Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Joni Vähämäki

Real-time climbing pose estimation using a depth sensor

Master's Thesis
Espoo, March 7, 2016

Supervisor: Assistant Professor Perttu Hämäläinen
Advisor: Raine Kajastila D.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Joni Vähämäki

Title:
Real-time climbing pose estimation using a depth sensor

Date:	March 7, 2016	**Pages:**	47
Major:	Media Technology	**Code:**	T-110
Supervisor:	Assistant Professor Perttu Hämäläinen		
Advisor:	Raine Kajastila D.Sc. (Tech.)		

Augmented feedback provided either by a human expert or an automated system enhances learning in sports. In the growing sport of rock climbing, few solutions currently exist for automating augmented feedback. If the climber's body pose could be accurately tracked, the pose information could be analyzed to provide augmented feedback to the climber. For tracking the climber's body on the climbing wall, off-the-shelf solutions like Microsoft Kinect are unsuitable, because they are optimized to detect front-facing standing poses that are typical in a living room environment.

In this thesis, recent human pose estimation methods are reviewed and the state-of-the-art offset joint regression method is implemented and evaluated in the novel domain of climbing pose estimation. The method utilizes machine learning, and is able to infer the locations of skeletal joints directly from a depth image captured using a depth sensor. The offset joint regression model is trained using computer-generated synthetic training data that mimics real world depth sensor images.

Promising results are achieved on a synthetic test dataset, suggesting that the chosen method is suitable for climbing pose estimation. However, the implemented method fails to detect some atypical poses correctly. In the future, the results could be improved further by improving the training data. Also, taking into account kinematic and temporal constraints in the pose estimation could improve the results.

Keywords: human pose estimation, body tracking, depth sensor, indoor climbing, machine learning, body part classification, offset joint regression, augmented feedback

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Joni Vähämäki		
Työn nimi:	Reaaliaikainen kiipeilyasennon estimointi syvyyssensorin avulla		
Päiväys:	7. maaliskuuta 2016	**Sivumäärä:**	47
Pääaine:	Mediatekniikka	**Koodi:**	T-110
Valvoja:	Apulaisprofessori Perttu Hämäläinen		
Ohjaaja:	Raine Kajastila D.Sc. (Tech.)		

Ihmisasiantuntijan tai automoidun järjestelmän antama palaute tehostaa oppimista urheilussa. Kiipeily on kasvava laji, jossa ei ole vielä hyödynnetty automoitua palautetta. Jos kiipeilijän asentoa voitaisiin seurata tarkasti, kiipeilijälle voitaisiin antaa palautetta analysoimalla asentotietoa. Valmiit ratkaisut, kuten Microsoftin Kinect, eivät sovellu kiipeilijän asennon tarkkaan seuraamiseen, sillä ne on optimoitu tunnistamaan vain asentoja, jotka ovat tyypillisiä olohuoneympäristössä.

Tässä diplomityössä esitellään viimeisimmät menetelmät ihmiskehon asennon estimointiin. Työssä toteutetaan offset joint regression -menetelmä, jota evaluoidaan kiipeilyasentojen tunnistuksessa. Menetelmä hyödyntää koneoppimista, ja tunnistaa ihmisen nivelten paikat suoraan annetusta syvyyssensorin kuvasta. Offset joint regression -malli opetetaan käyttäen koneellisesti generoitua synteettistä opetusdataa, joka jäljittelee oikeita syvyyssensorin kuvia.

Synteettisellä testidatasetillä saavutettiin lupaavia testituloksia, jotka vihjaavat, että menetelmä soveltuu kiipeilyasentojen tunnistukseen. Menetelmä ei kuitenkaan tunnistanut joitain epätyypillisiä asentoja oikein. Tulevaisuudessa tuloksia voitaisiin mahdollisesti parantaa kehittämällä opetusdataa. Lisäksi tuloksia voitaisiin parantaa ottamalla huomioon kinemaattiset ja ajalliset rajoitteet asennon tunnistuksessa.

Asiasanat:	ihmisen asennon tunnistaminen, kehon seuranta, syvyyssensori, kiipeily, koneoppiminen
Kieli:	Englanti

Acknowledgements

First, I would like to thank my supervisor, Assistant Professor Perttu Hämäläinen for introducing me to this topic and for his guidance on the thesis. I would also like to thank my advisor Raine Kajastila for giving valuable feedback on the thesis. Finally, I would like to thank Anni, Leo and Lari for their help.

Espoo, March 7, 2016

Joni Vähämäki

Contents

Chapter 1

Introduction

1.1 Overview

In recent years, computer vision algorithms for human body pose estimation from images have seen considerable improvement. New sensor technologies such as depth cameras have also enabled new possibilities for computer vision. For example, Microsoft Kinect brought affordable general purpose body tracking to the consumer market, coupling a depth camera with a software layer capable of extracting 3D human pose information from the depth camera's images in real-time.

Advances in real-time human pose estimation have inspired various types of applications. An active area of research is utilizing body tracking to provide augmented feedback in sports training [24]. Augmented feedback enhances learning by providing information about the athlete's performance that the athlete cannot perceive themselves during the performance.

In this thesis we focus on human pose estimation in the context of the rapidly growing sport of indoor climbing. In indoor climbing, athletes climb on artificial structures that resemble outdoor rock formations. Climbing is physically challenging, and successfully completing routes requires good technique. If a climber's body could be tracked reliably, the information could be used to analyze their movements and provide instantaneous feedback on how to improve their climbing technique. Real-time body tracking could also enable interactive experiences where the climber's body acts as an input device. For example, the augmented climbing wall[13] is an interactive climbing environment, which would benefit from efficient and robust body tracking.

We study existing research and methods on human pose estimation, and implement a real-time human pose estimation system for indoor climbing. In a climbing context, existing off-the-shelf body tracking solutions like Mi-

crosoft Kinect are too inaccurate, as they are unable to detect many typical climbing poses. This thesis does not propose any novel algorithms, but contributes through applying and evaluating the recent offset joint regression method introduced by Girshick et al.[10] in the novel domain of climbing poses.

1.2 Problem statement

In this thesis we study whether a climber's body pose can be accurately estimated in real time from depth images captured using a single depth sensor by utilizing recent computer vision methods that have previously been successfully applied in other contexts.

We focus on the problem of climbing pose estimation from a single depth image. In particular, we do not track motion across several consecutive images. We also assume that the climber is separated from the background. In the input depth image, pixels that are part of the background (typically the climbing wall in the context of this thesis) are assigned a large constant value. We parametrize human body pose as a set of 3D joint positions.

We implement the offset joint regression method introduced by Girshick et al.[10], and evaluate the accuracy of the approach in a climbing context by measuring mean average precision over a test set of computer-generated depth images of climbing poses labeled with 3D joint positions.

1.3 Structure of the thesis

In chapter 2, Background, we introduce the problem of human pose estimation and summarize the relevant recent research in the field. In chapter 3, Environment, we introduce indoor climbing, as well as the software foundations on which we build our climbing pose estimation system. In chapter 4, Methods, we describe in detail the computer vision methods we utilize in our real-time human pose estimation system. In chapter 5, Implementation, we provide details on the technical implementation of our system. In chapter 6, Evaluation, the performance of our pose estimation system is evaluated. In chapter 7, Discussion, we discuss the results of the evaluation and potential for future development. Finally, chapter 8, Conclusions, summarizes what was done in the thesis.

Chapter 2

Background

In this chapter we introduce the problem of human pose estimation and the recent research that is relevant to this thesis. We introduce random decision forests, which is a machine learning method utilized in many state-of-the-art human pose estimation algorithms.

2.1 Human pose estimation

Human body pose estimation from images has been researched extensively in recent years [16][7]. This is a difficult problem especially in a general case, because the human body has a highly articulated non-rigid structure. Often a skeletal model is used, where the body pose is parametrized in terms of rotations of skeletal joints. Due to the complexity of the human skeleton which consists of several chains of joints, the number of possible body poses is innumerable.

In addition to pose variations, variations in human shape and size are also challenges to a pose estimation algorithm. Some algorithms require a user-specific calibration step to achieve accurate results. Estimating the underlying rigid skeletal structure of a human is complicated by non-rigid deformations on the surface caused by skin and clothing. Due to self-occlusions, multiple camera views may be required before a body pose can be uniquely characterized.

The type of input data also poses further challenges to human body pose estimation. Separation of geometry and texture is difficult from intensity images captured using conventional cameras. Accurate 3D tracking is hard from two-dimensional intensity images even with a calibrated multi-camera setup. Vision-based motion capture systems often use specialized markers worn by the performer to ease the task of matching common features between

images. However, markers make the motion capture process more tedious and impractical for casual use.

Depth sensors such as Microsoft Kinect capture depth information, which can be utilized in 3D pose estimation. With accurate per-pixel depth information available, 3D joint positions can be estimated even from a single image. However, due to ambiguities caused by self-occlusions, the problem of body pose estimation from a single depth image is still inherently ill-posed.

In this thesis we focus on methods which do not require any user-specific calibration or markers. We use a single depth camera for input due to the affordability, performance and convenience compared to multi-camera setups.

2.2 Generative and discriminative pose estimation

Human body pose estimation methods can be divided into *discriminative* and *generative* methods [23]. Discriminative methods attempt to learn a direct mapping from input data to body pose typically utilizing machine learning methods. Generative methods use a parametric model to create synthetic observations that can be compared with the input data. The parameters of the model are optimized in order to match the synthetic observation with the input data. The optimization problem is multimodal and typically sampling-based optimization methods such as particle filters and particle swarm optimization (PSO) are used [12]. In related literature, discriminative methods are occasionally referred to as *detection* methods, while generative methods are sometimes called *tracking* methods.

Generative methods generally require a good initial guess of the pose to reach a good solution. Also, when optimizing the body pose parameters, usually only a local minimum can be found in a reasonable amount of time due to the high dimensional nature of the problem. Discriminative methods do not require an initial guess, but the accuracy of the pose estimates is tied to the robustness of the underlying machine learning model as well as the quality and amount of training data. Real world tracking systems typically combine both discriminative and generative methods to maximize robustness when tracking poses [21][26][27].

In this thesis we limit our scope by focusing on discriminative methods. Our pose estimation system needs to work without manual initialization over long periods of time, so discriminative methods are required. In later research, the implemented discriminative method may be combined with a generative method to improve accuracy.

2.3 Random decision forests

Discriminative pose estimation typically utilizes generic machine learning methods. Decision trees are a typical building block in machine learning algorithms. Prediction models for classification and regression tasks based on decision trees (CARTs) were introduced by Breiman et al. in the book "Classification and regression trees" [6]. So called *ensemble* methods combine several decision trees in order to increase the predictive accuracy and generalization of the model [9].

In a typical classification tree, each inner node (*split* node) of the tree stores a test (feature), and each leaf node stores a class label. Each inner node has a number of child nodes depending on the possible number of test outcomes. To classify an element, the nodes of the trees are iterated starting from the root of the tree. At each iteration, the current node's test is evaluated, and the result of the test determines which child node is evaluated next. Eventually a leaf node is reached, and the class label stored in the leaf becomes the classification result.

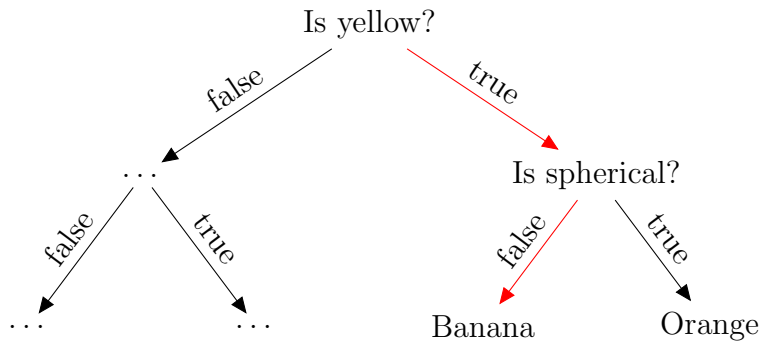

Figure 2.1: A toy example of a classification tree that analyzes color and shape features to classify fruit

Figure 2.1 shows a simplified example of a classification tree. The highlighted path shows how a fruit is classified as a banana according to simple visual features. In this example, the tests at the inner nodes can have only two outcomes, resulting in a binary tree shape, but real tests may have any number of outcomes.

Common algorithms for learning classification and regression trees perform a deterministic exhaustive search over the feature space in order to select the most discriminative features for each node of the decision tree. The resulting trees are optimal, but the training is computationally expensive, making such methods impractical for high dimensional data.

A randomized method for training the split nodes of an ensemble of decision trees was proposed by Y. Amit and D. Geman [1] in their work on handwritten digit recognition. The method randomly samples a set of candidate features at each node, and selects the candidate that maximizes the expected gain in information from the set. The method was later termed "random forests" by Breiman[5].

Breiman also introduced the concept of bootstrap aggregation, alternatively termed *bagging*[4]. In bagging, the training set is split into a number of equally sized *bags* by uniformly sampling the set *with replacement*, meaning the same training sample may be picked multiple times. A corresponding number of trees is then trained from the bagged training sets. The final prediction model averages the output of the trees. Breiman showed that bagging can improve prediction accuracy and reduce overfitting. Overfitting occurs when the training algorithm learns specific random features of the training data, which reduces the accuracy when the model is applied to predict output on unseen data.

2.4 Body pose estimation from single depth images

In this thesis we focus on discriminative pose estimation methods that can produce results in real-time and run on consumer-grade hardware. The methods described here estimate the position of skeletal joints from individual depth images captured with a depth sensor. These methods do not utilize any temporal information, but the pose estimates produced by the following algorithms may be used as initial guesses and for reinitialization to avoid drift in a generative motion tracking algorithm.

2.4.1 Body part classification

In [22], Shotton et al. introduced *body part classification* (BPC), which is a method for estimating the pose of a human in real-time from single depth images. The method's final output is a set of confidence-weighted skeletal joint position proposals. According to the authors, the method is a core component in the body tracking pipeline of Microsoft Kinect, where it is likely used for initialization and recovery.

In BPC, the surface area of the human body is divided into *body parts*. A random decision forest classifier is trained, which classifies depth pixels to body parts. The model is then used to classify pixels in unseen input

Figure 2.2: A depth image and the intermediate body part representation used in BPC

depth images, and joint position proposals for these images are formed by clustering the classified pixels. Because the joints are inside the body while the body parts lie on the surface the body, a joint-specific pushback is applied to produce the final joint proposals. The method is described in more detail in chapter 4.

Shotton et al. use a very large set of computer-generated training data to train the model. The random decision forest they use has three trees trained to depth 20 with 900,000 training images. The authors show that the quality and quantity of the training data has a significant effect on the predictive accuracy of the model.

At the time of its publication, BPC produced state-of-the-art results. The authors used synthetic data as well as real data captured using a Microsoft Kinect depth camera to evaluate the performance of the method. High prediction accuracy was achieved especially in cases with no significant self occlusions. However, the method will fail if joints are fully or mostly occluded. The method also does not enforce any kinematic constraints, such as joint rotation limits or distances between joints.

Although the training data cannot contain all possible poses, the authors note that the method generalizes well to unseen data, because the pose recognition is done in parts instead of trying to match a complete pose at once. Also, since decision tree evaluation is cheap, the algorithm can run in real-time. The per-pixel random decision forest evaluation is highly parallelizable, so a further speedup can be achieved by implementing the forest evaluation on a GPU.

2.4.2 Offset joint regression

In [10], Girshick et al. introduced *offset joint regression* (OJR), which is an improved method based on BPC that is able to predict joint positions directly from a single input depth image without the intermediate body part representation. In OJR, the leaf nodes store a distribution over relative joint offsets. When compared to BPC, the method is found to produce more accurate results. The method is also shown to produce better results in cases where some of the joints are occluded. The authors used a random decision forest with three trees trained to depth 20 using 300,000 training images to demonstrate their results.

Interestingly, best results are obtained when the structure of the random decision forest is trained using a BPC classification objective, while the leaf nodes are trained for joint regression. A method that attempts to train the tree structure by optimizing a regression objective function was found to be less accurate. The authors believe this is due to tradeoffs in the choice of the objective function that were necessary to make training efficient.

Like BPC, OJR estimates joint positions independently from each other and does not enforce kinematic constraints. In [25] Sun et al. present a conditional regression model that extends OJR by incorporating dependency between output variables through a global latent variable, such as torso orientation or human height. The model is shown to improve prediction accuracy over OJR.

2.4.3 Vitruvian manifold and metric regression forests

In [26], Taylor et al. propose an improved human pose estimation method termed "vitruvian manifold" building on the ideas in BPC and OJR. Unlike the previously introduced methods, the method also enforces kinematic constraints by fitting a skeletal model to the input depth images. The authors train a regression forest which directly regresses correspondences between points in the depth image and points on a 3D human model. With the correspondences known, the model fitting parameters can then be optimized efficiently. As was done by Girshick et al. in [10], a body part classification proxy objective is used at training time to train the structure of the trees.

More recently, Pons-Moll et al. introduced a regression objective function for the purpose of correspondence estimation in [19]. Compared to the body part classification objective, the authors achieve slightly better accuracy with considerably less training data, using only 5000 training images compared to [26] where 300,000 images were used. However, evaluating the new objective function is considerably more expensive, which increases the training time.

According to the authors, training the random decision forest with 5000 images took three times longer than it did with 300,000 images in [26].

2.5 Recent advances

Recently Ren et al.[20] proposed a method that optimizes a pre-trained random forest by refining the leaf nodes in order to utilize complementary information between multiple trees better. The trees are also pruned to reduce model size and overfitting. The authors demonstrate an increase in predictive accuracy and a decrease in model size when the optimization is applied to BPC.

Online training of random decision forests is also an active area of research. *Mondrian forests* introduced by Lakshminarayanan et al. in [14] enable efficient online training of random forests that perform almost as well as forests trained offline. In the context of human pose estimation, online training would, for example, allow training a discriminative pose estimation model on-the-fly with a generative model.

Chapter 3

Environment

In this chapter, we discuss the motivation for applying human pose estimation in a climbing context, and introduce the challenges specific to climbing pose estimation. We also describe the hardware we use and the software foundations on which we build our climbing pose estimation system.

3.1 Indoor climbing

3.1.1 Overview

Indoor climbing is an increasingly popular form of rock climbing which is performed on artificial indoor structures that resemble natural rock formations. Climbing is practiced as a form of exercise as well as recreationally. Indoor climbing gyms provide an environment where climbing can be practiced year-round in comfortable and safe conditions.

Typical climbing walls consist of wooden boards that form a surface with varying slopes. Varying types of artificial climbing holds are attached to the wooden boards. Typically climbers climb a route which consists of a selection of climbing holds. Since the climbing holds can be easily exchanged, various different types of routes can be constructed on a single static wall.

Our intent is to employ human pose estimation methods in order to provide real-time feedback to a climber while they are climbing on a climbing wall. For example, accurate pose estimation may be utilized in teaching new climbing moves to the climber. For image acquisition we use a single Kinect 2 depth sensor, which is placed in front of the climbing wall. We focus especially on bouldering walls. Bouldering walls are low walls with mattresses for safety instead of ropes. With a bouldering wall, the field of view of a stationary depth sensor is sufficient to cover the area of the wall, and background

extraction is not complicated by the presence of ropes.

3.1.2 Challenges for human pose estimation

Climbing poses are especially challenging for a vision-based human pose estimation algorithm. As an example, we compare climbing to the typical living room entertainment use case of Microsoft Kinect. In a living room scenario the user is standing in front of the sensor. Assuming standing and front-facing poses restricts the pose space significantly. For example, when standing unsupported, the user has to keep his balance in order to avoid falling down, limiting the number of possible poses. A climbing wall on the other hand allows highly varying poses, as the climber can use their hands and feet to support themselves on the wall. While the climber is mostly facing the wall, which constraints the poses somewhat, the poses are not limited to ones that maintain balance like in the living room scenario.

Occlusions are also common. Often the climber's torso may occlude hands or legs from the viewpoint of the camera. In these cases vision-based pose estimation methods will struggle.

3.2 Hardware

3.2.1 Microsoft Kinect depth sensors

Kinect is a family of body tracking sensors developed by Microsoft. Kinect sensors use computer vision algorithms to extract human pose from depth images captured with a depth camera embedded into the hardware component.

The first version of the Kinect sensor (Kinect 1) was released for use with Xbox 360 game consoles in 2010. The sensor emits an infrared structured light pattern, and estimates depth by analyzing deformations in the pattern when it is projected into the scene. In 2013, a second version of Kinect was released for Microsoft's Xbox One game consoles. Compared to Kinect 1, Kinect 2 features improved accuracy and a wider field of view. Kinect 2 uses time-of-flight technology to capture depth information, where the sensor emits pulses of infrared light and measures the travel time for each captured depth pixel. The Kinect 2 sensor produces depth images in the resolution of 512x424 pixels compared to the resolution of 320x240 in Kinect 1.

As the body tracking provided by Kinect's SDK is not suitable for our purposes due to its inaccuracy in recognizing climbing poses, we use Kinect only as a depth sensor providing the depth input to our own body tracking

implementation. As the sensor was originally designed for use in a living room scenario, it has an effective range from 0.4 meters to 4.5 meters. The field of view is sufficient to cover a climbing wall that is four meters wide and tall, when the sensor is placed about four meters away from the wall. As a pre-processing step before pose estimation, we extract the climber from the background using background subtraction [11].

3.2.2 Noitom Perception Neuron

To generate realistic synthetic training data for our pose estimation model, we need an extensive dataset of realistic climbing poses. For this purpose, we use motion capture to capture climbing movements. We use Perception Neuron[17], which is a motion capture system developed by Noitom. Perception Neuron captures full-body motion using a number of wearable inertial sensor units, which are placed around the body. Each sensor unit contains a gyroscope, an accelerometer and a magnetometer, which are utilized for precise tracking of motion.

Unlike vision-based motion capture systems, an inertial motion capture system such as Perception Neuron does not require cameras to be placed around the performer. Since we use motion capture to capture the movements of a climber on a climbing wall, surrounding the performer with cameras from all angles is not possible. Also, Perception Neuron is significantly easier set up than a vision-based system with several cameras. This makes it feasible to even perform motion capture sessions in actual climbing gyms. Figure 3.1 shows a motion capture session in progress on a climbing wall.

3.3 Software

We implement our pose estimation system in C++. In this section, the additional tools and libraries that we use in our implementation are listed.

MakeHuman For the purposes of training data generation, we use a 3D human mesh generated using MakeHuman[15], which is an open-source tool for 3D character creation.

MotionBuilder Our training data generation process requires retargetting motion-captured climbing motions to a HumanIK skeleton, which is compatible with the MakeHuman mesh. For this purpose we use Autodesk MotionBuilder[2].

Figure 3.1: Capturing movements on a climbing wall using Perception Neuron

3ds Max For additional adjustment of the 3D human mesh and the retargeted animations, we use Autodesk 3ds Max[3], which is a popular 3D modeling application. Most significantly, we use 3ds Max to apply a body part texture on the 3D mesh generated by MakeHuman.

Assimp We use the Open Asset Import Library[18] (Assimp) C++ library to import the 3D mesh and the retargeted animations for training data generation.

OpenMP OpenMP is a toolkit for parallel programming on the CPU for the C++ language. We use it to accelerate the training of our machine learning model as well as some parts of the evaluation of the model.

CUDA While the first graphics processing units (GPUs) were only capable of rendering graphics, modern GPUs are fully programmable and enable highly parallel general-purpose computing utilizing the GPU's

many cores. CUDA is a platform for GPU-accelerated computing, which is supported by graphics cards manufactured by Nvidia Corporation. We use CUDA in our implementation to speed up parts of the model evaluation.

Chapter 4

Methods

In this chapter, the methods used in the implementation of our body pose estimation system are described in detail. We first present the mean shift mode finding algorithm, which is used extensively by the pose estimation methods described later in this chapter.

We choose to implement and evaluate the offset joint regression (OJR) method introduced by Girshick et al. [10] due to its speed and robustness. In the training phase we utilize a body part classification (BPC) objective, which was originally introduced by Shotton et al. in [22].

4.1 Mean shift

Mean shift[8] is a method for locating the modes (local maxima) of a probability density function estimated from finite data samples. Kernel density estimation is used to estimate the density function from the samples. Given a number of initial mode estimates, mean shift iteratively performs a gradient ascent on the density, moving each mode estimate towards the peak of the density. At each iteration, each mode estimate x is updated as follows

$$x = m(x) = \frac{\sum_{x_i \in D} K(x_i - x) x_i}{\sum_{x_i \in D} K(x_i - x)} \tag{4.1}$$

where D is the set of data samples and $K(x_i - x)$ is a kernel function. $m(x)$ is the weighted mean in the neighborhood of x, where the kernel function $K(x_i - x)$ is used to control the weighting.

The initial mode estimates may be chosen to be the original data samples, although optimized implementations may choose a smaller subset of points for increased performance. The number of points that converge to the same mode can be used as a rough estimate of the density at the mode.

4.2 Body part classification

In body part classification (BPC)[22], a random forest classifier is used to classify pixels in depth images to body parts. At the split nodes, the classification tree uses features that perform simple depth comparisons in the neighborhood of a pixel as follows

$$\Omega(I, x, u, v) = d_I\left(x + \frac{u}{d_I(x)}\right) - d_I\left(x + \frac{v}{d_I(x)}\right) \tag{4.2}$$

where $d_I(x)$ is the depth at pixel x in image I, and u and v are pixel offsets chosen at random. Due to the normalization by $\frac{1}{d_I(x)}$, the features are invariant to 3D translation. At the leaves of the tree, a distribution over body part labels is stored. Training and evaluating the forest is explained in detail in the following sections.

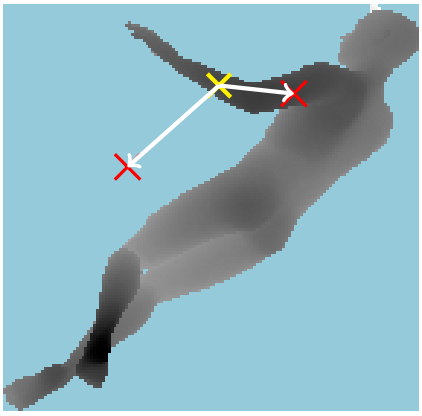

Figure 4.1: Example of a depth feature

Figure 4.1 shows an example depth feature. The yellow cross is the pixel being classified, while the red crosses indicate the pixel offsets from which the depth samples are taken. Since background pixels are given a large depth value, the feature in the image would return a large response.

In [22], the authors explain that although the individual features give only a weak signal about which body part the pixel belongs to, they are sufficient to accurately distinguish between body parts when several features are combined in a random decision forest.

4.2.1 Training

Each tree in the BPC random forest is trained individually on a subset of the training data. The training data is split into equally sized parts by

uniformly sampling from the complete set. For training the tree structure, a set of foreground pixel coodinates x is chosen from each image I by rejection sampling. For each split node, the feature offsets u and v, and a threshold τ are learned. The set of training samples $Q = \{(I, x)\}$ is then split into left and right subsets as follows

$$Q_L(u, v, \tau) = \{(I, x) | \Omega(I, x, u, v) < \tau\} \tag{4.3}$$

$$Q_R(u, v, \tau) = \{(I, x) | \Omega(I, x, u, v) >= \tau\} \tag{4.4}$$

The training then recurses for the subsets Q_L and Q_R. The feature offsets u and v, and the threshold τ are chosen by maximizing information gain G within a set of randomly sampled candidates

$$G(u, v, \tau) = H(Q) - \sum_{s \in \{L,R\}} \frac{|Q_s(u, v, \tau)|}{|Q|} H(Q_s(u, v, \tau)) \tag{4.5}$$

where H(Q) is the Shannon entropy computed from the normalized histogram of body part labels $l_I(x)$ over all training samples $(I, x) \in Q$. Shannon entropy is defined as

$$H(Q) = -\sum_{l \in L} p(l) \log p(l) \tag{4.6}$$

where L is the set of all body part labels, and p(l) is the normalized histogram of body part labels within the training samples Q. At the leaves of the tree, the histogram of body part labels over all pixels that reached the leaf is stored.

4.2.2 Evaluation

The trained model is used to classify pixels in an input depth image to body parts. Joint proposals are then formed by clustering the classified pixels. Finally, a learned joint-specific push-back offset is applied to alleviate the issue that the pixel positions lie on the surface of the body while the joint positions are inside the body.

More precisely, each pixel (I, x) in the test image I is passed down each tree in the random decision forest F. The final classification for the pixel (I, x) is produced by averaging together the body part distributions $P_t(l|I, x)$ stored at the leaves of the trees

$$P(l|I, x) = \frac{1}{|F|} \sum_{t \in F} P_t(l|I, x) \tag{4.7}$$

Some of the body parts may be merged together to localize joints. Joint proposals are formed by clustering the world-space coordinates of classified pixels using mean shift clustering[8] with a weighted Gaussian kernel. The density estimator is defined as follows

$$p_l(\hat{x}) \propto \sum_{i=1}^{N_p} w_{ic} \exp\left(-\left\|\frac{\hat{x} - \hat{x_i}}{b_l}\right\|^2\right) \tag{4.8}$$

where N_p is the number of pixels in the image, $\hat{x}$ is a 3D world-space coordinate, $\hat{x_i}$ is the world space coordinate of image pixel (I, x) and b_l is a part-specific bandwidth. The optimal values for b_l are learned through hyperparameter optimization.

The pixel weighting w_{ic} is defined as

$$w_{il} = P(l|I, x_i) \cdot d_I(x_i)^2 \tag{4.9}$$

where the first term is the body part probability, and the second term estimates the world-space surface area of the pixel. The second term ensures depth invariance. For example, a pixel that is further away from the camera covers more area than a pixel close to the camera, and thus should contribute more to the density estimate.

The final confidence-weighted joint proposals are the set of modes returned by mean shift, pushed back by a joint-specific push-back amount, and weighted by the total sum of the weights reaching each mode.

4.3 Offset joint regression

In offset joint regression (OJR)[10], the leaf nodes of the random decision forest store a distribution over relative offsets to each joint. This allows regression of joint positions directly without the intermediate body part representation which was used in BPC.

4.3.1 Training

In [10], the authors propose training the tree structure separately from the leaf predictors. The authors determined that the classification objective used in BPC worked well as a proxy for training the tree structure for regression.

When training the leaf predictors, a set of relative votes R_{lj} is collected for each leaf node l and for each joint j. Each training pixel that reaches leaf node l contributes a single relative vote for each joint j to the set R_{lj}. A

relative offset is a vector from the pixel's world-space position to the ground truth joint position.

Since the training datasets are very large, all offsets seen at training time cannot be efficiently stored in memory, so a compressed representation of the offset distribution is required. Reservoir sampling is used at training time to collect an unbiased sample of C offsets. With reservoir sampling, the list of relative votes R_{lj} is first filled up to size C. After this, new offsets replace a random vote in the list with probability $\frac{1}{n_{lj}}$, where n_{lj} is the total number of votes encountered so far for leaf l and joint j.

The distribution for each (l, j) is then represented by the K_m largest weighted modes found using mean shift clustering with a Gaussian kernel, using the following density estimator

$$p_j(z') \propto \sum_{z \in R_l j} \exp\left(-\left\|\frac{z' - z}{b^*}\right\|^2\right) \tag{4.10}$$

where b^* is a shared training-time aggregation bandwidth. Modes that exceed a per joint distance threshold λ_j may be discarded at this point, although this can also be done at test-time.

The output of the leaf node regression model training is then a set of weighted modes V_{lj} for each leaf node l and joint j found using mean shift. The weight of each mode is the number of relative votes that reached the mode.

4.3.2 Evaluation

In OJR, the evaluation phase can be split into vote collection, vote subsampling and vote clustering steps. In the vote collection stage, we collect a set of *absolute* votes V_j for each joint j. We pass each pixel (I, x) in the test image I down the random decision forest and aggregate the modes stored in the leaves. To get the set of *absolute* votes from the *relative* modes stored at the leaves, we add the world-space position of the pixel to each relative vote. Each vote weight is also multiplied by $d_I(x)^2$, which is the depth at pixel (I, x). This ensures depth invariance as there are more votes closer to the camera.

The vote collection step typically produces some thousands of absolute votes for each joint. For efficiency reasons, the set of absolute votes is subsampled. The approach preferred by the original authors was to take the top K_s weighted votes for each joint. This forms the subsampled set of absolute votes Z_j.

The joint proposals are formed by clustering the absolute votes using mean shift on the following density estimator

$$p_j(z') \propto \sum_{(z,w)\in Z_j} w \cdot \exp\left(-\left\|\frac{z'-z}{b_j}\right\|^2\right) \tag{4.11}$$

where (z, w) is a weighted absolute vote and b_j is a joint-specific aggregation bandwidth.

4.4 Generation of training data

Since hand-labeling body parts and positions of skeletal joints to real-world data is tedious, Shotton et al.[22] made use of computer generated data for training their model. The authors built a rendering pipeline, which outputs computer-generated images of humans with simulated variation in pose and appearance. To train the body part classifier in [22], the authors used a total of 900,000 generated training images. Similarly in [10], R. Girshick et al. used a synthetic training set consisting of 300,000 images to train the model.

Chapter 5

Implementation

In previous chapters we introduced recent machine learning methods that can be used to solve the human pose estimation problem. In this chapter, the implementation of our climbing pose estimation system is described, which is based on the previously introduced methods. We first present a high level overview of the components of the system. The subsequent sections then introduce each component in more detail.

5.1 Overview

We implement the offset joint regression method (OJR) introduced by R. Girshick et al.[10], which was described in detail in chapter 4. OJR estimates skeletal joint positions from single depth images. Like the original authors, we use the body part classification objective for training the structure of the random forest, while "retro-fitting" regression models at the leaves.

Our implementation consists of three components. The training data generator generates synthetic training data for the purpose of training the model, the trainer trains the model with the generated training data, and finally the evaluator uses the trained model to estimate poses from given depth images. We chose to implement all of the components mostly in C++ using OpenMP for parallel processing over multiple CPU cores. For the performance critical, highly parallelizable parts of the evaluator, we utilize the parallel processing power of modern GPUs using the Nvidia CUDA GPU programming toolkit.

5.2 Training data generation

In order to produce a large dataset of potentially thousands of training images, we employ the same technique used by Shotton et al. in [22]. We create a pipeline from which we can sample a large number of computer-generated training images. In order for the synthetic images to be useful in training a model that can be used to make predictions from real world data, these synthetic images should resemble real images, and they should contain similar variation that is expected to occur in the real world data.

In order to be able to generate training images with realistic variation in pose, we use a Perception Neuron motion capture system to capture various climbing poses on a climbing wall. The motion capture data is then retargeted onto a HumanIK skeleton, which is a standardized animation rig for animating human 3D meshes. The retargeted set of animations is compatible with human meshes generated using MakeHuman, which is a free software capable of automatically generating varied human 3D meshes. For the purpose of training data generation, we generate a generic human male model using MakeHuman. We assign a texture map to the model which divides the surface area to 31 body part colors.

Our training data generation pipeline imports the retargeted animations and the generated human model. We use a pose parameter vector with 3 translational and 45 rotational degrees of freedom. We sample each frame of the animation, and fit a multivariate Gaussian distribution to the samples.

For each image we want to generate, we sample a random pose from the distribution. For each sampled pose, a camera is placed so as to simulate the placement of the depth sensor in front of the a climbing wall. This means the human model is facing away from the camera, unlike in the typical living room scenario, where the user is facing towards the camera. For additional variance, we translate and rotate the model in the scene uniformly at random within reasonable ranges typical in our use case. We also flip the pose half of the time. We then capture synthetic depth images, body part labels and joint positions from the viewpoint of the camera.

5.3 Training the model

We train the structure of the decision trees in the forest separately from the leaf predictors. The first training pass trains the structure of the trees, while a second pass then trains prediction models for the leaf nodes.

5.3.1 Training the tree structure

Although we train the leaf predictors for regression, we choose to train the tree structure by optimizing a BPC classification objective. This was found to produce the best prediction results in [10]. For this purpose, we define a set of 31 body parts. During the synthetic training data generation, body part images are rendered in addition to depth images, where each pixel value corresponds to a body part index.

As in [22], we train each tree individually on a subset of the training data. For training the tree structure, we choose 2000 foreground pixel coodinates x from each image I by rejection sampling.

We use a depth-first implementation of random forest training as shown in algorithm 1. The decision tree could also be trained breadth-first by training each node at the current tree depth before moving to the next depth level. Theoretically the set of training images could then be iterated through only once for each depth, while the depth-first implementation iterates through the images for each node. Iterating through the training data only once is desirable if random access to training data is expensive. Breadth-first training can also be parallelized more effectively. However, breadth-first training requires more memory the deeper the tree training descends, as a set of histograms needs to be stored for each leaf node and each combination of candidate features and thresholds. To keep memory usage constant, the set of leaf nodes can be split into subsets of constant size. The subsets can then be trained separately, with the tradeoff that the number of iterations through the training images is increased.

We experimented briefly with breadth-first training, but we did not find it beneficial for the purposes of the experiments in this paper. Since we are using a small training dataset, we can fit the complete dataset into memory, avoiding slow random access due to streaming from disk. We are also training the trees using a single CPU, which does not benefit greatly from the parallelization potential of breadth-first training. For a distributed computing implementation or larger datasets, a breadth-first training approach would be necessary.

Our depth-first training algorithm evaluates candidate features and thresholds for each node of the tree starting from the root, choosing the best candidate according to equation 4.5. The training data is then split into left and right subsets according to equation 4.3, and the algorithm then recurses for these subsets.

A naive trainer would iterate through the training data for each candidate feature and threshold pair in order to compute the information gain for the split. The feature response r would be calculated multiple times, which

Algorithm 1 Depth-first classification tree training optimized to evaluate multiple candidate thresholds per feature

Require: Training dataset S, number of candidate features K_f, number of candidate thresholds K_t, tree depth D

function TrainDepthFirst(S, d)
  if $d =$ D **then** ▷ Reached desired tree depth
    return CreateLeafNode
  end if
  for $i = 0 \dots K_f$ **do**
    $u, v \leftarrow$ SampleOffset
    $t_i \leftarrow$ SampleThreshold, $i = 0, \dots, K_t - 1$
    $S_i \leftarrow$ InitHistogram, $i = 0, \dots, K_t$
    for all $(I, x) \in S$ **do**
      $r \leftarrow$ GetFeatureResponse(I, x, u, v)
      Select k according to equation 5.1
      AccumulateHistogram$(H_k, l_I(x))$
    end for
    for $i = 0 \dots K_t - 1$ **do**
      $H^L \leftarrow \sum_{j=0}^{i} H_j$ ▷ Calculate cumulative histogram
      $H^R \leftarrow \sum_{j=i+1}^{K_t} H_j$
      $I \leftarrow$ information gain for split (H^L, H^R) (see equation 4.5)
      if $I > I^*$ **then** ▷ Found a new best split
        $I^* \leftarrow I$
        $\theta^* \leftarrow (u, v, t_i)$
      end if
    end for
  end for
  $S^L, S^R \leftarrow$ PartitionTrainingSamples(S, θ)
  $N^L \leftarrow$ TrainDepthFirst$(S^L, d + 1)$
  $N^R \leftarrow$ TrainDepthFirst$(S^R, d + 1)$
  return CreateSplitNode(N^L, N^R))
end function

is costly. Criminisi et al.[9] suggest an optimization to this by evaluating several candidate thresholds simultaneously for each candidate feature. For each candidate feature, a set of candidate thresholds is sampled at once, and a histogram is initialized for each candidate threshold. The training data is then iterated through, and the feature is evaluated for each training sample (I, x). Each training sample accumulates a histogram H_k, where the bin index k is chosen as follows

$$k = \begin{cases} 0, & \text{if } r < t_0 \\ 1, & \text{if } t_0 \leq r < t_1 \\ \dots & \\ K, & \text{if } t_{K-1} \leq r \end{cases} \tag{5.1}$$

After the histograms have been accumulated, we iterate through each threshold. Each iteration i evaluates the information gain of a split with parameters $\theta = (u, v, t_i)$. We get the histogram for the left and right subsets from the cumulative histograms $H^L = \sum_{j=0}^{i} H_j$ and $H^R = \sum_{j=i+1}^{K} H_j$. From these the information gain can be evaluated according to equation 4.3.

The offsets u and v and the thresholds t_i are sampled from uniform distributions. The offsets are sampled from the range $[-122, 122]$, and the thresholds from the range $[-1, 1]$. These values were chosen in accordance with [22], as our use case is similar. They could be optimized further using grid search to maximize the accuracy of the model, but since the tree structure training is very expensive, we chose not to do this in the scope of this thesis.

To keep memory requirements constant, we also partition the set of training data into left and right subsets in-place. With in-place partitioning, the training data is reordered in memory so that the left and right subsets are separated. This is feasible since the algorithm operates on recursively smaller and smaller subsets. By using in-place partitioning, we avoid allocating new lists for the subsets on each split of the tree.

We choose to sample $K_f = 2000$ candidate features and $K_t = 50$ candidate thresholds uniformly. We train the trees to depth $D = 20$. These hyperparameter values were also used by Shotton et al. in [22]. For slightly improved model accuracy, the distribution of features and thresholds can be learned by training once with uniform sampling and recording the distribution of chosen feature and threshold values. The learned distributions can then be used to improve the sampling of features and candidates in later training passes.

5.3.2 Training the leaf predictors

After the tree structure has been trained, we can train the leaf predictors separately. We train an offset joint regression model[10] for predicting joint positions directly from input depth images. The training process is summarized in algorithm 2. We choose $C = 128$ for the reservoir size, and store $K_m = 2$ highest confidence modes as in [10]. According to the authors, using a larger reservoir or storing more modes did not improve the prediction accuracy significantly.

Algorithm 2 Training leaf predictors for offset joint regression

Require: training dataset S

function TRAINOJRLEAFPREDICTORS(S)
  Initialize list of relative votes R_{lj} for each leaf l and joint j
  for all $(I, x) \in S$ **do**
    $J \leftarrow$ set of ground truth joint positions from image I
    $x_w \leftarrow$ world coordinate of pixel (I, x)
    $l \leftarrow$ forest leaf node reached by pixel (I, x)
    for all joints j **do**
      $\Delta x = J_i - x_w$
      Store Δx in R_{lj} with reservoir sampling
    end for
  end for
  for all leaf nodes l **do**
    for all joints j **do**
      Cluster votes in R_{lj} using mean shift
      Store top K_m weighted modes in leaf node l
    end for
  end for
end function

5.4 Hyperparameter optimization

The model depends on various hyperparameters. For the training-time hyperparameters, we choose to use the same values that were chosen by the authors in [10], because training the forest takes a significant amount of computing time. The test-time hyperparameters consist of the per-joint aggregation bandwidths b_j and vote length thresholds λ_j.

Like Shotton et al. in [10], we optimize the test-time hyperparameters of the leaf regression model using grid search. Since the test-time per-joint

hyperparameters b_j and λ_j are independent for each joint j, we can optimize them independently, reducing the dimensionality of the grid search considerably. We optimize b_j in the range $[0.01\,\mathrm{m}, 0.2\,\mathrm{m}]$ and λ_j in the range $[0.025\,\mathrm{m}, 0.505\,\mathrm{m}]$.

The hyperparameter optimization maximizes mean average precision over joints on a validation set containing 1000 images. The mean average precision metric is explained in detail in chapter 6. The optimized values are summarized in table 5.1.

Joint	b_j	λ_j
Head	0.05	0.065
Neck	0.03	0.065
Left Arm	0.05	0.105
Right Arm	0.05	0.065
Left Forearm	0.05	0.145
Right Forearm	0.07	0.345
Left Wrist	0.11	0.265
Right Wrist	0.11	0.185
Left Hand	0.09	0.105
Right Hand	0.09	0.305
Left Leg	0.05	0.105
Right Leg	0.07	0.105
Left Knee	0.05	0.225
Right Knee	0.05	0.105
Left Foot	0.05	0.425
Right Foot	0.05	0.465

Table 5.1: Test-time hyperparameters optimized by grid search

5.5 Evaluating the model

Once we have a trained model, we can use it to draw joint estimates for previously unseen test images. OJR evaluation can be split into vote collection, vote subsampling and vote clustering steps. In the vote collection step, the decision forest is evaluated to collect a set of votes for each joint. In the vote subsampling step, the set of votes for each joint is subsampled for the purpose of speeding up clustering, by taking only the top K_s weighted modes. For the experiments in this thesis, we chose $K_s = 50$, which was determined to offer a good tradeoff between speed and accuracy by Girshick et al. in [10]. In the vote clustering step, the joint votes are clustered using mean

shift, producing the final set of weighted joint proposals for each joint. We detail the OJR evaluation process in algorithm 3.

Algorithm 3 Evaluating the OJR model

Require: test image I

function EVALUATEOJR(I)
  Initialize list of absolute votes V_j for each joint j
  for all pixels $x \in I$ **do**
    $x_w \leftarrow$ world coordinate of pixel (I, x)
    $l \leftarrow$ forest leaf node reached by pixel (I, x)
    for all joints j **do**
      for all relative votes $(\Delta x, w) \in R_{lj}$ **do**
        $x = \Delta x + x_w$
        $w_2 = w + d_I(I, x)$
        Store x in V_j with weight w_2
      end for
    end for
  end for
  Subsample the set of votes V_j by discarding all but the top K votes
  Initialize list of weighted joint proposals Z_j for each joint j
  for all joints j **do**
    Cluster votes in V_j using mean shift
    Store the weighted modes found by mean shift in Z_j
  end for
  return joint proposals Z_j for each joint j
end function

5.5.1 Evaluating on the GPU

To speed up the model evaluation phase, we implement the evaluation on a GPU using Nvidia's CUDA GPU computing framework.

On the CPU, the vote collection step is by far the slowest part of the algorithm even when distributed over multiple cores, taking more than 20 ms of computing time for each frame on our test system with a quad-core Intel CPU. Most of the time is spent traversing the decision trees and evaluating the depth features. For comparison, vote subsampling and clustering combined take about 4 ms on average. Since tree traversal and depth feature evaluation is done independently for each pixel in the input depth image and can thus be trivially parallelized, this problem is a good fit for GPU optimization. On the GPU, we assign a thread for each pixel, which traverses

the forest and collects the joint votes at the leaves into a list into the GPU's global memory. We use an atomic add operation to keep track of the list index.

We chose not to implement vote subsampling and clustering on the GPU. Vote subsampling requires k-selection from a small array for each joint, which is inefficient on the GPU. Instead, we accumulate all the votes into an array, and then transfer the complete array from GPU to CPU memory. We then proceed with vote subsampling and clustering on the CPU. This seems to be reasonably efficient despite the somewhat heavy memory transfers.

Chapter 6

Evaluation

In this chapter we evaluate our human pose estimation implementation with a synthetic test dataset.

6.1 Joint prediction accuracy

To evaluate the quality of the predictions of the trained model, we run our method on a set of synthetic test data generated using the same method we use to generate training data, which was described in chapter 5. We use a set of 1000 images for evaluating the model. We use a mean average precision (mAP) metric over all joints as in [22]. This is simply the mean of average precision scores for each joint. Average precision is chosen as the evaluation metric, because it considers not just the accuracy but also the confidence of the joint proposals.

In practice, we gather all retrieved joint position proposals for joint j into a list sorted by confidence, and then calculate average precision as

$$AP_j = \frac{\sum_{k=1}^{N} P(k)}{TP + FN} \tag{6.1}$$

where N is the number of joint proposals for the joint j, $P(k)$ is the precision at cut-off point k in the sorted list of joint proposals, TP is the total number of true positives, and FN is the total number of false negatives. The precision at cut-off point k is equivalent to the ratio of true positives in the set of k highest confidence joint proposals.

If a joint proposal is closer than D_1 to the ground truth position, it counts as a true positive. Other proposals within the range D_1, as well as proposals outside of the range count as false proposals. Joints that are not occluded, but receive no joint proposals count as false negatives. Figure 6.1 shows the

effect of true positive distance on the mean average precision score obtained on our test set containing 1000 images. With the true positive distance $D_1 = 5$ cm mean average precision is 0.907. Table 6.1 lists the per-joint average precision scores we obtain with our optimized model using a test set of 1000 images.

Joint	AP
Head	0.999
Neck	0.999
Left Arm	0.994
Right Arm	0.991
Left Forearm	0.916
Right Forearm	0.913
Left Wrist	0.804
Right Wrist	0.804
Left Hand	0.829
Right Hand	0.804
Left Leg	0.979
Right Leg	0.974
Left Knee	0.867
Right Knee	0.865
Left Foot	0.882
Right Foot	0.900

Table 6.1: Per-joint average precision scores with the optimized model

We also use a second metric to evaluate the performance of our system. Similarly to Taylor et al. in [26], we calculate the ratio of test images that have all highest confidence joint predictions within a range D_2 from the ground truth. Occluded joints count as missed predictions, but joints outside of the image do not. This more challenging metric is intuitive and gives a good idea of the difficulty of the problem, but does not take into account the confidence of the joint proposals. We choose $D_2 = 20$ cm, and achieve a score of 33.3% on our 1000 image test dataset.

6.2 Qualitative results

In figure 6.2 we show examples of the inferred body parts (BPC) and highest confidence joint proposals (OJR) obtained on a small set of synthetic test data. The input depth image is shown on the left alongside each result. Figure 6.3 shows results on a few selected frames of real world data captured

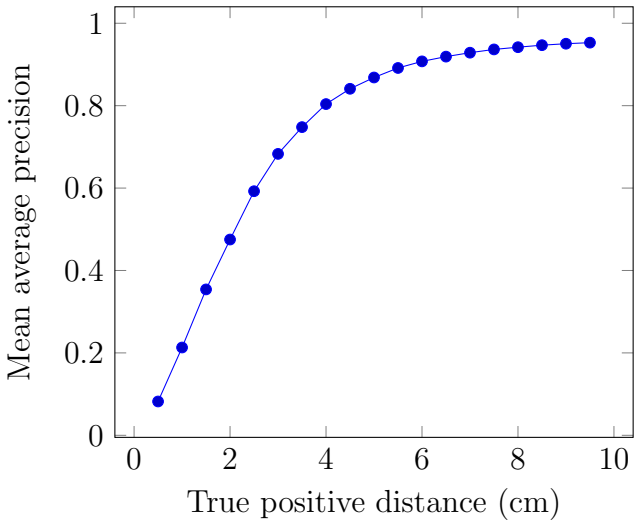

Figure 6.1: Mean average precision as a function of the true positive distance D_1

using a Microsoft Kinect 2 sensor. Figure 6.4 shows example cases where one of the feet joints is not correctly detected. Also, some occluded joints are placed in unrealistic positions.

Although we use the OJR method which directly regresses joint proposals without the intermediate body part representation, our random forest model was trained for body part classification. We show the inferred body parts, because they effectively visualize the random forest's ability to localize the joints. Note that the colors of the body parts and the colors of the joints do not correspond to each other in the images.

As the method does not enforce kinematic constraints, outliers in the joint votes occasionally result in unrealistic joint proposals that can be seen in the images. For example, hand joint proposals may be placed in the legs. However, the confidence weights assigned to the failed joint predictions are typically low, which keeps the mean average precision high. For the sake of clarity, we chose to only visualize the highest confidence joint proposals. Often even if the highest confidence proposal is not correct, one of the other proposals is.

Figure 6.2: Examples of inferred body parts and highest confidence joint proposals on a random selection of synthetic test data

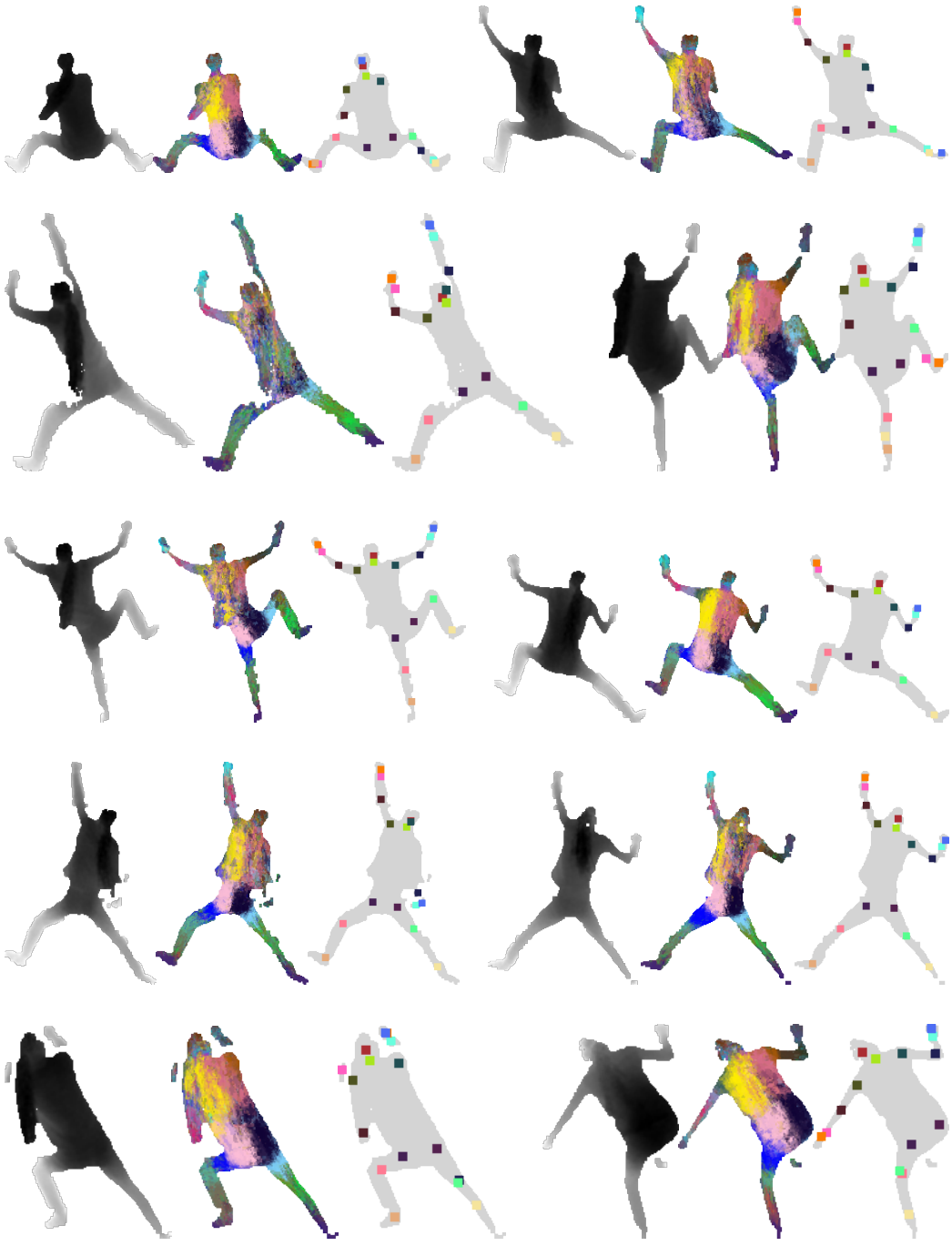

Figure 6.3: Examples of inferred body parts and highest confidence joint proposals on selected frames of real world data

Figure 6.4: Examples of failure modes where one of the feet joints is not correctly detected

Chapter 7

Discussion

In this chapter we discuss the performance and pose estimation quality of our implemented method. We also look at potential directions for future research.

7.1 Scalability

As expected, the random decision forest model is extremely fast to evaluate in runtime especially with GPU optimizations, which enables real-time pose estimation. However, training the model takes a considerable amount of computing time, which makes it difficult to tweak the model parameters and evaluate their effects. For this reason, in this thesis we chose to use mostly the same hyperparameter values as in previous work [22][10]. To optimize the training-time hyperparameters, it would be ideal to be able to distribute the workload to a computing cluster, since hyperparameter optimization can be effectively parallelized. However, this was not feasible in the scope of this thesis.

7.2 Importance of training data quality

Our implementation achieves reasonably high mAP scores on our synthetic test dataset of climbing poses. However, qualitative analysis reveals failures especially in less typical cases, such as in poses where the climber's legs are higher than the head, where our implementation often mixes up hands and legs. Augmenting the training data with more examples of such cases may be a feasible approach to improve the accuracy of the model in future work.

The mean average precision scores we obtained are suprisingly high compared to the results presented by Girshick et al. in [10], although our im-

plementation is very similar. This may be due to our dataset being less challenging than the one used by the original authors. Firstly, the motion-captured set of poses that we use for training data generation is fairly limited. Secondly, we account for variation in pose, but not in shape. Ideally we would use several human models with different shape characteristics, ranging from short to tall, thin to fat, and wearing varying clothing. We also do not account for real-world camera characteristics such as noise or missing depth measurements. Perfect background separation is also assumed, which is rarely achieved in real-world cases, where the user is close to a wall.

7.3 Future development

Since the performance of the implemented method depends heavily on the quality of training data, we will look at improving our training data in the future. To extend our dataset of climbing poses, we plan to explore different motion capture setups that allow robustly capturing a performer's poses when they are climbing on a wall. We also plan to improve the training data rendering pipeline by introducing simulated camera characteristics and shape variation.

Although our implementation performs well, further optimizations are possible. Parallelizing the training of the random decision forest and moving it to a GPU could yield a speed boost, which would be desirable especially with larger datasets. For training with larger datasets, it would also be necessary to stream the training data from the hard disk on the fly.

In this thesis, we focused only on *discriminative* or detection-based human pose estimation. We discarded temporal information, and attempted to estimate the pose of a human directly from a single depth image. We also did not enforce any kinematic constraints. The implementation presented here could be extended later by implementing a generative method, which uses the discriminative pose estimate for the initial guess of the body pose, and iteratively refines the body pose estimate taking into account kinematic and temporal constraints. Since trackers based on generative methods are prone to drifting, the methods presented in this thesis are useful for the purpose of reinitializing such a tracker.

Chapter 8

Conclusions

In this thesis, we looked at the problem of estimating the pose of a human on a climbing wall from single depth images. We presented an implementation of a human pose estimation method based on offset joint regression[10], which was previously shown to produce excellent results in a living room scenario, where the user is standing in front of the depth sensor. Offset joint regression uses a random forest regression model, which estimates skeletal joint positions directly from single depth images.

We trained the pose estimation model on a set of synthetic, computer-generated training data. To generate the training data, we built a rendering pipeline, which poses a human 3D mesh and renders depth images from typical camera angles. Ground truth joint positions and body parts are also captured for the purpose of training the model.

We implemented the model using C++. To accelerate the model evaluation, we implemented the evaluation using Nvidia CUDA GPU computing framework. For training the model, we used an optimized single CPU implementation. On synthetic data, we achieved good joint prediction results comparable with the results obtained by the original authors in [10]. The model also generalizes to real world data reasonably, although our training data does not capture all the variation seen in real world depth images. However, since the method does not enforce any kinematic constraints, the joint proposals may not always form a realistic skeleton.

In future work, generalization could potentially be improved by simulating the variation seen in real world depth images better in the synthetic training data. An alternative method such as Vitruvian manifold[26] could be used to enforce kinematic constraints. When tracking body pose over time, temporal constraints could also be utilized for more precise tracking.

Bibliography

[1] Amit, Y., and Geman, D. Randomized inquiries about shape; an application to handwritten digit recognition. Tech. rep., University of Chicago, November 1994.

[2] Autodesk, Inc. 3D Character Animation Software — MotionBuilder — Autodesk. `http://www.autodesk.com/products/motionbuilder/overview`. Accessed 29 Dec 2015.

[3] Autodesk, Inc. 3d Modeling & Rendering Software — 3ds Max 2016 — Autodesk. `http://www.autodesk.com/products/3ds-max/overview`. Accessed 29 Dec 2015.

[4] Breiman, L. Bagging predictors. *Machine Learning 24*, 2 (1996), 123–140.

[5] Breiman, L. Random forests. *Machine Learning 45*, 1 (2001), 5–32.

[6] Breiman, L., Friedman, J., Stone, C., and Olshen, R. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[7] Chen, L., Wei, H., and Ferryman, J. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters 34*, 15 (2013), 1995 – 2006. Smart Approaches for Human Action Recognition.

[8] Comaniciu, D., and Meer, P. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 24*, 5 (2002), 603–619.

[9] Criminisi, A., and Shotton, J., Eds. *Decision Forests for Computer Vision and Medical Image Analysis*. Springer, 2013.

[10] Girshick, R., Shotton, J., Kohli, P., Criminisi, A., and Fitzgibbon, A. Efficient regression of general-activity human poses from depth images. In *ICCV* (October 2011), IEEE.

[11] Greff, K., Brandão, A., Krauss, S., Stricker, D., and Clua, E. A comparison between background subtraction algorithms using a consumer depth camera. In *VISAPP (1)* (2012), pp. 431–436.

[12] Isard, M., and Blake, A. Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision 29*, 1, 5–28.

[13] Kajastila, R., and Hämäläinen, P. Augmented climbing: Interacting with projected graphics on a climbing wall. In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems* (New York, NY, USA, 2014), CHI EA '14, ACM, pp. 1279–1284.

[14] Lakshminarayanan, B., Roy, D. M., and Teh, Y. W. Mondrian forests: Efficient online random forests. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3140–3148.

[15] MakeHuman Team. MakeHuman — Open source tool for making 3d characters. `http://www.makehuman.org/`. Accessed 29 Dec 2015.

[16] Moeslund, T. B., Hilton, A., and Krüger, V. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding 104*, 2 (2006), 90–126.

[17] Noitom Ltd. Perception Neuron by Noitom. `https://neuronmocap.com/`. Accessed 27 Jan 2015.

[18] Open Asset Import Library Team. Official Open Asset Import Library Repository. `https://github.com/assimp/assimp`. Accessed 29 Dec 2015.

[19] Pons-Moll, G., Taylor, J., Shotton, J., Hertzmann, A., and Fitzgibbon, A. Metric regression forests for correspondence estimation. *IJCV* (August 2015).

[20] Ren, S., Cao, X., Wei, Y., and Sun, J. Global refinement of random forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 723–730.

[21] Salzmann, M., and Urtasun, R. Combining discriminative and generative methods for 3d deformable surface and articulated pose reconstruction. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (2010), IEEE, pp. 647–654.

[22] Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from a single depth image. In *CVPR* (June 2011), IEEE.

[23] Sigal, L., Balan, A., and Black, M. J. Combined discriminative and generative articulated pose and non-rigid shape estimation. In *Advances in neural information processing systems* (2007), pp. 1337–1344.

[24] Sigrist, R., Rauter, G., Riener, R., and Wolf, P. Augmented visual, auditory, haptic, and multimodal feedback in motor learning: A review. *Psychonomic Bulletin & Review 20*, 1 (2013), 21–53.

[25] Sun, M., Kohli, P., and Shotton, J. Conditional regression forests for human pose estimation. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (2012), IEEE, pp. 3394–3401.

[26] Taylor, J., Shotton, J., Sharp, T., and Fitzgibbon, A. The vitruvian manifold: Inferring dense correspondences for one-shot human pose estimation. In *Proc. CVPR* (June 2012), IEEE.

[27] Wei, X., Zhang, P., and Chai, J. Accurate realtime full-body motion capture using a single depth camera. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 188.