Human Pose Estimation and its Application for Visual Effects

Bachelor Thesis in the Course of Audiovisual Media

Submitted by:	Roman Neugebauer
Supervised by:	Prof. Katja Schmid Michael Dohne

Stuttgart, 02.09.2018

Statutory Declaration

Hiermit versichere ich, Roman Neugebauer, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Human Pose Estimation and its Application for Visual Effects“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen

Stuttgart, 02.09.2018

Roman Neugebauer

Abstract

Human Pose Estimation and its Application for Visual Effects

Recreating human movements is essential for the realization of many VFX-shots. E.g., for animating CG characters or masking humans with rotoscoping. Meanwhile, machine learning systems offer possibilities to estimate human poses with single image inputs automatically. These systems propose workflows to simplify the capture and manual recreation of human movements.

In this thesis, the use of such systems in standard VFX applications will be covered. Primarily, focusing on realizing and evaluating a workflow to use 2D pose estimation for rotoscoping, and secondly using and evaluating 3D pose estimation to animate humanoid characters.

Kurzfassung

Die Verwendung von Human Pose Estimation für Visual Effects

Menschliche Bewegungen sind essentieller Bestandteil vieler VFX-shots. Z.B. Um CG-Charaktere zu animieren oder Masken von Menschen zu erstellen. Mittels maschinellem Lernen können menschliche Posen anhand von einfachen Bildern automatisch ermittelt werden. Diese Systeme legen Workflows nahe, die das erfassen und darstellen von menschlichen Bewegungen vereinfachen.

Dementsprechend wird in dieser Thesis die Nutzung solcher Systeme innerhalb von standard VFX Software behandelt. Einerseits mit dem Fokus auf einem Workflow um 2D pose estimation für Rotoskopie zu verwenden, andererseits um menschliche CG-Charaktere mittels 3D pose estimation zu animieren.

Contents

List of Figures

1 Introduction

Human movements and poses are essential for realizing a variety of VFX shots. E.g., to display the performance of human or human-like, digital characters, or to realize the interaction of human actors with digital elements. On the other hand, to separate actors with rotoscoping, which means tracing their movements.

To process and work with human movements of the real world, they have to be made digitally usable. Either by manually recreating poses and motion through animation or by explicitly capturing the performance with motion capture (Mo-Cap). These tasks usually involve lots of manual work, or well planned, specific shootings with sophisticated technical setups.

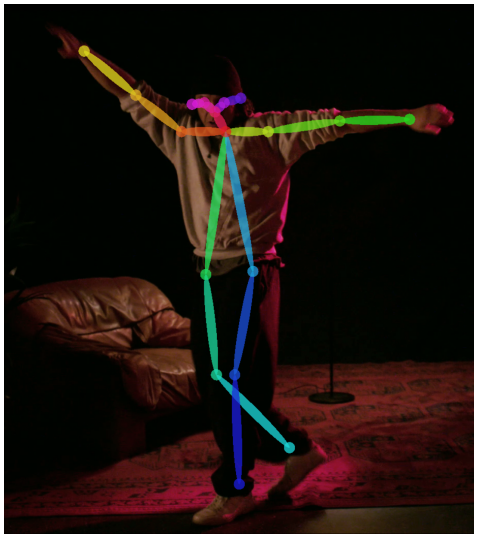


Figure 1.1: Pose Estimation

However, with the rise of machine learning in recent years, and the availability of large-scale datasets [Parloff, 2016], machine learning systems are able to estimate human poses automatically and with single image inputs [Cao et al., 2016], [Kanazawa et al., 2017]. With that, they propose workflows, which could benefit and simplify modern VFX production.

Besides a theoretical introduction, this thesis aims at the practical use of such systems in a modern VFX workflow. Especially covering how the systems can be used in combination with modern VFX applications and for which cases it benefits VFX production. The work focuses on large-scale poses and movements of the main limbs, disregarding animation of the face and hands.

The content of this thesis aims at people familiar with the standard software and techniques in modern VFX, especially rotoscoping, compositing, rigging, and animation.

1.1 State of the Art

Motion Capture (Mo-Cap) describes a variety of techniques to determine how an object moves in space. These techniques are currently the most common tools to capture and further process human poses and movement. Thus, Mo-Cap can be regarded as the state-of-the-art counterpart to human pose estimation. In VFX-production, Mo-Cap is usually used as a realistic human-like fundament for character animation. [Okun and Zwerman, 2010, p. 338 - 339]

The most popular high-end motion capture solutions base on optical locations, e.g., the OptiTrack system. The system works with multiple infrared sensors capturing reflective markers on the joints of an actor. Triangulation calculates the joint positions in 3D-space. Mechanical systems like xsens, measure joint angles with sensors in a suit. Given the biomechanical data of the actor, movement and position in space are calculated from the joint angles. [Kitagawa and Windsor, 2008, p. 8 - 10]

On the other hand, low-end solutions like the Microsoft Kinect capture human poses with one single RGB-Camera, one infrared sensor and without a special suit. Here the depth of every input pixel is calculated by triangulation. Humans poses are then estimated by separating humans and recovering joint positions with depth information. [Andersen et al., 2012]

These systems capture human poses and movements in real-time and are essential parts of many VFX-workflows. Nevertheless, they come with limitations because of their technical setup: While the infrared sensors only work with controlled lighting and in a limited space, a suit cannot always be worn together with a costume.

As Mo-Cap does not work on an already shot plate, a 2D-equivalent to human pose estimation would be separately planar tracking all body parts of a human. The position of each tracked body part defines pose and movement. This bases on tracing defined image features over time, but will not work, or need manual user input, if significant lighting changes, heavy motion blur or other temporal changes in pixel values occur. [Krishnant and Ravivt,] Moreover, different body parts occluding each other at times make continuous tracking difficult. E.g., planar-tracking the limbs of an actor rotating around his axis is somewhat tricky. The planar tracking data can and is often used to provide a fundament for animation of roto-shapes.

Recent systems based on convolutional neural networks (CNN) show promising results to estimate poses with simple setups. They can estimate 2D, and 3D human poses with a single RGB-image input.

1.2 Motivation

I chose to use human pose estimation systems, based on CNN's, and analyze the use of such for visual effects. More precisely, systems that estimate human poses with single RGB-images, as they could simplify common problems of rotoscoping, rotomation or Mo-Cap.

Manually recreating human poses by rotoscoping or rotomation is usually very time consuming and requires a lot of tedious work because an artist has to go through every frame and make manual adjustments to match pose and shape of the subject. Furthermore, for the simple recreation, no creative work nor input is needed. First off, the goal is to exactly match the poses and movement of a filmed subject and a digital character, respectively to match the edges and shape of a subject with roto-shapes. Hence, automation of such tasks could benefit VFX-production. Pose estimation can be the fundament for this: With HPE one can run pose estimation on a plate and automatically acquire the poses of a captured human. The output of pose estimation could be further processed to be used for rotoscoping or character animation.

Rotoscoping a human is in principle the manual articulation of the human shape, which bases on the corresponding pose [Okun and Zwerman, 2010, p. 570] . Thus, roto-shapes can be automatically placed at the estimated keypoints, which provides a fundament for refinement of the edges (see Chapter 3). Analogous, a character's skeleton can be animated according to the output of 3D pose estimation. In this case, pose estimation is the fundament for the animation itself (see Chapter 4).

Furthermore, pose estimation can be advantageous over established Mo-Cap techniques because of the simple setup: Respective systems, can output 3D keypoint locations from a single RGB-Image. Consequently, only the plate - which is fundamental in any case - or a single witness camera is needed. This is useful for decisions made in post-production, when no other information is available. On the other hand, if the quality of pose estimation fits the needs, pose estimation can make shooting more manageable, since no further hardware, other than a camera is needed. As a result, production is not limited to a Mo-Cap studio and actors are not limited to a specific motion capture volume, nor do they need to wear anything for Mo-Cap.

In short: Pose estimation systems are promising regarding easy setups and for decisions made in postproduction. Moreover, they could lead to the automation of every day but monotonous tasks.

1.3 Structure

First, a theoretical introduction to human pose estimation will be given. This covers the basic concepts of machine learning and the functionality of the used pose estimation systems.

Chapter 3 is about the use of pose estimation for rotoscoping. After a quick introduction into state of the art workflows, a new workflow using OpenPose is described in detail, and completed with an evaluation of this workflow.

Analogous to Chapter 3, in Chapter 4 a workflow for using 3D pose estimation to animate CG characters is described. Followed by an evaluation of the workflow.

Lastly, in Chapter 5 a conclusion from the gained knowledge is presented, and further prospects and possibilities are discussed.

2 Human Pose Estimation

Human pose estimation (HPE) is the study of obtaining human poses through images, by recovering the location, respectively the orientation of the main body parts. [Sigal, 2014, p. 1]

While modern Mo-Cap techniques capture the position and movement of an object in space, HPE is about explicitly inferring poses of humans. The respective systems base on machine learning and are especially trained on images of humans.

To work with these systems the fundamental theory behind machine learning will be covered. Especially the functionality of the systems practically used for this research

2.1 Representation of the Human Pose

The representation of human poses forms the basis, for the estimation of such. It is fundamental to understand what the relevant systems output and how poses are defined in standard VFX applications.

Biologically, the human skeleton is the basic structure for movement, and thus defines body poses. This serves as a basis for the most common representation of body poses as an kinematic tree $x = \tau, \theta_\tau, \theta_1, \theta_2, ..., \theta_n$. The root τ defines the position of a pose, and the joint angles θ define the orientation of the body parts, relative to their parent. The joints are defined by the directions in which they can move, called degrees of freedom (DoF). One joint can be defined by one to three DoF's. E.g., the knee can only move in one axis and is thus defined by only one rotational axis, while the neck-joint can move in all three axes. [Sigal, 2014, p. 2] (see a) figure 2.1 on the following page)

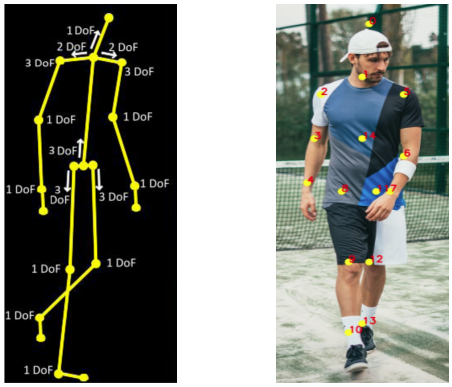
Alternatively, the body pose can be defined parametrically by the position of anatomical keypoints in either 2D - or 3D-space. E.g., $x = p_1, p_2, ..., p_n$, with p_i being the 2- or 3-dimensional coordinate. [Cao et al., 2016] (see b) figure 2.1)

[1]source: https://www.researchgate.net/publication/277006934 - access: 2018-08-31

[2]source: https://www.learnopencv.com/deep-learning-based-human-pose-estimation-using-opencv-cpp-python/ - access: 2018-08-31

This work aims at large scale pose estimation, so representations generally define the human main joints and rigid parts of the limbs and of the torso (limbs: shoulder, elbow, wrist, knee, and ankle. Torso: head, neck and hip).

This work covers pose estimation for rotoscoping and for CG characters. Human poses are defined differently for both: Poses of a CG character are principally defined by its skeleton. Which is such a described kinematic tree, bound to- and thus moving a mesh. Accordingly, one can transfer estimated joint angles onto a characters skeleton. [Okun and Zwerman, 2010, p. 345]



(a) Kinematic Tree

(b) Keypoint Locations

Figure 2.1: Pose Representations [12]

Rotoscoping, on the other hand, bases on mattes created with bezier curves or b-splines [Okun and Zwerman, 2010, p. 570]. These mattes are controlled by 2D control points and do not relate explicitly to any form of humanoid body pose representation. However, for rotoscoping a human they can be geometrically transformed according to anatomical keypoint coordinates, consequently relating human pose and roto-shapes (see Chapter 3).

Human movement can be seen as the succession of poses. So poses continually estimated over time can define the movement of a subject and thus the animation of a character or roto-shapes.

2.2 Machine Learning

A machine learning system adapts the processing of a given dataset, in regard to the functional context of input and output. In contrast, a regular algorithm always processes a specific input the same way. [Goodfellow et al., 2016, p. 98 - 99]

As a formal definition:

> "A computer is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E." [Mitchell, 1997] cited in [Goodfellow et al., 2016, p. 99]

For human pose estimation T would be to output keypoint locations, E would be training with datasets that contain pictures of humans and are labeled with keypoint locations, and P would be the accuracy of the estimation.

2.2.1 Artificial neural networks

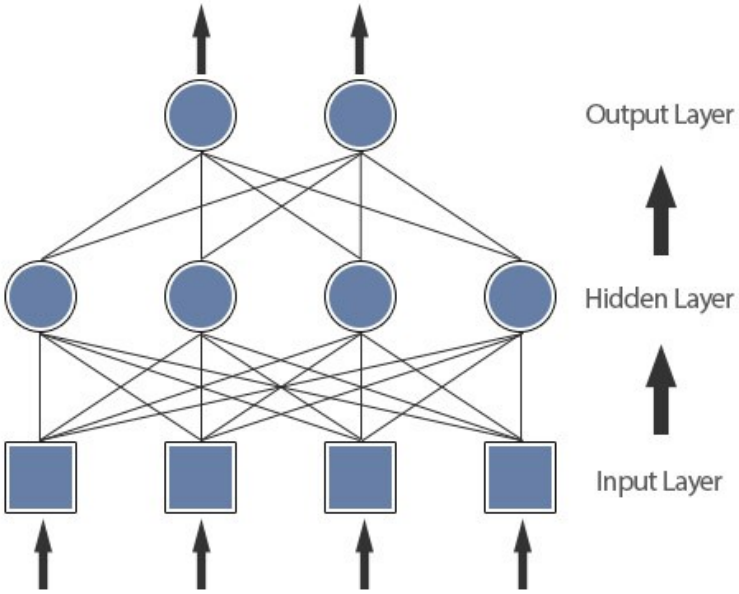


Figure 2.2: Basic feedforward network[3]

Since the human brain is the most efficient learning system we know, the structure of the human brain is reconstructed for machine learning. These reconstructions are called artificial neural networks, and build the basis for pose estimation. They consist of neurons that transmit data through different layers. Every network consists of an input layer, one or multiple hidden layers to process input data, and an output layer. There are two different types of networks: Feedforward neural networks, where information is passed forward in one direction, and recurrent neural networks, which allow loops inside the networks. [Nielsen, 2015, p. 14 - 16]

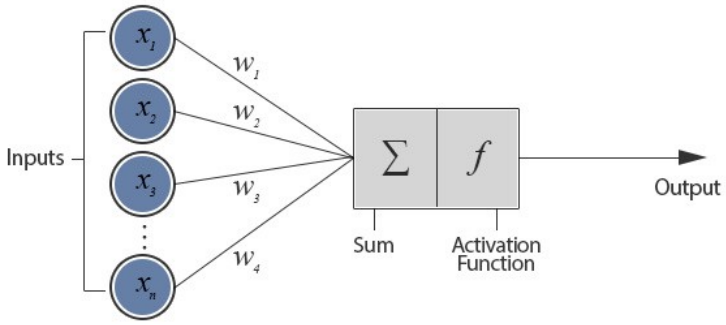


Figure 2.3: Structure of a Neuron[4]

In a basic network, the input of each neuron is the weighted sum of each neuron in the previous layer (See figure 2.3 on the following page). The sum is put through

[3]source: https://medium.com/technologymadeeasy - access: 2018-08-31
[4]source: https://medium.com/technologymadeeasy - access: 2018-08-31

an activation function. This activation function is a fixed mathematical operation to determine if a neuron *fires* or not, which bases on the sum at the input of the neuron. Nowadays the ReLU activation function is the most common. It thresholds the input at zero ($f(x) = max(0, x)$). [Goodfellow et al., 2016]

To understand the basic functionality, the architecture of a neural network to classify handwritten digits is often used: The fundament is a dataset containing grey scale images of handwritten digits. The images are 28 x 28 = 748 pixels, so the input layer would consist out of 748 neurons, one for each pixel. The output layer consists of 10 neurons, one for each digit to be classified (0...9). The hidden layer, between input and output, consists of 15 neurons. If the network is given a specific input digit, ideally the corresponding output neuron fires. [Nielsen, 2015, p. 16 - 18]

In the hidden layer the classification is broken down into simpler concepts. E.g., one specific neuron emphasizes only pixels that are part of a curve in the upper left part. This neuron, along with other neurons detecting curves in different parts, can be weighted heavily at the specific output neuron, which detects zeros. Causing this output neuron to fire. So the weights are the parameters of every network, emphasizing specific features in the dataset, to trigger a specific output. These weights are not defined by humans beforehand but are learned during training. [Nielsen, 2015, p. 16 - 20]

In other words: With any input x and a desired output function $f(x)$ the network is trained to approximate the output function by adjustment of the weights. A network defines a set of functions and the weights are the parameters of these functions. The universal approximation theorem shows that a neural network with at least one hidden layer can approximate any function. [Guliyev and Ismailov, 2016]

Deep learning

The principle of working with multiple hidden layers to solve complex problems is known as Deep Learning. E.g., a first layer to detect edges from an input-image, the second layer then recognizes specific collections of edges as contours. A specific collection of contours then would be recognized by the third layer, as a specific body part, and finally a collection of body parts can be recognized as a human. [Nielsen, 2015, p. 48]

2.2.2 Learning methods

Supervised learning

In supervised learning, the system learns to predict an output value from an input value, but with a known desired output. Therefore supervised learning algorithms base on datasets that contain specific labeled features. Without the right dataset, it is not possible to train such a network. Eventually, the results depend on the quality of the dataset. These systems are trained by comparing the desired output to the output of the neural network. [Goodfellow et al., 2016, p. 105]

In unsupervised learning, there is no known desired output. Of a dataset containing many different features, useful structures are learned. Classification and denoising tasks often use this type of learning. However, it is currently not relevant for pose estimation and the systems used in this research, which is why I will not go into further details.

Overfitting

Overfitting occurs when a network performs well on training data but not *in-the-wild*. Since deep networks can approximate complex functions, overfitting occurs when the network approximates the noise of the training data as well as the relation between input and output. To prevent that, a defined number of neurons can be deactivated (dropout), causing a more general output. [Goodfellow et al., 2016, p. 110]

Furthermore, the quality of the dataset influences how well a network generalizes. E.g., for 3D-Pose Estimation, images of people with labeled 3D joint position are only available in controlled surroundings, in a MoCap-Studio or similar. Therefore a system only trained on images of people in a MoCap-Studio will not perform well with more general input. [Mehta et al., 2016]

Datasets for pose estimation

As already mentioned, the datasets are crucial for training neural networks. The following datasets are used for pose estimation. All of these contain images of humans, with position-labels of the main body joints. However there are some slight differences which parts are labeled. The values of a dataset are call *ground-truth-values*.

1. MPII: Contains about 25.000 images with 40.000 people. These images are single frames collected from youtube-videos, based on searches of different human activities. [Andriluka et al., 2014]
2. COCO: Contains images with about 250.000 people, besides other "common objects in context". The images were collected from flickr with regard to being "non-iconic", so not especially staged, but real-life examples. [Lin et al., 2014]
3. Human 3.6M: Consists of 3.6 million video frames of actors performing multiple life-like situations. The images are labeled with 3D keypoint locations, acquired with optical marker-based Mo-Cap in one specific surrounding. [Ionescu et al., 2014]
4. MPI-INF-3DHP: Contains 1.3 million video frames of actors captured with a markerless Mo-Cap system in front of green screen. To imply different surroundings, and better generalization, actors are separated and in integrated into life-like surroundings. [Mehta et al., 2016]

2.2.3 Training

Training is the key to getting the desired output of a neural network. Before training the weights are initialized with random values, and the output of the system is not significant. During training the weights are adjusted to get the desired output from a network.

Neural Networks are trained with a process called back-propagation. It consist out of three steps:

Forward-Pass: The input-data x is passed through the network, generating a specific output $a\,(x)$.

Calculate Error: The output is compared to the desired output, specified by the dataset (ground truth values) with a specific error-function C:

$$C = \frac{1}{2n}\sum\_\mathrm{x}\|y-a\|\hat{}2$$

Backward-Pass: The gradient of C in regard to each weight is calculated. Then the weight is moved by a defined learning-rate η towards the minimal error:

$$\omega^{new} = \omega^{old} - \eta\frac{\delta C}{\delta w}$$

This is repeated until the error converges towards the lowest value. (For all steps, see [Nielsen, 2015, p. 26 - 28])

2.2.4 Convolutional neural networks

Convolutional neural networks, are artificial neural networks, which base on the convolution of input data. Due to efficiency, these types of networks are currently the most common.

Convolution

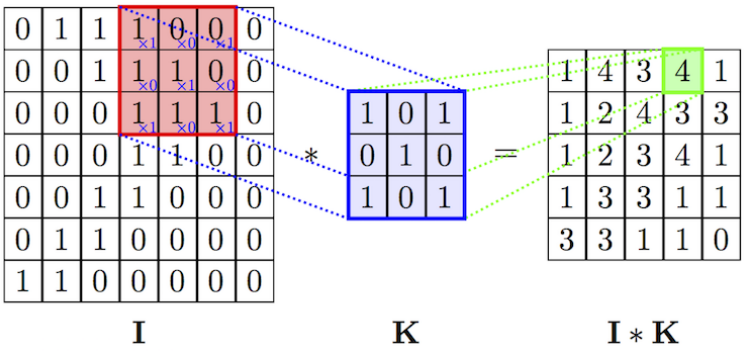


Figure 2.4: Convolution with a 3x3 kernel[5]

Convolution is the basic operation in any CNN. Given a 2D Image $\mathbf{I}$, and a filter kernel $\mathbf{K}$ with the size $(w * h)$, the convolution $\mathbf{I} * \mathbf{K}$ is the scalar product of the input and the filter coefficients. The filter is applied at any possible position of the image. The step-size defines how often the filter is applied. Larger step-sizes cause a smaller output. To control the size of the output, zeros are added around the border of the input. This is called zero-padding. [Goodfellow et al., 2016, p. 331]

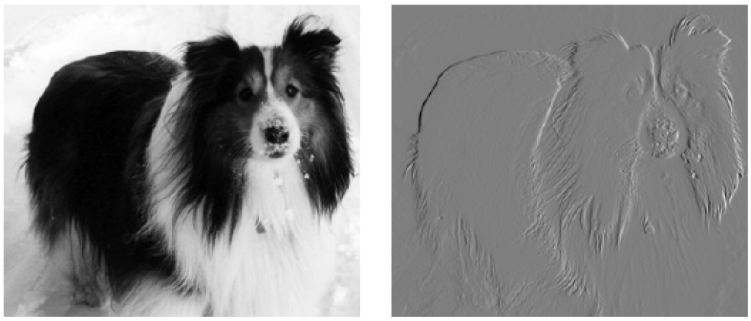


Figure 2.5: Output of a convolution.[6]This is the x-component of a sobel edge detector with the following kernel: $\begin{Bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{Bmatrix}$

[5]source: https://gasimof.com/tag/convolutional-neural-network/ - access: 2018-08-19
[6]source: [Goodfellow et al., 2016]

Convolutional layer

In a convolutional layer, convolution filters are applied to the input. The filter kernels are not defined beforehand, but are learned during training. Consequently, important features in the training dataset are extracted by the learned filters. The output of such a layer is the convoluted input, and called a feature map. [Goodfellow et al., 2016, p. 335]

The convolution operation has several benefits: In a CNN with a filter kernel that is smaller than the input, one input neuron would only contribute to specific neurons in the next layer. Pixels in specific parts will contribute to specific parts of the feature map. For one feature map, the same filter is applied on all neurons, contributing to the feature map. In contrast, in a regular NN every input neuron is connected with a specific weight to every neuron of the next layer. Every weight is used only once. [Goodfellow et al., 2016, p. 335]

E.g. given an input image with $72 * 72$ pixels a convolution with a $3 * 3$ kernel and a stepsize of 1, the output would be $70 * 70$ pixels big. The convolution would need $70 * 70 * 9 = 44100$ operations. However, in a regular NN, the same output through matrix multiplication would need $72 * 72 * 70 * 70 = 25401600$ operations.

So CNN's are more efficient than regular neural networks because less storage and fewer operations are needed. This makes the processing of bigger images possible and training with back-propagation faster.

Other Layers

Besides convolutional layers, these type of networks also contain so called pooling- and fully connected layers.

Pooling layers simplify the incoming data, in a way that small changes in the input do not change the result. This will also reduce the size of the incoming data. Common operations are max- or min-pooling, keeping only the highest- or lowest values of one specific area. [Goodfellow et al., 2016, p. 339, 342]

The fully connected layer defines the output of the network at the end of a CNN. The neurons are connected to every neuron in the previous layer and work like a regular NN

2.2.5 Pre-Trained networks

CNN's are not build from scratch for a specific use, but often base on already trained networks. The output of so called pre-trained networks can be used for further

processing, or a pre-trained network is refined for a specific use. Typically, networks for image classification are refined for human pose estimation. [MathWorks, 2018]

Pre-trained networks output satisfactory results on specific large-scale datasets. When refining such a network, it is given an unknown dataset with other specific features. Typically, the first layers and weights of the pre-trained network, to infer low-level features, are kept. On top of these layers, new layers for specific new features are added, and the whole network is trained on a new dataset. [MathWorks, 2018]

The systems used for this work, base on the pre-trained networks VGG-19 and ResNet-50. Both of these are trained on datasets of the ImageNet database and classify objects in images into one of 1000 categories. [Kanazawa et al., 2017], [Cao et al., 2016]

2.3 2D pose estimation

2.3.1 Functionality and available systems

2D pose estimation aims to localize anatomical keypoints in an image. This problem is usually treated as a supervised learning problem because large and diverse datasets with annotated keypoints exist. The estimated poses are either represented as 2D keypoint coordinates or by more complex representations. E.g., the DensePose system defines all pixels belonging to a specific body part [Güler et al., 2018]. This would offer a full rotoscoping alternative, as one can use the allocated pixels belonging to a human, as a matte. Moreover, neural networks trained for object detection and image segmentation can output mattes for a variety of objects in an image [Chen et al., 2017]. However, any inexact edge provided by such a complete solution would need to be redone entirely. So I decided to use keypoint locations as a fundament for the transformation of roto-shapes, which can be further refined by an artist.

With images containing multiple humans, the keypoints belonging to one human need to be associated with each other. This problem can be solved by running a person detector beforehand, and then estimate poses per person, but faulty person detection will result in faulty pose estimation. Additionally, this approach extends runtime per person. The OpenPose system proposed by Cao et al. efficiently and implicitly solves this problem by simultaneously predicting part-affinity-fields. [Cao et al., 2016]

The OpenPose system was further used for this research because it efficiently outputs keypoint locations for multiple persons. Moreover, it was chosen, because it has already implemented features to export the estimated keypoints, it is the winner of the 2016 COCO Keypoint Challenge and open source.

2.3.2 OpenPose

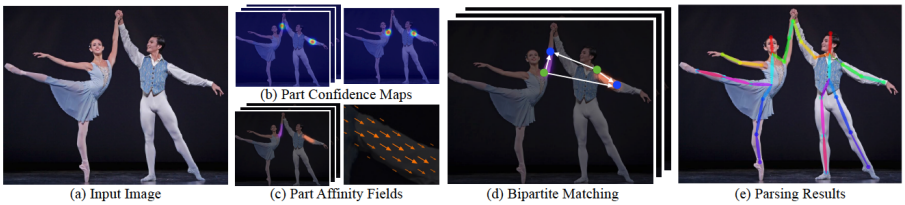


Figure 2.6: OpenPose pipeline[7]

OpenPose is a system for 2D human pose estimation of multiple people. The system works in real-time - on a gtx1080 at around 9fps. The system bases on two branches of a CNN. Given an input image, one branch predicts confidence maps of the keypoints, one for each part. The second branch simultaneously predicts *part affinity fields*, one for each pair of two keypoints. [Cao et al., 2016, p. 2]

The first branch is directly trained to predict keypoints with the ground truth data from the COCO and MPII human pose datasets. To keep close peaks distinct, the maximum of each confidence map is defined as the corresponding part. [Cao et al., 2016, p. 3]

When multiple humans are in the frame, parts can have multiple connections to each other. To output the pose for each human in the frame, the parts belonging together need to be associated with each other. E.g., one right shoulder could be connected to multiple predicted right elbows. OpenPose accesses this problem with a feature called part affinity fields (short PAF). Here the definition by Cao et al. :

> "The part affinity is a 2D vector field for each limb, also shown in Fig. 1d: for each pixel in the area belonging to a particular limb, a 2D vector encodes the direction that points from one part of the limb to the other. Each type of limb has a corresponding affinity field joining its two associated body parts." [Cao et al., 2016, p. 4]

So for each pixel belonging to a specific limb the direction of the limb is defined (See c) figure 2.6 on the previous page). The ground truth part affinity fields are defined by the position of the ground truth keypoints. [Cao et al., 2016, p. 4]

[7]source: [Cao et al., 2016]

Given the part candidates and the part affinity fields, every possible connection between two parts is scored in regard to the PAF. For that, the alignment of the limb, which contains the two part candidates, and the corresponding PAF is measured. Then the connections are sorted by its score. The connection with the highest score is valued as a final connection. If the next highest score does not contain any part of an already final connection, it is a final connection. Otherwise, it is rejected. This is repeated until all parts are connected. [Cao et al., 2016, p. 4]

Finally, the connected pairs need to be associated with the corresponding human. To achieve that, every pair is defined as one human. Now if two humans share the same part, they are defined as one human. This is repeated until there is no pair of humans sharing one part. [Cao et al., 2016, p. 5]

2.4 3D Pose Estimation

2.4.1 Key challenges

Estimating 3D poses from monocular RGB images includes additional difficulties. Out of an image, anatomical keypoints need to be localized, and reasonable positions of the keypoints in 3D space have to be inferred.

Handling 3D Pose Estimation as a supervised learning problem comes with limitations. When a CNN is trained on images with annotated 3D keypoints to output 3D estimates, the main problem is the lack of sufficient datasets. 3D keypoints cannot be labeled by hand, but can only be acquired in a MoCap-Studio or similar. So images with accurate labels are only available in controlled surroundings, and not in-the-wild. As a result, systems trained in such a way, do not generalize well. Metha et al. presented a synthetical dataset to counter this problem. It consists of multiple humans captured by a markerless MoCap-system and in front of a green screen. These subjects are integrated into general surroundings. This lacks realistic image qualities in terms of lighting and integration, but is closer to the variety of in-the-wild images. [Mehta et al., 2016]

Another significant problem with single image systems is the location of poses in space. With multiple image systems, the depth information can be calculated by triangulation, but a single image does not provide any depth information.

2.4.2 Functionality and available systems

To solve the problem of 3D pose estimation multiple approaches exist. First off, are approaches treating 3D pose estimation as a supervised learning problem, but as they will not generalize well (see the previous section), the use on any filmic plate or another general environment is not useful.

Other approaches consist out of two different steps. First, 2D keypoint positions are estimated. Subsequently the 2D positions are analyzed by another network, which is trained to estimate 3D positions out of the 2D positions. However multiple 3D-skeletons projected onto the image plane, can fit one 2D-skeleton. So assumptions about the human captured, need to be made beforehand. In this case, other image data besides the 2D joint positions is not used for the 3D estimation and the models need longer to train and compute results. [Kanazawa et al., 2017, p. 3]

Out of the publicly available systems, human mesh recovery (HMR) presented by Kanazawa et al., is one of the most promising ones in regard to VFX. The system implicitly predicts local joint rotations and a camera. So joint positions can be calculated relative to the camera, and joint rotations do not need to be separately extracted by an IK-Solver. The system will always output a complete skeleton and can even estimate a mesh, representing the human in the frame. Furthermore it is trained on unpaired 2D joint annotations and 3D skeleton data, therefore generalizing well. [Kanazawa et al., 2017]

In addition to HMR, the VNect System [Mehta et al., 2017] and RADiCAL Motion [Solutions, 2018] are promising. Both systems output a complete skeleton. Additionally to HMR, the skeleton moves in space. They show promising results in general surroundings. Moreover, the output of these systems can be used similar to known Mo-Cap systems. RADiCAL motion is a commercial product and its inner workings are not public, but movement in space is presumably acquired analogous to VNect, because the field of view of the used camera has to be set before processing: The estimated 3D keypoints are projected onto 2D keypoint locations. With the camera location that works best for this projection, the subjects position in space, relative to the camera can be infered.

VNect is not publicly accesible yet, so HMR and RADiCAL were practically used for this research. As the character design of CG characters is highly specific and developed early in production, it was chosen not to use the output mesh of HMR, but the underlying skeleton. The aim is to pose and animate any rigged character with the HMR output.

2.4.3 Human Mesh Recovery

Human Mesh Recovery (HMR) introduced by Kanazawa et al. is a framework for estimating full 3D meshes of humans.

The output mesh is represented by the SMPL model to describe human bodies. This model bases on a humans height, weight and body proportions as well as the pose. The pose representation of the model was further used (see Chapter 4). The pose is represented by joint rotations, that belong to a kinematic tree. [Loper et al., 2015]

Given images with annotated 2D joint positions, the network is trained to estimate the parameters of the mesh in a way, that the 3D joint positions match the annotated 2D joint positions after projection. This is achieved by iteratively regressing the reprojection error during training. If ground truth 3D data is available for an image, the network is additionally trained with direct supervision. [Kanazawa et al., 2017, p. 5]

Since certain combinations of joint rotations could fit the 2D joint positions after reprojection, but would not represent a realistic human body, a discriminator network checks if the output is a valid human pose. This network consists of one discriminator for each joint, trained to learn the angle limits for those. The discriminator network is trained on 3D meshes of human bodies, that do not necessarily need a corresponding image. [Kanazawa et al., 2017, p. 5]

2.4.4 RADiCAL Motion

RADiCAL Motion offers an application for complete pose estimation based Mo-Cap. One can capture and upload a video, the processing happens on their servers, and the results can directly be downloaded as an animated rig. The service is supposed to work in any environment and on videos captured by any device. [Solutions, 2018]

The cloud based approach comes with restrictions, as the service is not free to use and one has pay depending on the length of estimated videos. [Solutions, 2018]

3 Rotoscoping with Open Pose

Rotoscoping with OpenPose describes a possible workflow to use pose estimation as a fundament for rotoscoping. Pose estimation can be triggered on any plate, which requires rotoscoping of humans.The location of continually estimated keypoints is then used to position roto-shapes accordingly.

In this chapter, such a workflow is described, and it is further analyzed if this can benefit and simplify the process of rotoscoping while comparing it to the current best-practices.

3.1 Rotoscoping

Rotoscoping refers to tracing the movement of a captured subject. It is a basic technique to separate certain elements in a VFX-Shot. This is achieved by manually drawing the edges of the element to separate, most commonly with bezier-curves or b-splines, and thus generating an alpha matte of the element. To trace the subject, one has to manually adjust the control points of the shapes. [Okun and Zwerman, 2010, p. 570]

One of the key challenges in rotoscoping is to separate one or multiple moving actors. This process requires a lot of manual labour and can be very time consuming. To ensure accurate and consistent edges, while rotoscoping humans, the following best-practices have been established:

1. Using as few keyframes as possible.
2. Using multiple simple shapes, rather than one complex shape. For humans usually one shape per limb is used.
3. Adjusting shapes as a whole, rather than single points.

Additionally, planar tracking is often used to aid rotoscoping. In this case the movement of manually selected limbs and body parts are tracked over time. The roto-shapes are then be transformed based on the tracking-data, to match the movement of the tracked limb. The software Mocha by Boris FX is most often used for this purpose.

3.2 Used Software and Tools

The workflow uses the Foundry's Nuke, as it is the industry's standard compositing software. The gizmo itself bases on the integrated nodes and expressions, as well as Nuke's python API. It aims at the use with Nuke's standard roto nodes: *Roto* and *Rotopaint*.

With the respective roto nodes one can create and edit Bezier and B-spline shapes. During rotoscoping the principal work happens in the *Roto-Tab:* One can edit the shapes, set keyframes, and edit opacity and feathering. Furthermore, global transformation of the shapes can be controlled with the *Transform-Tab:* For every shape created, one can set, edit and animate translation, rotation, scaling and skew. This functionality was used to integrate OpenPose data into the rotoscoping workflow. [Foundry, 2018]

3.3 Workflow

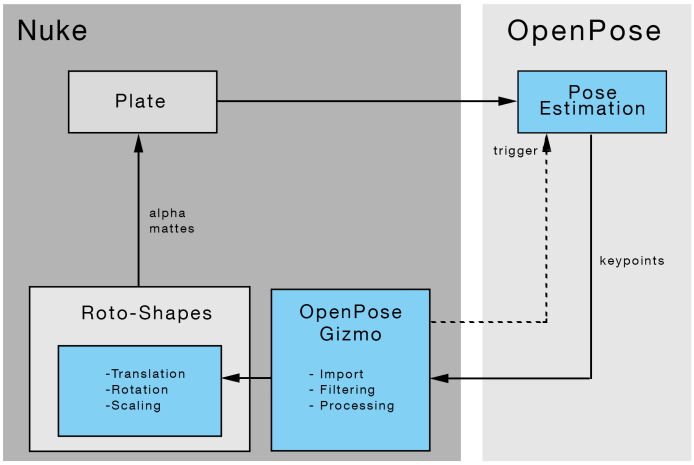


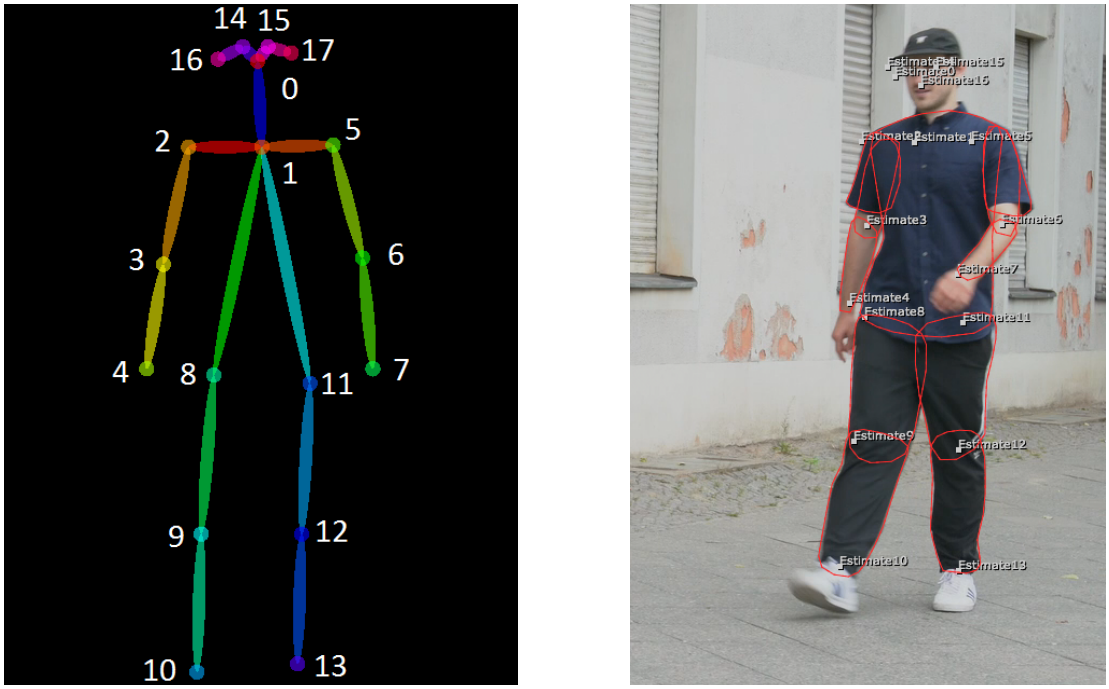
Figure 3.1: Workflow for rotoscoping with OpenPose

Figure 3.1 shows the general workflow for rotoscoping with OpenPose. Central part is a gizmo for Nuke, triggering pose estimation, and processing the pose data to be used with roto-shapes.

With this gizmo, edges are not automatically traced, but the basic motion of the roto-shapes is automatically acquired through pose estimation. As a consequence, no tracking of specific limbs is necessary. This is an alternative to tracking all limbs of an actor with mocha. It is important that the mentioned best practices (see previous section) for rotoscoping can be realized when working with this gizmo.

3.3.1 Pose Estimation

An artist can start pose estimation on the desired plate, directly from Nuke. This happens through a command line call, starting OpenPose for the chosen image sequence or video.



(a) OpenPose output[8]

(b) OpenPose estimates and corresponding rotos in nuke viewer

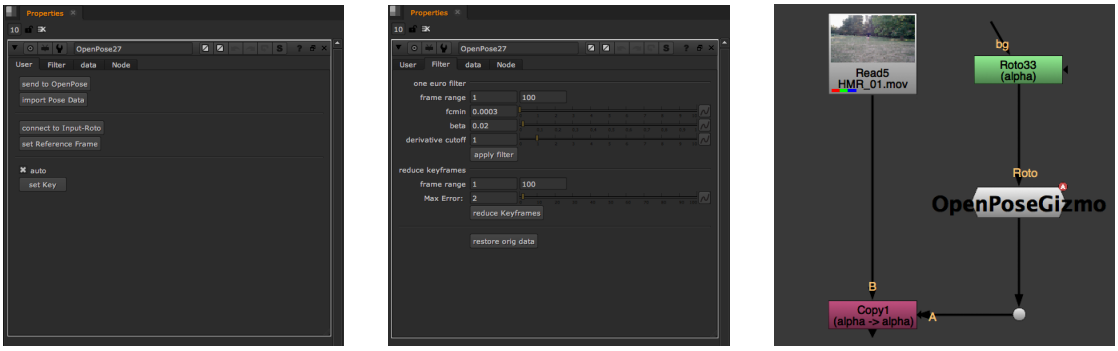
Figure 3.2: Keypoint order and corresponding roto-shapes

During pose estimation, OpenPose will output a JSON-File for every frame. The JSON-File contains the 2D-Location for each estimated joint. To distinguish the different joints, they are saved in a specific order (see a) figure 3.2). The output is normalized to [0,1] to avoid errors caused by different formats.

3.3.2 Import

After pose estimation, the results are imported into nuke. Inside the gizmo the pose data is stored in a node with translation-knobs, storing the (X, Y) coordinates for every keypoint. To import the estimated pose-data, the JSON module in python is used. The import simply sets keyframes on every frame for every successfully estimated keypoint.

[8]source: https://github.com/CMU-Perceptual-Computing-Lab/openpose - access: 2018-08-19



(a) main panel (b) filter options (c) node graph

Figure 3.3: User Interface of the OpenPose Gizmo

3.3.3 Connect Roto

The gizmo defines movement for roto-shapes, which cover specific body parts and limbs. For the limbs, movement is defined by the two keypoints at the limbs ends, whereas the torso moves according to both shoulder- and hip keypoints. The head moves according to the head- and neck estimates. Feet and hands are not defined, because they would only be represented by one keypoint, which does not work for automatic scaling and rotation. (see b) figure 3.2 on the previous page)

After import, an artist has to create roto-shapes, at least one for each body part and limb, and adjust them for a single reference frame in a way that fits the subject. Ideally, one chooses a neutral pose as a reference frame, to ensure appropriate scaling and placement of control points. It was chosen not to automatically create the roto-shapes, as a well-considered distribution of control points is crucial for a clean result. E.g., uneven edges require more control points, than straight ones.

To ensure that each shape moves along with the corresponding limb, a connection-panel was implemented. An artist has to connect the roto node to the gizmo's roto-input, and can then manually choose the name of a roto shape and set the matching body part or limb. To keep a clean and reasonable node graph, when a roto node is connected to the gizmo, all its channels will be forwarded through the gizmo and can be accessed at its output.

3.3.4 Transformation of Shapes

Based on the translation data of each keypoint, the transformation of each limb and bodypart and the corresponding roto-shapes is calculated. This is achieved with an expression linking the XY-knobs that contain the pose estimation data and the transform-knobs of the roto-shapes.

Given the two joint positions $J1 = (x_1, y_1)$ and $J2 = (x_2, y_2)$, and the corresponding differences $\Delta x = x_1 - x_2$ and $\Delta y = y_1 - y_2$ transformation, rotation and scaling of the limbs can be calculated with the following functions:

$$translation : t_x = (x_1 + x_2) * \frac{1}{2}, t_y = (y_1 + y_2) * \frac{1}{2}$$

$$rotation : \alpha = \arctan(\Delta x, \Delta y)$$

$$scaling : s = \sqrt{(\Delta x * \Delta x) + (\Delta y * \Delta y)}$$

When connecting the shapes to the pose estimates, reference values for this frame are set. This way the prepared shapes are not being transformed in the reference frame. Transformations for any other frames are calculated in regard to the reference frame:

$$translation : t_x = x_{cur} - x_{ref}, t_y = y_{cur} - y_{ref}$$

$$rotation : \alpha = \alpha_t - \alpha_{ref}$$

$$scaling : s = s_{cur}/s_{ref}$$

During rotoscoping, the artist can grab the pose estimates, which will directly affect the transformation of the shapes (see figure 3.4 on the following page). This is meant for the clean up of specific estimates over a certain frame range.

However when tracing the exact edges, it can and should be done directly on the roto-shape. The gizmo only affects the transform-knobs, hence keyframes of the shape can be set separately and on top of the provided animation.

3.3.5 Temporal Filtering

Temporal jitter in the pose estimation data, which results in inconsistent edges, is one major problem when rotoscoping with the help of pose estimation (more on this in section 3.3). To reduce this problem, the animation curves for each keypoint can be filtered.

(a) no adjustment (b) manual adjustment

Figure 3.4: Manual Adjustment of Estimates

Nuke offers an already implemented smoothing function for animation curves. This is based on box filtering of the curve, so the filtered value of one keyframe is the average value of its neighboring frames [Rueter, 2010]. This filter can reduce temporal jittering, but the results are not satisfying. If the jitter is reduced adequately, local maxima and minima of the animation curves are averaged out, which introduces temporal latency.

Curve Simplification

The user can apply the curve simplification algorithm by Onder et al., which was presented to specifically clean up MoCap data by reducing keyframes. Any keyframes over a certain threshold are not affected, while keyframes below a certain threshold are deleted, consequently smoothing the curve and reducing jittering.

1. Find start and end frame of the animation and set a keyframe (see b) figure 3.5 on the next page)
2. Find the frame with the biggest error value between original animation, and the new curve. Then set a keyframe with original values. (see c) figure 3.5 on the following page)
3. Repeat the process for two new segments, between first and middle keyframe, and between middle and last keyframe (see d) figure 3.5 on the next page)
4. Repeat step 3 for every new segment until the highest error frame is lower than a defined threshold.

The implementation in python is bases on the script provided by Richard Frazer and was adapted for the gizmo. [Frazer, 2015]

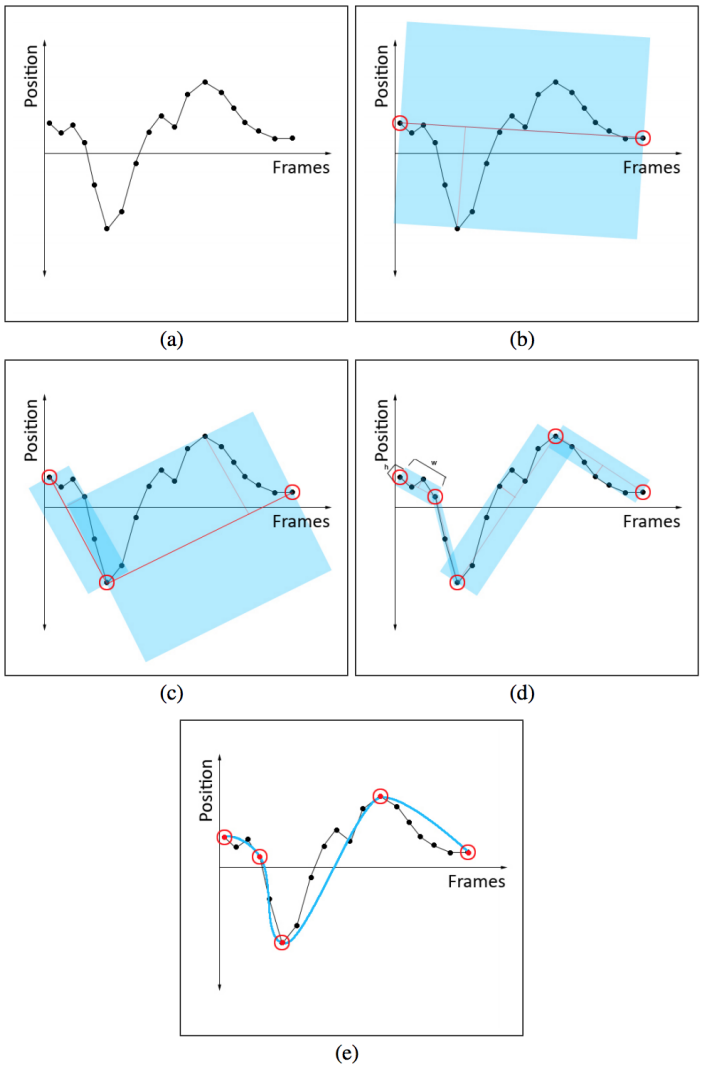


Figure 3.5: Graphical explanation of the Curve Simplification Algorithm[9]

1€ Filter

Besides the curve simplification algorithm, jittering can be reduced with the *1€ Filter* [Casiez et al., 2012]. This is a speed-dependent low-pass filter, which requires minimal resources.

The filter aims at the general problem that temporal latency is more obvious for fast movements, while jitter is more obvious for slow movements.

[9]source: [Onder et al., 2008]

As temporal jitter is high frequency, but human movements have lower frequencies, the signal is filtered with a low-pass. Any signal above the cutoff frequency f_c is weakened, whilst other signals are not affected. A low cutoff frequency will result in smoother motion but cause temporal latency. To avoid this f_c increases when the speed increases. $f_{c_{min}}$ is the lowest possible cutoff frequency, defined by the user. $\hat{X}$ is the derivative of the signal, in other words: The speed of the movements. [Casiez et al., 2012]

$$f_c = f_{c_{min}} + \beta \left| \hat{X} \right|$$

The user can decrease $f_{c_{min}}$, until jittering is minimized suitably. Then β can be increased to reduce temporal latency.

3.3.6 Workflow Considerations

The following details need to be considered to integrate the gizmo into a studios compositing pipeline. While compositing usually happens on 16Bit EXR- or DPX-Sequences, OpenPose currently cannot decode these files. However Jpeg's or png's can be analyzed, so it is necessary to export a sequence of either of those, with low compression and in full resolution. This could happen automatically when starting pose estimation from within Nuke, or it could happen beforehand, during the export of plates, as such a sequence is often used for match-moving as well.

Furthermore, the directory for the OpenPose output needs to be specified. Locating it inside the compositing or roto folder of the specific shot is plausible, as an artist could check the existence and the completeness of the JSON files for troubleshooting. This directory could be automatically generated, according to the pipeline specific shot data, when calling OpenPose.

3.4 Evaluation

To analyze the usability of rotoscoping with OpenPose, the workflow was carried out for several test shots. These tests included different types of an actors movement, multiple camera movements and a variety of lighting situations. The shots were done in front of different non homogenous backgrounds. Any filmic plate, where rotoscoping is necessary, would neither contain a homogenous background. All plates were acquired with a standard 180° shutter and 25 fps, to match temporal resolution of filmic plates.

To test the speed of the proposed workflow for rotoscoping, edges of the roto-shapes belonging to specific limbs were refined. For a variety of shots, a shape with a varying edge, due to clothing for example, and a shape with a clean edge, e.g., a plain limb, was refined. This was compared to the speed when rotoscoping based on planar tracking. The shapes were refined for 50 to 70 frames, depending on the shot.

(a) unusual pose (b) occluded limb (c) cropped input

Figure 3.6: Images relevant for the evaluation

3.4.1 General Performance

OpenPose works well, when checking individual frames. Out of the 2945 frames analyzed with OpenPose, only around 5% were visually mismatched. The mismatched frames needed cleanup to be used for rotoscoping, due to failure cases described in the next section.

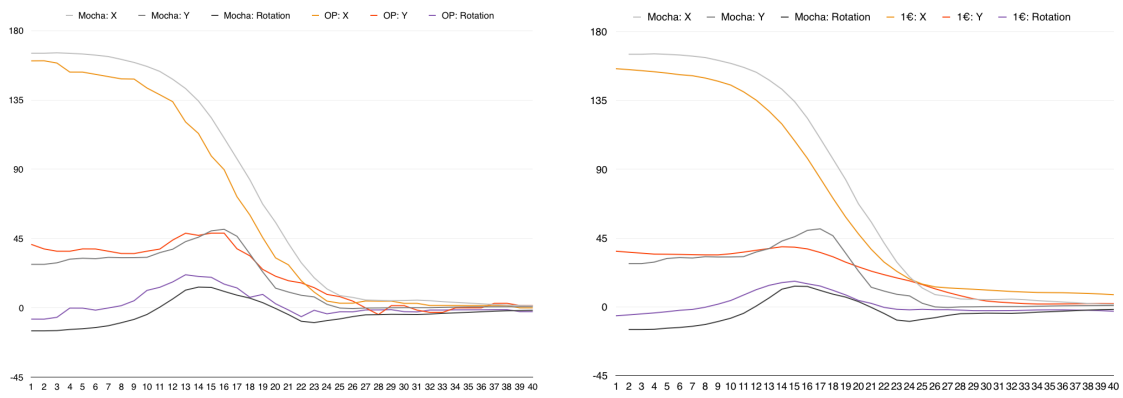


Figure 3.7: Comparison of animation curves of mocha, original OpenPose Data and after 1€ filtering.

Besides visually correct location, the movement and temporal consistency are relevant, when driving roto-shapes with the results of pose estimation. While reviewing a series of frames individually, the poses and joints seem to be correctly located. However, when reviewing continuous frames, temporal jittering is a significant issue. This was the case for every testshot analyzed, but it is less evident for fast and complex motion. One can clearly see the jitter in the animation curves (see figure 3.7). OpenPose estimates human poses for every single frame individually, and no temporal filter is applied. Thus slight inaccuracies of the predicted joint positions will lead to temporal jitter. One has to note, that even datasets contain some noise, as a visual label of an anatomical keypoint cannot always be at the same position for different images, due to differing perspectives, human shapes, and visibility.

Regarding rotoscoping temporal jitter is an unacceptable artifact. This would require animation for almost every frame to counter the jittering and keep a constant edge. When using planar tracking in contrast, an explicitly defined set of pixels can be tracked accurately over time, and relative to a reference frame. This can be observed when comparing the animation curves for both methods (see figure 3.8 on the next page).

To counter the temporal jittering the two filters, described in section 3.2.5 were implemented. The results of the 1€ filter are generally more accurate, if $f_{c_{min}}$ and β are chosen reasonably.

3.4.2 Failure Cases

Cao et al. describe multiple cases, in which pose estimation for an individual frame fails [Cao et al., 2016]. For these cases I observed the following in regard to

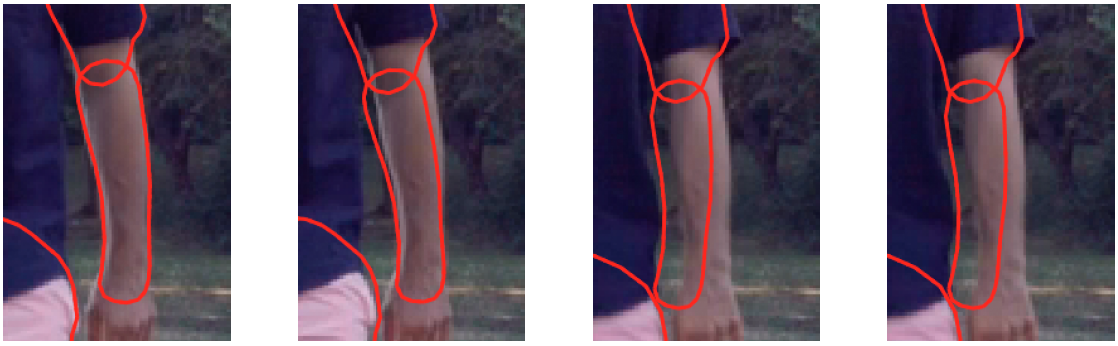


Figure 3.8: Jittering roto shape that would require counteranimation

rotoscoping:

Unusual poses: Unusual poses fail because they are rarely represented in the dataset. Hence the network is not trained for these poses. In this case, most of the main joints are not correctly located, and the data cannot be used for rotoscoping at all. (see a) figure 3.6 on page 26)

Occluded limbs: OpenPose fails on occluded keypoints. Estimates of the occluded body part are often predicted at a similar location, as the corresponding body part. E.g., with an occluded right elbow, the right elbow is estimated at the position of the left elbow. The rest of the skeleton is not influenced. Occluded body parts do not matter for roto, because they are not visible. Moreover, false predictions are highly visible in the animation curves of the estimates. As a result these moments can be quickly cleaned up by deleting the according keyframes. (see b) figure 3.6 on page 26)

Overlapping persons: In this case, the keypoints are correctly detected, but the prediction of PAF's is erroneous and keypoints are not correctly associated to a body part and a human. For the frames with overlapping persons, the data would need to be manually cleaned up.

Occlusions and overlapping person also disrupt planar tracking. A unusual pose itself would not be an issue for planar tracking, since any defined pixels can be tracked. Moreover, when two people overlap, at least the person in the foreground could be tracked consistently.

Cao et al. also mentioned false predictions on humanoid statues and similar. However this does not influence the successful pose estimation of other humans in the frame, and therefore is not relevant for rotoscoping.

	OpenPose	Mocha
prep	4:00	15:00
walking/lower arm	11:30	10:25
walking/lower leg	19:00	17:40
walking/upper leg	14:00	12:45
prep	4:00	6:00
gesturing/lower arm	6:45	5:40
gesturing/upper arm	5:40	3:50
gesturing/torso	6:00	4:45

(a) slow movements

	OpenPose	-
prep	4:00	0:00
running/lower leg	13:30	16:00
running/upper leg	8:10	11:30
running/torso	12:00	14:00
prep	5:30	0:00
dancing/lower arm	15:00	17:00
dancing/upper arm	12:40	14:30
dancing/torso	9:00	11:00

(b) fast movements

Table 3.1: Spent time for the adjustment of roto shapes (in minutes)

3.4.3 Slow Movements

To test regular movements, the subject was filmed from several perspectives and in front of multiple backgrounds, while walking or standing still and gesturing.

In these tests, jittering was vey obvious, especially when the actor was momentarily standing still. When the actor is constantly moving, the results of the 1€ filter are satisfying. But in moments of minimal movement, e.g., when stopping after walking, jittering remains. In these situations one can use the curve simplification algorithm or manually delete according keyframes.

As a consequence, especially minimal changes in movements are not estimated well. With minimal movement, the signal to noise ratio is very low. An issue that still remains after filtering. Mocha, on the other hand, captures these movements pretty well.

According to that, refinement of roto-shapes on top of OpenPose took longer than on top of Mocha-tracks: On average about 12%. This does not count in the time spent mocha tracking. For *easy-to-track* shots, without occluded limbs and high contrast, mocha tracking took less than 12% of the time and thus outperforms the OpenPose-workflow in terms of speed. (See table 3.1)

For movements, that require extensive manual adjustment of the mocha-tracks, around 25% of the time adjusting was additionally spent on tracking. The time spent for the OpenPose workflow only depends on the length of the shot. During testing this was around 10% of the time spent for refinement. Although adjustment take longer on top of OpenPose, even more time is spent during preparation. (See table 3.1)

3.4.4 Complex Movements

(a) succesful pose estimation with high motion blur (b) mocha reference frame (c) mocha failure

Figure 3.9: Performance in regard to motion blur

Complex movements were tested with the subject running, rotating around his axis, jumping, and changing direction.

Pose estimation on fast movements did work out well, since the jittering was less noticeable and it worked considerably well on frames with a lot of motion blur.

With more movement of the limbs, slight inaccuracies over time have less influence. Essentially, the signal to noise ratio is higher, and jittering is less obvious, but still needs to be reduced. In these situations, the curve simplification algorithm led to distinct inaccuracies, while the results of the 1€ filter were satisfying.

Complex movements often cause the occlusion of certain keypoints, which can result in false estimation of those and the need for cleanup. However, as frames are analyzed individually, this will not interrupt continuous estimation. Planar tracking complex movements and rotations is problematic and time consuming.

During testing, planar tracking was not a reasonable workflow, because stable results would have needed roughly as much refinement and manual input, as rotoscoping itself. Hence, the OpenPose workflow was compared to rotoscoping without a fundament. In these cases, the OpenPose workflow outperformed adjustments without foundation by 5%. One saves time, because the shapes do not have to be translated as a whole, but has to spend time for the preparations with OpenPose. Therefore, the difference is presumably bigger for longer frame ranges.

3.4.5 Visibility

Shot 1 - original contrast	5,3%
Shot 1 - low contrast	6,0%
Shot 2 - original contrast	10,0%
Shot 2 - low contrast	12,5%
Shot 3 - original contrast	4,1 %
Shot 3 - low contrast	4,9%

Table 3.2: Average difference in regard to keyed matte

One assumes intuitively that performance of pose estimation and the proposed workflow improves with the visibility of the subject, more precisely with the contrast between subject and background. To test this assumption, the contrast of shots in front of a homogenous background was reduced to 25%. The regular- and low-contrast version were put through pose estimation, and the difference between a matte gained by keying and the automated roto mattes were compared. (See table 3.2)

The results of this test clearly indicate that performance deteriorates with lower contrast. The low-contrast mattes cover a greater area than the regular roto-shapes. It can be concluded: With decreasing contrast, jittering increases and accuracy decreases. This can be further observed on low-key plates, as pose estimation fails for for very dark body parts.

3.4.6 Performance with changing Surroundings

The workflow was tested in regard to the movements of humans in the frame (see chapters above), but also in regard to changing surroundings. This includes different forms of camera movement and changing lighting on the subject.

Because of the fact that OpenPose analyzes each frame individually, temporal changes from on frame to another, do not have any effect on pose estimation. As a consequence, neither camera movements itself nor the changing lighting directly influence this workflow. According observations could be made on the test-set. (See figure 3.10)

However, camera movement can lead to change of perspective on the subject, causing occlusions of body parts, which can deteriorate the output of pose estimation (See

Figure 3.10: Consistent fundament for roto, despite significant lighting change

section 3.3.2). Changing lighting can cause parts of the image to become dark or loose contrast, which influences the output as well (See section 3.3.5).

Consistent planar tracking with changing surroundings is often problematic. Momentary motion blur or changing lighting can result in pixel values that differ significantly from the ones used as reference for tracking. E.g., for the handheld shots I tested I was not able to get a consistent track at all. In this case one can either do not use mocha as a fundament for rotoscoping at all, or spend significant time manually adjusting planar tracks.

Principally, the datasets are the reference for pose estimation, while for planar tracking only very specific features in a a small number of specific frames serve as a reference. This shows one of the definite strengths of such a workflow: Pose estimation works regardless of temporal changes and one does not manually need to define image features for reference.

3.4.7 Cropped Input

Plates that require rotoscoping can be framed in all kind of ways, so rotoscoping with pose estimation can be necessary for plates not showing the subject as a whole.

OpenPose still works given these inputs. (see c) figure 3.6 on page 26) To test the corresponding performance, wide shots of the test-set were cropped and a cropped- and original version was put through pose estimation. Keypoints visible in both versions differed with an average of around 0.3%. Consequently, these keypoint locations can be used similarly. Yet, one has to consider, that keypoints not in

the frame are either predicted not at all, or faulty, which can affect body parts in the frame. E.g., if a plate does not include the knee, but parts of the upper leg, OpenPose cannot provide any fundament for masking the upper leg.

While it is possible to use OpenPose for single limbs, it might be too much in such cases. The proposed workflow can provides basic movement for the whole body at once, but for few shapes planar tracking can be even quicker.

3.4.8 Resume

First and foremost, temporal filtering is essential to work with the proposed workflow. The filtered results are not a general solution, but work well in specific situations.

As mocha delivers a more accurate fundament for rotoscoping, the proposed workflow is not useful for shots that can be quickly and accurately tracked with mocha.

The accurate analysis of the time spent for both workflows, would require long time testing with multiple artists as it heavily depends on varying performance of the artist. However, one can definitely conclude that pose estimation works for shots, that cannot be consistently planar tracked. Moreover, one can quickly test if the tool works well for individual shots and if not, one can still tackle a shot with another technique.

So all in all, rotoscoping with pose estimation is less accurate, but it can work for *hard-to-track* shots.

3.4.9 Further Considerations

The development of the gizmo led to the following considerations to further progress pose estimation based rotoscoping:

Implement new pose representation

OpenPose was recently updated with the "BODY25" pose representation, which includes keypoints at the end of the feet [10]. Right now the corresponding roto-shapes are only translated according to the OpenPose output, with this they could be oriented as well. The representation also includes a central hip keypoint, which can be additionally used for the transformation of the torso. Besides that, one can use the available feature of body and hand estimation to automatically orient shapes of the hand.

Automatically Trigger Pose Estimation

Since pose estimation itself does not need any user input, OpenPose could be triggered automatically. E.g., one would specify in a VFX editoral tool, if rotoscoping of humans is necessary for a specific shot. And as soon as the corresponding plate is exported and ready for production, OpenPose will be triggered and export the matching pose data as JSON-files. The JSON-Files only need a small amount of storage (around 1KB per frame), and it would save time later in production, as an artist just needs to import the data.

Improve User Interface

Secondly, an improved UI would presumably benefit usability and speed when working with the tool. Right now, the connection of roto-shapes to the corresponding limb is rather unintuitive and leaves room for improvement. This could be improved by displaying a rig similar to the OpenPose-Output in the viewer. Ideally the artist could then just select the parts of the rig and the corresponding shapes, and connect both.

Hybrid Approach

To combine the benefits of pose estimation and planar tracking, a hybrid between these two approaches may be worth testing. This could counter the lack of accuracy

[10] https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md

in pose estimation, while simplifying continuos tracking for challenging situations.

A basic concept could be, to calculate translation, rotation and scaling - similar to the realized approach - for each shape in mocha, based upon pose data. As soon as tracking terminates, or the shapes move away from the set reference the artist could automatically realign the shapes with the human. Setting a keyframe, without manually- but automatically moving the shape. Alternatively, the pose data could dynamically define a search area for each tracking shape.

Adaptions of the Network Output

Networks that output a complete matte cannot serve as a basis for rotoscoping (see section 2.3.1). However, if such a systems would represent the pixels belonging to a specific limb as a bezier curve or b-spline, the output could be further adjusted. The systems would need to be trained on estimating poses by infering control points of a bezier curve or b-spline. This would enable animation on top of the output matte. One could keep only certain keyframes and adjust the shapes accordingly.

4 Using 3D Pose Estimation

In this chapter, a workflow to use the data of 3D pose estimation in a 3D-application is described and evaluated. More precisely, how to automatically pose and animate humanlike CG characters by transferring the HPE output onto a character's rig.

The systems Human Mesh Recovery and Radical Motion are used for this workflow. While the output of RADiCAL Motion is ready-to-use, HMR is not explicitly designed for such a purpose, and the estimated poses need further processing. However, the integration of such a system is compelling, as it offers flexibility and can guide the integration of future systems.

4.1 Used Software and Tools

Autodesk Maya - the industry standard 3D application for animation - is the fundament for this workflow. More specifically, its rigging tools and python API, and the HumanIK (HIK) system.

HumanIK is an animation middleware, for IK solving and retargeting (see next section) full bodies. HIK provides tools for automatic and refined retargeting between structurally different characters [Autodesk, 2018]. Thus, it is commonly used for retargeting Mo-Cap data onto characters.

To control and pose a character in maya, one has to create a skeleton, which consists out of a chain of joints. These joints form a kinematic tree (see section 2.1). The joints are bound to the mesh, thus deforming it after specific principles. To use HIK such a skeleton needs to contain specific joints. The defined joints then need to be characterized according to the HIK definition: One has to define which joint represents the characters corresponding limb or body part. [Autodesk, 2018]

Figure 4.1: HIK

After definition, one can use the HIK features to retarget animation from one character to another. Moreover, one can automatically create IK- and FK-controls to adjust the skeleton. [Autodesk, 2018]

4.2 Mo-Cap Workflow

Using pose estimation to automatically pose and animate characters is principally analogous to working with Mo-Cap data: Captured joint locations are transferred onto a characters skeleton. Consequently, this workflow builds a fundament for using pose estimation data.

After solving, which is the process of transforming optical data into a skeleton, by calculating the root position and respective joint angles, a Mo-Cap solution will output an animated skeleton as an FBX-File or similar. [Okun and Zwerman, 2010, p. 345]

The animation of the Mo-Cap skeleton is applied onto another one in a step called retargeting: The root positions and joint rotations are transferred onto the corresponding joints. However, with anatomical divergencies, movements possible with one skeleton can lead to unwanted behavior within another character. E.g, mesh penetrations, or an unwanted change in the animations visual appearance. To avoid issues during retargeting, the proportions of the captured actor should match the final character closely. [Okun and Zwerman, 2010, p. 345] With HIK one can retarget Mo-Cap data onto the desired character and bake the animation onto its skeleton and control rig.

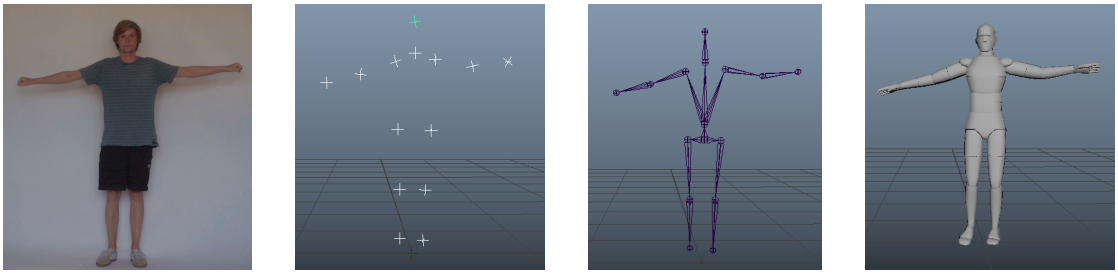
After retargeting, the character moves according to the Mo-Cap output, but animation generally needs further keyframe adjustments to clean up faulty tracks and to enhance or alter a performance. According to the Mo-Cap data, the skeleton or control rig have keys for every frame. Working and altering this is tedious, so animation layers are commonly used to work on top of Mo-Cap. For a defined set of controls, additional animation layers can be defined. First off, such a layer does not contain any keyframe. One can then set keyframes in these layers, which either add to the base animation or completely replace it. The animation on top of Mo-Cap needs to keep essential qualities, like realism and subtle details, gained by Mo-Cap. [Yates, 2016]

4.3 HMR Workflow

Human Mesh Recovery works on a single RGB image or image sequence and outputs 3D-poses. If its desirable to recreate the poses and motion of the captured human digitally, it can be executed on any video, respectively image. This can be a plate or a specific animation reference.

The workflow to use HMR, bases on the python script by R. Joosten [Joosten, 2018b], which I further adjusted. With this, an HIK-skeleton will appropriately controlled and animated, accordingly to the raw keypoint locations.

4.3.1 Pose Estimation and Import



(a) RGB input (b) HMR output (c) HIK skeleton (d) HIK character

Figure 4.2: Processing of HMR output

The trigger of pose estimation and the import of the pose data is similar to the OpenPose Gizmo. HMR is triggered through a command line call. With the adaptions of the demo by R. Joosten [Joosten, 2018a], pose estimation can be automatically looped over a set of frames, with an export of the keypoint 3D-coordinates and joint rotations as a JSON file.

The exported JSON file can then be imported by the artist with a simple import dialogue. Next, locators for every keypoint defined by HMR will be automatically created, placed at the estimated coordinates, and animated over time.

4.3.2 Skeleton Binding

The HMR-workflow bases on retargeting Mo-Cap data with HIK: A HIK-skeleton is created and animated according to positional information of the estimated skeleton. This is principally achieved by constraining joints to locators, carrying the pose data.

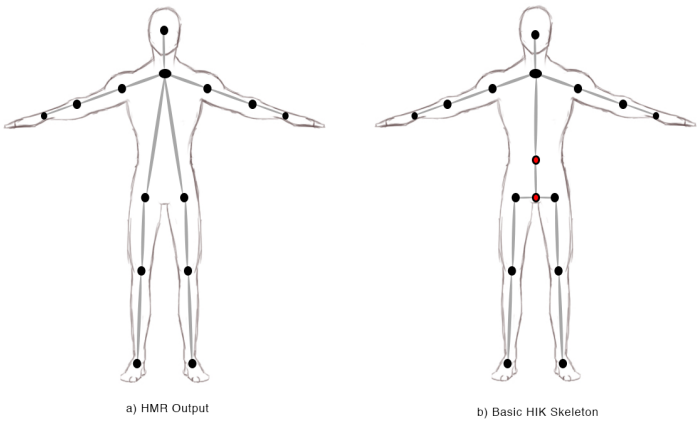


Figure 4.3: Differences between the HMR output and a basic HIK skeleton

HMR works internally with the SMPL-skeleton, which differs from a standard HIK-skeleton (See figure 4..3). Thus, the position of the missing joints is gained relative to existing joints: The SMPL skeleton does not contain a center hip joint, so its position P_{center} is defined by the position of the left- and right hip P_l, P_r. More precisely, in the middle of both: $P_{center} = 0.5P_l + 0.5P_r$. Likewise, a spine joint is necessary for HIK, but the SMPL skeleton does not contain one. its position is defined by the position of the hip- and neck joint. Between the two, but closer to the hip: .... All joints of the limbs, as well as the head- and neck joints, are defined in both skeletons and can be positioned exactly at the predicted position in 3D space. Respectively, joints are point constrained to the corresponding locators without offset. [Joosten, 2018b]

To ensure correct rotation of the missing joints, each of those is aim constrained towards the hierarchical next joint. This way the local Y-Axis of those joints points to their children. Otherwise retargeting, which bases on joint rotations, will not work. The rotation axis should fit the structure of the kinematic chain. If not, values between all rotation axes are interpolated with further animation, which results in uneven rotations [Hewitt, 2013]. Since these joints are part of a kinematic tree, and rotations are relative to each other, rotations of the left- and right hip need to be recalculated as well. [Joosten, 2018b]

To adopt the estimated orientation of the head, the neck-joint is aim constrained towards the head-joint, with the nose as an *up-object.* Because of that, the Y-Axis of the neck joint will always point towards the head, while the Z-Axis is turned towards the nose. The head will turns in the direction of the estimated nose.

Finally animation on every joint is baked and the locators and constraints are deleted.

Limited by the output of HMR, this skeleton is a very basic humanoid skeleton, but uses all the information estimated. With HIK it can be retargeted onto more complex skeletons.

4.3.3 Position in Space

To ensure consistent and appropriate placement of the character in world space, poses for every frame are positioned at the origin. More precisely, the root, defined by estimates of the left- and right hip, is placed at the origin. To have the character standing on the ground plane, the root is translated along the Y-Axis by the distance between hip and ankle.

With moving joints, animated by continuously estimated poses, it is desirable that the character itself does also move in space. HMR implicitly predicts a camera position, to minimize the reprojection error between the 3D- and the 2D keypoints [Kanazawa et al., 2017]. It was tested if the translation of the predicted camera can be used to provide sufficient movements in space. If an analyzed plate is shot with a locked-off camera, the position in space relative to the camera, can be acquired. However, HMR uses the simplified weak-perspective camera model [Kanazawa et al., 2017], and the predicted depth is too inaccurate for that purpose. Even for a simple forward-motion, this approach caused very inconsistent and thus unacceptable motion.

As a result, only a character's pose, but not the position in space, can be acquired with this system. E.g., a character performs a walking motion, without actually moving forward. The position in space can be manually animated with a handle transforming the whole skeleton.

4.3.4 Temporal Filtering

The two methods described in section 3.2.5 (1€-Filter and curve simplification) were tested for the use with HMR as well. The curve simplification algorithm did produce unsatisfying results. Even a low error threshold resulted in impossible human poses and mesh penetrations. Whereas the 1€ filter did produce better results.

Since translation and rotation for each joint of a skeleton is relative to the corresponding parent, filtering is applied onto the raw pose estimation locations. This way, the noisy signal of single keypoints can be separately filtered.

4.4 RADiCAL Motion - Workflow

Working with the output of RADiCAL is straightforward: The output skeleton is downloaded as an FBX-file and directly imported into Maya.

The skeleton contains all joints needed for a HIK character and the joint names match the HIK character definition. Thus, it can be manually or automatically characterized as an HIK skeleton. Before characterization the skeleton does need to be in a T-Pose with joint rotations at zeroed out. From then on, the animation can be retargeted onto any other HIK character and be further processed with standard workflows (see section 4.2).

4.5 Evaluation

The use of 3D pose estimation, was tested analogous to the use of 2D pose estimation: the capture and representation of poses and different types of movement in multiple surroundings. For this, both of the previously described workflows were executed.

3D pose estimation was evaluated on a test-set, which consists of monocular videos with an actor performing fast and complex motions (Fighting, dancing and running) and slow linear movements (walking and gesturing while standing still). These movements were captured in front of a homogenous white background and inhomogeneous land- and cityscape. Shots were principally captured as a long shot, with 25 fps and a 180° shutter. To test higher temporal resolution, movements were additionally recorded with 50 and 100 fps.

A benchmark for the same movements was acquired with a Microsoft Kinect, as it is a currently common Mo-Cap technique with a similarly small setup and without markers.

4.5.1 General Performance

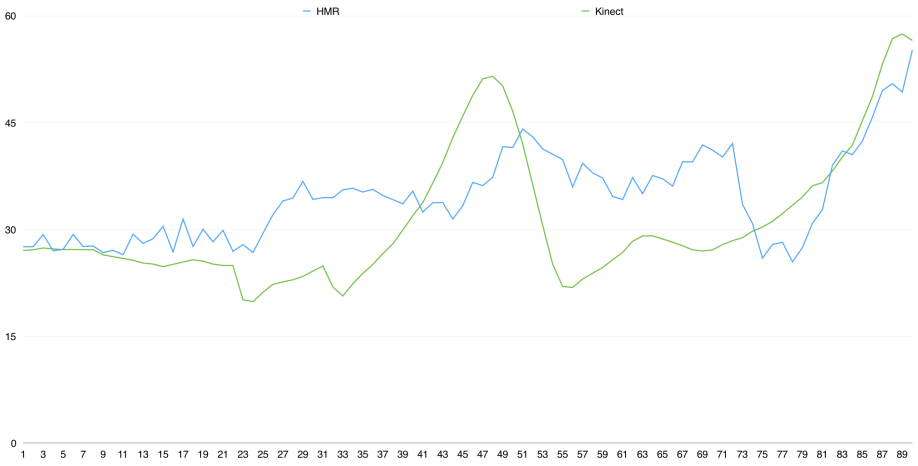


Figure 4.4: Animation curves of HMR and Kinect

Most of the time, individual poses estimated by both systems and the corresponding input frames are visually coherent. The test-set consisted of 2055 frames, and only around 3% of the poses estimated by HMR had one or more clearly mismatched limbs. It has to be taken into account that visual coherence and comparison to ground truth values can diverge: Kanazawa et al. describe that even poses with a high error are still visually reasonable [Kanazawa et al., 2017, p.7].

In regard to coherently predicted poses, one should note that a discriminator network controls all poses predicted by HMR, which ensures probable and biomechanically

correct poses. Contrary to that, Mo-Cap with a Kinect can produce unacceptable poses.

For character animation not only coherent prediction of individual frames is critical, but eventually the visual quality of continuous movements. The output of HMR shows similar problems as 2D pose estimation: Due to poses being individually estimated per frame, slight deviations between frames result in temporal jitter. This can be controlled to some extent by the 1€ filter but remains a problem when the actor is standing still or moving. One can clearly observe the jitter, when comparing the animation curves of both systems (See figure 4.4 on the previous page).

Moreover, on account of the high noise in the signal, subtle movements are not captured well. The estimated movements of HMR and RADiCAL clearly lack details, compared to the output of the Kinect. This can be especially observed for slow movements. Respectively, fast and distinct movements work better. On account of the high noise, higher frame rates do not produce remarkable differences. As Mo-Cap is generally used to realize life-like performances, which feed off subtle details, the lack of such details seriously reduces quality [Okun and Zwerman, 2010, p. 368].

When comparing the output of RADiCAL to HMR, the movements lack even more detail. It did produce more erroneous frames and the output of RADiCAL seems very mushy in comparison to HMR. The mushy quality of movements is most likely due to strong temporal filtering applied to the results, which cannot be manually adjusted.

Besides subtle details, pose estimation does not work well for movements in space. While poses estimated with HMR do not move in space at all, the output of RADiCAL suffers foot sliding and lacks accurate placement in space, compared to Mo-Cap with a Kinect.

4.5.2 Failure Cases

The main failure cases described in section 3.3.2 can be observed for 3D pose estimation as well: First off, unusual poses that are underrepresented in the dataset are likely to completely fail. And secondly, occluded body parts are often estimated at a similar position as the respective counterpart.

Besides problems analog to rotoscoping with OpenPose, both systems are not capable of estimating multiple persons at the same time. A workaround would be to separate both persons in the frame and run pose estimation independently on

	Frames	Erroneous Frames	
HMR/inhomogenous BG	1262	37	3.6%
HMR/homogenous BG	793	19	2.1%
RADiCAL/inhomogenous BG	1329	71	5.3%
RADiCAL/homogenous BG	585	30	5.1%
HMR Total	2055	58	2.8%
RADiCAL Total	1914	101	5.3%

Table 4.1: Number of significant mispredictions with 3D pose estimation

both. Yet, this will not work for close subjects, and neither for the interaction between two persons.

Another problem was observed, when using HMR for wide shots. The system expects input images to be tightly cropped around the subject. Nevertheless, HMR can compute the correct bounding box, if given the OpenPose output for the input frames, and thus can run on images that are not cropped accordingly [Kanazawa et al., 2017]. However, if one of the main keypoints is mismatched, HMR will compute an oversized bounding box, and run on irrelevant pixels. During testing erroneous OpenPose output led to errors in 3D pose estimation. Consequently, one has to preprocess and crop frames accordingly, to reduce possible sources of error.

4.5.3 Rotations and Perspective

Compared to Mo-Cap with a single Kinect, pose estimation captures rotations remarkably well. The Kinect requires the subject to face the sensor, so an actor turning around its own axis will result in erroneous poses. Pose estimation is not principally limited to the front perspective of humans because datasets contain humans from all kind of perspectives, thus capturing rotations in higher quality. (See figure 4.5 on the following page)

However, the visual quality of pose estimation declines, if the actor's back faces the camera: The movements seem less distinct. This is most likely due to the training datasets containing less images of people with their back towards the camera.

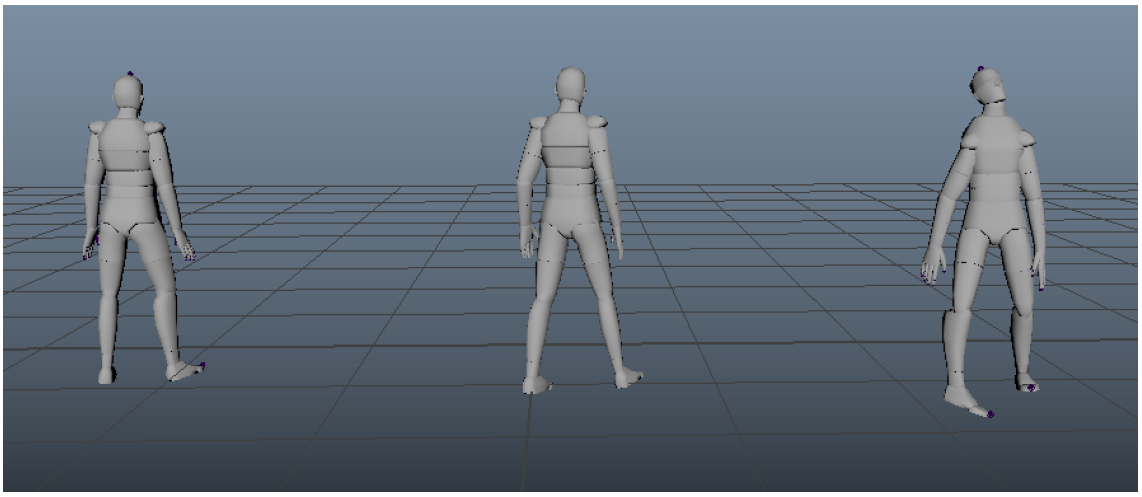


Figure 4.5: Raw Mo-Cap data just after rotation of the actor. Corresponding systems from left to right: RADiCAL, HMR, Kinect

4.5.4 Use on a Plate

While one cannot use Mo-Cap for an already shot plate, pose estimation enables this. Consequently, 3D pose estimation was evaluated regarding qualities, that matter for the use on plates. This covers the performance regarding camera movements, surroundings and subjects not framed as a whole.

First off, the used systems analyze frames individually, similar to OpenPose. Thus, temporal lighting changes will neither influence results.

Movement in space through projection assumes a locked-off camera. Due to this principle, any camera movement directly influences movement in space for RADiCAL. So one does need to constrain the root to a specific point. Moreover, RADiCAL will fail to estimate a pose if the actor is not fully framed.

Camera movement is not relevant for the HMR-workflow, because subjects do not move in space and frame are estimated individually. HMR always outputs a complete skeleton, and it still produces reasonable output, even if the subject is not fully framed. In this case, the body parts in the frame are visually coherent, while anything outside the frame is represented by a probable pose, defined through the discriminator network. Body parts outside the frame do not match the performance.

Both systems produced reasonable output in front of homogenous and inhomogeneous backgrounds. No differences could be observed regarding the general visual quality, but the number of erroneous poses increased in front of a inhomogeneous background. This could also be a result of the restricted movements in front of a homogenous background.

4.5.5 Resume

First off, pose estimation cannot be used like any state-of-the-art Mo-Cap system, because the output is missing subtle details and the results are not reliably positioned in space.

Nevertheless, 3D pose estimation offers other possibilities. Principally as an animation reference, previz and preview tool. When working with video reference, poses and motion can be automatically previewed on the respective character. This can be especially beneficial and save time, when working with a variety of iterations. A lot of animation references can be easily and automatically previewed on the respective character. Accordingly, one can capture a variety of explicit animation references with a single camera. Ideally in front of a homogenous background and while facing the camera.

In addition, pose estimation can build the basis for *Pose-to-Pose*[11] animation. One can only pick distinctive poses of the output and discard the remaining frames. While the animation lacks details, key poses of HMR are biomechanically correct, and coherent to the capture. It would not be necessary to pose a character from scratch. This could be based upon specifically captured references, but also be used on a plate. Ergo pose estimation can be the fundament for rotomation, because poses can be predicted for shots with a moving camera, and HMR will always output a complete skeleton, despite parts of the body not being in the frame.

When using pose estimation, HMR is the better choice, because temporal filtering can be controlled, and it does not come with the limitations and costs of the cloud-based workflow. Furthermore, movements in space of both systems need manual adjustments: One has to completely create movements in space for HMR, but the movements of RADiCAL need distinct cleanup.

[11]The process of generating key poses first, then refining inbetweens

4.5.6 Further Considerations

Testing with 3D pose estimation led to the following additional ideas.

Orientation of Hands and Feet:

Similarly to rotoscoping with OpenPose, the HMR workflow does not orient the hands and feet. While OpenPose currently features a pose representation that enables this, the HMR output is missing the appropriate keypoints. Since hand and feet can be essential to the impression of a pose, bringing in the orientation of these is desirable.

One has to consider, that the estimated keypoints depend on the labels of the training datasets. Right now neither COCO, nor MPII feature the required keypoints. A network cannot be trained on missing data, so the orientation of hand and feet requires extended datasets.

Use of HPE for Crowds:

In the near future, optical or mechanical Mo-Cap solutions will most likely remain the standard for big budget productions and lead characters. However, HPE could be used for simple animation of inconspicuous characters in backgrounds and crowds.

With pose estimation, one can create a large animation library for crowds. This would only need the appropriate video input, gained from specific shoots or from general resources. A greater variety of animations can be obtained more quickly than with special Mo-Cap sessions or keyframe-animation.

Hybrid Approach:

Mo-Cap with a Kinect can produce unreasonable poses, contrary to the HMR-output, which is controlled by a discriminator network.

So one can control the depth based output-skeleton of the Kinect with a similar discriminator, to ensure probable anatomy. Moreover, estimated poses can work as a guidance for the depth-based skeleton.

5 Conclusion and Prospects

5.1 Conclusion

In this thesis, workflows for the use of pose estimation in VFX applications were described and evaluated. More precisely, 2D pose estimation as a rotoscoping fundament and 3D pose estimation to pose and animate CG characters.

After gaining a fundamental understanding of machine learning, the knowledge was used to utilize and evaluate HPE practically.

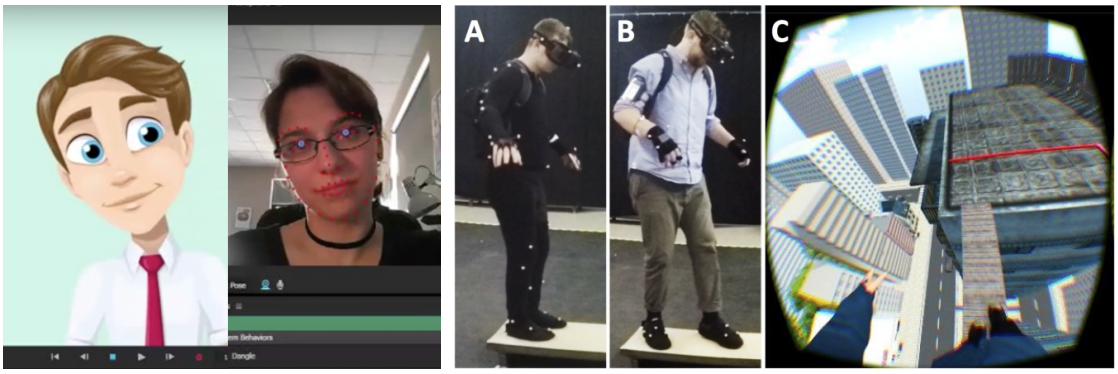
2D pose estimation was integrated with a nuke gizmo: The desired plate is analyzed with OpenPose and roto-shapes are positioned along anatomical keypoints. The fundamental movements of the human in the frame, acquired by OpenPose, are transferred onto the roto-shapes. Temporal jittering is an issue with this workflow but can be controlled to some extent by filtering. This workflow lacks accurate capture of detail but outperforms planar tracking for hard-to-track shots in terms of speed.

CG characters can be posed and animated with the described workflows for 3D pose estimation. The primary challenge is the transfer between estimated keypoints and a skeleton that suits animation. Problems with temporal consistency and accuracy are similar to 2D pose estimation. Hence, 3D pose estimation is not a complete Mo-Cap alternative but offers fast workflows for previewing and a fundament for pose-to-pose animation.

The findings show first and foremost, that human pose estimation is a promising tool for acquiring poses and movements with simple setups and in situations where it was not possible before. However, it is not on par with state-of-the-art techniques concerning accuracy and temporal consistency. Further developments, especially regarding smooth, but precise human movements, would dramatically increase the value of such workflows.

All in all, HPE is a specific tool, which can benefit some productions, but is currently not a universal solution. The proposed workflows shall give suggestions for the use and encourage further developments.

5.2 Prospects and possibilities



(a) Character Animator facial animation

(b) Embodied VR experience

Figure 5.1: Prospects and possibilities[12][13]

The possibility to acquire human poses with a single RGB camera offers a wide range of further opportunities for visual effects and media technology, which were not covered in this work. An overview about these opportunities will be given in the following chapter.

First off, analogous to rotoscoping with pose estimation, these systems could be used to animate 2D characters. Adobe Character Animator offers tools for the animation of *puppets*: A 2D-character only visible from one perspective, is divided into limbs and body parts, which can be moved by a rig. Moreover, Character Animator provides the possibility to convert videos into facial animation, using facial recognition. [Nece, 2017] (See a) figure 5.1) Pose estimation would offer an equivalent functionality for full bodies, by driving the rig with keypoint coordinates. Obviously the current main problems - temporal jitter and lack of accuracy - would affect the output. Nevertheless this could allow quick blocking, preview and usage as a fundament.

Besides standard postproduction workflows, real-time 3D pose estimation [Mehta et al., 2017] offers possibilities for interactive applications and virtual production. With a direct link of pose estimation to a game engine, one would only need a simple locked-off witness camera to preview animated digital characters.

This can be suitable for interactive installations reacting to human interaction. Spectators would not need extra equipment to be captured and are not limited to a defined Mo-Cap Volume. Only the field of view and exposure of the witness camera

[12] source: https://www.youtube.com/watch?v=z1nZGyLYydc - access: 2018-08-31

[13] source: http://cg.web.th-koeln.de/a-simplified-inverse-kinematic-approach-for-embodied-vr-applications/ - access: 2018-08-31

does need to be considered. VR or AR experiences with multiple users can similarly take advantage of that. Accordingly, poses of every user can be captured by a witness camera, transferred onto characters and then be displayed in a users view. To achieve that, poses would need to be accurately positioned in space, relative to the camera defining the view of a user. Likewise, embodied VR experiences can be realized without further equipment than the virtual reality glasses.

Furthermore, the animation gained with a witness camera in real-time could be used for virtual production and live preview on set. In combination with real-time camera tracking, an animated character can be previewed, while capturing a shot. This would benefit virtual production outside of a studio, and if wearing a Mo-Cap suit is not feasible.

All in all, the primary benefits of the respective opportunities lie in the simple setup, which can be used in any uncontrolled environment.

Nomenclature

API	Application Programming Interface (short API) allows third parties to add functions to an existing system. The API defines the standard structure and commands.
JSON	JavaScript Object Notation (short JSON) is a standard file format for the exchange between different applications. It consists out of pairs of attributes and values, respectively arrays. Its format makes it compact and readable by humans.
Rotomation	Rotomation refers to the animation of a CG character, to recreate the motion of a captured subject from a plate.

Abbreviations

CG	**C**omputer **G**raphics
CNN	**C**onvolutional **N**eural **N**etwork
Mo-Cap	**Mo**tion **Cap**ture
HPE	**H**uman **P**ose **E**stimation
Previz	**Previs**ualization
Roto	**Roto**scoping
VFX	**V**isual Effects

Bibliography

[Andersen et al., 2012] Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P. (2012). Kinect Depth Sensor Evaluation for Computer Vision Applications.

[Andriluka et al., 2014] Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). 2d Human Pose Estimation: New Benchmark and State of the Art Analysis. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3686–3693, Columbus, OH, USA. IEEE.

[Autodesk, 2018] Autodesk (2018). Maya Documentation. http://help.autodesk.com/view/MAYAUL/2018/ENU/. Last checked on Aug 30, 2018.

[Cao et al., 2016] Cao, Z., Simon, T., Wei, S.-E., and Sheikh, Y. (2016). Realtime Multi-Person 2d Pose Estimation using Part Affinity Fields.

[Casiez et al., 2012] Casiez, G., Roussel, N., and Vogel, D. (2012). 1 € filter: a simple speed-based low-pass filter for noisy input in interactive systems. page 2527. ACM Press.

[Chen et al., 2017] Chen, L.-C., Hermans, A., Papandreou, G., Schroff, F., Wang, P., and Adam, H. (2017). MaskLab: Instance Segmentation by Refining Object Detection with Semantic and Direction Features. *arXiv:1712.04837 [cs]*. arXiv: 1712.04837.

[Foundry, 2018] Foundry (2018). Nuke Documentation. https://learn.foundry.com/nuke/11.0/. Last checked on Aug 30, 2018.

[Frazer, 2015] Frazer, R. (2015). Keyframe Reduction script for Nuke. http://richardfrazer.com/tools-tutorials/keyframe-reduction-script-for-nuke/. Last checked on Aug 30, 2018.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. The MIT Press, Cambridge.

[Güler et al., 2018] Güler, R. A., Neverova, N., and Kokkinos, I. (2018). DensePose: Dense Human Pose Estimation In The Wild. *arXiv:1802.00434 [cs]*. arXiv: 1802.00434.

[Guliyev and Ismailov, 2016] Guliyev, N. J. and Ismailov, V. E. (2016). A single hidden layer feedforward network with only one neuron in the hidden layer can approximate any univariate function. *Neural Computation*, 28(7):1289–1304. arXiv: 1601.00013.

[Hewitt, 2013] Hewitt, S. (2013). Rotation Gimbal & Gimbal Lock. https://www.youtube.com/watch?v=V-qSuzEsdD8t=163s. Last checked on Aug 30, 2018.

[Ionescu et al., 2014] Ionescu, C., Papava, D., Olaru, V., and Sminchisescu, C. (2014). Human3.6m: Large Scale Datasets and Predictive Methods for 3d Human Sensing in Natural Environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339.

[Joosten, 2018a] Joosten, R. (2018a). hmr: Project page for End-to-end Recovery of Human Shape and Pose. https://github.com/robertjoosten/hmr. Last checked on Aug 30, 2018.

[Joosten, 2018b] Joosten, R. (2018b). video2mocap: Convert a video file to animated humanIK skeletons for Maya. https://github.com/robertjoosten/video2mocap. Last checked on Aug 30, 2018.

[Kanazawa et al., 2017] Kanazawa, A., Black, M. J., Jacobs, D. W., and Malik, J. (2017). End-to-end Recovery of Human Shape and Pose. *arXiv:1712.06584 [cs]*. arXiv: 1712.06584.

[Kitagawa and Windsor, 2008] Kitagawa, M. and Windsor, B. (2008). *MoCap for Artists*. Elesevier Inc., Burlington, MA.

[Krishnant and Ravivt,] Krishnant, S. and Ravivt, D. 2d Feature Tracking Algorithm for Motion Analysis. page 24.

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2014). Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*. arXiv: 1405.0312.

[Loper et al., 2015] Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., and Black, M. J. (2015). SMPL: a skinned multi-person linear model. *ACM Transactions on Graphics*, 34(6):1–16.

[MathWorks, 2018] MathWorks (2018). MATLAB Documentation. https://www.mathworks.com/help/nnet/ug/pretrained-convolutional-neural-networks.html. Last checked on Aug 30, 2018.

[Mehta et al., 2016] Mehta, D., Rhodin, H., Casas, D., Fua, P., Sotnychenko, O., Xu, W., and Theobalt, C. (2016). Monocular 3d Human Pose Estimation In

The Wild Using Improved CNN Supervision. *arXiv:1611.09813 [cs]*. arXiv: 1611.09813.

[Mehta et al., 2017] Mehta, D., Sridhar, S., Sotnychenko, O., Rhodin, H., Shafiei, M., Seidel, H.-P., Xu, W., Casas, D., and Theobalt, C. (2017). VNect: real-time 3d human pose estimation with a single RGB camera. *ACM Transactions on Graphics*, 36(4):1–14.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill Education, New York, NY, 1 edition edition.

[Nece, 2017] Nece, V. (2017). Unique and Powerful Animation Features Added to Character Animator CC.

[Nielsen, 2015] Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

[Okun and Zwerman, 2010] Okun, J. A. and Zwerman, S. (2010). *The VES Handbook of Visual Effects*. Elsevier.

[Parloff, 2016] Parloff, R. (2016). The AI Revolution: Why Deep Learning Is Suddenly Changing Your Life. http://fortune.com/ai-artificial-intelligence-deep-machine-learning/. Last checked on Aug 30, 2018.

[Rueter, 2010] Rueter, F. (2010). Smoothing Animation Curves - Expressions - Nukepedia.

[Sigal, 2014] Sigal, L. (2014). Human pose estimation. In *Computer Vision*. Springer, New York, NY.

[Solutions, 2018] Solutions, R. (2018). RADiCAL - AI-powered Motion Capture.

[Yates, 2016] Yates, B. (2016). Motion Capture Part 2 : Understanding the HIK and Animation Layers. https://www.youtube.com/watch?v=U31sq0dwNMA. Last checked on Aug 30, 2018.