3D Mesh and Pose Recovery of a Foot from Single Image

by

Frédéric Boismenu-Quenneville

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The pandemic and the major shift to online shopping has highlighted the current difficulties in getting proper sizing for clothing and shoes. Being able to accurately measure shoes using readily available smartphones would help in minimizing returns and trying to get a better fit. Being able to reconstruct the 3D geometry of a foot irregardless of the foot pose using a smartphone would help for the online shoe shopping experience. Usually, systems reconstructing a 3D foot require the foot to be in a canonical pose or require multiple perspectives. There is no system to our knowledge that allows capturing the precise pose of the foot without expensive equipment. In many situations, the canonical pose or the multiple views are not feasible. Therefore, we propose a system that can infer the 3D reconstruction and the pose estimation of the foot from any pose in only one image. Our kinematic model, based on popular biomechanical models, is made of 18 rotating joints. To obtain the 3D reconstruction, we extract the silhouette of the foot and its joint landmarks from the image space. From the silhouette and the relation between each joint landmark, we can define the shape of the 3D mesh. Most 3D reconstruction algorithms work with up-convolutions which do not preserve the global information of the reconstructed object. Using a template mesh model of the foot and a spatial convolution network designed to learn from sparse data, we are able to recover the local features without losing sight of the global information. To develop the template mesh, we deformed the meshes of a dataset of 3D feet so they can be used to design a PCA model. The template mesh is the PCA model with no variance added to its components. To obtain the 3D pose, we have labelled the vertices of the template mesh according to the joints of our kinematic model. Those labels can be used to estimate the 3D pose from the 3D reconstruction by corresponding the two meshes. To be able to train the system, we needed a good dataset. Since, there was no viable one available, we decided to create our own dataset by using the previously described PCA model of the foot to generate random 3D meshes of feet. We used mesh deformation and inverse kinematics to capture the feet in different poses. Our system showed a good ability to generate detailed feet. However, we could not predict a reliable length and width for each foot since our virtual dataset does not support scaling indications of any kind, other than the ground truths. Our experiments led to an average error of 13.65 mm on the length and 5.72 mm on the width, which is too high to recommend footwear. To ameliorate the performance of our system, the 2D joints detection method could be modified to use the structure of the foot described by our kinematic foot model as a guide to detect more accurately the position of the joints. The loss functions used for 3D reconstruction should also be revisited to generate more reliable reconstructions.

Acknowledgements

I would like to thank my supervisor Professor John Zelek, who provided invaluable guidance throughout my degree. I would also like to thank my thesis readers, Professor James Thong and Professor Mohammad Javad Shafiee, for their time and input. I would also like to thank the Systems Design Department Staff for their help with my admittance, and all their administrative help throughout my degree.

I would like to thank MegaComfort for supplying resources, invaluable guidance and financial support that made this thesis possible. I would also like to thank Mitacs and NSERC for their financial support.

Dedication

This thesis is dedicated to my parents who were there for me my whole life, and supported me through my degrees, even while I was away from home. I would also like to dedicate this to my friends who followed me throughout my academic progress.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

With the COVID-19 pandemic, we have seen an increase in online shopping. For some stores, it became their only way of selling their products. This is especially true for clothing stores. Trying clothes in store meant putting yourself at risk of contracting the COVID-19 virus. For most types of clothes, you can easily buy online the item you want once you know your measurements. This is not true for shoes. Shoes are usually only described by length. If you are lucky, you will also be able to choose the width. Although these are the most influential measurements, there are eight measurements [56], presented in Figure 2.1, that should be considered to assure comfort. They are the stick length, the ball width and girth, the heel width, the instep height and girth, the arch height, and the toe height. Those measurements are not taken in a shoe store, but trying shoes can tell you if those measurements are respected within a certain limit. However, multiple studies show that even when trying shoes in a shoe store, some customers buy shoes of the wrong size. According to [11], up to 72% of the participants did not wear shoes that accommodate length or width. This study also highlights that incorrectly sized footwear leads to foot pain and foot disorders. With all this in mind, it is clear that the shoe industry can benefit from systems capable of measuring the foot with the newest technologies. With the major shift to online shopping, those systems should be easily usable at home.

A system that can measure your foot at home should use a technology that is already in your home to be of interest. If customers have to buy an expensive piece of equipment to be able to scan their feet, they will probably try to go to the shoe store instead. Nevertheless, there are technologies that almost everybody has at home: a camera. And even better, a smartphone with a camera integrated. With a camera, capturing the foot in a way that allows measurement can be simple. Most recent smartphones are even capable of capturing the depth alongside the picture. Retrieving the measurements is less trivial as there are

multiple factors that come into play. Those factors can be the position of the foot and its orientation in regard to the camera, its pose, and the quality of the picture to name a few. This thesis will present a system able to take measurements of the foot from a single smartphone image.

There are multiple ways of taking the measurements of the foot from one or multiple images. It could be as simple as using a Structure from Motion (SfM) algorithm. A SfM algorithm is the process of reconstructing a 3D structure from its projections into a series of images taken from different viewpoints [68]. The downside of this method is that it is very dependent on the quality of the images. For example, if the foot in the image makes a steep angle with the camera, then the accuracy will be greatly affected. To go around this problem, a system call FootNet [37] uses only the silhouette of the foot on each of the images to get the measurements. More specifically, they use a neural network to parameterize a Principal Component Analysis (PCA) model using the silhouettes as input. They need the user to take from 1 to 20 pictures of their foot at different angles, but 3 pictures is the minimum for accurate measurements according to their results. They show promising results as long as the foot is in the right pose and that the pictures are taken in a wide range of angles. They need to make sure to have information from all around the foot to be able to model it. However, a user's burden must not be overestimated. The more steps they have to do, the more likely they are to mess it up. With that in mind, a system that can get all the needed measurements from a single image from any angle is the ideal solution. Since we cannot obtain all the needed measurements from one image, 3D reconstruction is the way to go. It will allow measurements to be estimated from anywhere on the foot, even where the image does not cover. Those measurements are possible because a 3D reconstruction algorithm is able to predict the hidden side of the foot based on the information of the visible side since the foot is an object with specific constraints.

When it comes to reconstructing a 3D object, the most intuitive way to do it is by looking at all sides and replicating it. This goes by the idea that you need information from every side to know how to reconstruct the object. In the simple case of the cube, looking at all sides is the equivalent of looking at its development which looks like a cross formed by six squares. In most cases, however, seeing all sides will be redundant. Most objects have near symmetry in their shape. This is why engineering drawings are often shown with a three views representation. Still, these drawings assume that you do not know the represented object. When you do, you do not need three views. For some objects, you only need one view as there is enough information in that one view to reconstruct the whole object. Although it is considered an ill-posed problem [28, 78], an object with enough constraints can be reconstructed from a single image. This is the case

of the foot. A foot is described by strong attributes like toes, heel, ankle, arch, etc. Using the visible attributes of a view, it is possible to reconstruct the whole foot. Some, like Lunscher [44], have studied the problem using a specific angle of view. Using this angle, a trained approach would only have to learn to recreate the opposite side of the foot. This approach then transforms one side of the foot into the other. On the other hand, when using a random angle, a trained approach would have to learn the whole shape of the foot to be able to infer any sides. It will also be more robust to variations in the images from a user compared to using a specific angle. This is preferred for practical usage. This is why we will focus on any angle of view of the foot, excluding some angles like behind the foot where the foot is in great part hidden behind the leg. There are many methods that have been designed for 3D reconstruction [15, 27, 42, 78]. These methods take an object from any view and are able to reconstruct it. They generate intermediary results to constrain the under-determined problem of 3D reconstruction from a single image.

One of the downsides of using a single image is that this image does not have enough information to give a precise scale of the object. A solution to this problem is to pair the image with camera parameters. There are two sets of camera parameters. The intrinsic parameters describe how the internal components of the camera affect the picture. The extrinsic parameters describe how it relates to the scene it is capturing. Using those camera parameters, one can obtain a coordinate of the world coordinate system for every pixel of the image. It is then a matter of measuring the distance between two pixels to get measurements of the shown object. This approach works better for scene presenting only one object or a few objects close to one another. Another solution is to capture in the picture an object of known measurements along with the object of interest. For example, marine biologists will often capture an experienced diver in a picture of a shark to have an idea of its length. Knowing the height of the diver, it is possible to recover the length of the shark. Both those solutions provide a way to provide scale indicators, which are any ways describing the measurements of the objects in an image in units used in the real world such as metres and millimetres. To understand the impacts of scale indicators, we will experiment with three different types of image. The first one we will try is RGB images, which are images with three colour channels: red, blue and green. The second one is depth maps, which are one-channelled images where each pixel gives a distance instead of an intensity of light. The third one is RGBD images, which are images with four channels: red colour, green colour, blue colour, and depth.

Another ill-posed problem is 3D pose estimation from a single image. Much like 3D reconstruction, this task suffers from the self-occlusion of the object in the image. Constraints on the objects are needed to obtain acceptable results. Current methods suggest leaning on the structure of the object, either on the joints of the object [35, 45, 84] or

directly on its 3D reconstruction [15, 99]. This goes to show that the 3D pose estimation is intrinsically related to the 3D reconstruction, and we are taking advantage of the connection between the two for our system.

To be able to reconstruct a 3D mesh from a single image of a foot, there are multiple steps that need to be taken. Those steps can be seen as a pipeline, which was designed similarly to the one described by HandMesh [15]. However, the work of HandMesh [15] relied heavily on hand models created by other parties. This is not our case as the foot is not as well studied as the hand is. This means that there is no model publicly available and we had to create our owns. The resulting pipeline starts with the input image. In this image, there is information about the 3D structure of the foot. The image represents the projection of the 3D object, the foot, on a 2D surface. This means that 2D cues of the 3D object can be seen in it. Specifically, as highlighted in [15], a 3D mesh of an object is characterized by its shape and its pose. From the image, we can find indications of its shape by looking at the silhouette of the foot. FootNet [37], the system discussed earlier, uses this 2D cue to model their 3D mesh. We can get indications of the pose of the foot by finding the location of its joints. The joints we are looking for correspond to our kinematic foot model. This model was developed based on popular biomechanical foot models, which are the Milwaukee model [36], the Oxford model [14], the Salford model [51], and the Glasgow-Maastricht model [54]. These models help to understand which foot joints are important for pose estimation. Once the 2D cues have been extracted, we can reconstruct the 3D mesh. To do so, we deform a 3D template of the foot of our making. This template is a 3D mesh and has the advantage of already having polygons information for the surface. The deformation then only acts on the vertices which is done with an algorithm called Inception Spiral Module [15]. This algorithm uses a learned operator called SpiralConv [15, 25] to extract the variation of the mesh. It orders the vertices so that it can make sense of the sparse data. Once the mesh is deformed to represent the foot in the image, we get a mesh in a particular pose. To be able to have a precise pose estimation, we use previously selected vertices on the template to extract the 3D position of the joints of our kinematic foot model from the predicted mesh. Once this is done, we have a 3D mesh with a known shape and a known pose.

A learned approach always implies a dataset to train on. In our case, a dataset of images of feet with pose data for our application did not exist. We decided to create our own virtual dataset. To do so, we had access to a dataset of 3D feet meshes. However, all those meshes had a different number of vertices and the same pose. We had to adapt the template deformation algorithm of Allen *et al.* [2] to modify the dataset so that all the meshes had the same number of vertices. We have then used this dataset to create a general model using PCA. Knowing the number of vertices on this model and their relative

position, we were able to determine the position of the joints of our model. We then linked each joints with their counterpart in our kinematic foot model discussed earlier. This allowed us to generate a dataset of different feet in random poses. The PCA foot model was also used as the template for the Inception Spiral Module [15].

Applying the previously described algorithm with our virtual dataset did not result in us observing the results we hoped for. Our worst results, for RGB images, were inaccurate by an average of 137.07 millimetres for the length and 37.72 millimetres for the width. Our best observations, for RGBD images, were inaccurate by an average of 13.02 millimetres for the length, and 6.41 millimetres for the width. The main cause of the disparity between the two results come from the use of a single image without or scale indicators to guide the 3D reconstruction. The high error on RGBD images is due to the joints locations prediction that shows a high failure rate with an average of 41.39 pixels error. As it is standing now, the system does not give measurements accurate enough to be used in selecting correctly sized footwear.

Overall, the aim of this thesis is to describe and implement a system to recover the 3D mesh and pose of a foot from a single image, and measure this foot for a shoe size recommendation task. The contributions are as follows:

1. We propose a kinematic foot model to describe the 3D pose of the foot in eighteen joints. To our knowledge, this is the only model, based on the biomechanics of the foot, which uses the anatomical joints of the foot to describe its pose in a 3D pose estimation task.

2. We demonstrate the potential of our system to recover the 3D mesh of a foot from a single RGB image from any angle of view. To our knowledge, this is the only system designed to achieve the 3D reconstruction of a foot from a single RGB image from any viewpoints.

3. We demonstrate the potential of our system to estimate the 3D pose of a foot from a single image. To our knowledge, this is the only monocular system that takes into account the biomechanics of the foot to understand its pose.

4. We propose a synthetic dataset that allows a system to train on foot with different realistic poses.

Chapter 2

Background

2.1 Foot Measurement

Before discussing how to reconstruct a 3D mesh of a foot from a single image or how to get its 3D pose estimation, it is important to understand why it is relevant to do so. The most popular device to measure feet, which can be found in most if not all shoe stores, is the Brannock Device ® [19]. This instrument is shown in Figure 2.1. It is a ruler where you can put your foot on with a slider attached to a second ruler. The foot must lie flat on the Brannock Device ® to have accurate readings. The first ruler gives you the length of your foot while the second ruler uses the slider to measure the width. This device was created in 1926 and is still widely used. It does give precise measurements of the length and width of a foot. The rulers are usually graded for a specific shoe sizing system. In Canada and the United States of America, the US sizing system is the most popular. A Brannock Device ® graded for the US sizing system will have the precision of a quarter of one size on the length, and half one size on the width. Since shoes are sold to half sizes in length and full size in width, the Brannock Device ® can be used to give the right shoe size. Additionally, it is a simple tool which can be used by anybody. However, with technologies far superior to what existed a century ago, it should be possible to have a tool as simple of use to get even better measurements of the foot.

Some tools have been created to try to replace the Brannock Device ®. Pedographs, also called "wet foot test", are devices that will capture the imprint of the naked foot on the ground. It can be as simple as stepping with your foot wet on a sheet of paper. Usually, a pressure-sensitive pad is used. Pedographs can be used to get information on your arch in addition to the length and width. However, they are not as precise as the Brannock

Figure 2.1: The Brannock Device ® [20]

Device ® for the length and the width as they do not take into accounts the shape of the foot that do not touch the ground. They are designed more towards orthotics insoles recommendations [1]. In the more expensive options, there are many systems that can capture the shape of the foot. 3D scanners, which are used in orthopedics and biomechanics studies, and motion capture cameras with markers, which are used in gait analysis and biomechanics studies, are only a few of the options available.

With all this in mind, we can see that a system that measures a foot needs to be simple to use, inexpensive and accurate. It would ideally use a piece of equipment that is already in most households so that it can be used for online shopping. This is one problem of an instrument like the Brannock Device ®. It can only be used to measure your feet. This means that most households do not have one. What most households have is a smartphone with an integrated camera. A smartphone can be used for many applications. It can be used to find the length and the width of feet as proven by FootNet [37]. Since the camera can capture the entirety of the foot, then it can take more measurements that the width and the length.

The foot being a complex object, there is a large number of ways to take measurements from it. However, not all measurements would be useful for a task like shoe size recom-

Categories	Measurements
Length	Stick Length
Width	Ball Width
	Heel Width
Height	Toe Height
	Arch Height
	Instep Height
Girth	Ball Girth
	Instep Girth

Table 2.1: Eight measurements of the foot of Dr. Orvitz [56]

mendations. Dr. Orvitz [56], a renown doctor of podiatric medicine, has suggested eight measurements of the foot that would assure comfort in a shoe size recommendation task. Those measurements are listed in Table 2.1. They can be visualized in Figure 2.2. As a few specifications, stick length is the distance between the heel and the longest toe, no matter which toe is the longest. Arch length is the distance between the foot under the arch and the ground when the foot is resting on a flat surface. The two girths, ball girth and instep girth, are not represented in Figure 2.2. However, they can be understood by the ball width and the instep height respectively. Where those measurements are located on the foot, the girth goes around the foot following them. In this thesis, we will only use the stick length and the ball width. We will refer to them simply as length and width. We only use these two as shoes are usually only described by their manufacturers to the public by those two measurements.

When it comes to shoes, feet are not measured in millimetres or inches. They are measured in shoe sizes. There are multiple different sizing systems used to measure the foot. The most common ones are mondopoint, European, UK, and US [33]. A full-size length increment is 5 millimetres in the mondopoint system and 6.67 millimetres in the European sizing system. Shoes in the UK and US sizing system are usually given in half sizes. Both systems have the same half-size length increment of 4.23 millimetres, being different by a shift in the sizing values. A good system should produce measurements below these length increments. The Brannock Device ® is at half of each, depending on which sizing system is graded on the device. The system described in this thesis has for aim to be below the half size of US and UK sizing systems.

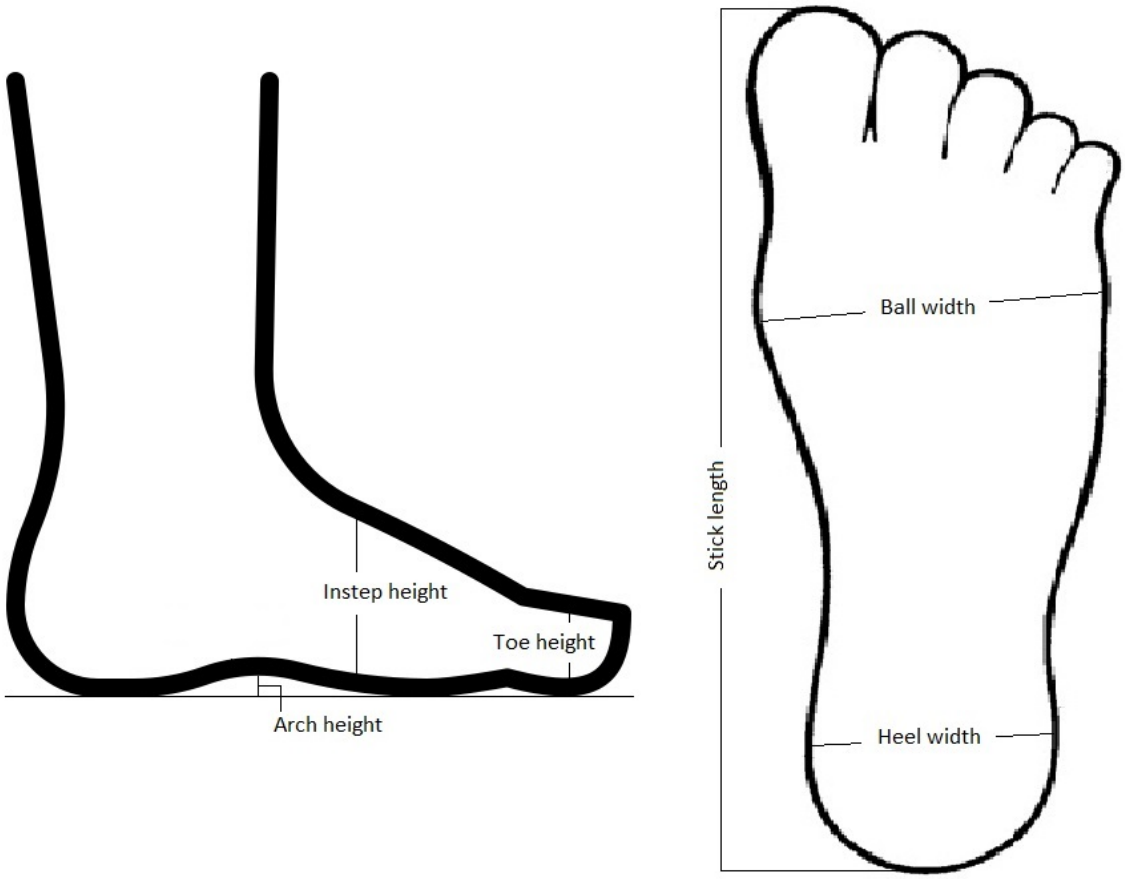

Figure 2.2: Schematization of the eight measurements of the foot of Dr. Orvitz [56], made with art from [87, 7]

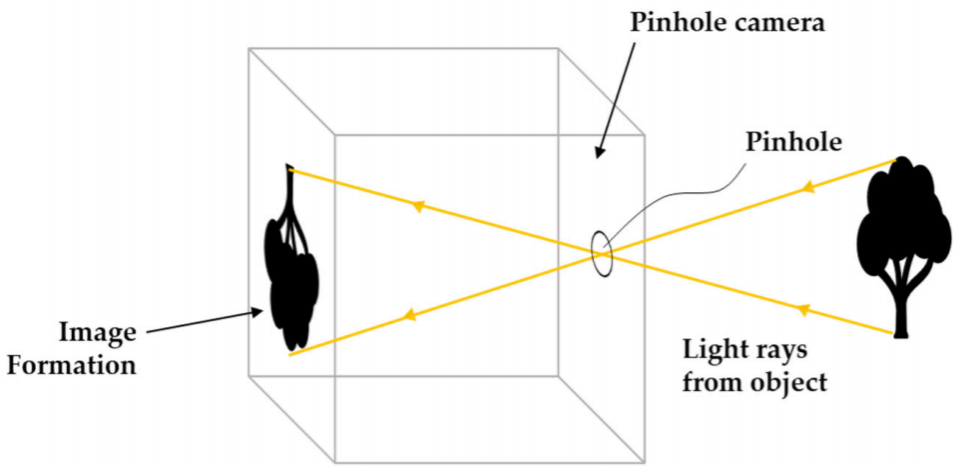

Figure 2.3: Schematization of the pinhole camera model, as illustrated by [94]

2.2 3D Projection into 2D Plane

The system we have developed takes an image as input to extract the 3D shape and the 3D pose of the foot. To understand how the foot can be extracted, it is important to understand how it has been captured in the image in the first place. A photograph is an image depicting the projection of a 3D scene into a 2D plane. This section will describe how a camera can generate such photographs and what we should expect to find in this image to reconstruct a 3D object. In our case, this 3D object is the foot to be measured.

A camera captures the rays of light emitted by the 3D scene. We describe the pinhole camera model here as it describes well the basic concepts of how a camera works. The idea behind the pinhole camera model is to capture the rays of light coming from the scene in one single pinhole, called the aperture. This concept is schematized in Figure 2.3. The captured rays can then land at the back of the camera, where a sensor will be able to register them. By using the aperture, we can restrain the rays going into the camera. Letting in more rays with a larger aperture would mean a brighter but blurrier image, while a smaller aperture would mean a darker but sharper image. The most interesting part of the pinhole camera model is that it allows representing the projection from 3D space to 2D space by a system of linear equations [94, 106]. Using the pinhole camera model, we can describe how the foot is projected from a 3D scene to a 2D plane. However, if we do not know where the foot is in the image, we cannot recover it as easily.

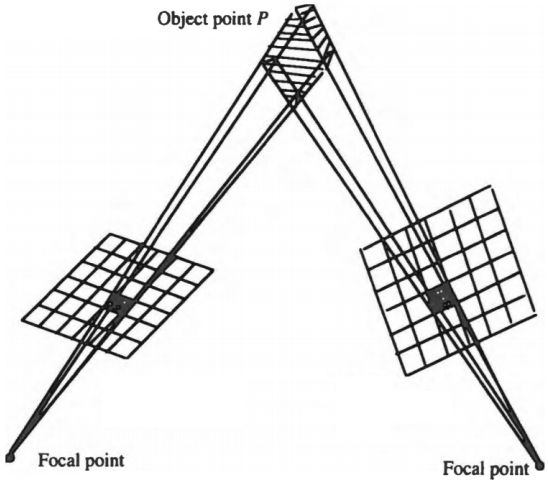

Figure 2.4: Schematization of the different angles of view of the same object, as illustrated by [106].

A 3D object's structure is described by two characteristic: its shape and its pose [15]. When looking at the object in an image, the projected shape in the 2D space appears as the silhouette of the object [37, 46]. The projected pose appears in the position of the joints visible in the image [13, 52]. The joints are where the object is not rigid. There are different kinds of joints, such as the joints allowing elastic, rotating, or translating movements. Some joints can allow more than one type of movement. By combining the silhouette and the joints, the 3D object can be properly detailed. The shape indicates the general structure of the object. The pose describes how that structure can be modified by movement within the object. For example, a hand has five fingers that protrude from the palm. That is the shape of the hand. If the fingers are clenched into a fist or if they are making bunny ears, the shape of the hand does not change. Only the pose does, as there are still five fingers. A system that is designed to reconstruct the 3D structure of an object seen in an image has to be able to recognize both shape and pose through the silhouette and the joints to succeed.

When projecting the 3D object in 2D space, the silhouette of the same object can appear differently in two distinct images. This is caused by the angle of view. This concept is illustrated in Figure 2.4. Capturing the foot from under or from one of the sides will yield different silhouette and joints landmarks. However, it is still the same object and should be detected accordingly. Using the silhouette and the joints, it will even be possible to infer the angle of view if the 3D pose can be estimated.

2.3 3D Geometry

There are many ways to represent a 3D object in a digital world, such as depth maps, voxels, parametric models, point clouds and meshes. Each representation was designed for specific purposes, from engineering design to digital art. This section will compare the popular 3D representations according to our specific needs.

The first distinction to detail the 3D representations is if they are constituted of Euclidean or non-Euclidean data. When working with images, which is in a Euclidean space, each pixel is ordered geometrically. This is not the case in non-Euclidean space. Since there have been huge amounts of research that went into processing images with machine learning, many researchers believe that a task like 3D reconstruction is more effectively completed by adapting what was done in two dimensions to three dimensions. Many 3D representations using Euclidean data emerged from this idea. We present here two of them: depth maps and voxels.

A depth map is an image with one channel where each pixel indicates a distance from the camera, compared to a classic greyscale image where each pixel indicates an intensity of luminosity. Depth maps can be processed exactly as greyscale images. Every tool that can be used on images can be used on depth maps. Convolution operators are one set of popular tools that can be used on depth maps [6, 28, 43, 44, 73, 78, 88, 89, 92]. However, for a full representation of a 3D object, multiple depth maps are required to get the whole structure of the represented object. The reason for this is that an object in a depth map appears under self-occlusion just as an object in an image does. But multiple depth maps might bring redundant information that an algorithm has to sort out. The same feature of the represented object can appear in two distinct depth maps.

A voxel is the 3D version of a pixel. While an image is a 2D grid of pixels, voxels are the elements of a 3D occupancy grid describing a 3D object. Each voxel contains an intensity of luminosity and whether part of the object is in this particular voxel. This 3D grid has to be wider than the object to be able to describe it completely as an image has to show a bit of background to make sure that the captured object is completely in the image. A grid of voxels can be processed by the tools we use on images with few modifications [18, 21, 28, 43, 64, 73, 88, 89, 90, 91]. However, they take a lot of computer space and their discretization error is worse than what we see in images due to the third dimension. A popular use of voxels is to complete the shape of an object from a depth map by reconstructing the side that is self-occluded [28, 43, 73, 88].

The other category of 3D representations is the representations designed with non-Euclidean data. In this category, the data is not ordered geometrically. Most tools de-

veloped for images cannot be used on non-Euclidean data [10]. This is especially true for convolution operators with kernels and sliding windows. This tool is one of the pillars of image processing as it allows the extraction of local information and its efficient encoding. The idea is then to find a way to process local information of non-Euclidean domains in a way that it can be efficiently reused. We present here three representations: parametric models, point clouds and meshes.

A parametric model represents the 3D object using parameterized shapes to build the 3D object. Those shapes can be basic shapes such as cubes, cylinders, cones and spheres. They can also be more complex shapes such as Non-Uniform Rational Basis Spline (NURBS). An algorithm aiming to represent a 3D object from a task such as 3D reconstruction infers the parameters of the shapes [27, 28, 40, 100]. Parametric models are the most precise 3D representations at a local level as their nature makes them continuous compared to all other representations presented in this section which are discrete. However, their application is limited to shapes that can be represented by the shapes used. Also, these methods usually have imprecision where multiple shapes connect to each other. This is only a disadvantage when recreating a complex shape. For 3D objects made of multiple moving parts, a parametric model can be the best choice if each of those parts can be represented by one parametric shape. This is why parametric models are popular in healthcare [67, 69, 75], as components of the body can be expressed by parametric shapes.

A point cloud is a set of points that determines the outline of the shape in 3D space. The points usually lie where there are fluctuations in the structure of the 3D object. Where there are no points, we expect that an imaginary plane connects the surrounding points. A mesh is a point cloud with polygons, sometimes called faces, that characterize these imaginary planes. The points of a mesh are called vertices. The faces are planes that are connected to multiple vertices. For example, a triangular mesh is a mesh that has polygons that connect to three vertices, linking them together. The popularity of meshes as 3D representations comes from the flexibility that the representation presents. The quality of the meshes can be controlled by the number of vertices. The polygons can show colours and textures for digital art. However, the face values are difficult to produce. The points are in non-Euclidean domain, which means that they are not ordered geometrically. This makes those 3D representations difficult to handle for many algorithms. Still, there are many methods that can learn the structure of point clouds and meshes and achieve tasks such as 3D reconstruction [12, 15, 25, 28, 42, 44, 62, 86, 93]. It is only a matter of being able to build features that take into account the nature of the non-Euclidean data [10]. As a last note, when showing a parametric model on a computer, the shapes are often discretized as meshes as they are easier to handle compared to a continuous shape that has to be discretized by the graphic processor of the computer.

We have presented in this section two 3D representations of Euclidean domain, depth maps and voxels, and three 3D representations of non-Euclidean domain, parametric models, point clouds and meshes. While they all have their advantages and disadvantages, the flexibility of meshes and their ability to show textures and colours is the reason why this representation is appropriate for our system. The discretization error can be controlled through the number of vertices, and they can be easily projected on images. We will have to use methods oriented toward non-Euclidean data to be able to learn to recreate 3D meshes of feet.

2.4 3D Reconstruction

The task of 3D reconstruction from a single image is an ill-posed problem. For one object in an image, there is an infinite number of different reconstructions possible [28, 78, 88, 91, 102]. To solve this under-determined problem, we have decided to construct a pipeline that collects the necessary cues from the image to be able to properly constrict the 3D reconstruction. This section discusses the advantages and disadvantages of using a pipeline for the design of our method.

To address the under-determined problem of 3D reconstruction from a single image, we need to combine low-level image cues, structural knowledge, and high-level object understanding [6, 28]. As we have presented in 2.2 3D Projection into 2D Plane, the structural knowledge can be found in the joints of the foot, while the high-level object understanding can be described by its silhouette. As for the low-level image cues, they can be extracted by a convolutional neural network [10]. These concepts often lead to the creation of a pipeline that extracts information from an image, then discards the image to only keep the byproducts for the next steps of the pipeline. The problem with this approach is that it generates an enormous model which is computationally heavy. The hardware needs are often difficult to meet. The main reason for this is that computations made in some regions of the input image are then discarded. The final output is generated considering only a fraction of the computations made, which is a waste of computation time and power.

Pipelines are not the only way to create methods for 3D reconstruction. There are many methods that recover the 3D structure directly from the image [28]. Those methods are lightweight and require less training to achieve good results. However, methods like these are biased toward recognition and retrieval [79]. Usually, those methods implicitly classify the input image to output an averaged mesh of its category with slight variations. The model works by averaging the relevant meshes it learned during training [78].

Methods that infer the 3D structure directly from the input image also have poor generalization over unseen examples [5, 6, 28, 78, 79, 80, 102]. This comes from the classification effects that these methods yield. As we want our method to be able to generalize to unseen example of feet for accurate measurements, a pipeline is the way to go. Generalization is important in our case as the foot can have an infinite number of different poses. The training data cannot hold all these different poses, which means that our method needs to be able to handle new cases.

A last advantage that pipelines bring is the possibility of intermediate supervision. When training a method that predicts directly the 3D mesh from an image, only the results can be tested against a label. For a pipeline, it is possible to produce intermediary results which can be tested [15, 28, 52]. For example, if the pipeline generates silhouettes after the first step, they can be compared against masks of the images of the training data to see if those silhouettes are accurate.

2.4.1 Literature Review of 3D Reconstruction Methods

One of the primary goals of our system is to recover the 3D reconstruction of a foot. There are a wide variety of methods designed for this very purpose. This section presents a short literature review on the subject. We will not cover the impacts of the context of training have on each method [26], assuming that it does not have as much impact as the method itself.

Tatarchenko *et al.* [78] designed a method that recreates an object in any angle desired. To do so, they take as input a single image of the object and the desired angle of view, which are both encoded with a different encoder. The encoder for the image uses convolutional layers while the encoder for the angle uses fully connected layers. The decoder then takes both latent space to output the object in the desired angle of view using up-convolutional layers. Their output is a RGBD image. Using this method, they generate six views from around the object and created a 3D mesh out of them. This method is great at reconstructing the 3D shape of a self-occluded view of the object. They achieve good results even with cluttered backgrounds. However, they state that while the shape is well kept, they lose some details due to the fusion of the depth maps and the averaging done by the network to output the best result according to its Euclidean loss.

Wu *et al.* [88] designed MarrNet. This method achieves 3D reconstruction with a pipeline of two steps. The first step is to generates a set of three feature maps with a neural network based on ResNet [29]. The feature maps represent the surface normals, a depth map and the silhouette. The second step uses a 3D convolutional neural network

to output a mesh of the object. This method has a powerful segmentation algorithm that allows it to generate the silhouette of the object from a cluttered background. It also shows a good generalization to unseen classes. However, when the objects to be reconstructed have thin structure or are complex, the reconstruction often fails.

Choy *et al.* [18] designed 3D-R2N2. This method uses 2D convolutions and 3D deconvolutions to reconstruct an object from a single image or multiple images of the same object. Their network learns a mapping from images of objects to their underlying 3D shapes using a recurrent neural network. They use voxels as the 3D representation of their output. Their system can reconstruct most objects with great fidelity, even when the object presents a lack of textures. However, they cannot recreate the finer details due to the discretization error.

Liu *et al.* [43] designed SATNet. This method is designed to be able to take either a RGB image, a depth map or a RGBD image for a 3D scene reconstruction task. It first start by segmenting the objects in the input with a neural network based on ResNet [29]. It then uses camera parameters to project the 2D input into 3D space. It completes the 3D scene recovery with a 3D convolutional neural network that produces a set of voxels that are each label with their class found in the segmentation step. Apart from its competitive results, this method prides itself on its adaptivity and three-step pipeline that can easily be modified to keep up with the state of the art. However, segmenting the image before projecting into 3D space leads to a few instances where the objects are not properly recognized and thus reconstructed to appear different as they are supposed to.

Rezende *et al.* [64] created a method that is designed with the goal to be as general as possible. To do so, they use a conditional generative model to compute the probabilistic inference that allows them to recover the 3D structure of the object of the input image. They use a set of manifolds to describe the 3D object, allowing their network to generate parameters for those shapes. This method has a great generalization over unseen classes. It is also resistant to noise in the image. However, the method is only accurate for simple shapes. As the objects become more complex, the results get blurrier.

Groueix *et al.* [27] designed AtlasNet. This method parameterizes surfaces to fit the object. They use many surfaces for one reconstruction. The idea is to segment the object to reconstruct into multiple basic shapes and then use the surfaces to cover those basic shapes. They also predict how the surfaces will be parameterized on the hidden side of the object. Then, they stitch it up together to have a 3D model. They can use RGB images or point clouds as input. This method is able to recover the finer details of the shape it reconstructs. It can be applied to complex or thin objects without losing accuracy. However, the reconstructed 3D objects show imperfections at the edges of the parameterized

surfaces that constitute it.

Lei *et al.* [40] designed Pix2Surf. This method uses their own normalized coordinate space to parameterize a set of surfaces to represent the 3D object. The input image is processed using an encoder-decoder architecture to predict multiple intermediary results. They map the object into their normalized coordinate space. They predict the silhouette of the object. They predict feature maps of the object priors. With all of those, they are able to infer a set of continuous surfaces that represent the shape of the 3D object. The colour of the input image is able to follow throughout the network to colour the resulting surface. This method allows the pixel to follow through the network all the way to the resulting set of surfaces, assuring that information is not lost by the network. The method is also recreating meshes from the same object but at different angles of views in a consistent manner, meaning that the resulting 3D objects will all be very similar. However, this method does not create watertight objects and the edges of the surfaces show discontinuities and imprecision.

Cai *et al.* [12] designed a method that predicts meshes from drawn caricatures of faces. They have designed a Principal Component Analysis (PCA) model of a mean caricature face. First, they start by finding priors of the face in the image using an encoder based on ResNet [29]. Their decoder is four fully connected layers. These layers transform the latent space of ResNet to a point cloud. The last layer is initialized with the values of the components of their PCA model. The network can then base itself on the parameterized model to learn how the mesh should be. This method generates meshes that are smoother than what most methods generate, meaning that their network generates a low level of noise. However, due to the nature of their input being caricatures, they cannot have inputs with cluttered backgrounds. Also, this method cannot be generalized well as the PCA model parameters used to initialize the last layer of the decoder are for a very specific mesh.

Li *et al.* [42] designed PC-GAN. This method uses a Generative Adversarial Network (GAN) to generates point clouds from RGB images. With the idea that the sparse data of a point cloud cannot be handled like images, they decided to modify the well-known Wasserstein GAN objective by redefining the Wasserstein distance. To do so, they computed a lower bound and an upper bound that would allow the Wasserstein distance to be used with sparse data such as point clouds. The idea behind this is that the Wasserstein distance is affected by the lack of geometric order of sparse data. By bounding the distance instead of trying to find the true value, they can get a close approximation. They also point out that most GAN model uses an approximation of the Wasserstein objective as it is expensive to compute the true distance. They show that their bounding scheme yields better results than this approximation on Euclidean data. This method also generalizes

well on unseen data as it learns a pointwise transformation from pixels to points in the point cloud, which encourage the model to learn the building components of the objects. However, this method works better on objects that shows an axis of symmetry. If not, the reconstructed 3D object can be blurry.

Mescheder *et al.* [49] designed ONet. This method learns to implicitly represent the 3D surface as the continuous decision boundary of a deep neural network classifier. To do so, it uses an occupancy function to learn how to recreate the shape of the 3D object in a voxel grid. It then learns a surface that wraps around the voxels to create a precise 3D structure. The model reevaluate the corresponding pixels corresponding to the edges of the voxel grid to give more variations to the surface. This method is able to create 3D structure with a surprisingly good quality, not showing the noise we can often see in other reconstructed 3D structure. It also generalizes well to unseen classes and real data. However, the surface created is dulling some sharp edges in its attempt to create smooth results.

Chen *et al.* [15] designed HandMesh. This method generates 3D meshes of hands. They have designed a pipeline to do so. The first step is to generate the silhouette and a map of the landmarks of the joints of the hand. Using those, they are able to use a variation of SpiralNet++ [25] to deform a template of a hand into the hand that they want to reconstruct. The variation, called Inception Spiral Module, reorders the input of each linear layer used according to the structure of their template. They repeat this action for multiple scales of the template mesh to acquire information at different scales. This method is able to create reliable reconstruction of the hand in any pose and angle of view. However, it cannot generalize to objects other than hands due to the network working by deforming a template representing a hand.

Kanazawa *et al.* [34] designed HMR. This method works on full-body 3D reconstruction. To do so, they first start by encoding the image with a convolutional encoder. The purpose of the encoder is to segment the body from the background, segment the body into body parts, and extract local information. Then, an iterative 3D regression is done on the latent space of the encoder. The objective of the 3D regression is to infer the 3D mesh and the camera parameters. This is done in a way that the 3D joints of this mesh are projected into the same 2D space as the input image so that it can be compared to the annotated 2D joints of the labelled ground truth. To train the network to give good meshes in addition to accurate 3D joints, the meshes go through an adversarial discriminator that tries to predict if the mesh has been generated by HMR or if it comes from a dataset of 3D mesh of human bodies. This method has the ability to create full-body meshes in any pose and any angle of view. However, since they use a template of the human body, they cannot generalize to other objects without training. Also, the meshes do not show all features of

the person in the image. For example, the face does not look like the face of the person in the input image.

Kok *et al.* [37] designed FootNet. This method reconstructs feet from one to twenty RGB images. To do so, their first step is to segment the silhouette of the foot from all the input images using a convolutional neural network. Using only the silhouettes, their second step is to predict the PCA parameters of a foot model. In parallel, they predict a set of scale factors for their PCA model. Using the set of scale factors and the parameters of the PCA model, they are able to reconstruct the foot. This method is able to achieve impressive results from different angles of view. However, they need at least three images from different angles of view to achieve those results.

Betchtold *et al.* [6] designed a method that uses depth maps as input to generates a 3D mesh. It learns a hierarchy of priors at different levels of locality from the input depth map the same way inception networks do [77]. By extracting information at different scales, the method learns to faithfully recreate the visible areas of the depth map that can be found in the resulting mesh. From local and global priors, the method is able to recreate the hidden area to reconstruct a full 3D mesh. One strength of this method is that it recreates reliably the visible areas of the depth maps, which is not something that most methods are able to do. A second strength is their generalization capability, as their method can achieve a 3D reconstruction from a wide range of different objects without having to retrain their network. However, both those strengths lead the network to have issues with self-occlusion. For example, if one leg of a table is occluded, this method might recreate the table with only three legs as it does not have any visible cue of the existence of a fourth leg.

Song *et al.* [73] designed SSCNet. This method reconstructs 3D scenes. It projects a depth map of the scene into 3D space to get a 3D volume made of voxels. Using a 3D convolutional neural network, it employs local geometric and contextual information to generate a complete 3D scene where each voxel are labelled. More specifically, the network uses the probability of each voxel being part of a category to help the 3D completion process. This method recreates objects in the scene that were shown with only a few pixels of the depth map with surprising accuracy. However, as it classifies the object before completing them, some of them are modified into completely different objects. Also, some thin objects are forgotten due to being confused with what is behind it. For example, a painting on a wall might just appear as the wall in the reconstructed scene.

Genova *et al.* [24] designed LDIF. This method proposes a new 3D representation. Their new representation is a set of implicit functions that tell if a coordinate is inside or outside the 3D shape. With this set of functions and a learned algorithm, they are able to reconstruct 3D shape shown in a depth map with known camera parameters. They start

by projecting the depth maps into 3D space. Then, they use a neural network based on ResNet [29] and PointNet [62] to recover the set of implicit functions. This method has the advantage to be able to represent details without creating a shape that is larger than what appears in the input. Some reconstruction algorithms will create balloons when facing 3D shape with small holes in it, but this method does not. However, thin structures tend to disappear as the set of functions do not cover the small space where their shape exists.

Lunscher [44] designed a method that uses depth maps to reconstruct a 3D model of feet. To do so, they take as input a depth map of the foot using an angle of view that shows the interior of the foot. Then, a depth map from the other side of the foot, the exterior of the foot, is predicted. Combining those two halves projected in 3D space and applying a filter for noise, they are able to produce a point cloud of the foot. This method can recreate the foot with a length measurement that is precise enough for a shoe recommendation task. However, they have no constraint for how the predicted half and the input half are connected, making it difficult to have an accurate width reading.

Qi *et al.* [62] designed PointNet. This method takes a point cloud as input and generates a segmented point cloud as output. While this does not exactly meet our goal of 3D reconstruction, they do have an interesting approximation of a convolution operator. When working on non-Euclidean data such as point clouds, one way to be able to learn from your data is to approximate a convolution operator [10]. With a good approximated operator, a convolutional layer can be recreated. PointNet [62] does it by connecting every point of the point cloud to a shared multi-layer perceptron. This method has the advantage of allowing the learned model to get an understanding of the point cloud at a local level which can lead to a good 3D reconstruction when used for this task. However, the global information is lost when the point clouds go through the neural network.

Wang *et al.* [86] designed DGCNN. This method builds on PointNet [62]. They achieve the same task of classifying the points of a point cloud, and their contribution lies on its approximation of a convolution operator, called EdgeConv, which can be used afterwards for 3D reconstruction. Their operator works by finding the nearest points to each point of the point cloud. Then, they encode each point with the distances to the k nearest points, k being a free parameter. The encoding is done by a multi-layer perceptron. Since only k points are associated with each point, this method is able to learn local variations as well as global priors which in turn leads to better performances. However, due to using the free parameter, the method does not scale well with unseen examples. If the unseen examples have a large difference in their number of points compared to the point clouds used for training, then this method will not be able to learn their shape properly.

This section was a small literature review of 3D reconstruction methods. Most methods

presented do not generate the reconstruction directly. Methods that do usually cannot reconstruct the finer details of the object, inferring a blurrier version of it. Methods that do not reconstruct the objects directly from their input tends to have difficulties with complex objects. One way used to go around this problem is using a template, but it then affects the generalization ability of the method. We have seen many different kinds of method that uses multiple steps to reconstruct a 3D object. Some use pipeline and create byproducts that help generate the results, such as silhouettes, depth maps and joints' landmarks. Some produce parameters for 3D parametric shapes. Some methods presented take as input only depth maps or point clouds. While the methods processing point clouds as input cannot be directly used on our input, they can be applied to processed features. It is possible to generate depth maps and point clouds from RGB images as some methods presented in this section have shown. As for the other byproducts, multiple methods have presented that extracting silhouettes is not challenging. Joints' landmarks, which were useful for 3D reconstructions that show different poses, can be found with a pose estimation algorithm, which will be the subject of the next section.

2.5 Pose Estimation

As we have seen in the last section, joints' landmarks are useful when it comes to reconstruct an object in different poses [12, 15, 34] as we want to do with the foot. A set of joints can also help find the 3D pose of a reconstructed 3D object. These subjects will be covered in this section.

There are many ways to get the joints of an object in a RGB image. One way is to generate a set of feature maps, sometimes called confidence maps or heat maps, that indicate the probability in 2D space of a joint landmark [13, 15, 39, 47, 48, 52, 63, 70, 74, 83, 96, 101]. Each confidence map produces the location of one joint. Convolutional neural networks excel at creating feature maps, which is one reason explaining the popularity of confidence maps. From the confidence maps, algorithms use patterns of all sorts to generate the final pose estimation. While this technique is efficient, it often fails when occlusions occur. They also have minimal ways of telling when the object is not in the image.

Another popular way of locating the joints in the image is to detect the structure of the object [13, 16, 17, 35, 84, 85, 97, 99, 103]. If the global structure can be found, then the joints are along this structure. This technique is more resilient to occlusion and is better at telling if an object is in the image or not. However, the more complex the object is, the more difficult it is to extract its structure from the image.

From the joints' landmarks, many methods are able to extract a 3D pose using a specific joints' layout. The most popular task is full human body pose estimation. To do so, there are mostly two techniques. The first creates a 3D object out of the input [15, 34, 99, 103], usually a mesh. Using regressors or by recognizing features on the 3D object, the 3D pose can be estimated. The accuracy of this technique depends on the accuracy of the 3D reconstruction. The second technique infers a 3D pose estimation from the found 2D joints [17, 35, 45, 47, 48, 50, 70, 83, 84, 96, 97, 101]. To do so, some methods rely on the structure of the known object. For example, knowing that the elbow only bends one way, the pose estimation should not estimate an arm bending the wrong way. Some other methods will rely on cues from the input image to put the 2D joints in context. Either way, the technique relies on the accuracy of the found 2D joints.

No matter what techniques are used for 3D pose estimation, the main variation in the methods come from the training data. A lot of methods are designed on their own dataset, meaning that they present an accuracy that is controlled by the designers of the method. A reason to create a dataset is that the method cannot be tested thoroughly using the existing datasets. This is why training data will be covered in the next section.

2.6 Training Data

To be able to train a learned approach, training data is necessary. A dataset needs to be able to provide labels for our two goals: 3D reconstruction and 3D pose estimation of feet. In an ideal case, we would have used real life data of feet. However, there is none available in the public space for supervised training. We then looked for synthetic datasets of feet. Such dataset, or combination of datasets, does not exist in the public space to our knowledge. The only dataset that applies to feet is presented by OpenPose [13] and consists of feet labelled with three 2D keypoints. This is not enough for our needs. We have also looked for training data in a more popular area, full human body shape recovery and pose estimation [31, 32, 41, 47, 48, 58, 60, 61, 65, 70, 71, 82, 83, 95, 98, 101]. They did not do either. The feet were described by one to three keypoints. The number of vertices in the 3D mesh of feet is low, making them unrealistic. There is almost no variation in the feet from one mesh to the other. The pose of the feet is usually all the same, or show fewer than ten different poses in thousands of meshes. Since we were unable to find a dataset to train our dataset, we chose to build our own. This section will cover the necessary background to do so.

2.6.1 3D Mesh Foot Model

To create the needed training data for our system, we need to have a representation of the shape and for the pose. This subsection will discuss the making of a 3D mesh foot model to represent the shape of the foot.

The 3D mesh foot model needs to be easily morphable so that we can describe many feet with the same model. We explore two techniques to deform meshes. The first technique is to deform the meshes by acting directly on the vertices [2, 53]. Using affine transform matrices to modify the meshes at a vertex level, we can shape them in any way we want. However, to keep the quality of the meshes intact, a set of constraints needs to be applied to the affine transform matrices so that the vertices move towards their expected positions. With this scheme, meshes can be deformed. To be able to generate a wide variety of meshes of feet, we need to be able to steer the deformation towards specific variations. To do so, this technique needs to be paired with a parametric model such as a PCA model [2]. If the components of the PCA model are linked to the vertices and control their movements, then we can specify the variations needed.

The second technique that can be used to deform meshes is to parameterize surfaces or shapes according to our 3D representation [30, 76, 100]. With a parameterized object, it is easy to obtain variations of it. Compared to the previous technique where the vertices have to be constrained, the parameterization already make sure that each vertex follows the modification of the mesh. However, this technique can become heavy to compute when the target meshes become more complex. Some artefacts can also occur, such dulled edges and blurry features.

While both techniques have their disadvantages, they both are capable of achieving the task that we need it to do. The choice of a technique will come to how easy it is to combine it with the results of the next section which address the pose.

2.6.2 Kinematic Foot Model

The kinematic foot model is to be used for describing the pose of the foot. It needs to be able to replicate all poses a foot can perform while being constrained against making movements that are not possible by a healthy foot. To do so, the model has to be based on the biomechanics of the foot. The model also needs to be based on features of the foot that can be seen from the exterior as it will describe the joints mentioned in 2.5 Pose Estimation.

There are a lot of biomechanical models of the foot. They are mostly used to understand foot injuries. Bishop *et al.* [8] did a survey of them to see which ones were thoroughly tested clinically. Only two came out, the Milwaukee model [36] which is used for the adult population, and the Oxford model [14] which is used for the paediatric population. However, many new models have been created since then. Those models are presented in the following. To be able to follow how the models are made, Figure 2.5 shows the bone structure of the foot and the ankle.

The Milwaukee foot model [36] is made of four segments. Each segment is made of bones that have a considerable impact on the kinematic of the foot. The first segment is comprised of the tibia and the fibula. This segment represents the leg and stops at the ankle. It creates a joint that shows the rotation of the ankle. The second segment is made of the calcaneus, the talus and the navicular. This second segment is the heel, and stops under the ankle. It shows the movement of the arch of the foot with the help of the third segment. The third segment is formed of the cuneiforms, the cuboid, and the metatarsals. These bones start under the ankle and stop right before the toes. It shows the movement of the arch of the foot with the help of the second segment. The fourth segment is constituted of the proximal phalanx of the hallux. The hallux is the great toe, and the proximal phalanx are the first set of bones in the toes. This means that only the first bone of the great toe is considered in this model. It is assumed that the other toes follow the great toe as if it was a rigid block. It shows the rotation of the toes around the end of the third segment. The goal of this foot model is to be able to describe the dynamic characteristics of the foot and ankle structure during ambulation.

The Oxford foot model [14] is similar to the Milwaukee foot model [36]. It is also made of four segments. The first segment is the same, being comprised of the tibia and the fibula. The second segment, called the hindfoot, is similar, being made of the calcaneus and the talus. However, the navicular is part of the third segment compared to the Milwaukee foot model [36]. The third segment, called the forefoot, also includes the cuneiforms, the cuboid, and the metatarsals. The fourth segment, called hallux, comprises all the bones in the toes. This is not to be confused with "hallux" being another name for the great toe. The model is based on rigid body assumptions. This means that each segment is not supposed to move within it. The toes are then considered to be able to move only where they are attached to the foot and not in the middle of them. The joints formed by each segment of this model are the same as the Milwaukee model [36]. The Oxford foot model [14] is made to describe the mechanisms that transmit motion between segments, especially between the hindfoot and the forefoot.

One of the newer models is the Salford foot model [51]. It is a six segments foot model. The first segment is the same as the Milwaukee foot model [36] and the Oxford foot model

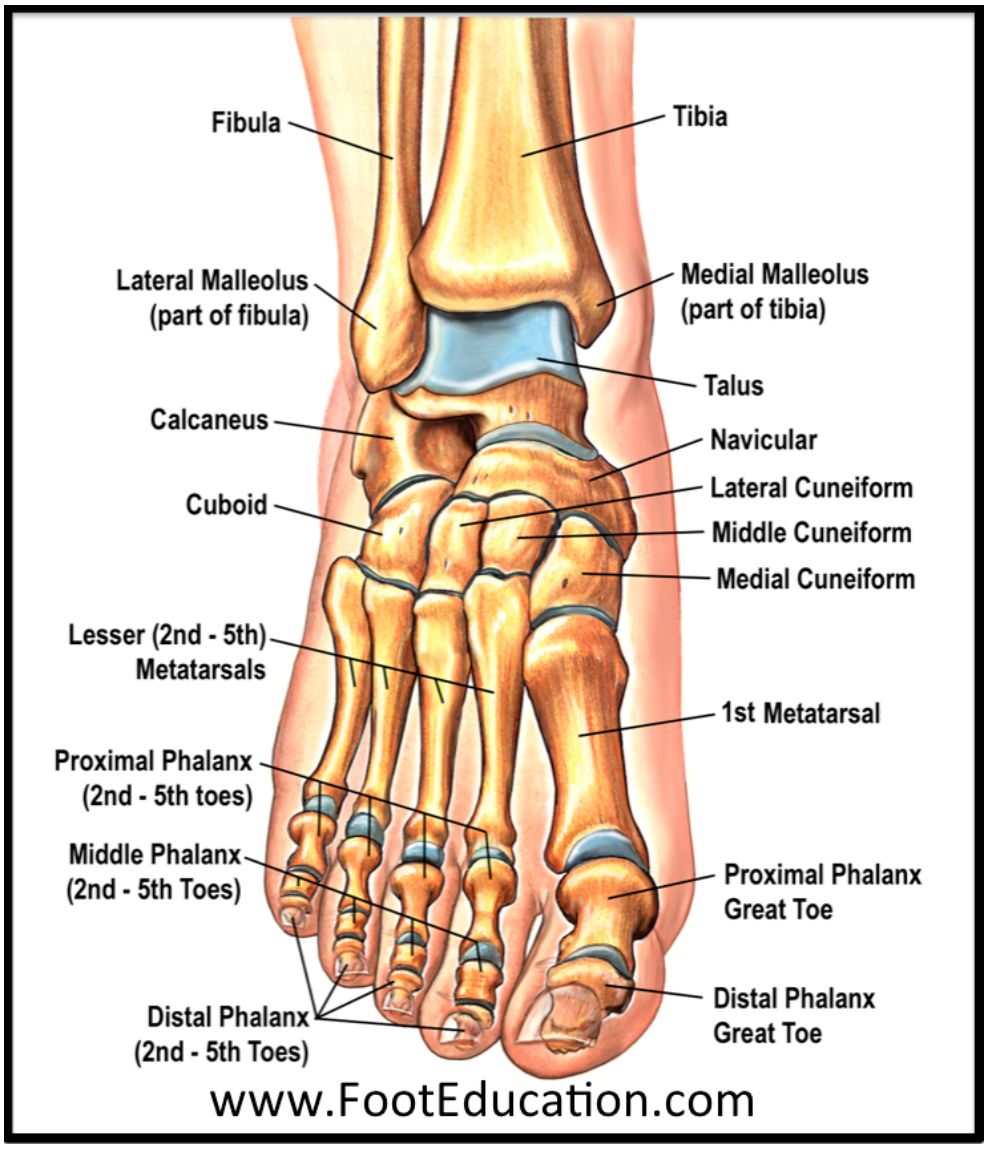

Figure 2.5: Bones of the foot and ankle [55]

[14]. It is made of the tibia and the fibula, and represents the leg that begins at the ankle. The second segment is only the calcaneus, which is the bone that makes up most of the heel. It is located at the far end of the foot and part of it is under the ankle. The third segment, called midfoot, is comprised of the navicular and the cuboid. These bones are next to the calcaneus and can be located right in front of the ankle. These three segments together form the joint that shows the rotation of the ankle. The fourth segment, called the lateral forefoot, is made of the fourth and the fifth metatarsals. The metatarsals link the cuboid and the navicular to the toes. The fifth metatarsal link to the smallest toe. The fourth metatarsal is right next to it. The fifth segment, called the medial forefoot, is also made of only one bone, in this case the first metatarsal. This metatarsal links to the big toe. The fourth and fifth segments together form a joint that controls the movement around the length of the foot. With the third segment, they also form a joint that controls the movement of the arch of the foot. The sixth and last segment, called the hallux, is comprised of all the toes. As in the Oxford foot model [14], the hallux is considered a rigid block. This segment creates a joint that shows the movement of the toes. The goal of the Salford foot model [51] is to track pain-free movement in an adult foot. Compared to the Milwaukee foot model [36] and the Oxford foot model [14], the Salford foot model [51] only uses features that can be seen from outside the foot to build the foot model. It is worth mentioning, however, that the Oxford foot model [14] only needed X-rays for particular cases of deformation of the hindfoot.

Every foot model seen so far tried to divide the foot in segments to simplify its kinematic. This is not the case with the Glasgow-Maastricht foot model [54]. This foot model has 26 segments. There are 26 bones in a human foot, and each segment of the Glasgow-Maastricht foot model [54] represents one bone. This model also has constraints between each segment for realism. The goal of the Glasgow-Maastricht foot model [54] was to create a model that could represent any movement of the foot and any injuries that could occur to the foot.

2.7 Summary

This chapter offered a general background to understand the choices that were made to design our system. Feet measurements were discussed to explain the need our system is filling. Notions of 3D projection and 3D geometry were detailed as a basis for 3D reconstruction. The problematic of 3D reconstruction was conveyed before a brief literature review on the subject. Pose estimation followed as the 3D pose estimation task can be achieved complementary with the 3D reconstruction task. Finally, the lack of an available

dataset was discussed. We have decided to create our own training data. Relevant notions were exposed on this subject. With this background in mind, we present our solution in the next chapter.

Chapter 3

Methods and Algorithms

3.1 Pipeline Overview

The task of extracting the 3D mesh and pose of a foot from a single image is complex. To simplify it, we have built a pipeline that separates the complex process into smaller and easier tasks. There are three different types of input possible for our pipeline: RGB images, RGBD images, and depth maps. RGB images are images with three colour channels. Depth maps are images with one channel where each pixel describes a distance between the camera and the captured scene. RGBD images are images with four channels, the first three channels being the same as for RGB images and the fourth channel being a depth map.

Instead of directly inferring a 3D reconstruction from our input, which is an ill-posed problem as seen in the last chapter, we have decided to first start by acquiring two intermediary byproducts from the input. The first intermediary result is a silhouette of the foot, which shows the outline of the foot in the input. It allows us to discard the background which does not contain any information about the foot. The second intermediary result is a set of 2D joints, which describe the pose of the foot in the input. It allows us to find keypoints of the foot which will help to direct the 3D reconstruction and to facilitate the 3D pose estimation. The 2D joints are the reason why this approach is not as ill-posed as a direct inferring of the 3D reconstruction from the input. While the latter would have trouble to estimate the hidden side of the foot, the use of 2D joints, paired with a template, allows us to predict accurately the foot from any angles. The landmarks of the 2D joints will have to be carefully chosen to make sure that we can estimate the orientation of the foot in the input in a way that does not allow multiple solutions.

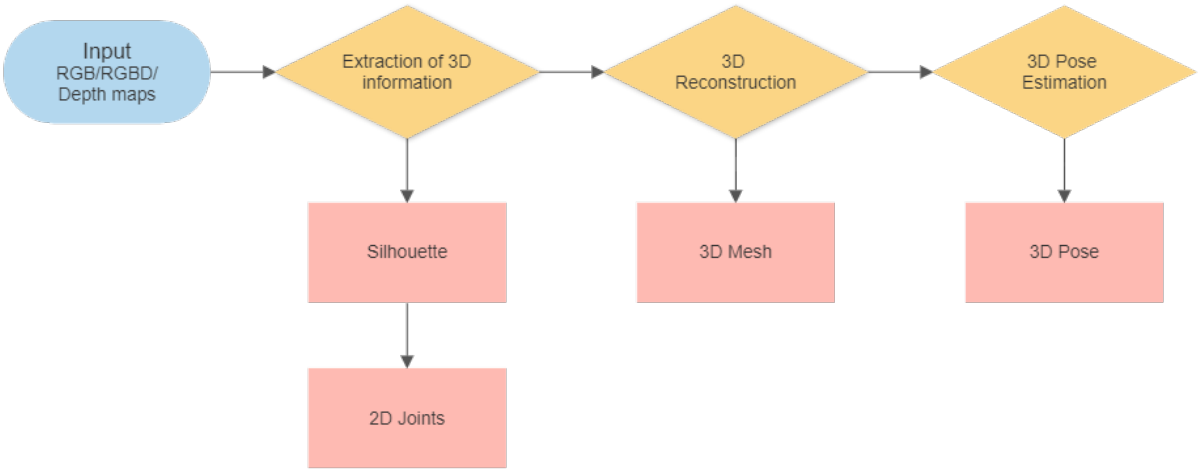

Figure 3.1: Overview of the pipeline. The rounded rectangle represents the input which can be either RGB images, RGBD images, or depth maps. The diamonds represent the modules of the pipeline, which are described in this chapter. The rectangles represent the outputs of the system.

Based on the two intermediary byproducts, we can achieve the 3D reconstruction of the foot. To do so, we have experimented with two methods. The first method is EdgeConv [86] which predicts point clouds. The second is Inception Spiral Module [15] which predicts meshes. For the 3D pose, we estimate it using specific vertices of the 3D mesh produced during 3D reconstruction. This is where the pipeline end. At this point, we have from the input: a silhouette, a set of 2D joints, a point cloud or a mesh depending on the algorithm used for 3D reconstruction, and a set of 3D joints describing the 3D pose.

To train our network, we have created our own training data as there was no suitable dataset available in the public domain. We have created two models to represent the foot in our training data. The first one is a 3D mesh foot model made by deforming meshes using affine transformations at the vertices level, followed by the development of a Principal Component Analysis (PCA) model. It is inspired by the method of Allen *et al.* [2]. The second model is an 18 joints kinematic foot model that we developed based on popular biomechanical foot models to be able to describe the 3D pose of the foot. The biomechanical foot models that inspired our kinematic foot model are the Milwaukee model [36], the Oxford model [14], the Salford model [51], and the Glasgow-Maastricht model [54]. With this training data, our pipeline is trained end-to-end. We have implemented it using

PyTorch [57].

3.2 Extraction of 3D Information

This section describes the first step of the pipeline. The goal of this step is to acquire 2D cues of the 3D object from the image, which are the shape and the pose [15]. The 3D object in our context is a foot. To obtain the 2D cues, we first have to extract the silhouette of the foot from the image which represents the shape. This can be done with a segmentation algorithm. We also have to get the joints of the foot from the image. They represent the pose. The joints can be found using a pose estimation algorithm. We have chosen to use the Stacked Hourglass Network [52] for its flexibility. It was also the algorithm chosen by Chen *et al.* [15] and Martinez *et al.* [45] due to its precision and its reliability. Although the Stacked Hourglass Network method was developed for estimation of human pose estimation, its design is suitable for joint detection such as what we are trying to do. It is developed to generate confidence maps. The joints we are looking for correspond to the joints of our kinematic foot model which is presented in 3.5.2 Kinematic Foot Model. An illustration of our kinematic foot model is presented in Figure 3.9.

The Stacked Hourglass Network [52] is comprised of a series of encoder-decoder networks that predict confidence maps. Each network has the goal of refining the pose detection through reevaluation of features and initial estimates. This process can be seen as an iterative algorithm with a fixed number of iterations. The operation of going back and forth between scales helps preserve the spatial location of features that can be lost in neural networks with a large number of layers. The first hourglass network processes the image to bring out the interesting information. By doing so, it also segments the foot from the background. From this segmentation, the silhouette can be obtained. The next two hourglasses then refine the output of the previous one to get the joints location. This architecture is schematized in Figure 3.2. The output appears as a set of heat maps. Each heat map predicts the confidence of the position of one joint.

Another advantage of using multiple encoder-decoder networks is that it allows us to introduce intermediate supervision in the training of the network. Intermediate supervision is the process of comparing features generated from inside the network with labels to assess the progress in the training of each encoder-decoder network. This stratagem helps to propagate the loss throughout the whole system. We use the binary cross entropy loss to learn where the area of interest is. There are only two choices in the heat maps: if the pixels are of interest or not. For example, if there is a joint at that specific location or not. This is why we went for a binary classification loss. We chose specially the binary cross

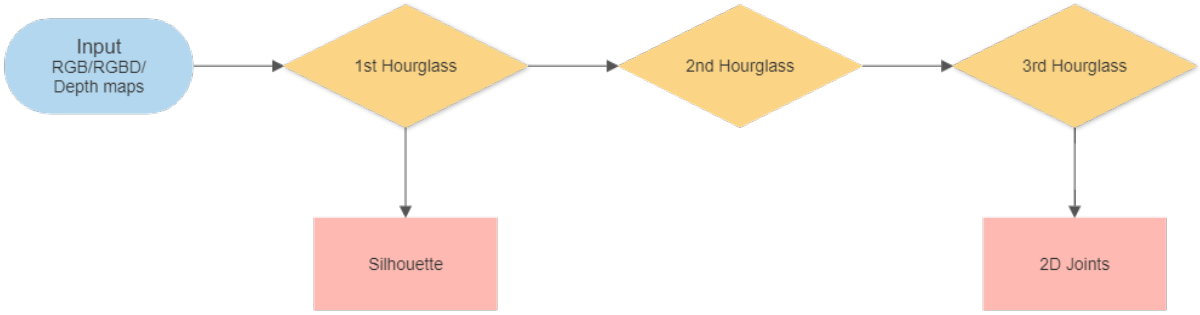

Figure 3.2: Illustration of the module that generates the 2D cues, which are the silhouette and the 2D joints. The rounded rectangle represents the input which can be either RGB images, RGBD images, or depth maps. The diamonds represent encoder-decoder networks. The rectangles represent the outputs of the module.

entropy for its stability and efficiency. It also provides a maximum likelihood estimation with a strong cost function that fits well with the problem we are solving as the joints' landmarks are not always highlighted by strong indicators in the image.

The architecture we use for our encoders is based on the backbone of ResNet-18 [29]. The architecture of ResNet-18 is presented in Figure 3.3. ResNet-18 has shown great results in acquiring information from images compared to other architectures with large networks due to the residual layers that help the propagation of the loss on the weights. These residual layers of different sizes also allow the network to capture information at every scale of the image. Global evidence of the object in the image is important to understand the whole shape of the foot. It especially helps understand the silhouette of the foot and the relationship between the joints. Local information is also important as it allows the network to capture small variations in the image that indicates the location of interesting features. Those features are crucial in finding the position of the joints of the foot. Using the backbone of ResNet-18 also allows us to start with pretrained weights for an easier convergence. Those weights were pretrained on ImageNet [66] which is a dataset oriented toward image classification. It allows the backbone to efficiently extract features in images. The pretrained values serve as initialization for the training of the encoders of our system.

ResNet-18 is comprised of four residual layers, one lone convolutional layer and one fully connected layer. A residual layer is made of two blocks containing each two convolutional layers. The lone convolutional layer comes first. It generates feature maps which will be used by the residual layers. Before going to the residual layers, the feature maps go

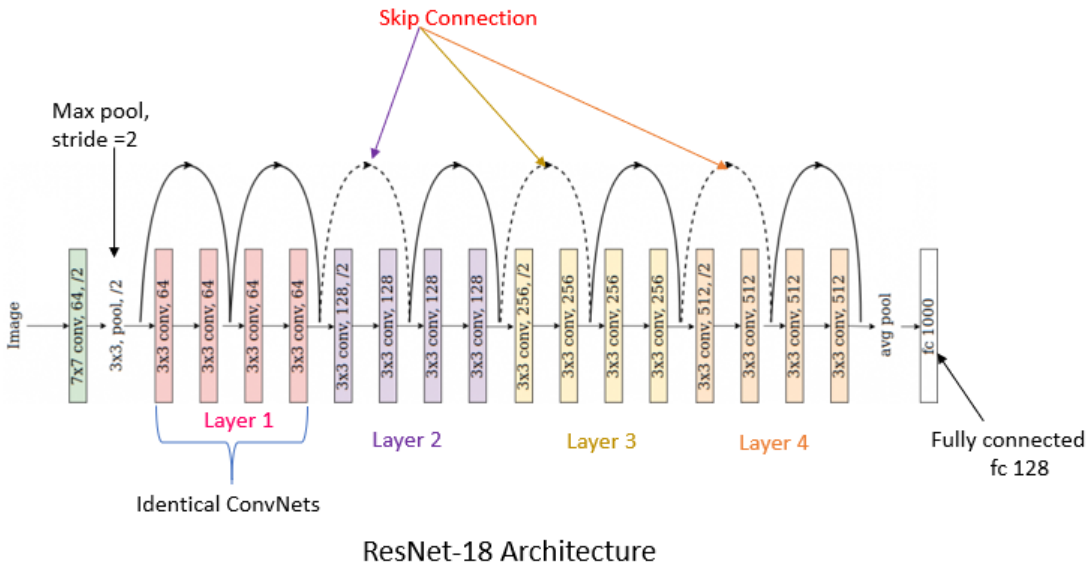

Figure 3.3: Architecture of ResNet-18 [29] as illustrated by [72], used as encoder for each hourglass network designed for our system.

through a max-pool layer. This operation samples from the feature maps only the relevant information and reduce the size of the maps, which reduce the number of parameters of the network to reduce the computational cost. After comes the residual layers. Each is composed of two convolutional blocks which are in turn composed of two convolutional layers of the same size. The input of a block is added to the output of the same block, as shown in Figure 3.3 as skip connections, thus keeping the global information alive through the network. It also helps the propagation of the loss through the network as the output of the first layer is only a few additions away from the output of the network. In the case of the conventional convolutional neural network, the loss needs to be propagated through multiple convolution operations. An addition is a simple operation while a convolution is a much more complex one, and many complex operations tends to reduce the impact of the loss as it goes through. This effect is known as vanishing gradient. The skip connections of the residual layers make sure that the gradient does not vanish as the loss is propagated in the system. After the last residual layer, an average-pooling layer is used to sample the results of the blocks. An average-pooling layer has the same purpose as a max-pooling layer, but gives smoother results. It is useful to keep the trends in the feature maps, while the max-pooling is better at highlighting the important features. The last layer of the

network is a fully connected one. It is there to find patterns of interests in the feature maps and create a latent space where those patterns are accessible for a decoder network to handle.

The decoder is simpler as it is not used in a task as complex as extracting features from an image. We want to generate a set of confidence maps that shows which landmark from the image is associated with a specific joint. Those landmarks were already processed by the encoder, so the purpose of the decoder is only to select the proper ones. We are looking for 18 confidence maps, one for each joint of our kinematic foot model. To create these confidence maps, we have designed our network with five convolutional layers. The input of those layers are first up-sampled using a bilinear interpolation algorithm. The idea is to enlarge the input of the convolutional layers so that the ratio of sizes between the kernel and the features is more appropriate to select the right landmarks at a local level. This means that the convolution layers of the decoder are not up-convolutions. They do not need to learn how to generate larger output than their input which can result in unwanted artifacts. The bilinear interpolation will give an output blurrier but more predictable than the output of an up-convolutional layer. Then, for the first four convolutional layers, we concatenate the interpolated input with the output of the residual layer of the same size from the encoder. This residual layer's output is taken after the sum with the skip connection. The goal of adding information from the encoder to the decoder is that the encoder has global information from the image that does not make it to the latent space between the encoder and the decoder. By using this scheme, we are able to bring back relevant global information that is crucial in selecting the right landmarks to find the position of the joints. This idea was suggested by Stacked Hourglass Network [52]. As for the fifth convolutional layer, the interpolated input is fed to the layer without any more modifications. Its purpose is to separate everything learned from the previous four layers into 18 different confidence maps. We then use a sigmoid layer to highlight the chosen features and prepare the output for the binary cross entropy loss discussed earlier.

To be able to accept an image with a different number of channels, we have decided to modify the input of the ResNet-18 architecture. It will allow us to use RGBD images which have four channels. The depth part of those images will be most helpful as the silhouette will be easier to segment from the background. It will also give more information to infer the location of the joints. The same goes for depth maps, which are only one channel. By changing the number of channels, we are taking away our ability to use pretrained weights. It is not a setback as it only means that we will need to expect more training for each network that does not have an input with three channels.

3.3 3D Reconstruction

Once the silhouette and the 2D joints are acquired, we can reconstruct the 3D mesh. To do so, we have experimented with two different algorithms. The first one is called EdgeConv and was designed by Wang *et al.* for Dynamic Graph CNN [86]. It will be presented in section 3.3.1 EdgeConv. The second one is called Inception Spiral Module. This algorithm is based on the work of Gong *et al.* for SpiralNet++ [25] and their learned operator is called SpiralConv. HandMesh [15] has upgraded it into Inception Spiral Module based on inception networks [77]. SpiralConv will be presented first, followed by the upgrades that makes it into Inception Spiral Module. It can be found in section 3.3.2 Inception Spiral Module. Both those operators are approximations of a convolution in a non-Euclidean domain [10].

Every 3D reconstruction methods presented in this chapter work on meshes. It is then necessary to have a definition for it. A mesh M is a 3D representation that consists of a set of vertices V and a set of faces F. Each vertex is a coordinate of three dimensions: $\{\mathbf{v}_i = (x_i, y_i, z_i)\}_{i=1}^{n}$. For the purpose of this thesis, we will assume that the meshes all have triangular faces, which means that each polygon is connected to three different vertices. This is the smallest number of connections that a face can make, and it is the most versatile shape of faces which is the reason why we chose it. Faces with more anchors, such as quadrilateral and hexagonal faces, will not show as much flexibility for the construction of the mesh, but will be less affected by noise than a mesh with triangular faces.

3.3.1 EdgeConv

The idea behind EdgeConv is to make a graph out of the vertices based on geometric location. Dynamic Graph CNN [86] wanted to exploit the local geometric structures that comes out of the sparse data of the vertices. To do so, they construct a local neighbourhood graph. A patch around one vertex of this graph is illustrated in Figure 3.4. Then, they apply an approximated convolution operator to the edges of the connecting neighbouring pairs of vertices in the spirit of graph neural networks. However, compared to regular graph neural networks, EdgeConv computes the graph for every layer. The reason behind this dynamic graph, thus the name, is that the geometric distance between the features will change after passing through each layer. This means that the graph should change accordingly so that every feature of the graph is connected to the geometrically closest features after every layer. Since the layers affect the distance of the features, the model learns how to make the graphs. This is useful to us as it means that the model can learn to create new shapes out of their input.

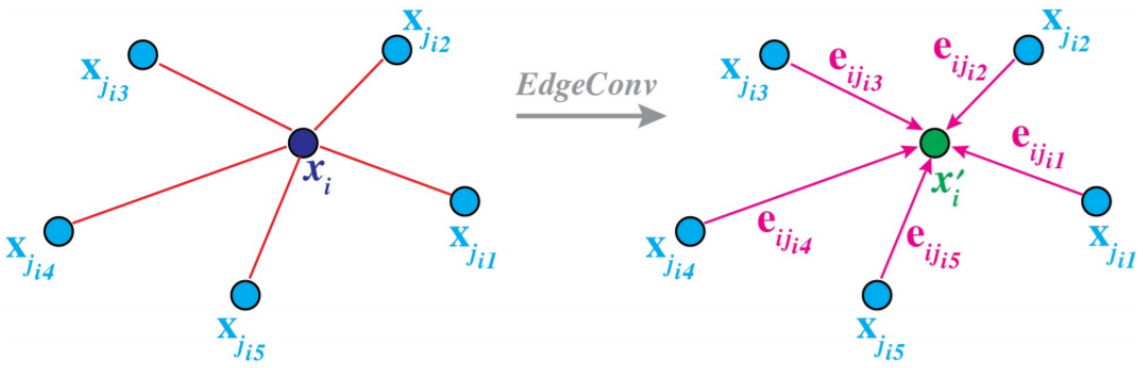

Figure 3.4: Patch around one vertex of the graph built by EdgeConv as illustrated by Dynamic Graph CNN [86]

To construct the graph where the nodes are vertices and edges are the shortest distance between those vertices, we start by computing the pairwise distance from each vertex to every other vertices in the mesh. This can be done with equation 3.1.

$$d(i,j) = ||\mathbf{v}_i - \mathbf{v}_j|| \qquad i,j = 0,1,...,n \tag{3.1}$$

In equation 3.1, d indicates the pairwise distance function, and n indicates the number of vertices. As for i and j, it is worth mentioning that we allow $i = j$. This means that, in the graph we are building, each node also points to itself. Those self-loops are important because EdgeConv learns from the edges between the vertices and not from the vertices themselves. To acknowledge every vertex, we need to have an edge that loops from a vertex to itself so to be captured by the network. As an analogy to classic 2D convolutions, the self-loop is the centre of the kernel, while the edges that go from that vertex to the other nearest neighbouring vertices are the patch around the centre of the kernel. A method that would directly input the vertices in a linear layer, such as PointNet [62] is a variation of EdgeConv where only the self-loop edges are computed. This could be compared to a classic convolutional layer with a kernel of size 1x1. One advantage of EdgeConv over this approach is that the receptive fields of its neural network is as big as the set of vertices V without losing sights of the vertex level. The receptive fields of PointNet is at the vertex level, which means that this method only takes the local information of its input in consideration. A receptive field is the size of the input that generates one feature of the feature maps that are generated by a layer of the neural network [3]. When computing the

edges, equation 3.1 is very simple but not very useful in an applied algorithm. We decided to use the form shown in equation 3.2.

$$d(i,j) = \mathbf{v}_i^2 + \mathbf{v}_j^2 - 2\mathbf{v}_i\mathbf{v}_j \qquad i,j = 0,1,...,n \tag{3.2}$$

Equation 3.2 is the developed form of equation 3.1 when using a Euclidean norm to compute the edges as a geometric distance between two vertices. With the pairwise distance of all possible pairs of vertices in the mesh, a nearest neighbours algorithm can be applied to get a patch of the nearest vertices for each vertex. For a patch, the edges between the vertex in the centre of the patch and the other vertices of the patch are the smallest in a geometric sense. The number of edges chosen with the nearest neighbours algorithm is a hyperparameter of the model. Each vertex has one patch centred around it, but can be part of multiple patches. Every patches are then aggregated together and fed to a linear layer which will learn the shape of the mesh. This small pipeline, computing the pairwise distance, finding the nearest neighbours and applying a linear layer, is an approximation of a convolution operator. The nearest neighbours algorithm creates the kernel and the linear layer learns from it, while the pairwise distances give sense to the sparse data so that the shape can be understood by the network. We have already mentioned that some algorithms have similar approaches. For example, PointNet [62] only uses the linear layer, which means that their approximation of a convolution operator can only take the smallest size possible. Dynamic Graph CNN [86] and PointNet [62] also suggests concatenating the output of each layer together before using a last linear layer so that the model can learn from the mesh at different scales. However, they use their model for classification of meshes. Since we are using our model for 3D reconstruction, we have modified their model by adding up-sampling layers with the same approximated convolutional operator at the end of their architecture to predict a set of vertices V instead of a classification score.

To train the neural network described, we chose to use the Chamfer distance [93] as loss. The Chamfer distance can be described as the average mismatch between two point clouds. It is a way to detect the differences between the predicted point cloud and its ground truth. We have decided to use this function as we do not expect our results to be exactly like the ground truth. We do not mind if the points are not at the exact same locations, as long as they are on the surface of the point cloud to describe the same foot as the ground truth. Using a different loss function such as a L1 loss, we would be penalizing at a local level the position of the points. With the Chamfer distance, we are looking at a global level if the two point clouds look alike.

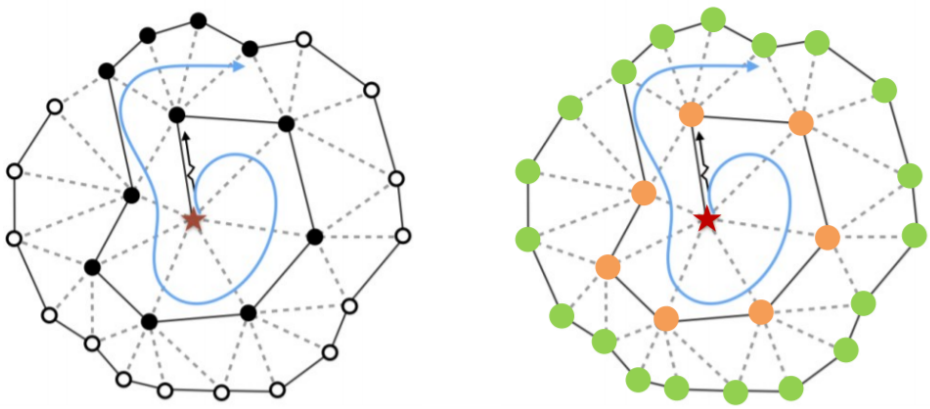

Figure 3.5: Example of spiral sequences from SpiralConv as illustrated in [15] and [25]. **Left**: Illustration of SpiralConv. **Right**: Illustration of Inception Spiral Module with hierarchical receptive fields (depicted in red, orange, and green).

3.3.2 Inception Spiral Module

The idea behind SpiralConv is to create a graph out of the vertices of the mesh, and using it for a graph-based convolutional operator to learn from the mesh. We want to create a graph that allows us to understand the shape of the mesh. To do so, SpiralConv orders the vertices in the mesh in spiral sequences as illustrated in the left side of Figure 3.5. The ordering process starts by finding all the adjacent vertices for each vertex. The values of the faces F are helpful to find the adjacent vertices as each polygon is represented by its connection to three different vertices as we are using triangular meshes. We start by defining a neighbouring *ring* centred on one vertex $\mathbf{v}$ with equation 3.3 and equation 3.4.

$$0\text{-}ring(\mathbf{v}) = \{\mathbf{v}\} \tag{3.3}$$

$$(k+1)\text{-}ring(\mathbf{v}) = N(k\text{-}ring(\mathbf{v}))/k\text{-}disk(\mathbf{v}) \tag{3.4}$$

Equation 3.3 shows the null case of a neighbouring *ring*. In equation 3.4, $N(V)$ represents the set of all vertices adjacent to any vertex in the set V. $k\text{-}disk(\mathbf{v})$ can be described as follows:

$$k\text{-}disk(\mathbf{v}) = \cup_{i=0,\dots,k} i\text{-}ring(\mathbf{v}) \tag{3.5}$$

The equation 3.5 shows that $k\text{-}disk(\mathbf{v})$ is the union of multiple neighbouring *rings*. This disk represents the mesh with vertices ordered as necessary to understand the shape of the mesh. Each ring within the disk has the same length, which is a hyperparameter of the model. If the number of vertices of the mesh is not a multiple of the ring's length, the last vertices that cannot get into the last ring are ignored. The number of rings within a disk is defined by the length of the ring and the number of vertices in the mesh. The disk is computed only once before we use it to learn from the mesh. Now, if we introduce $f(\mathbf{v})$ being the node feature of a vertex as seen in graph convolutions, we can create a learned operator. In thus, SpiralConv can be described as follows:

$$SpiralConv(\mathbf{v}) = Wf(k\text{-}disk(\mathbf{v})) + b \tag{3.6}$$

Equation 3.6 shows how SpiralConv can be used as a learned operator with W and b being the weights and the biases, respectively. Algorithm 1 describes how the ordering of the vertices is done.

Algorithm 1 Algorithm to order the vertices for SpiralConv, with length(ring) being the desired ring length set as a hyperparameter.

```
for each vertex v_0 in mesh do
    Spiral ← Index(v_0)
    while length(Spiral) < length(ring) do
        Get vertices v_n adjacent to v_0 using polygons' edges
        for v in v_n do
            if Index(v) not in Spiral then
                Spiral ← Index(v)
            end if
        end for
    end while
    if length(Spiral) == length(ring) then
        Disk ← Spiral
    end if
end for
```

One flaw of SpiralConv is that the receptive fields are the size of the length of one ring, which is too small to have a good understanding of the general shape of the foot. One simplistic way to solve this problem would be to greatly increase the length of the rings. However, this scheme comes with many drawbacks. Using a ring's length that is appropriate to extract information about the general shape of the foot leads to it being

too large to get the local information. Also, since we ignore the vertices that do not fit in the last ring if the number of vertices is not a multiple of the ring's length, it could lead to ignoring a significant number of vertices. To enhance the receptive fields of the mesh, SpiralNet++ [25] has decided to dilate the spiral convolution. They did that by skipping every other vertex in the spiral sequence. The number of vertices in the sequence does not change, the ring simply considers vertices farther from the original one. This scheme expand the receptive fields without sacrificing local information. However, it does not enhance the receptive fields enough for meshes with several thousand vertices which is our case.

HandMesh [15], on the other hand, decided to get inspiration from inception networks to enhance the receptive fields. Inception networks [77] are convolutional neural networks that apply filters of different sizes on its input. For example, a convolutional layer of an inception network could take for input an image. Then, it would apply to the image convolutions with a kernel of size 1x1, 3x3, and 5x5. The feature maps that result from the convolutions are then put back together with some sort of filter concatenation. This operation has for effect to extract features from the input at different scales. In a classification task, sometimes the same object appears in two different images but at different scales. For example, in one image, the object could take the whole image while in the second image, it takes only a tenth of the image. Since it is the same object, we expect to classify it in the same category. Using different sizes of filters can make it more effective for the model to single out the object and properly classify it. The receptive fields are widened as there are multiple filters of different sizes applied to the input. With this idea in mind, HandMesh decided to use segments of the rings of the spiral sequences to create different filters. An illustration of this concept can be seen on the right side of Figure 3.5. Each segment chosen has a different size and is used as a filter, much like the kernel of a sliding window in a convolutional layer. The first filter has only a size of one and can be compared to a convolution with a kernel of size 1x1. This is the filter that extract features at the smaller scale, yielding local information. The second filter is comprised of the first third of the spiral sequence. It also extracts local information, while not being as close to the mesh as the first filter. The third one uses the two third of the sequence and is designed toward extracting global information out of the mesh without losing sight of the local information. The fourth and last filter uses all the vertices of the spiral sequence. It mainly gives global information on the mesh. It is a SpiralConv operator as presented by SpiralNet++ [25]. They call their algorithm Inception Spiral Module. The first three filters are concatenated together. They are then summed with the fourth filter. With this scheme, Inception Spiral Module can use longer ring's length without having the drawbacks described earlier.

To get the best results, we have chosen to use Inception Spiral Module from HandMesh

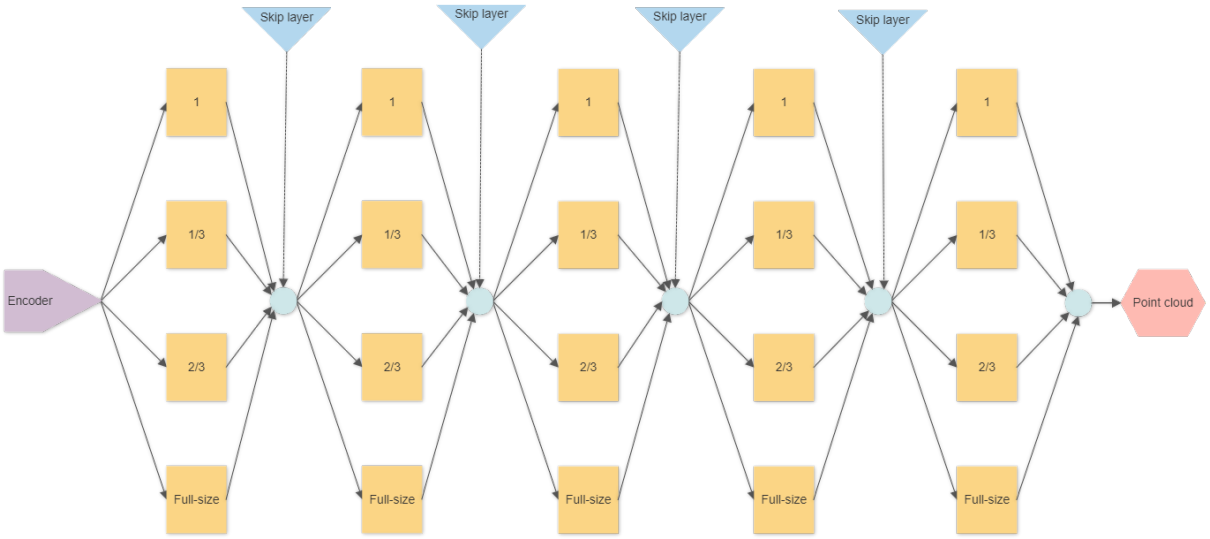

Figure 3.6: Illustration of the decoder that generates the 3D reconstruction using Inception Spiral Module. The pentagon represents the encoder which is the same as the encoders described in 3.2 Extraction of 3D Information. The encoder has for input the 2D joints. The rectangles represent filters of Inception Spiral Module paired with a multilayer perceptron. Filters labelled "1" only take the first values of each ring of the SpiralConv operator. Filters labelled "1/3" and "2/3" take a fraction of the values of each ring. Filters labelled "Full-size" take the full size of the ring. Each column of filters represents a layer. The input of each filter of the 5th layer has a size equal to the number of vertices in the template. The 4th layer has half that size as input, the 3rd layer a quarter, etc. The circles represent concatenation operations, to which we add the skip layers represented by triangles. The output of the network is illustrated by a hexagon and is a point cloud. Adding the faces of the template to this point cloud will recreate the 3D reconstructed mesh.

[15]. The first step is to extract the spiral sequence from a mesh, which is done using OpenMesh [9] and Scikit-Learn [59]. We have chosen to use the mesh model described in 3.5.1 3D Mesh Foot Model as a template mesh. This template is the 3D mesh of a foot. Inception Spiral Module learns to deform this mesh into the foot that is seen in the input image. With the spiral sequence, we can then start to create our neural network. However, since we are building a decoder, we need to be able to create layers of different sizes. To do so, we use the algorithm QSLIM [23] to scale down the template. QSLIM is an algorithm made to simplify meshes. It works by removing vertices while minimizing the movement of the edges. It also reconfigures the polygons so that the unnecessary vertices can be removed right away. We are looking to reduce the number of vertices by half with each scale down. The amount at which we choose to scale down the meshes is a hyperparameter of our model. With scaled down meshes, we can acquire spiral sequences from meshes of different sizes. It allows us to create layers of different sizes for our decoder. We have decided to use the same architecture as the one detailed in 3.2 Extraction of 3D Information, but replacing the interpolations and convolution layers by layers of Inception Spiral Module. We keep the skip layers suggested by Stacked Hourglass Network [52] as they bring information about the expected shape of the foot. The architecture is schematized in Figure 3.6. Our model only predicts the new position of the vertices of the mesh. The number and the ordering of the vertices are the same as the template mesh as Inception Spiral Module keeps track of the vertices through the spiral sequences. With this information, we can use the set of faces F from the template mesh to complete the reconstructed 3D mesh of the foot.

To train the Inception Spiral Module algorithm, we have chosen to rely on three different losses to penalize bad predictions. The first loss is a simple L1 loss. This loss minimizes the difference between the coordinates of the vertices of the predicted mesh and of the label. The goal of this loss is to have the predicted mesh exactly as the labelled mesh. It is the only loss out of our three losses that takes into consideration the dimensions of the mesh, which is done through the coordinates of the vertices in the labelled mesh. The second loss is the edge length loss. To compute this loss, we use the set of faces F of the template. Inception Spiral Module is designed in a way that the set of faces F of the template is the same as for the predicted mesh. Each polygon is recorded as indexes of the vertices they are connected to. This means that each face connects three vertices together. We can use this knowledge to extract edges from the triangular polygons. We get a set of pairs of vertices that we know are close to each other. We can then compute the distance between each of those vertices. Doing so with the ground truth and the predicted mesh, we can then get the difference between the edges of the label and the edges of the predicted mesh. The edge length loss minimizes this difference. The goal of this loss function is to make sure that the 3D structure is the same between the label and the prediction. This loss is

not as restrictive as the L1 loss, but is necessary as the L1 loss will not converge when the label and the prediction are too different from each other. The third loss is the normal loss. It computes the direction of the normal vector for each vertex of the mesh. It then penalizes for the variation in the direction of the normal vectors between the label and the prediction. The goal of this loss is to smooth the predictions as the two other losses tend to create meshes that have an irregular surface. As this loss is used to smooth the results and as it can generate high loss values, we have decided to multiply the loss by a factor of 0.1. The three losses are then summed together for the training process.

3.4 3D Pose Estimation

With the mesh generated by the methods of 3.3 3D Reconstruction, we can acquire a 3D pose estimation. As stated earlier, the joints we are looking for in the pose are defined by our kinematic foot model described in 3.5.2 Kinematic Foot Model. They are illustrated in Figure 3.9. When building the dataset, we were able to link vertices of the 3D mesh foot model to the kinematic foot model. Since the 3D mesh foot model is also used as a template for the 3D reconstruction step, we can simply recover the 3D pose from the reconstructed mesh by selecting the same vertices on the predicted model. As Inception Spiral Module is a deformation algorithm, the position of the vertices around the meshes are not expected to move considerably. For example, some feet have their second toe longer than their great toe. When deforming the template which has the great toe being the longest toe, the vertices will move. The vertex at the end of the second toe will still be expected to be at the end of the toe in the resulting mesh, even though the length of the toe has changed. The L1 loss is guiding the deformation in that sense. Then, we need to interpolate the position of the joints. For some joints, the assigned vertices are on multiple sides of the foot so that the interpolation between the position of those vertices situates the joint inside the foot. An illustration of the vertices used for the joints of the toes can be seen in Figure 3.7. The white spots on the foot show the vertices assigned to a joint. For the white spots on the top of the toes, there are similar spots under the toes so that the joints can be interpolated inside the toes.

During training, the L1 loss of Inception Spiral Module penalizes when the meshes do not have the right dimensions. The coordinate of each vertex of the ground truth meshes are in millimetres. It is then expected for the resulting meshes to have measurements close to their labels. For RGB images, there might be variations in the proportions between the predictions and the labels as the system bases itself on the visible features of the foot to determine its measurements. In this sense, depth maps and RGBD images have the

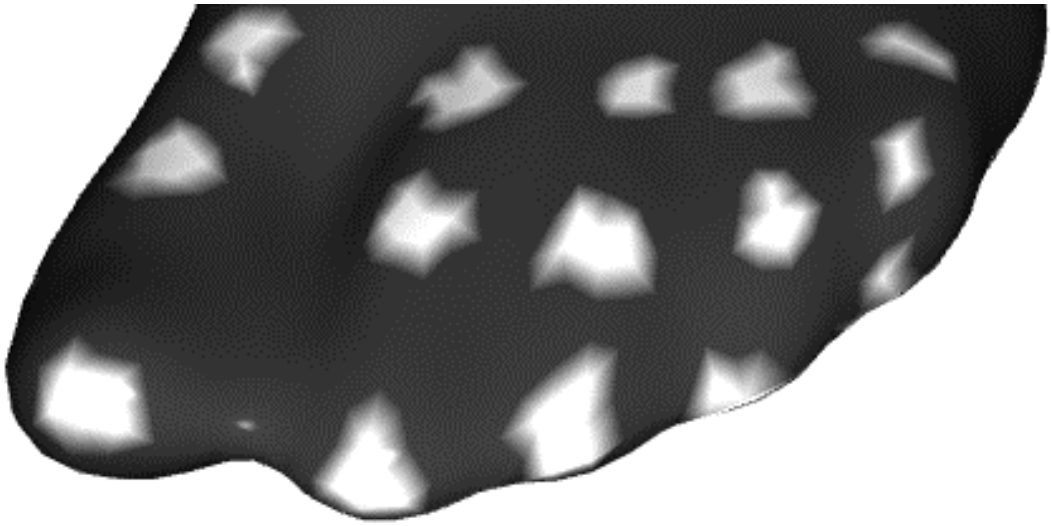

Figure 3.7: Illustration of some of the vertices, in white, that are connected to the joints of the kinematic foot model.

potential to bring better results. As some dimensions are recorded in their depth values, those two types of input have more information to bring to the system to infer correct measurements.

The 3D pose estimation module is the last step of the pipeline of our system. It is apparent that the 3D pose hinders on the kinematic foot model. Also, the whole system relies on the training data. This is what will be discussed in the next section.

3.5 Training Data

Now that we have described the whole pipeline, it is time to explain how we have created the dataset used for training. We would have used an already made dataset if one had been available. However, a dataset of images of feet with corresponding 3D mesh and 3D pose does not exist to our knowledge. We have decided to create our dataset by designing two foot models, a 3D mesh foot model and a kinematic foot model. With those two models, we are able to cover the 3D reconstruction and the 3D pose estimation. By projecting the 3D mesh foot model into 2D space, we can create an image that can be used as input to our system. By projecting the kinematic foot model into 2D space, we can get the position

of the joints in that input image. They can be used as labels for the training of our system. In the following, we will describe each model. The 3D mesh foot model will be described in 3.5.1 3D Mesh Foot Model. The kinematic foot model will be explained in 3.5.2 Kinematic Foot Model. Afterwards, how both models are used together and how the input images for our system are generated will be discussed in 3.5.3 Other Considerations.

3.5.1 3D Mesh Foot Model

The first model we have developed is a 3D mesh foot model. The purpose of this model is to represent the shape of the foot. This model should be able to take the shape of any foot we require and be precise over the eight measurements [56] described in Figure 2.1. However, as we have also stated earlier, we will only focus on length and width. This does not mean that we will not try to have a foot model that can produce the eight measurements, only that the six other measurements will not be used to test the 3D mesh foot model.

To create this 3D mesh foot model, we had access to a dataset of 3D meshes of feet. However, every mesh in this foot dataset has a different number of vertices which are mostly scattered at random around the mesh. We need our model to have vertices at specific locations so that we can associate vertices to our kinematic foot model which is needed for our 3D pose estimation. The meshes of the foot dataset only have a dozen vertices out of the thousands forming the meshes that keep the same location. Those vertices are called markers in this chapter. However, the number of markers is not sufficient for our kinematic foot model. Knowing this, we realized that we could not simply use an algorithm such as QSLIM [23] to bring all meshes to the same number of vertices. However, the foot dataset could be useful if we can leverage the shape of the meshes to create a 3D mesh foot model that meets all our criteria.

To design our 3D mesh foot model, we first need to state what we want our model to do and what characteristics we want it to have. The first ability we want our model to have is to be able to take a wide variety of different shapes of feet. This is mainly fulfilled by using a mesh as 3D representation which is very flexible. The second point we need to consider is, since we want to have a kinematic foot model with defined joints to connect with the mesh, we want to have vertices with fixed locations. This means, the same vertex on two different feet will be at the same location around the foot. For example, a vertex at the end of the big toe on one foot will be at the same location on the second foot. It will allow to easily bind the joints of the kinematic foot model to the mesh surface. A third characteristic linked to the previous one is that the model should be deformable to show the pose of the foot. This pose will be determined by the kinematic foot model. But,

since the 3D mesh foot model presents the physical shape of the foot, it must also be able to show the pose. We can associate the surface of the 3D mesh foot model to the skin of the foot while the kinematic foot model is the skeleton of the foot. The skin is deformed when your skeleton moves, helped by the muscles. The 3D mesh foot model should be able to do the same thing. A fourth and last characteristic that the 3D mesh foot model must have is the possibility to be projected into 2D space so that the resulting image looks convincing. This is again solved by using a mesh as the 3D representation. If we look at the three previous points, they can be achieved with a point cloud. However, a point cloud projected into 2D space will not give a detailed foot in the resulting image. In light of this reasoning, we have decided to use the template fitting algorithm of Allen *et al.* [2] to bring every mesh to the same number of vertices. Since we are deforming a template with this algorithm, the vertices will be at the same location in every mesh. Then, we can use all the transformed meshes of the dataset to generate a PCA model of the foot.

We first start by using the template fitting algorithm of Allen *et al.* [2] to transform the meshes of the foot dataset into meshes with all the same number of vertices. To do so, we begin by choosing one mesh of a foot as a template. We are going to use the position of the vertices of this template for all the other meshes, which means that all the meshes deformed by this algorithm will have the same number of vertices as this template mesh. We want to morph the template into the shape of another mesh, called the target. The idea behind the template fitting algorithm of Allen *et al.* [2] is to move the vertices of the template toward the surface of the target mesh. For this reason, we associate an affine transformation matrix to each vertex. Each affine transformation matrix has the goal to move its vertex toward the surface of the target mesh. To do so, we are using three error functions to penalize iteratively the difference between the template mesh and the target mesh. Those error functions are illustrated in Figure 3.8.

The first error term that we want to minimize is called by Allen *et al.* [2] the data error. This error simply penalizes the distance between the vertices of the template and the surface of the target mesh. It can be formulated as in the following.

$$E_d = \sum_{i=1}^{n} w_i dist^2(\mathbf{T}_i \mathbf{v}_i, S_D) \tag{3.7}$$

In equation 3.7, n is the number of vertices $\mathbf{v}_i$ in the template, w_i is a weighting term to control the influence of data in different regions, $\mathbf{T}_i$ is an affine transformation matrix associated with the vertex $\mathbf{v}_i$, S_D is the surface of the target mesh, and $dist$ is a distance function that computes the Euclidean distance between the template's vertex and the surface of the target mesh. This error moves the vertices from the template to

the closest point of the surface. Here we use "point" as any position on the surface, not as "vertex". The vertices of the template are not necessarily moved towards the vertices of the target mesh. As a simple example, let us consider a spherical mesh as a template and a target mesh displaying a cube. The cube can be formed by only eight vertices. The sphere will most likely contain a large number of vertices to reproduce its curve. When moving the vertices from the sphere towards the surface of the cube, most will not land in the corners where the eight vertices of the cube lay. Most vertices will move towards the planes that constitute the faces of the cube where there are no vertices. As we are not trying to match each vertex from the template to each vertex of the target, this effect is encouraged. However, this behaviour can be harmful when the target mesh contains steep slopes that are not in the template, spikes, noise in the position of each vertex, holes in the mesh, etc. The second error is there to compensate in those situations.

The second error term, called the smoothness error, penalizes when the affine transform matrices of two neighbouring vertices are different. This error can be formulated as in the following.

$$E_s = \sum_{\{i,j|\{\mathbf{v}_i,\mathbf{v}_j\}\in edges(S_T)\}} ||\mathbf{T}_i - \mathbf{T}_j||_F^2 \tag{3.8}$$

In equation 3.8, S_T is the surface of the template, $||\cdot||_F$ is the Frobenius norm which computes the Euclidean distance between matrices, and *edges* is one side of the triangular faces of the mesh. To be more specific, we use the edges as a tool to help us find each neighbouring vertex. What this error does is to force the vertices to move as parallelly as possible toward the surface of the target mesh. Using only this error, we would only be able to scale and translate the template without changing its shape. Combined with the data error presented in equation 3.7, the smoothness error makes the algorithm of Allen *et al.* [2] resilient to noise. In the case of holes in the target mesh, the surface of the template will be encouraged by this error term to cover it because the vertices will be pushed to go parallel rather than sideways toward the edge of the holes. Obviously, holes of considerable size will still be an issue. In the foot dataset, there is no hole in the meshes. However, the smoothness error is necessary when there is finer details that could disrupt the data error, such as between the toes. The two errors presented as of yet used together is enough to reconstruct the meshes in the cases where the two meshes are close to each other. This is our case as we are morphing a mesh of a foot into another mesh of a foot. Still, Allen *et al.* [2] has created a third error for objects that are farther away, like for our example of a sphere being morphed into a cube. We have decided to use this third error as it improves the quality of the transformation. Moreover, it will ascertain that some vertices are at a

specific position on the meshes which will lead to a more accurate 3D pose estimation.

The third error term is designed by Allen *et al.* [2] as the marker error. As said earlier, the meshes of feet from the dataset have their vertices scrambled but for a dozen vertices. We are calling those vertices markers and using them as control points in the deforming algorithm. By making sure that the markers on the template and the target mesh are at the same position at the end of the morphing process, we can assure that the general shape of the foot is respected. This does imply that the markers are placed around the mesh in a way to capture the general shape of the foot. In our case, they are placed this way as these vertices comes from the physical anthropometric markers put on the feet when they were scanned. The marker error can be defined as follows.

$$E_m = \sum_{i=1}^{m} ||\mathbf{T}_{k_i}\mathbf{v}_{k_i} - \mathbf{m}_i||^2 \tag{3.9}$$

In equation 3.9, $\mathbf{m}_i$ represents the vertices used as markers on the target mesh, while $\mathbf{v}_{k_i}$ represents the vertices used as markers on the template mesh with their associated affine transform matrices $\mathbf{T}_{k_i}$. This error is assuring us that we have the general shape of a foot after the morphing transformation. It also make sure we do not fall into a local minimum. One such minimum is that a valid solution to the combination of the data error and smoothness error is the trivial solution where all the affine transform matrices are null. This then means that the solution to this minimum is that the shape should remain as the shape of the template mesh, which is counterproductive in our application. The marker error prevent this as it forces certain vertices to move to a different position so that their affine transform matrices are not null. The smoothness error would then force the nearby vertices to move with the markers, which denies the trivial solution.

We have now defined the three error terms: the data error, the smoothness error, and the marker error. We can see an illustration of the three error terms in Figure 3.8. To efficiently morph the template into the target, we need to run an optimization algorithm on the three errors. We combine them with a weighted sum, as seen in the following.

$$E = \alpha E_d + \beta E_s + \gamma E_m \tag{3.10}$$

In equation 3.10, α, β and γ are the weights for each of the error terms. We use the iterative L-BFGS-B [105] solver to find the solution to the optimization problem. We also vary the weights over the iterations since the marker error has a bigger impact on the first iterations. We follow the suggestions from the original paper [2] to set up the weights properly.

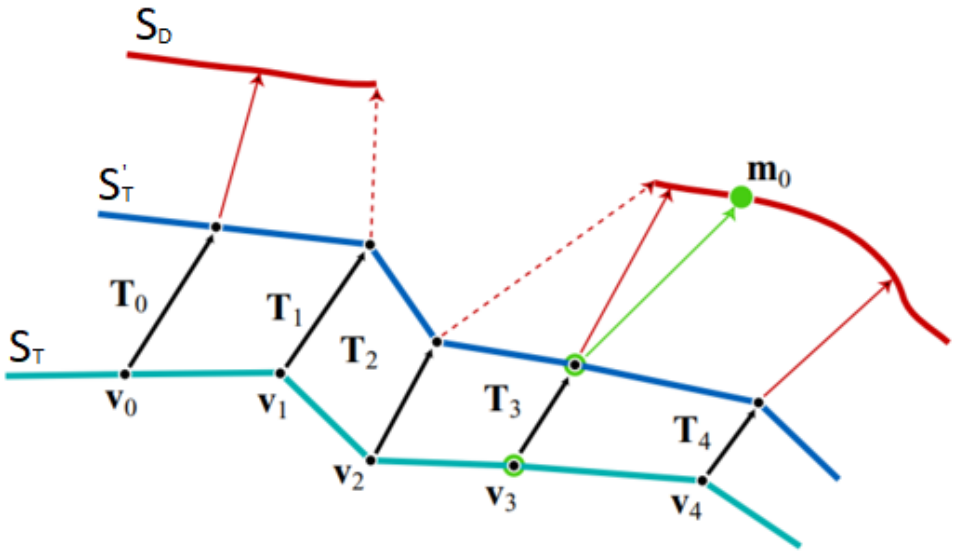

Figure 3.8: Illustration of the three error terms of the template fitting algorithm as illustrated by [2]. $\mathbf{S}'_\mathbf{T}$ represents the surface of the template mesh after one iteration of the template fitting algorithm. The data error for the next iteration is represented by the red arrows. The influence of the smoothness error can be seen in the black arrows, which represent the movement of the surface of the template during the first iteration. They are mostly parallel to each other even with the presence of a hole in the surface of the target mesh $\mathbf{S}_D$. The marker error can be seen in the green arrow that redirect $\mathbf{v}_3$ towards the marker $\mathbf{m}_0$.

We are now able to morph all the meshes of the foot dataset into the same foot but with the same fixed number of vertices with known location. We could use those meshes to generate our own virtual dataset, but we have decided to use the meshes instead to compute a PCA model so that we can create a larger number of feet than what there is in the original dataset. To do so, we have stacked all the meshes in a matrix. There are around 40,000 meshes in the foot dataset. If n is the number of vertices in the template and k is the number of meshes in the dataset, the matrix is of size $3n \times k$ as there are three coordinate values for each vertex. We were then able to apply PCA to the matrix and get the components. Acting on the variance of those components, we can generate a large number of realistically looking 3D meshes of feet. We did not keep the components that had a negligible impact on the PCA mesh to simplify the model.

At this point, we have a 3D mesh that can be reshaped by simply modifying the variance of the PCA components to obtain different feet. The mesh with no modified variance is used as the template for 3.3.2 Inception Spiral Module. Each mesh that we get by modifying the variances of our 3D mesh foot model has vertices at the same location. This means that we can connect some of those vertices to the joints of the kinematic foot model. This model is what we are going to describe in the next section.

3.5.2 Kinematic Foot Model

The goal of the 3D mesh foot model is to describe the shape of the foot. The goal of the kinematic foot model is to describe the pose of the foot. To do so, we need to determine what characteristics our kinematic foot model should have.

The first characteristic that the kinematic foot model must have is the ability to mimic any movements a healthy foot can make. We are not looking into showing possible foot injury with our model. We assume that every foot analyzed by our system does not present any injuries or considerable deformities. The second characteristic is complementary to the first one, we need to assure that the kinematic foot model is not able to do movements that a healthy foot cannot do. It is important to make sure that the model is not able to do impossible movements as we want our dataset to be as close as possible to real life. We do accept, however, movements that a foot can do only when subject to exterior stress. For example, it is possible to have a toe pointing upward and the toe next to it pointing downward. It cannot be done consciously, but it is possible to hold toes in that position with exterior help without causing injuries. During the training of our system, we do not want to learn a pose of the foot that would be unfeasible by a real life foot. A third characteristic we need is that the kinematic foot model should have cues of its existence

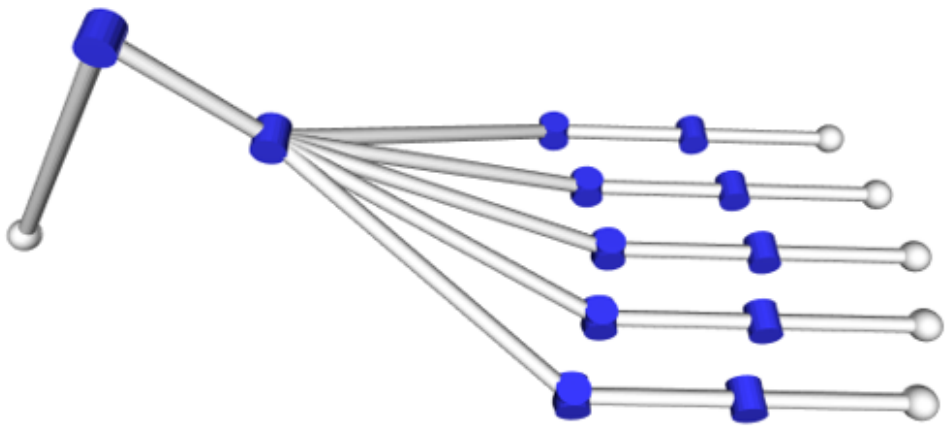

Figure 3.9: Illustration of the kinematic foot model we have designed. The round joints indicate the end of a branch, while the cylinders indicate rotating joints.

on the shape of the foot which is represented by the 3D mesh foot model. We need this characteristic because our system must be able to detect the structure we want from an image of the foot.

To respect the needed characteristics, we mapped out an eighteen joints model. The model is illustrated in Figure 3.9. The joints include the end of the branches even if these do not describe movements. We call them joints too as they are to be detected by our system like the rotating joints. Each joint at the end of a branch represents an extremity of the foot. The link between two joints usually represents a bone or a group of bones. The rotating joints themselves represent articulations within the foot. To better understand the making of the kinematic foot model and for an easier read, we show Figure 2.5 a second time in Figure 3.10. We have separated our kinematic foot model into six different branches. Each branch starts at the midfoot, which represents the articulation between the calcaneus and a group of bones formed of the cuboid, the cuneiforms, and the navicular. This articulation was studied and deemed a crucial articulation by the Oxford foot model [14]. It was also an important part of the Milwaukee foot model [36]. We gave this articulation a small range of rotation as it is comprised anatomically of mostly only elastic tissues. We have shown the numerical bounds on rotation for every joint of our kinematic model in Table 3.1. These angles have been derived from the literature covered in 2.6.2 Kinematic Foot Model and tuned up experimentally with the help of the 3D mesh foot model.

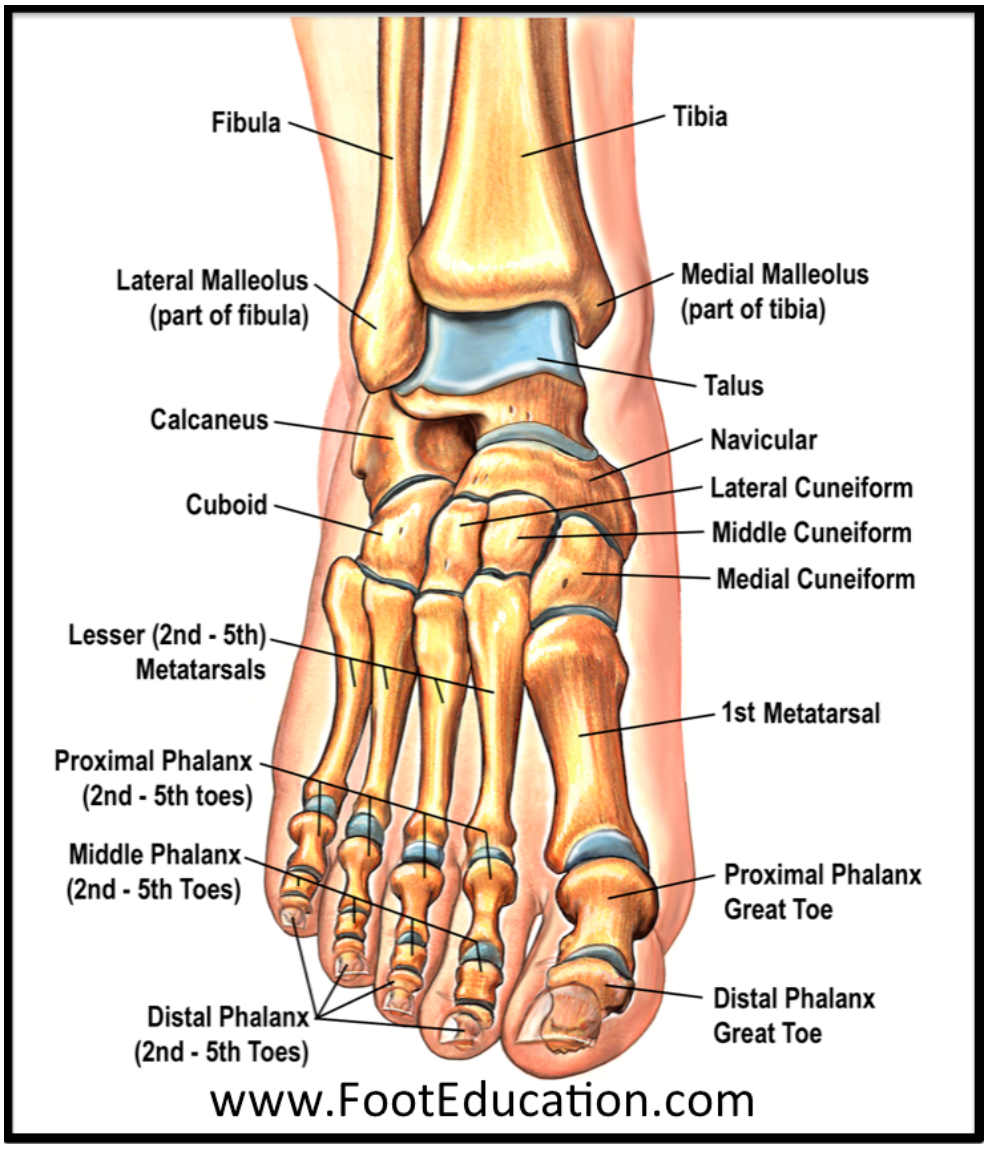

Figure 3.10: Bones of the foot and ankle [55]

Joints	Axis	Lower bound	Upper bound
Midfoot-Ankle	Y	-0.35	0.8
Ankle	Y	-0.1	0.1
Midfoot-Metatarsals	Y	0	0.1
Proximal phalanxes	Y	-1.4	1.4
Proximal phalanxes	Z	-0.13	0.13
Distal phalanxes	Y	0	1.6

Table 3.1: Range of rotation in radians on joints of the kinematic foot model. Figure 3.9 shows the model when all angles are null. Proximal phalanxes are the only joints permitting two degrees of freedom. All other joints permit only one degree of freedom.

The first branch goes toward the heel. From the joint at the midfoot, it goes to the ankle before going down to the end of the heel. The joint in the ankle represents the articulation between the calcaneus, the fibula and the tibia, and the midfoot. We have allowed only a small range of movement. We do not represent the movement between the foot and the leg in our model. The last joint of this branch is located at the end of the calcaneus. It does not represent a rotation, but the end of the branch. It also represents the back of the foot and will be particularly useful to get the length of the foot. The calcaneus is important in many biomechanical foot models such as the Oxford model [14], the Milwaukee model [36] and the Salford model [51].

The five other branches have the same topology. They all start at the midfoot. The first joint is at the articulation between the metatarsals and the proximal phalanges. It represents the articulation where the toes begin on the foot. This is the articulation that allows you to wiggle your toes. This articulation can rotate on multiple axes as seen in Figure 3.9, and each of those movements are ample. The second joint of these branches is the articulation between the proximal phalanges and the distal phalanges. This is the articulation that allows you to bend your toes. This articulation also has a wide range of movements, but only allows the toe to bend downward. The last joint of these branches represents the end of the toes. They show the limit of the foot on the front side. The way we chose to represent these branches does not follow the biomechanical foot models we presented in 2.6.2 Kinematic Foot Model. These foot models usually take the toes as a rigid block that moves together. This is the case for biomechanical foot models like the Oxford model [14], the Milwaukee model [36], or the Salford model [51]. However, in our application, we want to know what impact the pose of the foot has, and the toes are part of the foot that changes the pose the most. In this case, we decided to follow more a biomechanical foot model such as the Glasgow-Maastricht model [54].

With our kinematic foot model designed, we only had to make sure to be able to use forward and inverse kinematic techniques. Observing our kinematic foot model, we realized that each branch could be handled separately. This simplified the problem of inverse kinematic, and allowed us to use simple and efficient algorithms that could not be used with multi-branch systems.

We now have a kinematic foot model that can be modified at will to take the pose of a healthy foot. We now need to connect it to the 3D mesh foot model. With the two models, we will then be able to create images for our dataset where the foot is in the desired pose. This will be the subject of the next section along with other considerations taken when creating the images.

3.5.3 Other Considerations

We have discussed in length of the two foot models we have built to represent the foot in the images of the dataset, which are the 3D mesh foot model and the kinematic foot model. We have yet to talk about how to use those two models together. We have also not detailed how we are creating those images. As a last consideration, we have not discussed more precisely what data we are generating alongside the input images. We will do so in this section. Before getting to the heart of the matter, it is important to mention that the dataset is created to show only left feet. The reason for this is that our 3D mesh foot model is a left foot. To have right feet in the dataset, we would need to create a new 3D mesh foot model that represents a right foot. We chose the left foot randomly as there is no difference between using the left foot and using the right foot to test our system.

We combine the 3D mesh foot model and the kinematic foot model by grafting the joints of the kinematic foot model to vertices of the 3D mesh foot model. To do so, we have manually selected vertices for each joint. An illustration of some of those vertices can be seen in Figure 3.7. For the joints at the end of a branch, a group of vertices at the proper location have been selected. For example, for the big toe, a group of five vertices has been associated with that specific joint. By averaging the positions of all those vertices, we can have the desired position of the tip of the big toe. For the rotating joints that are inside the foot, we have selected vertices around the mesh so that we can interpolate the location of the joints from the position of those vertices. For example, for the ankle joint, we have selected vertices on the lateral malleolus and the medial malleolus. Those are the protuberances that can be found on either side of the ankle as seen in Figure 3.10. Forty-two vertices were selected for the lateral malleolus that cover the whole bone structure protruding from the mesh while forty vertices were selected for the medial malleolus for the same reason. This interpolation can be seen in Figure 3.11.

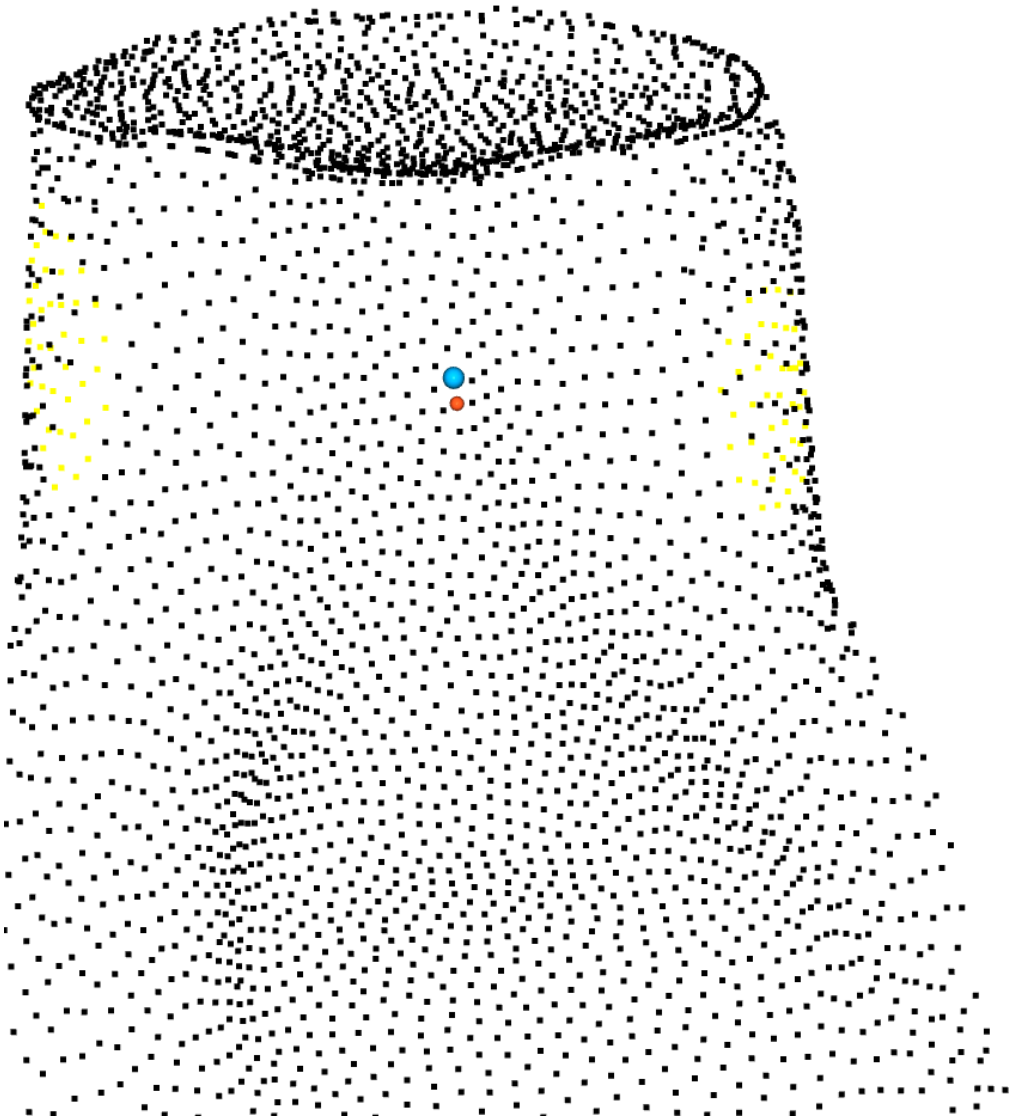

Figure 3.11: Example of the interpolation used to acquire the position of the ankle joint. The presented view looks into the foot toward the heel, which can be seen by the depression of the vertices in the lower part of the illustration. The yellow vertices are the selected vertices to which the ankle joint is grafted. The blue dot is the interpolated position of the ankle. The red dot is located on the backside of the foot and is shown in the illustration for perspective. The red and blue dots are of the same size.

Now that the kinematic foot model is attached to the 3D mesh foot model, we can deform the 3D mesh foot model so that it can take different poses. We start by rotating the joints of the kinematic foot model to show the desired pose by using inverse kinematic computations. For random poses for the training data, we randomly choose a reachable position for the end of each branch of the kinematic foot model and compute the joints' angle to accommodate the pose. We then compute the new positions of the vertices that are associated with each joint using the traveled distance of the joint and its known rotation. From there, we can use the deform-as-rigid-as-possible algorithm from Open3D [104]. This algorithm deforms the mesh by moving the vertices not associated with the joints of the kinematic foot model so that the shape of the foot is kept during the deformation.

With the 3D mesh foot model deformed to show the pose, we can now generate the images for our virtual dataset. We first need to colour our 3D mesh foot model in a realistic skin colour. We have used the bounds in the red, blue and green channels suggested by Kolkur *et al.* [38] to colour our meshes. The bounds are used for a task of skin colour detection, which means that they were calibrated for a wide range of skin tones to be able to detect all possible skin colours. With our coloured mesh, we can now project them into the 2D space of the image. We have selected different images from royalty-free databases as background for our images. We can then render a projection of the foot with a random view using the library Trimesh [22]. The chosen view is randomly selected with a range of angles that make sure that the view does not show the back of the foot. We designed it this way so that we can see more than only the back side of the foot in the image. If we do not restrict the angle of view, we can have instances where the only visible part of the foot of the image is the heel. In that case, only two out of the eighteen joints can be seen, which can make it too difficult for the system to handle. In a last note on the images, the size of the generated images is 224x224, which is the expected input for the backbone of ResNet [29] that we use for the detection of the 2D cues as described in 3.2 Extraction of 3D Information.

At this point, we have generated images that can be used as input for our system. We also need labels so that we can supervise the training of our neural networks. For the 2D cues, we need labels for the silhouette and the landmarks of the joints. The ground truth for the silhouette can be acquired by projecting the 3D mesh model into a black image. Then, we can colour white every pixel that is not black. The ground truth for the joints can be acquired by projecting the kinematic foot model into 2D space and taking notes of their position pixel-wise. For the 3D reconstruction, we save the vertices of the deformed 3D mesh foot model as a point cloud. We do not need the face values as they are the same as the face values of the template. We can then save computer resources by using the face values of the template with every saved 3D mesh foot model, which will allow us

to use those resources for other tasks such as training. For the 3D pose, we take note of the positions of the joints of the kinematic foot model after having deformed the 3D mesh foot model. We were also able to generate depth maps with the library Trimesh [22]. In this case, Gaussian noise was added to the depth maps to keep the dataset as realistic as possible as 3D sensors generate this kind of noise.

3.6 Summary

This chapter described the system we have designed and its possible variations. The 3D reconstruction is done through the use of intermediary results, being the silhouette and the joints of the foot. To acquire those, we have adapted the algorithms of Stacked Hourglass Network [52] with the backbone of ResNet-18 [29]. For the 3D reconstruction, we have experimented with two different methods. The first one is EdgeConv [86], which creates a graph out of the features produced by the network based on geometric distance. The second method is Inception Spiral Module [15], which used the ordering of the vertices of a template to change the order of input of the layers of the network. For the 3D pose estimation, we are able to acquire the 3D pose from the results of the 3D reconstruction by leveraging the shape of the 3D foot. Finally, we have created our training data by deforming meshes of a private foot dataset we had access to to create a PCA model of the foot. We have also designed a kinematic foot model made of eighteen joints that is able to reproduce the possible movements of the foot. The experiments we have conducted to test our system are presented in the next chapter.

Chapter 4

Experiments and Discussion

In the last chapter, we have described our pipeline designed to achieve the 3D reconstruction and the 3D pose estimation of a foot from a single image. This pipeline is divided into smaller tasks. Each task can be evaluated since we have generated a panoply of labels along with the virtual dataset for this very purpose. Those labels have also been used during the training of the system. From our virtual dataset, we have generated 50,000 samples for the training process and 10,000 samples for evaluation. The results shown in this chapter have been obtained on the evaluation's samples. We were also able to use real world RGB images to test the generalization of our system. Measurements were acquired from these images as ground truths using the method presented by Ballester *et al.* [4].

With the idea to understand the impact of different kinds of data type on our pipeline, we have tested our pipeline on three different inputs. We have RGB images. Those images have three channels, each representing a colour. We have RGBD images. Those images have four channels. The first three are the same as the RGB images. The fourth channel is a depth map that indicates the 3D structure of the object in the image. Finally, we have depth maps which are the same as the fourth channel of the RGBD images.

With the idea to test the impact of different angles of view on our system, we have produced results for multiple viewpoints around the foot along with the results of any viewpoint. The viewpoints are taken around two axes of the foot. The rotation of the angle of view around the foot is done around one axis at a time. By using an angle of view at every 30 degrees, we have generated 24 viewpoints. The angle at 0 degree and the angle at 180 degrees are captured twice doing so. We then have 22 different viewpoints. The angle at 0 degree is shown in Figure 4.1. The two axes used to generate viewpoints around the foot are presented in the corner.

Figure 4.1: Reference viewpoint with axes for the results on specific angles of view.

Metrics	RGB	RGBD	Depth maps
Incorrect detections	6.153	8.200	7.188
False positive detections	2.464	7.850	6.914
False negative detections	3.689	0.350	0.274

Table 4.1: Pixel-wise error rates of our silhouettes detector in percentages. Incorrect detections refer to all the wrongly labelled pixels. False positive detections refer to pixels labelled as part of the silhouette while they are not in the ground truth. False negative detections refer to pixels labelled as background while they are labelled as part of the silhouette in the ground truth.

4.1 Extraction of 3D Information

We start the assessment of our pipeline with the evaluation of the 2D cues to the 3D object in the input image. We have two byproducts of our system to evaluate: the silhouette and the landmarks of the joints. Both are inferred as binary confidence maps. We take the largest cluster in each confidence map generated with the module presented in 3.2 Extraction of 3D Information to obtain the results presented in this section.

The silhouette indicates the shape of the 3D object in the input image. We are looking for the outline of the foot. When training, we use masks that give the outline of the foot as seen in the left column of Figure 4.2. The other columns of this figure show results for the three different inputs. The masks are binary images where the white values mean detection and the black values mean the opposite. As we can see, the RGB images have multiple instances where they do not detect the shape at all. In other instances, they also do not

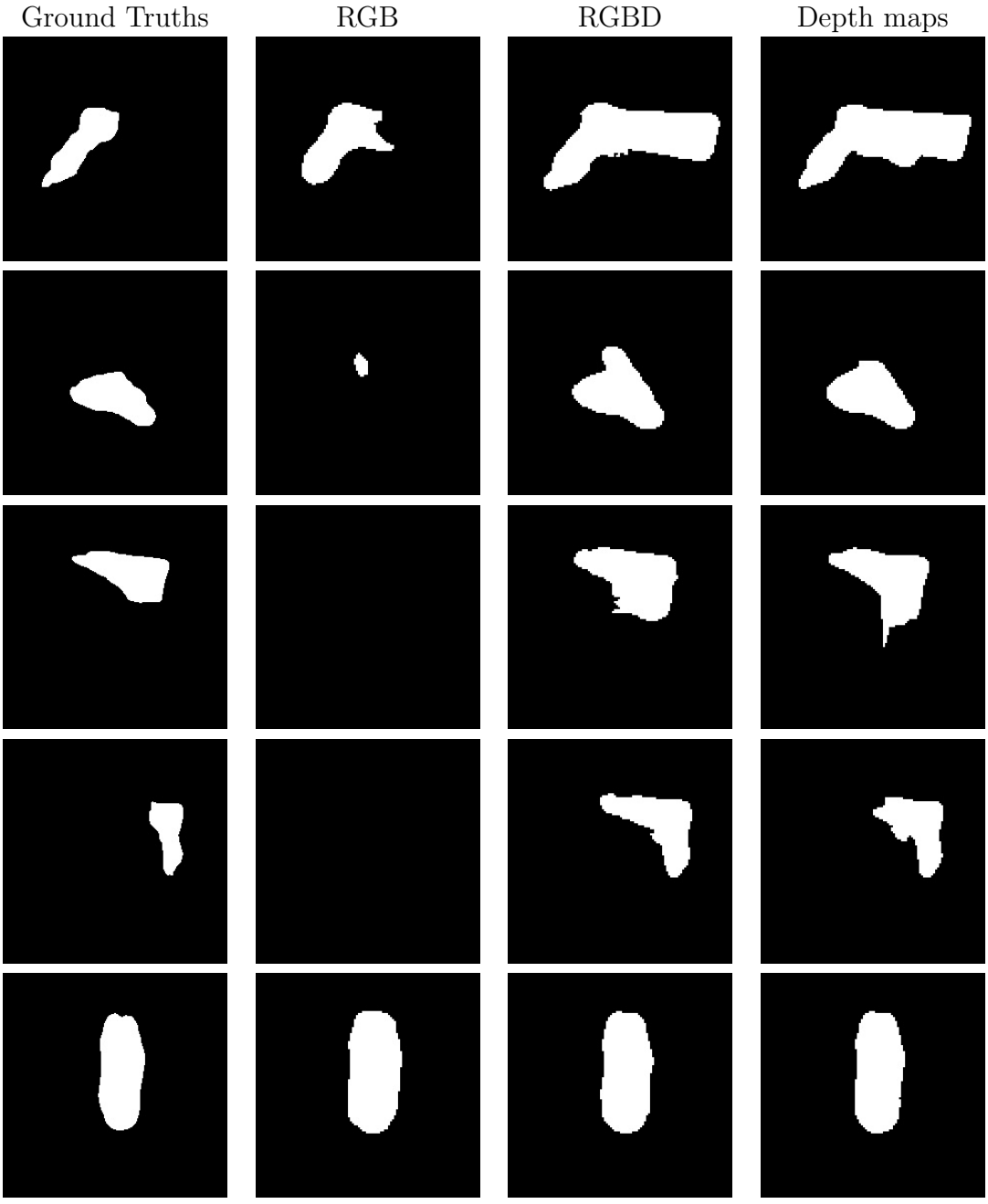

Figure 4.2: Examples of silhouette detection generated by our system. First column shows the expected silhouettes, while the other three show the output for different types of input. Each row shows a different example.

properly segment the foot. However, there are cases where the segmentation worked well. For RGBD and depth maps, the results are not better. They often capture more than what is expected. This can be seen when they segment part of the leg with the foot. This is worst with the RGBD than with the depth maps. Both also capture part of the image that is not the foot nor the leg. It protrudes from the detection. The depth maps are more inclined to make this mistake than the RGBD images. We do not see this problem from the RGB images. We believe that it comes from the Gaussian noise added to the depth maps, which represent the noise from a real 3D sensor. For the evaluation of the silhouette, we have chosen to rely on binary classification of each pixel. To do so, we have compared each pixel of the predicted masks to the corresponding pixel of the labels. We can see the results in Table 4.1. To better understand where the incorrect detections came from, we have also computed which of those incorrectly detected pixels were false positives and which were false negative. A false positive is a white pixel meaning a predicted detection where it should be a black pixel. A false negative is a black pixel meaning the predicted absence of a detection where it should be a white pixel. From the Table 4.1, we can see that RGB images have the lower incorrect detection percentage with 6.153% against 8.200% for RGBD images and 7.188% for depth maps. By comparing this result with the qualitative results of Figure 4.2, we can get to the conclusion that the incorrect detection percentage for RGB images comes from a too small number of detections while it comes from a too large number of detections for RGBD images and depth maps. This is what the false positive detection percentage and the false negative detection percentage are here for. For RGB images, they show that there are mistakes in detections and in the absence of detections. For RGBD images and depth maps, they show that the incorrect detections are almost only false positive. This means that the foot is found in every case, but that it also segments parts of the image that are unwanted. These parts often take a lot of image space, which is why the incorrect detection percentage is higher compared to RGB images.

The landmarks of the joints indicate the projection of the pose of the foot. We have eighteen joints, which we evaluate individually. We want to assess the distance between predictions and labels pixel-wise. The results are shown in Table 4.2. The joints are numbered as seen in Figure 3.10. The big toe is considered the first toe, the toe next to it is the second toe, etc. What we can see from the table is that some toes are more difficult to find than others. No matter the input, the big toe has higher distances from the label for each of its joint with 102.81 pixels for RGB images, 47.49 pixels for RGBD images, and 49.59 pixels for depth maps. This comes at a surprise since it is visually easy to find. We believe that the difficulty in its detection comes from the fact that, when the pose of the foot changes, the big toe is the one that modifies the shape of the foot the most as it is the biggest toe. Its high variability then causes the failed predictions. As

Joints	RGB	RGBD	Depth maps
Midfoot	141.4499	28.1029	35.8103
Proximal phalanx 1	95.4399	41.9504	54.9826
Distal phalanx 1	101.2051	40.7713	46.6286
Tip 1	102.8083	47.4862	49.5915
Proximal phalanx 2	97.7619	30.9208	43.3532
Distal phalanx 2	90.4097	23.4257	44.0197
Tip 2	98.8282	32.9785	45.4639
Proximal phalanx 3	99.2838	38.1556	42.2738
Distal phalanx 3	100.9541	31.2178	39.5547
Tip 3	86.0554	35.5017	45.9784
Proximal phalanx 4	96.5194	46.2231	43.2921
Distal phalanx 4	114.0157	42.4195	43.9353
Tip 4	93.1365	48.3664	44.9592
Proximal phalanx 5	118.1127	46.7550	47.6220
Distal phalanx 5	109.1565	48.5688	41.0690
Tip 5	85.7960	51.0431	41.7108
Ankle	119.9500	37.6013	33.7193
Heel	88.3186	73.6124	62.7961
Mean	102.1779	41.3945	44.8200

Table 4.2: Average of the distances between each joint and its label in pixels. All images are of dimensions 224x224 pixels. The joints mentioned in the first column correspond to the joints of our kinematic foot model shown in Figure 3.9. The midfoot is the central joint from where each branch of the kinematic foot model starts. The Proximal phalanges refer to the closet joints from the midfoot in the branches that go to the toes. The Distal phalanges are the joints following in the same branches. The Tips refer to the joints that represent the tip of the toes at the end of the branches. The order of the joints follows Figure 3.10. The ankle represents the joint in the articulation of the same name while the heel represents the end of this branch. The three other columns show the results for each different type of input.

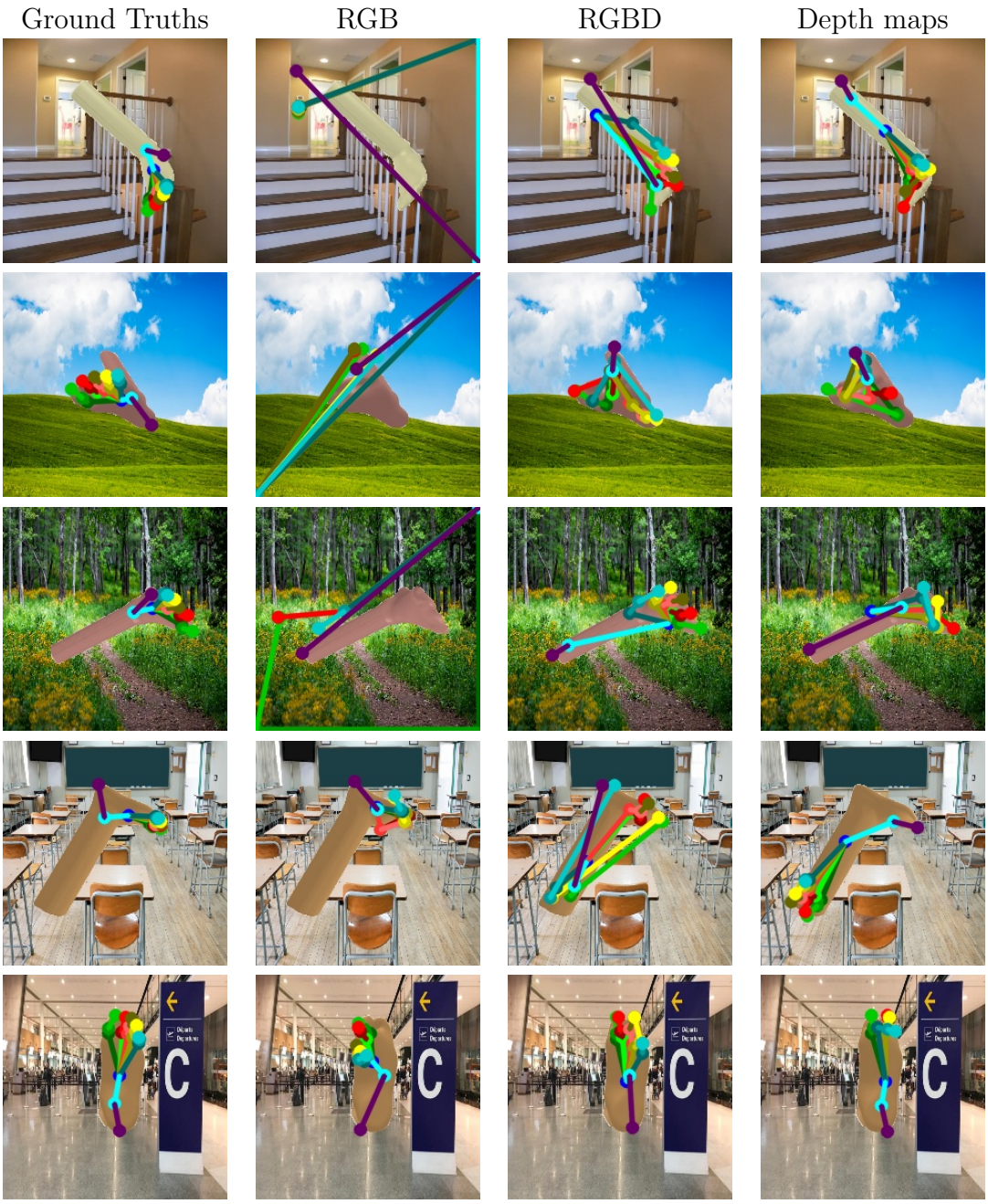

Figure 4.3: Examples of joint detection generated by our system. First column shows the expected joints' positions, while the other three show the output for different types of input. Each row shows a different example.

for the midfoot, it is one of the joints with the lower error on position for RGBD images and depth maps, with 28.10 pixels for the former and 35.81 pixels for the latter. However, for RGB images, it is the joint with the worst error on position with 141.45 pixels. This can be explained by the low performance of the outline detection for the RGB images as this joint relies more on the global shape of the foot to be found as opposed to the toes can be found from the contrast of colours that are created between the toes by shading. As for the last branch of our kinematic foot model that goes toward the heel, it shows interesting results. For RGB images, the ankle is one of the joints with the higher distance from the expected position with 119.95 pixels, but the heel is one of the joints with the lowest error with 88.32 pixels. Visually, the heel is easy to find as it is the end of the foot, while the ankle can be challenging as it is not close to any edge of the foot. As for the midfoot, an understanding of the global shape is necessary to find this joint, which cannot happen with the RGB images as the silhouette is not found in multiple instances. For the RGBD images and the depth maps, the ankle, like the midfoot, is one of the joints with the lowest distance from its expected position, with 37.60 pixels for RGBD images and 33.72 pixels for depth maps. This is not the case for the heel which is the joint with the worst results for both inputs, with 73.61 pixels for RGBD images and 62.80 pixels for depth maps. The reason for this is that often, the heel is detected at the end of the leg closer to where the knee would be instead of being detected in the foot. The model has learned that the heel should be at the other end of the detected shape compared to the toes, and since the RGBD images and the depth maps both often detect the leg during outline detection, the end of the leg is what transpires from the joint detection for the heel. With all results considered, we see that RGBD images are the ones that give the better results when looking for the joints. Depth maps are not far behind RGBD images as their distances are only slightly higher. Since RGBD images are the ones with more information with its four channels, it is not surprising. RGBD images are able to use features from its colour channels to increase its accuracy compared to depth maps. RGB images are by far the worst, being more than double the distances of depth maps for most joints. This is explained by their poor performances in the silhouette detection, as the silhouette is used to find the location of the joints. Some examples of the joints detection is shown in Figure 4.3. In this figure, the depth maps are replaced by an RGB image showing the exact same scene to be able to better understand the difference with the other two inputs. For RGBD images, we only show the colour channels for the same reason. It is clear from the figure that RGB images are incapable most of the time to acquire the joints location. We can see that RGBD images and depth maps are better, but still far from what is expected.

If we consider results from specific viewpoints, we do not observe improvements. We have not generated data for the outline detection as finding the silhouettes depends min-

		RGB		RGBD		Depth maps	
Axis	Angle	Mean	STD	Mean	STD	Mean	STD
X	0	101.7379	52.4761	64.2487	45.9150	57.6006	44.5248
	30	124.1537	39.6167	57.4607	34.6371	57.1504	39.7537
	60	127.3864	37.9605	76.5973	36.6179	82.7418	38.1946
	90	120.5195	53.8613	66.7300	33.7922	81.4572	33.2472
	120	129.9640	62.4508	80.8935	26.2933	90.0529	34.0814
	150	124.8109	71.0624	49.3808	34.8933	78.5390	36.7218
	180	107.8418	49.7383	63.6565	45.2832	73.9217	42.7499
	210	120.4148	67.0843	42.0881	29.5072	45.5719	38.9324
	240	107.8701	60.9658	18.5360	26.1151	11.6937	23.4442
	270	113.5183	69.7716	7.2607	13.0911	9.8040	13.1631
	300	134.6578	52.6842	10.1120	13.9650	9.0054	12.8096
	330	127.0392	47.3572	41.4809	29.1837	31.8469	26.0504
Y	0	101.7378	52.4761	64.2487	45.9150	57.6006	44.5248
	30	110.7245	42.3885	71.2428	43.5466	71.3060	45.0730
	60	100.5156	51.4323	60.2694	38.4064	50.2370	35.3346
	90	84.2025	54.2156	59.9286	36.2507	37.4746	36.8793
	120	94.6759	47.7915	57.0206	38.3125	45.3483	37.8622
	150	93.9282	50.2442	72.3060	45.3096	60.0556	45.2438
	180	107.8418	49.7383	63.6565	45.2832	73.9217	42.7499
	210	98.9863	41.2255	45.8884	33.6869	39.6008	39.9714
	240	109.7636	40.6522	37.4754	25.4196	28.1440	25.5633
	270	121.1759	33.0384	51.9177	33.0362	29.2665	23.3535
	300	119.8826	49.8583	55.2947	31.1271	36.3799	23.1848
	330	128.0396	46.2091	61.8191	33.9593	46.4146	31.9908

Table 4.3: Average of the distance error and standard deviation on the error for all joints in pixels for different viewpoints at different angles in degrees. All images are of dimensions 224x224 pixels. X and Y axis refer to the axis shown in Figure 4.1, and the angle at 0 degree refers to the viewpoint shown in the same figure. The rotation is done counter-clockwise and only to one axis at a time.

imally on the structure of the foot. We have focused our attention on the joints. Those results can be seen in Table 4.3. The results that can be seen in this table show the algorithm's unreliability described earlier. We do see better results for 240 to 300 degrees around the X axis with RGBD images and depth maps. The best results are at 270 degrees, with a mean of 7.26 pixels and a standard deviation of 13.09 pixels for RGBD images, and a mean of 9.80 pixels and a standard deviation of 13.16 pixels for depth maps. These viewpoints show the front of the foot with the toes pointing toward the camera. We believe that those results are misleading. With a silhouette detection that is within the error values shown in Table 4.1, the segmented section would show a smaller area that if the camera was capturing the side of the foot. We can combine this with the fact that the depth maps, either alone or as the fourth channel of the RGBD images, will show a high difference between the distance of the toes from the camera and the distance of the leg from the camera, which increases the accuracy of the silhouette detection. We can come to understand that all the joints will land within the segmented outline which is not a large area of the image. Then, even if the joints are scattered at random with the detected silhouette, they will not be far from their label's position. This effect is shown in the standard deviation that is higher than the mean values. For the other viewpoints, the standard deviation is around half of the mean. Those are high values, meaning that the results are inconsistent for every input types and every viewpoint. As for why the RGB images do not show better results for the viewpoints at 240 degrees to 300 degrees, we believe that it was unable to successfully detect the silhouettes.

From the results we have seen, we come to the conclusion that the depth maps, either alone or in RGBD images, improve drastically the results for the joints. It provides more information to find the location of the joints, and produces silhouettes that better show where the foot is. However, it is not enough as they are still not close to what is expected. We can realize that the foot does not present enough visible cues for our system to pinpoint with accuracy the locations of the joints. Seeing that the joints of RGBD images and depth maps land on the foot compared with RGB images, we come to understand that a different approach can definitely improve the results when using those types of input.

To improve detection of the 2D cues, the algorithm should be changed. Instead of using the same network to get both the silhouette and the 2D joints, a customized algorithm for each task would be more appropriate. The method used for each 2D cue could rely on the structure of the foot instead of using only visual cues. Our method only gets the joints' location based on foreground detection and features extracted at a local level in the input image. A method that would also use the structure of the foot will have a better understanding of the global information found in the image. To do so, the method would have to predict where the object is in the image and fit a 2D template on it. The template

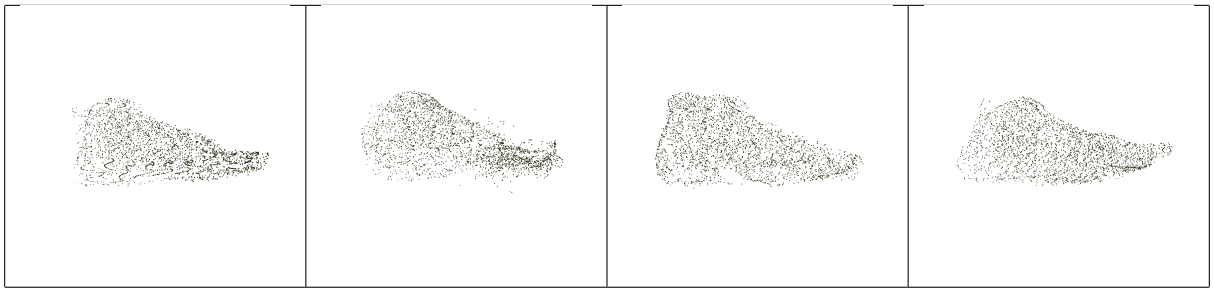

Figure 4.4: Examples of 3D meshes reconstructed by our system using EdgeConv.

could be a projection of our kinematic foot model which describes the structure of the foot. If the algorithm is able to infer the rotation and size of the foot in the image, then those transformations can be applied to the kinematic foot model which can then be projected on the image. At this point, the method would have to make small adjustment to the projection of the kinematic foot model to fit it properly to the foot in the image. Another improvement, instead of changing the whole method, can be to impose hard constraints through the loss function for the joints' detection algorithm to frame where the joints can be found. One of such hard constraint could be to have all the tip of the toes close to each other. Another hard constraint could be to force the joints' location to be in the detected silhouette. This would help to network to converge toward features of the foot that indicates the location of the joints.

4.2 3D Reconstruction

We have described two 3D reconstruction algorithms in 3.3 3D Reconstruction. We evaluate both in this section. We assess how close the vertices of the predicted meshes are to their ground truths. We are not evaluating if the meshes show the correct foot measurements in this section, but if the inferred meshes are structured at a local level like their labels. However, since the measurements are intrinsically related to the meshes, we are able to see their influence in the results presented in this section.

Using EdgeConv, we were able to reconstruct the 3D structure of the foot. It appears as a point cloud as we do not have the polygon values to make it a mesh. EdgeConv generates a new set of points that is related to its input only in structure. This means that we cannot use the face values of the input as we do with Inception Spiral Module. Examples of reconstructed point clouds can be seen in Figure 4.4. Being our first iteration, EdgeConv was designed to directly input a point cloud instead of joints and an outline.

Method	Averaged distance
EdgeConv	11.3751
ISM with RGB	107.9240
ISM with RGBD	9.8856
ISM with depth maps	51.7147

Table 4.4: Averaged distance between each vertex of the mesh and its label in millimetres. ISM stands for Inception Spiral Module.

Going from a depth map to a point cloud is easy with libraries such as Open3D [104]. The averaged distance between the points of the inferred point clouds and their labels are presented in the first entry of Table 4.4. As we can conclude, even when using a point cloud as input, we do not obtain the best averaged distance, with 11.38 millimetres for EdgeConv against 9.89 millimetres for Inception Spiral Module using RGBD images as input. This can be explained by the fact that EdgeConv do not deform a template like Inception Spiral Module, but generates new position for each point it generates. This means that the quality of the point cloud at a local level is inferior. It also means that this method is more susceptible to noise as we can see in Figure 4.4. The noise is particularly strong where there are more details to reconstruct, such as the toes. This noise is another reason why the results of EdgeConv is higher than for Inception Spiral Module with RGBD images.

Using Inception Spiral Module with as input the joints and the silhouette obtained in the previous section, we predict meshes as seen in Figure 4.5. We can see that those meshes are not perfect. They have some deformities and their surface is not the smoothest due to noise. The main reason for this is the input which is the silhouette and the joints produced in the previous section. As the input does not reliably represent the joints and the outline of the foot, we can understand why the meshes do not have the expected shape. Still, the predicted meshes are looking good considering that the input is below expectations. As for quantitative values, the results are presented in Table 4.4. We come to the conclusion that RGBD images are by far the best input for our system for the 3D reconstruction which can be confirmed by Figure 4.5. RGBD images show an averaged distance of 9.89 millimetres, against 51.71 millimetres for depth maps and 107.92 millimetres for RGB images. The main reason why the RGB images produce meshes as far from their labels is that their meshes are smaller than their ground truths. It is due to the fact that they do not have a good reference for scale like RGBD images have with its depth channel. The information from the depth channel of the RGBD images come from the skip layers we are using between the encoders and the decoders. We realized with this experiment that the edge length loss

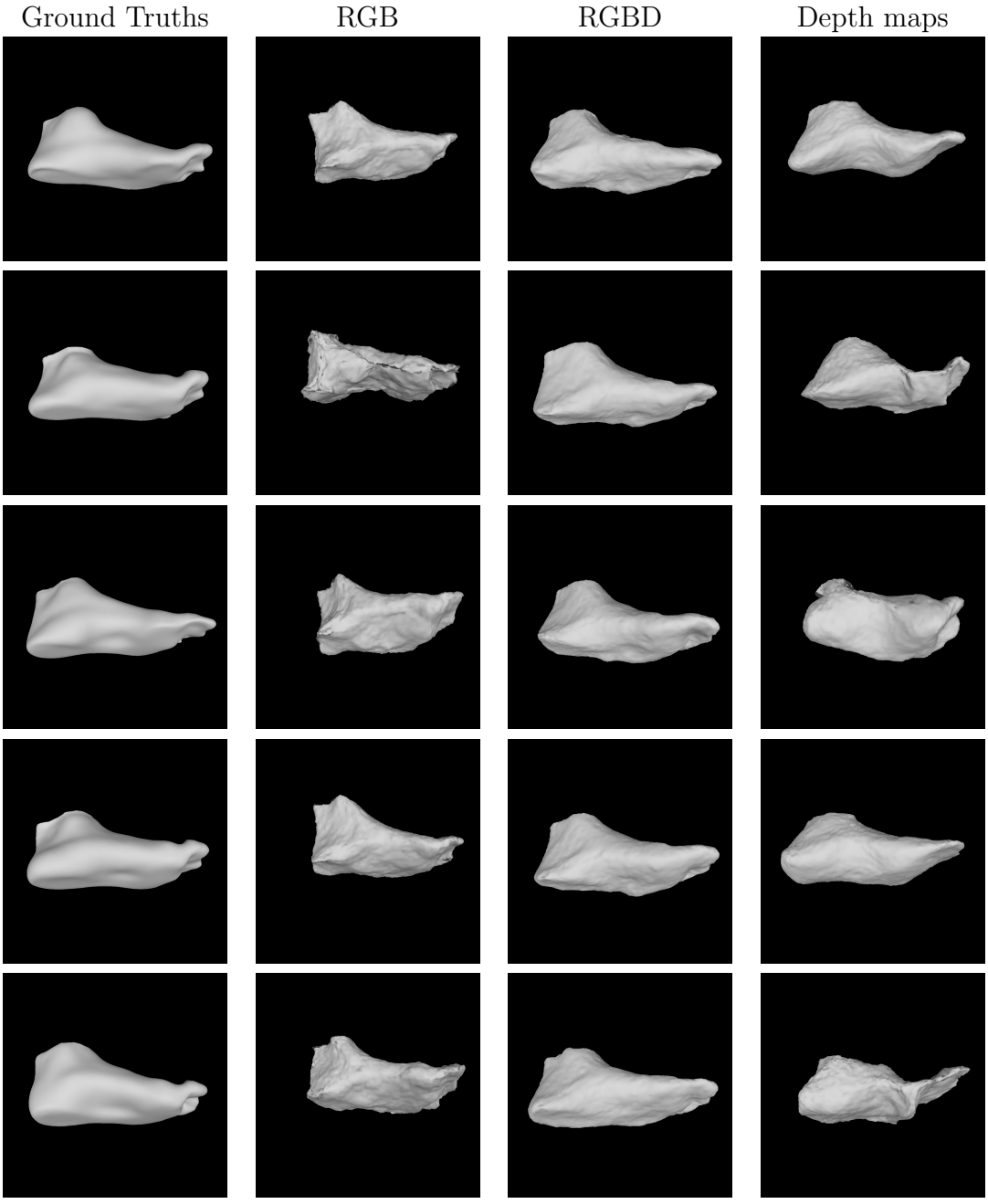

Figure 4.5: Examples of 3D meshes reconstructed by our system using Inception Spiral Module. First column shows the expected meshes, while the other three show the output for different types of input. Each row shows a different example.

has the effect to shrink the meshes and make them crumble into themselves. When they do not have strong indications from the network to scale the output, the meshes tend to get smaller and more distorted. It also happens for meshes inferred from depth maps, but not as severe as for RGB images. When using depth maps, we do have indications of the scale of the mesh. However, as we have seen in the previous section, the masks generated for the depth maps are bigger than the actual object in the image. This means that our system considers a halo of Gaussian noise around the foot to be part of the object. As the Gaussian noise has an amplitude much smaller than the depth values of the foot, the meshes get shrank and deformed to adapt to their input. When using RGBD images, the three colour channels help in this separation between noise and object. This is also one of the reasons why some meshes from the depth maps are as deformed as seen in Figure 4.5. This halo of Gaussian noise being included in the depth maps is the reason why the quantitative results of the depth maps are worse than the results of RGBD images.

From the results of this section, we can understand that the most adapted method for our needs is Inception Spiral Module. EdgeConv showed that it was able to reconstruct the 3D shape, but is not adapted to our application as it does not produce a reliable reproduction at a local level. We have two major issues with Inception Spiral Module. The first one is that the input to this module of our pipeline is below our expectations, which was bound to provide its share of imprecision. The second issue is that our virtual dataset does not have a way to give the scale of what is shown in the images, especially for the RGB images. This lack paired with the edge loss which crumbled the meshes into themselves create smaller 3D reconstruction than expected. What would be needed is scale indicators that can be used to put the image in relation to a world coordinate system. Another possible improvement is to change the edge loss. A better loss function would have the ability to monitor the structure of the template mesh deformed by Inception Spiral Module without forcing the dimension of the meshes to reduce because of the lack of information. Nonetheless, Inception Spiral Module and the architecture we used with it proved to be up to the task of reconstructing the foot. All the recreated meshes can be recognized as feet, even if they are distorted. The output has the advantage to be a mesh instead of a point cloud, which has more applications as stated in 2.3 3D Geometry. Also, while the edge loss did not meet our expectation, the L1 loss behaved as intended as the results from RGBD images have shown.

4.3 3D Pose Estimation

One of the goals of our system is to measure the foot. This section will address the accuracy of the 3D pose estimation and of the two measurements we have decided to evaluate from the eight measurements suggested by Dr. Orvitz [56], the length and the width.

We evaluate the 3D pose by comparing the labels from our virtual dataset with the interpolated joints from the inferred meshes. We have eighteen joints which we evaluate individually with the exception of the midpoint. We use this joint to make sure the predicted pose and its label are aligned. This is the obvious choice as all the other joints are connected by their branch to it. The distances between each joint and its label, in millimetres, are presented in Table 4.5. EdgeConv is not mentioned in this table as we did not try to recover the 3D pose with this method. We have only used feet in a canonical pose when developing it. From the table, we can see that RGBD images produce the best results. It is not surprising considering the results presented in the last section. Again, the distorted and the smaller meshes are the cause of the disparities between the results of each different type of input. For all, the ends of branches are the joint farther from their labels. This is mainly caused by the noise from the 3D reconstruction. Another reason for why the joints are not close to their labels is that the deforming of the template seems to have a smoothing impact on the 3D pose. When the 3D pose is far from what we can see in Figure 3.9, the meshes show a 3D pose that is between what can be seen in this figure and the expected 3D pose. This effect can be seen in Figure 4.5. Looking at the toes, we can see that often they do not show the same magnitude of movement compared to the rest of the foot. This smoothing effect comes from the outline and the joints inferred in 4.1 Extraction of 3D Information. We believe that this outcome comes from the Inception Spiral Module. During training, it receives silhouettes and joints' landmarks that have no correspondence to the labelled meshes used as ground truth. Since it cannot match its input with its expected output, our network learns to produce an average foot in an average pose that would satisfy most labels. It then uses information from the skip layers between the encoder and the Inception Spiral Module's decoder to create small variations in the predicted meshes.

To evaluate the measurements, we start by bringing the 3D pose from the predicted mesh into a canonical pose. This canonical pose can be seen in the Figure 3.9. When we acquire the predicted pose of the 3D reconstruction, we perform inverse kinematic computations to acquire the angles of the joints. With the known predicted pose, we can then perform forward kinematic computations to bring every joint to a null angle. With the feet in the canonical pose, we can measure them accurately to get their length and their width. We present the difference between the measurements of our predicted

Joints	RGB	RGBD	Depth maps
Midfoot	0	0	0
Proximal phalanx 1	73.7352	8.7766	59.1848
Distal phalanx 1	95.2273	7.8123	68.8644
Tip 1	145.3024	11.8725	88.3400
Proximal phalanx 2	75.7431	8.5281	63.5305
Distal phalanx 2	87.5758	10.0677	74.3038
Tip 2	108.7411	12.2707	91.9460
Proximal phalanx 3	76.7667	9.1221	61.8672
Distal phalanx 3	84.3324	12.8453	71.1015
Tip 3	106.3652	15.0703	83.2080
Proximal phalanx 4	67.4602	9.4579	59.0092
Distal phalanx 4	74.1224	13.3048	65.7476
Tip 4	93.3045	14.3247	74.9543
Proximal phalanx 5	70.6358	9.3169	52.6291
Distal phalanx 5	67.5797	10.4133	57.8618
Tip 5	74.6813	13.0806	66.5793
Ankle	127.0723	8.8717	49.6120
Heel	189.6622	13.1635	77.7892
Mean	95.1946	11.0764	68.6193

Table 4.5: Average of the distances between each 3D joint and its label in millimetres. The order of the joints follows Figure 3.10. The joints mentioned in the first column correspond to the joints of our kinematic foot model shown in Figure 3.9. The midfoot is the central joint from where each branch of the kinematic foot model starts. It was used to align each joint with its label and is not taken into account in the mean results. The Proximal phalanges refer to the closet joints from the midfoot in the branches that go to the toes. The Distal phalanges are the joints following in the same branches. The Tips refer to the joints that represent the tip of the toes at the end of the branches. The order of the joints follows Figure 3.10. The ankle represents the joint in the articulation of the same name while the heel represents the end of this branch. The three other columns show the results for each different type of input.

feet and their labels in Table 4.7. It is not a surprise that RGBD produces the best measurements after seeing the results of the 3D pose for individual joints, with an error of 13.02 millimetres on length and 6.41 millimetres on width. However, it is not enough for a shoe size recommendation. As we have seen in 2.1 Foot Measurement, the increment for half a size in the American and UK shoe sizes, which is the smallest increment in the most common shoe sizing systems, is of 4.23 millimetres. We can see that none of our methods was able to achieve this precision. Even when considering the European with its increment of 6.67 millimetres, we do not meet the required precision. As for width, the measurement in the shoe sizing system depends on the length size. We use the same value of 4.23 millimetres to compare the width as it is one of the largest increment on the width of the American shoe sizing system. Although Inception Spiral Module with RGBD images and with depth maps come closer to this increment value, the width is still too high to accomplish our goals. As for the measurements obtained with EdgeConv, the difference comes mainly from the noise in the mesh. A way to solve this noise problem would be to find the outliers in the mesh that constitute this noise and remove them. We did not do this to keep the results relevant between EdgeConv and Inception Spiral Module as noise in the mesh is not the principal problem for Inception Spiral Module.

Using specific viewpoints to measure the meshes do not lead to better measurements. We have evaluated the precision of the position of the joints in 3D space, shown in Table 4.6, and the precision of the two measurements, the length shown in Table 4.8 and the width shown in Table 4.9, for multiple angles of view. We come to the conclusion that the results are inconsistent. This inconsistency is caused by the failure to get accurate positions for the 2D joints and the shrinking effect on the 3D reconstruction, which have a too important impact on the results to be able to determine if a viewpoint can lead to better measurements. The data shown in the mentioned tables present a high standard deviation, proving the unreliability of the results. For some cases, the error on the measurement is twice as the presented average, and in other cases the error is near zero. There are no patterns to where the results for specific viewpoints are lower. Nonetheless, the three tables highlight the performance of RGBD images and display the potential of our system to use this type of input to get measurements from the foot once our weaker parts of our system are resolved. The quantitative results do not show any limitations to the measurement of the foot using the chosen method, meaning that our kinematic foot model was successful in the applications we designed it for.

We come to the conclusion that the problems encountered for the detection of the joints and the problem of the size of the meshes are weighting down the 3D pose estimation. The results of this section are inconclusive as the effect of those problems transpires heavily in the quantitative data. The only issue that can be seen is that the 3D pose is smoothed

		RGB		RGBD		Depth maps	
Axis	Angle	Mean	STD	Mean	STD	Mean	STD
X	0	129.1840	136.4098	9.1719	4.0395	118.8415	31.2136
	30	165.1673	178.1795	11.3431	4.5224	112.0987	30.4351
	60	244.8462	353.5721	13.3171	5.4528	92.3884	38.3581
	90	223.1415	294.6888	10.8116	5.4934	49.8611	39.8873
	120	206.5543	268.5045	12.7296	4.9869	36.5079	25.8877
	150	212.7466	270.0473	10.5727	5.0130	48.5498	30.0709
	180	111.9091	113.2947	10.2322	4.3297	109.7411	40.9049
	210	81.8583	30.3397	12.8548	4.9586	114.5839	41.1498
	240	122.6360	128.3805	10.8947	4.5243	87.8807	47.4673
	270	173.9548	246.6972	9.3411	4.0890	58.6018	39.9991
	300	193.0865	188.8963	8.6865	3.5943	104.8561	33.3690
	330	146.4879	163.9832	11.4262	4.4083	99.6092	36.3598
Y	0	129.1840	136.4098	9.1719	4.0395	118.8415	31.2136
	30	129.2834	101.7185	8.1338	4.1584	101.5997	37.2590
	60	94.7057	33.6611	9.8327	4.3086	80.3531	38.6780
	90	82.3300	22.32569	11.0552	4.9370	73.0109	39.2220
	120	84.6310	45.7631	10.2039	4.3974	46.2453	37.5198
	150	92.6202	62.4193	11.3192	4.4150	72.8982	45.3013
	180	111.9091	113.2947	10.2322	4.3297	109.7411	40.9049
	210	117.7183	94.6409	11.0723	5.1929	107.1022	30.9631
	240	101.7966	73.7302	11.6792	5.2216	87.7193	40.1777
	270	93.7354	51.0918	12.8190	4.8885	87.4082	29.8712
	300	83.9253	15.7232	11.4899	5.4841	73.191	32.1366
	330	136.3576	131.9301	11.4381	4.5320	105.5914	33.5029

Table 4.6: Average of the distance error and standard deviation on the error for all joints in millimetres for different viewpoints at different angles in degrees. X and Y axis refer to the axis shown in Figure 4.1, and the angle at 0 degree refers to the viewpoint shown in the same figure. The rotation is done counter-clockwise and only to one axis at a time.

Method	Length		Width	
	Mean	STD	Mean	STD
EdgeConv	31.9814	18.9065	20.8811	12.6861
ISM with RGB images	137.0694	72.3914	37.7235	33.6489
ISM with RGBD images	13.0168	9.0346	6.4146	4.6543
ISM with depth maps	53.9000	36.0676	6.8823	5.3234
ISM with real world RGB images	156.6162	46.4440	49.3263	15.2973

Table 4.7: Averaged error and standard deviation on the error in length and width for each method in millimetres. ISM stands for Inception Spiral Module.

toward a canonical pose. However, the faulty silhouette and joints' landmark detection is again the cause of this issue as it forces the Inception Spiral Module to learn how to create an average foot with an average pose. The bright side to this is that it goes to show that the Inception Spiral Module is able to recreate the foot from almost nothing, and we can expect it to achieve much better predictions with an improved 2D cue extraction method. Moreover, the kinematic foot model proved to be efficient in providing measurement of the foot. From the results we obtained with RGBD images as input, we also see that adding scale indicators in the learned process has the potential to lead to a precise recommendation of length and width.

4.4 Generalization with Real World Data

To test generalization, we have used real world RGB images of feet with our system. As those images come from a private dataset, we are not able to show them here. This private dataset contains around 100,000 pictures of left feet. In every image, there is one foot flat on the floor in an indoor setting. The pictures are taken around the Y axis shown in Figure 4.1. Since we cannot show this private dataset, we have instead chosen to illustrate in Figure 4.6 the use of our system on real world images by using royalty-free images [81]. Considering the results presented by the previous sections of this chapter, we have obtained the expected results with real world data. We only had access to the length and width of the feet in the real world images as labels, which is why we have only tested the real world data for those measurements. With them, we obtained the results shown in the last entry of Table 4.7. Although the mean results are worse than with synthetic RGB images, with 156.62 millimetres on length for real world images against 137.07 millimetres on length for synthetic images, the standard deviation shows that both inputs lead to

		RGB		RGBD		Depth maps	
Axis	Angle	Mean	STD	Mean	STD	Mean	STD
X	0	95.2195	63.7565	11.9164	8.1790	29.7702	25.0835
	30	122.6232	42.5451	10.8957	7.8421	35.3282	30.7953
	60	173.6128	93.0083	14.7944	10.0273	47.4624	37.1390
	90	163.6561	114.5945	13.6362	11.1723	41.6987	29.2272
	120	148.4966	85.1209	12.7264	6.3939	36.8834	16.4084
	150	133.8308	67.9266	12.6051	10.4197	53.8345	28.7886
	180	109.6979	71.0780	13.2038	7.9849	50.9711	37.1899
	210	124.8644	38.3651	14.5840	9.7015	24.4927	20.2315
	240	140.2061	61.1687	13.8404	8.8095	79.0590	63.3163
	270	135.5023	75.3233	12.3089	10.3610	44.7248	36.5704
	300	215.1601	147.4572	14.1142	7.8011	59.6627	38.8276
	330	151.5992	89.3508	10.5449	7.2333	49.9465	24.4530
Y	0	95.2195	63.7565	11.9164	8.1790	29.7702	25.0835
	30	116.4295	116.6471	14.1478	8.4603	51.3150	35.3383
	60	97.4340	60.2282	9.4490	7.6597	55.9462	35.2343
	90	110.4910	50.3771	13.4152	9.0116	81.0861	46.2267
	120	164.3012	55.3656	9.8974	5.8648	54.1876	33.4450
	150	167.6986	80.3067	14.1337	9.1859	86.7290	47.1532
	180	109.6979	71.0780	13.2038	7.9849	50.9711	37.1899
	210	123.3579	102.8866	12.1763	10.4304	53.8723	46.9966
	240	152.6220	66.7061	12.4751	11.5764	52.2121	33.5924
	270	124.1857	38.4960	16.9174	9.2950	43.7308	38.5158
	300	149.0296	16.0573	14.0627	11.8904	47.7003	26.2752
	330	161.7572	75.6201	11.8935	8.0175	49.7267	24.7024

Table 4.8: Averaged errors in length for each method in millimetres for different viewpoints at different angles in degrees. X and Y axis refer to the axis shown in Figure 4.1, and the angle at 0 degree refers to the viewpoint shown in the same figure. The rotation is done counter-clockwise and only to one axis at a time.

		RGB		RGBD		Depth maps	
Axis	Angle	Mean	STD	Mean	STD	Mean	STD
X	0	24.3633	22.8677	5.4138	4.4575	6.0248	3.8152
	30	77.7357	124.4068	5.3518	4.1185	8.2141	11.1440
	60	28.9778	17.7209	7.4859	5.8603	6.6004	4.2213
	90	27.4013	23.9594	5.4189	4.1548	5.7054	5.6064
	120	56.7557	72.5454	10.1107	5.9719	6.4083	4.2650
	150	121.5672	179.6723	6.1208	4.0320	6.6688	5.6551
	180	28.9286	19.3266	6.7383	5.1349	8.1295	6.5377
	210	24.2322	8.8900	6.3430	5.4321	6.5732	4.6063
	240	24.8769	12.9334	6.8433	4.9717	7.0280	5.0985
	270	25.7308	12.5217	7.2545	5.4067	7.0680	5.6384
	300	53.2812	66.9569	7.0490	4.2006	6.8992	6.1007
	330	51.5064	62.8744	5.0158	4.3069	6.4794	4.5128
Y	0	24.3633	22.8677	5.4138	4.4575	6.0248	3.8152
	30	39.6391	39.9165	5.4278	3.6971	5.6621	4.7359
	60	21.5304	10.5684	4.6925	3.4186	4.6244	3.6474
	90	21.0344	9.3269	5.3118	3.6947	6.6004	5.1630
	120	30.5567	19.1762	5.6853	4.4491	5.6259	3.9057
	150	41.7562	44.6171	6.0660	4.7473	5.2338	5.0513
	180	28.9286	19.3266	6.7383	5.1349	8.1295	6.5377
	210	28.3005	26.8038	7.3049	3.0220	10.3816	7.1793
	240	28.2943	22.5795	6.4289	5.7068	9.6351	7.3982
	270	22.0485	10.2491	8.1934	6.3090	6.4961	4.6816
	300	26.4841	7.5249	6.6733	5.2590	7.1784	4.8572
	330	34.4362	20.6993	5.6503	3.6454	6.1040	3.7527

Table 4.9: Averaged errors in width for each method in millimetres for different viewpoints at different angles in degrees. X and Y axis refer to the axis shown in Figure 4.1, and the angle at 0 degree refers to the viewpoint shown in the same figure. The rotation is done counter-clockwise and only to one axis at a time.

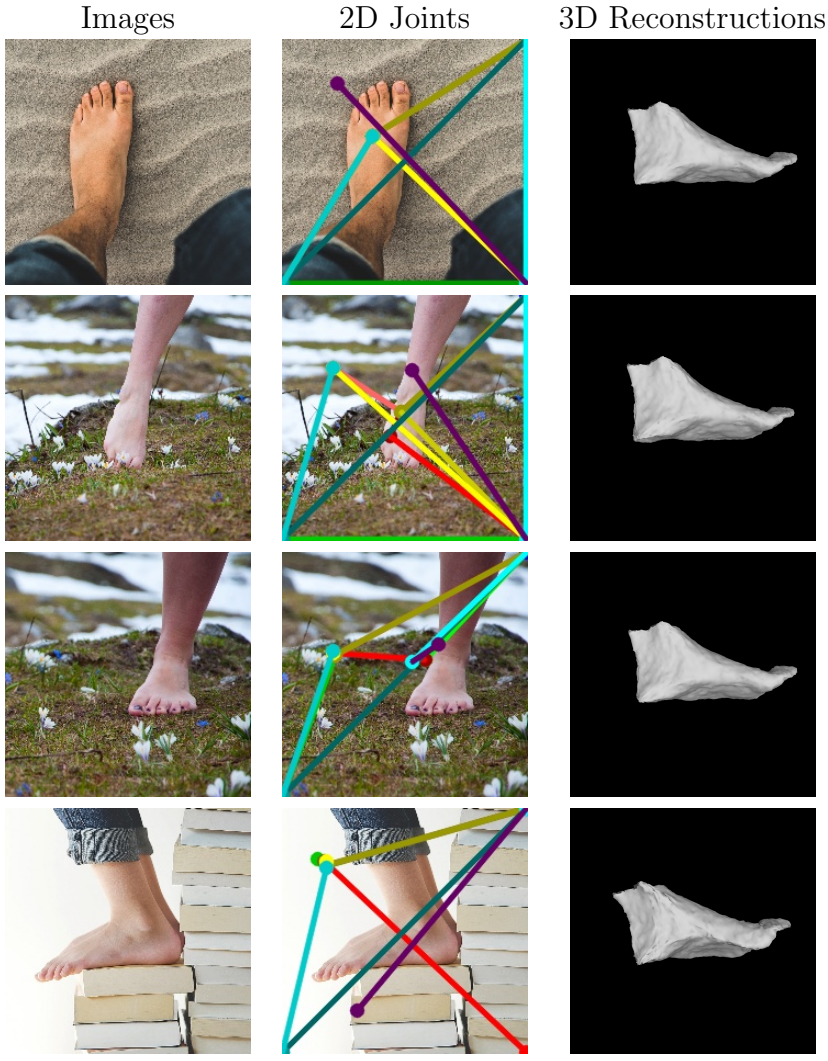

Figure 4.6: Examples of our system's outputs on real world data. First column shows the input of our system which is RGB images. Second column shows the predicted 2D joints. Third column shows the 3D reconstruction of the foot. Each row shows a different example.

similar measurements, with 46.44 millimetres for real world images and 72.39 millimetres for synthetic images. For synthetic images, the standard deviation tells us that the acquired measurements are more spread than for the real world images. It tells us that the real world data generate more consistent reconstructions than our synthetic dataset. We believe that this comes from the texture that contrasts more with the scene in which it was taken and from the more defined variations in the feet.

With our system's flaws presented in the previous sections, we cannot conclude properly on the generalization of our system over real world images. We have seen that the mean of the synthetic data is within the same uncertainty range as is the real world data due to the large values of standard deviation. This hints that our system shows a good generalization, but we cannot confirm the correlation.

The real world data we had access to was missing some case that could be seen in the wild. For example, none of the feet has shown tan lines of any sorts. In the same idea, there was no tattooed feet in the dataset. Also, there was no scene with strong luminosity that cast noticeable shadows. Such cases were not handled either by our synthetic data. It would certainly be a challenge to acquire the joints from such data.

4.5 Summary

This chapter presented the results of our experiments to test our system. Multiple flaws became apparent. The production of our intermediary results is our biggest problem. We have seen that the segmented silhouette is either absent for RGB images or includes large unwanted sections of the image for depth maps and RGBD images. We have seen that the precision of the location of the joints suffers from the bad silhouette segmentation. The joint localization also suffers from the algorithm having difficulties to capture the visible cues that indicate where the joints are. The other major flaw of our system is the shrinking of our meshes. We have discovered that the edge loss makes the meshes crumble into themselves when there is no clear indication of scale. Since our training data do not contain any scale indicators, we cannot indicate to our system what dimensions the meshes should have. This is worse for the RGB images, as depth maps and RGBD images can fall back on their depth values for some scale indications. Another flaw of our system that we have noticed is that the 3D pose estimation is averaged toward the canonical pose where all the angles of the joints are null. We believe that this problem comes from the disparity between the intermediary results and the labelled meshes. As Inception Spiral Module cannot find relations between the two, it learns to generate average feet with average poses. However, it is a testimony to the strength of Inception Spiral Module and

the architecture we have engineered, as it is able to reconstruct a mesh of a recognizable foot from a distorted input. For all those flaws, we have seen that using a specific angle of view do not improve the observations. Nevertheless, Inception Spiral Module and the architecture we designed to support it behaved exactly as intended, allowing us to deform the template in a controlled manner. From those meshes, measurements can be efficiently estimated with the help of our kinematic foot model, which has successfully fulfilled the purpose for which it was engineered. Additionally, generalization was tested using real world RGB images. We saw that the results from the real world data are in the same uncertainty range as the results of synthetic RGB images. We were not able to conclude if our system has difficulties to generalize as our system's flaws discussed earlier have a great impact on those results. However, the standard deviation on the error was smaller for real world data than for synthetic images, showing that the results from real world images are more consistent than with synthetic images. We believe that the skin textures and tones in real world images present a stronger contrast with the background than in our synthetic images, which improves the intermediary result detection. It shows that training on a real world dataset could improve the results, especially when using RGB images as input. Finally, our best input type for measuring the foot was RGBD images. However, it generated measurements of our meshes with imprecision that were too high to be applied to a shoe recommendation task, being higher than an increment of one size of all the shoe sizing systems. Nonetheless, it goes to show that our system has the capacity to produce precise measurements of the foot using RGBD imaging. Although they require more tuning, RGB images and depth map have also shown this potential.

Chapter 5

Conclusions

This thesis was about a system that produces a 3D mesh and a 3D pose estimation of a foot out of a single image. To do so, the image was first used to extract the silhouette and the joints of the foot using a Stacked Hourglass Network. This step had the goal to get meaningful information from the image to be able to understand how the foot is projected into 2D space. Then, with the acquired 2D cues, a 3D reconstruction was done by deforming a template of a 3D mesh of a foot with an algorithm called Inception Spiral Module. The pipeline ended with the recovery of the 3D pose from the predicted 3D mesh. Using the known structure of the 3D mesh, the pose was estimated by corresponding it to our kinematic foot model. Since this is a learned approach, a dataset was needed. With no suitable dataset available, we have created our own. To do so, we developed a PCA model of a 3D mesh of a foot using a dataset of 3D meshes of feet. Then, we developed a kinematic model of the foot with eighteen joints based on popular biomechanical foot models. Merging the kinematic foot model and the 3D mesh foot model, we were able to generate different feet in any possible poses.

The developed system did not lead to the results we were hoping for. We were aiming to have a length and a width under one size increment of the American shoe sizing system, which is of 4.23 millimetres. We tried three different types of input to achieve this result. Our best results came from RGBD images, with an error on length of 13.02 millimetres and on width of 6.41 millimetres. Using depth maps, we obtained an error on length of 53.90 millimetres and on width of 6.88 millimetres. Using RGB images, we got an error on length of 137.07 millimetres and on width of 37.72 millimetres. To test generalization, we have also experimented with real world RGB images. It led to an error on length of 156.62 millimetres and on width of 49.33 millimetres. Those results were in the same uncertainty range as with the synthetic RGB images, which means that we cannot conclude on the

generalization capabilities of our system. None of the tried input types resulted in an accuracy that would be acceptable for a shoe size recommendation task.

We believe our system has two main flaws. The first one is in the production of the intermediary results which are the silhouette and the joint location. The inaccuracies that are introduced in this step have a significant impact on the rest of the system. The second flaw is the 3D reconstruction algorithm with Inception Spiral Module that shrinks and crumbles the predicted meshes due to a lack of scale information. The lack of scale indicators of any kind reduce drastically the quality of our reconstructions. Those two flaws are the reason why our measurements of length and width are as inaccurate. We have also noticed that they force our system to learn an average foot in an average pose. The implications of this effect is that the generated meshes do not show all the expected variations, and most of them are not as ample as they should be.

Our system did achieve some of its goals. While the module that generates the 2D cues did not meet expectations, Inception Spiral Module and the architecture we designed with it functioned as expected and was able to produce meshes of feet from a flawed input. In the case of RGBD images as input, the measurements acquired from the meshes had an impressive accuracy knowing that the input was not as precise as planned. Additionally, the kinematic foot model we engineered was able to accomplish its purposes to the extent we desired. With it, we were able to guide the mesh deformation and get all the measurements of the foot we required.

With this being said, there are many refinements that could be done to ameliorate the accuracy of the system. Most improvements should focus on finding the location of the joints as it is the weak link of the pipeline. One improvement is to change the algorithm that predicts the two intermediary results. Instead of using the same network for both, a customized network for each task would lead to better results. The networks could also use the structure of the foot to predict those intermediary results instead of relying only on local visual cues. To do so, it could try to project our kinematic foot model in the images by predicting the rotation and size of the foot in the image. Then, only slight modifications would have to be made on the projection to get accurate joints' positions. Another improvement would be to put hard constraints on the algorithm that detects the joints. A hard constraint could make sure that all the tip of the toes are close together as the toes are close to one another on the foot. Another constraint could be that the joints must be found inside the previously found silhouette. This would help the model to converge towards features of the foot that indicates the location of those joints. As for ameliorations that can be made on the 3D reconstruction, the edge loss should be replaced by a function that monitors the structure of the deformed meshes without forcing the dimension of the meshes to reduce. Using scaling indicators could also be an amelioration,

from the dataset, to improve the 3D reconstruction.

Another improvement to be made on this system is to train it on real life data. Although the virtual dataset is made to resemble real life data, it is no confirmation that our system will work as well in a real application. Additionally, the virtual dataset had a poor display of skin texture in the images. We observed that the results from real life data had a lower standard deviation than the results from synthetic data, despite producing slightly bigger errors on measurements. It shows that real life data leads to more consistent 3D reconstructions, which we attribute to better skin textures and tones in real life images. We can conclude that using real life data to train our system can improve its performance by allowing easier 2D joints detection. Also, real life data will show unusual cases that will further test the ability of the neural network to work with new data. Although we have tested the generalization on real world RGB images, there are some cases that were still not tested. No feet in the real world data that we have used showed tan lines or tattoos. Also, there was no scene with strong luminosity such as direct sunlight which would cause strong shadows. Those are cases that can be problematic when trying to find the 2D joints which we use to achieve the 3D reconstruction and pose estimation.

Even if the system presents high inaccuracies, it goes to show that a system that can infer a 3D mesh of the foot from a single image and extract its 3D pose is possible. However, the subject of the foot in 3D mesh recovery is not very popular. Hopefully, this thesis will help bring interests on this specific problem. It has brought several interesting points. The pose of the foot can be used to bring a foot in any pose into a canonical pose for more accurate measurements as we did in 4.3 3D Pose Estimation. The eighteen joints kinematic foot model, presented in 3.5.2 Kinematic Foot Model, we have developed from biomechanical foot models is useful to understand how the foot can move without needing the knowledge of the anatomical components. Finally, the system as a whole has clear applications which it will be able to fulfill once improved.

References

[1] Abdul Hadi Abdul Razak, Aladin Zayegh, Rezaul K. Begg, and Yufridin Wahab. Foot plantar pressure measurement system: A review. *Sensors*, 12(7):9884–9912, 2012.

[2] Brett Allen, Brian Curless, and Zoran Popovic. The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Trans. Graph.*, 22:587–594, 07 2003.

[3] André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. https://distill.pub/2019/computing-receptive-fields.

[4] Alfredo Ballester, Eduardo Parrilla, Julio Vivas, Ana Piérola, Jordi Uriel, Sergio-Antonio Puigcerver-Palau, Paola Piqueras, Clara Solves, Marisol Rodríguez, Juan González, and Sandra Alemany. Low-cost data-driven 3d reconstruction and its applications. *6th International Conference and Exhibition on 3D Body Scanning technologies*, Technical Session 10: RGB-D Sensors & Low Cost Systems, October 2015.

[5] Miguel Ángel Bautista, Walter Talbott, Shuangfei Zhai, Nitish Srivastava, and Joshua M. Susskind. On the generalization of learning-based 3d reconstruction. *CoRR*, abs/2006.15427, 2020.

[6] Jan Bechtold, Maxim Tatarchenko, Volker Fischer, and Thomas Brox. Fostering generalization in single-view 3d reconstruction by learning a hierarchy of local and global shape priors. *CoRR*, abs/2104.00476, 2021.

[7] ClipArt Best. Best photos of feet outline template printable. http://www.clipartbest.com/clipart-RiGBMBkdT. Accessed: 2021-08-26.

[8] Chris Bishop, Gunther Paul, and Dominic Thewlis. Recommendations for the reporting of foot and ankle models. *Journal of Biomechanics*, 45(13):2185–2194, 2012.

[9] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt. Openmesh: A generic and efficient polygon mesh data structure, 2002.

[10] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

[11] Andrew K. Buldt and Hylton B. Menz. Incorrectly fitted footwear, foot pain and foot disorders: a systematic search and narrative review of the literature. *Journal of Foot and Ankle Research*, 11(1):43, 2018.

[12] Hongrui Cai, Yudong Guo, Zhuang Peng, and Juyong Zhang. Landmark detection and 3d face reconstruction for caricature using a nonlinear parametric model. *Graphical Models*, 115:101103, 2021.

[13] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018.

[14] M.C. Carson, M.E. Harrington, N. Thompson, J.J. O'Connor, and T.N. Theologis. Kinematic analysis of a multi-segment foot model for research and clinical applications: a repeatability analysis. *Journal of Biomechanics*, 34(10):1299–1307, 2001.

[15] Xingyu Chen, Yufeng Liu, Chongyang Ma, Jianlong Chang, Huayan Wang, Tian Chen, Xiaoyan Guo, Pengfei Wan, and Wen Zheng. Camera-space hand mesh recovery via semantic aggregationand adaptive 2d-1d registration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[16] Yifei Chen, Haoyu Ma, Deying Kong, Xiangyi Yan, Jianbao Wu, Wei Fan, and Xiaohui Xie. Nonparametric structure regularization machine for 2d hand pose estimation. *CoRR*, abs/2001.08869, 2020.

[17] Sangbum Choi, Seokeon Choi, and Changick Kim. Mobilehumanpose: Toward realtime 3d human pose estimation in mobile devices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2328–2338, June 2021.

[18] Christopher B. Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *CoRR*, abs/1604.00449, 2016.

[19] The Brannock Decive Company. Our history. https://brannock.com/pages/about-us, 2019. Accessed: 2021-08-11.

[20] The Brannock Decive Company. Welcome to the brannock device co, inc. https://brannock.com/, 2019. Accessed: 2021-08-11.

[21] Angela Dai, Charles Ruizhongtai Qi, and Matthias Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. *CoRR*, abs/1612.00101, 2016.

[22] Dawson-Haggerty et al. Trimesh 3.2.0, December 2019. https://trimsh.org/.

[23] Michael Garland. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University, 1999.

[24] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas A. Funkhouser. Deep structured implicit functions. *CoRR*, abs/1912.06126, 2019.

[25] Shunwang Gong, Lei Chen, Michael Bronstein, and Stefanos Zafeiriou. Spiralnet++: A fast and highly efficient mesh convolution operator. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[26] Ankit Goyal, Hei Law, Bowei Liu, Alejandro Newell, and Jia Deng. Revisiting point cloud classification with a simple and effective baseline. https://openreview.net/forum?id=XwATtbX3oCz, 2021.

[27] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. *CoRR*, abs/1802.05384, 2018.

[28] Xian-Feng Han, Hamid Laga, and Mohammed Bennamoun. Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era. *CoRR*, abs/1906.06543, 2019.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[30] Philipp Herholz and Olga Sorkine-Hornung. Sparse cholesky updates for interactive mesh parameterization. *ACM Transactions on Graphics*, 39(6), 2020.

[31] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014.

[32] Hanbyul Joo, Natalia Neverova, and Andrea Vedaldi. Exemplar fine-tuning for 3d human pose fitting towards in-the-wild 3d human pose estimation. *CoRR*, abs/2004.03686, 2020.

[33] Ales Jurca, Jure Žabkar, and Sašo Džeroski. Analysis of 1.2 million foot scans from north america, europe and asia. *Scientific Reports*, 9(1):19155, 2019.

[34] Angjoo Kanazawa, Michael J. Black, David W. Jacobs, and Jitendra Malik. End-to-end recovery of human shape and pose. *CoRR*, abs/1712.06584, 2017.

[35] Pierre Karashchuk, Katie L. Rupp, Evyn S. Dickinson, Sarah Walling-Bell, Elischa Sanders, Eiman Azim, Bingni W. Brunton, and John C. Tuthill. Anipose: A toolkit for robust markerless 3d pose estimation. *Cell Reports*, 36(13):109730, 2021.

[36] S.M. Kidder, F.S. Abuzzahab, G.F. Harris, and J.E. Johnson. A system for the analysis of foot and ankle kinematics during gait. *IEEE Transactions on Rehabilitation Engineering*, 4(1):25–32, 1996.

[37] Felix Kok, James Charles, and Roberto Cipolla. Footnet: An efficient convolutional network for multiview 3d foot reconstruction. In *Proceedings of the Asian Conference on Computer Vision (ACCV)*, November 2020.

[38] S. Kolkur, D. Kalbande, P. Shimpi, C. Bapat, and J. Jatakia. Human skin detection using rgb, hsv and ycbcr color models. *Proceedings of the International Conference on Communication and Signal Processing 2016 (ICCASP 2016)*, 2017.

[39] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. Pifpaf: Composite fields for human pose estimation. *CoRR*, abs/1903.06593, 2019.

[40] Jiahui Lei, Srinath Sridhar, Paul Guerrero, Minhyuk Sung, Niloy J. Mitra, and Leonidas J. Guibas. Pix2surf: Learning parametric 3d surface models of objects from images. *CoRR*, abs/2008.07760, 2020.

[41] Vincent Leroy, Philippe Weinzaepfel, Romain Brégier, Hadrien Combaluzier, and Grégory Rogez. Smply benchmarking 3d human pose estimation in the wild. *CoRR*, abs/2012.02743, 2020.

[42] Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabás Póczos, and Ruslan Salakhutdinov. Point cloud GAN. *CoRR*, abs/1810.05795, 2018.

[43] Shice Liu, YU HU, Yiming Zeng, Qiankun Tang, Beibei Jin, Yinhe Han, and Xiaowei Li. See and think: Disentangling semantic scene completion. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[44] Nolan Lunscher. Deep learning 3d scans for footwear fit estimation from a single depth map. Master's thesis, University of Waterloo, Canada, 2017.

[45] Julieta Martinez, Rayat Hossain, Javier Romero, and James J. Little. A simple yet effective baseline for 3d human pose estimation. *CoRR*, abs/1705.03098, 2017.

[46] Oier Mees, Maxim Tatarchenko, Thomas Brox, and Wolfram Burgard. Self-supervised 3d shape and viewpoint estimation from single images for robotics. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6083–6089, 2019.

[47] Dushyant Mehta, Helge Rhodin, Dan Casas, Oleksandr Sotnychenko, Weipeng Xu, and Christian Theobalt. Monocular 3d human pose estimation using transfer learning and improved CNN supervision. *CoRR*, abs/1611.09813, 2016.

[48] Dushyant Mehta, Oleksandr Sotnychenko, Franziska Mueller, Weipeng Xu, Srinath Sridhar, Gerard Pons-Moll, and Christian Theobalt. Single-shot multi-person 3d body pose estimation from monocular RGB input. *CoRR*, abs/1712.03453, 2017.

[49] Lars M. Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. *CoRR*, abs/1812.03828, 2018.

[50] Gyeongsik Moon and Kyoung Mu Lee. I2l-meshnet: Image-to-lixel prediction network for accurate 3d human pose and mesh estimation from a single RGB image. *CoRR*, abs/2008.03713, 2020.

[51] Christopher J. Nester, Hannah L. Jarvis, Richard K. Jones, Peter D. Bowden, and Anmin Liu. Movement of the human foot in 100 pain free individuals aged 18–45: implications for understanding normal foot function. *Journal of Foot and Ankle Research*, 7(1):51, 2014.

[52] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. *CoRR*, abs/1603.06937, 2016.

[53] Katja Oberhofer, S. Lorenzetti, and Kumar Mithraratne. Host mesh fitting of a generic musculoskeletal model of the lower limbs to subject-specific body surface data: A validation study. *Applied Bionics and Biomechanics*, 2019:1–8, 02 2019.

[54] Michiel Oosterwaal. *The Glasgow-Maastricht Foot Model: Development, repeatability and sources of error of a 26 segment multi-body foot model*. PhD thesis, Maastricht University, United Kingdom, 2016.

[55] Orthopaedia. Anatomy of the foot and ankle. https://orthopaedia.com/page/Anatomy-of-the-Foot-Ankle, 2021. Accessed: 2021-08-11.

[56] Dr. Kevan Orvitz. Doctor of podiatric medicine, president & co-founder of MEGA-Comfort. Personal communications, 2021.

[57] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[58] Priyanka Patel, Chun-Hao P. Huang, Joachim Tesch, David T. Hoffmann, Shashank Tripathi, and Michael J. Black. Agora: Avatars in geography optimized for regression analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13468–13478, June 2021.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[60] Leonid Pishchulin, Stefanie Wuhrer, Thomas Helten, Christian Theobalt, and Bernt Schiele. Building statistical shape spaces for 3d human modeling. *CoRR*, abs/1503.05860, 2015.

[61] Albert Pumarola, Jordi Sanchez-Riera, Gary P. T. Choi, Alberto Sanfeliu, and Francesc Moreno-Noguer. 3dpeople: Modeling the geometry of dressed humans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[62] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[63] Haziq Razali, Taylor Mordan, and Alexandre Alahi. Pedestrian intention prediction: A convolutional bottom-up multi-task approach. *Transportation Research Part C: Emerging Technologies*, 130:103259, 2021.

[64] Danilo Jimenez Rezende, S. M. Ali Eslami, Shakir Mohamed, Peter W. Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. *CoRR*, abs/1607.00662, 2016.

[65] Kathleen M. Robinette and Hein A. M. Daanen. The caesar project: a 3-d surface anthropometry survey. *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No.PR00062)*, pages 380–386, 1999.

[66] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[67] Lukas Rustler, Bohumila Potocna, Michal Polic, Karla Stepanova, and Matej Hoffmann. Spatial calibration of whole-body artificial skin on a humanoid robot: comparing self-contact, 3d reconstruction, and cad-based calibration. In *Humanoid Robots (Humanoids), IEEE-RAS International Conference on*, 2021.

[68] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[69] Joshua Selvakumar.L and Padma Priya.R. Reconstruction of images into 3d models using cad techniques, 2014.

[70] Akash Sengupta, Ignas Budvytis, and Roberto Cipolla. Synthetic training for accurate 3d human pose and shape estimation in the wild. *CoRR*, abs/2009.10013, 2020.

[71] Leonid Sigal, Alexandru Balan, and Michael Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International Journal of Computer Vision*, 87:4–27, 03 2010.

[72] Gaurav Singhal. Transfer learning with resnet in pytorch. *PluralSight*, May 2020. https://www.pluralsight.com/guides/introduction-to-resnet. Accessed: 2021-09-14.

[73] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) oral presentation*, July 2017.

[74] Anugrah Srivastava, Tapas Badal, Apar Garg, Ankit Vidyarthi, and Rishav Singh. Recognizing human violent action using drone surveillance within real-time proximity. *Journal of Real-Time Image Processing*, 18(5):1851–1863, 10 2021.

[75] B. Starly, Z. Fang, W. Sun, A. Shokoufandeh, and W. Regli. Three-dimensional reconstruction for medical-cad modeling. *Computer-Aided Design and Applications*, 2(1-4):431–438, 2005.

[76] Liang Sun, Weigang Yao, Trevor T. Robinson, Cecil G. Armstrong, and Simão P. Marques. Automated mesh deformation for computer-aided design models that exhibit boundary topology changes. *AIAA Journal*, 58(9):4128–4141, 2020.

[77] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[78] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Single-view to multi-view: Reconstructing unseen views with a convolutional network. *CoRR*, abs/1511.06702, 2015.

[79] Maxim Tatarchenko, Stephan R. Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? *CoRR*, abs/1905.03678, 2019.

[80] Anh Thai, Stefan Stojanov, Vijay Upadhya, and James M. Rehg. 3d reconstruction of novel object shapes from single images. *CoRR*, abs/2006.07752, 2020.

[81] Unsplash. Photos for everyone. https://unsplash.com/. In accordance to Unsplash license. Accessed: 2021-10-15.

[82] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from synthetic humans. *CoRR*, abs/1701.01370, 2017.

[83] Timo von Marcard, Roberto Henschel, Michael J. Black, Bodo Rosenhahn, and Gerard Pons-Moll. Recovering accurate 3D human pose in the wild using IMUs and a moving camera. In *European Conference on Computer Vision (ECCV)*, volume Lecture Notes in Computer Science, vol 11214, pages 614–631. Springer, Cham, September 2018.

[84] Bastian Wandt, Marco Rudolph, Petrissa Zell, Helge Rhodin, and Bodo Rosenhahn. Canonpose: Self-supervised monocular 3d human pose estimation in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13294–13304, June 2021.

[85] Jiahang Wang, Sheng Jin, Wentao Liu, Weizhong Liu, Chen Qian, and Ping Luo. When human pose estimation meets robustness: Adversarial algorithms and benchmarks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11855–11864, June 2021.

[86] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829, 2018.

[87] Good Ware. Image from flaticon.com. https://www.flaticon.com/authors/good-ware. In accordance to Flaticon attribution license. Accessed: 2021-08-26.

[88] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, William T. Freeman, and Joshua B. Tenenbaum. Marrnet: 3d shape reconstruction via 2.5d sketches. *CoRR*, abs/1711.03129, 2017.

[89] Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. *CoRR*, abs/1406.5670, 2014.

[90] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. *CoRR*, abs/1612.00814, 2016.

[91] Bo Yang, Hongkai Wen, Sen Wang, Ronald Clark, Andrew Markham, and Niki Trigoni. 3d object reconstruction from a single depth view with adversarial learning. *CoRR*, abs/1708.07969, 2017.

[92] Jun Yang, Dong Li, and Steven L. Waslander. Probabilistic multi-view fusion of active stereo depth maps for robotic bin-picking. *IEEE Robotics and Automation Letters*, 6(3):4472–4479, 2021.

[93] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Interpretable unsupervised learning on 3d point clouds. *CoRR*, abs/1712.07262, 2017.

[94] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21:2140, 03 2021.

[95] Zhixuan Yu, Jae Shin Yoon, In Kyu Lee, Prashanth Venkatesh, Jaesik Park, Jihun Yu, and Hyun Soo Park. Humbi: A large multiview dataset of human body expressions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2987–2997, 2020. Funding Information: This work was partially supported by National Science Foundation (No.1846031 and 1919965), National Research Foundation of Korea, and Ministry of Science and ICT of Korea (No. 2020R1C1C1015260). Publisher Copyright: © 2020 IEEE.; 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020 ; Conference date: 14-06-2020 Through 19-06-2020.

[96] Ye Yuan, Shih-En Wei, Tomas Simon, Kris Kitani, and Jason Saragih. Simpoe: Simulated character control for 3d human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7159–7169, June 2021.

[97] Andrei Zanfir, Eduard Gabriel Bazavan, Mihai Zanfir, William T. Freeman, Rahul Sukthankar, and Cristian Sminchisescu. Neural descent for visual 3d human pose and shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14484–14493, June 2021.

[98] Mihai Zanfir, Elisabeta Oneata, Alin-Ionut Popa, Andrei Zanfir, and Cristian Sminchisescu. Human synthesis and scene compositing. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[99] Jianfeng Zhang, Dongdong Yu, Jun Hao Liew, Xuecheng Nie, and Jiashi Feng. Body meshes as points. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 546–556, June 2021.

[100] Sen Zhang, Zhigang Li, Hui Zhang, and Junhai Yong. Multi-resolution mesh fitting by b-spline surfaces for reverse engineering. In *2011 12th International Conference on Computer-Aided Design and Computer Graphics*, pages 251–257, 2011.

[101] Tianshu Zhang, Buzhen Huang, and Yangang Wang. Object-occluded human shape and pose estimation from a single color image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[102] Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Josh Tenenbaum, Bill Freeman, and Jiajun Wu. Learning to reconstruct shapes from unseen classes. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[103] Zerong Zheng, Tao Yu, Yebin Liu, and Qionghai Dai. Pamir: Parametric model-conditioned implicit representation for image-based human reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021.

[104] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[105] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.

[106] H. Zhuang and Z.S. Roth. Camera-aided robot calibration (1st ed.). *CRC Press.*, 1996.