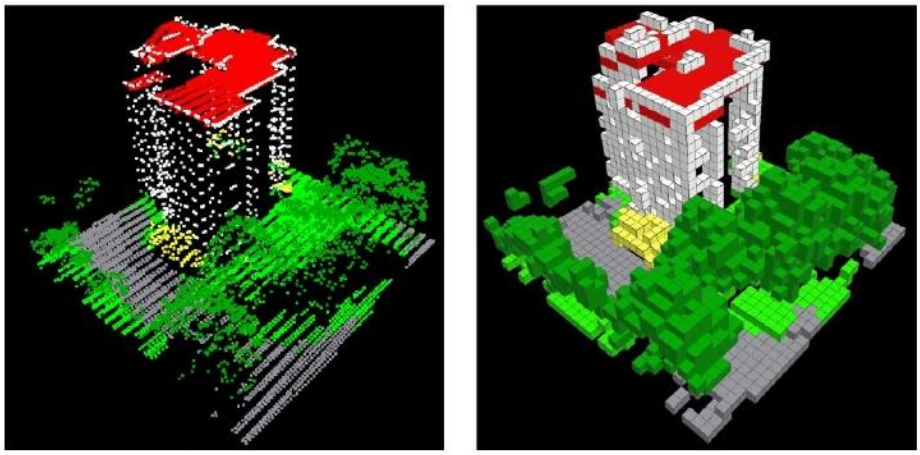


Figure 7. Example of conversion from a point cloud (left) into a voxelized version (right) [17]

2.2.3 Direct learning on points

Even though transforming the point cloud to an intermediate format sometimes results into good results, the process of conversion can be expensive and may cause the loss of essential information. That is the reason why other methods try to avoid it, and focused on learning directly on unstructured and irregular point cloud data. But this type of data presents some difficulties: a point cloud is an unordered set of points, so the network needs to be invariant to any permutation. At the same time, the relationships between a point and its neighbors are meaningful, so that also needs to be taken into account by the network.

PointNet [18] network learns to summarize an input point cloud by a sparse set of key points, which roughly corresponds to the skeleton of the objects. There is a base architecture for classification, and modified/extended versions for 3D object part segmentation and semantic segmentation in scenes. Examples of the output of each version are shown in Figure 9. The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features (as point segmentation requires a combination of local and global knowledge) and outputs per point scores.

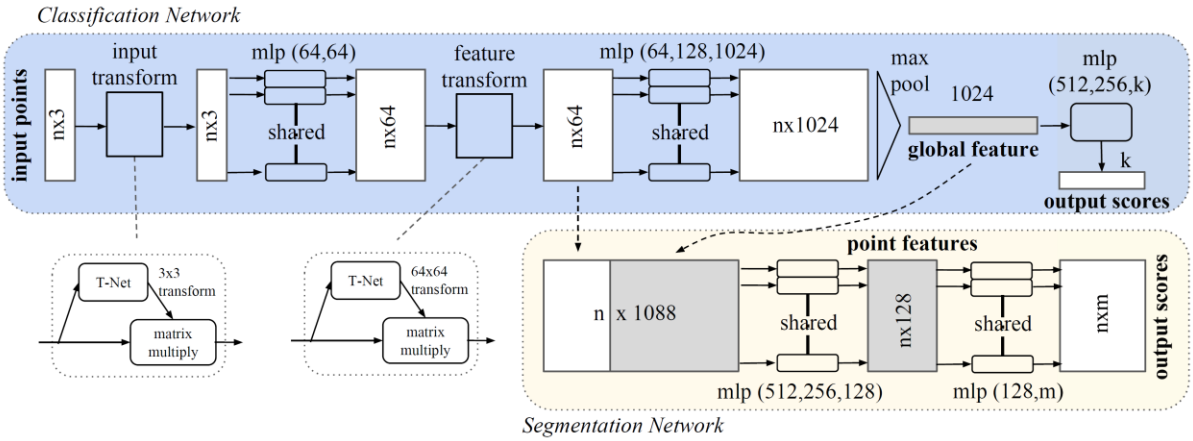


Figure 8. PointNet architecture [18]. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

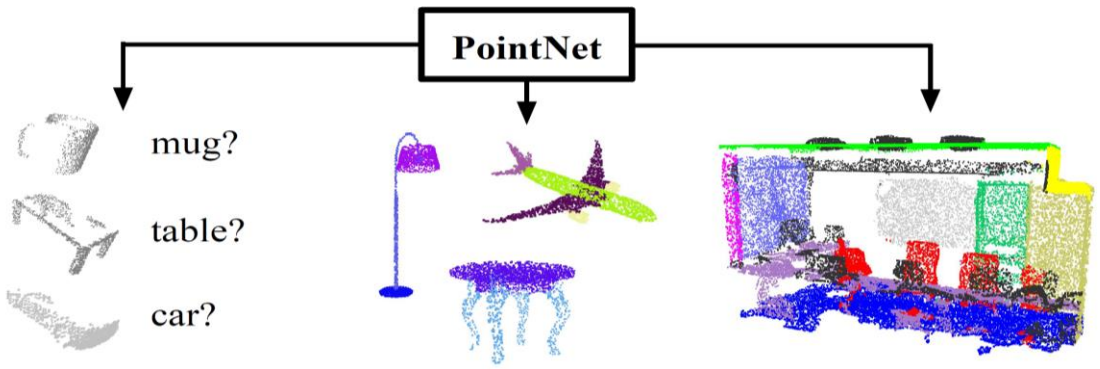


Figure 9. Application of PointNet [18]. Classification (left), part segmentation (middle), semantic segmentation (right

PointNet does not capture the local structure induced by the metric space, limiting its ability to recognize fine-grained patterns. That is something that tries to solve its successor PointNet++ [19]. PointNet++ leverages local neighborhood structure by first partitioning the input sets into overlapping local regions, extracting local features capturing fine geometric structures from the small neighborhood. This process is repeated hierarchically to generate increasingly high-level features. Although PointNet and PointNet++ achieved state-of-the-art result at that time, points in local regions are still processed independently, meaning it does not consider relationship between points.

2.3 Scene flow and temporal interpolation of point clouds

We have seen some neural network architectures that work by extracting high-level features in order to perform tasks of classification and segmentation. There are other network architectures focused on finding relationships between points. In this section we present some examples.

FlowNet3D [13] uses a neural network architecture to estimate the scene flow from a pair of input point clouds. Scene flow is the same concept to the optical flow on 2D video explained in Section 2.1, but led to the 3D space.

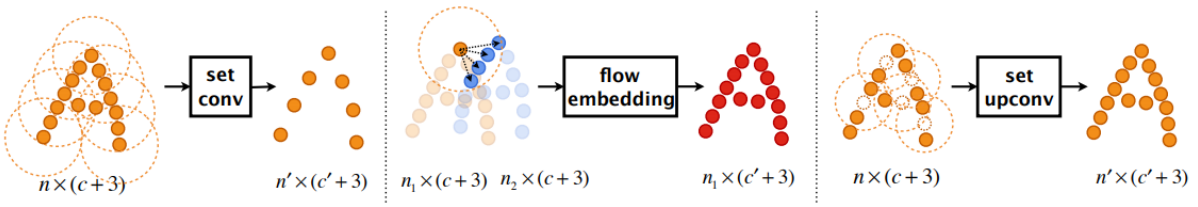


Figure 10. The three trainable layers from Flownet3D.

Flownet3D uses convolutional layers from PointNet++, called set conv layer, to learn deep hierarchical point cloud features. Then the flow embedding layer learns geometric relations between two point clouds to infer motions. Finally, the set upconv layers up-sample and propagate point features in a learnable way, giving as result the refined flow 3D.

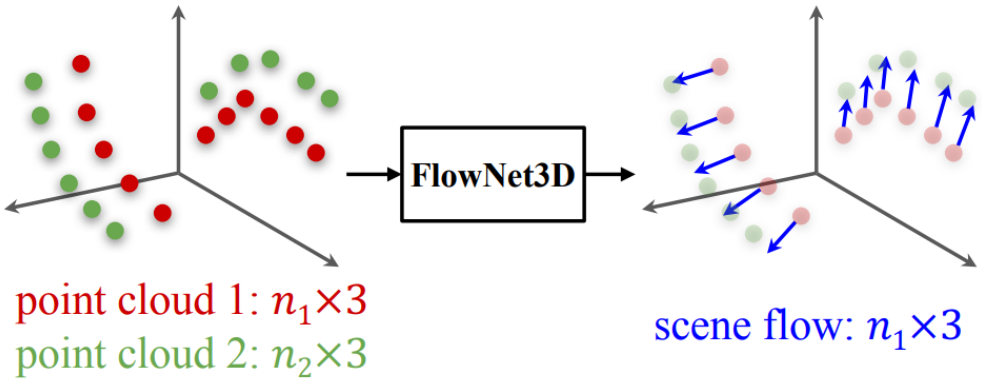


Figure 11. Scene flow estimation from point clouds (Flownet3D)

The network is trained with synthetic scene flow ground-truth data, which proves to transfer well to the real-world data. However, the model is only evaluated on data featuring rigid motions, thus not including deformations, which is the case of human bodies.

The paper Temporal Interpolation of Dynamic Digital Humans using Convolutional Neural Networks [7] is one first attempt for temporally interpolating on point clouds. It presents a proposal of a temporal interpolation architecture using convolutional neural networks. It is capable to increase the temporal resolution of dynamic digital humans represented using point clouds. The architecture achieves bandwidth savings by dropping frames in the sender side and recreating them in the receiving side.

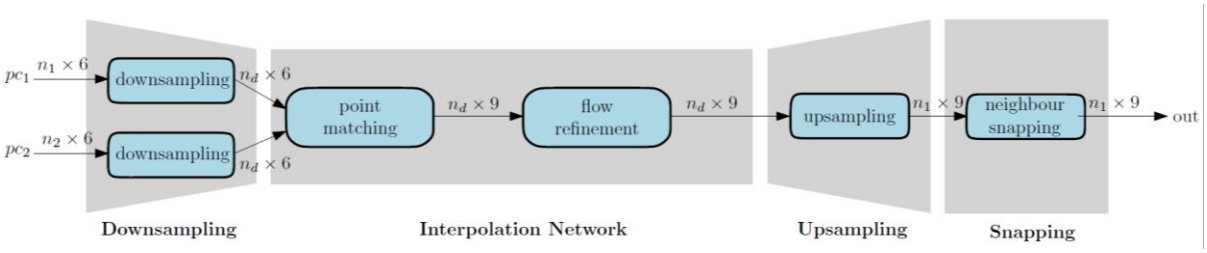


Figure 12. High-level structure of the temporal interpolation architecture

The network takes as input a pair of point clouds, which are downsampled and then the scene flow between is estimated. The scene flow is refined with a technique called neighbor snapping, that makes an existing scene flow estimation fully-connected in order to provide

more smoothness in the temporal interpolation (check Figure 13 for a better understanding of this technique). With this information, a new point cloud is computed among the two input point clouds.

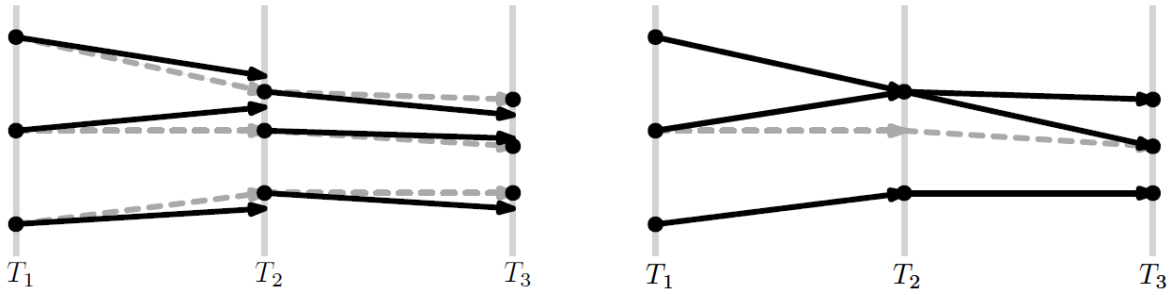


Figure 13. Neighbor snapping. These Figures show the position of a number of points (vertical axis) over time (horizontal axis). On the left we have a scene flow estimation that is not fully-connecting, while on the right side we have the same scene flow estimation after neighbor snapping has been applied.

The results were above the current state of the art but this kind of networks are shown to work better for predicting rigid transformations than deformations, which is the case of a dynamic human body. Our hypothesis is that those results could be further improved by segmenting the body in parts so each part movement is closer to a rigid transformation.

2.4 Segmentation and Pose estimation

Since we want to segment human body parts, Section 2.4.1 explores different methods for performing segmentation in point clouds. As our thesis is based on human bodies, in Section 2.4.2 we also explore 3D pose estimation methods. They can be used to implement segmentation based on the body joints.

2.4.1 Segmentation

Some tasks in geometry processing and analysis, such as segmentation and classification, require to find similarities between shapes. Similarity can be found by extracting feature descriptors of the geometrical structure. There are many papers proposing local feature descriptors for point clouds, to solve different problems. The paper *Recent Trends, Applications, and Perspectives in 3D Shape Similarity Assessment* [20] gives a good overview on point features and similarity assessment.

We already talked about PointNet and PointNet++, which extract point features and use them for classification, and its extension for segmentation. In the same line there is PointCNN [21], which uses k-nearest neighbor search and MLPs to learn a transformation, and uses that transformation in order to weigh the input features, and permute the input points into a latent and potentially canonical order. Once points have been transformed, convolution can be applied.

The paper DGCNN [22] aims at improving performance by also applying a conversion. In this case it converts point clouds into k-nearest neighbor graphs. They propose a module named EdgeConv, which operates directly over the point cloud and incorporates some important properties, such as local neighbor information. The module can easily be inserted into existing

architectures and has been proven to capture and exploit fine-grained and global properties of point clouds, giving good results both in classification and segmentation. Figure 14 shows its architecture. Figure 15 shows some visual results of the segmentation task.

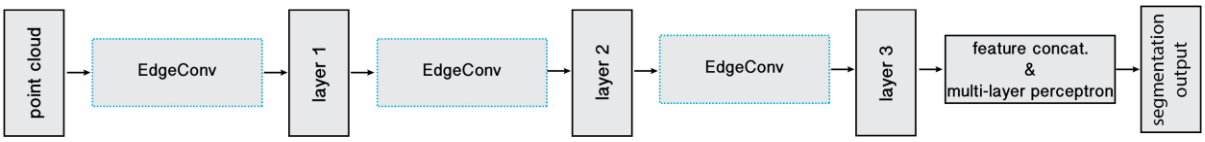


Figure 14. Schematic neural network architecture used in DGCNN

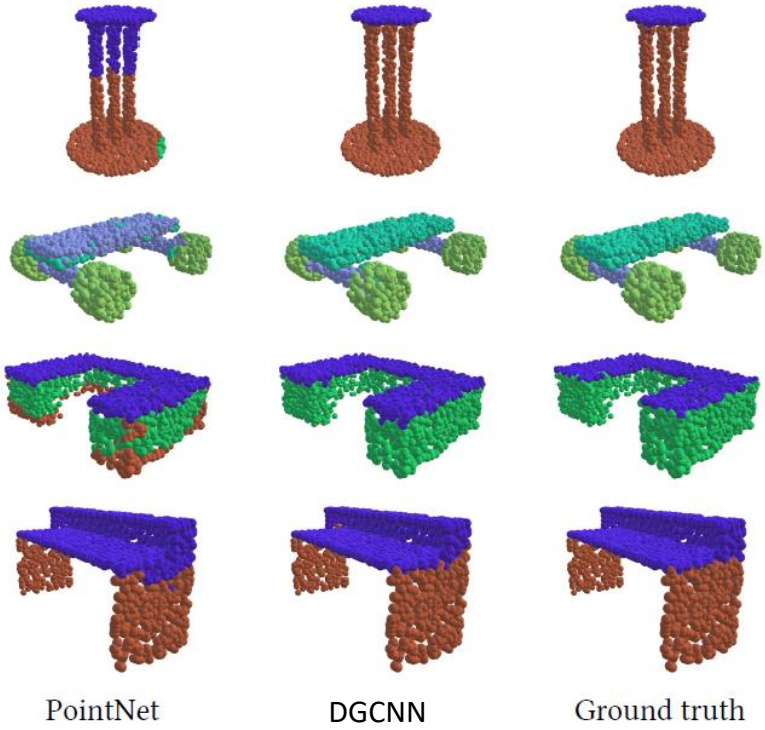


Figure 15. Comparison of segmentation results

2.4.2 Human 3D pose estimation

3D human pose estimation methods can be divided into two categories: discriminative and generative methods. Discriminative methods directly estimate 3D human pose from the input images. There are several CNN contributions based on discriminative methods used to predict 3D human pose [23], [24]. The paper from Marin-Jimenez *et al.* [25] uses a linear combination of the pose prototypes to estimate the 3D human pose from the depth. The accuracy of these methods could be poor due to extraordinary poses and occlusions.

Generative methods make use of the 2D information extracted from an input image to infer reasonable 3D poses [26]–[28]. The paper DeepPose proposes a method for human pose estimation based on Deep Neural Networks (DNNs). The pose estimation is formulated as a DNN-based regression problem towards body joints. The advantage of these methods is that they only require easily accessed annotations such as 2D joint locations. But at the same time,

it can become a drawback, because the performance of generative 3D human pose methods is largely affected by the accuracy of these 2D information extractions.

The paper *Weakly Supervised Adversarial Learning for 3D Human Pose Estimation from Point Clouds* [29] innovates by combining direct learning on point clouds using a Generative Adversarial Network (GAN), with a view-based method. So instead of using either 2D or 3D representations of depth images, they exploit both point clouds and the input depth image, combining discriminative and generative methods.

The network consists of two modules: the point clouds proposal module and the 3Dpose regression module. On the point clouds proposal module, they use 2D CNN to extract 2D human joints from the input depth image. They use the estimated pose to sample and normalize the extracted point clouds from depth. This process is visually described in Figure 16.

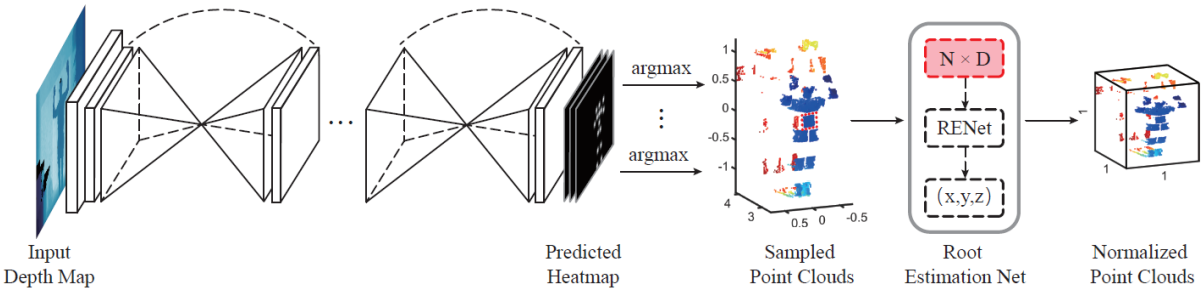


Figure 16. Illustration of the point clouds proposal model

On the 3D pose regression module, which is built upon PointNet, they use the initial 3D pose converted from the estimated 2D pose, and the normalized point clouds, to predict the final 3D human pose, using a GAN Network. The visual overview of the system is shown in Figure 17.

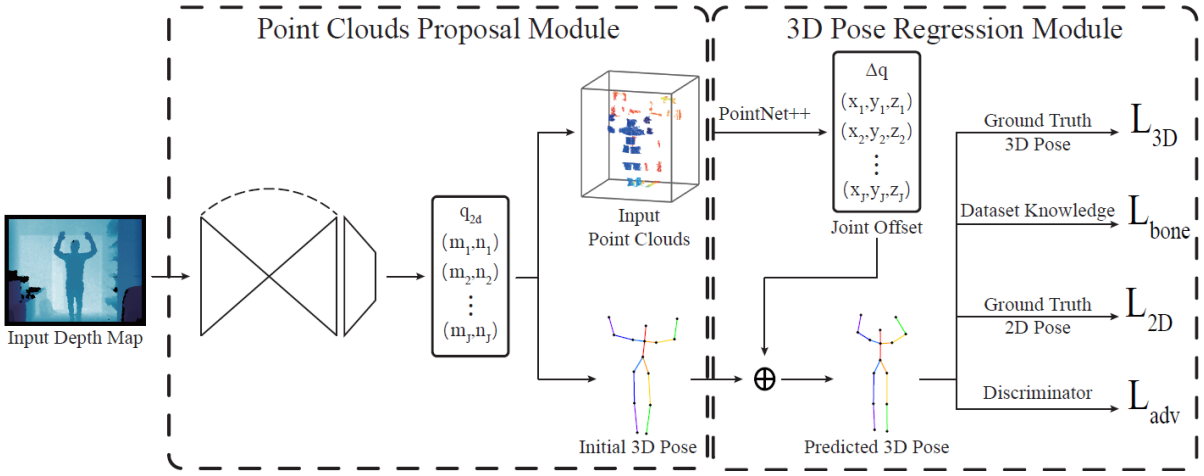


Figure 17. Overview of the 3D human pose estimation network

This thesis wants to prove first the effectiveness of using body-part segmentation in human point cloud interpolation, so if our hypothesis results in positive results, we would consider using pose estimation in our end-to-end system.

3. Datasets

In this chapter we start in Section 3.1 by providing an overview of the existing point clouds and image-based dataset focused in dynamic humans. We continue in Section 3.1 with an exhaustive vision of the 3Dpeople dataset on which we have based our point cloud dataset. Finally, in Section 3.3 we describe the process of generating our dataset while enumerating the encountered challenges and our proposed solutions.

3.1 Available datasets

One of the main problems when dealing with point cloud learning is the lack of point cloud datasets. This is due to the novelty of the format but also the fact that is more complex to generate annotations in 3D data [30]. It is true that new datasets are appearing, mostly thanks to the high interest arousing around the topic of self-driving cars.

In our case we need to find a dataset of dynamic point cloud sequences, if possible, with annotations about body part segmentation. Some of the datasets presented below are not point clouds but color and depth images. We will also consider them as they can be converted easily to point clouds.

3.1.1 Direct Point clouds

The Microsoft Voxelized Upper Bodies [31] data set contains five dynamic point cloud sequences of the upper bodies of human subjects. The sequences are captured using four frontal RGBD cameras, and as such only the front of each subject is visible. Since we were looking for full body data, we did not consider this one.

The 8iVFB v2 [32] dataset consists of four dynamic point cloud sequences. Each one captures a different human subject, at 30 FPS over a 10 second period, resulting in 1200 frames in total. The point clouds are captured from 42 RGB cameras capturing the subject from all directions. Due to post-processing, this data set is clean. The resolution is high with point clouds frames from 700.000 to 1 Million points. So, the quality of the data is great, but the quantity is scarce.

Table 1 summarizes the point cloud datasets that we considered to use.

Dataset	Characteristics
Microsoft Voxelized Upper Bodies	- 5 dynamic point cloud sequences - Captured using 4 frontal RGB cameras
8iVFB v2	- 4 dynamic point cloud sequences - 1200 frames (30 fps) = 10 seconds per action - Captured using 42 RGB cameras from multiple directions - Resolution ≈ 700.000 to 1 Million points

Table 1. Summary of the dynamic point cloud datasets

3.1.2 Image-based

The Standford EVAL [33] dataset consists on a set of depthmaps captured from a single point of view, representing 3 subjects performing 8 sequences each. It has annotations of the

estimated 3D joint positions. Since we want to reconstruct the full body, so one single point of view is not enough.

In the same line, we find the ITOP dataset [34] that consists in 100K real-world depth images taken from multiple camera viewpoints (front/side view and top) representing 20 people performing 15 actions. The data is labeled with real world 3D joint locations.

The Berkeley MHAD [35] is a dataset captured with five different capturing systems: optical motion, 4 multi-view stereo camera arrays, 2 Microsoft Kinects, 6 wireless accelerometers and 4 microphones. It represents 12 people performing 11 actions each.

Surreal [36] is a dataset with synthetically-generated but realistic images of people rendered from 3D sequences of human motion capture data. It consists of 6 million frames together with ground truth pose, depth maps, and segmentation masks. Since it only has one point of view, we discarded this one.

UBC3V [37] is a synthetic dataset representing 16 characters performing a total of 19.000 postures. The data has been captured by 3 randomly located virtual cameras for each pose. This is a big one, but is based on independent postures. As we are looking for sequences, we discarded this one.

In the end we decided to use the 3DPeople dataset [8], an image-based dataset of dynamic human sequences with body part and cloth annotations. In the next section we describe it in depth. The reasons to choose this dataset among others was of course the size of it, the available information (body part segmentation and optical flow), but also the ease of access to the data (everything comes in RGB images or .mat files). In other datasets for example it was necessary to download models and install software to generate the raw data.

Dataset	**Characteristics**
Standford EVAL	- depthmaps captured from a single point of view - 3 subjects performing 8 sequences each
ITOP	- 100K real-world depth images - Multiple camera viewpoints - 20 people performing 15 actions - 3d joints

Berkeley MHAD	- 12 people performing 11 actions each
	- 5 different capturing systems:
	  - optical motion
	  - 4 multi-view stereo camera arrays
	  - 2 Microsoft Kinects,
	  - 6 wireless accelerometers
	  - 4 microphones

Surreal	- 6 million frames - synthetically-generated - human motion capture data - ground truth poses - depth maps - segmentation masks

UBC3V	- synthetic - 16 characters - 19.000 postures - 3 randomly located virtual cameras for each pose
3DPeople	- 80 avatars performing 70 actions each - 4 camera views - Camera info - Rgb images - Depthmaps - Normal maps - Optical flow - Body and clothes segmentation - 3D joints

Table 2. Summary of the image-based datasets

3.2 The 3DPeople dataset

To generate our dataset, we are making use of the 3DPeople dataset. It aims to be the first dataset for computer vision research, of dressed humans with specific geometry representation for clothes. It contains around two million images of 80 avatars (40 males, 40 females) performing 70 actions each. Every subject-action sequence has been captured from 4 camera views, so for each camera it provides:

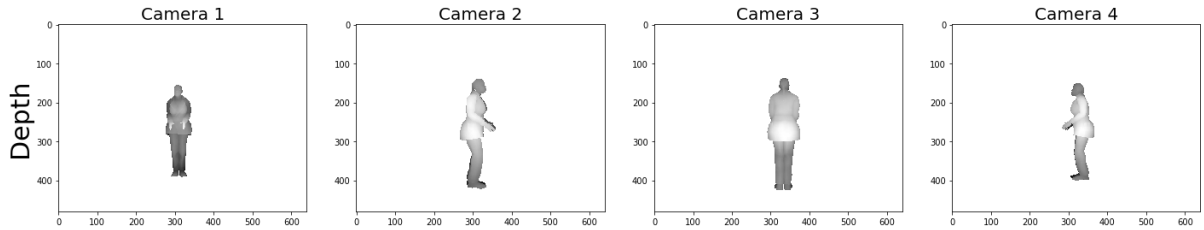
- Camera parameters: provides the intrinsic and extrinsic matrices.

```
Intrinsic matrix
700.000000 0.000000 320.000000
0.000000 700.000000 240.000000
0.000000 0.000000 1.000000

RT matrix
1.000000 0.000000 0.000000 -6.916235
0.000000 0.000000 -1.000000 11.055365
0.000000 1.000000 0.000000 53.095127

Camera projection matrix
700.000000 320.000031 0.000043 12149.077148
0.000000 240.000122 -700.000000 20481.585938
0.000000 1.000000 0.000000 53.095127

Blender params: location, rotation
6.916235 -53.095119 11.055357
90.000003 -0.000000 0.000000

Action bounding box: minx miny minz maxx maxy maxz
2.581615 -6.837263 -0.493293 11.250856 4.798149 22.604010
```

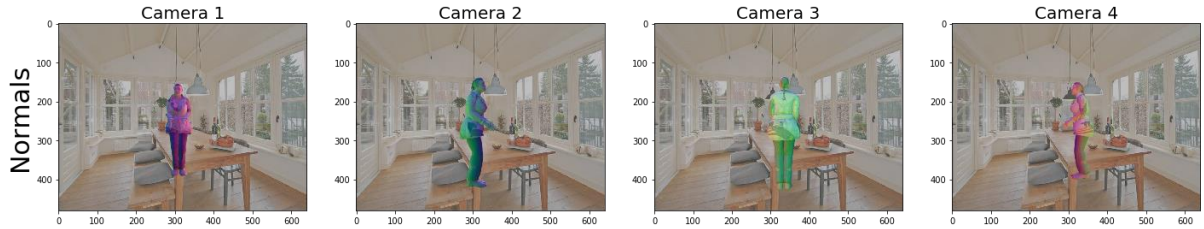
- RGB Images

- 3D skeleton joints

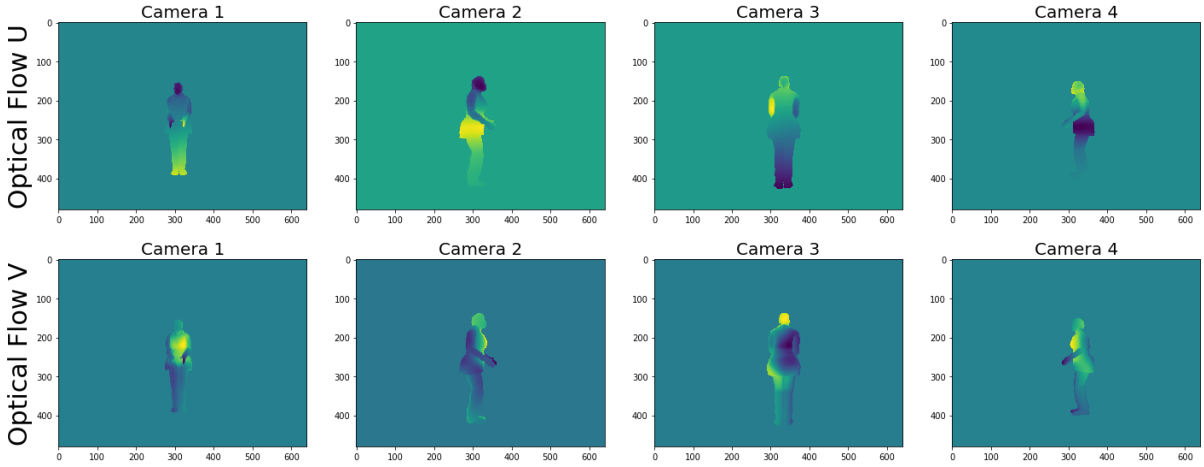
- Depthmaps



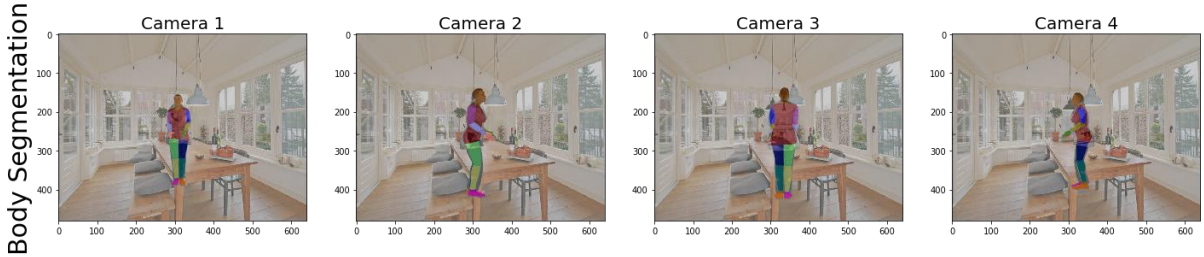
- Normals



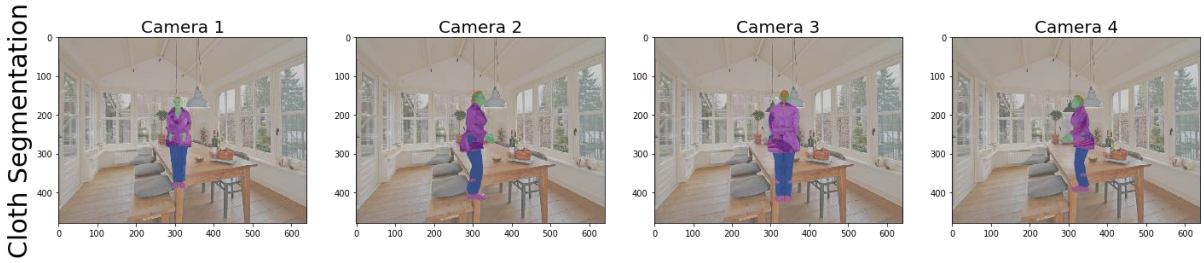
- Optical flow (u,v)



- Body segmentation



- Cloth segmentation



No 3D meshes nor raw files of the models are provided for copyright reasons, as the models used belong to Adobe. We will comment later some problems derived from it.

3.3 Generation of our dynamic point clouds dataset

For the building of the dataset we implemented some python scripts making use of the open3d library. We encountered some challenges, so they are described in this section, as well as the solutions developed for each of them.

3.3.1 Corrupted depthmap

The first step was to use the RGB and depth frames to reconstruct the 3D point cloud. Although the process is straightforward, since camera matrices are given, we found the alignment was not optimal as can be seen in Figure 18.

Figure 18. Reconstruction problems due to data corrupted in purpose to avoid copyright issues

The misalignment arises from the fact that the authors added noise to the data to avoid reconstructing the original meshes, for copyright reasons. We considered moving to another dataset, but they were not so complete, so we dedicated quite some effort to be able to use the available data. We noticed some sort of spherical distortion on the depthmap, but only affecting the depth component. So, we developed a correction process that consists in applying the opposite distortion to the depthmap. The process of generation of the correction matrix M can be seen in Figure 19.

The operation applied on the depthmap to obtain the corrected one is shown in Equation 1.

$$corrected\_depthmap = original\_depthmap - 4 \times M$$

Equation 1. Corrected depthmap function

```
M = np.ones((480,640),dtype=np.float32)
for i in range(480):
    for j in range(640):
        M[i][j] = (distance2D(i,j,240,320)/320)**2.
plt.imshow(M)
```

```
<matplotlib.image.AxesImage at 0x265938db520>
```

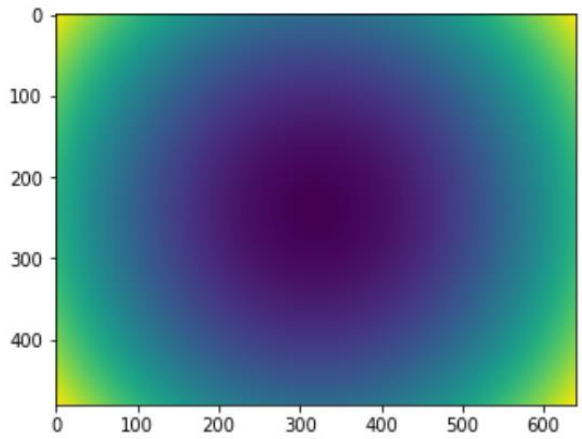


Figure 19. Code snippet of the generation of the correction matrix

After some trial and error, we managed to find a sufficient to go on with the thesis and its experiments. Part of the process of trial and error can be seen in Figure 20 where the color encodes the 4 tiles of the point cloud, each one as corresponding to each of the 4 cameras. This color encoding helped us to appreciate the spherical distortion on the original point clouds, and the improvement while applying different scaling factors of the correction matrix M.

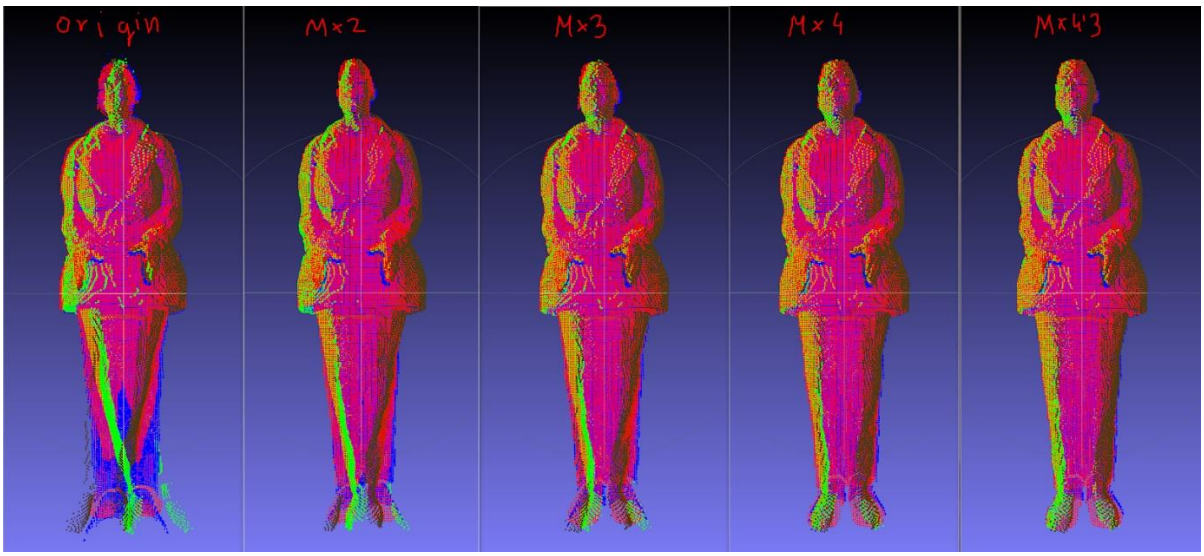


Figure 20. Original reconstruction (left) and the resulting points after applying n times the correction matrix. The color encodes the 4 tiles, one for each camera.

Figure 21 shows the reconstructed point clouds generated with the corrected depthmap.

Figure 21. Resulting point clouds reconstructed after applying the correction to the depthmap.

3.3.2 Correcting Misalignments

On the generation of the point cloud from the RGB and depth frame, some pixels of the background were assigned to the body, so the reconstructed point cloud had some noise. We improved this by applying erosion on the depthmap using a small kernel size.

The same process of combining the depthmap with a color frame was used to get the data of the body and cloth segmentation for each point. Similar problems occurred with the alignment. In this case the erosion on the depthmap was not enough, especially in the case of the body segmentation. This segmentation does not consider the clothing, resulting in lots of grey pixels from the background entering the point cloud.

Figure 22. 2d approach to fill gray pixels

Our first trial to solve this was the implementation of an algorithm that worked in the 2D space, by assigning the closest color to the gray pixels. Figure 22 visually describes the process. Even that in the 2D space it looked promising, the generated data in the 3D space was not optimal, as can be seen in Figure 23

Figure 23. Point cloud resulting of the reconstruction using the 2D approach to remove the background

Then we tried to apply the same idea but directly in the 3D space. By using a nearest neighbor search, we converted every gray point to the fashion of the points in a certain radius (increasing the radius size if only gray points where found). It was not enough, again due to some misalignment problems, we performed a 2nd step of the algorithm. In that case, for every point, the algorithm applies the fashion of the colors in a fixed radius. The results were promising as shown in Figure 24

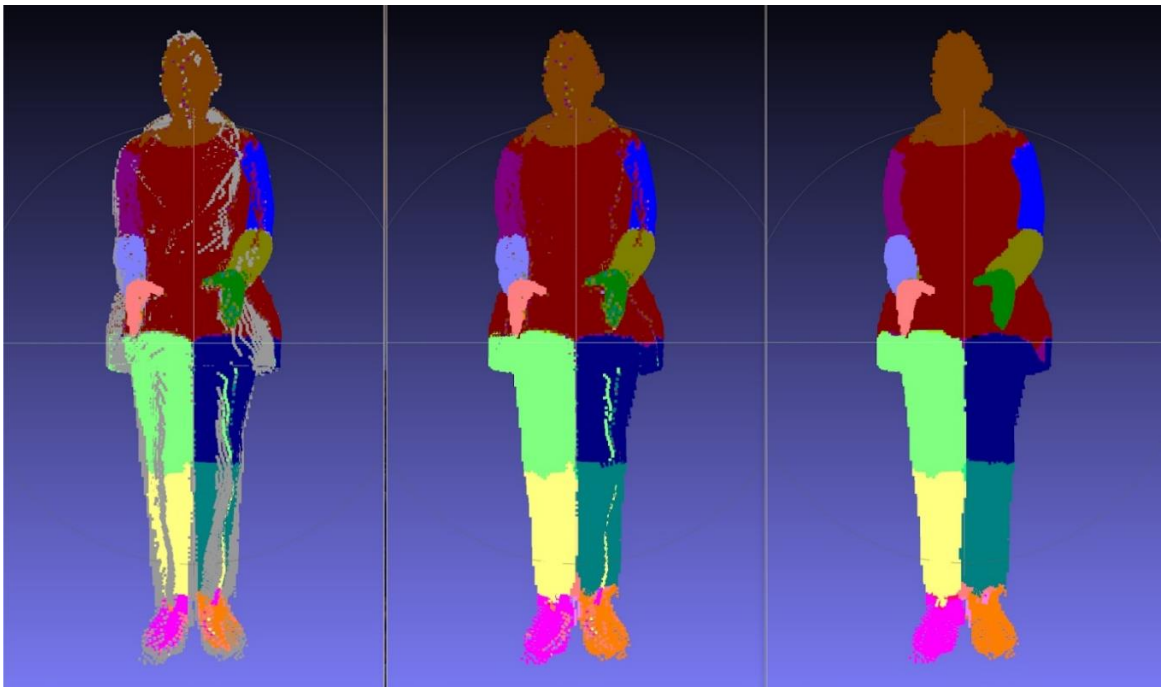


Figure 24. Point cloud of the body part segmentation reconstructed using the 3D approach in 2 steps

3.3.3 Optical flow

The optical flow data gives information about the movement of the pixels in the 2D plane. That means that the conversion to the 3D space is not straight forward. Some attempts were made in order to compute the depth component using the depthmaps, but the results were not good, mainly due to occlusions, and possibly because some corrupted data. So, we were not able to reconstruct the scene flow 3D from the optical flow.

3.3.4 Final point clouds

After all the process reported in this section, we achieved to get a dataset that could be used in our experiments. The final point clouds dataset consists in 6202 frames for each avatar, resulting in a total of 248.080 point cloud frames. They are stored in ply files following the next structure:

```
ply
format ascii 1.0
element vertex 57238
property float x
property float y
property float z
property float nx
property float ny
property float nz
property uchar red
property uchar green
property uchar blue
property uchar tile
property uchar bodysegm
property uchar clothsegm
end_header
```

Figure 25. Data structure of one of the output point clouds generated stored in a ply file

The image resolution of the 3D people dataset (640x480) is limiting the resolution of the output point clouds. Still, the number of points oscillates between 40.000 and 60.000 points.

The generated dataset is one of the contributions of this master thesis. It is a complete point cloud dataset and it will allow us to train the interpolation neural network, as well as perform multiple experiments related with point clouds, in the future.

The usage of this dataset goes beyond this thesis since it can be used for multiples experiments, such as tiling, compression and segmentation.

4. System Architecture

As the goal of the thesis was to prove our hypothesis (segmenting body parts and predicting the interpolation of each part gives better results), we decided to reuse the Flownet3D architecture but applying some modifications. In this section we discuss those modifications, while explaining the reason behind those decisions.

The system will take as input two raw point clouds P_1 and P_2 composed by its (x_n, y_n, z_n) position coordinates and its (r_n, g_n, b_n) color coordinates, and will output the temporal interpolated frame P_{pred}. Figure 26 shows a high-level diagram of our system architecture. In the next subsections we will describe every module.

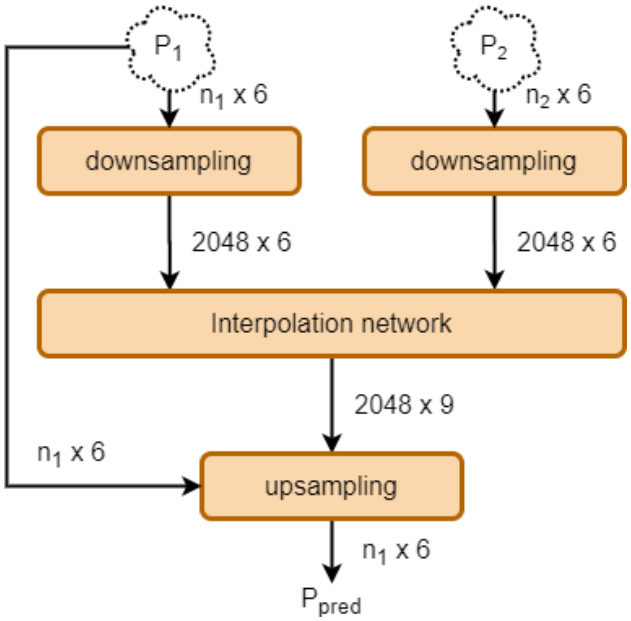


Figure 26. High-level architecture of our system

4.1 Downsampling

The first step is to downsample the raw point clouds P_1 and P_2 to a lower spatial resolution. This is necessary because neural networks require that all inputs have the same number of points, but also because computational time and memory requirements scales $O(n^2)$ and our resources are limited.

We considered different techniques for point cloud spatial downsample. Voxel downsampling seemed a good approach, because it gives more informative results, but requires more computation and is difficult to configure the output number of points, because some voxels may be empty. We ended using a random downsampling technique, fast and easy to configure. An example of the downsampling of the head using 2048 points is shown in Figure 27.

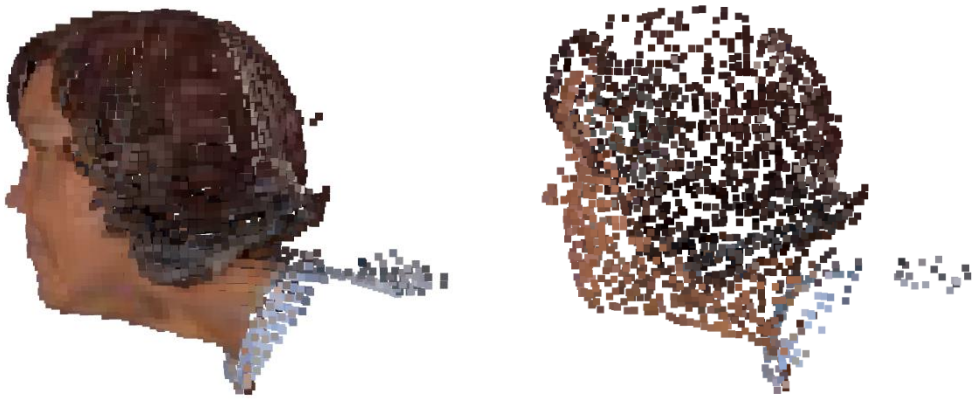


Figure 27. Example of downsample of the head. Left: raw point cloud. Right: downsampled point cloud to 2048 points

4.2 Interpolation Network

The systems we were considering to use as our interpolation network were Flownet3D and the solution from [7]. Both systems take as input two point clouds P_1 and P_2 and are able to predict the scene flow F between them. So, both systems are practically identical concerning inputs and outputs. Both systems also use scene flow as groundtruth to train their networks. We chose to work in Flownet3D given that it proved to work well in predicting rigid motions.

We observed the pretrained model from Flownet3D was not working well. Probably because it was trained to predict the movement of buildings and cars, and our point clouds are quite different in size and point density. There was a need to retrain the model. But as we already commented in Section 3.3.3 Optical flow, the fact that we were not able to obtain the scene flow in our dataset made us consider other options for training the network. We decided to train the network using the dropped frame P_d (the frame we want to predict) as our groundtruth data.

In order to achieve this, we had to change the original loss function, because it was comparing the error between the predicted flow vectors and its ground truth vectors. In our case, we need a loss function able to compare point cloud distortion. We decided to use as loss function the point to point error metric. In each batch step, we compare the predicted (x_p, y_p, z_p) coordinates with the groundtruth (x_g, y_g, z_g) data. For each point p_i in the predicted point cloud P_{pred} with N number of points, we find the nearest neighbor p_j in the groundtruth point cloud P_{gt}, and compute its distance. The loss then results as the mean squared error of those distances as shown in Equation 1.

$$loss(P_{pred}) = \frac{1}{N}\sum_{i=0}^{N}(p_j - p_i)^2$$

Equation 2. MSE used as loss function

We also needed to adjust some parameters of the network to fit our data. Specifically, we changed the radius sizes for the feature extraction and flow refinement layers to be coherent with our data scale.

Finally, we added a final layer in order to adjust the output to the new loss function. The original network was giving as output the 3D vectors representing the scene flow. So, we added a last layer that consists in a direct sum between the original (x_1, y_1, z_1) coordinates and ½ of the predicted flow vectors $\vec{f}(x, y, z)$. The ½ factor is because we want to predict the point cloud P_{pred} between P_1 and P_2, which is supposed to be approximately in a half way between P_1 and P_2. The resulting architecture is shown in Figure 28.

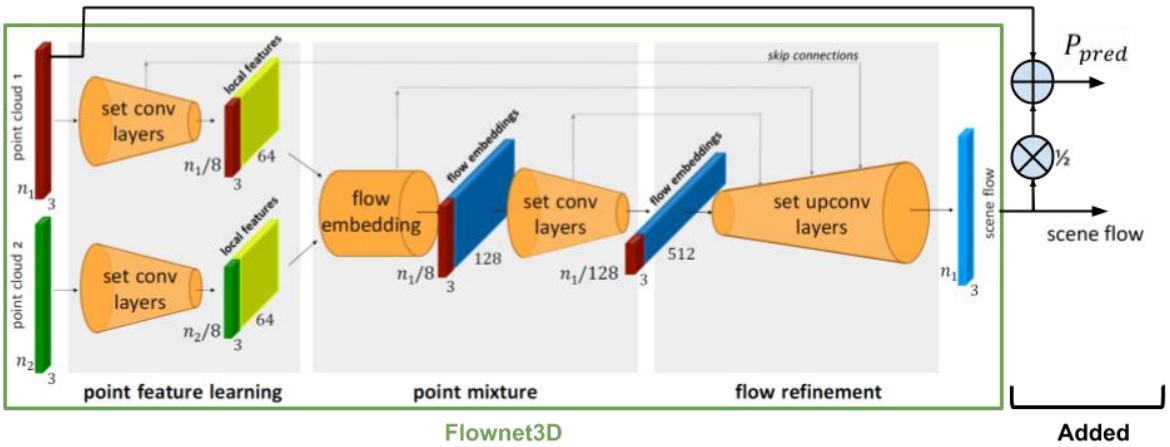


Figure 28. Network architecture consisting in a small modification of Flownet3D. Image adapted from [6]

Even that the output has been modified to match the datatype needed for computing the loss function, we are also saving the original predicted scene flow vectors. That will give us flexibility in the future in the case we want to predict more than one frame between P_1 and P_2.

4.3 Upsampling

The output of the network is the scene flow vectors corresponding to the downsampled input point clouds, so we need to upsample it to obtain the movement for each of the points in the raw point cloud. We do this using 3D interpolation [19]. Given the raw point cloud P_1 , its downsampled version P_{1_down} and the low resolution scene flow estimation F_{down} , we compute the upsampled scene flow estimation F_{up} as described in Equation 3.

$$\forall\, p \in P_1, \qquad \vec{f}_{up}(p) = \sum_{p_i \in knn(p,\ P_{1\_down},\ 10)} w(p, p_i) * \vec{f}_{down}(p_i)$$

Equation 3. Upsampling the predicted flow

Here *w* is a normalized inverse-distance weight function. In other words, for each point $p \in P_1$ we take a weighted average of the scene flow vectors of all points from P_{1_down} that are in the 10-nearest neighborhood of p, assigning higher weights to the closer points.

5. Evaluation and Results

In this chapter we present the results of the performed experiments in order to prove our hypothesis, showing visual results and objective metrics. On Section 5.1 we describe the system used to carry out the experiments and how the dataset was split for the different steps of the training. Section 5.2 explains each of the experiments in detail. Finally, Section 5.3 shows the performance metrics for the different blocks of the system.

5.1 System description

All the training processes have been performed in a computer with a processor Intel i9-9900K working at 3.6GHz, 16GB RAM and a NVIDIA RTX 2080Ti graphics card with 11GB GDDR6.

The evaluations have been performed in laptop machine due to the availability of the resources. Its characteristics are a processor Intel i7-7700HQ working at 2.8GHz, 16GB RAM and a NVIDIA GTX 1070 Max-Q graphics card with 8GB GDDR5.

For the training we have generated a small dataset of 1000 frame triplets. We chose 1000 due to time limitations. Each triplet is formed by three time-consecutive point clouds. P_t, P_{t+1} (the one we want to predict, so it will be used as groundtruth), and P_{t+2}. From these 1000 triplets, 800 are used for the training step, and 200 are used for the validation step.

We trained all our models 1000 epochs, as we considered this number of epochs good enough to achieve convergence, meaning that more epochs will rarely improve our results. We use an adaptive learning rate, starting at 0.0005 and decreasing in time. For the batch size we selected a size of 4 point clouds. The reason for not having a bigger batch size is memory limitations, because the loss computation requires a lot of available memory.

Another dataset of 200 frame triplets is used for the evaluation. Table 3 shows the distribution of avatars and actions used in each stage. We selected different avatars and actions for each stage. For a better understanding: for the train step we used 800 frame triplets selected randomly from the full dataset. They correspond to the avatars Man 01-04 and woman 01-04 performing actions 01-20.

Stage	Number of frames triplets	Avatars	Actions
Train	800	Man 01-04, woman 01-04	01-20
Validation	200	Man 05-06, woman 05-06	21-40
Evaluation	200	Man 11-12, woman 11-12	41-60

Table 3. Distribution of the data for each stage of the experiments

5.2 Experiments

We chose to do 5 experiments in order to prove our hypothesis and check the performance of the system. A brief description of the experiments is shown in Table 4.

#	Brief description of the experiment
1	Compare model trained with only heads, with the model trained with the full body (using same radius)

2	Compare model trained with only heads, with the model trained with the full body (using different radius)
3	Check performance with model trained with the right hand
4	Check performance with model trained with the right foot
5	Check performance with model trained with the chest

Table 4. Description of the experiments

In order to read the experiments properly, in Table 5 we give a reference of the mean distance between points in the downsampled point clouds. Note that, as expected, the bigger parts have a bigger distance between points.

	Number of points	Mean distance between points
Chest	2048	0.0118
Hands	1024	0.0069
Heads	2048	0.0078
Full body	2048	0.0196

Table 5. Number of points and mean distance between points in the downsampled point clouds

In order to deal with some frames with less points than the selected number of points, we allow repeating points in the randomized downsample, only for those cases.

5.2.1 Experiment 1

The first experiment consists in training two models. The first model M_{heads}, is trained using only heads, the second one M_{fb}, is trained using the full body. The hypothesis is that M_{heads} will give better results compared with the prediction of the head using M_{fb}. First, because the movement of individual body parts is more rigid, and second, because thanks to the segmentation, the downsampling factor is smaller.

The experiment has been performed following the guidelines explained in Section 5.1, using 2048 points in both models as shown in Table 5. In this experiment the radius used for the Features extraction and Feature propagation layers are the same for both models as shown in Table 6.

	Radii M_{heads}	Radii M_{fb}
R1	0.01	0.01
R2	0.02	0.02
R3	0.03	0.03
R4	0.04	0.04

Table 6. Radii used in experiment 1

Figure 29 and Figure 30 show the evolution of the loss for the training and validation steps. The model was saved on epoch 958 for M_{heads}, and in epoch 740 for M_{fb}, that corresponds to the minimum loss value obtained on the validation step. We see the validation loss goes down until $2e^{-4}$ for $\mathrm{M_{heads}}$ and $6.2e^{-4}$ for $\mathrm{M_{fb}}$, so we expect $\mathrm{M_{heads}}$ to perform better.

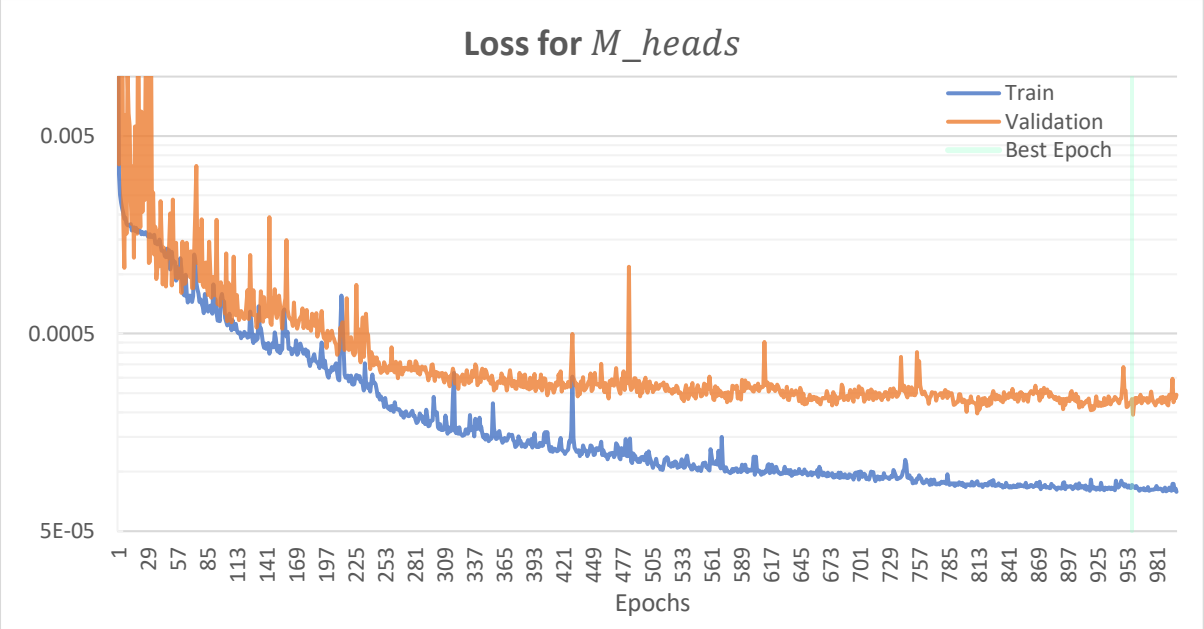


Figure 29. Logarithmic plot of the evolution of the loss for the model M_{heads}. The model was saved on epoch 958

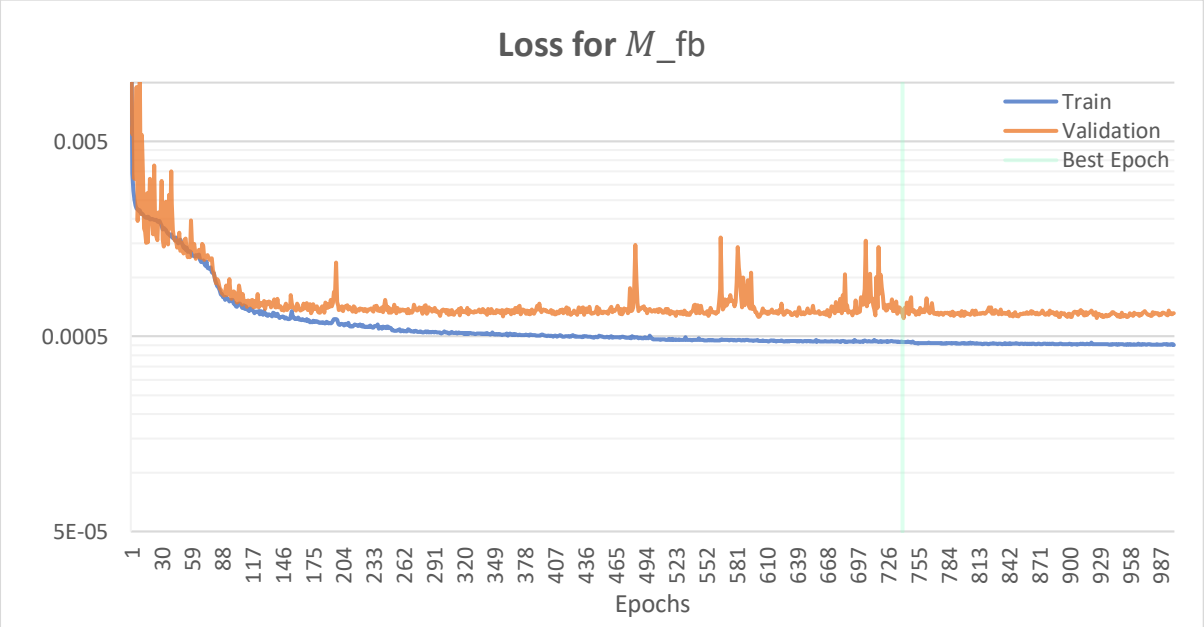


Figure 30. Logarithmic plot of the evolution of the loss for the model M_{fb}. The model was saved on epoch 740

The loss values obtained are coherent with the reference values presented in Table 5.

Next, we present the objective error metrics obtained with the evaluation set, corresponding to 200 point clouds. We have computed the point to point and point to plane metrics, using MSE and Hausdorff distance, between the ground truth point cloud and the predicted point cloud. The results are presented in Figure 31, Figure 32, Figure 33, Figure 34, and Figure 35.

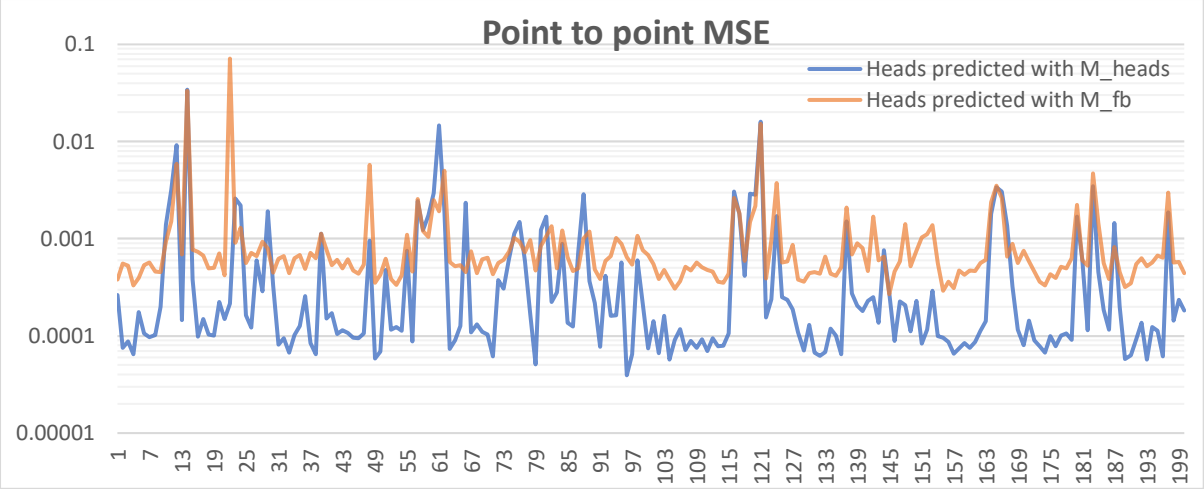


Figure 31. Point2Point MSE for both models in experiment 1

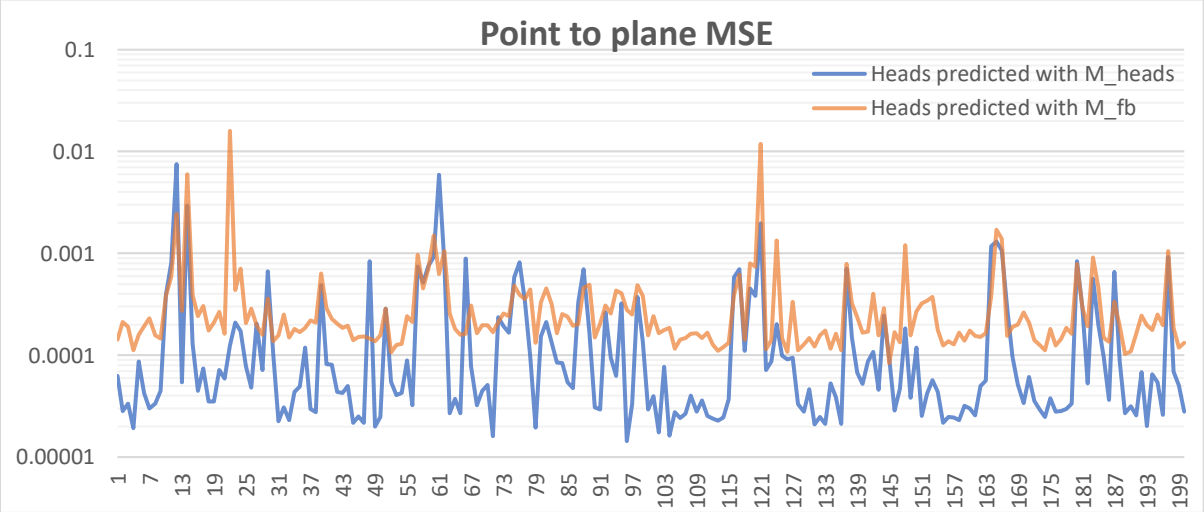


Figure 32. Point2Plane MSE for both models in experiment 1

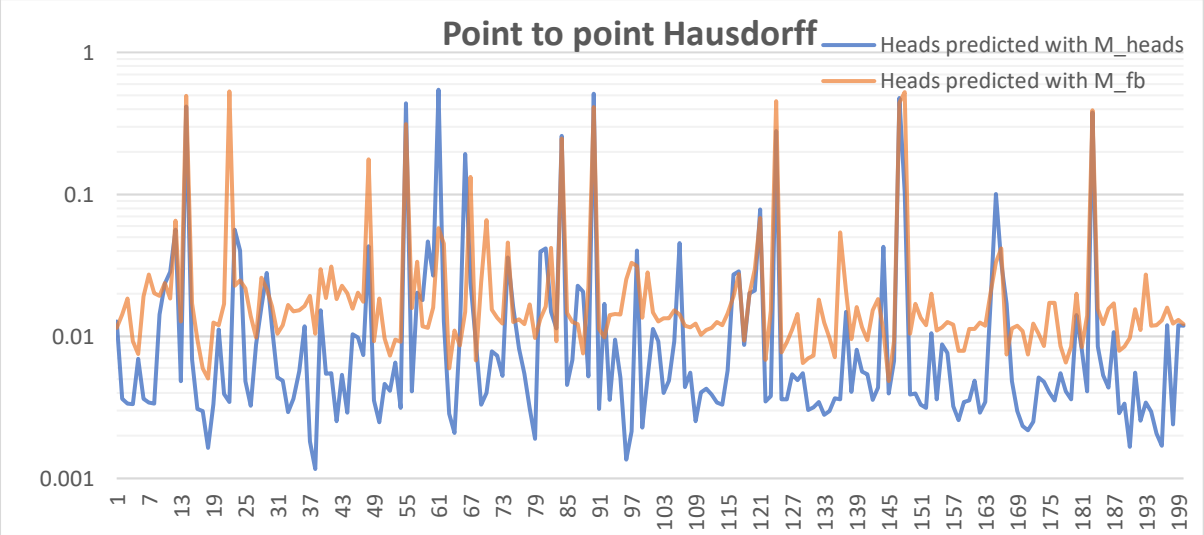


Figure 33. Point2Point Hausdorff distance for both models in experiment 1

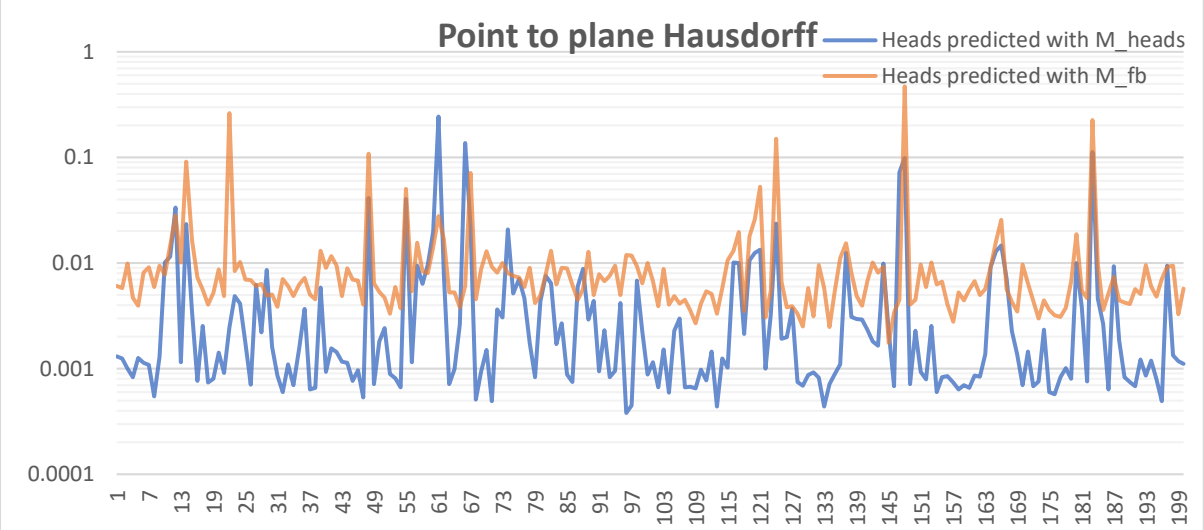


Figure 34. Point2Plane Hausdorff distance for both models in experiment 1

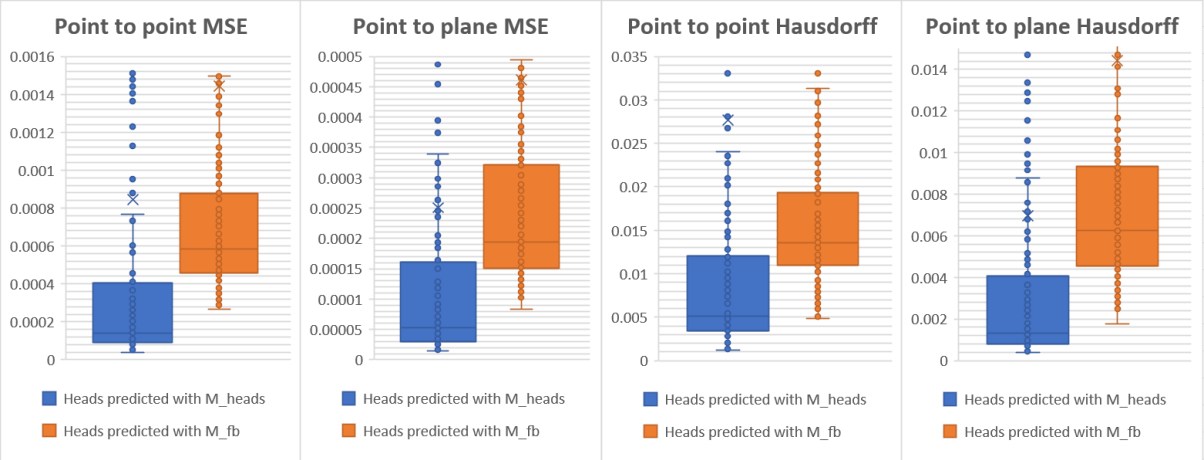


Figure 35. Box plot of the error metrics (zoomed in) in Experiment 1

A median value always smaller for the model M_{heads}, tells us that this model makes better predictions in most of the cases, except for some outliers. Because of those outliers, the plots in Figure 35 are zoomed in, otherwise the difference in the results would not be discernible.

A visual comparison in Figure 36 confirm those results, giving as clear winner the model M_{heads}, and therefore, confirming our hypothesis.

In the downsampled predicted point clouds from M_{fb} we have observed that the model seems to be learning the skeleton. This effect is shown in Figure 37. In order to identify possible causes of this behavior, we decided to perform Experiment 2.

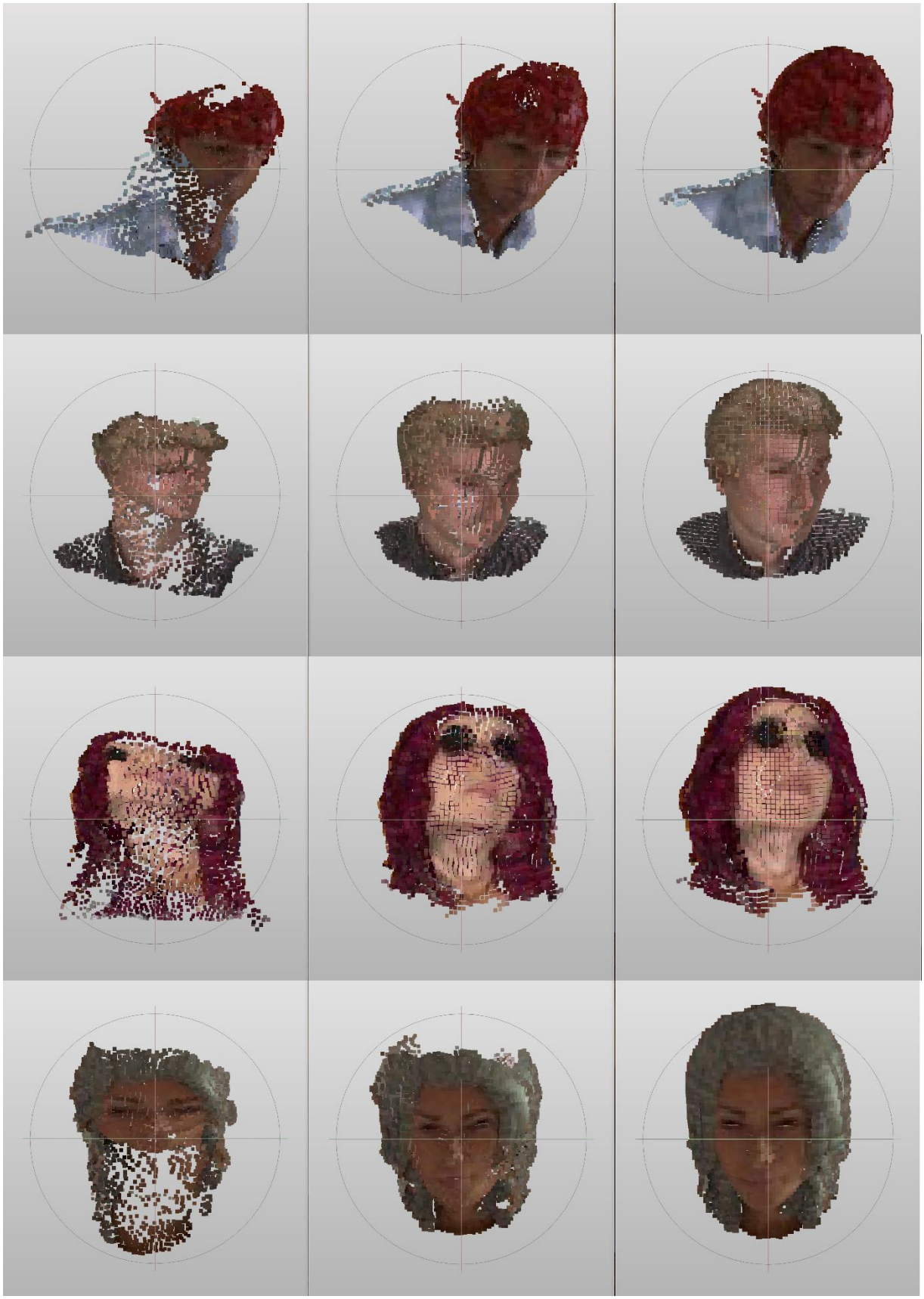


Figure 36. Visual comparison of the prediction of the heads using both models in Experiment 1. Left: using M_{fb}. Middle: using M_{heads} . Right: groundtruth

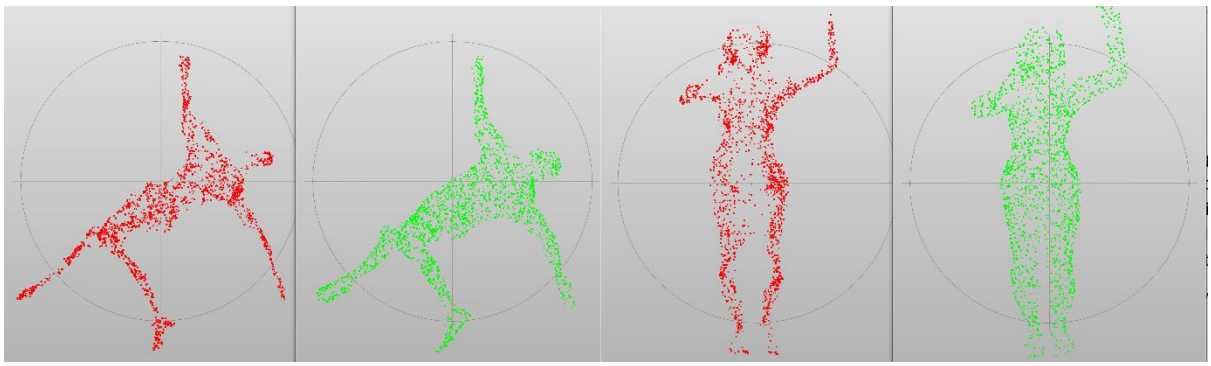


Figure 37. The model M_{fb} seems to be learning the skeleton of the body

5.2.2 Experiment 2

The second experiment wants to check the influence of using different radii for the Features extraction and Feature propagation layers, when training the model with full bodies. We decided to perform this experiment because the full body point clouds have a bigger size, and the points are more distant due to a bigger downsample. Also, because in Experiment 1 we observed that the model M_{fb} seems to be learning the skeleton of the point cloud.

For the experiment we are going to compare the model M_{fb} in Experiment 1 with a new model. The new model M_{fb_2} , trained with full bodies, also uses 2048 points. Its radii are shown in Table 7.

	Radii M_{fb}	**Radii M_{fb_2}**
R1	0.01	0.1
R2	0.02	0.2
R3	0.03	0.3
R4	0.04	0.4

Table 7. Radii used in experiment 2

Figure 38 shows the loss values obtained on the training and validation steps. We observe slightly better results in M_{fb_2} , with a minimum of the validation loss around $6e^{-4}$, in comparison with M_{fb} with a minimum of the validation loss around $6.2e^{-4}$, but it is not a big difference.

Figure 38. Logarithmic plot of the evolution of the loss for the model M_{fb_2}. The model was saved on epoch 779

The error metrics are presented in Figure 39, Figure 40, Figure 41, Figure 42 and Figure 43. In this case the values obtained are not conclusive. We observe a very similar performance in both models, being slightly better the model $\mathrm{M_{fb}}$ from experiment 1.

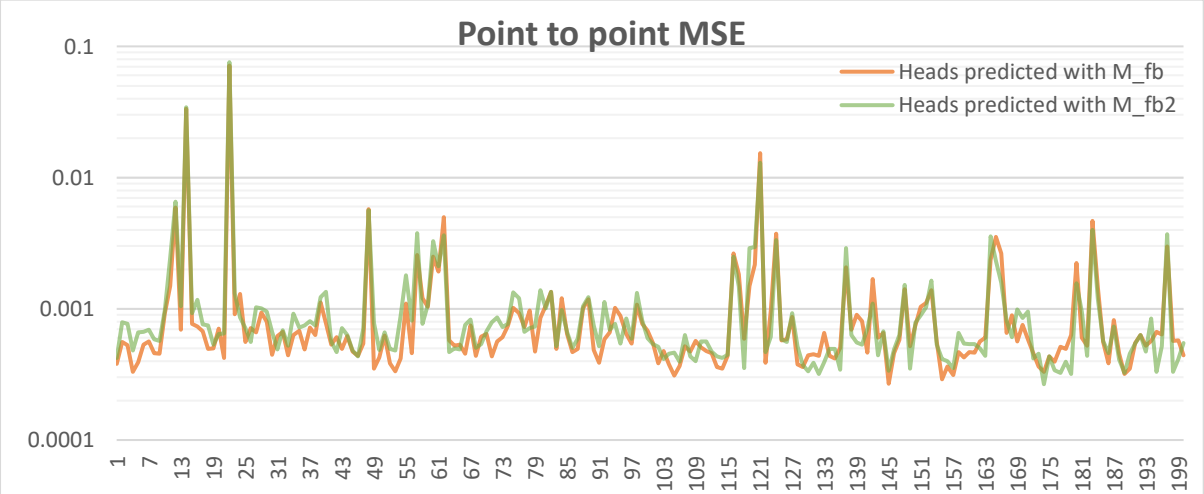


Figure 39. Point2Point MSE for both models in Experiment 2

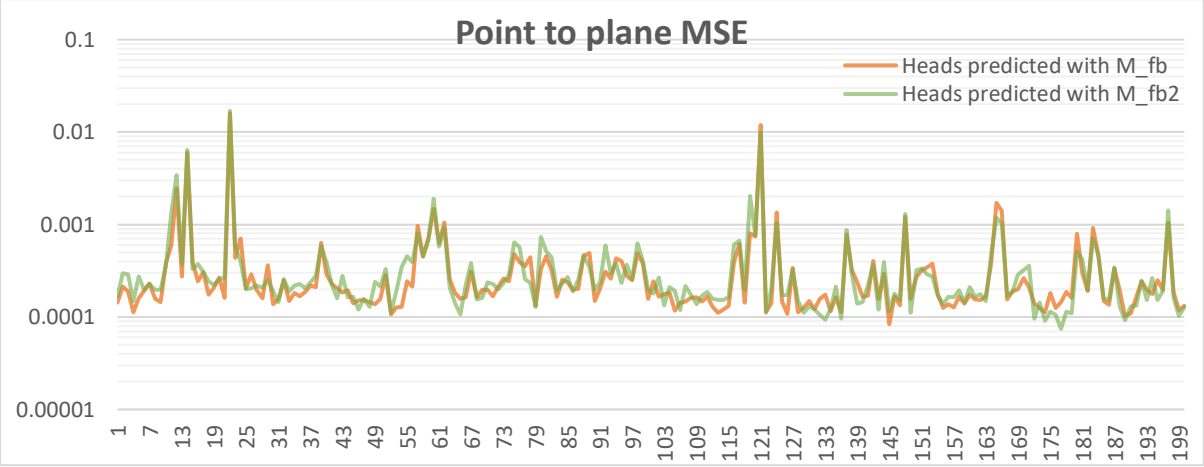


Figure 40. Point2Plane MSE for both models in Experiment 2

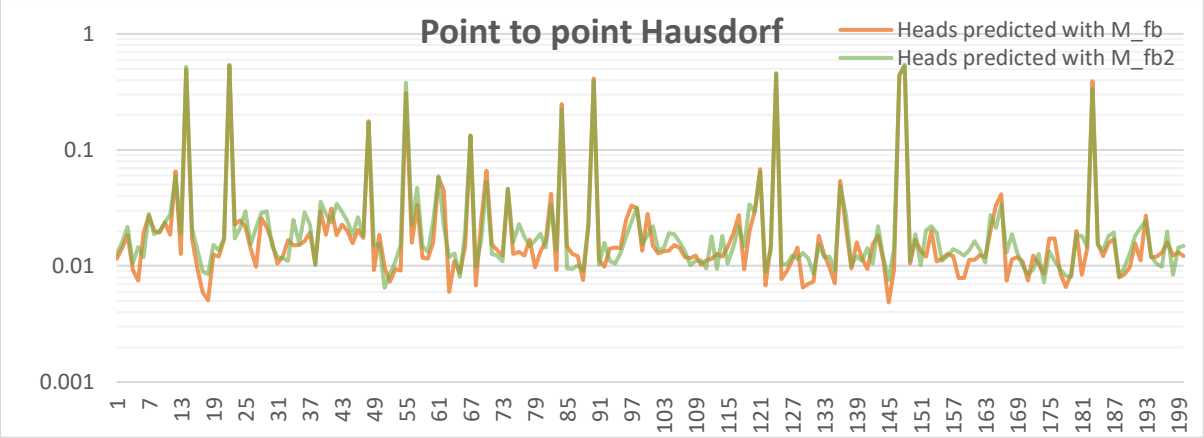


Figure 41. Point2Point Hausdorff distance for both models in Experiment 2

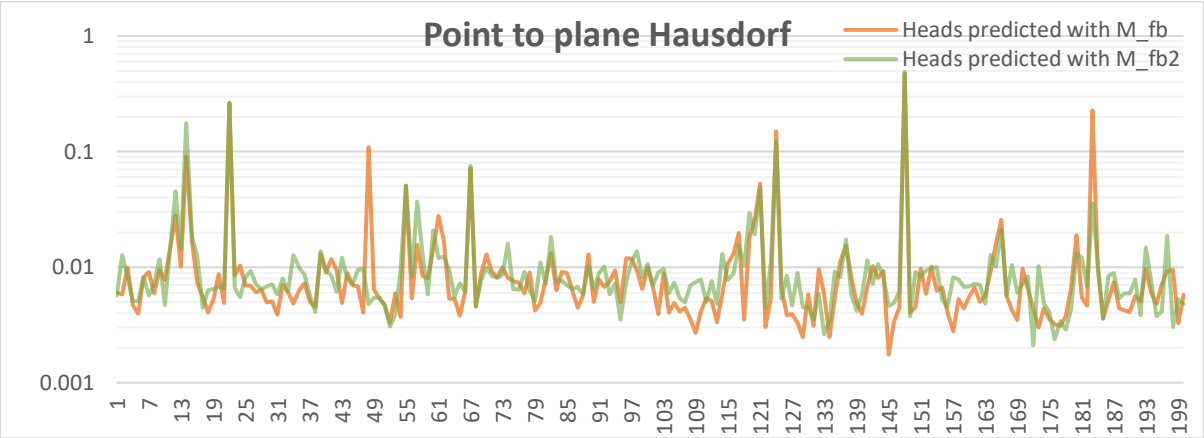


Figure 42. Point2Plane Hausdorff distance for both models in Experiment 2

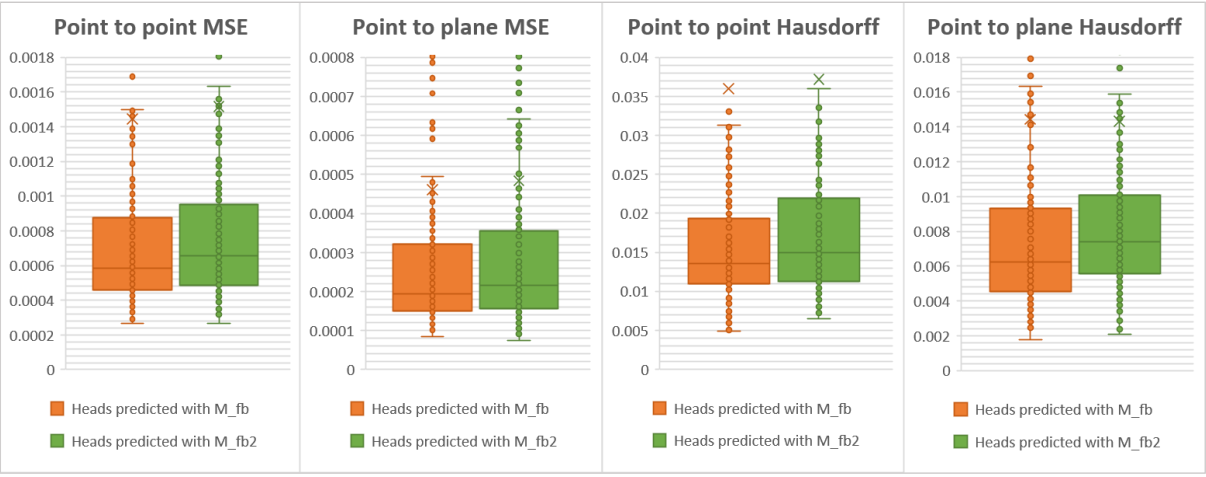


Figure 43. Boxplot of the error metrics (zoom in) in Experiment 2

Figure 44 shows the visual results of the prediction of heads using both models. Again, we observe that the visual results are not conclusive.

Since there is no clear winner, the results allow as to conclude that the radii sizes are not the ones causing lower performance for the full body models.

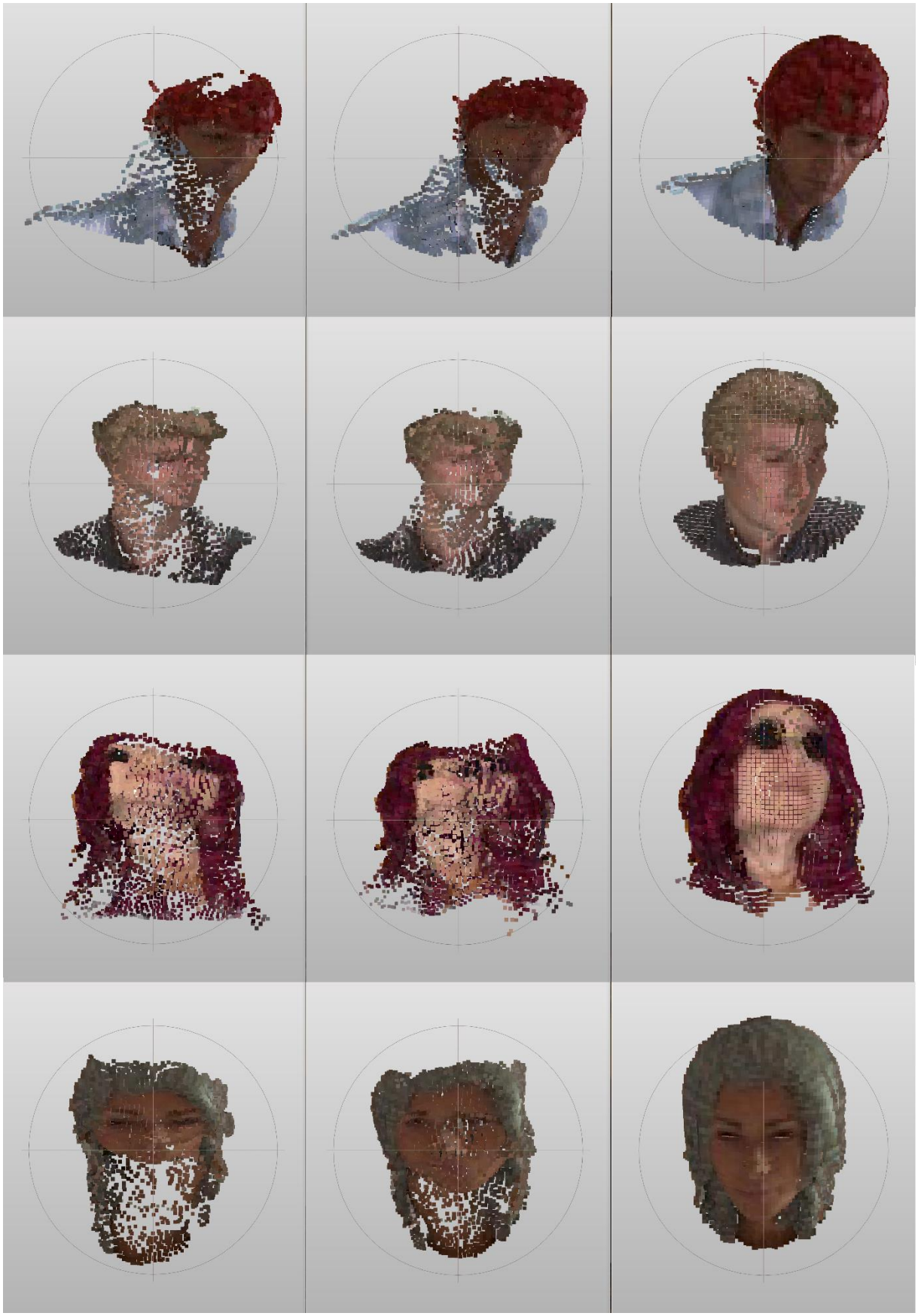


Figure 44. Visual comparison of the prediction of the heads using both models in Experiment 2. Left: using M_{fb}. Middle: using M_{fb_2}. Right: groundtruth

5.2.3 Experiment 3

The third experiment wants to check how our system performs for predicting the movement of the hands. We have trained a model $\mathrm{M}_{\mathrm{hands}}$ using only hands. As hands are smaller in size, in comparison with other body parts, we are using 1024 points for this model, to avoid duplicating lots of points. We are going to compare the performance while predicting hands with $\mathrm{M}_{\mathrm{hands}}$ versus the performance of predicting hands with M_{fb} (from experiment 1). The radii used for the Features extraction and Feature propagation layers are the same than the ones used for the heads only model, as shown in Table 8. We saw in experiment 2 that they were not causing a decrease in performance, so there is no need to change them.

	Radii $\mathrm{M}_{\mathrm{hands}}$	**Radii M_{fb}**
R1	0.01	0.01
R2	0.02	0.02
R3	0.03	0.03
R4	0.04	0.04

Table 8. Radii used in experiment 3

Figure 45 shows the evolution of the loss values obtained on the training and validation steps. If we compare this graph with the one obtained in Experiment 1 for $\mathrm{M}_{\mathrm{heads}}$, we see a big drop in performance. While in $\mathrm{M}_{\mathrm{heads}}$ we achieved a validation loss around $2e^{-04}$, in $\mathrm{M}_{\mathrm{hands}}$ the validation loss stays around $5e^{-03}$.

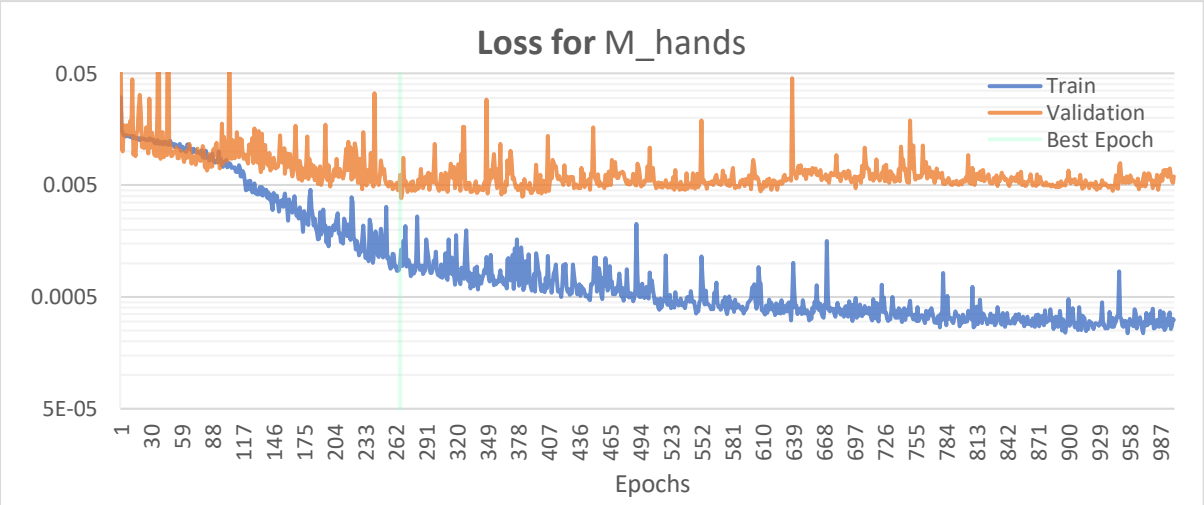


Figure 45. Logarithmic plot of the evolution of the loss for the model M_{hands}. The model was saved on epoch 266

The error metrics are presented in Figure 46, Figure 47, Figure 48, Figure 49 and Figure 50. The lines plots, as well as the boxplot, show a slightly better performance for the model $\mathrm{M}_{\mathrm{hands}}$, but the results are very noisy.

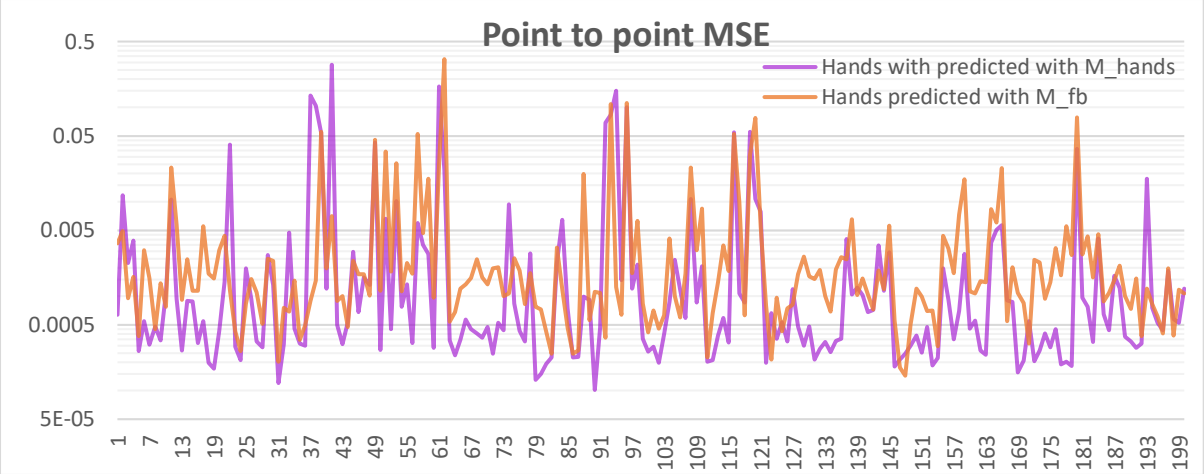


Figure 46. Point2Point MSE for both models in Experiment 3

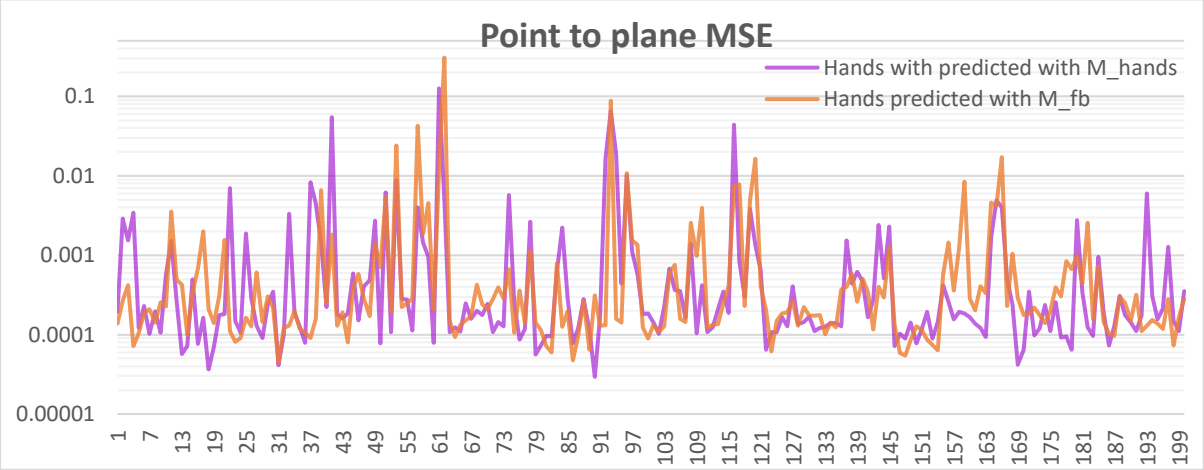


Figure 47. Point2Plane MSE for both models in Experiment 3

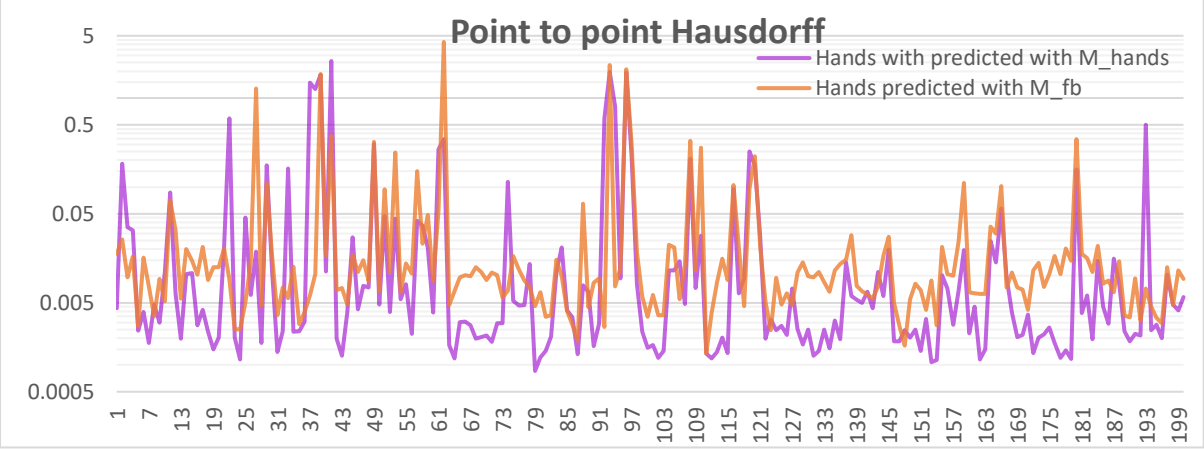


Figure 48. Point2Point Hausdorff distance for both models in Experiment 3

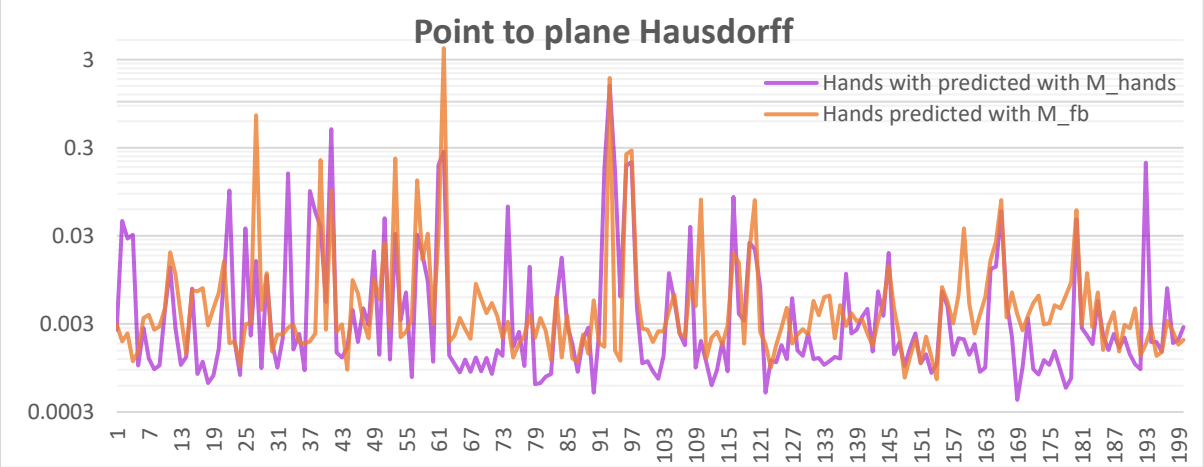


Figure 49. Point2Plane Hausdorff distance for both models in Experiment 3

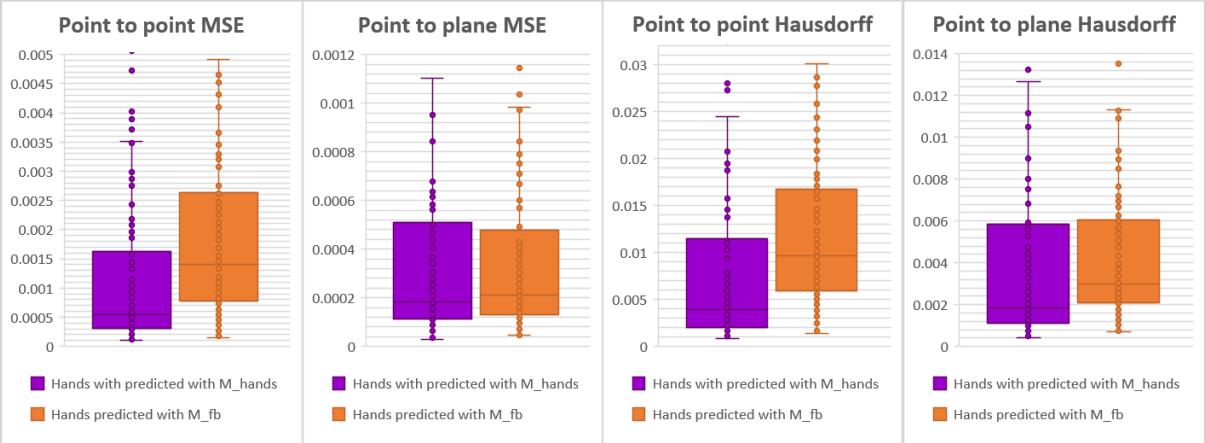


Figure 50. Boxplot of the error metrics (zoomed in) in Experiment 3

The visual results in Figure 51 show also that the predictions differ greatly from the desired value, being closer to the groundtruth the predictions using $\mathrm{M}_{\mathrm{hands}}$.

This drop in performance does not surprise us, because it proves once more our main hypothesis: the prediction of rigid motions is easier than the deformation. While is true that we have segmented the hand, the hand itself, unlike the rest of the body segments, has its own joints, so it is affected by deformation.

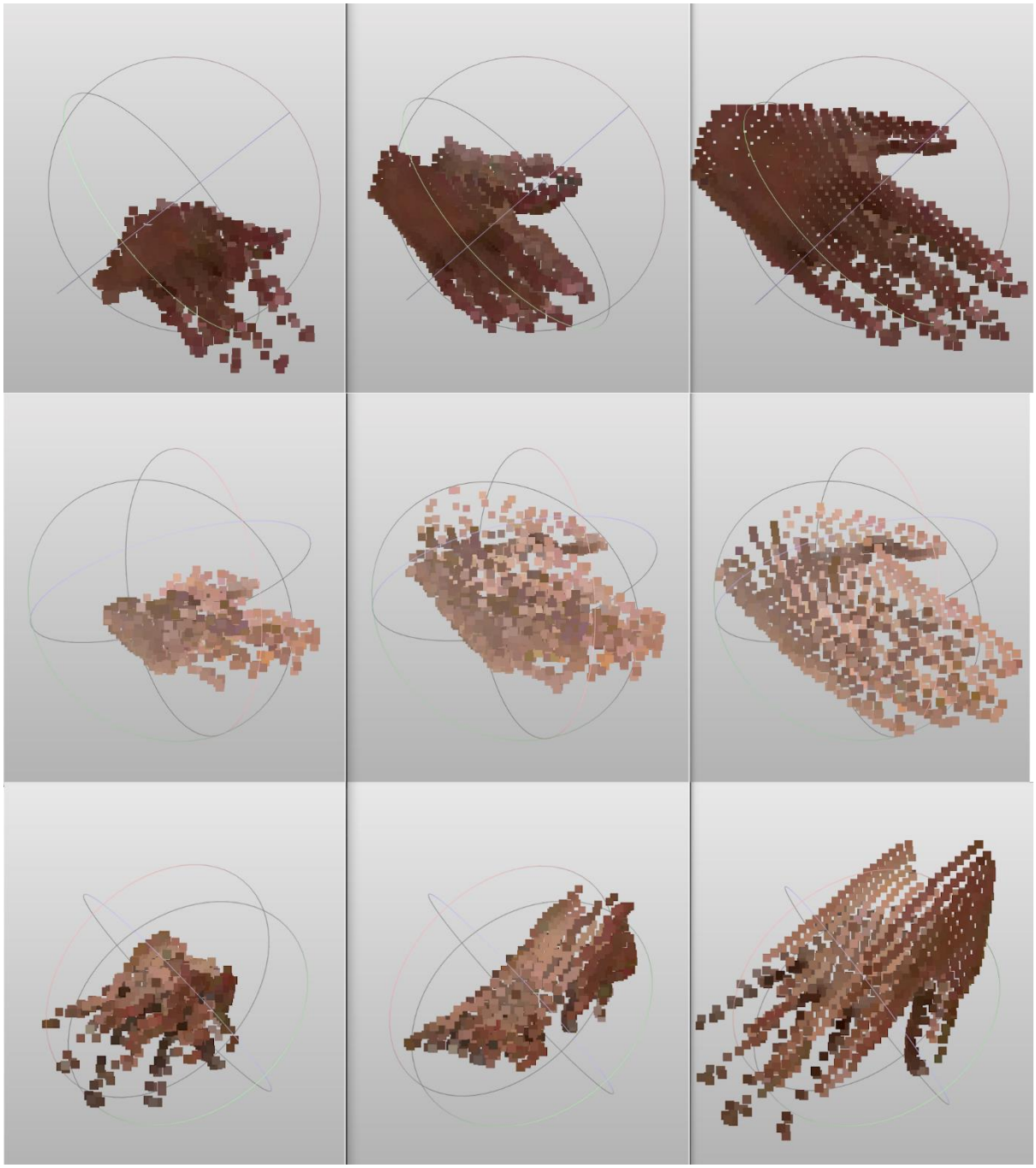


Figure 51. Visual comparison of the prediction of the hands using both models in Experiment 3. Left: using M_{fb}. Middle: using M_{hands}. Right: groundtruth

5.2.4 Experiment 4

On Experiment 3 we saw the prediction of the hands was poor and we attributed it to the fact that they have articulations and therefore, it is not a rigid movement. We want to give more solidity to this affirmation by training a model $\mathrm{M}_{\mathrm{feet}}$ only with feet, and by checking the performance of the system for the feet. In that case, as the feet are wearing boots, we hypothesize that the system will be able to make good predictions. In the experiment we are going to compare the performance of $\mathrm{M}_{\mathrm{hands}}$ with the model M_{fb} from Experiment 1.

Again, we are going to use the same radii from Experiment 1, as shown in Table 9. We have selected 2048 points for the downsampling.

	Radii $\mathrm{M}_{\mathrm{feet}}$	**Radii M_{fb}**
R1	0.01	0.01
R2	0.02	0.02
R3	0.03	0.03
R4	0.04	0.04

Table 9. Radii used in Experiment 4

The loss curve in Figure 52 already shows us a similar behavior than in $\mathrm{M}_{\mathrm{heads}}$. In this case the validation loss decays until a value of $5e^{-4}$, 10 times smaller than $\mathrm{M}_{\mathrm{hands}}$.

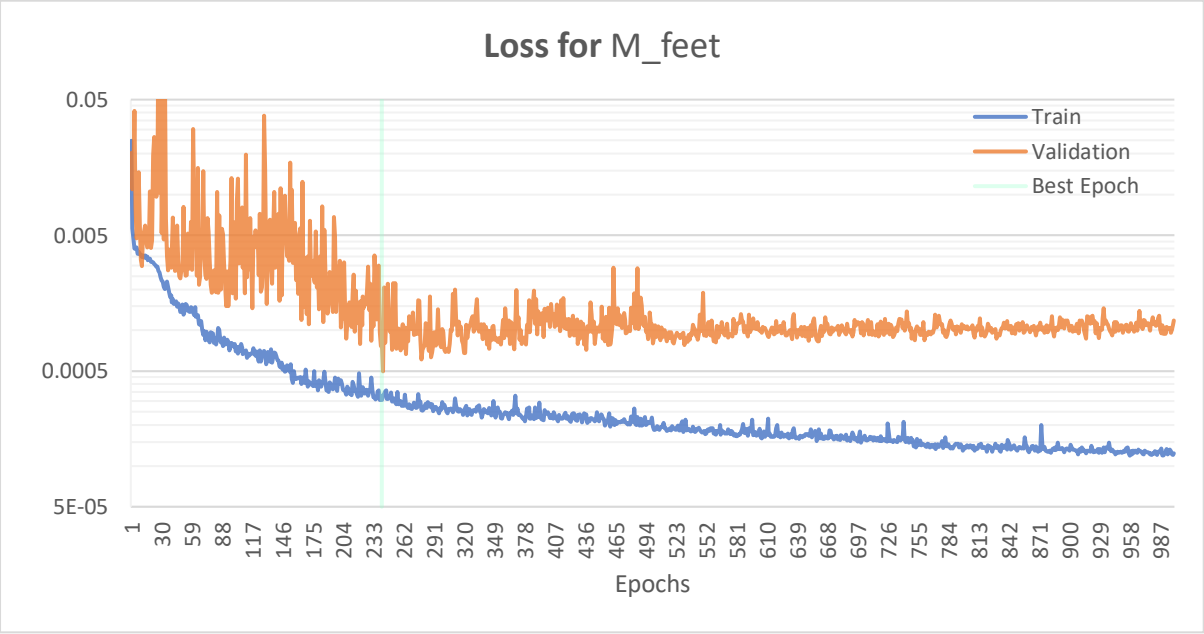


Figure 52. Logarithmic plot of the evolution of the loss for the model M_{feet}. The model was saved on epoch 241

The error metrics are shown in Figure 53, Figure 54, Figure 55, Figure 56 and Figure 57. We can see that the median is around half for the model $\mathrm{M}_{\mathrm{feet}}$, in comparison with M_{fb}. So, it confirms once again our hypothesis.

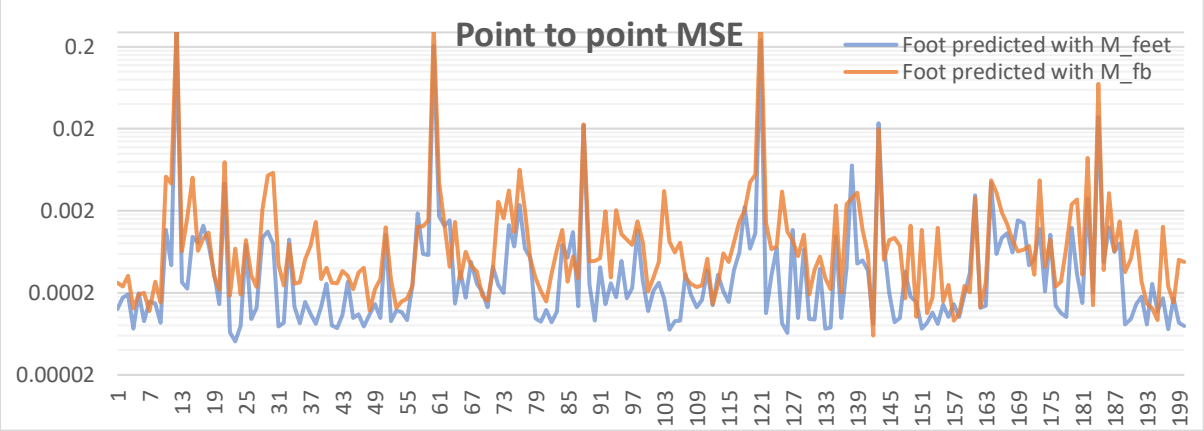


Figure 53. Point2Point MSE for both models in Experiment 4

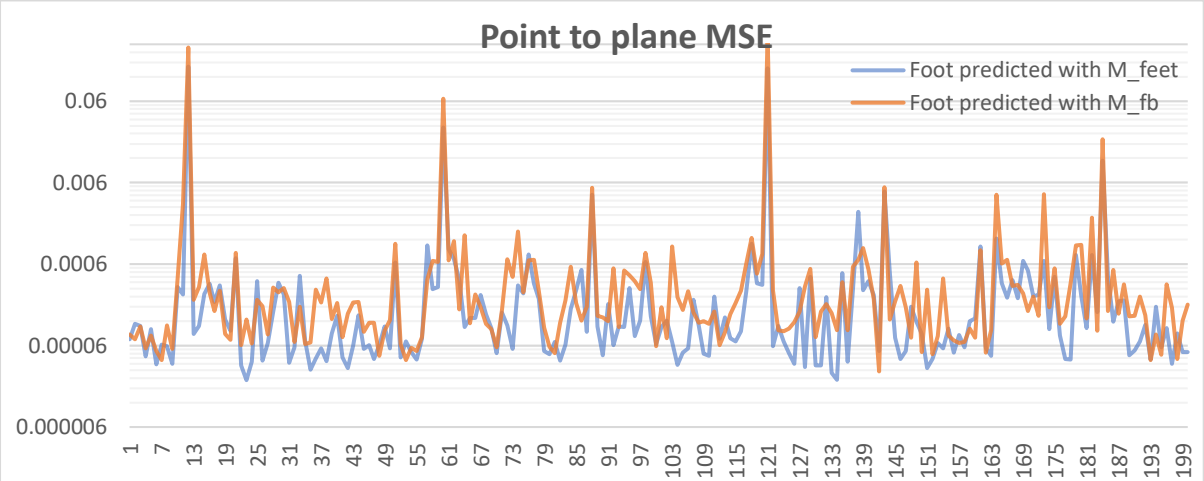


Figure 54. Point2Plane MSE for both models in Experiment 4

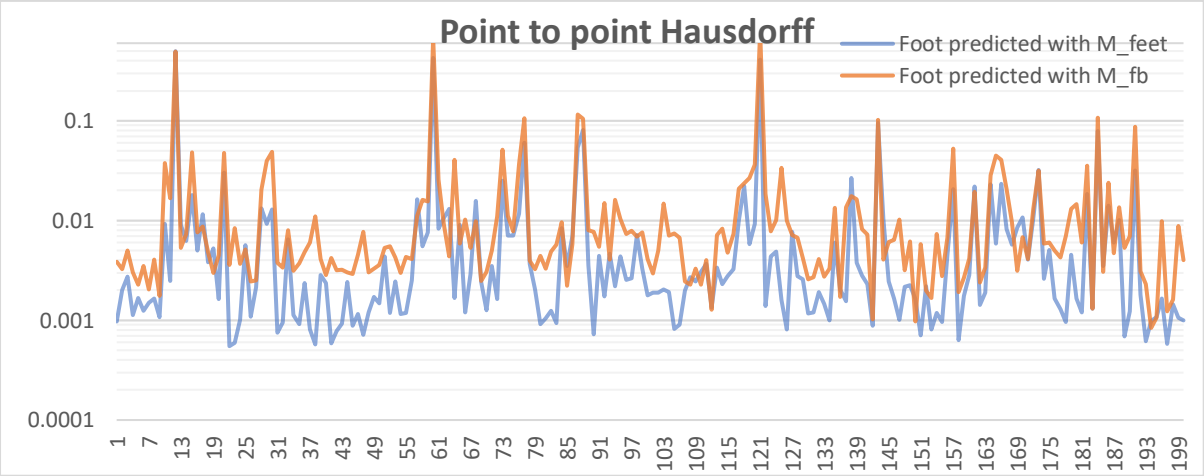


Figure 55. Point2Point Hausdorff distance for both models in Experiment 4

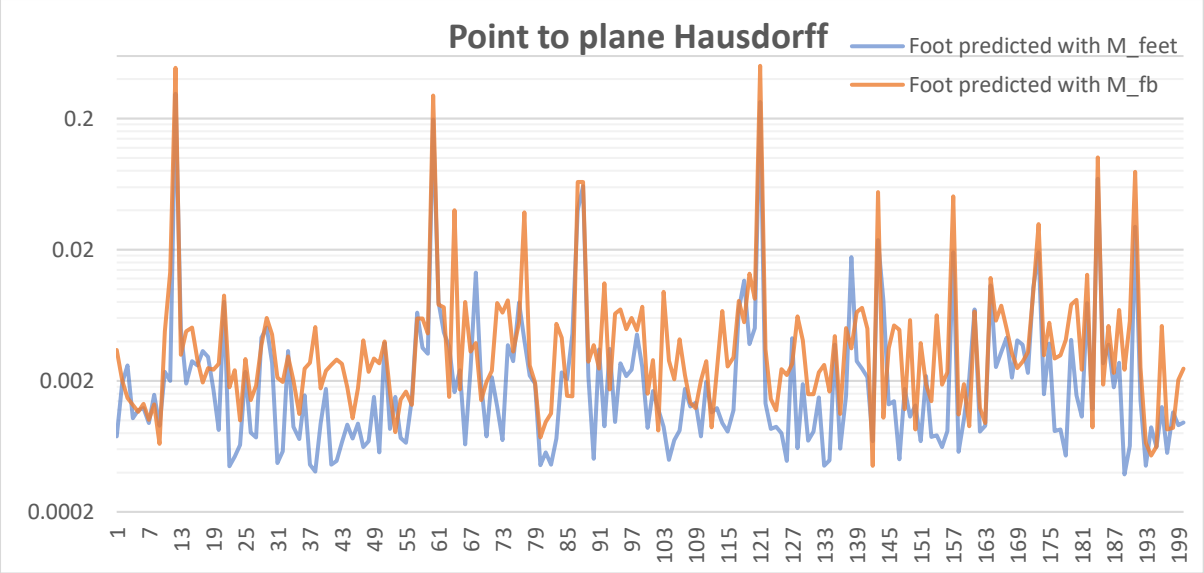


Figure 56. Point2Plane Hausdorff distance for both models in Experiment 4

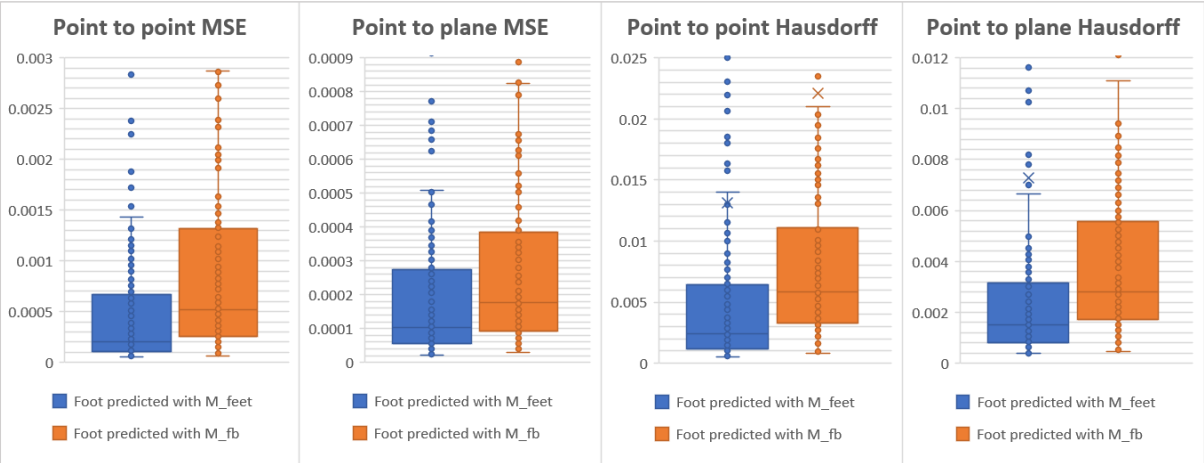


Figure 57. Box plot of the error metrics in Experiment 4

The visual results are shown in Figure 58. Results show a slight performance gain for M_{feet}; however, they are not conclusive. This might be due to the characteristics of the body part itself, having a similar shape to that of the skeleton, and obtaining benefit from the effect discussed in Section 5.2.1 Experiment 1.

Figure 58. Visual comparison of the prediction of the foot using both models in Experiment 4. Left: using M_{fb}. Middle: using M_{feet}. Right: groundtruth

5.2.5 Experiment 5

Experiment 5 is another attempt to solidify our hypothesis. In this case we are going to check the performance in the biggest part of the body: the chest. We are going to train a model $\mathrm{M}_{\mathrm{chests}}$ only with chests, and we are going to compare its performance against predicting chests with the M_{fb} model from Experiment 1.

We are going to use the same radii from Experiment 1, as shown in Table 9. We have selected once more 2048 points for the downsampling, as this is a big part of the body.

	Radii $\mathrm{M}_{\mathrm{chests}}$	**Radii** M_{fb}
R1	0.01	0.01
R2	0.02	0.02
R3	0.03	0.03
R4	0.04	0.04

Table 10. Radii used in Experiment 5

The loss curve in Figure 59 shows the evolution of the loss for the train and validation steps in $\mathrm{M}_{\mathrm{chests}}$. We see that the validation loss goes down to $2.8e^{-4}$, very close to the results on $\mathrm{M}_{\mathrm{heads}}$ where we obtained an error of $2e^{-4}$, and lower than M_{fb} where we got around $6.2e^{-4}$.

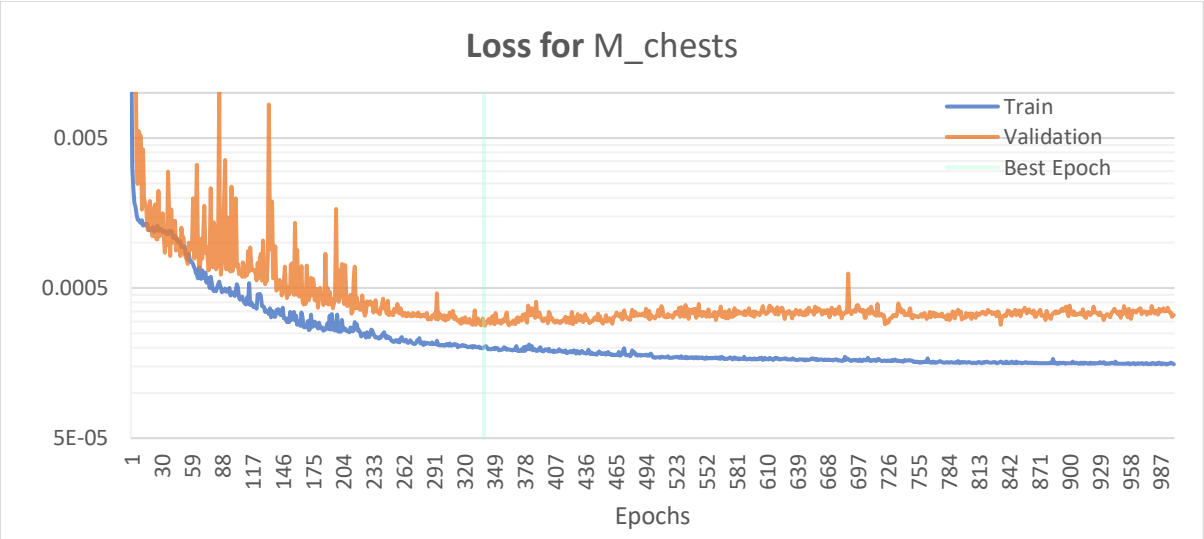


Figure 59. Logarithmic plot of the evolution of the loss for the model M_{chests}. The model was saved on epoch 339

The error metrics are presented in Figure 60, Figure 61, Figure 62, Figure 63 and Figure 64. In all of the lines graphs we can clearly see that $\mathrm{M}_{\mathrm{chests}}$ is performing better. The boxplot graph also shows that the median for $\mathrm{M}_{\mathrm{chests}}$ is half of M_{fb}, giving $\mathrm{M}_{\mathrm{chests}}$ as a clear winner.

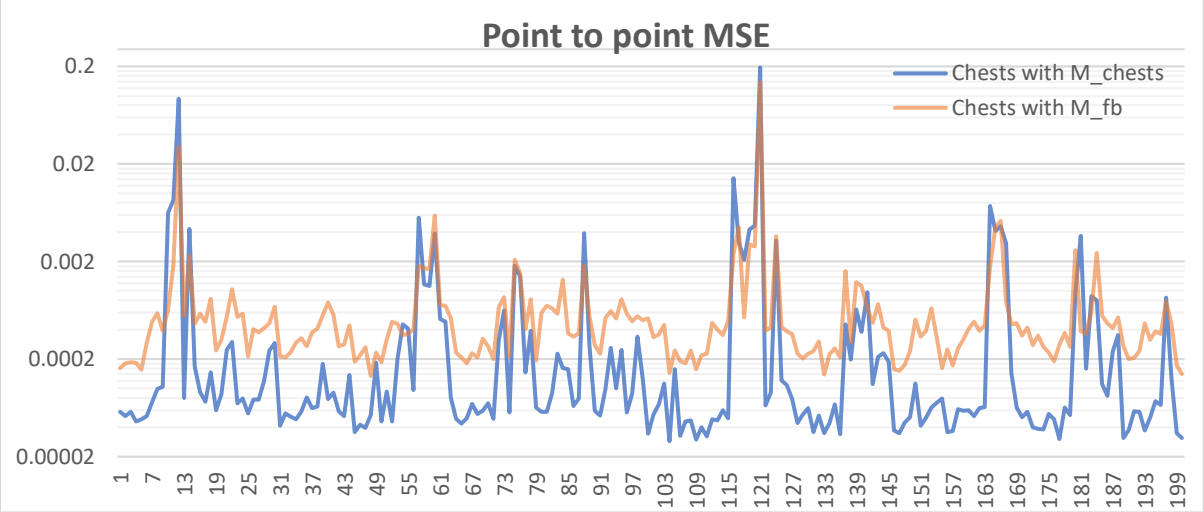


Figure 60. Point2Point MSE for both models in Experiment 5

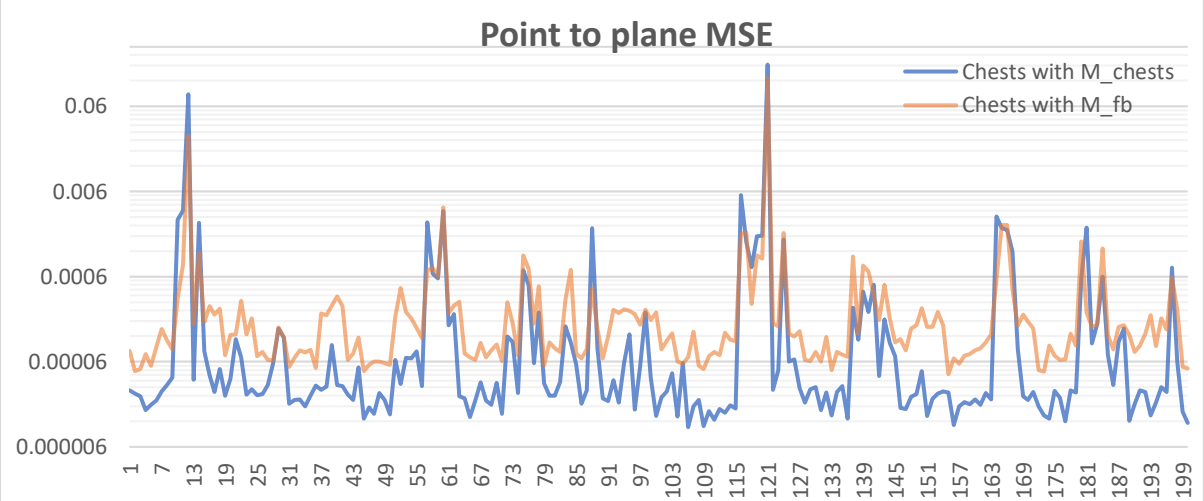


Figure 61. Point2Plane MSE for both models in Experiment 5

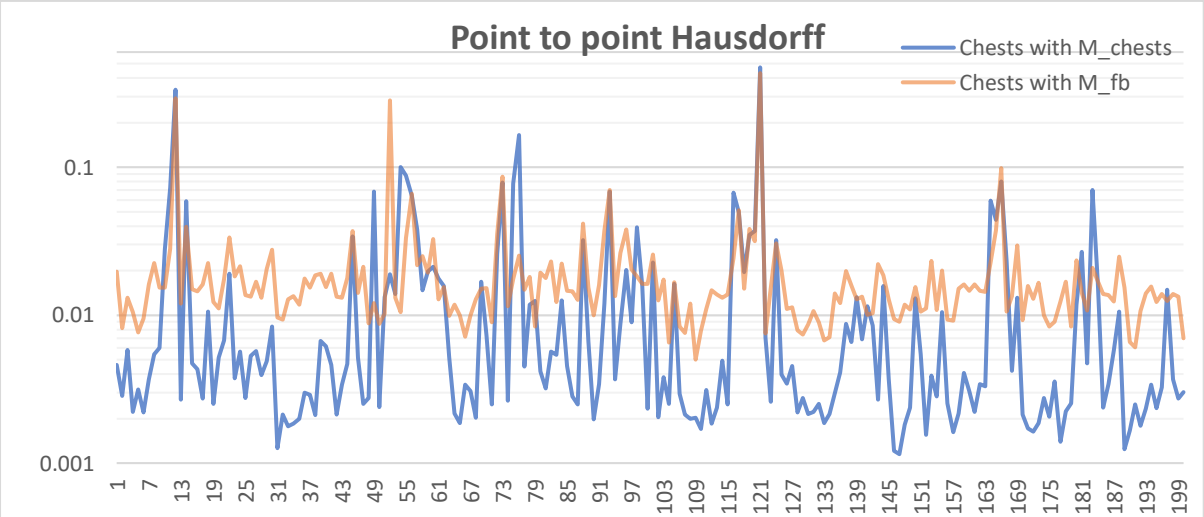


Figure 62. Point2Point Hausdorff distance for both models in Experiment 5

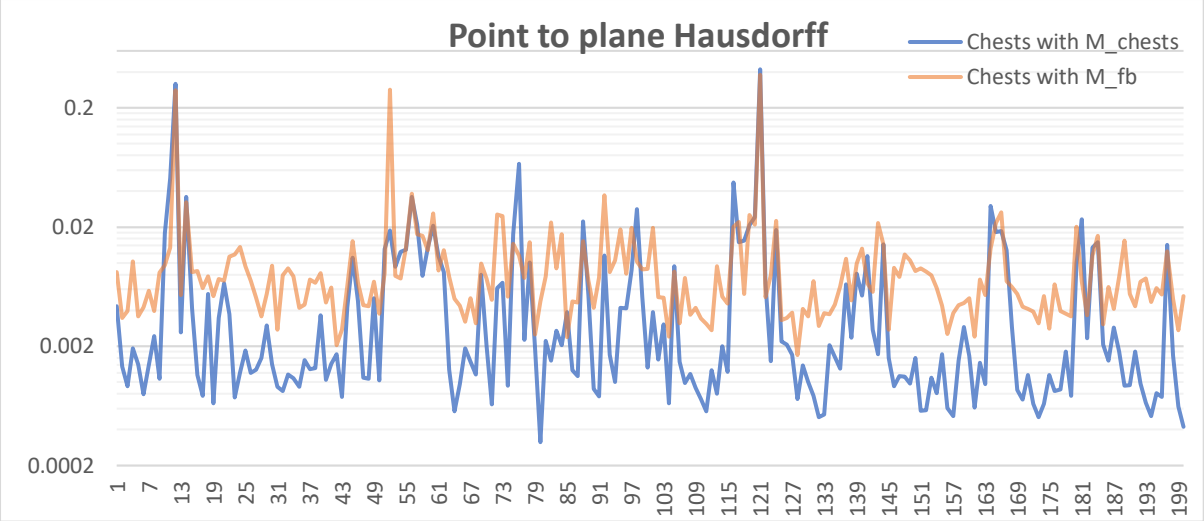


Figure 63. Point2Plane Hausdorff distance for both models in Experiment 5

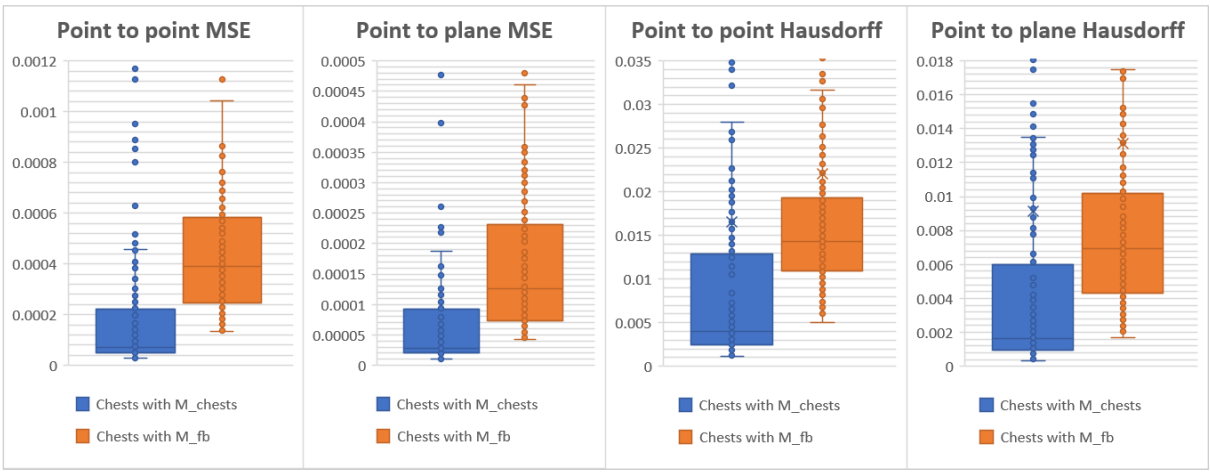


Figure 64. Boxplot of the error metrics in Experiment 5

The visual comparisons in Figure 65 clearly show, that the predictions using $\mathrm{M}_{\mathrm{chests}}$ are closer to the desired value, proving once again our main hypothesis.

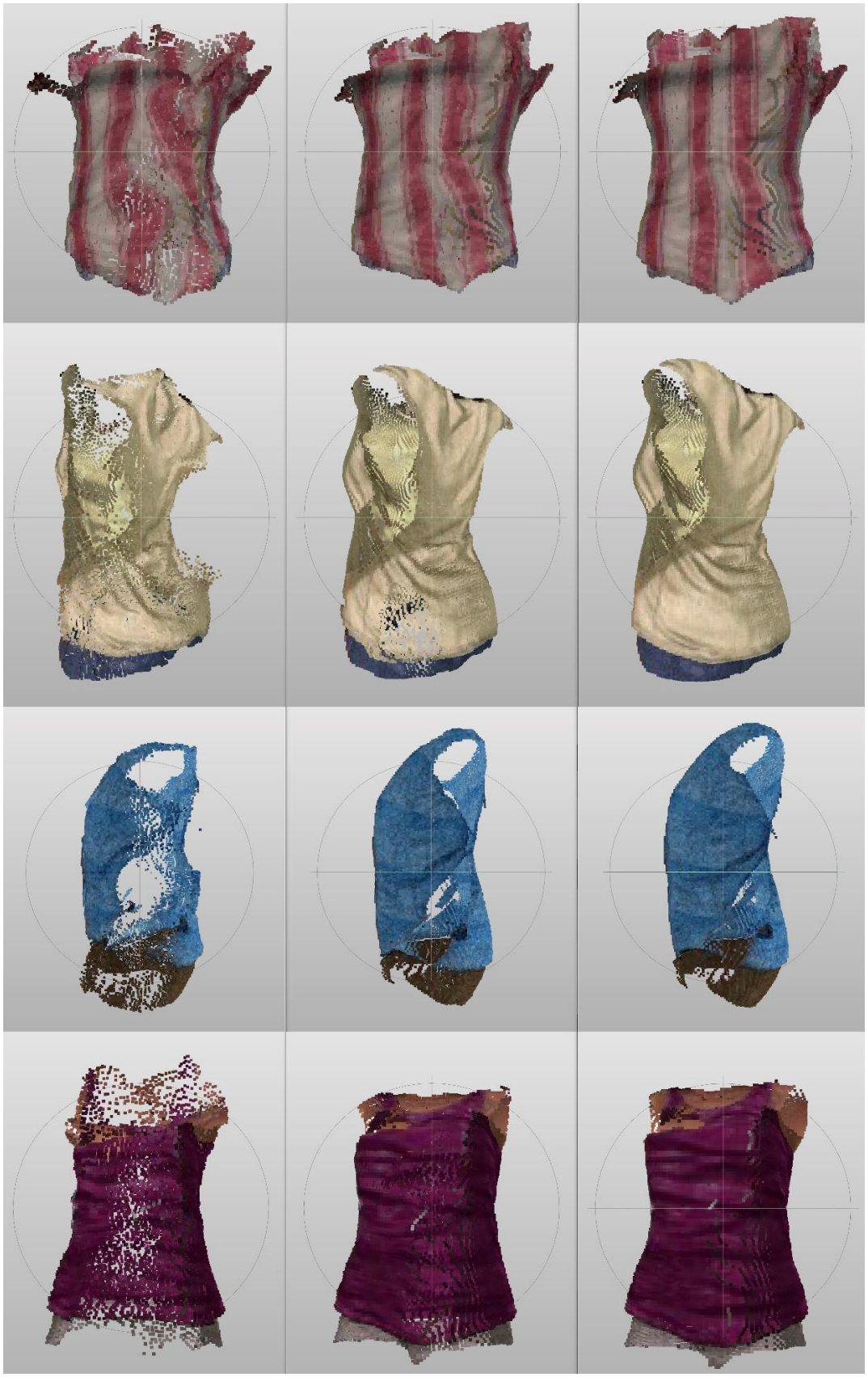


Figure 65. Visual comparison of the prediction of the chest using both models in Experiment 5. Left: using M_{fb}. Middle: using M_{chests}. Right: groundtruth

5.3 System performance

In this section we are going to analyze the performance of each block of the prediction module, in order to see the viability of bringing the current system to work in a real-time pipeline. If we want to create a pipeline to work real-time, at 30fps, we need that the full process works at most 15fps (in the case we want to interpolate only half of the frames).

Figure 66 shows a block diagram of the different blocks of the interpolation system. In our case we are reading from file, but in a real-time pipeline this block could be the one reading the point clouds from the network.

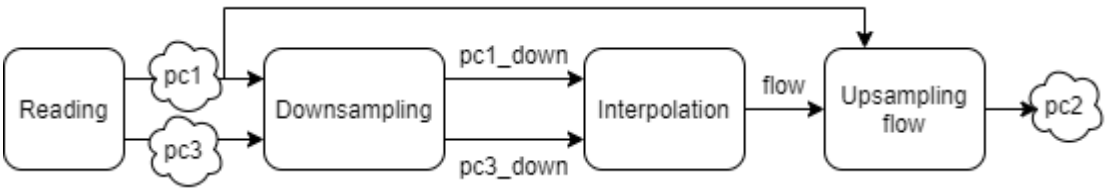


Figure 66. Block diagram of the interpolation system

The time on the reading block depends on the number of points of the raw point cloud, as can be seen in Table 11. So, reading large point clouds could be a bottleneck.

	Head	Hand	Foot	Chest	Full body
Avg. # points (raw)	6467	960	2053	20129	60000
Time (ms)	7.3	3.3	5.4	17.6	38.4
Fps	136.99	303.03	185.19	56.82	26.04

Table 11. Performance values of the reading block

The time of the downsampling process also depends on the number of points of the raw point cloud. Random downsample is really fast, as can be seen in Table 12, so it will not be a bottleneck.

	Head	Hand	Foot	Chest	Full body
Avg. # points (raw)	6467	960	2053	20129	60000
Time (ms)	0.60	0.26	0.37	1.38	3.59
Fps	1669.06	3778.43	2717.02	723.03	278.26

Table 12. Performance values of the downsampling block

The time predicting/interpolating the downsampled frames seems to be correlated with the number of points of the downsampled point cloud. As we can select this number, it will not be a bottleneck.

	Head	Hand	Foot	Chest	Full body
# points (downsampled)	2048	1024	2048	2048	2048
Time (ms)	34.3	24.4	34.6	36.3	38.2
Fps	29.15	40.98	28.90	27.55	26.18

Table 13. Performance values of the interpolation block

The time upsampling depends on the number of points of the raw point cloud, because for each point in the raw point cloud, the algorithm finds its 10 nearest neighbors to compute the upsampled flow of the point. We can see in Table 14 that this process is extremely expensive as it is right now, implemented in a sequential form. We can affirm that this is the current bottleneck and it is killing the system performance. This process could be easily speed up by applying parallelism to the computation. This is possible because the computation for each point does not depend on any other computation.

	Head	Hand	Foot	Chest	Full body
Avg. # points (raw)	6467	960	2053	20129	60000
Time (ms)	437.7	66.8	134.4	1332.8	4285
Fps	2.28	14.97	7.44	0.75	0.23

Table 14. Performance values of the upsampling block

The top part of Table 15 presents an overview of the accumulated time considering the whole pipeline. We can see that in the current state, the system is unable to work in real time, having a poor throughput fps. In the lower part of Table 15 we can see the accumulated time considering the whole pipeline but the upsampling block. This allows us to see that we still have some margin, so by solving the upsampling bottleneck, the system will be able to work in real time to interpolate individual body parts.

	Head	Hand	Foot	Chest	Full body
Acc. Time (ms)	479.90	94.76	174.77	1388.08	4365.19
fps	2.08	10.55	5.72	0.72	0.23
Acc. time without upsampling (ms)	42.20	27.96	40.37	55.28	80.19
fps	23.70	35.76	24.77	18.09	12.47

Table 15. Performance values of the full pipeline(top) and the pipeline without the upsampling block (bottom)

6. Conclusions and Future Work

In this chapter summarize the contributions of this master thesis, discuss the encountered difficulties as well as the lessons learned in the process of carrying out this thesis. We will end the chapter with a discussion about the remaining challenges and the future work.

6.1 Summary

The goal of this project and the main research question, was: *How to perform temporal interpolation of dynamic point clouds?* In order to answer this question, we explored the state of the art solutions in point cloud learning, scene flow and temporal interpolation of point clouds. We considered as possible solutions Flownet3D [6] and [7].

With the focus on social VR, we reflected about how to use current solutions to perform temporal interpolation of human bodies, in order to reduce the bandwidth needed for transmission. We decided to reuse the architecture from Flownet3D. We realized that one of the weakness of such approach, is that while it performs well for interpolating rigid motions, it is not so good for predicting deformations, which is the case of dynamic human bodies. Based on this, our hypothesis arose: *If we segment the human point clouds in different body parts, the movement of each part is closer to be a rigid motion, so by predicting each part movement individually, the performance should increase.* That through our research sub question: *How pose estimation/segmentation can be used to improve temporal interpolation of human bodies?*

In order to design some experiments to test our hypothesis, first we needed to find a suitable dataset of dynamic human point clouds, with body part segmentation annotated. As we did not find any dataset fulfilling our requirements, we decided to generate our own dynamic human point cloud dataset, based on the 3DPeople [8] image dataset. That was not an easy process due to various drawbacks presented in Section 3.3 . We solved most of those drawbacks, resulting in one of the main contributions of this master thesis: *a dataset of dynamic human point clouds with body part annotations.*

We chose to use the neural network from Flownet3D to estimate the scene flow and use it to compute the interpolated frames. Flownet3D was designed to use the scene flow as ground truth. Unfortunately, our dataset does not contain scene flow information, so we decided to modify the network to use the dropped frames (the ones we want to predict) as ground truth. We had to change also the loss function in order to work with the new ground truth data. We also tuned some parameters of the network in order to be consistent with our data size and points density. Even though we have not designed from scratch the network core architecture, the fact that we were able to predict scene flow without actually having the scene flow as ground truth, adds a novelty to the thesis.

Once we had our suitable dataset and neural network, we designed a series of experiments to test our hypothesis. On the first experiment, we put the focus on the heads, and we clearly see that the interpolation gives better results in the model trained to predict heads motion, when comparing to a model trained to predict full body motions. To discard that the radii parameters in the network are affecting the full body performance, we adjusted the radii of

the Features extraction and flow refinement layers, to be more consistent with our point cloud size. We saw similar results and can conclude it is not the radii the one affecting its performance.

The third experiment puts its focus on the hands. Using the same protocol that in the first experiment: training one model to interpolate hands movement and the other one with the full body. The results show a noisy interpolation in both of the cases, but still it provides slightly better results for the model train to predict only hands. This drop in performance does not surprise us, but it reinforces our hypothesis, because especially the hand has multiple articulations, and therefore, the movement is not so rigid as in other body parts.

The fourth and fifth experiments want to explore the performance in other body parts (feet and chests). The results give more solidity to our hypothesis, giving as clear winners the models trained with single body parts (feet and chests), against the model trained to predict the full body movement.

With the different experiments carried out in this thesis, we have been able to validate our hypothesis. We conclude by affirming that the usage of body part segmentation can boost the performance of temporal interpolation of point clouds, in the case of dynamic digital humans.

6.2 Discussion

Because of the lack of datasets fulfilling our requirements, we cannot confirm that our model is generalizable. In order to acquire body part annotations, we considered using other segmentation techniques mentioned in Section 2.4 Segmentation and Pose estimation. The main reason not to compare different segmentation techniques is time. It would add extra complexity to the project. It is worth mentioning that using those techniques would add more noise to our experiments, in the sense that we would be relying on the performance of those systems instead of having a ground truth.

We were not able to generate the scene flow from the 2D optical flow. As we mentioned in Section 3.3.3 Optical flow, this is a complex process due to occlusions and point correspondences. Scene flow is something that would be easy to get from a synthetic dataset based on meshes, where we can easily know at every moment the exact position of each point. On the other hand, working with point clouds generated from meshes is not ideal to train systems to work with real data. Models use to be not very realistic and the animations can be sort of unnatural. The data acquired with real capturing systems contain more noise and interferences, and can be affected by different lightning conditions. These divergences between real data and synthetic data, can affect the learning. Again, because of the availability of dynamic point cloud datasets with our desired characteristics, we decided that the selected dataset was our best option.

Most of the work related with learning on point clouds is focused on segmentation and classification tasks. These networks are built with some layers that perform downsampling of point clouds in order to obtain high-level features. These high-level features are then used to classify or segment the point cloud. Flownet3D is using this kind of layers too. It downsamples

the input point cloud in the features extraction layers and computes the flow for those features. Then it performs the opposite process in the feature propagation layers to obtain the upsampled scene flow. We think that in this process, the network is losing the information of fine-grained patterns. This explains why the network performs well in big body parts like the chest, and not so well in other parts like the hands, where fine grained parameters are more important.

It could be advantageous to be able to compute the scene flow estimation on point clouds of higher resolution than 2048 points, as this would allow the network to make a more fine-grained estimation. Actually, our solution of performing interpolation of each part independently, works to improve that aspect. By working with individual body parts, the downsampling factor is much lower.

Another limitation of the neural networks for this type of data, is the fact that they require inputs of the same size. As we commented already in the introduction, point clouds are an unstructured type of data, so especially for dynamic point clouds, it is practically impossible to get the same number of points between adjacent frames without applying any adjustments. We believe that this is a key point, that requires investigation, if we want to continue using neural networks for this type of data.

Regarding the loss function, the point to point error metric implementation worked good enough for our purpose, but it has its limitations. The nearest neighbor search does not consider duplicity of points, so this could be one of the reasons behind the skeleton effect commented in the first experiment, because the network can consider it cheaper to concentrate lots of points around a single point in order to decrease the error. The optimal solution could be to apply a point cloud version of the Hungarian algorithm [38], but this is of the order of $O(n^3)$, hindering even more the scalability of these systems. We could also explore other error metrics considering not only the geometry, but also the color.

We have performed our experiments using a custom dataset of 1000 triplets for training and validating the system. Given that we had a full dataset of 248.080 point cloud frames, we could have used a bigger size dataset. Again, the reason to choose only a thousand triplets, was time. We plan to perform new experiments with a larger size dataset to explore if that improves our current results.

In this thesis, we focused only on analyzing the performance of different body parts independently, but we have not checked yet the big picture. In other words, we need to explore also the result of putting all the interpolated parts together, to see if the composition works straight forward or if we need to apply some refinements.

In the last section 5.3 System performance, we saw that nowadays, the main limitation of our system, if we want to bring it real time, is the upsampling process. As it is implemented right now, it is clearly a bottleneck, but it could be easily improved by applying some parallelization strategies. For this reason, we consider this limitation to be mostly an engineering problem rather than a research one.

6.3 Future work

In the discussion above, we have already given clues about the paths we still have to travel in our future work. In this section we summarize them.

From the dataset point of view, we have three remaining tasks: 1) Generate our own dynamic point cloud dataset using our own capturing system. The plan is to use the new Kinect Azure cameras, which already incorporate a skeleton detection framework. This could provide us already enough information in order to perform body part segmentation; 2) Explore the solutions on body part segmentation mentioned in Section 2.4 Segmentation and Pose estimation, in order to incorporate them in our end to end pipeline; 3) Obtain a dynamic point cloud dataset with scene flow ground truth data in order to retrain our network with the original architecture, and compare the differences in performance with the current system.

From the point of view of the interpolation system we have identified five lines of action for future work: 1) explore different error metrics to implement the loss function; 2) retrain our models with a larger dataset to check if that increases the performance; 3) explore the results of training a single model with all the different body parts, to test if that gives better results than having a specific model for each body part; 4) explore the results of reassembling all the interpolated parts together; 5) reimplement the upsampling module in order to achieve real-time performance.

References

[1] “Facebook Horizon.” https://www.oculus.com/facebookhorizon/?locale=es_ES (accessed Jun. 17, 2020).

[2] “vTime XR: Social AR and VR - Out Now for Oculus Go, Samsung Gear VR, Windows Mixed Reality, Oculus Rift, iPhone, Google Daydream View, and Google Cardboard.” https://vtime.net/ (accessed Jun. 17, 2020).

[3] “Hubs by Mozilla.” / (accessed Jun. 17, 2020).

[4] “VRTogether - Social VR like never seen before.” https://vrtogether.eu/ (accessed Mar. 16, 2020).

[5] J. Jansen *et al.*, “A pipeline for multiparty volumetric video conferencing: transmission of point clouds over low latency DASH,” in *Proceedings of the 11th ACM Multimedia Systems Conference*, Istanbul Turkey, May 2020, pp. 341–344, doi: 10.1145/3339825.3393578.

[6] X. Liu, C. R. Qi, and L. J. Guibas, “FlowNet3D: Learning Scene Flow in 3D Point Clouds,” *arXiv:1806.01411 [cs]*, Jul. 2019, Accessed: Mar. 04, 2020. [Online]. Available: http://arxiv.org/abs/1806.01411.

[7] I. Viola, J. Mulder, F. De Simone, and P. Cesar, “Temporal Interpolation of Dynamic Digital Humans using Convolutional Neural Networks,” in *2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, Dec. 2019, pp. 90–907, doi: 10.1109/AIVR46125.2019.00022.

[8] A. Pumarola, *Dataset-3DPeople*. 2020. https://github.com/albertpumarola/3DPeople-Dataset (accessed Mar. 16, 2020).

[9] “Using Frame Interpolation.” https://files.support.epson.com/docid/cpd5/cpd52094/source/adjustments/tasks/frame_interpolation.html (accessed Mar. 18, 2020).

[10] S. Niklaus, L. Mai, and F. Liu, “Video Frame Interpolation via Adaptive Separable Convolution,” *arXiv:1708.01692 [cs]*, Aug. 2017, Accessed: Mar. 17, 2020. [Online]. Available: http://arxiv.org/abs/1708.01692.

[11] S. Niklaus, L. Mai, and F. Liu, “Video Frame Interpolation via Adaptive Convolution,” *arXiv:1703.07514 [cs]*, Mar. 2017, Accessed: Mar. 17, 2020. [Online]. Available: http://arxiv.org/abs/1703.07514.

[12] H. Men, H. Lin, V. Hosu, D. Maurer, A. Bruhn, and D. Saupe, “Technical Report on Visual Quality Assessment for Frame Interpolation,” *arXiv:1901.05362 [cs]*, Mar. 2019, Accessed: Mar. 17, 2020. [Online]. Available: http://arxiv.org/abs/1901.05362.

[13] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view Convolutional Neural Networks for 3D Shape Recognition,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, Dec. 2015, pp. 945–953, doi: 10.1109/ICCV.2015.114.

[14] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for real-time object recognition,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 922–928, doi: 10.1109/IROS.2015.7353481.

[15] Y. Zhou and O. Tuzel, “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection,” *arXiv:1711.06396 [cs]*, Nov. 2017, Accessed: Feb. 07, 2020. [Online]. Available: http://arxiv.org/abs/1711.06396.

[16] R. Klokov and V. Lempitsky, “Escape from Cells: Deep Kd-Networks for the Recognition of 3D Point Cloud Models,” *arXiv:1704.01222 [cs]*, Oct. 2017, Accessed: Mar. 17, 2020. [Online]. Available: http://arxiv.org/abs/1704.01222.

[17] “ALS Point Cloud Classification with Convolutional Neural Networks | Institute for Photogrammetry | University of Stuttgart.” https://www.ifp.uni-stuttgart.de/en/research/remote_sensing/als_point_cloud_classification/ (accessed Mar. 18, 2020).

[18] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation - Slides,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, Jul. 2017, pp. 77–85, doi: 10.1109/CVPR.2017.16.

[19] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: deep hierarchical feature learning on point sets in a metric space,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Long Beach, California, USA, Dec. 2017, pp. 5105–5114, Accessed: Feb. 06, 2020. [Online].

[20] S. Biasotti, A. Cerri, A. Bronstein, and M. Bronstein, “Recent Trends, Applications, and Perspectives in 3D Shape Similarity Assessment,” *Computer Graphics Forum*, vol. 35, no. 6, pp. 87–119, 2016, doi: 10.1111/cgf.12734.

[21] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, “PointCNN: Convolution On X-Transformed Points,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 820–830.

[22] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic Graph CNN for Learning on Point Clouds,” *arXiv:1801.07829 [cs]*, Jun. 2019, Accessed: Mar. 30, 2020. [Online]. Available: http://arxiv.org/abs/1801.07829.

[23] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis, “Coarse-to-Fine Volumetric Prediction for Single-Image 3D Human Pose,” *arXiv:1611.07828 [cs]*, Jul. 2017, Accessed: Jun. 06, 2020. [Online]. Available: http://arxiv.org/abs/1611.07828.

[24] X. Sun, B. Xiao, F. Wei, S. Liang, and Y. Wei, “Integral Human Pose Regression,” *arXiv:1711.08229 [cs]*, Sep. 2018, Accessed: Jun. 06, 2020. [Online]. Available: http://arxiv.org/abs/1711.08229.

[25] M. J. Marin-Jimenez, F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, “3D human pose estimation from depth maps using a deep combination of poses,” *arXiv:1807.05389 [cs]*, Jul. 2018, Accessed: Feb. 11, 2020. [Online]. Available: http://arxiv.org/abs/1807.05389.

[26] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, “Convolutional Pose Machines,” *arXiv:1602.00134 [cs]*, Apr. 2016, Accessed: Jun. 06, 2020. [Online]. Available: http://arxiv.org/abs/1602.00134.

[27] “[1611.08050] Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.” https://arxiv.org/abs/1611.08050 (accessed Jun. 06, 2020).

[28] J. Martinez, R. Hossain, J. Romero, and J. J. Little, “A simple yet effective baseline for 3d human pose estimation,” *arXiv:1705.03098 [cs]*, Aug. 2017, Accessed: Jun. 06, 2020. [Online]. Available: http://arxiv.org/abs/1705.03098.

[29] Z. Zhang, L. Hu, X. Deng, and S. Xia, “Weakly Supervised Adversarial Learning for 3D Human Pose Estimation from Point Clouds,” *IEEE Trans. Visual. Comput. Graphics*, pp. 1–1, 2020, doi: 10.1109/TVCG.2020.2973076.

[30] C. Kendrick, K. Tan, T. Williams, and M. H. Yap, “An Online Tool for the Annotation of 3D Models,” in *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, May 2017, pp. 362–369, doi: 10.1109/FG.2017.52.

[31] “JPEG Pleno Database: Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset.” http://plenodb.jpeg.org.website-eu-central-1.linodeobjects.com/pc/microsoft (accessed Jun. 05, 2020).

[32] “JPEG Pleno Database: 8i Voxelized Full Bodies (8iVFB v2) - A Dynamic Voxelized Point Cloud Dataset.” http://plenodb.jpeg.org/pc/8ilabs/ (accessed Jun. 05, 2020).

[33] V. Ganapathi, C. Plagemann, D. Koller, and S. Thrun, “Real-Time Human Pose Tracking from Range Data,” in *Computer Vision – ECCV 2012*, vol. 7577, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 738–751.

[34] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, “Towards Viewpoint Invariant 3D Human Pose Estimation,” *arXiv:1603.07076 [cs]*, Jul. 2016, Accessed: Feb. 27, 2020. [Online]. Available: http://arxiv.org/abs/1603.07076.

[35] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy, “Berkeley MHAD: A comprehensive Multimodal Human Action Database,” in *2013 IEEE Workshop on Applications of Computer Vision (WACV)*, Jan. 2013, pp. 53–60, doi: 10.1109/WACV.2013.6474999.

[36] G. Varol *et al.*, “Learning from Synthetic Humans,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4627–4635, Jul. 2017, doi: 10.1109/CVPR.2017.492.

[37] A. Shafaei and J. J. Little, “Real-Time Human Motion Capture with Multiple Depth Cameras,” in *2016 13th Conference on Computer and Robot Vision (CRV)*, Victoria, BC, Canada, Jun. 2016, pp. 24–31, doi: 10.1109/CRV.2016.25.

[38] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics*, vol. 2, no. 1–2, pp. 83–97, Mar. 1955, doi: 10.1002/nav.3800020109.