Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

The Impact of Synthetic Data On The Reliability of Hand Tracking Systems

Daniel Desmond Dennis

[redacted]

April 2020

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer Engineering)

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university

Signed: [redacted] .   Date: 30th April 2020

Abstract

This dissertation investigates the merits of using synthetic data for training a hand tracking model to address the problem of the lack of training data for such systems. This is achieved by looking at two different synthetic data generation strategies, and training an existing hand tracking model to see how it performs on those generated synthetic datasets. An existing real dataset, the MSRA dataset is used as a control. The MANO model is used as the basis for generating synthetic data, and V2V-Posenet is used as the existing hand tracking model. The first strategy generates the data based on random parameters for MANO, this is referred to as the *Random MANO Dataset*. The second strategy generates the data based on parameters determined from the groundtruth of the real dataset to recreate that real dataset using inverse kinematics, this is referred to as the *IK MANO Dataset*.

In recent years, the accuracy of hand tracking systems has improved with the use of *convolutional neural networks* (CNNs). However, these systems need comprehensive training data, and such a dataset does not exist currently. This need is underlined by the fact that current hand tracking systems do not generalise well for a given training dataset, and performs poorly for hand poses that are not covered by that training dataset. The reasons for this lack of data are twofold. Firstly, annotating these datasets is difficult. Secondly, the human hand is a uniquely complex object, and a comprehensive dataset needs to cover a diverse range of; camera perspectives, hand poses, hand shapes, and interactions with objects, so the amount of data that needs to be collected to cover these ranges is also unknown. Synthetic data offers a way around this issue, since the annotation is theoretically perfect, and the capture of that data can be automated.

This dissertation shows that generating synthetic based on the *Random MANO Dataset* strategy shows promise, but more testing is required by generating more synthetic images. State-of-the-art results on this test dataset is demonstrated at an 8.90mm average per-joint mean squared error, but poor performance when comparing with the real test dataset. It also shows that trying to recreate a real dataset with the *IK MANO Dataset* is challenging, with efforts to achieve this showing poor performance at 77.44mm. The key contributions of this dissertation are (1) performing a side-by-side comparison between a synthetic and real hand tracking dataset using the state-of-the-art synthetic hand data generation method, MANO, (2) giving the basis for a new synthetic dataset based on the MANO model with the *Random MANO Dataset*, and (3) providing a new framework for generating a synthetic dataset by using a real dataset as a basis with the *IK MANO Dataset*.

Acknowledgements

Thank you to my supervisor, Dr Gerard Lacey who guided me through this project and provided invaluable advice. Thank you to Tejo Chalasani who also provided helpful advice on hand tracking, and was always available for help if I had a problem. Thank you to Yalong Li (*dragonbook* on GitHub) whose PyTorch implementation of the V2V-PoseNet network was central to the experimental setup of this project. Thank you to Xinghao Chen (*xinghaochen* on GitHub) who hosts a comprehensive list of papers and other works related to hand tracking. Thank you to my father Gus, as well as friends Alec Barber and Davy Nash, who exposed the most egregious errors and omissions in assisting the editing process of this document. Thank you to Dr Mike Brady for providing the LaTeXtemplate for this document. Finally, thank you to the many people on Stack Exchange websites (mostly Stack Overflow and TeX) as well as other assorted websites who have already asked and answered many of the questions I had in relation to this project and beyond.

Contents

List of Figures

List of Tables

Nomenclature

CNN	Convolutional Neural Network
FCN	Fully Convolutional Neural Network
LBS	Linear Blend Skinning
MSE	Mean Square Error
ROI	Region Of Interest

$\mathbf{D}^t$	Training dataset, where $\mathbf{D}^t_k \in \boldsymbol{Z}$
$\mathbf{D}^v$	Test dataset, where $\mathbf{D}^v_k \in \boldsymbol{Z}$
E	Performance metric
$\boldsymbol{v}$	3D centre of hand in depthmap image $\boldsymbol{Z}$, where $\boldsymbol{v} \in \mathbb{R}^3$
$\mathbf{Y}^i$	Groundtruth hand joints, where $\mathbf{Y}^i_n \equiv \mathbf{\Phi}$, $\mathbf{Y}^i_{n,m} \equiv \mathbf{\Phi}_m$
$\mathbf{Y}^o$	Predicted hand joints, where $\mathbf{Y}^o_n \equiv \mathbf{\Phi}$, $\mathbf{Y}^o_{n,m} \equiv \mathbf{\Phi}_m$
$\boldsymbol{Z}$	Depthmap image, where $\boldsymbol{Z}_{m,n} \in \mathbb{N}$
$\boldsymbol{\beta}$	MANO Hand shape parameter, where $\boldsymbol{\beta} \in \mathbb{R}^{10}$
$\boldsymbol{\theta}$	MANO hand pose parameter, where $\boldsymbol{\theta} \in \mathbb{R}^{45}$
$\mathbf{\Phi}$	Hand joints, where $\mathbf{\Phi} = \{\mathbf{\Phi}_1, \mathbf{\Phi}_2, ..., \mathbf{\Phi}_N\}, \mathbf{\Phi}_n \in \mathbb{R}^3$

a	A scalar value
$\boldsymbol{a}$	A vector value
$\boldsymbol{A}$	A matrix value
$\mathbf{A}$	A tensor value

Articulation	The particular position of the fingers of a hand with respect to each other.
Annotation	The marked groundtruth for a depthmap $\boldsymbol{Z}$.
Groundtruth	True value for N keypoints.
Joint	A type of keypoint.
Keypoint	Known reference point in a hand in 3D cartesian space.
Point Cloud	A series of 3D points which together describe the shape of a 3D surface.
Pose	An alternative name for articulation.
Prediction	Prediction of the keypoints from a hand tracking model.

All notation follows the convention set by [3].

Conventions

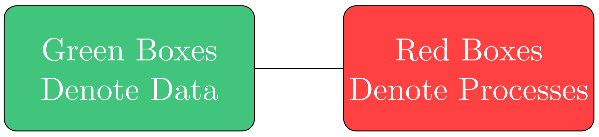

Figure 0.1: Box diagram convention. All boxes flow from left to right. Green boxes denote data, and red boxes denote a process on that data.

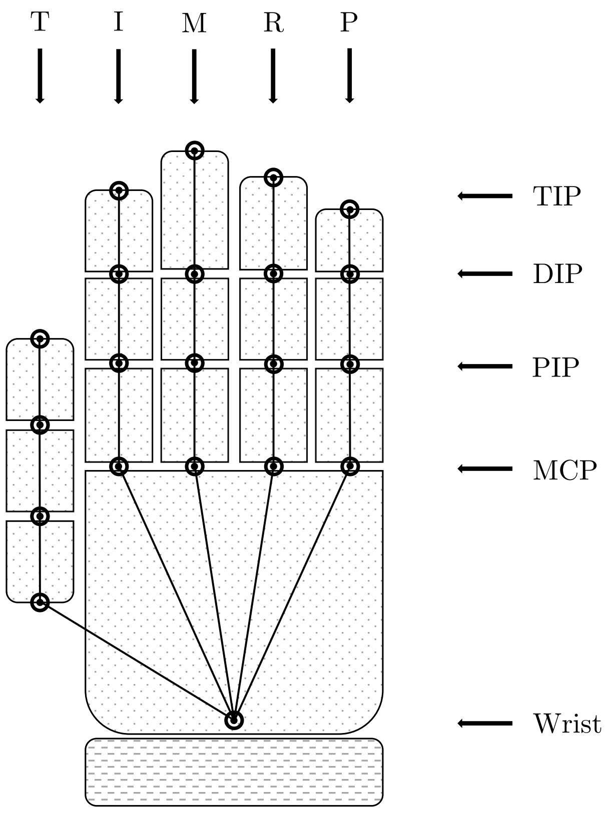

Figure 0.2: Schematic of hand and keypoints. The goal of a hand tracking system is to predict the 3D locations of these keypoints. A keypoint name is the combination of the finger name and joint type, for example, the MCP joint of the Index finger is called the IMCP keypoint. Each of these keypoint names has an equivalent index in $\mathbf{\Phi}$. From left to right, the finger names used are: thumb, index, middle, ring, and pinky.

1 Introduction

The focus of this dissertation is on hand tracking, and specifically the merits of using synthetic data for training hand tracking systems. A hand tracking system is one that can automatically estimate the 3D locations of the joints in the hand, given an image of that hand. The overall purpose of a hand tracking system is to enable human-computer interaction with a hand as a medium (other forms of human-computer interaction include the use of keyboards, mice, and punchcards). Figure 1.1 illustrates an example of hand tracking for interaction with a Virtual Reality headset.

Recent hand tracking systems have become more accurate and robust, however these systems suffer under many scenarios such as from the ego-centric perspective (where the camera perspective of the hand is captured from a head-mounted camera, or similar positions). Hand tracking is notoriously difficult because the hand has many degrees of freedom (the different ways in which fingers can position themselves), and it can self-occlude (where one part of the hand blocks the view of another part of the hand in an image). Hand tracking systems require a lot of data to be able to perform well in different scenarios, but acquiring that data is difficult. While the process of capturing hand data is trivial, the challenge lies in annotating that data. For the type of hand tracking system mentioned above, annotating the data required to train a hand tracking system involves recording the 3D positions of a set amount of keypoints (known points in the hand that the hand tracking system is trying to predict the location of in 3D space, such as a knuckle). Manually labelling the data at scale is impossible in practical terms, particularly when the hand self-occludes, and recent attempts at manual annotation only offer limited resolution and size, [4] for example only managed 11703 images with 2D annotation, and they did not annotate self-occluded joint. Recent projects such as BigHand2.2M[1] and ICVL[5] that automatically annotate datasets have collected millions of annotated images of hands, but they rely on a small amount of subjects, and there is no way of evaluating the accuracy of that annotation. Since the annotation is theoretically perfect, synthetic data offers a

Figure 1.1: An example of the use of Virtual Reality using only hands to interface with the computer[1].

way to address the issue of accuracy and scale. Past attempts at generating synthetic data however lack the realism that real data can provide[6, 7]. As mentioned by [8], even with such large datasets, current systems still struggle to interpolate between poses in large datasets. More recently, the MANO model has attempted to address the lack of realism in synthetic hand data [9]. Building on the previous work in full body rendering for computer graphics, the authors of MANO learn a synthetic hand data generation model based on real-world subjects, including hands interacting with objects.

1.1 Motivation

In recent years and decades, as computer technology has improved, new possibilities for human-computer interaction have emerged. This ranges from punch cards, to keyboards, to mice, and touch screens. The developments that these innovations have led to need not be enumerated here. Progress has been made in the development of new technologies and ways for human-computer interaction, which is being lead by the emergence

[1]CC-BY-2.0 License (`https://creativecommons.org/licenses/by/2.0/deed.en`)© *John F. Williams* (title: 150914-N-PO203-142), obtained from `https://www.flickr.com/photos/usnavyresearch/23496662185/` (accessed 29th April 2020)

of new technologies such as Augmented Reality (AR) and Virtual Reality (VR). While both technologies have been around for some time, one problem with them is how a user interacts with them. The use of human hands offers the potential to improve the human-computer interaction with these technologies, but hand tracking notoriously difficult problem. More recently, as of writing the world is under lockdown due to Covid-19. Computer mice and keyboards, due to the need to touch them are inevitable vectors of disease, and interacting with a computer using hand gestures alone offers a less risky way of using them.

The specific motivation about using synthetic data to train a hand tracking system in this project is largely influenced by the *5th International Workshop on Observing And Understanding Hands in Action* (*Hands 2019*)[2]. The goals of this challenge are discussed more formally in [8]. The *Hands 2019* challenge provides three different tasks for: depth-based hand pose estimation, depth-based hand pose estimation with hands interacting with objects, and RGB-based hand pose estimation with hands interacting with objects. In all three of these tasks, participants are encouraged to use the MANO model to train with synthetic data.

1.2 Objectives

The objective of this dissertation is to evaluate the current state of generating synthetic data in 2020 towards improving the performance of hand tracking systems. This is achieved by comparing synthetic data generated with real data: the MSRA dataset [10]. The comparison is performed by training an existing hand tracking system, the V2V-Posenet model [2], with both kinds of data, and cross comparing the resultant models using a performance metric. The performance metric is Mean Squared Error (MSE), and is discussed in Section 4.3.

1.3 Overview

Two synthetic data generation strategies are investigated. The first strategy generates the data based on random parameters for MANO, this is referred to as the *Random MANO Dataset*. The second strategy generates the data based on parameters determined from

[2]`https://sites.google.com/view/hands2019/home` (accessed 29th April 2020)

the groundtruth of the real dataset to recreate that real dataset using inverse kinematics, this is referred to as the *IK MANO Dataset*.

The key contributions of this dissertation are

1. Generating a new synthetic dataset using the MANO model.
2. Performing a side-by-side comparison between a synthetic and real hand tracking dataset using the state-of-the-art synthetic hand data generation method, MANO.
3. Providing a new framework for generating a synthetic dataset by using a real dataset as a basis.

The rest of this dissertation is divided as follows. Section 2 discusses and evaluates the previous work around hand tracking systems and datasets. Section 3 discusses the underlying theory of what synthetic data is, the implications of training a hand tracking system with it, and the strategies used to generate such data. Section 4 discusses the practical details of the experimental setup used for this dissertation. Section 5 presents the results of the experiments to compare the synthetic and real dataset. Section 6 concludes this dissertation.

2 Literature Review

This section will outline the state-of-the-art in hand tracking. Given the nature of this project, a particular focus is put on works that produce datasets for hand tracking, as well as systems that use depth sensors. As well as this, other works investigated include the related problem of full body pose estimation, as well as hand tracking with RGB sensors. The aim in this section is to establish the general idea of what a modern hand tracking system entails, how these systems adapt to the real world, and any relevant gaps and problems in research.

2.1 Previous Work

2.1.1 Data

A common issue highlighted in literature is a lack of data [8], leading many to produce new datasets for training hand tracking systems. There are broadly two kinds of hand tracking datasets. Firstly, depth-based datasets consist of depth images. Such images capture the depth information of the scene. Secondly, RGB-based datasets consist of colour images of the hand. Some systems use both depth and RGB images for hand tracking. Different datasets also employ different labelling regimens. Most datasets aim to annotate the 3D locations of keypoints. These keypoints consist of certain points of interest in the hand, typically the knuckles, wrist, and finger joints. Some datasets only annotate the 2D keypoint locations which is easier to produce, while most datasets discussed here annotate the 3D keypoint locations, as this provides a richer feature space to extract information about the hand, although it is more difficult to estimate 3D keypoints as opposed to 2D keypoints. Another labelling strategy is to segment the images into different parts, such as the region that contains particular fingers. The other distinction between hand tracking datasets is whether they are real or synthetic datasets.

The *NYU* dataset [11], which is based on depth images is one of the first modern hand-tracking datasets. The authors describe a comprehensive pipeline for acquiring data, and then training a *convolutional neural network* (CNN) with that data, to produce a system that can infer the continuous pose shape of a hand. In the dataset creation pipeline, the hand is first segmented using a *randomised decision forest*. Then using an initial approximation of the pose parameters, a *particle swarm optimisation* algorithm is used whereby a candidate pose is used to generate a synthetic image using an *linear blend skinning* (LBS) model. The captured image is compared against the generated image using an objective function and the candidate pose is updated using a loss function. The *Nelder–Mead* method is used to refine the pose estimates. The initial guess of the first frame is manually annotated, and is passed in a chain to each subsequent frame to be labelled. Their dataset is from two subjects, with 72,000 training, and 8,000 test images.

Two new datasets are introduced in [12]. Their first dataset, *FingerPaint* consists depth of images of hands whose groundtruth consist of pixel-wise classifications of the fingers of the hand as well as the palm, describing what region of the hand it belongs to, such as the palm, index finger, etc. The ground truth does not describe the location of keypoints. The annotations are generated with the help of an RGB sensor also recording the hand sequence. The hands are 'painted' and segmented with a colour segmentation algorithm to generate the labels, with any errors corrected manually. This dataset is not intended for training, instead just for evaluating the performance of a hand tracking system. The other dataset that they generate is called *Synthetic*, which consists of 1000 synthetic depth images rendered using a 3D mesh model. Unlike *FingerPaint*, these images are not in a sequence and are in random articulations. Exact details are not available, but it is likely that the groundtruth consists of 3D keypoint positions.

Expanding on the labelling strategy of [11], [10] introduce the *MSRA* dataset of 76,500 depth images from 9 subjects for evaluating a hand tracking system. Each of the subjects recorded were asked to produce 17 different poses which the authors say are mostly drawn from American Sign Language. The ground truth is generated from another hand tracking system and manually corrected for any errors.

[5] introduce a dataset consisting of 180,000 depth images with 3D keypoint annotation. In what they call the *ICVL* dataset, it contains hand poses from 10 different subjects.

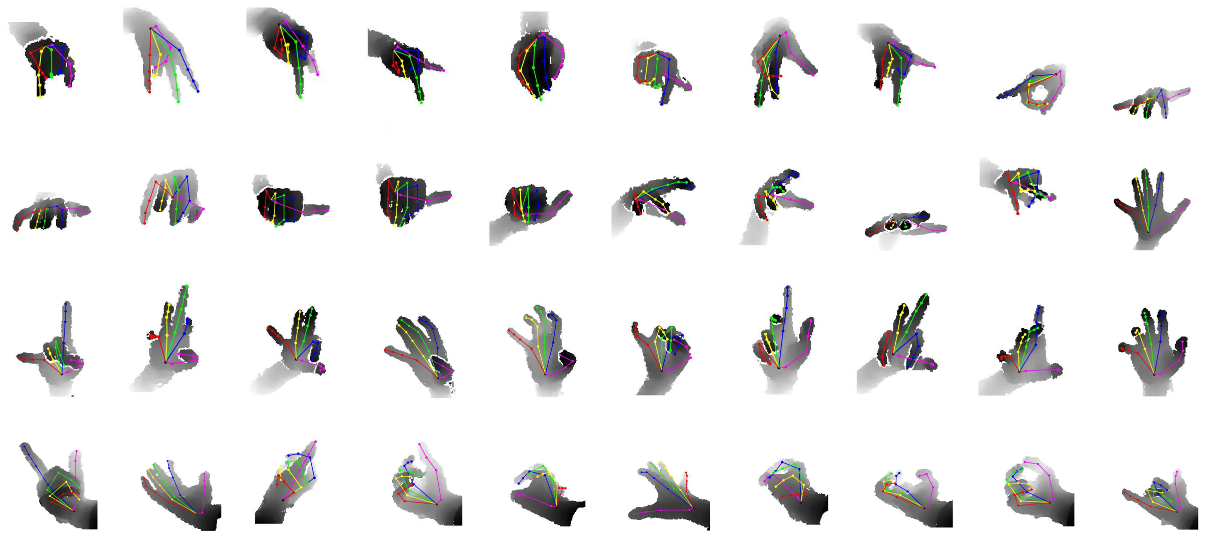

Figure 2.1: Sample images from the BigHand2.2M dataset with the annotation superimposed. Image taken from the original paper [1].

Each of the subjects are presented with the same 26 poses to recreate with their hands. They use Intel's *Creative Interactive Gesture Camera* to capture the data which they say provides far better depth image quality than previous technologies. The annotation is generated by fitting a point cloud (a series of 3D points which together describe the shape of a 3D surface) version of the depth image to an articulated synthetic hand model. The authors highlight that this annotation system often fails at difficult poses, which means that their dataset largely contains easy frontal poses (where the camera is positioned in front of the subject). To account for this limitation, they compare their dataset with the *Synthetic* dataset introduced by [12].

BigHand2.2M [1] is the first large scale hand tracking dataset, containing 2.2 million annotated depth images of hands for training and evaluating a hand tracking model. They use Intel's *RealSense SR300*.Sample images can be seen in Figure 2.1. They highlight the lack of available datasets for hand tracking, and hypothesise the reason is the lack of scalable methods for annotation. They claim existing automatic methods are not accurate enough, or that they compromise the dataset (such as the use of gloves to aid annotation). They say that manual annotation is too slow to make a dataset sufficiently large. They introduce a new capture method for annotation where the hand is fitted with magnets on each fingertip as well as the palm. With a depth camera capturing the data, the annotations are generated using magnetic sensors. They produce 21 3D keypoints using *inverse kinematics* of the hand to generate the subsequent annotations. Their dataset is from ten subjects.

[6] introduce a new RGB and depth synthetic dataset, *SynthHands*. They define an egocentric viewpoint as the camera perspective of the hands as is seen from the head or chest. The reasoning behind making such a dataset is that it is challenging to capture the real world annotations of hands, and that existing datasets lack hand-object interactions and realistic hand poses, nor do they show real world noise and backgrounds. They highlight that annotating an egocentric dataset is particularly challenging due to the increased likelihood of self occlusion, which is where the hand blocks part of itself from the camera view. The gestures rendered in the synthetic dataset are generated from real gestures captured from a human hand, with the camera retargeted to an egocentric viewpoint. As well as this, the synthetic hands have artificial interactions with objects, and the background is composited with different scenarios including desktops, offices, corridors, and kitchens.

[7] propose a new dataset, *SynHand5M*, which contains 5 million synthetically generated depth images of hands in different poses, and camera viewpoints. They claim this approach to be superior to [1] and [9], since they are both based on the data of a small amount of subjects, the nature of their synthetic data generation algorithm means they can vary the shape of a hand, which cannot be done with real data. They also claim that since the data is synthetically generated, the groundtruth is perfect. Their goal in making this dataset is to cover most, if not all possible articulations, shapes, and camera perspectives. To this end, in order to ensure that the hand shapes are realistic, they use limits defined by an anthropometric database. They also set limits to the articulations of the hand. This is because it is easy to deform the synthetic hand into shapes that a real world hand would not be able to achieve, which could lead the hand tracking system that trains on this dataset to learn unrealistic hand articulations. They admit that their specific implementation is not fully realistic, but claim that this is sufficient as a pretraining step for the CNN that they use for their hand-tracking system.

[4] highlight the lack of data available for training RGB based systems. They show that existing RGB based datasets are synthetic, or captured in a lab setting. Unlike depth images, RGB images have to have a background to simulate real life use, and the background has to vary too or else the model may just learn that background and it will not be able to segment other types of backgrounds. Their new dataset is called *OneHand10K* which consists of 11703 RGB images with varying backgrounds, and annotated for 21 2D keypoints. The groundtruth is manually annoted by at least three different people marking, and keypoints that are not visible in the dataset are not annotated.

2.1.2 Rendering Methods

A new full-body statistical model is introduced in [9] called MANO. Built on previous full-body statistical models, MANO is compatible with mainstream computer graphics systems and allows synthetic poses using a set of shape and articulation parameters to model different body shapes and poses seen in real life respectively. MANO places a special emphasis on hands, the authors highlight that recent advances in full-body models have often neglected hands, given their complex articulated nature with respect to the rest of the human body. The model is learned by capturing the full body and hands separately. The hand model data is captured from 31 subjects performing 51 different actions including hand-object interaction. The combined model uses LBS, and the authors say that certain corrective blend shapes are used to mitigate the drawbacks of LBS. The model takes two parameters $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ which outputs 778 vertices describing the shape of a hand in 3D space. These vertices can then be put into a standard graphics pipeline for rendering. They use this mesh model to improve the hand rendering performance of the full body model SMPL [13].

2.1.3 Hand Tracking Systems

A hand tracking system is one that in general, takes a depth, RGB, or both images as an input and outputs a set of 2D or 3D coordinates showing the location of the hand's keypoints. Modern (within the last 3-4 years) systems typically employ some form of CNN architecture, however the basic architectures that have seen success in other tasks such as ordinary image classification are typically not enough and a more sophisticated model is required.

2D CNN Approaches

[6] approach the problem of hand pose estimation using a combination of RGB and depth cameras from an egocentric perspective as mentioned in the dataset discussion. They combine the RGB and depth images into a coloured depth image (RGBD) first, and from that crop the image to the *region of interest* (ROI, the part of the image that contains the hand, either a bounding box, or semantic segmention). This is done by passing the image into a CNN which outputs a heatmap. The centre of this heatmap is

supposed to be the MCP joint of the middle finger (see Figure 0.2 for an illustration). In order to ensure that the location of the centre of the heatmap is temporally consistent, if the confidence of this centre is less than 0.1, and it is more than 30 pixels away from the centre of the previous image, they update the heatmap centre using the sum of the previous frame and the estimate of the current location based on the motion of previous frames that were marked as confident. In a special function, it locates the centre of this heatmap, and using the depth information, it removes pixels that are far away from a 3D Eucliadian perspective. From there, this cropped RGBD image, it is passed into a CNN to regreess the 3D keypoints, as well as a 2D heatmap of these keypoints. The regressed 3D keypoint coordinates are refined using a kinematic hand model, and further refined with the help of the 2D heatmap.

[7] employs a two-stage CNN architecture where given a depth image, they aim to estimate the 3D keypoint locations of the hand, as well as a vertex mesh representing a reconstruction of the hand's surface. Within this CNN, there are three parallel steps for estimating: the hand articulation $\boldsymbol{\theta}$, hand shape $\boldsymbol{\beta}$ (these parameters are not to be confused with similar notation of MANO), and scale α. Inside the same CNN, α, $\boldsymbol{\beta}$, and $\boldsymbol{\theta}$ are then passed into a non-linear differentiable layer (which means that this layer can be trained with backpropagation together with the previous layers) that they call a *Hand Pose and Shape Layer*. This layer consists of stages to construct a hand representation using the shape parameters α, $\boldsymbol{\beta}$, and they use the $\boldsymbol{\theta}$ then to reconstruct the hand pose using Linear Blend Skinning.

[4] approach the hand tracking problem in terms of discovering a set of 2D keypoints in an RGB image. They highlight the advances made in hand tracking using depth sensors and the comparative slowness in progress for RGB systems. Many existing systems rely on strict lighting and backgrounds. Their model is divided into two parts, both are fully-convolutional networks, where the first segments a silhouette (a pixel-wise segmentation of the image into foreground that contains the hand, and background that does not contain the hand), and the second regresses the 2D keypoints. They believe that this approach can be a step towards RGB-based systems for 3D keypoint estimation.

[14] address hand tracking in terms of estimating a large set of vertices that describe the surface of a human hand from a depth image. The authors argue that estimating a small set of keypoints is underconstrained, and that estimating the vertices of a hand has more useful applications. To this end, they propose a *fully convolutional network* (FCN)

architecture that estimates these vertices. There are two consecutive hour-glass FCNs. The first FCN has two heads (outputs), one with a generic feature map, and one with estimates 2D mesh coordinates of the image. In a process the authors call *extension*, the feature map and 2D mesh estimation are combined and passed into another FCN to give an initial estimate of the 3D vertices. This initial estimate is refined using a kinematic model of the hand.

[15] introduce a novel feature boosting algorithm for a CNN network for inferring the 3D keypoints of an RGB image of a hand or human body. They introduce a new *long short-term dependence-aware* (LSTD) module that understands the relationship between different keypoints to boost the feature outputs of a CNN network. For example, the keypoint relationship between two different keypoints in the same finger are highly correlated, but less so with keypoints in other fingers of the same hand. They highlight that many existing pose estimation methods do not factor the relationship between keypoints, or use keypoint relationships to refine the estimate as a post processing stage. Their network can be stacked into multiple blocks to further refine the keypoint estimate. To account for the unreliability of the features generated by a CNN due to occlusions and textures in the RGB input image, they also introduce a *context consistency gate* to act as a gating mechanism to the LSTD module which analyses the input features in terms of their context with respect to their neighbouring keypoints, and adjusts any irregularities to improve the estimation of the keypoint locations overall. They claim that their results achieve state-of-the-art results on human body estimation datasets.

3D CNN Approaches

[2, 16] both propose similar systems where depth image representation of a hand, is inputted to a 3D CNN to predict the 3D keypoints of the hand. The depth image, a 2D pixel array is converted into a 3D voxel (a voxel is the 3D equivalent to 2D pixels) representation and then passed into the CNN for inference. The authors cite the fact that a depth image really conveys 3D information, and they hypothesise that passing in a 2D depth image into a 2D CNN discards the 3D data, leading to reduced performance. [2] discuss this hypothesis by running four experiments; a depth image is used to infer 3D keypoints, a depth image is used to infer voxelised 3D keypoints, a voxelised depth image is used to infer 3D keypoints, and a voxelised depth image is used to infer voxelised 3D keypoints. Their experiments show that a voxelised depth image used to infer vox-

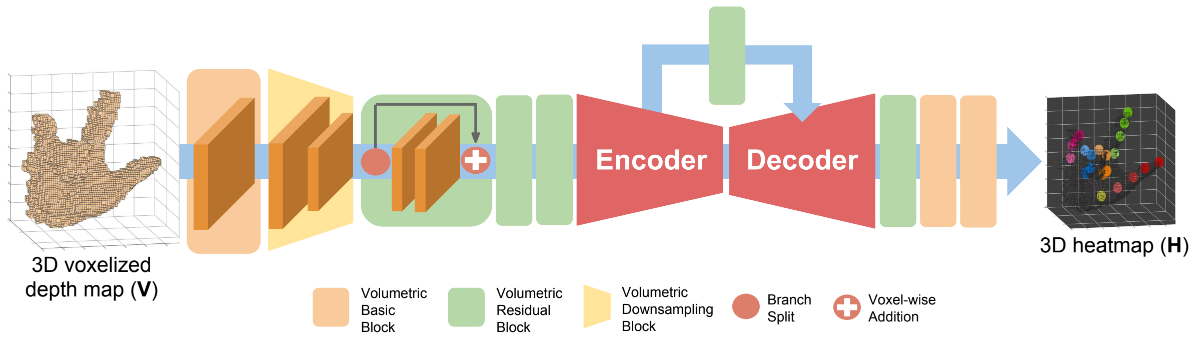

Figure 2.2: An illustration of the V2V-Posenet model. Image taken from the original paper [2].

elised 3D keypoints delivers superior performance over the other methods, they call their new architecture *V2V-PoseNet*. [16] investigates this hypothesis by running these four experiments. Firstly a depth image is passed into a CNN to estimate the heatmap for each keypoint location. Second, the voxel representation of the hand is used to generate multiple depth map images from different perspectives and passed into a 2D CNN to estimate a series of heatmaps for keypoints for each input. Third, the single depth image is passed into a CNN to directly estimate the 3D keypoints. Fourth, the voxelised image is passed into a 3D CNN to estimate the 3D keypoint locatioins. [17] takes *V2V-PoseNet* one step further by also trying to estimate hand shape in addition to pose using more 3D CNN architectures in what they call *HandVoxNet*.

Generative Approaches

Generative Adversarial Networks (GANs) are a recent neural network architecture where two components are trained in tandem, a generator to generate synthetic data, and a discriminator to tell whether it is real or not [18]. The idea is that both components engage in a race to outperform the other, and ultimately we get a good system for generating synthetic data, and a good system for telling whether that is real or fake by both networks working against each other.

[19] address hand tracking for RGB images by adopting this GAN approach. In this paper, the authors use the generator to estimate the hand pose in a two stage process where an initial estimate is generated, and further refined using a parametric hand model. The discriminator acts effectively as a loss function, distinguishing between the predicted keypoints and groundtruth.

[20] highlight how many recent works in hand pose estimation rely on a 3D hand model as part of the estimation process. They hypothesise that this is not an ideal approach given that these hand models are simplified and do not take the actual anatomical characteristics of the hand into consideration. Given a depth image as an input, they aim to estimate a set of keypoints describing the hand articulation, which is achieved as follows. After obtaining a box-shaped ROI of the hand, this is passed into a CNN to give an initial estimate of the 3D coordinates, which is passed into another CNN model that generates a synthetic image, in a similar approach to the generator in [19]. The real and synthetic depth images are passed into another CNN that updates the estimate of the keypoints, which is passed back into the generator CNN again in an iterative process.

[21] propose a simple GAN for RGB-based pose estimation. They highlight the recent advances made in hand pose estimation using depth images, but they believe RGB-based methods to be ultimately superior because RGB sensors are cheaper and more readily available. Combining these two ideas, they aim to train a system that generates a depth image from an input RGB image in one module. In the other module, the 2D keypoints are directly estimated on the RGB image, then the 3D keypoints are regressed from that. This estimate is then refined using the generated depth image. The generative component needs to be trained with paired RGB and depth images, but it can then be trained on RGB images alone after that and they claim that the generative component can improve the estimate versus using RGB alone such as [4].

Cascaded Regressors

[10] highlight recent successes in full body classification algorithms that classify the body into different regions based on what body part that they belong to, but that this approach does not work well for hands. They believe that this is because full body images are easy to classify, they tend to be all frontal-based. Hand images in contrast tend to have more varied camera angles and the hand is capable of deforming in far more diverse orientations. They therefore believe that regressing the pose of the hand, that is estimating the locations of 3D keypoints describing the locations of certain parts of a hand in 3D space is a more principled approach to the problem. They believe however that current regression techniques such as CNNs do not have the capacity to learn all of the possible variations in human hand poses. To address this hypothesis, they believe a cascaded regression approach for hand tracking is superior. Cascaded regression works

by using a pipeline of weak regressors. They extend previous work for 2D cascaded hand pose regressors into 3D.

[22] address the problem that modern hand tracking systems are typically ‘black box’ systems. That is, that the hand is passed into some model and it outputs the parameters without us knowing what is actually going on inside the system beyond the basic mathematics. This is a general problem within machine learning. They specifically address this issue for *analysis by synthesis* depth hand tracking systems such as [12]. In a process that they call *hierarchical sampling optimisation*, it uses knowledge of the hands structure to estimate the keypoints of the hands in a hierarchal way from the wrist towards the fingertips. Then, like [12], this is used to render a synthetic version of the hand and compared against the original depth image using a scoring function.

[23] propose a hierarchal CNN for estimating the keypoints of a hand from a depth image. They highlight the progress made in accurate hand tracking in terms of reduced error, but also highlight the fact that some of the more accurate systems have a high computational overhead, particularly V2V-Posenet [2]. Their system passes a depth image through an encoder, then the keypoints for the thumb, palm, and each finger are separately estimated in discrete CNN blocks to output 3D keypoints. Their system is more accurate than all of the systems that they survey except for [2, 24], however it is still faster than them.

Mesh Recovery

[12] introduce an *analysis by synthesis* approach for hand tracking. *Analysis by synthesis* is an approach to computer vision that uses *Bayesian inference*. This idea posits traditional machine learning where a model learns from data as ‘bottom up’ processing, and they argue that we can gain more accurate insights about a problem by also considering a ‘top down’ approach [25]. The idea is that given an input image, an attempt is made to reverse the process of how that image was generated. In a simple example of classifying a pattern of text, this method says that we should separate this problem into components such as the probability of the font, noise, overlap, obstruction etc to get towards the core problem, that is: what the actual text is. In terms of hand tracking, this approach works by taking a depth image as an input, first segmenting the ROI. Then, the global rotation and subsequently local articulation are estimated to generate a series of candidate poses which are rendered into synthetic images. This synthetic image can then be compared against the original using a simple pixel-wise MSE function to choose the best

one. This initial estimate is then refined using *particle swarm optimisation* combined with information from previous frames in the same video sequence.

[26] uses the MANO model [9] at the centre of their hand tracking system. Their system aims to predict 778 3D vertices describing a hand shape from an RGB input, in a similar 'top down' approach to hand tracking as [12]. Instead of trying to predict this directly, they have a three-stage pipeline employing CNN architecture to achieve this. The first stage consists of two parts; one CNN generates a 2048-dimensional feature vector of the foreground of the image containing the hand, and another CNN estimates the 21 2D keypoints of the hand. A second CNN takes these features and keypoints as an input and outputs a 63-dimensional vector. The first 45 parameters are the articulation parameters for the MANO model $\boldsymbol{\theta}$, the next 10 are the MANO shape parameters $\boldsymbol{\beta}$, the next four are the camera rotation in quaternion space, one for scale, and the final three are for the camera translation. As previously described, the MANO model outputs 778 vertices given the articulation and shape parameters. The 778 vertex output can then be rotated, scaled, and translated as appropriate using the rotation, scale, and translation parameters. These 778 vertices are then passed to a regressor to obtain 21 keypoints in 3D space.

2.1.4 Related Tracking Systems

[27] looks at the problem of full-body tracking. The authors specifically address the problem of the lack of training data for full-body tracking for RGB-based systems. Using the full-body version of MANO - (SMPL+H) [9], they produce two datasets, one containing fully-synthetic data, and one that is combined with a real dataset as an effective augmentation strategy. A large emphasis in the paper is trying to generate photo-realistic data, given that this is an RGB-based system. The generation of photorealistic data is not a problem limited to synthetic and is a general problem in computer graphics. One of the important insights about synthetic data from the paper is that some synthetically-generated images can convey more information than others. They use Mask R-CNN for segmentation and an existing model to generate shape parameters. To aid in training with synthetic data, they propose an adversarial approach which they call a 'student teacher framework'. In this framework, synthetic images are grouped into ten groups based on camera position and the distance of a person to the camera. In training, a real image that has the highest loss per keypoint in the previous N images is used to

determine which type of synthetic image to use for training to ensure the network trains on the most difficult synthetic images.

2.2 The Modern Hand Tracking System

The goal of hand tracking system is to find and extract information about a hand from an image containing one. That information must be sufficiently accurate to distinguish it from some other meaning.

The general idea of modern hand tracking systems is as follows. Given an image (RGB and/or depth) that contains a hand, the image is first segmented to a Region of Interest (ROI) which is either semantically segmented (pixel-wise), or segmented with a tight bounding box. From there, the ROI is passed into a model that employs a neural network and sometimes an additional system to take the shape of a hand into account. From there, the system outputs a series of 3D points that describe the locations of certain keypoints in the hand. This selection of keypoints can then be used to extract what information the hand is conveying in a low-dimensional environment, examples of this include [28, 29, 30]. Most research effort appears to go into the use of depth cameras because the input is often less ambiguous than RGB images, and it preserves spatial information about the hand. This has been encouraged by the increasing popularity and quality of depth sensors. RGB-based hand tracking at this time appears far more elusive, given the increased ambiguity in the image due to diverse backgrounds, different skin colour, lighting, among other ambiguities. To give an example of this difference, it is easy to segment a depth image of a hand in front of a wooden floor by discarding all distant pixel values, in contrast with an RGB image, the skin color might be similar to the wooden floor so the segmentation task is far more challenging.

One issue highlighted in literature is the lack of data, many of these same papers also produce a dataset of their own to address this. Part of the problem is that it is difficult to make such datasets, as well as this, many of these datasets rely on few participants, usually in the order of 10-20 participants at most. The core of many modern hand tracking systems use CNNs, which require large datasets to train on. Exactly how much a hand tracking system needs is unknown, but other CNN tasks require in the order of 100,000-1,000,000 images or more. The reason for the lack of hand tracking data is that labelling it is difficult. It is far too labourious for a human to label a hand dataset at scale, and indeed the accuracy may not be good. Notably in [4], they manually annotate their dataset, but this is only for 2D keypoints, and they simply omit keypoints that are occluded from the image. Other computer vision applications have seen a lot of real-world success because of the existance of large datasets for their domains, notably image classification[1]. For image classification tasks, it is far easier to label, notably the *reCAPTCHA* project got internet users to label images by combining the task with spam detection[2]. This is not possible with hand tracking however, and no scalable system exists to label a large hand tracking dataset without the help of an automatic method.

To automatically label datasets is a chicken and egg problem. The problem is thus: hand tracking systems need more data, and the data needs to be more accurate, but there is not sufficient data. This data can be captured relatively easily, but to label it at scale is practically impossible, therefore a hand tracking system should be used to annotate the data, but no such system exists. To get around this, past datasets have attached gloves, or other sensors to the hand to find the keypoint locations, but this then compromises the resultant images because the glove or sensor can be seen in the image. More recent examples have used a kinematic model of a hand to refine the estimate of the hand such as in the NYU dataset [11]. Other labelling efforts have involves placing discrete magnetic sensors on the fingertips and using an inverse kinematic model to estimate the other points[1]. One problem with both of these datasets is that they both have a small amount of participants (two and ten respectively). This problem can be solved relatively easily. A problem that is harder to address however is the quality of the annotation, which goes back to the chicken and egg problem highlighted above. We can only hope that the annotation strategies of [1, 11] are accurate. It is doubtful that the authors manually scrutinised each and every image. This is a big problem, because for any machine learning task, the system will only perform as well as the training data given to it. One way around this is to use synthetic data. This idea is explored in [7], when they create their *SynHand5M* dataset with perfect groundtruth, but they used a different hand model which they themselves admitted had limited realism. They also designed a new hand tracking system at the same time, which means that their dataset has not been validated independently. Progress has been made since then with the advent of the MANO model[9]. They specifically set out to address the problem of a lack of realism with hand data, and to the best of my knowledge, no further attempt at creating a large

[1]`http://image-net.org` (accessed 29th April 2020)

[2]`https://www.ted.com/talks/luis_von_ahn_massive_scale_online_collaboration` (accessed 29th April 2020)

depth dataset has been attempted since *SynHand5M*.

These same problems are discussed by [8], and they posit MANO as a synthetic data generation strategy to address these problems with existing hand data. As discussed in Section 1.1, the research question of this project is particularly influenced by the *Hands 2019* challenge. Given these problems, this dissertation explores the most recent synthetic hand data generation technique, the MANO model to see how it can address them.

3 Synthetic Data

Modern hand tracking systems typically employ machine learning, and the type of machine learning model usually used is a CNN [2, 4, 6, 7, 14]. Machine learning offers a way to solve a particular task without being explicitly programmed to perform that task. Instead, Machine Learning 'learns' how to do this task by 'training' on the data. This learned task is further tested on another dataset that the model did not train on to see how it performs. This idea of having a training dataset and a testing dataset is achieved in different ways. For example, in *k-fold cross validation*, $\frac{1}{k}$ of the dataset is used for testing and the remainder for training. The model is trained k times with a different section of the dataset used for training and testing each time. This method is useful for evaluating the performance of a model. For the purposes of this dissertation, a simpler method of: splitting the dataset into one training and test dataset, so the model is only trained once with the training set before evaluating performance on the test set.

Synthetic data is any kind of data that is not obtained through direct empirical observation. In the context of hand tracking, real data is obtained by a camera, while synthetic data is obtained typically through graphics rendering. Ideally, any machine learning system will be trained with the same kind of data that it is going to be used for future inference, and so synthetic data should aim to replicate what a real-world camera might see as opposed to a perfect image. There are different ways in which training data is made, depending on what kind of pipeline is being used for hand tracking, but this dissertation will follow the patterns used by [1, 10]. That is, the synthetic data that is generated consists of K images and associated groundtruths (the annotation of the keypoints) describing the location of N keypoints in 3D space for each image. Each depthmap image $\boldsymbol{Z}$ is an $n \times m$ matrix, where $\mathbb{Z}_{n,m} \in \mathbb{N}$ represents an individual pixel in the image, each pixel represents the depth to the nearest visible point, thus it is called a depthmap. A keypoint is a particular location of interest in the hand, such as a knuckle. An illustration of the keypoints used in this dissertation is shown in Figure 0.2. The goal of a hand tracking system is to take $\boldsymbol{Z}$ as an input and predict N 3D keypoints $\boldsymbol{\Phi}$, where $\boldsymbol{\Phi} = \{\boldsymbol{\Phi_1}, \boldsymbol{\Phi_2}, ..., \boldsymbol{\Phi_N}\}, \boldsymbol{\Phi}_n \in \mathbb{R}^3$. The groundtruth $\boldsymbol{Y}^i$, (where $\boldsymbol{Y}^i_n \in \boldsymbol{\Phi}$) represents the locations of N keypoints in 3D Cartesian space for K depthmap images in a dataset $\mathbf{D}$, where $\mathbf{D}_k \in \boldsymbol{Z}$. The hand tracking system is trained with a training dataset $\mathbf{D}^t$, and is tested with a separate test dataset $\mathbf{D}^v$. When testing the hand tracking model, the set of predictions $\boldsymbol{Y}^o$ (where $\boldsymbol{Y}^o_n \in \boldsymbol{\Phi}$) is compared with the groundtruth $\boldsymbol{Y}^i$. This comparison is performed using a performance metric E.

The goal of this dissertation is to train a hand tracking system on synthetic data such that it can accurately predict future keypoints from a real world sensor. The quality of the synthetic data is evaluated by training an existing hand tracking model V2V-Posenet[2] and evaluating its performance with the synthetic dataset in comparison to a real dataset, the MSRA dataset[10]. To obtain a fair evaluation, the model is not modified. An assumption is made that the right and left hand are the same, all generation of a hand focusses on the right hand in this dissertation, and that the left hand can be obtained by flipping the image on the y-axis. The methods used to generate the synthetic data are described herein.

3.1 Synthetic Data Generation Methods

The spread, or domain of data, describes the nature of the data. In the context of hand data, this implies what kinds of hand articulations, hand shapes, and camera angles the data covers. Previous work has generally focussed on 'frontal' views, where the camera is facing towards the person. The other main type of perspective for hands is the 'egocentric' perspective, where the camera is facing away from the person towards the hand, an example of this is a head-mounted camera looking at the hands. Egocentric perspectives are typically harder for a hand tracking system to recognise because the possibility of self-occlusion is greater.

The MANO model is used as the model for generating synthetic data. In theory, it is possible to generate a comprehensive dataset covering all possible hand articulations, shapes, and camera positions. This is problematic in practice however for several reasons. Firstly, the question of what constitutes a comprehensive dataset is unknown, and given the dimensionality of the problem, the generation of an ever-increasingly comprehensive dataset becomes computationally infeasible for the hardware available in this dissertation

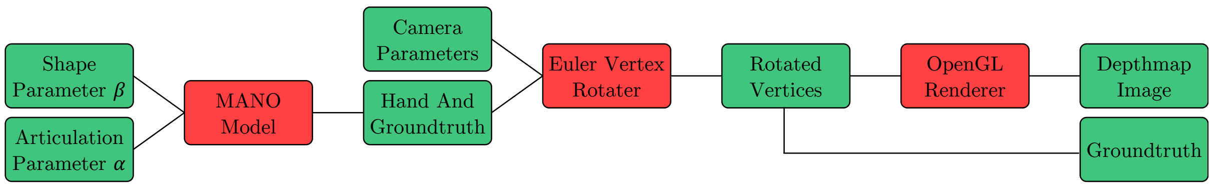

Figure 3.1: A diagram of the process for generating the Random MANO Dataset. The parameters α, β, and camera parameters are inputs, and the depthmap image and grountruth are outputs.

(the generation method described in Section 3.1.2 takes 607 milliseconds on average to generate a single image). The subsections herein describe two ways of generating synthetic hand data, and they both highlight the dimensionality issue. The first method sees all generation parameters determined at random, which gives a completely random dataset described in Section 3.1.1. The other generation method attempts to recreate a real dataset using the groundtruth of that real dataset as a reference.

3.1.1 Random Generation Method - *Random MANO Dataset*

This is the first synthetic data generation strategy investigated, this dataset is called the *Random MANO Dataset*. This method involves generating an entirely random synthetic dataset, drawn from a uniform distribution, the meaning of which is described below. A diagram of this process is shown in Figure 3.1.

Generating Poses and Hand Shapes

The basic MANO model takes two parameters, $\boldsymbol{\beta} \in \mathbb{R}^{10}$ and $\boldsymbol{\theta} \in \mathbb{R}^{45}$, describing shape and articulation respectively, giving a total of 55 parameters. Following on from [26], The full 45-dimensional $\boldsymbol{\theta}$ parameter is used as opposed to the *principal component analysis*-based 6-dimensional subspace used in the original MANO implementation. This generates 778 vertices describing a hand based on the input parameters. The groundtruth can be obtained from an additional regressor that takes the 778 vertices as an input and outputs 16 keypoints as a grountruth. I am not aware of any a priori relationship between the input and output of this model, that is, knowing what $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ input produces a particular 778 vertex output. Given the size of the input parameters, a random number is drawn for each point $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, drawn from a uniform distribution in the range $[-2.0, 2.0]$, the range of values in-between is limited by floating point hardware alone. The underlying theory behind using this regimen is that, given a sufficient number of random draw of values for $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, they will eventually converge towards a 'comprehensive' dataset insofar as the MANO model can accommodate. Giving the MANO model a zero vector for $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ returns a hand in its rest pose (the pose that the hand naturally regresses to when no muscular force is imparted on the fingers), and the greater each index in these vectors deviates from zero, the more diverse of a shape that is outputted (with fingers going in progressively more complex articulations for larger values for each index of $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$). This particular range was chosen through empirical observation, a range greater than this tends to give unrealistic or impossible poses that a real hand can produce. The original MANO model keypoint regressor outputs 16 keypoints, which does not match with the 21 keypoints used with the MSRA dataset, so in this dissertation, the MANO setup for this project is taken from the code of [26, 31].

Camera

Before rendering the 778 vertices into a depthmap image, the camera must be set appropriately. The MANO model always generates the 778 hand vertices and 21 keypoints in the same global orientation, therefore they must be rotated afterwards to augment the dataset for different camera perspectives. The image is rendered using OpenGL, and in OpenGL, the camera is always at the zero vector $\mathbf{0}$ position in Cartesian space, facing in the negative z direction. To manipulate the camera therefore, all of the other vertices in the scene are manipulated, and they must eventually be in the space between $[-1.0, 1.0]$ for each Cartesian axis to appear in the render[1].

This version of the model does not include a human body, so the camera must not view the hand in any form of ego-centric perspective because the wrist is cut off from the rest of the body and a hole will be seen in the rendered hand's wrist. The groundtruth describes not only the articulation of the hand, but also the orientation and position of the hand in 3D space. To account for this, any rotation of the 778 vertices must also be performed on the 21 keypoint groundtruth. The rotations are performed using three Euler rotations in practice. Specifically, all over a uniform distribution (in radians), the x-axis is rotated in the range $[0, 2\pi]$ radians, the y-axis is rotated randomly over $[-\frac{2}{3}\pi, -\frac{1}{3}\pi]$ radians, and the z-axis is rotated randomly over $[\frac{2}{3}\pi, \frac{1}{3}\pi]$ radians. The 778 vertices are rendered into a depthmap $\boldsymbol{Z}$. Example images generated can be seen in Figure 3.2.

[1]`https://www.khronos.org/opengl/wiki/Viewing_and_Transformations` (accessed 29th April 2020)

Normalisation

The mean and variance of the rendered image need to be adjusted to match the MSRA images. The variance is scaled to the same as the average variance of the MSRA dataset, and the mean is the same as the mean value of the z coordinate of each groundtruth value.

3.1.2 Recreating A Real Dataset - *IK MANO Dataset*

This is the second synthetic data generation strategy investigated, this dataset is called the *IK MANO Dataset*. The random data generation method described in Section 3.1.1 produces a good dataset in terms of image realism and groundtruth reliability, but it is difficult to compare with the MSRA dataset. As a reminder, the MSRA dataset [10] is based on certain poses within American Sign Language, and the distribution of poses likely does not match the distribution of data from the random data generation method. To address this concern, this section describes an alternative method of generating synthetic data. The goal of this method is to replicate the MSRA dataset image for image, which offers a direct way of comparing the datasets. This is a similar strategy to [6], where they generated their synthetic data based on the hand pose estimated from a hand captured with a real sensor. To achieve this, the MANO model can be loaded into Maya (as an FBX[2] file) as the full human body version (specifically the male version), see Figure 3.4. In this setup, the hand has 16 keypoints, one for the wrist, and an MCP, PIP, and DIP joint (see Figure 0.2), and each of these joints have three Euler rotation parameters. As well as this, each of these joints have translation parameters which can be used to change the shape of the hand, however this aspect is outside the scope of this dissertation. For hand articulation alone, this gives 45 rotation parameters, which is grossly overconstrained, since it allows us to deform the hand in ways that are not possible for a human. As mentioned in [7], having implausible hand images might not necessarily be detrimental towards a hand tracking system, however since the purpose of this is to recreate a dataset, this is not be a problem regardless. The following paragraphcs describes the process for reverse engineering the MSRA groundtruth for the purposes of deforming the hand articulation, and how the camera is adjusted to recreate the same camera perspective as the MSRA dataset.

[2]`https://www.autodesk.com/products/fbx/overview` (accessed 29th April 2020)

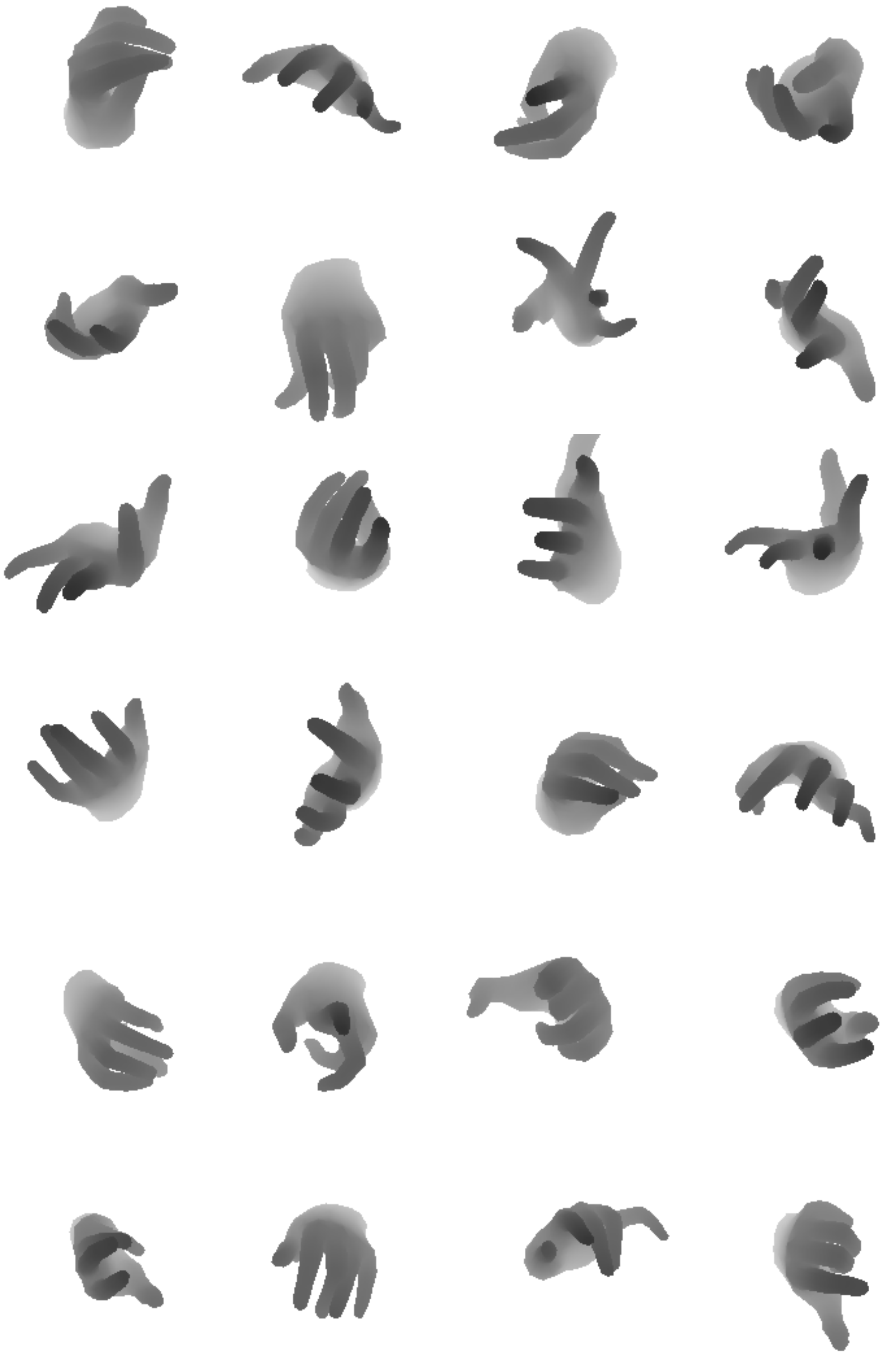

Figure 3.2: Examples of images generated using the random generation method.

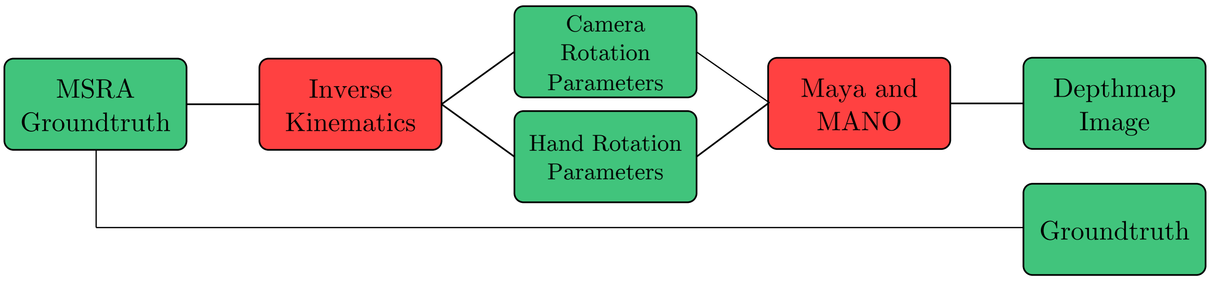

Figure 3.3: A diagram of the process for producing the IK MANO Dataset. Since the recreated depth image should have the same groundtruth as the equivalent MSRA, no processing is performed on it.

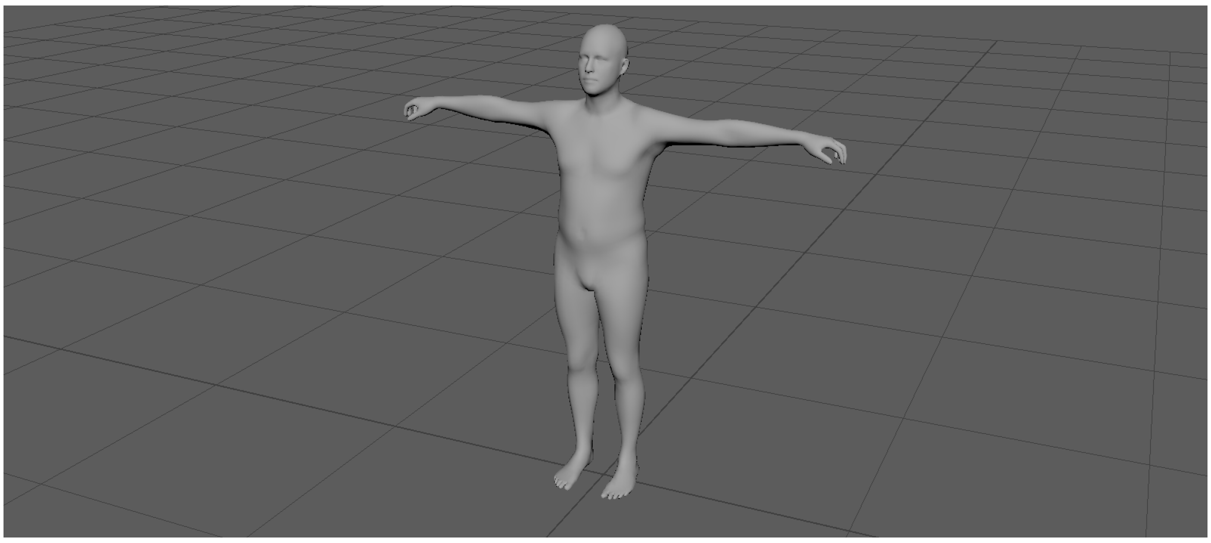

Figure 3.4: Screenshot of Maya with the full body MANO model, also called SMPLH.

Extracting hand articulation information from MSRA groundtruth

There is not a simple mapping from 21 keypoints to 15 finger joint rotations, therefore a mapping needs to be determined to find rotations of the finger to manipulate the MANO model in Maya. Formally, this process is Inverse Kinematics, a subset of Robotics. Human hands though are different to robots since the bone lengths of the MANO model and annotations are different. Taking advantage of the fact that the MSRA dataset includes an annotation for the tip of the finger, the property of the dot product ($\boldsymbol{a} \cdot \boldsymbol{b} = ||\boldsymbol{a}||\,||\boldsymbol{b}|| \cos \alpha$) is used to convert the 21 MSRA joint annotation to 15 finger rotations, in a similar way to how rotations are calculated in the *Cyclic Coordinate Descent Method*[32]. These rotations are then used to deform the hand model within Maya. Two joint vectors are subtracted from each other to produce a 'phalanx vector' describing the vector orientation between the two joints in consideration, this is described in Equation 1. To assist in mapping equation 1 to the finger rotations in Maya, it is used under the following assumption:

1. The palm of the hand (the region between the MCP joints and the wrist) is a rigid body.
2. All MCP joints have two degrees of freedom - x-axis and y-axis.
3. All PIP and DIP joints have one degree of freedom - x-axis only.
4. The four finger MCP joints form a perfectly straight line in 3D space

For simplicity, the wrist is also considered to be a joint. Figure 3.5 provides an illustration of this process, for example, the vectors $\boldsymbol{\Phi}_a$, $\boldsymbol{\Phi}_b$, and $\boldsymbol{\Phi}_c$ could represent the 3D locations of the MCP, PIP, and DIP respectively for the index finger, and the angle formed by Equation 1 is the angle that the index finger is deformed at the index PIP joint in the x-axis. An enumerated list of rotations performed in the x-axis can be found in Table 3.1.

As stated above, the MCP joint is also assumed to rotate in the y-axis, those rotations are calculated as follows. For the pinky finger, index finger, and thumb, their respective PIP and MCP joints are used in conjunction with the next MCP joint. For the middle and ring fingers, this is used for MCP joints on either side, and whichever adjoining combination produces a larger angle in absolute terms is used. See Table 3.2 for a fuller explanation.

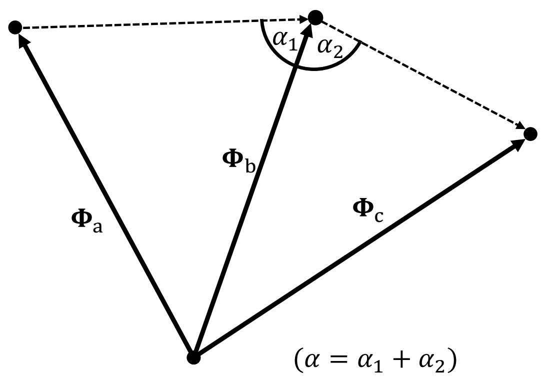

Figure 3.5: A visualisation of how the rotation of a joint is found. $\mathbf{\Phi}_a$, $\mathbf{\Phi}_b$, and $\mathbf{\Phi}_c$ are passed to Equation 1 to find the angle α.

$$\alpha = \arccos\left(\frac{(\mathbf{\Phi}_c - \mathbf{\Phi}_b)\cdot(\mathbf{\Phi}_b - \mathbf{\Phi}_a)}{||\mathbf{\Phi}_c - \mathbf{\Phi}_b||\ ||\mathbf{\Phi}_b - \mathbf{\Phi}_a||}\right) \tag{1}$$

Joint To Be Rotated (x-axis)	Rotation Parameters		
	$\mathbf{\Phi}_a$	$\mathbf{\Phi}_b$	$\mathbf{\Phi}_c$
TMCP	WRIST	TMCP	TPIP
IMCP	WRIST	IMCP	IPIP
MMCP	WRIST	MMCP	MPIP
RMCP	WRIST	RMCP	RPIP
PMCP	WRIST	PMCP	PPIP
TPIP	TMCP	TPIP	TDIP
IPIP	IMCP	IPIP	IDIP
MPIP	MMCP	MPIP	MDIP
RPIP	RMCP	RPIP	RDIP
PPIP	PMCP	PPIP	PDIP
TDIP	TPIP	TDIP	TTIP
IDIP	IPIP	IDIP	ITIP
MDIP	MPIP	MDIP	MTIP
RDIP	RPIP	RDIP	RTIP
PDIP	PPIP	PDIP	PTIP

Table 3.1: Fully enumerated list of how each joint was rotated in the ***x*-axis** for the inverse kinematics algorithm. See Equation 1 in Section 3.1.2. See Figure 0.2 for details on the joint names.

To calculate the x rotations for each MANO joint, the corresponding MSRA joint, as well as the previous and next joint is used in the above formula. For example, to calculate the x rotation for the PIP rotation of the ring finger, the RMCP, RPIP, and RDIP joints are inputted into Equation 1 letting $\mathbf{\Phi}_a$ =RMCP, $\mathbf{\Phi}_b$ =RPIP, and $\mathbf{\Phi}_c$ =RDIP.

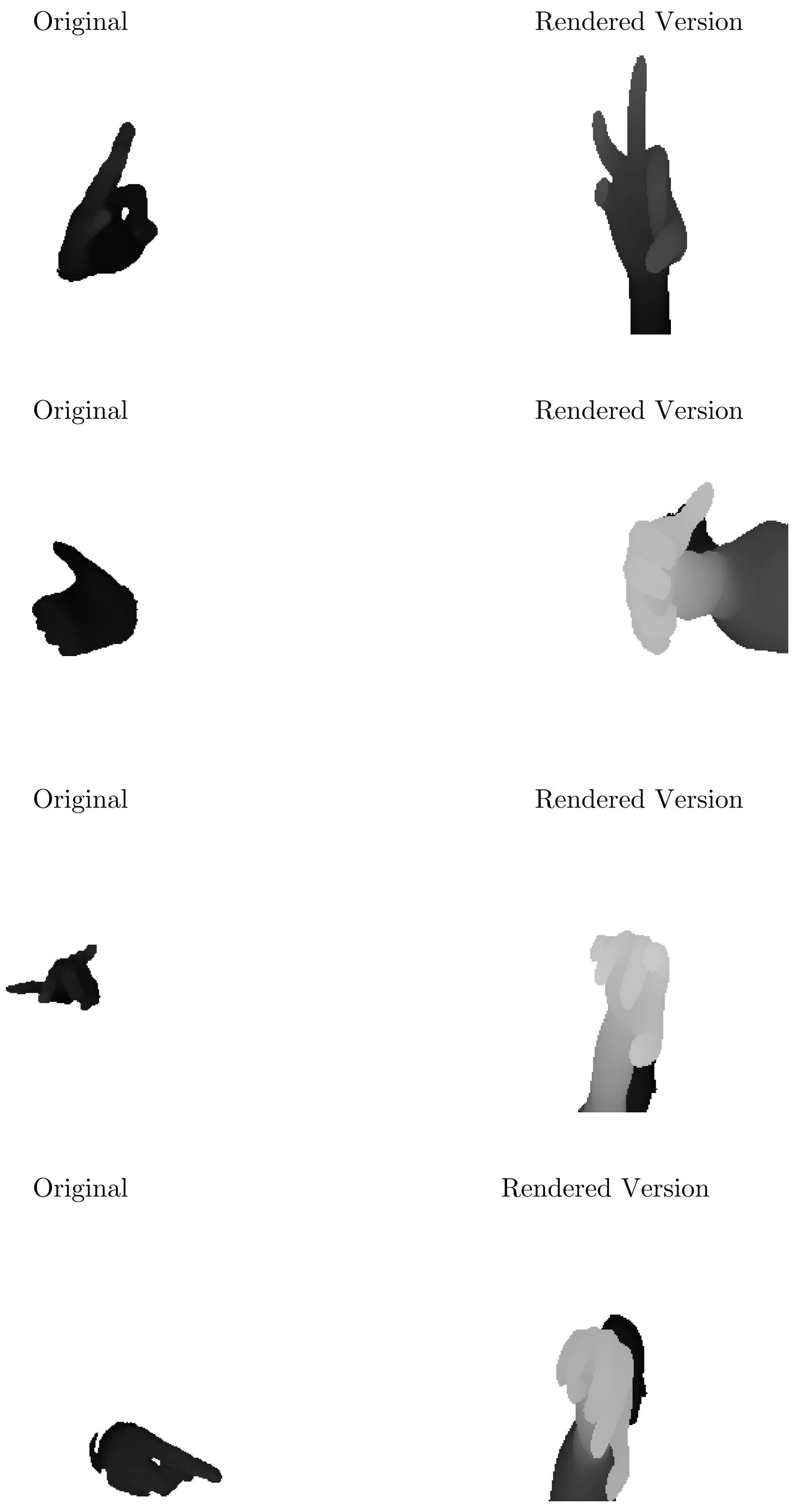

Figure 3.6: Examples of images generated by trying to recreate the MSRA dataset. Note that the difference in colour is not significant and is dependent on the visualisation method only. The visible body is also segmented in training.

Joint To Be Rotated (y-axis)	Rotation Parameters					
	Set 1			Set 2		
	$\mathbf{\Phi}_a$	$\mathbf{\Phi}_b$	$\mathbf{\Phi}_c$	$\mathbf{\Phi}_a$	$\mathbf{\Phi}_b$	$\mathbf{\Phi}_c$
PMCP	PPIP	PMCP	RMCP	*None*		
RMCP	RPIP	RMCP	PMCP	RPIP	RMCP	MMCP
MMCP	MPIP	MMCP	RMCP	MPIP	MMCP	IMCP
IMCP	IPIP	IMCP	MMCP	*None*		
TMCP	TPIP	TMCP	IMCP	*None*		

Table 3.2: Fully enumerated list of how each joint was rotated in the ***y*-axis** for the inverse kinematics algorithm. See Equation 1 in Section 3.1.2. Where a joint to be rotated has two sets of $\mathbf{\Phi}_a$, $\mathbf{\Phi}_b$, and $\mathbf{\Phi}_c$ vectors, the combination that produces the greatest magnitude of α in absolute terms with Equation 1 is chosen. See Figure 0.2 for details on the joint names.

Camera

The camera in Maya is the point in the scene where the renderer views from. It takes six parameters, three for Cartesian coordinates, and three for rotation: roll, pitch, and yaw ($\boldsymbol{R}_x$, $\boldsymbol{R}_y$, and $\boldsymbol{R}_z$). The camera for generating synthetic data is focused on the palm of the hand. The radius is set to a constant value, and the camera is always pointing towards a fixed point in the palm of the hand P, and it is moved by giving it the pitch, roll, and yaw parameters.

To achieve this, the camera is initially placed at point $[0, 0, r]$, where r is the distance to the palm of the hand. Using the three rotation parameters, this point is rotated on the x, y, and z axes by $\boldsymbol{R}_x$, $\boldsymbol{R}_y$, and $\boldsymbol{R}_z$ respectively. That rotated point is then translated to look at the hand by summing it by P, the resultant value is the new position in 3D space for the camera. The camera is then rotated by the same parameters $\boldsymbol{R}_x$, $\boldsymbol{R}_y$, and $\boldsymbol{R}_z$. This means that the camera will always be focused on the hand regardless of the input parameters $\boldsymbol{R}_x$, $\boldsymbol{R}_y$, and $\boldsymbol{R}_z$. The values of $\boldsymbol{R}_x$, $\boldsymbol{R}_y$, and $\boldsymbol{R}_z$ are obtained thus. The wrist and Index MCP coordinates are subtracted from each other to get a 3D vector describing the global rotation of the hand. The x, y, and z rotations are then found using the right-angled triangle definition $\alpha = \arctan\frac{a}{b}$. The hyperparameters for this method were optimised by iterating over different values. This iterative method works by generating a small dataset (e.g. 1000 images), and testing the performance on the V2V-Posenet model that has been trained on the MSRA dataset.

This method does not robustly recreate the camera perspective found in the MSRA

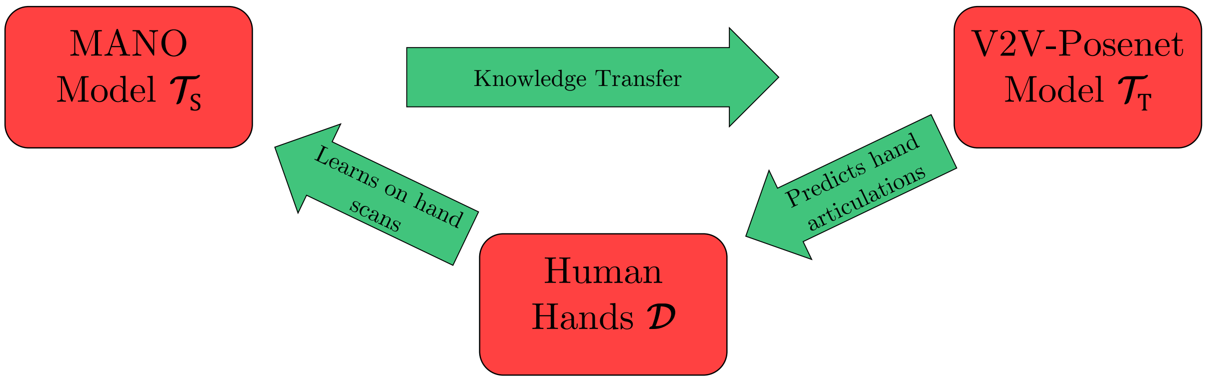
MANO Model $\mathcal{T}_S$
Knowledge Transfer
V2V-Posenet Model $\mathcal{T}_T$
Learns on hand scans
Predicts hand articulations
Human Hands $\mathcal{D}$

Figure 3.7: High level overview of knowledge transfer in terms of training a hand tracking model using a synthetic hand generator. A task is denoted by $\mathcal{T}$, and a domain of data is denoted by $\mathcal{D}$.

dataset, and unfortunately, the original MSRA dataset does not contain camera data.

Normalising rendered image

The process is the same as is described in Section 3.1.1.

3.2 Transfer Learning

Formally, training a hand tracking system on synthetic data is a form of Transfer Learning, or knowledge transfer. Transfer learning is the idea of transferring the knowledge gained from training one model to another model [33]. The theory is that although the two tasks may be different, there may be some aspects of that knowledge that can be transferred, which reduces the need for new expensive labelled data. For example, if a CNN model has been trained to find if there are cars in an image, when training a CNN system to find if there are lorries in the image, there might be aspects of the knowledge in the car classification model that can be transferred to the lorry classification model. In practice, an example of how that can be achieved is by reusing the weights of some of the layers in the car CNN for the truck CNN and training only the end layers of that truck CNN. The key questions for Transfer Learning are: when should this approach be used, what knowledge should be transferred, and how should this be done in practice.

It is important to distinguish transfer learning from machine learning. In machine learning, a task is learned *from scratch* with training data. A successful machine learning task will train a model entirely on a dataset that is within the domain of the future examples that it is trying to predict. Using the example in Figure 3.8, this model has learned to classify data into classes *red*, *green*, and *blue*. It learns these classifications given input data x_1 and x_2. Mathematically, this model can predict data from any domain, but it is only likely to be correct for data that it learned from. For example, the point marked with '?' lies in an ambiguous domain space, and we cannot be sure what class it belongs to. In contrast, in this dissertation the knowledge of the hand has already been learned in the MANO model, and the task is simply to transfer that knowledge to the V2V-Posenet model, the goal is to be able to train the V2V-Posenet model to learn from synthetic data without ever having to use real data.

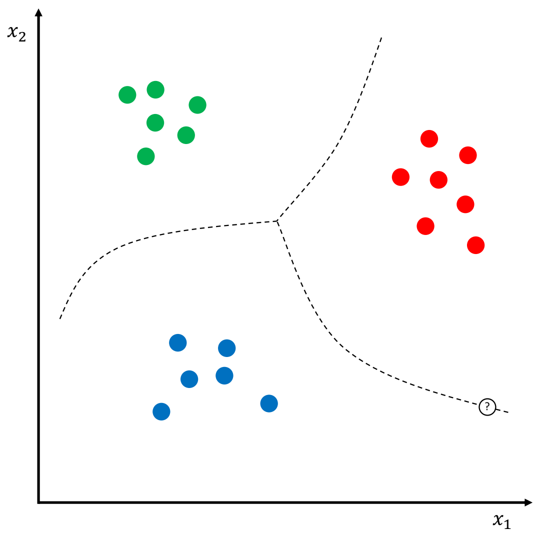

Figure 3.8: A toy example example of machine learning. This model learns to classify input data $\{x_1, x_2\}$ into three classes; red, green, and blue.

The question of when transfer learning should be used can be answered by asking if *negative transfer* has occurred. *Negative transfer* is where the data from the source model leads to the reduced performance of the target model. In general, an example of when this can occur is when the two tasks are too dissimilar. In the context of hand tracking, it is important that the method used to achieve transfer learning is robust. In hand tracking, the occurance of *negative transfer* is determined by comparing how the model performs with real data versus synthetic data.

In terms of answering what to transfer, it is worth first reflecting on what the synthetic hand generation model is, and what the hand tracking model is. The synthetic hand generation model knows about the structure of the hand and how that is related to 3D space, and by giving it plausible articulations, that can be transferred to the hand tracking model.po Using the the terminology in [33], the process of training a hand tracking system with synthetic data is an Inductive Transfer Learning task. In the context of this, the MANO model is the source task $\mathcal{T}_S$, and the target task $\mathcal{T}_T$ is the V2V-Posenet model. Both tasks have a common domain of human hands $\mathcal{D}$. A high level illustration of this concept is shown in Figure 3.7.

4 Experimental Setup

This section discusses practical considerations related to the experiments undertaken in this dissertation. The important details of the hand tracking system used are discussed in Section 4.1. The details of the nature of the real dataset used in this dissertation are described in Section 4.2. The performance metric of the hand tracking system used is defined in Section 4.3, this is particularly important as it is central to how the performance of a dataset is evaluated. Internal testing of the experimental setup is discussed in Section 4.4. Details of the experiments performed are discussed in Section 4.5. Finally, miscellaneous matters are discussed in Section 4.6.

4.1 Hand Tracking Model

The V2V-Posenet model[2] is used as the hand tracking system for training on, an overview of the network can be seen in Figure 2.2. It takes a depthmap $\boldsymbol{Z}$ as an input, and a 3D vector $\boldsymbol{v}$ denoting the centre of the hand in $\boldsymbol{Z}$ in 3D space. For this dissertation, where depthmap $\boldsymbol{Z}$ has an image resolution of $(a \times b)$, $\boldsymbol{v}_x = \frac{a}{2}$ and $\boldsymbol{v}_y = \frac{b}{2}$ and $\boldsymbol{v}_z = \boldsymbol{Z}_{\frac{a}{2},\frac{b}{2}}$. It outputs 3D predicted joints $\boldsymbol{\Phi}$ given a depthmap image $\boldsymbol{Z}$. Knowledge of the pipeline of this model is critical for generating the correct synthetic data. In the pipeline, the input depthmap is converted to a Point Cloud using $\boldsymbol{v}$ as a basis. In the context of this experiment, a Point Cloud is defined as an array of 3D vertices that describes the shape of an object in 3D space. Given that this is generated from a planar depth image, the Point Cloud does not define the entire surface of the hand. As such, self occlusions will also be visible in the Point Cloud. The algorithm for converting from a depthmap to a Point Cloud takes the indices of a pixel as an x, y coordinate pair, and the value in that pixel as the z coordinate. Points which are greater than a certain radius away from $\boldsymbol{v}$ are discarded, thus acting as a segmentation step between foreground and background. This Point Cloud is then turned into a voxel representation. Voxels are the 3D analogue to 2D pixels, and in the case of this setup, it is the 3D analogue to a 2D binary image. Formally, it is a 3D tensor, and can be seen as a verbose version of the Point Cloud.

An open source implementation of V2V-Posenet written in Python3 was used for this dissertation[1].

4.2 MSRA Dataset

As previously mentioned, the MSRA dataset[10] is used as a baseline to compare the synthetic dataset. The MSRA dataset consists of depth images of nine subjects, and each subject performs seventeen gestures drawn from American Sign Language, to which there are approximately five-hundred frames each[2]. The dataset is split 90/10 for training and testing respectively. The implementation described in Section 4.1 contains code for training on the MSRA dataset and it was not substantially modified.

4.3 Performance Metrics

The performance metrics discussed in this section are the cornerstone from which synthetic data is compared with real data. As previously discussed, the goal of a hand tracking system is to extract information about a hand from an image of a hand, and one way of evaluating the hand tracking system's performance is by using the metrics described below. Therefore, the questions asked about synthetic hand data in this dissertation are answered using these metrics.

The primary performance metric used is mean per-joint error, as used in[10]. This performance metric shall be referred to as *Global Error*. Specifically, for *each joint*, the average euclidian distance (in millimeters) in 3D space is computed between the groundtruth keypoint $\mathbf{Y}^i_{n,m}$ and predicted keypoint $\mathbf{Y}^o_{n,m}$. An important implication of this is that the prediction could predict the correct articulation of the hand, but the per-joint error is still high because the global orientation or location of $\mathbf{Y}^i_{n,m}$ is different to $\mathbf{Y}^o_{n,m}$. The model

[1] `https://github.com/dragonbook/V2V-PoseNet-pytorch` (accessed 29th April 2020)

[2] `https://jimmysuen.github.io/txt/cvpr15_MSRAHandGestureDB_readme.txt` (accessed 29th April 2020)

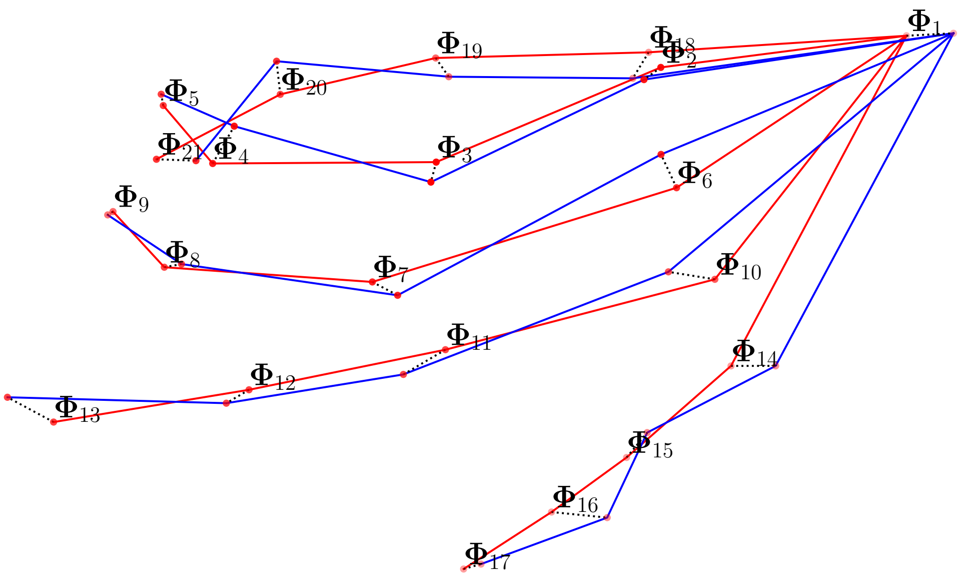

Figure 4.1: A visualisation of the *Global Error* performance metric. The prediction and groundtruth are shown in blue and red respectively. The performance metric measures the average distance for each joint between the prediction and groundtruth, denoted for this example by the dotted black line. Each $\mathbf{\Phi}_n$ corresponds to a keypoint as in Figure 0.2.

must therefore aim to correctly predict orientation as well as articulation to perform well on this metric.

There are K images in the test dataset, and the hand tracking model predicts N joints $\mathbf{\Phi}$ for each image $\boldsymbol{Z}$. E^G is the average error for all joints, and E^G_n denotes the average error for an individual joint $\mathbf{\Phi}_n$ (e.g. the per-joint error for the index MCP joint).

$$E^G_n = \frac{1}{K}\sum_{m=1}^{K} ||\mathbf{Y}^i_{m,n} - \mathbf{Y}^o_{m,n}|| \tag{1}$$

$$E^G = \frac{1}{N}\sum_{k=1}^{N} E^G_k \tag{2}$$

Through empirical observation, it was found that the V2V-Posenet model would often correctly predict the *articulation* and global *orientation* of the hand correctly, but not the global *position*. To account for situations where the model predicts the articulation and orientation correctly, but not the global position in 3D space, the following metric is also used. The above metric is modified by translating $\mathbf{\Phi}$ by the difference between the predicted and groundtruth values for the location of this wrist. This performance metric shall be referred to as *Local Error*. This means that the error for the wrist joint will be zero (thus this value is not reported in the local error metric). This second metric exists to provide more insight into performance. While the former metric is most important to optimise for a robust hand tracking system, this second metric can provide an intermediate goal to achieve in optimising the hand tracking model.

$$E_n^L = \frac{1}{K}\sum_{m=1}^{K} ||\mathbf{Y}^i_{m,n} + (\mathbf{Y}^o_{m,n} + (\mathbf{Y}^i_{m,0} - \mathbf{Y}^o_{m,0}))|| \tag{3}$$

$$E^L = \frac{1}{n}\sum_{k=1}^{N} E_k^L \tag{4}$$

Typically, the scale that a hand lies within in local space is *approximately* $200mm$, that is the maximum span of an adult human hand. The precise value of this range is not important, but it does serve to illustrate how the above performance metrics are interpreted, particularly the second metric. For example, if the average per-joint error of one joint is $80mm$, that is far away, if it is $800mm$, then there is likely something very wrong with the system, and if it is $8mm$, that is perhaps a reasonable average error. There is no exact interpretation to this however, which is why a baseline exists to judge experimental results.

As discussed above, the MSRA dataset is used as the baseline for this experiment. The crux of this dissertation is to evaluate the merits of using synthetic data for training a hand tracking system. To answer this question, the V2V-Posenet model is trained and tested using the respective training sets of MSRA, which produces a particular error value. The answer is provided by comparing the performance of training the V2V-Posenet model with the MSRA dataset versus training it with the synthetic datasets.

4.4 Software Testing

Software testing is the primary method used for improving the synthetic data pipeline. Software testing for this dissertation and the evaluation of the merits of using synthetic data are two separate concerns. While the latter is evaluated using the performance metrics in Section 4.3, software testing refers to the performance of the internal components of the synthetic data generation system. The performance metric is directly reliant on the internal performance of this system, and the performance metric *might* suffer because of a bad design of the synthetic data pipeline, and not because my hypothesis about synthetic data is false. It is imperative that the synthetic data pipeline is optimal so that the

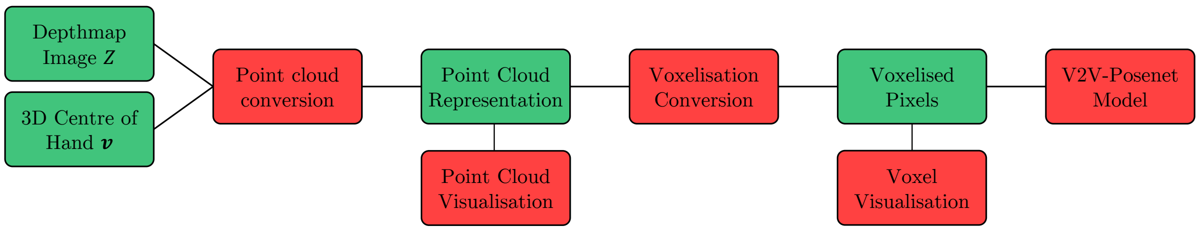

Figure 4.2: A diagram of the preprocessing steps for the V2V-Posenet model, from input depthmap image $\boldsymbol{Z}$ and centre $\boldsymbol{v}$ to voxels, this voxelised representation is then passed into the 3D CNN. The two red boxes for *Point Cloud Visualisation* and *Voxel Visualisation* denote the two visualisation stages described in Section 4.4.1.

performance metric in Section 4.3 solely answers the hypothesis, and not a failing in my design of the pipeline. To address this concern, different tests were devised to evaluate the performance

In both the synthetic data generation pipelines, these are aspects that I believe are liable to failing.

4.4.1 Data Visualisation

Visualisation served as the backbone of software testing in this dissertation. Ideally, unit testing or some other sort of numerical test would be used, but given the complicated nature of the data being handled, I am not aware of any reliable unit testing regimen. The following parts describe the individual visualisation tests that were performed.

Depthmap Visualisation

This step visualises the conversion of the mesh outputted by MANO into the depthmap, as well as the normalisation of this image. This step was important for seeing whether the camera was behaving as expected, as well as visualising the domain of the foreground pixels and ensuring that there is a clear boundary between these foreground pixels and the background. Examples can be seen in Figure 3.2 and Figure 3.6.

Pointcloud Visualisation

The conversion of the depthmap to the pointcloud it one of the most important steps in the pipeline. An example of this can be seen in Figure 4.3. As previously described, the

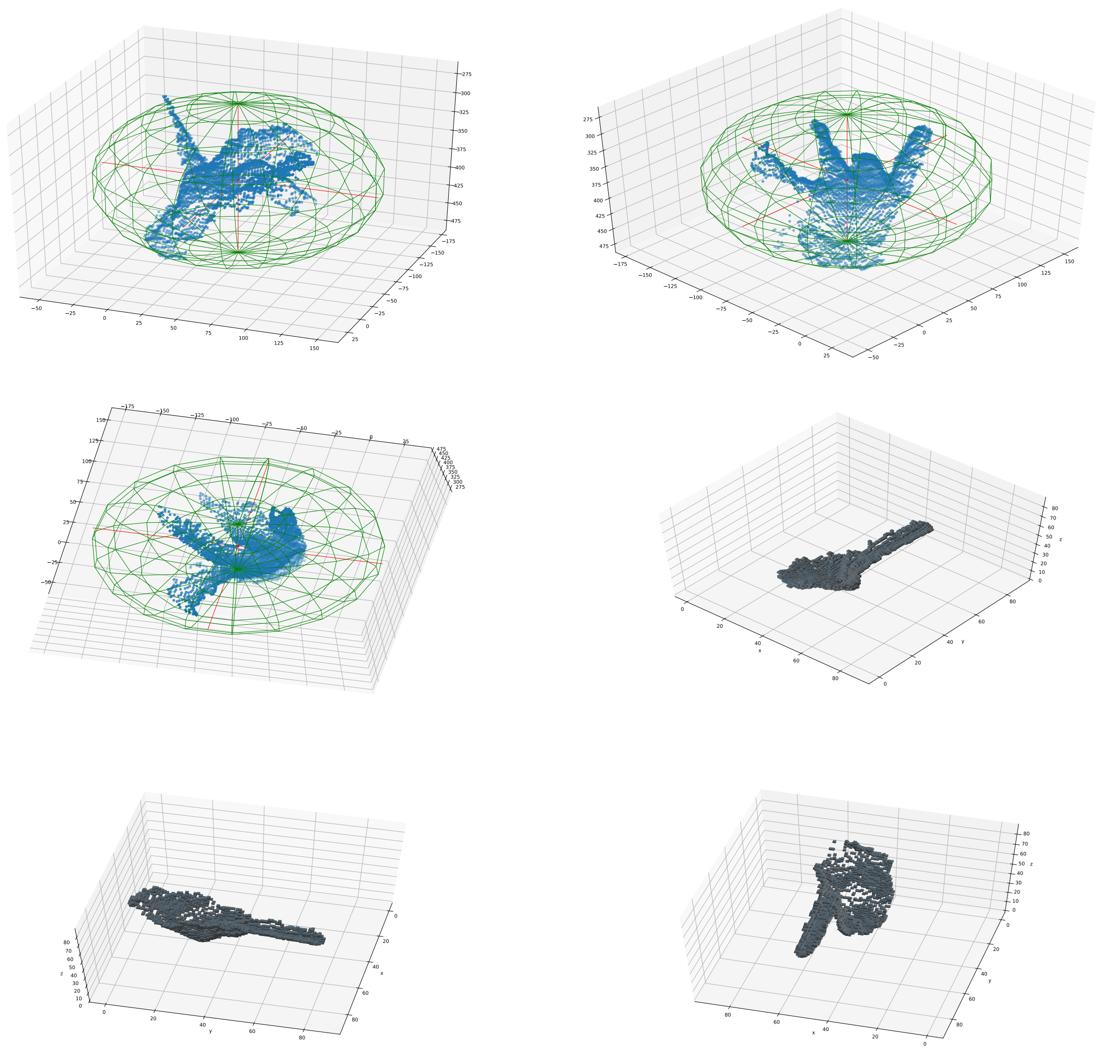

Figure 4.3: Examples the visualisation tools for the pointcloud (top row and middle left) and voxelisation (bottom row and middle right). For the pointcloud, all blue points that fall within the green sphere are voxelised, note as well that the background pixel points are not shown.

step of converting the depthmap to a Point Cloud uses a 3D centre point as a reference to the 'centre' $\boldsymbol{v}$ of the hand, and all points further than a certain distance to this point are discarded. In Figure 4.3, this centre is denoted by the confluence of the three thin red lines. The threshold in which points beyond $\boldsymbol{v}$ are discarded is denoted by the green sphere. The purpose of this step is to ensure that most points lie within this green sphere. The definition of *most* is defined by my empirical observation of the MSRA dataset visualised at this step. It was found that *most*, but not all points describing the hand lie within this green sphere. The parameters that can be optimised in this step are the centre point of the image. This step was implemented with *Matplotlib*.

Voxel Visualisation

Converting from pointcloud to voxels is the last step in the pipeline before the data is passed to the V2V-Posenet model. This step is largely robust for a given pointcloud representation since it is part of the implementation of the V2V-Posenet model that is used for this dissertation. It does however serve to confirm that the pointcloud visualisation step itself works. This step also was implemented with *Matplotlib*. Examples are also in Figure 4.3.

4.5 Experiments

This section details the experiments undertaken in this dissertation. All experiments involved training the V2V-Posenet model for 5 epochs with a batch size of 12. Every dataset consists of 67893 training images, and 8498 test images.

4.5.1 Baseline

The baseline of this dissertation is training the V2V-Posenet model with the real MSRA training dataset, it is then validated with the MSRA test dataset. It is also validated with the test datasets for the Random MANO and IK MANO datasets too.

4.5.2 Analysis Of Synthetic Data

The Random MANO and IK MANO datasets are investigated separately. Both respective training datasets are used to train the V2V-Posenet model under two scenarios; random initialisation of the V2V-Posenet weights, and one where the weights obtained from training the V2V-Posenet model on the MSRA training dataset are used as a base. In total, this gives four models obtained from training with synthetic data. Each of these four models are tested with the respective test synthetic dataset, and the MSRA test dataset, which gives eight testing scenarios. There are two error metrics described in Section 4.3, so this gives sixteen sets of results.

4.6 Miscellaneous

4.6.1 Execution Model

Random MANO Dataset The MANO parameters $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are determined at random for generating the Random MANO dataset. To make the generation of the data is made consistent over different runs, the random number generator is seeded with a constant value. When training, the images are generated at runtime before the model is trained and are stored in a temporary directory using the *tempfile* library[3] in Python. To give a fair comparison with the MSRA dataset, the same size for the training and test sets are used at 67893 and 8498 respectively. A different seed for the random number generator is used for training and testing.

IK MANO Dataset

As previously mentioned, recreating the real dataset is achieved in Maya[4], a 3D computer graphics programme. The actual renderer used is Arnold[5]. This was running on a different computer to the one used to run the V2V-Posenet model, and it lacked a GPU which meant that it was not feasible to generate the dataset in real time with training the model (it takes 12 hours and 53 minutes to generate the training and test dataset, or 607 milliseconds per image). Given the size of the dataset, the process was scripted. Maya comes with a Python environment for writing scripts which has API calls for scripting tasks within Maya. An important implication of using Maya Python is that it uses Python 2.7.11, which was released in December 2015 and lacks the support of many modern libraries. As well as this, it does not come with *PIP*, a common package manager in Python, which made it more challenging to use certain third party libraries. As a way around this problem, as well as to make the execution of the programme more efficient, two processes were used in the synthetic data generation process; Maya Python, as well as another process running Python 3. This was achieved using the *subprocess* library in

[3]`https://docs.python.org/3.7/library/tempfile.html` (accessed 29th April 2020)
[4]`https://autodesk.com/maya` (accessed 29th April 2020)
[5]`https://www.arnoldrenderer.com` (accessed 29th April 2020)

Python[6].

The calculation of the camera position, hand articulation, as well as the rendering happend within the Maya Python process. The Python 3 process did extra processing on the image (using the Python *subprocess* library), this consisted of normalising the rendered images, then compressing them to save disk space (using the numpy.savez[7] method). The execution of these processes worked in a producer-consumer model. The child process runs a modern version of Python to perform the post-processing of each depth image. The parent and child processes communicate using a Unix-style pipeline. Every time a new image is rendered in Maya, it saves the file to disk, when this occurs, the name, path, and grountruth of that image are passed to the child process in the pipeline, and the child performs the processing based on this information. In the execution pattern of the child, it waits for this information to be given to it. In order to strive towards deterministic execution, a `SIGALARM` is set to check a boolean that says whether the programme should stop when waiting for input, and if it receives nothing in the next three seconds, and that boolean is set to true, the child process stops. In the parent process, a wrapper class for calling the child process is implemented, and when that wrapper class is garbage-collected, it send a `SIGTERM` signal to the child, in the child, this sets the boolean to true, and if it is in the middle of processing an image, it will complete that first, and it will continue to process whatever is still in the pipeline before stopping.

4.6.2 Execution Speed

Execution speed refers to how quickly the synthetic data pipeline can execute, from generating, processing, training, and testing on the synthetic data. While this is not necessarily needed in terms of ensuring that the synthetic data pipeline is optimal towards improving the performance metric, it is important towards improving the speed of the pipeline itself, since it means that less time can wasted between per development cycle. Since the software for this dissertation was written in Python, this primarily focussed on the optimal use of libraries within Python. Python is an Interpreted language, and is not compiled to machine code before execution so code written in it executes slowly in comparison to software that is compiled (such as C, C++ and Fortran). Some libraries

[6]`https://docs.python.org/release/2.7.11/library/subprocess.html` (accessed 29th April 2020)

[7]`https://numpy.org/devdocs/reference/generated/numpy.savez.html` (accessed 29th April 2020)

in Python are written in compiled languages however. This means therefore that Python code should aim to offset as many instructions within the code to libraries that are written in compiled languages as possible.

4.6.3 Computers

Two computers were used in the experiments. The rational behind this is that one computer had a GPU which is needed for modern CNN training tasks, but it was not portable and lacked a GUI, a laptop was also used. The first computer, which was used for data visualisation and generating the MSRA-based synthetic dataset with Maya ran using a 2.6 GHz 6-core Intel i7 8850H, with 16GB of 2400MHz DDR4 RAM running macOS 10.15.3. The second computer used for the rest of the tasks in this dissertation used a 3.6 GHz 6-core AMD Ryzen 5 3600 with 32GB of 2400MHz DDR4 RAM and an Nvidia RTX 2060S with 8GB of GDDR6 RAM, running Ubuntu Server 18.04 LTS.

Since two computers were used for this dissertation, a practical concern during this project was how to communicate between the two computers. *Secure shell* (SSH) proved to be the most useful tool to achieve this. The internet connection for the server does not allow for port forwarding, which means that it cannot be an SSH server that is accessible from the internet. To solve this problem, a VPN server was deployed on Amazon Web Services (AWS). The AWS-based VPN server had a permanent connection to this computer, so when an SSH connection was made to the server from my laptop, it connected via this VPN server. Given the nature of this project, large quantities of data transfer were required, and AWS charges for this beyond a threshold, so the server did not connect to the internet via the VPN save for SSH communication. For the data visualisation tasks described in Section 4.4.1, the programme in the server interacts at different stages of the V2V-Posenet pipeline to extract information, and writes this data as a Python script, which can be copy and pasted onto my laptop. Given the small file size, this proved to be computationally feasible.

5 Results

Section 5.1 provides a summary of the baseline and results using the *Global Error* E^G and *Local Error* E^L metrics. Sections thereafter provide the full baseline and results using the per-joint *Global Error* E^G_n and *Local Error* E^L_n metrics. See Section 4.3 for details on the performance metrics. The real dataset is the *MSRA* dataset, and the synthetic datasets are the *Random MANO* dataset and *IK MANO* dataset, see Section 3.1 for details. See Figure 0.2 for details on the joint names.

Results are colour-coded, with good results (around 10mm) highlighted in **green**, progressively worse results are highlighted in shades of **yellow** and **orange**, with the worst results highlighted in **red** (> 90mm).

5.1 Summary

	Random MANO	IK MANO	MSRA
E^G	90.93 mm	88.57 mm	10.33 mm
E^L	118.89 mm	86.79 mm	14.19 mm

Table 5.1: The experimental baseline, showing the results when the model is first trained on the MSRA training dataset, then tested on the test MSRA dataset and test synthetic datasets.

	Trained With The Random MANO Dataset			Trained With The IK MANO Dataset		
Start From Random Weights	Test on MANO	E^G	8.90 mm	Test on MANO	E^G	77.44 mm
		E^L	6.88 mm		E^L	51.82 mm
	Test on MSRA	E^G	105.42 mm	Test on MSRA	E^G	59.03 mm
		E^L	120.09 mm		E^L	66.30 mm
Start From MSRA Weights	Test on MANO	E^G	8.25 mm	Test on MANO	E^G	74.45 mm
		E^L	6.54 mm		E^L	52.54 mm
	Test on MSRA	E^G	68.13 mm	Test on MSRA	E^G	54.45 mm
		E^L	96.29 mm		E^L	63.97 mm

Table 5.2: Summary of results over all experiments undertaken. Each experiment reports the average error over all joints for the two metrics described in Section 4.3. See Section 4.5 for details on the experiments performed.

5.2 Baseline Tests

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
10.20 mm	8.82 mm	10.10 mm	10.88 mm	13.40 mm	7.33 mm	8.23 mm	9.87 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
13.05 mm	7.56 mm	7.88 mm	9.54 mm	12.68 mm	9.13 mm	8.57 mm	9.27 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
12.04 mm	9.74 mm	11.02 mm	12.00 mm	15.55 mm	10.33 mm		

Table 5.3: Global error when model trained with the MSRA dataset initialised with random weights and tested on the MSRA test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
N/A mm	10.82 mm	13.18 mm	14.75 mm	17.31 mm	10.76 mm	13.45 mm	15.20 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
17.71 mm	11.59 mm	13.63 mm	15.21 mm	17.47 mm	13.20 mm	13.57 mm	14.14 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
16.62 mm	8.72 mm	12.60 mm	15.07 mm	18.86 mm	14.19 mm		

Table 5.4: Local error when model trained with the MSRA dataset initialised with random weights and tested on the MSRA test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
123.25 mm	105.61 mm	84.00 mm	86.13 mm	84.01 mm	82.64 mm	79.98 mm	70.77 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
72.60 mm	77.96 mm	72.01 mm	81.24 mm	72.84 mm	84.75 mm	79.84 mm	86.36 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
85.79 mm	138.94 mm	118.87 mm	102.89 mm	119.04 mm	90.93 mm		

Table 5.5: Global error when model trained with the MSRA dataset initialised with random weights and tested on the Random MANO dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
N/A mm	121.88 mm	117.93 mm	127.07 mm	130.78 mm	115.38 mm	113.78 mm	109.93 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
121.37 mm	107.26 mm	105.41 mm	115.54 mm	114.41 mm	114.18 mm	124.42 mm	108.09 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
112.18 mm	136.50 mm	128.28 mm	136.64 mm	116.67 mm	118.89 mm		

Table 5.6: Local error when model trained with the MSRA dataset initialised with random weights and tested on the Random MANO test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
73.79 mm	75.54 mm	86.05 mm	96.61 mm	108.52 mm	73.57 mm	89.94 mm	100.02 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
112.83 mm	75.50 mm	87.29 mm	92.89 mm	98.37 mm	78.78 mm	86.27 mm	92.02 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
98.80 mm	70.36 mm	77.56 mm	85.33 mm	99.91 mm	88.57 mm		

Table 5.7: Global error when model trained with the MSRA dataset initialised with random weights and tested on the IK MANO test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
N/A mm	67.93 mm	84.63 mm	96.64 mm	108.02 mm	69.45 mm	93.01 mm	103.37 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
115.10 mm	71.51 mm	90.29 mm	96.56 mm	100.27 mm	75.71 mm	88.43 mm	94.82 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
101.63 mm	42.45 mm	64.63 mm	77.68 mm	93.73 mm	86.79 mm		

Table 5.8: Local error when model trained with the MSRA dataset initialised with random weights and tested on the IK MANO test dataset.

5.3 Training With The Random MANO Synthetic Dataset

Wrist 8.39 mm	**IMCP** 8.30 mm	**IPIP** 8.68 mm	**IDIP** 9.48 mm	**ITIP** 11.10 mm	**MMCP** 7.70 mm	**MPIP** 7.90 mm	**MDIP** 8.35 mm
MTIP 9.97 mm	**RMCP** 7.28 mm	**RPIP** 7.54 mm	**RDIP** 8.29 mm	**RTIP** 10.46 mm	**PMCP** 7.25 mm	**PPIP** 7.73 mm	**PDIP** 8.75 mm
PTIP 10.97 mm	**TMCP** 8.29 mm	**TPIP** 8.65 mm	**PDIP** 9.68 mm	**TTIP** 12.12 mm	**Average** 8.90 mm		

Table 5.9: Global error when model trained with Random MANO dataset initialised with random weights and tested on Random MANO test dataset.

Wrist N/A mm	**IMCP** 5.08 mm	**IPIP** 6.13 mm	**IDIP** 7.41 mm	**ITIP** 9.74 mm	**MMCP** 4.84 mm	**MPIP** 5.89 mm	**MDIP** 6.93 mm
MTIP 9.16 mm	**RMCP** 4.68 mm	**RPIP** 5.69 mm	**RDIP** 7.09 mm	**RTIP** 9.87 mm	**PMCP** 4.89 mm	**PPIP** 6.03 mm	**PDIP** 7.56 mm
PTIP 10.23 mm	**TMCP** 3.95 mm	**TPIP** 5.08 mm	**PDIP** 6.89 mm	**TTIP** 10.37 mm	**Average** 6.88 mm		

Table 5.10: Local error when model trained with Random MANO dataset initialised with random weights and tested on Random MANO test dataset.

Wrist 75.91 mm	**IMCP** 99.64 mm	**IPIP** 108.10 mm	**IDIP** 106.09 mm	**ITIP** 108.97 mm	**MMCP** 109.41 mm	**MPIP** 104.96 mm	**MDIP** 105.99 mm
MTIP 110.81 mm	**RMCP** 110.22 mm	**RPIP** 106.17 mm	**RDIP** 107.01 mm	**RTIP** 107.00 mm	**PMCP** 103.82 mm	**PPIP** 111.16 mm	**PDIP** 112.72 mm
PTIP 111.82 mm	**TMCP** 85.61 mm	**TPIP** 101.98 mm	**PDIP** 112.97 mm	**TTIP** 113.48 mm	**Average** 105.42 mm		

Table 5.11: Global error when model trained with Random MANO dataset initialised with random weights and tested on MSRA test dataset.

Wrist N/A mm	**IMCP** 109.93 mm	**IPIP** 121.02 mm	**IDIP** 120.88 mm	**ITIP** 126.14 mm	**MMCP** 121.30 mm	**MPIP** 126.35 mm	**MDIP** 126.83 mm
MTIP 130.08 mm	**RMCP** 123.69 mm	**RPIP** 128.24 mm	**RDIP** 127.39 mm	**RTIP** 125.41 mm	**PMCP** 117.20 mm	**PPIP** 130.32 mm	**PDIP** 133.18 mm
PTIP 133.56 mm	**TMCP** 68.03 mm	**TPIP** 99.87 mm	**PDIP** 114.25 mm	**TTIP** 118.20 mm	**Average** 120.09 mm		

Table 5.12: Local error when model trained with Random MANO dataset initialised with random weights and tested on MSRA test dataset.

Wrist 7.77 mm	**IMCP** 7.45 mm	**IPIP** 7.79 mm	**IDIP** 8.70 mm	**ITIP** 10.49 mm	**MMCP** 6.88 mm	**MPIP** 7.23 mm	**MDIP** 7.94 mm
MTIP 9.75 mm	**RMCP** 6.66 mm	**RPIP** 6.82 mm	**RDIP** 7.76 mm	**RTIP** 9.87 mm	**PMCP** 6.61 mm	**PPIP** 7.05 mm	**PDIP** 8.12 mm
PTIP 10.19 mm	**TMCP** 7.79 mm	**TPIP** 7.96 mm	**PDIP** 9.12 mm	**TTIP** 11.29 mm	**Average** 8.25 mm		

Table 5.13: Local error when model trained with Random MANO dataset initialised with MSRA weights and tested on Random MANO test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
N/A mm	4.84 mm	5.72 mm	6.90 mm	8.98 mm	4.73 mm	5.66 mm	6.72 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
8.84 mm	4.50 mm	5.39 mm	6.70 mm	9.18 mm	4.60 mm	5.61 mm	7.10 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
9.53 mm	3.94 mm	4.97 mm	6.97 mm	9.92 mm	6.54 mm		

Table 5.14: Local error when model trained with Random MANO dataset initialised with MSRA weights and tested on Random MANO test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
66.27 mm	56.85 mm	81.36 mm	94.68 mm	92.93 mm	52.51 mm	74.99 mm	79.10 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
79.83 mm	48.45 mm	57.67 mm	63.70 mm	64.44 mm	57.67 mm	68.19 mm	72.50 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
75.13 mm	45.56 mm	48.79 mm	66.76 mm	83.37 mm	68.13 mm		

Table 5.15: Global error when model trained with Random MANO dataset initialised with MSRA weights and tested on MSRA test dataset.

Wrist	**IMCP**	**IPIP**	**IDIP**	**ITIP**	**MMCP**	**MPIP**	**MDIP**
N/A mm	83.09 mm	103.47 mm	117.67 mm	119.24 mm	83.72 mm	106.87 mm	110.43 mm
MTIP	**RMCP**	**RPIP**	**RDIP**	**RTIP**	**PMCP**	**PPIP**	**PDIP**
109.72 mm	83.35 mm	94.33 mm	98.46 mm	98.78 mm	87.96 mm	97.24 mm	102.23 mm
PTIP	**TMCP**	**TPIP**	**PDIP**	**TTIP**	**Average**		
107.56 mm	57.74 mm	71.35 mm	88.34 mm	104.33 mm	96.29 mm		

Table 5.16: Local error when model trained with Random MANO dataset initialised with MSRA weights and tested on MSRA test dataset.

5.4 Training With The IK MANO Synthetic Dataset

Wrist 64.53 mm	**IMCP** 71.31 mm	**IPIP** 76.66 mm	**IDIP** 83.73 mm	**ITIP** 91.78 mm	**MMCP** 71.68 mm	**MPIP** 76.27 mm	**MDIP** 79.25 mm
MTIP 84.55 mm	**RMCP** 73.99 mm	**RPIP** 77.06 mm	**RDIP** 79.68 mm	**RTIP** 83.36 mm	**PMCP** 76.68 mm	**PPIP** 77.78 mm	**PDIP** 80.07 mm
PTIP 84.91 mm	**TMCP** 66.25 mm	**TPIP** 71.62 mm	**PDIP** 74.36 mm	**TTIP** 80.73 mm	**Average** 77.44 mm		

Table 5.17: Global error when model trained with IK MANO dataset initialised with random weights and tested on the MANO test dataset.

Wrist N/A mm	**IMCP** 40.75 mm	**IPIP** 49.01 mm	**IDIP** 58.38 mm	**ITIP** 68.74 mm	**MMCP** 40.14 mm	**MPIP** 49.02 mm	**MDIP** 54.42 mm
MTIP 60.87 mm	**RMCP** 43.87 mm	**RPIP** 50.40 mm	**RDIP** 54.28 mm	**RTIP** 58.48 mm	**PMCP** 50.06 mm	**PPIP** 54.64 mm	**PDIP** 58.27 mm
PTIP 63.69 mm	**TMCP** 26.88 mm	**TPIP** 43.10 mm	**PDIP** 50.68 mm	**TTIP** 60.74 mm	**Average** 51.82 mm		

Table 5.18: Local error when model trained with IK MANO dataset initialised with random weights and tested on the MANO test dataset.

Wrist 58.33 mm	**IMCP** 51.33 mm	**IPIP** 57.47 mm	**IDIP** 65.04 mm	**ITIP** 82.37 mm	**MMCP** 44.27 mm	**MPIP** 43.59 mm	**MDIP** 54.36 mm
MTIP 76.42 mm	**RMCP** 41.48 mm	**RPIP** 40.94 mm	**RDIP** 51.88 mm	**RTIP** 68.88 mm	**PMCP** 44.63 mm	**PPIP** 49.97 mm	**PDIP** 59.52 mm
PTIP 72.52 mm	**TMCP** 63.81 mm	**TPIP** 71.49 mm	**PDIP** 71.15 mm	**TTIP** 70.23 mm	**Average** 59.03 mm		

Table 5.19: Global error when model trained with IK MANO dataset initialised with random weights and tested on the MSRA test dataset.

Wrist N/A mm	**IMCP** 53.50 mm	**IPIP** 66.73 mm	**IDIP** 77.73 mm	**ITIP** 98.87 mm	**MMCP** 53.44 mm	**MPIP** 62.64 mm	**MDIP** 71.36 mm
MTIP 88.12 mm	**RMCP** 54.36 mm	**RPIP** 58.71 mm	**RDIP** 62.11 mm	**RTIP** 73.09 mm	**PMCP** 58.79 mm	**PPIP** 67.05 mm	**PDIP** 74.22 mm
PTIP 83.94 mm	**TMCP** 33.65 mm	**TPIP** 53.55 mm	**PDIP** 62.18 mm	**TTIP** 72.02 mm	**Average** 66.30 mm		

Table 5.20: Local error when model trained with IK MANO dataset initialised with random weights and tested on the MSRA test dataset.

Wrist 68.10 mm	**IMCP** 68.19 mm	**IPIP** 69.85 mm	**IDIP** 75.70 mm	**ITIP** 86.24 mm	**MMCP** 67.60 mm	**MPIP** 69.82 mm	**MDIP** 76.65 mm
MTIP 83.70 mm	**RMCP** 66.68 mm	**RPIP** 70.17 mm	**RDIP** 74.45 mm	**RTIP** 77.76 mm	**PMCP** 71.64 mm	**PPIP** 73.91 mm	**PDIP** 77.46 mm
PTIP 79.96 mm	**TMCP** 69.41 mm	**TPIP** 75.58 mm	**PDIP** 77.99 mm	**TTIP** 82.66 mm	**Average** 74.45 mm		

Table 5.21: Global error when model trained with IK MANO dataset initialised with MSRA weights and tested on IK MANO test dataset.

Wrist N/A mm	**IMCP** 42.04 mm	**IPIP** 49.49 mm	**IDIP** 57.48 mm	**ITIP** 66.56 mm	**MMCP** 40.58 mm	**MPIP** 48.88 mm	**MDIP** 57.31 mm
MTIP 63.22 mm	**RMCP** 43.23 mm	**RPIP** 49.33 mm	**RDIP** 54.20 mm	**RTIP** 57.32 mm	**PMCP** 51.02 mm	**PPIP** 55.80 mm	**PDIP** 60.81 mm
PTIP 64.36 mm	**TMCP** 28.52 mm	**TPIP** 45.98 mm	**PDIP** 53.08 mm	**TTIP** 61.59 mm	**Average** 52.54 mm		

Table 5.22: Local error when model trained with IK MANO dataset initialised with MSRA weights and tested on IK MANO test dataset.

Wrist 44.48 mm	**IMCP** 44.75 mm	**IPIP** 50.11 mm	**IDIP** 64.87 mm	**ITIP** 78.88 mm	**MMCP** 42.33 mm	**MPIP** 41.90 mm	**MDIP** 56.19 mm
MTIP 74.84 mm	**RMCP** 41.85 mm	**RPIP** 40.55 mm	**RDIP** 50.45 mm	**RTIP** 64.50 mm	**PMCP** 45.85 mm	**PPIP** 49.38 mm	**PDIP** 59.83 mm
PTIP 71.03 mm	**TMCP** 47.58 mm	**TPIP** 53.30 mm	**PDIP** 56.41 mm	**TTIP** 64.41 mm	**Average** 54.45 mm		

Table 5.23: Global error when model trained with IK MANO dataset initialised with MSRA weights and tested on MSRA test dataset.

Wrist N/A mm	**IMCP** 49.42 mm	**IPIP** 61.99 mm	**IDIP** 77.78 mm	**ITIP** 90.02 mm	**MMCP** 49.63 mm	**MPIP** 58.49 mm	**MDIP** 71.92 mm
MTIP 87.89 mm	**RMCP** 51.45 mm	**RPIP** 56.66 mm	**RDIP** 65.87 mm	**RTIP** 76.86 mm	**PMCP** 55.80 mm	**PPIP** 62.23 mm	**PDIP** 73.03 mm
PTIP 83.21 mm	**TMCP** 32.27 mm	**TPIP** 50.75 mm	**PDIP** 58.25 mm	**TTIP** 65.79 mm	**Average** 63.97 mm		

Table 5.24: Local error when model trained with IK MANO dataset initialised with MSRA weights and tested on MSRA test dataset.

6 Conclusion

This final part3 discusses the results of Section 5.1 in Section 6.1. Future work is outlined in Section 6.2. A summary of this dissertation is provided in Section 6.3. Finally, a personal reflection is provided in Section 6.4.

6.1 Discussion Of Results

6.1.1 Baseline

The baseline for this dissertation is training the V2V-Posenet model on the MSRA training dataset. As anticipated, it performs well when tested on the *global error* metric, matching state of the art performance at 10.33mm, the slightly higher error value on the *local error* indicates that the V2V-Posenet model is also correctly predicting the global position of the hand. The results of the Random MANO test dataset are poor, and this is understandable, given that the datasets inevitably cover a different range of hand poses, however the results of the IK MANO dataset are more concerning. The IK MANO dataset is generated from the labels of the MSRA dataset, so in an ideal world, it should have similar performance, so this alone provides early evidence that the attempt to recreate the MSRA dataset synthetically failed.

6.1.2 Experiments

The following two sections describe the scenarios where the V2V-Posenet model is trained on the two respective synthetic test datasets. See Section 4.5 for details.

Random MANO Dataset

This section discusses the results when the V2V-Posenet model is trained on the Random MANO dataset.

It is first worth noting that these two datasets cover different kinds of poses. The MSRA dataset is a series of frames from a video where 10 subjects perform 17 different hand poses. Given this scenario, many of the frames in the MSRA dataset are similar, with only slight variations between frames for camera angle and pose. In training V2V-Posenet however, depthmap images are passed in at random. In contrast, the Random MANO dataset is entirely random, the hand shape changes in each frame, so in theory, if there are 10,000 images, this could have come from 10,000 different subjects. Each pose is entirely drawn from random, so no two images will be similar in the same way that they are in the MSRA dataset which comes from a video sequence. As well as this, the Random MANO dataset is likely to have higher quality images, while the MSRA dataset is likely to contain noise due to the nature of real world sensors. These differences are important when weighing the differences between these two results. Both the *global error* and *local error* metrics perform well when tested with this synthetic dataset, and better than the MSRA dataset. However, the *local error* performs better which indicates that the *global error* might improve if the global position of the hands is predicted better, which may indicate a problem with the MANO vertices-to-joints regressor. In my opinion, this dataset performed better than the MSRA dataset because the quality of the labels was better, they were more consistent with respect to each image. The MANO model can output accurate labels, while the MSRA dataset determined the labels using an optimisation algorithm [10] which will not always find the correct annotation. Indeed, in the ICVL Dataset [5], they admit that when the labelling algorithm fails, they just discard the image leading to an easier dataset to train on (less challenging images). It is curious that the cross comparing the MSRA dataset and Random MANO datasets does not perform well (i.e. training on one dataset and testing the trained model on the other dataset). While both datasets cover different ranges of poses, I do not think that that is the reason. Instead, a possible explanation is that there is a discrepancy in the domain of the groundtruth $\mathbf{\Phi}$ for a given depthmap image Z. The V2V-Posenet model learns to map a depthmap image to a particular domain of predictions to satisfy the *global error* metric, but given another dataset, it may map to a different domain of predictions. For example, for a depthmap image where the average foreground pixel value is 300, and for that type of image where the average foreground image is 300, it has a groundtruth where the average z value $\mathbf{\Phi}_{n,2}$ is 100, but in a *different* dataset, that same type of depthmap image might map to an average $\mathbf{\Phi}_{n,2}$ of 200. As long as the annotation mechanism is consistent, this domain issue is not a problem for datasets, but it is for comparing datasets. I am not aware of any recent dataset that makes use of data from previous datasets, and I think this is quite telling. The key theme of this dissertation has been the lack of data for hand tracking datasets, and yet, when an attempt is made to create a new dataset, while past techniques are considered, no author ever *uses* the previous data, and I believe this can be partly explained by this domain issue. Therefore, that is why I think the MSRA and Random MANO datasets do not compare well with each other.

IK MANO Dataset

This section discusses the results when the V2V-Posenet model is trained on the IK MANO dataset, the dataset that is generated within MAYA to recreate the MSRA dataset synthetically. The results overall are poor, with test results poor even on the test version of the synthetic dataset. The *local error* does significantly better than the *global error* when tested with the MANO dataset in both MSRA weights and random weights scenarios, and this to me indicates a problem with the camera step, that the global position of the hand in the groundtruth with respect to the images is inconsistent. Looking at the sample images in Figure 3.6, the hand articulation is generally correct, but the actual orientation of the hand varies widely. The camera orientation was chosen by looking at the orientation of the hand, since there was not information about the camera in the original annotation of the dataset, which clearly is not a good approach.

The method for for deforming the hand joints has shown promise, but the method for finding the camera location was not robust.

Results Conclusion

The purpose of this dissertation was to investigate the merits of using synthetic data to train hand tracking systems. Given the results discussed in Section 6.1.2, I believe that the use of synthetic data is a valid step in improving the performance of hand tracking systems, but more work is required. These experiments have shown that it is easier to generate synthetic data directly from the MANO model. In contrast, generating synthetic data from the grountruth of a real dataset is harder, since it involves the use of a dubious inverse kinematics algorithm in the absence of proper camera annotation of that real dataset.

6.2 Future Work

Generating synthetic data at random proved to be easier than making a pipeline for reverse engineering the grountruth. The error of the Random dataset was very low, far lower than the equivalent on the MSRA dataset. It would therefore be interesting to make a very big dataset, in the order of 10^7 or 10^8 to see how it performs on the V2V-Posenet model in comparison to a real dataset. As well as this, the problems reverse engineering the grountruth could be fixed, and hand shape could be factored in by calculating the bone lengths in the grountruth and deforming the model in Maya as appropriate. This grountruth-reversal method could also provide a way to augment real datasets like Bighand2.2m. An example of this could be to supplement this dataset with synthetic counterparts, using the grountruth of this dataset as a starting point, the angles of the individual joint deformations could be adjusted, the camera could also be adjusted. In addition to this, a synthetic dataset with hand-object interactions could be investigated for depth images like is done for RGB images in [6].

6.3 Summary

In this dissertation, a comprehensive evaluation of synthetic data in 2020 was performed using the MANO model as the state-of-the-art in generating synthetic data, with a view towards addressing the lack-of-data problem in hand tracking. Two regimens were demonstrated for generating such data: at random (The Random MANO Dataset), and based on another dataset (The IK MANO Dataset). The random generation method worked by drawing random parameters from a pre-selected uniform distribution for camera, hand articulation, and hand shape. The method where the synthetic data is based on another dataset worked by using inverse kinematics to deform the MANO model within MAYA, with a view towards augmentation of a real dataset.

6.4 Postscript

This dissertation marks the concluding work that I submit as part of a five year Engineering programme at Trinity College Dublin. This degree has taken me on a journey from a churlish youth to what I hope is someone prepared to take on life ahead. In the midst of submitting this dissertation, the world at large is in lockdown, and it is probably one of the worst times to be a graduate, but life must go on. I will be in a lifetime of emotional debt to all of the great people I encountered through my time here, and I mean this in a most sincere way. A special word of mention is in order towards *Dublin University Boat Club*, I will forever hold fond memories of both on-water and off-water endeavors. I often suspected, that I spent more time in boats than in lecture theaters.

References

[1] Shanxin Yuan, Qi Ye, Bjorn Stenger, Siddhant Jain, and Tae-Kyun Kim. Bighand2.2m benchmark: Hand pose dataset and state of the art analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4866–4874, 2017.

[2] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. V2V-posenet: Voxel-to-voxel prediction network for accurate 3D hand and human pose estimation from a single depth map. In *Proceedings of the IEEE conference on computer vision and pattern Recognition*, pages 5079–5088, 2018.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Yangang Wang, Cong Peng, and Yebin Liu. Mask-pose cascaded cnn for 2d hand pose estimation from single color image. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(11):3258–3268, 2018.

[5] Danhang Tang, Hyung Jin Chang, Alykhan Tejani, and Tae-Kyun Kim. Latent regression forest: structured estimation of 3D hand poses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(7):1374–1387, 2016.

[6] Franziska Mueller, Dushyant Mehta, Oleksandr Sotnychenko, Srinath Sridhar, Dan Casas, and Christian Theobalt. Real-time hand tracking under occlusion from an egocentric RGB-D sensor. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1284–1293, 2017.

[7] Jameel Malik, Ahmed Elhayek, Fabrizio Nunnari, Kiran Varanasi, Kiarash Tamaddon, Alexis Heloir, and Didier Stricker. Deephps: End-to-end estimation of 3D hand pose and shape by learning from synthetic depth. In *2018 International Conference on 3D Vision (3DV)*, pages 110–119. IEEE, 2018.

[8] Anil Armagan, Guillermo Garcia-Hernando, Seungryul Baek, Shreyas Hampali, Mahdi Rad, Zhaohui Zhang, Shipeng Xie, MingXiu Chen, Boshen Zhang, Fu Xiong, et al. Measuring generalisation to unseen viewpoints, articulations, shapes and objects for 3D hand pose estimation under hand-object interaction. *arXiv preprint arXiv:2003.13764*, 2020.

[9] Javier Romero, Dimitrios Tzionas, and Michael J Black. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics (ToG)*, 36(6):245, 2017.

[10] Xiao Sun, Yichen Wei, Shuang Liang, Xiaoou Tang, and Jian Sun. Cascaded hand pose regression. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 824–832, 2015.

[11] Jonathan Tompson, Murphy Stein, Yann Lecun, and Ken Perlin. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (ToG)*, 33(5):1–10, 2014.

[12] Toby Sharp, Cem Keskin, Duncan Robertson, Jonathan Taylor, Jamie Shotton, David Kim, Christoph Rhemann, Ido Leichter, Alon Vinnikov, Yichen Wei, et al. Accurate, robust, and flexible real-time hand tracking. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3633–3642, 2015.

[13] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015.

[14] Chengde Wan, Thomas Probst, Luc Van Gool, and Angela Yao. Dual grid net: hand mesh vertex regression from single depth maps. *arXiv preprint arXiv:1907.10695*, 2019.

[15] Jun Liu, Henghui Ding, Amir Shahroudy, Ling-Yu Duan, Xudong Jiang, Gang Wang, and Alex Kot Chichung. Feature boosting network for 3D pose estimation. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[16] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. Real-time 3D hand pose estimation with 3D convolutional neural networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(4):956–970, 2018.

[17] Jameel Malik, Ibrahim Abdelaziz, Ahmed Elhayek, Soshi Shimada, Sk Aziz Ali, Vladislav Golyanik, Christian Theobalt, and Didier Stricker. Handvoxnet: Deep voxel-based network for 3D hand shape and pose estimation from a single depth map. *arXiv preprint arXiv:2004.01588*, 2020.

[18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[19] Yiming He, Wei Hu, Siyuan Yang, Xiaochao Qu, Pengfei Wan, and Zongming Guo. Graphposegan: 3D hand pose estimation from a monocular RGB image via adversarial learning on graphs. *arXiv preprint arXiv:1912.01875*, 2019.

[20] Markus Oberweger, Paul Wohlhart, and Vincent Lepetit. Generalized feedback loop for joint hand-object pose estimation. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[21] Liangjian Chen, Shih-Yao Lin, Yusheng Xie, Yen-Yu Lin, Wei Fan, and Xiaohui Xie. Dggan: Depth-image guided generative adversarial networks fordisentangling RGB and depth images in 3D hand pose estimation. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 411–419, 2020.

[22] Danhang Tang, Qi Ye, Shanxin Yuan, Jonathan Taylor, Pushmeet Kohli, Cem Keskin, Tae-Kyun Kim, and Jamie Shotton. Opening the black box: Hierarchical sampling optimization for hand pose estimation. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2161–2175, 2018.

[23] Cheol-hwan Yoo, Seung-wook Kim, Seo-won Ji, Yong-goo Shin, and Sung-jea Ko. Capturing hand articulations using recurrent neural network for 3D hand pose estimation. *arXiv preprint arXiv:1911.07424*, 2019.

[24] Shile Li and Dongheui Lee. Point-to-pose voting based hand pose estimation using residual permutation equivariant layer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11927–11936, 2019.

[25] Alan Yuille and Daniel Kersten. Vision as Bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308, 2006.

[26] Seungryul Baek, Kwang In Kim, and Tae-Kyun Kim. Pushing the envelope for RGB-based dense 3D hand pose estimation via neural rendering. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1067–1076, 2019.

[27] David T Hoffmann, Dimitrios Tzionas, Micheal J Black, and Siyu Tang. Learning to train with synthetic humans. *arXiv preprint arXiv:1908.00967*, 2019.

[28] Guillaume Devineau, Fabien Moutarde, Wang Xi, and Jie Yang. Deep learning for hand gesture recognition on skeletal data. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 106–113. IEEE, 2018.

[29] Xuan Son Nguyen, Luc Brun, Olivier Lezoray, and Sébastien Bougleux. Skeleton-based hand gesture recognition by learning SPD matrices with neural networks. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pages 1–5. IEEE, 2019.

[30] Yiwei Wang, Cheolkon Jung, Inyong Yun, and Joongkyu Kim. Spfemd: Super-pixel based finger earth mover's distance for hand gesture recognition. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4085–4089. IEEE, 2019.

[31] Xiong Zhang, Qiang Li, Wenbo Zhang, and Wen Zheng. End-to-end hand mesh recovery from a monocular RGB image. *arXiv preprint arXiv:1902.09305*, 2019.

[32] Martin Fêdor. Application of inverse kinematics for skeleton manipulation in real-time. In *Proceedings of the 19th spring conference on Computer graphics*, pages 203–212, 2003.

[33] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.