A Hybrid Method for 3D Pose Estimation of Personalized Human Body Models

Ammar Qammaz

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Antonis Argyros*

This work has been performed at Computer Vision and Robotics Laboratory (CVRL) of the Institute of Computer Science (ICS) of the Foundation for Research and Technology–Hellas (FORTH) .

The work has been supported by a scholarship from the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS) and partly funded through the EU H2020 ICT projects Co4Robots and ACANTO.

University of Crete
Computer Science Department

A Hybrid Method for 3D Pose Estimation of Personalized Human Body Models

Thesis submitted by
Ammar Qammaz
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: ______________________
Ammar Qammaz

Committee approvals: ______________________
Antonis Argyros
Professor, Thesis Supervisor

____
George Papagiannakis
Associate Professor, Committee Member

____
Xenophon Zabulis
Principal Researcher, Committee Member

Departmental approval: ______________________
Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, October 2018

A Hybrid Method for 3D Pose Estimation of Personalized Human Body Models

Abstract

We propose a new hybrid method for 3D human body pose estimation based on RGBD data. We treat this as an optimization problem that is solved using a stochastic optimization technique. The solution to the optimization problem is the pose parameters of a human model that register it to the available observations. Our method can make use of any skinned, articulated human body model. However, we focus on personalized models that can be acquired easily and automatically based on existing human scanning and mesh rigging techniques. Observations consist of the 3D structure of the human (measured by the RGBD camera) and the body joints locations (computed based on a discriminative, CNN-based component). A series of quantitative and qualitative experiments demonstrate the accuracy and the benefits of the proposed approach. In particular, we show that the proposed approach achieves state of the art results compared to competitive methods and that the use of personalized body models improve significantly the accuracy in 3D human pose estimation.

Μια υβριδική μέθοδος 3Δ οπτικής παρακολούθησης εξατομικευμένων μοντέλων του ανθρώπινου σώματος

Περίληψη

Προτείνουμε μια νέα υβριδική μέθοδο για την τρισδιάστατη παρακολούθηση της στάσης του ανθρώπινου σώματος η οποία βασίζεται στην εκτίμηση δεδομένων που προκύπτουν από κάμερες χρώματος και βάθους. Αντιμετωπίζουμε το πρόβλημα ως πρόβλημα βελτιστοποίησης που επιλύεται χρησιμοποιώντας μια στοχαστική τεχνική. Τα αποτελέσματα που προκύπτουν από την εκτελούμενη βελτιστοποίηση είναι οι παράμετροι θέσης ενός ανθρώπινου μοντέλου που ταιριάζουν όσο το δυνατόν ακριβέστερα στις διαθέσιμες παρατηρήσεις. Η μέθοδος μας μπορεί να κάνει χρήση οποιουδήποτε τρισδιάστατου μοντέλου ανθρώπινου σώματος για την εκτέλεση της οπτικής παρακολούθησης. Ωστόσο, εστιάζουμε σε εξατομικευμένα μοντέλα που μπορούν εύκολα να αποκτηθούν χάρη στις υπάρχουσες σύγχρονες τεχνικές τρισδιάστατης ανακατασκευής.

Οι παρατηρήσεις συνίστανται στην τρισδιάστατη δομή του ανθρώπου (που αποτυπώνεται από την κάμερα) και τις θέσεις των αρθρώσεων του σώματος (που υπολογίζονται με βάση ένα συνελικτικό νευρωνικό δίκτυο). Μια σειρά από ποσοτικά και ποιοτικά πειράματα καταδεικνύουν την ακρίβεια και τα οφέλη της προτεινόμενης προσέγγισης. Συγκεκριμένα όπως αποδεικνύουμε, η προτεινόμενη προσέγγιση επιτυγχάνει σημαντική ακρίβεια παρακολούθησης σε σχέση με ανταγωνιστικές μεθόδους και η χρήση εξατομικευμένων μοντέλων σώματος βελτιώνει σημαντικά την ποιότητα των αποτελεσμάτων τρισδιάστατης εκτίμησης της ανθρώπινης θέσης και στάσης.

Acknoweledgments

- First and foremost I would like to thank Prof. Antonis Argyros for believing in me and giving me the opportunity to join the FORTH-CVRL group. His trust, his support, his guidance and all the time he generously committed are what made this thesis possible.
- I would like to sincerely thank Damien Michel. He has been an amazing coworker and his expertise on the subject of body detection and body tracking shined every time I needed it. His mature codebase is the basis of the groundtruth used in this thesis. His code along with his insight into solving bugs and diagnosing problems has been invaluable for this work and saved me countless hours of effort.
- I would like to thank Pashalis Padeleris, Nikos Kyriazis and Iasonas Oikonomidis for their pioneering work on the MBV toolkit as well as their continuous support and knowledge sharing every time I needed it.
- I am grateful to Elina Paflioti for standing by me and supporting me through all of the phases from starting to finishing this thesis. Her support was not only emotional but also practical since she was the first person to voluntarily get 3D scanned, helped me with dataset recording, supplementary material editing but most importantly made my life a better life to live these past years.
- My parents, for their encouragement, their selflessness and their financial support in a very harsh economic situation for us as well as our country.
- I would like to thank all of the persons that volunteered and donated their 3D scans to form the dataset used in this work. Dennis Bautembach, Aggeliki Tsoli, Kostas Bacharidis, Iason Oikonomidis, Vassilis Nicodemou, Giorgos Karvounas, Konstantinos Roditakis, Eirini Tsiki, Eleana Solakidou, Dimitra Anastasopoulou, Konstantina Zavitsanou, Giasemi Eptaminitaki, Elena Terezaki, Eleni Kostopoulou, Victoria Manousaki and Kostas Papoutsakis you have my heartfelt gratitude.
- I would like to thank the Foundation for Research and Technology for its financial support and the generous supply of computing and networking resources.
- This work was partially supported by the EU H2020 ICT projects Co4Robots and ACANTO.

Contents

Table of Contents **i**

List of Tables **iii**

List of Figures **v**

Preface **i**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem specification . . . 4
- 1.3 Related work . . . 4
- 1.4 Our approach for a solution . . . 5
  - 1.4.1 Body detection and tracking . . . 5
  - 1.4.2 Providing the tracker with high quality models . . . 6
  - 1.4.3 Fusing the parts to a single framework . . . 7

2 3D Reconstruction **9**
- 2.1 Basic concepts . . . 11
- 2.2 Reconstructing indoor spaces and rigid objects . . . 12
- 2.3 Reconstructing rigid human meshes . . . 13
- 2.4 From rigid human meshes to articulated models . . . 15
  - 2.4.1 Improving blending using dual quaternions . . . 15
  - 2.4.2 Choosing the correct vertex count . . . 16
  - 2.4.3 Representing orientations . . . 17
  - 2.4.4 Choosing a file container for our models . . . 17

3 Human Body Detection **19**
- 3.1 Discriminative methods . . . 19
- 3.2 2D body detectors . . . 21
  - 3.2.1 Cascade based . . . 23
  - 3.2.2 Feature based classifiers . . . 25
  - 3.2.3 PCA based classifiers . . . 25
  - 3.2.4 2D DNN neural networks . . . 25

3.2.5 Uplifting 2D bodies to 3D 28
3.3 3D body detectors 29
3.3.1 3D DNN detectors 29
3.3.2 Random forests 30
3.3.3 Dynamic programming 31

4 3D Tracking **33**
4.0.1 Direct Linear Transform (DLT) 33
4.1 Generative Methods 35
4.1.1 Iterative Closest Point (ICP) 35
4.1.2 Particle Filters (PF) 36
4.1.3 Particle Swarm Optimization (PSO) 36
4.2 3D Tracking with PSO 37
4.2.1 The single object case 38
4.2.2 Articulated objects and occlusions 38

5 Our proposed tracking framework **41**
5.1 Overview of proposed framework parts 41
5.1.1 Human body model 43
5.1.2 Localizing body joints 45
5.1.3 Objective function 45
5.1.3.1 The depth term E_D 45
5.1.3.2 The joints location term E_J 46
5.1.3.3 Incorporating heuristics to the objective function . 47
5.1.3.4 CPU/GPU acceleration for the objective function 48
5.1.3.5 GPU rendering surface optimizations 54

6 Experimental Evaluation **61**
6.1 Ground truth and the need for a synthetic dataset 61
6.2 Ground truth public dataset 65
6.2.1 Color and depth data 65
6.2.2 3D rigged model data 67
6.2.3 DNN 2D joint tracking output 69
6.2.4 Ground truth data 70
6.2.5 Camera intrinsics 70
6.3 Optimization parameters 71
6.4 Evaluation metrics 74
6.5 The experiment system 76
6.6 Experimental results 78
6.6.1 Quantitative results 78
6.6.2 Qualitative results 83
6.6.3 Computational performance 83

7 Discussion **89**
7.1 Our contribution 89
7.2 What can still be improved 89
7.2.1 Live human reconstruction 90
7.2.2 Body/Clothes/Face and Hair separation 90
7.2.3 Multiple person tracking 90
7.2.4 Person plus objects tracking 91
7.2.5 Person plus hands tracking 91
7.2.6 RGB/Stereo Camera support 92
7.2.7 Performance Improvements 92
7.2.8 Different Optimization Schemes 93
7.3 Applications that can benefit from the framework 93
7.4 Epilogue 94

Bibliography **94**

List of Tables

5.1 Limits (in radians) of joint angles of the human body model **H**. Besides Neck and Spine, the joints are coded with two letters. The first (L, R) stands for Left/Right and the second (C, S, E, W, H, K, A) for Collar, Shoulder, Elbow, Wrist, Hip, Knee and Ankle. The last column is the weight of the particular joint (w_i) when calculating scores in the objective function E_J 44

5.2 Average performance measurements of the various code segments of the optimization framework when using a medium quality polygon model. The first three columns refer to the optimizer configuration. Bigger tile sizes and particle numbers ensure better quality at the expense of computing time. Render columns in the table refer to the rendering times which are split to the setup and uploading of the articulated model matrices and the actual rendering. The thrust column refers to the time consumed by CUDA/Thrust to perform scoring. Finally the Neural column refers to the time it takes to score the neural-network part of the objective function (Eq:5.4). It can be concurrently calculated on the CPU while the GPU handles the rendering or happen serially after rendering is done. These two different pipelining options are shown on the last 2 columns. We observe that greater particle numbers mean more sphere comparisons while GPU loads also influence thread scheduling. Neural network computation times also appear to be slightly impacted on larger tile sizes despite not directly performing more computations. Each of the PSO generations takes the sum of the times presented in the last four columns. For each received observation frame we repeat the optimization step depending on the number of PSO generations desired. So rendering tiles sized 64x64 on a 64 particle 20 generation (1280 renderings) configuration we will consume 20 * (2585 + 475 + 874 + 1009) = 98860 microseconds achieving performance of roughly 10 fps. Running the neural network computations on a separate thread can further boost the framerate to 12 fps. 57

5.3 Average performance measurements of the various code segments of the optimization framework when using the raw high polygon model acquired by scanning. As stated in Figure 5.2, the Neural column can be concurrently calculated on the CPU while the GPU handles the other three columns. Each of the PSO generations takes the sum of the times presented in the last four columns. The neural time can be subtracted if we run it on a separate thread instead of a serial execution. For each frame we repeat these steps depending on the number of PSO generations selected so for a 64x64 sized 64 particle 20 generation configuration we will achieve 20 * (2652 + 16829 + 863 + 1064) = 428160 microseconds or roughly 2.3 fps. Running the neural network computations on a separate thread does not produce a meaningful speed-up as in the case of the optimized medium-polygon model since the bottleneck in this case is the time to render the geometry. Of course rendering the raw model on a 64x64 tile is wasteful especially since we know that our camera sensor cannot output observations of good enough quality. The way we achieve this quality during model capture with the same sensor is with extended exposure times for each frame to reduce noise as elaborated in Section 2. An off-line 3D tracker application interested in a very high level of detail should also use a larger tile size to harness the high quality renderings as well as a bigger number of PSO generations to ensure enough optimization budget to converge to a good solution. 58

6.1 Error Δ (in mm) for experiments isolating different terms of the objective function. Rows correspond to the sequences MS, FS, each of which is tracked with the proper model (**M**, **F**, respectively). Columns correspond to different weighting schemes (depth only, joints only, balanced). Boldface indicates best results. 80
6.2 Error Δ for the canonical, baseline PSO versus perturbed PSO for different values of the weighting factor w_D. 81
6.3 The error Δ when the sequences MS, FS are tracked with models **M**, **F**, **P**. Boldface font indicates best results. 82
6.4 The error Δ when the sequences MS_i, FS_i are tracked with models $\mathbf{M_i}$, $\mathbf{F_i}$, **P**. Boldface font indicates best results. 82

List of Figures

1.1 Overview of the proposed approach for 3D human pose estimation. (a) The input to the method is a sequence of RGBD frames. (b) We leverage models that can be acquired easily [102] and yield personalized, parametric, skinned human bodies [21]. (c) We also employ state of the art estimation of 2D human body joints [13]. We define a hybrid approach for fitting the model (b) in the observations (a) given the hints provided by the 2D joints estimation (c). The result of this process is visualized in (d). The gray skeleton is the neural network suggestion and the yellow is the result of the proposed approach. Although the neural network estimation is far from the ground truth, the obtained solution is better because of the high quality model and the utilization of depth information. 8

2.1 Left: Edwin Catmuls digitized hand, circa 1973 Right: A decade later sculpture artists create a physical 3D human model and digitize it, circa 1988 10

2.2 Metric Space of the Voxel Storage, Resolution of the Metric space, Visualization of the TSDF. 13

2.3 Left: A successful reconstruction. Middle : Successful reconstruction but wrong T-Pose by the subject. The arm vertices are connected to the torso, this mesh will exhibit severe artifacts around the arms after the automatic rigging. Right: Subject moved both of his arms during scanning and articulated reconstruction could not repair them. Their volume no longer corresponds to observations and tracking will be adversely impacted 14

2.4 Left: A human body that is being represented with 1845 vertices and 7402 faces. Middle : The same body with 8708 vertices and 34848 faces. Right: The original scanned resolution with 50778 vertices and 203124 faces. We notice a substantial increase in geometry complexity that has much less pronounced impact in the observed silhouette of the 3D model for the T-Pose. However skin weights can be adversely impacted in the step of skeleton rigging from our decimated geometry. In this work we use the full quality model since we are interested in maximum accuracy, there is however space for performance optimization using a less detailed model, that will be faster to render and evaluate. 16

3.1 Top Left: Highlighted solution over an observed frame. With green color we have highlighted the correct location of each of the body joints which is our ground truth. With gray color we have highlighted the joint positions as estimated by the chosen Neural Network Discriminative human detector. Although they are not totally inaccurate they are still relatively far away from the true solutions. With Yellow we can see the joint colors that have been refined using a generative method that are much closer to the correct solution and thus mostly occlude the green markers. Top Right: The observed Depth Map contains both background and foreground. Thus a generative component would find correspondences with other objects in the scene as well as our target, discriminative methods on the other hand given enough training can completely reject background and are much less prone to false positive results. Bottom : A batch rendering of 64 random configurations of a human body that adhere to our movement limits and are close to the previous known configuration. The depth color has been inverted to make the figures more visible. Our configurable 3D body model will be used to form such hypothesis that will be iteratively scored, assesed and refined until we converge to the good solution shown top left. 20

3.2 2D Body detectors work on 2D color or monochrome images and can be broadly taxinomized in two families. The one as seen on the left image return a vector of detected joints. The second is a family of detectors that return 2D bounding boxes that contain detected persons as seen in the picture on the right. 22

3.3 Bad cases of 2D body detections that showcase the importance of the list of detector details explained in 3.2. Left: Incorrect detection on a heavily occluded pose. Middle: False positive detection on the background. Right: False positive detection and false negative detection on our foreground. 23

3.4 An overview of how a HAAR cascade is formed and the wavelets that comprise it. 24

3.5 An overview of how eigenfaces work 25

3.6 Patterns mistakenly identified with very high certainty by deep neural networks. Image taken from [65]. 27

3.7 Image of the 22 layer network from [110]. Blue nodes perform convolutions, red pool inputs and yellow perform softmax operations. 27

3.8 A visualization of the problems encountered if we naively try to directly re-project 2D joint positions back to 3D using the depth map registered to our color frame. This is natural since the depth sensor can only sample points in the surface of the observed volume. Left: This is a very simple case of a cube where we can see the discrepancy of the real center of the cube with the depth measurement we get. Right: For a more complex human model of varying thickness, in the best case all of the 3D points extracted (red color) will lie on the surface of the human volume. In case of self-occlusions or other occlusions the 3D points will be even further away from their true location (green color). 28

3.9 A summary of illustrations from [104] that give us insight on the Random Forest classifier. Left: Visualization of features used by the random forest algorithm which is difference of depth. Right: Learning is facilitated by creating a forest of binary decision trees that use random samples and feature sets. Each decision node is selected to provide maximal information gain at each step. 30

3.10 Left: Random Forest body detection classification accuracy as a function of the training set size. Right: Classification accuracy as a function of tree depth and training set size. 31

4.1 Equation 4.1 visualized assuming a rigid human model. In our case however the model is fully articulated and so the DLT method does not suffice as a method to perform tracking. 34

5.1 A block diagram that gives a connectivity overview for the various modules of the proposed framework. Green arrows are raw input from our sensor, pink arrows symbolize intermediate input provided by a module of the framework, the red arrow is the final output pose. Our input source is an RGBD Kinect sensor that provides us with a stream of VGA color and depth images. The DNN 2D Detector, that is described in Chapter 3, only needs the Color stream and works as a black box that provides us with 2D estimations about joint locations. Both the color and depth stream are used by the reconstruction module that is described in detail in Chapter 2. The 4 views of the person are first combined to a rigid mesh that is then, in a subsequent step, converted to a fully skinned articulated body model. This procedure only needs to be performed once. Finally our 3D Tracker receives as input a skinned model, a depth frame and an array of 2D joints estimations and outputs a 3D tracking hypothesis that matches the observation of the Kinect sensor. . . 42

5.2 The level of detail of the meshes rendered in the optimization loop will dramatically increase optimization times, especially on lower end graphics cards. By simplifying the mesh through Quadratic Edge Collapse [34],[49] a method that is readily available on 3D modeling tools like Blender and Meshlab, we can reduce triangle counts and yield significant performance gains. In this figure we can see gradual simplified versions of our original 119484 faces model at 50%,10%,5%,1%,0.5% and 0.1% face count. We can see that our model is over-sampled and over-tessellated especially in the context of our QVGA(320x240) kinect sensor and our QQVGA(160x120) renderings and comparisons. Even when discarding 99.5% of the original vertex count and using only 597 faces (0.5%) the volume occupied by the model is still accurate while much easier to render. However further reducing the face count to 0.1% of the original model the collapsing algorithm starts to impact useful features of the model like the head and feet and devolves to something that barely even resembles a human. A good performance to quality ratio selection is the 1194 face (1%) model. Further tweaking the model in the 0.1-0.5% face range could yield highest performance benefits but at serious expense of tracking accuracy. 56

5.3 Different rendering sizes of a 64 (8x8) PSO particle configuration , displayed overlayed for comparison. Input depth observations come encoded in frames of 640x480 pixels, a resolution which is internally upscaled from the 320x240 depth sensor of the ASUS Xtion/Kinect. However performing comparisons in the original resolution proves very demanding for contemporary GPUs and thus an important technique that enable interactive framerates is to subsample and work with more manageable image dimensions. The largest 1280x960 tile-batch consists of tiles sized 160x120 pixels each that share the same aspect ratio with the input depth frame. However we can further reduce sizes by using "zoomed" projection matrices that clip the rendering viewport and that can also alter the aspect ratio by squeezing it. A 100x100 "zoomed" projection matrix has more resolution than the 160x120 scaled one since almost two thirds of the 160x120 image are empty. This way and combined to a lower polygon mesh we can drastically improve rendering times as well as perform fewer CUDA reduction operations and gain significant speed-ups reaching 9fps framerates when including the neural network times. 59

6.1 Snapshots of the 15 models captured using [102] posing in the default t-pose. The parametric model is also included for comparison in the bottom right. The 4 subjects selected for the experiments are seen in the first row. The selection was based on maximizing their BMI difference to make experiments more representative of the importance of body type. Their BMI measurements are the following from left to right. Male 1 (M_1) is 1.78cm tall and 85kg (26.8BMI), Female 1 (F_1) is 1.58m and 47kg (18.8BMI), Male 2 (M_2) is 1.90cm and 80kg (22.2BMI) and Female 2 (F_2) 1.71m, 67kg (22.9BMI). . 63

6.2 The datasets provided in the public archive are the four used in the comparitive analysis of this work. $M1$ is Ammar, $M2$ is Dennis, $F1$ is Elina and $F2$ is Aggeliki2 65

6.3 Each of the dataset subfolders hold all of the relevant information to allow someone to use the stored data to perform his own experiments. 65

6.4 The datasets folders color (left) and 2_color (right) that hold the two views of our simulated color renderings of our ground truth. . 66

6.5 The datasets folders depth (left) and 3_depth (right) that hold the two views of our simulated 16-bit depth renderings of our ground truth. File managers typically expect 8-bit png images so the 16-bit depths may appear black until processed by a proper PNG reader. 67

6.6 Opening a dataset using the RGBDAcquisition Editor [84] 67

6.7 Viewing the $M1$ dataset using the RGBDAcquisition Editor [84] . 68

6.8 Viewing the $F1$ model using the Blender 3D Editor in its original .dae OpenCollada container as outputed by [21]. 68
6.9 Opening a groundtruth .csv file using LibreOffice Calc, notice that Semicolon is checked as a seperator option 70
6.10 Viewing the ground truth data in Libre Office Calc 71
6.11 Contents of .calib file 72
6.12 Snapshots of the four scanned models used in the experiments. Top row: The **M**ale and **F**emale models that were acquired automatically based on [102]. Bottom row: sample RGB frames that result from the rendering of the **M** and **F** models in the MHAD [116] dataset background, based on the CMU [115] mocap data. 74
6.13 The error Δ as a function of the number of particles and generations in PSO optimization. See text for details. 75
6.14 Snapshot of an SVG log file generated to document a specific frame of a tracking attempt. The svg format proved to be very useful choice for output since it offered both an easily parseable XML file (left), as well as an immediately visualizeable format with highlighted errors and all relevant information conviniently placed. SVG files are natively supported by all graphics programs in Linux and all internet browsers. 76
6.15 Right: Snapshot of an experiment script and the parameters passed to the optimization framework. Left: A sample log directory output for an experiment . For each optimization frame we get an svg file as described in Figure 6.14 as well as the derived foreground for the person, an overlay frame with the proposed pose and finally a color frame where the skeletons are all overlayed on top of the image to better visualizize what happened 77
6.16 Tracking accuracy $A(t)$ for different weights w_D of the depth term E_D in the objective function. 78
6.17 The accuracy $A(t)$ of the proposed method for the cases of balanced depth and joints localization terms (red curves), joints-localization-only term (green curves) and depth-only term (blue curves) for the MS (left) and the FS (right) sequences. 79
6.18 The accuracy $A(t)$ when a sequence is tracked with the proper model (red curves), with a scaled model of the opposite sex (green curves) and with a scaled primitives-based model (blue curves). Results are shown for the MS (left) and the FS (right) sequences. See text for details. 81
6.19 Table 6.4 error Δ measurments visualized along with the models used. 83
6.20 The accuracy $A(t)$ of the Dual RGBD method [57] (green), the OpenNI method [74] (blue) and of the proposed one (red) in all four sequences of the dataset. 84

6.21 Sample qualitative tracking results on the synthetic sequences MS_1. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth. 85
6.22 Sample qualitative tracking results on the synthetic sequences MS_2. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth. 85
6.23 Sample qualitative tracking results on the synthetic sequences FS_1. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth. 85
6.24 Sample qualitative tracking results on the synthetic sequences FS_2. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth. 86
6.25 Sample qualitative tracking results on real sequences with MS_1 subject. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method. 86
6.26 Sample qualitative tracking results on real sequences with FS_1 subject. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method. 87

7.1 The presented framework is very well suited for applications that require detailed tracking. Left: Physiotherapists could use the tracking output to track patient rehabilitation progress after injuries. The parameter space of the problem can be easily constrained to only track the limbs focused by the doctor to better facilitate measurements. Right: Humanoid robot teleoperation could also benefit from the proposed method. The list of angles returned by the tracker can be directly mapped to robot motors and be used to control humanoid robots. 94

Preface

I have put a lot of love, effort and time in writing this thesis and have tried to reflect all of the thought process and design choices made in every stage, from its conception to its implementation. I have also tried to provide context for the various methods discussed and to distil some high-level principles and intuition that governs the body tracking problem.

The range of relevant algorithms and methodologies to the examined domain is great. I have tried not to get carried away and over extend the scope of Chapters. Instead I have selected the most fundamental methods for each of the concepts discussed and have an extensive list of citations that can guide the reader to the appropriate literature in case he requires more detailed information.

For the reader that has a strong computer vision background, is familiar with the topic and wants to quickly go through this work, it has also been published in WACV '18 [82]. The 8 page published paper is a compact version of this thesis and can be used as a brief summary.

Having said this, I would like to thank you in advance for your time if you choose to go through this text and hope that it will prove to be a valuable source of knowledge.

Chapter 1

Introduction

Computers have changed many forms in the last 4 decades. From giant mainframes to personal computers and from game consoles to internet cloud services we are surrounded by computers, billions of mobile phones, embedded devices, desktop PCs and internet servers that facilitate and support most modern activities and services we enjoy in western societies.

The computer industry, both hardware and software, is now mature and streamlined and has penetrated most of the aspects of our modern lives while software and hardware companies top the charts year after year in the lists of most successful businesses worldwide.

We are witnessing more and more of a withdrawal of computers from their traditional static nature to a more ubiquitous versatile platform with laptops, netbooks, tablets and smartphones that are able to be carried anywhere. They no longer have to rely on traditional low fidelity input sources, such as keyboards and mice but can sense the world through cameras, location tracking via GPS and even interact with objects and other machines through NFC, RFID and other wireless protocols.

As processing power increases and becomes readily available in mobile devices computer perception will become a necessity and require more and more robust applications. The next paradigm shift seems to be towards augmented reality, autonomous cars and ultimately home robotics which I personally believe will create huge new markets with profound consequences to human lives.

This work builds upon state of the art computer vision algorithms and aims to provide a unified framework that can facilitate a high quality human tracking technique that can be used in many of those diverse contexts.

1.1 Motivation

In retrospect this MSc thesis is much more focused in contrast to the work I did for my BSc thesis. My BSc thesis had to do with a very broad implementation of a “Guarddog Robot” with everything it entailed on both hardware and software ends. This included designing and building the physical platform, and making every decision and managing every minute detail from motors and batteries to writing all of the software stack that run on top of it. This broad surface of problems to solve however meant that many compromises had to be made and that critical modules like human perception were limited to just a HAAR caascade face detector[118]. This was of course the best thing possible given the input from a stereo pair and the computing power available. However from my experience when building that robot what I believe was the most core component was proper human perception. So although this work can be useful in a variety of scenarios where we move away from traditional input peripherals such as keyboard, mouse and touch interfaces, at the back of my mind I always had the main motivation of creating software that can be used as a plug & play module in a mobile robot platform.

During the period (2016-2018) there has been a boom with a multitude of household robots that are beginning to appear and have been funded through crowd-funding or by large corporations. I will proceede naming a few of them while also listing their funding amount. Of course these sums of money neither directly reflect quality nor can be predictors of success but they are certainly telling about the rising consumer demand. BUDDY [30] which is a small sized robot razed $659,700 in indiegogo. AIDO [36] which is marketed as virtual assistance and telepresence robot razed $891,589. Alpha 2 [37] a small humanoid robot was crowd-funded with $1,407,164 and OLLY [89] a table robot with $327,454, Laika [12] a companion for dogs with $77,428 and the list goes on and on with also many other unsuccessful robots such as Nixie [17] the personal robot, Amy [91] the thoughtful assistant, Robit [52] marketed as "the world's most affordable home robot", TAPIA [61] a talk robot companion, JIBO [9] and many more. Bigger companies such as Google and Amazon have also entered the market cautiously with immobile devices that only use speech as the HCI medium and which are better suited for their search engine results while avoiding all the complexities of a moving device and also beeing able to offer them with a much cheaper price tag. In the Consumer Electronics Show that was held in Las Vegas in January 2018, LG [47] unveiled a line of helper robots for service, hospitality and shopping, Sony relaunched an improved version of its AIBO [107] dog robot while HONDA [33] had a series of indoor robots as well as autonomous all terrain vehicles for outdoor environments. Finally Aeolus [90] debuted an autonomous robot that also features arms in order to be able to handle household chores. All this business activity is not a coincidence and this trend means a great need for vision-related technology such as the one presented here.

Although robots are not yet mainstream consumer products nor household commodities due to their complexity that directly reflects the complexities of indoor environments, a first glimpse of massive use of body tracking HCI technology in consumer products was observed when the kinect sensor debuted in 2010. Despite its original planned use as just an accessory for console gaming, the RGBD camera combined a low price point, good output quality and compatibility with personal computers thanks to the USB connection that made it quickly be adopted both by the scientific community as well as tinkerers and hackers from all over the world. This new sensor rendered older complex multi camera setups optional (if not obsolete) and required no calibration or other setup procedures. Having direct access to a 3D depth frame-buffer with virtually no extra computing power required, in turn made a lot of new applications feasible. At the time of writing this document, this 3D camera technology has been just incorporated in the new flagship series of Iphone-X mobile phones that for now enable user authentication by scanning the geometry of human faces but as user adoption grows there are bound to be used for more and more applications.

All these different projects including the Kinect can be ultimately divided in two broad categories which are mobile vs static devices. Static devices have a much more constrained environment to work with, tend to rely more on voice commands and any vision problems are simplified since the backdrop is always static. This makes perfect sense in order to reduce their cost and make them affordable. Mobile robots however have to operate on a much more complex world and need to be able to accurately track humans in order to interact with them. There is a huge market demand for household robotics especially if one takes into account future demographic projections for the ageing populations of western nations and although we are now seeing the first attempts on such projects there will still be a lot of room for improvement. Vision algorithms are also bound to mature and evolve to become better accommodating as adoption of household robots becomes more widespread.

Even without taking into account the robotics industry, a multitude of other industries can benefit from robust 3D human tracking. Most of them are currently using simple sensors and approximations to partially facilitate the functionality that can ultimately be provided by a robust human tracker. Just to briefly name a few others, movie industries have been long using CGI effects in order to reduce production costs while producing visually spectacular movies. Most of them are based on tracking acquired using motion capture suits in studios with very expensive setups. High quality body tracking using off the shelf devices could dramatically cut down costs and enable smaller movie studios to better compete in the entertainment industry. Security cameras that typically have very limited computing resources available rely on simple movement detection or at best simple face detection, body tracking could enable much finer logging. Medical applications that require measurement of human motion for rehabilitation use invasive mechanical sensors or manual logging and can be greatly simplified. Sign language recognition is still a largely unsolved problem, sports rely on humans judging the legality of athlete interactions that happen during the game but also during training. Commerce and on-line shopping could really benefit from virtual dressing rooms or track consumer behavior on stores and storefronts optimizing shelf space for increased profits.

1.2 Problem specification

We are aiming for a framework that can facilitate high quality marker-less 3D body tracking by observing a person through a single RGBD Camera. Tracking should be robust to self occlusions or occlusions by other objects that may be present on the observable scene. There should be automatic initialization of the tracker as well as a way to recover from tracking errors. The computational complexity of the tracker should allow it to operate at interactive frame-rates of at least 10 frames per second. Finally we would like the tracking procedure to require as minimal of a prior setup as possible.

To better formulate our requirements the proposed method we will operate on an input stream of color and depth images. An RGBD frame pair is denoted as $o = (c^o, d^o)$, where c^o and d^o stand for the RGB and depth frames, respectively. The output of our algorithm will be a parameter vector h that will fully encode the position and orientation of the observed human H. As time progresses and we receive new RGBD pairs $o = (c^o, d^o)$ and for time t our prior state such as $h_{t-1}, h_{t-2}, /etc$ may be used to further help tracking but we should always be able to provide a tracking vector even with no prior state, which is the case when a new person enters a previously empty scene.

1.3 Related work

As reported in [58], discriminative human pose estimation methods [106, 8, 103, 80, 105, 112] map a set of extracted image features to the human pose space. This is achieved through training over a large database of known poses. A variety of methods is defined based on the employed features, the mapping method and the actual training poses database. Recent approaches based on CNNs have produced very promising results [92, 124, 48, 55, 93, 13, 120]. A very recent work is VNect [55] which uses a convolutional neural network to acquire a 2D pose from an RGB frame and then performs regression using a kinematic skeleton to estimate 3D joints from the data. Although performance is real-time and output quality is very good, the lack of detailed knowledge about the tracked model and the absence of depth information have a negative impact on accuracy. The LCR-net [93] is another detection plus regression framework which converts 2D proposals from RGB images to 3D joints but lacks the high quality model and 3D rendering capabilities that are permitted when an RGBD sensor is used. Discriminative methods perform single frame pose estimation, so they don't rely on temporal continuity. Thus, they do not require initialization and they don't suffer from drift. Their offline training is computationally demanding, while their online runtime is rather good.

Generative approaches [22, 18, 20, 86, 26, 25, 117, 16, 56, 127] use a model of the human body and estimate its position, orientation and joint angles that bring the appearance of this model in accordance to the visual input. The model is usually made of a skeleton and an attached surface, which in some cases [26] is allowed to deform. Instead of estimating the full body model in a single step, a variety of methods first identify body parts. Then, they either report them as the final solution or they further assemble them into a full model [103, 105]. Generative methods rely on an objective function that quantifies the discrepancy between a model pose hypothesis and the actual visual input. The minimization of the objective function over the possible poses, determines the one that best explains the available observations. This amounts to the exploration of the high dimensional space of human poses. The size of the search space can be reduced by employing kinematic constrains based on biomechanical data that exclude non realistic poses. Further reductions can be achieved by constraining also the dynamics, i.e., by employing Kalman filters [59]. However, this requires learning of the dynamics of specific human motions and thus reduces the generality of the approach. Another way to deal with the high dimensionality of the search space is to perform local searches in the vicinity of the solution of the previous frame. This works fine under the assumption of human motion with temporal continuity. However, the violation of this assumption may cause drift and track loss. Local search also means that tracking needs to be initialized for the first frame. Due to their generative nature, the computational cost of the online process is typically high. On the other hand, the employed model can be changed easily, and the whole search space can be explored without the requirement for offline training.

Hybrid methods that integrate discriminative and generative components have been proposed [32, 2, 27, 79, 121, 125, 57] to combine the benefits of both words. Hybrid methods achieve the accuracy of the generative ones without need for initialization and with robustness to tracking failures. The method proposed in this work falls in this category of human pose estimation methods. A good comparative overview of many human perception methods can be offered by [62] and [14].

1.4 Our approach for a solution

In order to properly formulate a proposed framework, we need to be able to address each of its specified requirements and provide some background information from the literature. In general markerless body pose estimation can be broadly taxinomized in two major categories, generative and discriminative approaches.

Although all of these methodologies will be thoroughly assessed in the next chapters they are briefly mentioned here right after the problem specification in order to help the reader comprehend the early design decisions for the formulation of the tracking framework.

1.4.1 Body detection and tracking

Generative body pose estimation techniques formulate an optimization problem to be solved by having a parametric model that corresponds to a comparable solution to our observations. They then treat the problem as a regression through the optimization space where we constantly try to minimize discrepancies between observations and hypothesis. This is typically a very good methodology to follow especially when we investigate small search spaces that are close to a global optimum solution. On the other hand as the search space grows larger and our samples are far away from a solution they might not converge to the correct solution.

Discriminative body detectors on the other hand leverage large pre-learned training sets and work directly on the raw image input, dismissing areas that greatly differ from the learned set and selecting others that are close to it, they typically are less precise but can more efficiently scan a bigger search space and substitute the need for a carefully crafted parametric model of the solution with a large training set.

Since we require automatic initialization as well as error recovery it was immediately evident that some sort of discriminative body detector would have to be employed at least for those 2 scenarios. However in order to perform the fine-grained tracking of our goal a generative approach would also be desirable to refine the detected bodies. The improvement offered by the generative method of course would depend on how close our hypothesis could match the real world observation.

1.4.2 Providing the tracker with high quality models

There have been various works that thoroughly model, digitize and animate human bodies and human parts such as the face, hands and hair etc. Although generic models that can be modified to fit specific criteria like height, weight are not uncommon, in order for our generative component to render hypothesis closely resembling reality we wanted to be able to go as far as actually mirroring the observed human body. Thus our human model had to actually come from the observations and not be a generic mesh.

This meant that there would have to be provisions for 3D scanning the person and a way to convert the 3D geometry to a skinned and articulated 3D mesh that could finally be rendered and closely resemble our observations. Using this automatically 3D scanned model despite operating on a "marker-less" body, the detailed knowledge of its 3D structure would theoretically make our tracker operate like "having markers" since the unique shape of each body should make it easier to be tracked.

Acquiring the 3D scanned model could theoretically be done on the fly during tracking. We could initially start tracking using a generic human model and as we would sample new observations we could mold the generic model into a more precise model tailored to our observations. In case of errors during the simultaneous tracking and 3D modeling however these errors would have a very negative impact that could lead to a catastrophically bad model that would in turn lead to complete failure tracking. Another more modest strategy would be to have a brief first stage where we would only do 3D scanning and only after this stage concluded move to the tracking stage. This latter idea would make both scanning more robust as well as decouple tracking errors from the model where they would tend to accumulate. Unfortunately it would slightly inconvenience users since there would be a slight delay before tracking but that would of course only be experienced the first time tracking a new person. Consecutive uses of the tracker would not suffer from any overhead since human bodies tend not to change very rapidly. In case of a significant change caused by a different set of cloths it should be easy to switch to the first step again, get a new scan and then proceed again to track with the updated model.

1.4.3 Fusing the parts to a single framework

Our goals lead us to shape the final solution with the following formulation. We will use a 2 stage architecture that will isolate scanning from tracking for the reasons explained in the previous paragraph. The first stage will observe a person for a few seconds and acquire a 3D model . This should only be done once per person and could altogether be omitted by opting to use a generic human model in case of scanning failure. The second stage will be to perform tracking using the personalized model which will also consist of two separate logical steps. The first sub step will be human detection and the second human tracking working on prior detections and tracks. The decoupling of scanning from tracking will be beneficial towards performance since all of our computational budget will be distributed and dedicated completely for each of the tasks. That being said though the second stage will also need to be hardware accelerated with GPGPUs in order to meet our interactive performance requirements although we can tolerate a relatively slower scanning step since it will be seldomly performed.

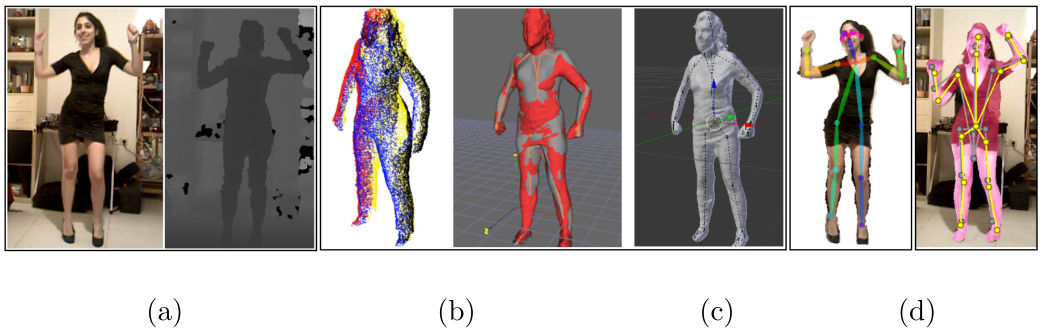


Figure 1.1: Overview of the proposed approach for 3D human pose estimation. (a) The input to the method is a sequence of RGBD frames. (b) We leverage models that can be acquired easily [102] and yield personalized, parametric, skinned human bodies [21]. (c) We also employ state of the art estimation of 2D human body joints [13]. We define a hybrid approach for fitting the model (b) in the observations (a) given the hints provided by the 2D joints estimation (c). The result of this process is visualized in (d). The gray skeleton is the neural network suggestion and the yellow is the result of the proposed approach. Although the neural network estimation is far from the ground truth, the obtained solution is better because of the high quality model and the utilization of depth information.

Chapter 2

3D Reconstruction

3D reconstruction is defined as the process of capturing the shape and appearance of real objects in a digital form. Creating a virtual version of a real object enables computers to perform complex operations on the vertex data and transform it at will. In our case the object we want to digitize is a human body. A virtual representation of the body will enable us to use it as a tool for tracking a real body observed in the real world. If the model can change its shape in time, and this is the case for humans that can bend, twist and move their limbs, this is referred to as non-rigid reconstruction.

Humans have always been trying to replicate their surroundings and evidence of that appears since the earliest known human history with cave drawings. The works of all classic artists and our sense of history much relies on these sculptures and paintings during Greek and Roman times and after seeked to immortalize the human form from volatile flesh to marble and metal. 3D Reconstruction can be though as an even more timeless representation since objects become purely mathematical and abstract collections of points in space.

Digitization of real objects has been one of the earliest applications for computers with Ivan Sutherland‘s “Sketchpad” in 1963 being the first digitization tool that relied on manual input of coordinates of the structures modeled. This lead to more applications that were initially used for better design and manufacturing of industrial parts during the 1980s. Although not relevant to this thesis it is important to state that today the Computer Aided Design industry or “CAD” is a multi million industry that offers tools that are essential for Architects, Engineers and other disciplines. The recent advent of 3D Printers has further increased the need for this accurate representation of objects this time with the goal of replicating them with the 3D printer. In contrast to the simpler models that “CAD” target with an immaculately detailed scale and 1:1 mapping of the 3D coordinate of any point on the object’s profile to the real object, 3D reconstruction of humans was initially mostly been practiced in the domain that is generally called 3D Modeling. Instead of CAD‘s focus in scale and measurement details, 3D Modeling was and

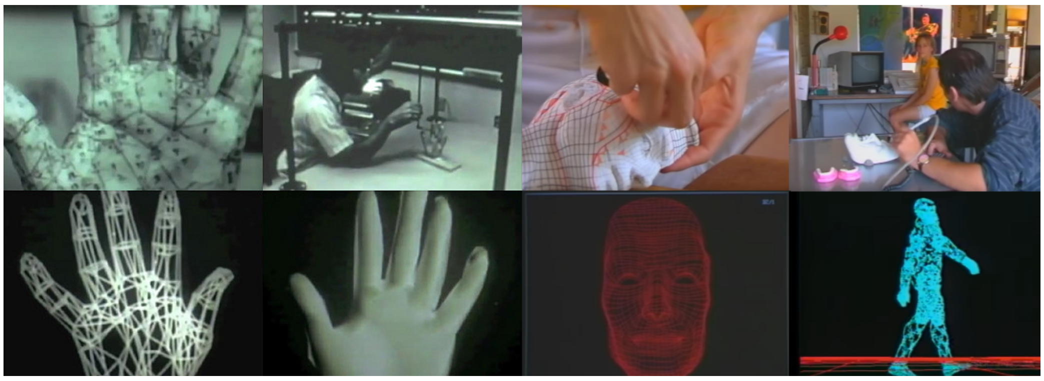


Figure 2.1: Left: Edwin Catmuls digitized hand, circa 1973 Right: A decade later sculpture artists create a physical 3D human model and digitize it, circa 1988 .

is more aesthetically focused and used in a wider variety of fields, such as Computer Graphics, Computer Animation, Virtual Reality, Special Effects for Films, digital media, etc. The first 3D representation of a human in a computer was the "Boeing man" in 1960 by William Fetter while Frederic Parke created the first 3D human face in 1972 and Edwin Catmul created a digitized hand in 1973. All of these early reconstructions relied on drawing lines using ink markers on top of the human skin or building human models made out of clay, and then manually digitizing them using light pens as the input source. In order to animate facial expressions across time the procedure was repeated for each one of the frames and the final animation was the playback of all of the recorded states rasterized and then rendered one after the other on a TV-screen.

Gradually, digitization shifted from the fully manual input of every vertex for every frame to the use of motion capture suites that featured easily identifiable fiducials and calibrated camera sets that viewed the scene from various different angles. This required a costly setup but it did not only made the capture of motions easier and more realistic but it also encoded them as a series of incremental transformations that could be applied on any digitized mesh. The target meshes themselves still came from artists that digitized physical models and textured them accordingly using photographs to make them appear more diverse and realistic. This gradually led to life-like digital avatars that populated virtual 3D worlds in computer games and that started to become so detailed that made it possible for them to appear next to real actors in block-buster films like the Lord of the Rings in 2002 and even TV series in more recent studio productions such as Game of Thrones 2011.

2.1 Basic concepts

As stated in the introductory chapter 1.4.2 in this thesis we are only interested with rigid object 3D reconstruction in order to acquire a high quality 3D human model. The reader however, might have come across 3D reconstruction in a different topic since it can also referenced in the literature as SLAM (Simultaneous Localisation and Mapping) which is a more fitting term when talking for the much more unconstrained problem of reconstruction of indoor or outdoor spaces for robot navigation. Other synonyms are SFM (Structure from motion) or MVS (Multi-View Stereo) which specifically target moving or stereo camera setups, specific subdivisions that have different terms due to their different constrains. Of course in our context we will not deal with any of these problems but they share the same mathematical background and are mentioned here to provide useful context and disambiguation. Many of the citations in this section are works that initially targeted different scenarios than our 3D human reconstruction but all of which have ultimately contributed to net improvements in all 3D reconstruction applications.

A great book for the reader to become familiar with the basic concepts for 3D reconstruction is Multiple View Geometry in Computer Vision [29] that thoroughly covers the topic. In order for someone to gain the foundations to sufficiently understand the linear-algebra used in the book, [81] is a good source for Linear Algebra foundations and [24] can help understanding 3D transforms and provide basic knowledge of 3D graphics and computer rendering.

Our scenario involves an RGBD camera which makes reconstruction more straightforward when compared to the general case of RGB cameras. The depth information we have available offers more constrained data to work with and this makes the reconstructed output more dense and more accurate. The general case of reconstruction has to simultaneously deal with two problems. The first is calculating the ego-motion of the camera from frame to frame or “visual odometry” as it is called in a SLAM context. The second is to try and extract an estimate about the depth of every pixel in every view which will be ultimately combined in a single final mesh. If we are working with a camera with unknown calibration parameters a third problem is also implicitly included since the camera calibration parameters need to be found since they are required in the computations performed. Depending on the scene and the number of salient features the quality of reconstruction might greatly differ and typically a 3D scene is incrementally updated with voxels (3D pixels) as they are accumulated in computer memory to form the final result.

Having the luxury of an RGBD sensor provides us with calibration information and more importantly with a depth frame for each view. This makes the reconstruction problem significantly easier than the general case. Assuming a perfect sensor with precise depth information would make the problem devolve to just the calculation of ego motion. However the depth frames extracted by the sensor suffer from noise as well as artifacts caused by specular reflection of the infrared signal that the RGBD sensor emits. We are thus left with two problems to face one being ego motion and the second a way to integrate multiple received depth frames to gradually clean artifacts and fill empty regions.

2.2 Reconstructing indoor spaces and rigid objects

Despite the fact that we are approaching 7 years from the release of the original KinectFusion paper [64], it remains the most well suited and state of the art method for reconstruction due to its simplicity and extremely good architecture that uniquely complements commodity hardware. It was made specifically to leverage the capabilities of the Kinect RGBD sensor as well as modern GPGPUs. It is widely used and has a variety of implementations with the most notable being the closed-source Kinect SDK version which is now part of the Windows Operating System as well as an Open Source implementation in the PCL framework [98]. The original Kinect Fusion paper[64] has been improved in various ways over the years that mostly address resource consumption and make it more extendable for reconstructing larger spaces. There have been works that introduced Octrees [35] or Voxel hashing [67] to make mapping more scalable and memory efficient. Works like Kintinuous [122], and Moving Volume KinectFusion [95] tried to dynamically tile the rendered output to accommodate larger structures. Finally works like Real-time non-rigid reconstruction using an RGB-D camera [128] focused on non rigid objects and Dense visual SLAM for RGB-D cameras [42] improved reconstruction by leveraging photometric and depth error across all pixels. Finally SLAM++ [100] introduced an object graph where identified and repeated objects would boost output quality and Kinect noise modeling [66] was used to improve sampling, a technique that in turn improved output quality.

As stated despite a lot of work that has been done on improving the method our requirements for scanning a single human fall very much inside the capabilities of the original Kinect Fusion algorithm and the extensions provided by the scientific community would not benefit us. The scale of our scan is very small and the distance we are working on (0.5m - 1.5m) is one that has the smallest amount of depth noise as seen in figure 6 of [66], since Kinect was originally specifically designed for watching humans from similar distances.

$$ICP_{Matrix} = \begin{bmatrix} R & T \end{bmatrix} = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{bmatrix} \tag{2.1}$$

The Kinect fusion algorithm separates the problem of camera motion estimation from the model updates, a tactic that was popularized by PTAM [43]. Camera Motion Estimation is performed using ICP (Iterative Closest Point) [6] that solves the problem by reformulating it as a non-linear optimization problem where the closest points between tested orientations of frames are used to iteratively align the surfaces in question. The matrix computed during the ICP operation can be seen in Equation 2.1 and has a distinct form for its rotation component, that is easier

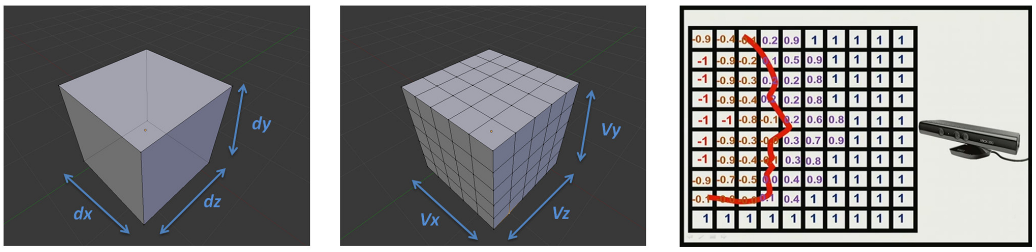


Figure 2.2: Metric Space of the Voxel Storage, Resolution of the Metric space, Visualization of the TSDF.

to calculate and suitable for small angle differences between consecutive frames. The last column translation part is encoded normally but the first 3x3 part of the matrix encodes rotations using just three variables α β and γ in order to make fitting more efficient. In case of larger motions and ill-posed data however kinect fusion enters a "lost" state where it will wait until relocalisation is achieved.

The GPU implementation of ICP used by Kinect Fusion enables it to achieve this on a framerate that matches the image acquisition framerate. Being able to derive a transformation from each newly observed view to a single World Coordinate frame then leads to the next problem of how to store the accumulated mesh. Kinect Fusion uses a voxel space that encodes samples with a truncated signed distance function (TSDF) representation and a threshold that is denoted as μ. We can imagine this as a way of thresholding while also trying to model both currently unoccupied space and the uncertainty of the sensor readings for each point.

The presence of the TSDF map allows the Kinect Fusion algorithm to predict the incoming surfaces from the depth sensor. After acquiring a new pose estimation using ICP this predicted surface is compared with the observed depth by casting a ray through every pixel in order to accurately update the map.

This concludes our brief summary of the Kinect Fusion algorithm. As stated before we are only interested in its fundamentals since our subject is not mapping indoor environments but fully focused on a single human. For this reason a lot of its implementation details have been omitted since they are out of the scope of this work and are readily available in the original paper [64] for anyone that wishes to delve into the details.

2.3 Reconstructing rigid human meshes

In contrast to Kinect fusion where we are dealing with a freely moving camera, and although the body tracker should be able to track humans even from a moving camera, during the scanning phase we assume that we will be operating from a static vantage point. We want to have the same reconstruction quality as Kinect Fusion but we also want to be able to scan the human without the help of a

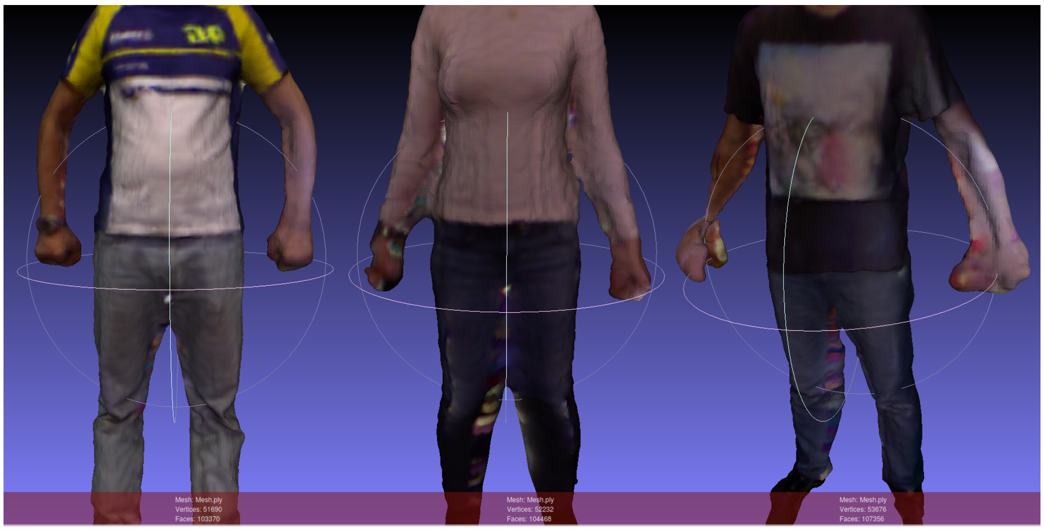


Figure 2.3: Left: A successful reconstruction. Middle : Successful reconstruction but wrong T-Pose by the subject. The arm vertices are connected to the torso, this mesh will exhibit severe artifacts around the arms after the automatic rigging. Right: Subject moved both of his arms during scanning and articulated reconstruction could not repair them. Their volume no longer corresponds to observations and tracking will be adversely impacted .

separate person that will hold the camera to complete the 360 degree scan. There have been works that involved three Kinect sensors that are used along with a turntable [111] ot just two Kinect cameras [126] to solve the problem, but this is something that is not feasible in our single camera case that shouldn't require any additional tools. Another difference from the regular indoor mapping and tracking scenario is that humans may find it difficult to stand completely still for an extended period of time and movements in the Kinect fusion modeling are ultimately handled by averaging. That design choice is good for some input cases but can produce noticeable artifacts with rapid motions in brief scanning periods resulting in characteristic "faded" body parts seen in Figure 2.3.

In order to satisfy all these requirements we chose a framework [102] that uses the motor of the Kinect camera to provide two different vertical views from the same body pose of the subject and thus only requires the subject to sit still in 4 orientations. The method involves audio output giving the subject audible commands in order to know when to sit still and when to perform a 90 degree rotation. The meshes are then automatically fused using Kinect fusion and a form of super resolution is achieved by integrating 200 frames per view. There is limb segmentation that helps reduce artifacts from the reconstruction of limbs that might have had slightly moved during the scan and the floor is removed by fitting a plane on the output depth with RANSAC [23]. Poisson surface reconstruction [41] is applied in order to extract a watertight mesh that also contains color information from the RGB camera. The final output is a rigid and watertight 3D mesh with colors per vertex that is metrically accurate.

2.4 From rigid human meshes to articulated models

The human model extracted using [102] can only be transformed with basic 3D transformations since it is rigid. We want to create a deformable version of it where we will be able to configure all of the joints and limbs in order to make different poses that will facilitate our tracking search. Rigging meshes can be done manually using 3D modeling software that can create bones and associate them with the underlying geometry. In our case however we want the procedure to be as fast and automated as possible and to these ends [21] is perfect for our requirements but also exceeds them by allowing reshaping that could in principle ensure that our extracted model could always be fine-tuned to compensate for changes in weight of the subject that can happen during a course of months or years. The selected framework [21] uses SCAPE [1] in order to create a morphable human model . The rigid 3D human body is fitted to the morphable human model produced by SCAPE [1] and mesh correspondences are established. These correspondences can transfer both skeleton and skin binding weights from the template mesh onto the input scan to generate our fully animateable human body.

Our final result is a mesh that has a hierarchical skeleton with associated weights from each joint to each vertex. This articulated model can be configured at will by supplying 4x4 transformation matrices per joint and recursively applying the transforms on the vertex data of our model. The poses we choose can be thus rendered and rasterized as RGB and depth frames and if we choose calibration parameters that correspond to our camera we can have render output that is directly comparable for pixel-by-pixel comparison with unfiltered Kinect camera input. This will be our basic generative tracking building block and the quality of the tracking will be directly affected by the quality of our articulated model.

2.4.1 Improving blending using dual quaternions

It would appear that at this point we have concluded our articulated model formulation, however there are still issues that we can improve. Since our method for acquiring and skinning the rigid human mesh is completely automatic, there are a visual artifacts like candy-wrapping and elbow-collapsing that can be caused by linear blending of the skinned model. These can be mitigated by manually rebalancing the vertex weights (which can be a very time consuming task and breaks our requirements) or by using a better method of blending which helps makes weight inconsistencies much less pronounced. A good improvement with hardly any performance sacrifice and minimal implementation overhead can be achieved by using dual quaternion blending [40].

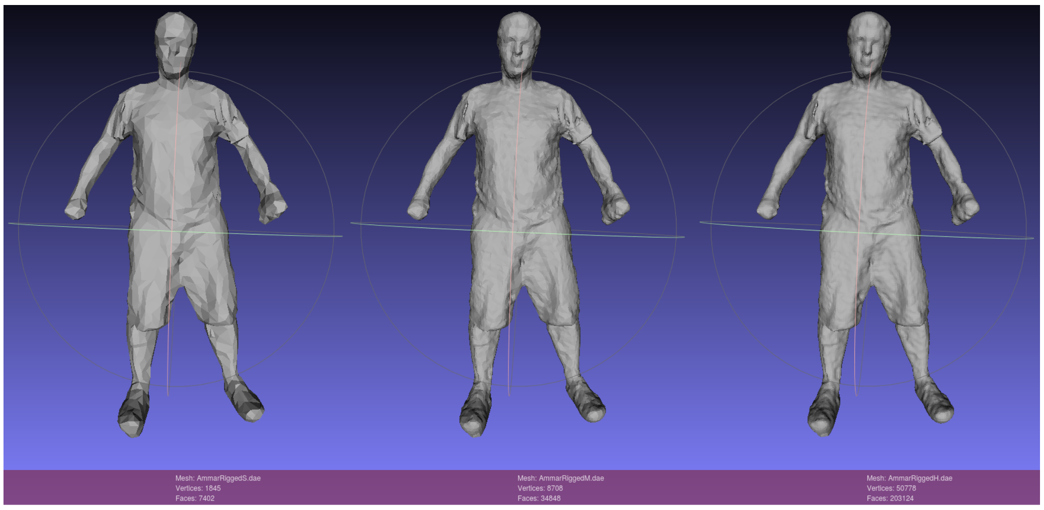


Figure 2.4: Left: A human body that is being represented with 1845 vertices and 7402 faces. Middle : The same body with 8708 vertices and 34848 faces. Right: The original scanned resolution with 50778 vertices and 203124 faces. We notice a substantial increase in geometry complexity that has much less pronounced impact in the observed silhouette of the 3D model for the T-Pose. However skin weights can be adversely impacted in the step of skeleton rigging from our decimated geometry. In this work we use the full quality model since we are interested in maximum accuracy, there is however space for performance optimization using a less detailed model, that will be faster to render and evaluate.

2.4.2 Choosing the correct vertex count

Another point worth considering is the vertex count of our geometry. Although modern graphics cards can handle massive amounts of polygons passing through complex shaders the complexity of our meshes can have a small but noticeable impact on rendering times. Given the fact that Kinect depth input includes noise and its resolution is VGA, (with an even smaller bounding box of the observed human) having an oversampled model might just prove to be a waste of resources. We can use edge decimation in order to reduce vertex count and create simpler versions of the original mesh. The visual difference of the resulting renderings with different levels of detail can be seen in figure 2.4 where even if we go as far as removing 97% of our initial triangles, the human silhouette and depth map remain relatively unchanged. The most affected aspect of the mesh is its skinning as reduced polygon counts make artifacts more pronounced. Although there is space for optimization in model simplification we will delve into the subject cautiously by just assessing the benefits and problems caused by reducing the level of detail of our model. Our original mission is to use the highest quality model attainable and willingly over-reducing it can be counter productive to our goal.

2.4.3 Representing orientations

An important decision we need to make is how to encode joint rotations. Mathematically the best way to encode them is to use quaternions or 4x4 matrices that do not suffer from Gimbal Locks. The quaternion/matrix format however contains configurations outside of the physically possible configurations of the body that are difficult to be filtered by composite rules. Another problem is that in order to specify a quaternion or 4x4 matrix one needs to provide many parameters which will add unnecessary degrees of freedom to the parametrization space and ultimately our optimization problem as we will see in later chapters. Human joints, on the other hand, are severely limited in their motions and most joints can rotate in just one or two axis. Due to these natural constraints that automatically prevent gimbal locks, we are safe to use euler angles to represent our joint rotations. Euler angles are much easier to constrain and filter and have the added benefit of being directly transferable as motor commands in applications like humanoid remote control. One of the early testing applications of this work was tele-operatation of a NAO humanoid by mirroring the skeleton pose. Instead of using inverse kinematics to derive angles from joint positions this parametrization scheme can be directly used although knee and hip orientations can result in the robot falling off balance and typically only upper body angles are used.

2.4.4 Choosing a file container for our models

The output of the Fast Avatar Capture [102] rigid reconstruction uses the Stanford Triangle Format [113] (or Polygon File Format, .ply as it is commonly known) that was initially released in 1994, is well documented, easy to parse and offers both ASCII as well as binary representations of the geometry. This file format however lacks any accommodation for skinned models, vector weights, joint hierarchies, motion limits and other things that we need to keep as part of our stored model and its state. During the conversion of the rigid model to its articulated version [21] the model is converted to COLLADA which is an xml based format originally drafted by Sony Computer Entertainment, Khronos group and a consortium of 3D developers. Although COLLADA is widely supported, well documented and standardized, the same does not hold for the various modules of graphics tools that handle import and export from other representations. This fact combined with many of the COLLADA features being incrementally added to the file format, often create problems which were also present during our experience. Although the Autorigging and Reshaping tool [21] could provide output that was flawlessly imported in the USC SmartBody suite that was not the case when imported in the popular Open Source 3D Modeler Blender, or the OpenMesh tools. These COLLADA importers needed various scale changes, lacked color information on some editors and small conversions had to be performed in order to make the model appear correctly. Another negative of the xml nature of COLLADA was the unnecessarily large file sizes. We ended up creating a small Open Source container format that is called .tri that shrinked the 25MB collada 50k vertice models down to 4MB, it is compact, very easy to interface since it is written in pure C does not need external libraries to compile, and can hold all of the information. This file format is documented in Section 6.2.2 that covers the technical details of our framework and groundtruth and provides the required links.

Chapter 3

Human Body Detection

Humans depicted on digital images exhibit regularities and patterns that are immediately apparent to any human observer due to our innate abilities. These patterns of course are also reflected on camera sensors as light intensities or 3D points the raw captured RGB and depth image data. The variability of the human figure along with the lack of a way to efficiently group and label all possible combinations of pixel clusters in contemporary computers prohibits an explicit general solution to these problems. It makes sense thus that the first and most straightforward approaches to facilitate object and human detection by computers would be to create programs that can derive their own rules for accepting and rejecting image regions based on example sets that we can provide with guides of what we want to recognize. This idea is generally referred as machine learning or pattern recognition when specifically talking about image patterns and computer vision.

There are a lot of interesting ideas and applications that try to tackle the problem and they can be divided using multiple criteria. Their need for human supervision, the class of algorithms that they use to formulate a solution, using probabilistic, symbolic or logic based methodologies etc. In the same manner as in the previous section we will provide some context for different methods. However due to the vast range of methods in the machine learning field, it is out of the scope of this text to adequately cover all of them, so we will instead focus on the most important techniques that are geared towards human detection and as stated in the introduction chapter, we will ultimately divide them to discriminative and generative methods.

3.1 Discriminative methods

Discriminative methods are modeling techniques that try to derive a result x by leveraging an observation o and a knowledge function $P(x|o)$ that can help approximate x from o. As already stated, the function P can vary from a purely statistical and probabilistic mathematical function to a dynamic programming algorithm. What is common between all methods regardless of the exact method details, are

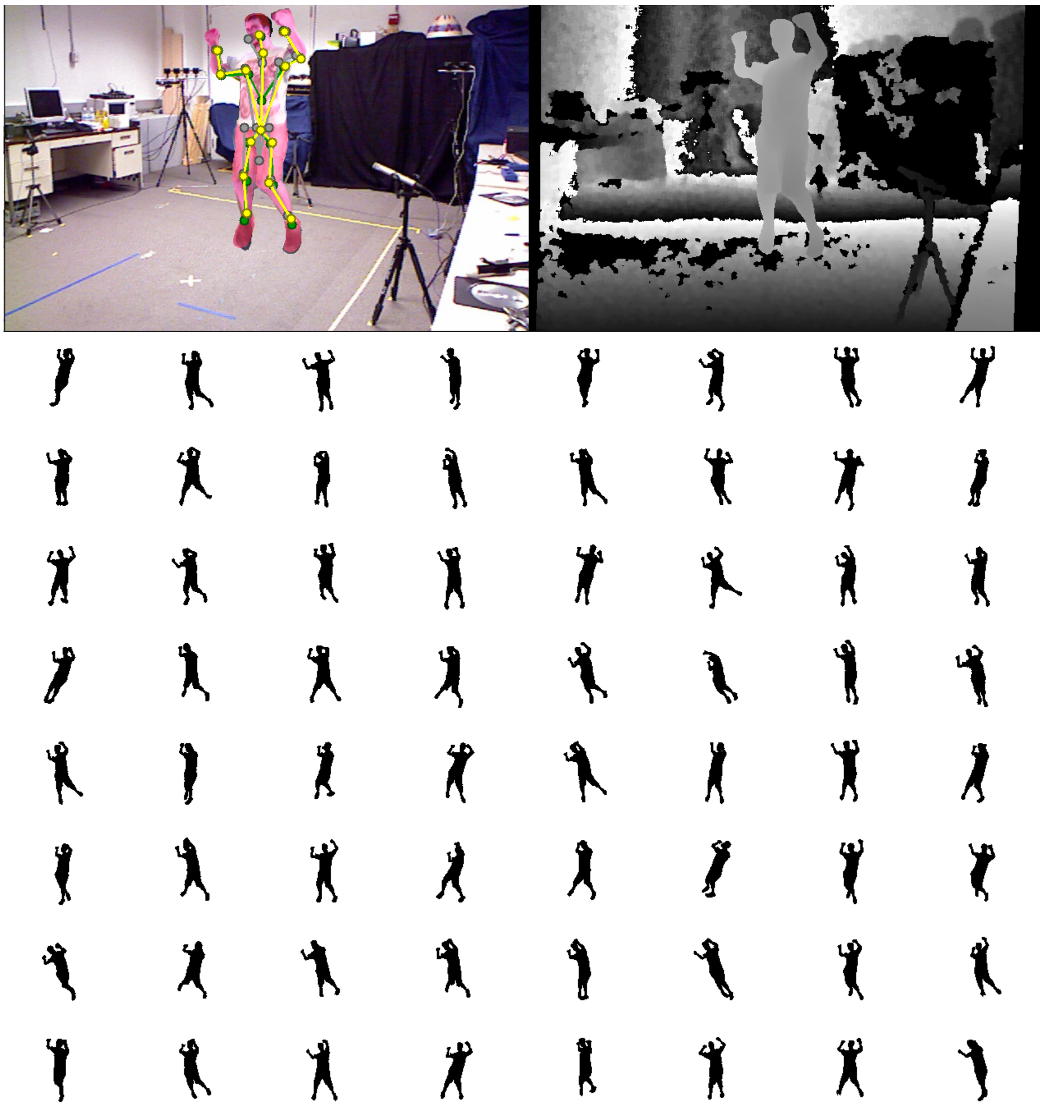


Figure 3.1: Top Left: Highlighted solution over an observed frame. With green color we have highlighted the correct location of each of the body joints which is our ground truth. With gray color we have highlighted the joint positions as estimated by the chosen Neural Network Discriminative human detector. Although they are not totally inaccurate they are still relatively far away from the true solutions. With Yellow we can see the joint colors that have been refined using a generative method that are much closer to the correct solution and thus mostly occlude the green markers. Top Right: The observed Depth Map contains both background and foreground. Thus a generative component would find correspondences with other objects in the scene as well as our target, discriminative methods on the other hand given enough training can completely reject background and are much less prone to false positive results. Bottom : A batch rendering of 64 random configurations of a human body that adhere to our movement limits and are close to the previous known configuration. The depth color has been inverted to make the figures more visible. Our configurable 3D body model will be used to form such hypothesis that will be iteratively scored, assesed and refined until we converge to the good solution shown top left.

the overarching constrains they work with, their performance and ultimately their overall accuracy at the detection task. From our original problem specification and our attempt to formulate a solution, the constraints we have and want to cover are automatic initialization and reinitialization after a tracking error. These two cases follow the pattern of having an incoming frame pair $o = (c^o, d^o)$ and wanting to derive the person pose x by using a pre-trained algorithm P without any access to a recent good estimation of the pose.

Discriminative models, do not generate new comparison samples but instead indirectly compare an observed scene to a training set. If we were to use a generative method to perform body detection it would involve enumerating all the possible positions and configurations of the human body and then trying to generate enough samples to densely cover the whole space. These millions of samples would then have to be separately tested and compared to the observation in order to find the sample closest to the true observed position and configuration. Iteratively we could then generate more and more samples as we approach the solution in order to achieve the desired accuracy. This "generative detector" would of course be a huge waste of resources since almost all of the generated samples would be far from the solution and would provide little to no hints towards the solution. They would more or less have random correspondences of depth with the background geometric volumes as seen in Figure 3.1, and only if our model and our comparison function are very accurate would we end up with a good solution after tremendous effort. On the other hand discriminative methods although ultimately limited by their training sets have an architecture that allows far superior performance when examining large areas of solutions since they perform far fewer computations.

3.2 2D body detectors

Our use case has a relatively rich camera setup since it also includes depth information. However as we will see historically this was not the case and it still isn't. The majority of camera systems are monocular and only contain color or even monochrome information. 2D body detectors typically rely only on a color c^o observation and their output consists of a vector of 2D points or 2D bounding boxes that describe body parts in the image plane as seen in Figure 3.2.

Bounding box output is typically faster to compute than 2D articulated skeletons since it skips the computations required to derive the exact locations of each of the limbs and just provides an aggregate result. Trackers that produce such output are very useful in applications that require fast detection rates and where the configuration of the bodies is not useful or important. Examples of scenarios and applications that can be accommodated with bounding box detections are human avoidance algorithms for autonomous cars, human detection by household robots or security cameras which just need to monitor persons. In our case though, what we want as a final result of this work will be a full 3D configuration of the body so

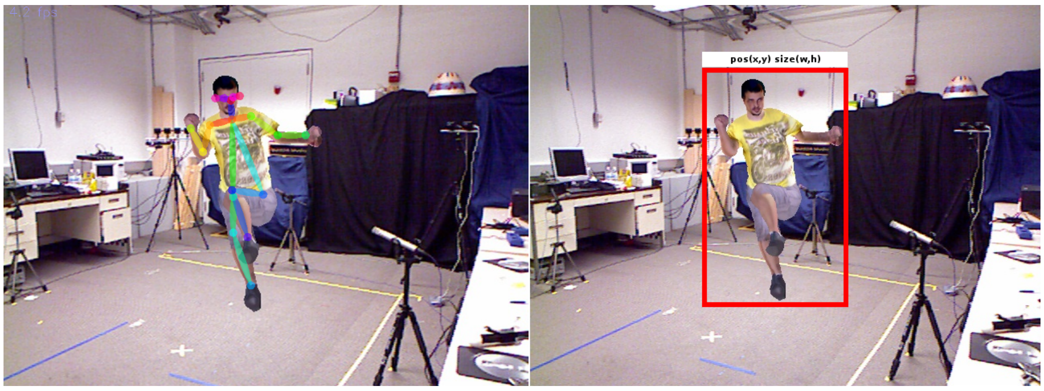


Figure 3.2: 2D Body detectors work on 2D color or monochrome images and can be broadly taxinomized in two families. The one as seen on the left image return a vector of detected joints. The second is a family of detectors that return 2D bounding boxes that contain detected persons as seen in the picture on the right.

although a 2D bounding box would certainly be beneficial in order to narrow down our search space, a 2D vector of joints would provide far more detailed constraints that can help us achieve our goal.

The detailed output of 2D Body detectors that produce full 2D joint output gives rise to many ambiguities that have to be properly defined in order to standardize it. The following list is a concise compilation of 2D body detector features that are important although frequently overlooked in favor of performance or accuracy metrics :

1. The number of returned joints might greatly differ for different methods.
2. Results may or may not respect physical limitations of human bodies.
3. They may or may not contain information for joints that are occluded.
4. The detector can track specific persons across frames or provide a label-less list of all visible persons for each frame.
5. Output skeletons may have varying limb sizes for subsequent frames or in the case of specific person tracking the estimation of the true limb sizes might be attempted.
6. The detector can provide a vector of confidence values that signals the certainty for each of the joints.

We would ideally want output that contains at least Wrist, Elbow, Shoulder, Neck, Head, Hip, Knee and Feet joints. We can tolerate the detector not abiding to physical limitations, and to provide output with big variations in limb sizes since our generative component can take care of these problems. We would prefer

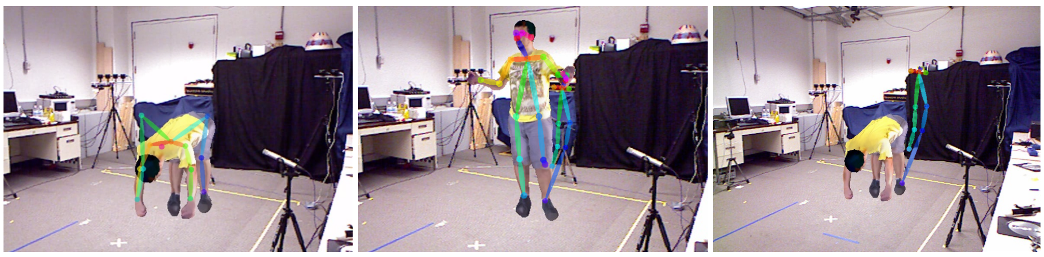


Figure 3.3: Bad cases of 2D body detections that showcase the importance of the list of detector details explained in 3.2. Left: Incorrect detection on a heavily occluded pose. Middle: False positive detection on the background. Right: False positive detection and false negative detection on our foreground.

to be provided with a vector of confidence values or if this is not the case, to have occluded joints be completely omitted. The generative component will ultimately treat these joints as a series of hints so not having any sort of information would be preferable compared to information pointing to the wrong direction. Finally our end-goal tracker will be oriented towards specific persons (with specific information about their geometry) so we could both work with 2D detector output that just tracks a specific person as well as a list of skeletons from which we can automatically pick the one that corresponds to the tracked person and omit the rest.

The latter case can however produce inconsistencies in case a frame exhibits simultaneously a false negative detection of our tracked person as well as a nearby false positive detection of a non-existing person detected on the background. This example can be seen in Figure 3.3 and will have an adverse effect to our tracker. The simplest way to combat this is to add heuristics that threshold 2D detections that oscillate too far from previous detections, but adding such filtering rules can also backfire in case of tracking drift. If our tracker starts to lose track of the person and the 2D detector is still detecting the correct position we would not want to discard this helpful 2D detector output because of its distance to our tracking state and remain lost. Since there is ultimately no good way to recover from incorrect detections we implement a minimal skeleton selection algorithm that uses the closest 2D detections and utilize a high quality 2D body detector that does not exhibit high false positive/negative rates.

3.2.1 Cascade based

One of the earliest, most robust and efficient methods for human detection was Rapid object detection using a boosted cascade of simple features [118] from P. Viola and M. Jones. This work dating back in 2001 and still being widely used today greatly reduced the body detection problem complexity by providing a reliable and computationally efficient way to detect body parts. Its use of Summed Area

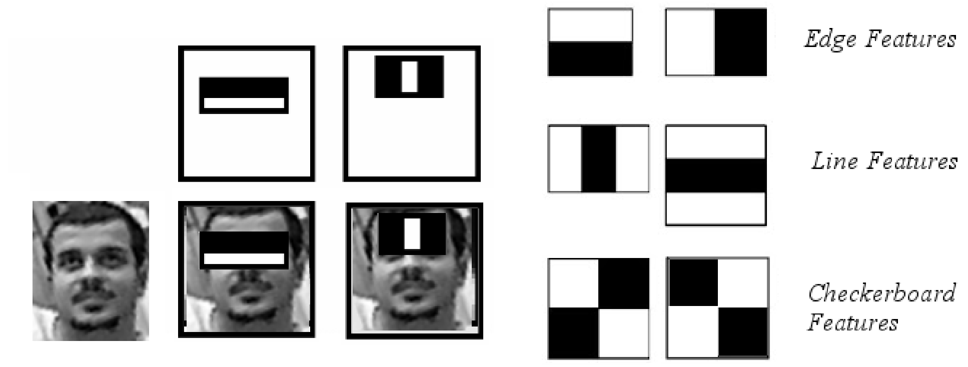


Figure 3.4: An overview of how a HAAR cascade is formed and the wavelets that comprise it.

Tables speeded up computations and enabled computers to perform arbitrarily large image region comparisons with constant complexity. On the other hand the ability to group the cascade into different sequential stages make it very fast when rejecting areas of the image that contain no faces since only a subset of the whole cascade needs to be computed. Using this building block along with a cascade of rectangular areas and a big training set, the algorithm adjusts threshold values for each of the cascade items in order to maximize fitness on the training data. These HAAR cascades are typically not full body detectors but just detect human parts such as the face. The face is easier to identify than a full body since it is more consistent across different persons and only has one joint, the jaw. Other details like facial hair, glasses, skin color as well as accessories such as hats can be included by extending the cascades and training sets. The algorithm is susceptible to lighting changes and does not generalize well on different skin-tones. Keeping in mind these limitations it still makes a very efficient and accurate building block for more elaborate detectors. A cascade of HAAR cascade detectors can be used to facilitate body detection. These full body detectors typically first perform rough person detection and outputs regions that may contain persons and then secondary specialized HAAR detectors work inside these rectangular areas to provide fine details such as the head, eyes, mouth, hands etc. The final output is typically a list of rectangular areas that have been detected by the HAAR detectors. As stated before in this Section we aim for joints and not bounding boxes so our requirements cannot be met by this method, however it has been a milestone in the computer vision literature on body part detection problems and thus is rightfully included in this section.

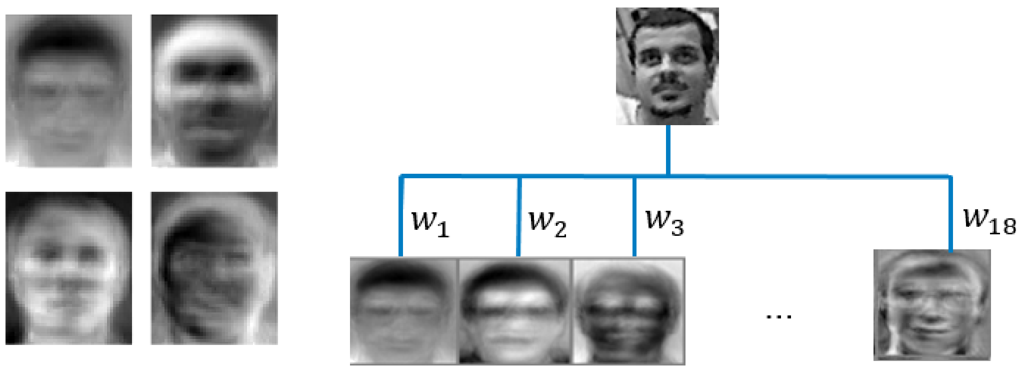


Figure 3.5: An overview of how eigenfaces work

3.2.2 Feature based classifiers

A second class of body detection algorithms are those that convert the input 2D image to a vector of features and then classify humans by comparing observed clusters of features to prelearnt clusters. Some of them utilize SIFT[51] or SURF[3] feature descriptors in RGB, YCrCb, HSV or monochrome color-spaces [7], [60]. Each of the feature sets can help form a specific human body part detector, and with the combination of multiple robust part detectors [60] a full human body detector can be approximated.

All of these methods ultimately try to bypass the problem of direct pixel by pixel comparison of an observed image with every image of the trained library. Being able to perform optimization that minimizes the distance of a subset of the detected features. However they can still struggle with lighting, body articulation and even facial expression changes in the case of face detectors.

3.2.3 PCA based classifiers

Other methods employ eigenfaces [114] which perform a combination of the ideas behind the Cascade classifier as well as the feature extraction methods. The big training set of faces results to a much smaller number of eigenfaces that greatly reduce the complexity of the problem.

3.2.4 2D DNN neural networks

Finally convolutional Neural Networks, or Deep Neural Networks are the technique we have chosen [13] as our body detector module since they offer a remarkable new addition to the arsenal of Computer Vision methods. A great introductory book on the topic of Deep Learning is [11]. They mimic the organizational structure and connectivity of vision neurons in animals [54] albeit in a purely mathematical formulation. The complexity of the detection task is handled by the organization of the artificial neurons that react correctly due to the “experience” acquired while training with sample images. Theoretical research on the topic dates back to 1969 [10] but a practical formulation with tangible results was only produced by Alex Krizhevsky as late as in 2012 [44]. This however immediately led to the neural network boom we are experiencing today. A detailed historical survey of DNN methods is a fascinating read and can be found here [101]. Of course pioneering work during the 70s had neither the hardware nor the available number of training sequences to be trained and tested with, and was mostly theoretical and not fit for working applications. However the foundations for the method were laid then.

This technique frees the computer vision developer from feature selection and many additional tasks that are needed to facilitate detections and automate much of the labor required to create a detection engine from a training set while surpassing state of the art accuracy.

A convolutional neural network consists of multiple layers of nodes each of which only receives input from a small number of nodes from the previous layer and only gives output to a small number of the nodes in the next layer. Each node has configuration “weights” and a operation “type” which controls the operation performed on the input data and outputted to the next layer. Training the network is facilitated by a technique called back propagation [31] which iteratively updates the node weights of a neural network to achieve better and better classification scores.

The layout of the Network as well as the types of operations conducted in each layer are very important to efficiently facilitate learning of the data. The bottom “classification” layers are interconnected with all of the top layers but very deep and complex neural network connectivities can lead to the problem of over-fitting to data. Over-fitting can be described as the network learning how to respond to all of the training example queries but not having developed the internal structures that allow it to generalize and cope with input data that is not part of the training set. Typically thinner networks, networks where the back-propagation procedure is stopped early as well as networks where the training data is clustered randomly tend to not exhibit this effect.

The internals of neural networks and the types of layers (convolutional, pooling, flattening, fully connected etc.) are very important but far beyond the scope of this work. Back propagation, the method that drives learning is also a very interesting algorithm to study but again all these details should be covered by a proper introduction to the topic with a book like [11].

The main caveat of neural networks is that they can sometimes classify input with extreme certainty extremely incorrectly as seen in [109], [65] and Figure 3.6 . This fact in conjunction with their opaqueness is a serious drawback that is often overlooked due to their overall high accuracy and ease of use. New work has been done [73] on visualizing their inner workings by trying different inputs and monitoring which of the artificial neurons get most excited by what input. Although there is currently no way to formally prove that the output will be correct for a specific type of input or a good way to hint as to how the structure

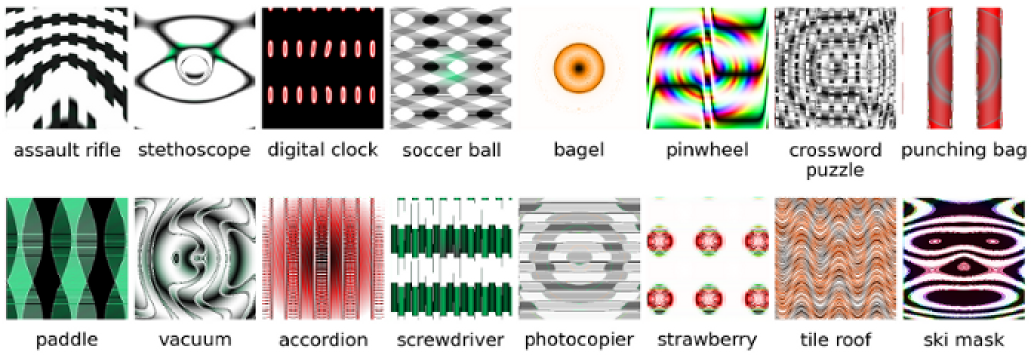


Figure 3.6: Patterns mistakenly identified with very high certainty by deep neural networks. Image taken from [65].

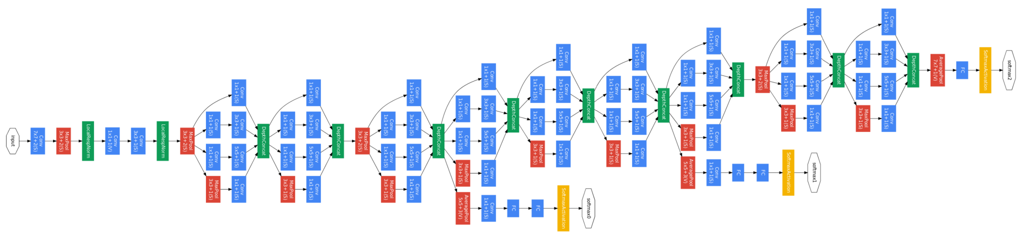


Figure 3.7: Image of the 22 layer network from [110]. Blue nodes perform convolutions, red pool inputs and yellow perform softmax operations.

of a network could be improved. This means that most of the networks are used as black boxes and typically creating a new network, most of the time consists of merging parts of other pretrained networks or combining multiple networks that are known to perform well for similar tasks in an attempt to re-purpose them.

Another issue that neural networks face is that they require sizeable datasets as well as sufficient computational resources in order to be properly trained in a reasonable time-frame. Luckily big internet companies are in a unique position to handle both a very large number of human generated content (video, audio, pictures and text) as well as own data-centers that can handle the volume of data. The already trained algorithms can be also used to automatically annotate new incoming data thus making it a form of semi-autonomous or semi-supervised (depending on how you look at it) method. As already mentioned, people without the resources to train a network from scratch can leverage already existing nets and retrain them for other purposes. New hardware such as the Movidius Compute stick and the planned Volta Tensor-GPGPU chipsets will offer hardware acceleration for training and classification tasks that will make small scale training more affordable.

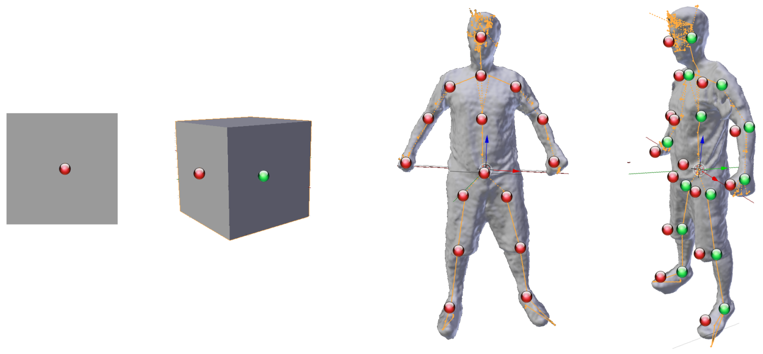


Figure 3.8: A visualization of the problems encountered if we naively try to directly re-project 2D joint positions back to 3D using the depth map registered to our color frame. This is natural since the depth sensor can only sample points in the surface of the observed volume. Left: This is a very simple case of a cube where we can see the discrepancy of the real center of the cube with the depth measurement we get. Right: For a more complex human model of varying thickness, in the best case all of the 3D points extracted (red color) will lie on the surface of the human volume. In case of self-occlusions or other occlusions the 3D points will be even further away from their true location (green color).

Given a very large dataset with training examples they can achieve incredible classification precision as demonstrated by the LeNet inception classifier [110] with its 22 layer network in figure 3.7 .

Other positive features of neural networks despite their ability to perform generic classification with incredibly accuracy in very hard domains, is that they require little to no pre-processing at runtime. Typically the only non-automatic parts of them have to do with performing some sort of post-processing on the output which is also the case in the 2D body detector chosen for our framework.

The method selected as our 2D Body tracker is the [13] which features both classification for joints as well as confidence affinity fields that contain hints for joint orientations. Their combination results in a very robust detector that not only can successfully detect human parts but is also capable of correctly clustering and grouping up the detections to form full 2D skeletons.

3.2.5 Uplifting 2D bodies to 3D

We have described a variety of methods that can perform 2D joint detection in RGB images. Our final goal is to use them as cues for our final 3D tracker formulation. However since working with an RGBD sensor gives us the luxury of having depth maps registered to the incoming RGB images, we can maybe take advantage of this fact. We already have 2D coordinates that correspond to color pixels as well as color to depth registration so we can try to “upgrade” our 2D estimations to 3D in order to acquire even stronger cues for our optimization formulation. The easiest way to facilitate this 2D to 3D upgrade is to directly sample the depth map at the pixel values ouf our 2D detection. We know the intrinsic calibration parameters of the camera f_X, f_Y, c_X, c_Y and we also have a $Depth_{Sample}$ for each of the pixels of our RGB frame. We can thus convert X_{2D}, Y_{2D} to a 3D point X_{3D}, Y_{3D}, Z_{3D}.

$$X_{3D} = (X_{2D} - c_X) * (Depth_{Sample}/f_X). \tag{3.1}$$

$$Y_{3D} = (Y_{2D} - c_Y) * (Depth_{Sample}/f_Y). \tag{3.2}$$

$$Z_{3D} = Depth_{Sample}. \tag{3.3}$$

This initial naive uplifting to 3D is correct mathematically but unfortunately the depth map only accounts for the surface of the volumes observed in our scene. This means that our reprojected 3D points will all correspond to the surface of the human body as seen in figure 3.8 and will not lie in their true position.

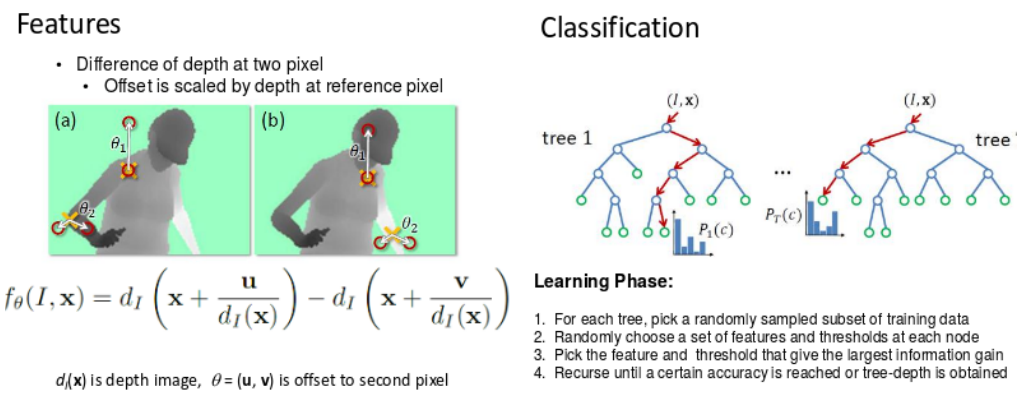
An easy improvement we can perform is to artificially push back all of the points since having the 3D scan of the body as well as the 2D detector we can calculate the thickness of the observed body and apply a translation to the reprojected points. Of course and once again noted, this will still not work for points in case of serious occlusions but the 3D positions overall will offer us a stronger cue since they also constrain the scale of the model, and we can handle imperfect alignment with our generative component.

More information on the reasons behind the reasons of using this 3D approximation will be offered in the "Framework" chapter.

3.3 3D body detectors

3.3.1 3D DNN detectors

An interesting work that extends the 2D DNN Detectors mentioned in the previous section is [99]. The DensePose method works by establishing dense correspondences between RGB images and a surface-based representation of the human body. Its dataset consists of 50K persons from the COCO dataset that are manually annotated and have assigned coordinates of joints to depth surface pixels. The resulting dataset is then used in order to train a CNN-based system that can retrieve surfaces even in the presence of background, occlusions and scale variations. The final system improves accuracy by cascading and yields real time results. It is important to note that while the main concept remains the same as the 2D detectors the clever formulation of the 3D correspondences through the UV parametrization of the body makes an important differentiating factor that enables 3D output from RGB only 2D input with the only overhead beeing the extra annotation during the dataset creation phase.



$$f_\theta(I,\mathbf{x}) = d_I\left(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}\right) - d_I\left(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}\right)$$

$d_I(\mathbf{x})$ is depth image, $\theta = (\mathbf{u}, \mathbf{v})$ is offset to second pixel

Learning Phase:

1. For each tree, pick a randomly sampled subset of training data
2. Randomly choose a set of features and thresholds at each node
3. Pick the feature and threshold that give the largest information gain
4. Recurse until a certain accuracy is reached or tree-depth is obtained

Figure 3.9: A summary of illustrations from [104] that give us insight on the Random Forest classifier. Left: Visualization of features used by the random forest algorithm which is difference of depth. Right: Learning is facilitated by creating a forest of binary decision trees that use random samples and feature sets. Each decision node is selected to provide maximal information gain at each step.

3.3.2 Random forests

One of the most widely and commercially successful 3D body detectors in the industry, that also coincided with the development of the Microsoft Kinect sensor was the Random Forest Body Detector [104] from Microsoft Research. It managed to achieve constant framerates of 200fps on commodity hardware and formed the core component of the X-Box Kinect gaming platform. The method uses a very big training set of 500K frames that was acquired by 15 real actors of different body types that where recorded doing various different activities using a Motion Capture system. The motion capture system automatically annotated the resulting 3D meshes that where then processed by a learning algorithm. The training created a decision "forest" that comprised of multiple binary decision trees 20 layers deep each. Each of the trees was a basic classifier that could predict the likelihood of a pixel x belonging to body part class c. Features used where differences of depth as seen in Figure 3.9 and non-leaf nodes corresponded to thresholded features while leaf nodes modeled the learned distribution $P(c|I,x)$ where I is the input image. The reference implementation of the algorithm used 3 different trees with 31 different body parts and 300,000 training images per tree randomly selected from 1 million training images. Each Image had 2000 training example pixels per image and 2000 candidate features with 50 candidate thresholds per feature. It overall offered a very accurate method as seen in Figure 3.10 with a very compact way of performing comparisons and calculating feature values. Up to this day it still remains one of the most widely used body detection algorithms for RGBD sensors with various improvements done in its more recent versions that has been included the Windows Kinect SDK. A secondary notable implementation of the same body

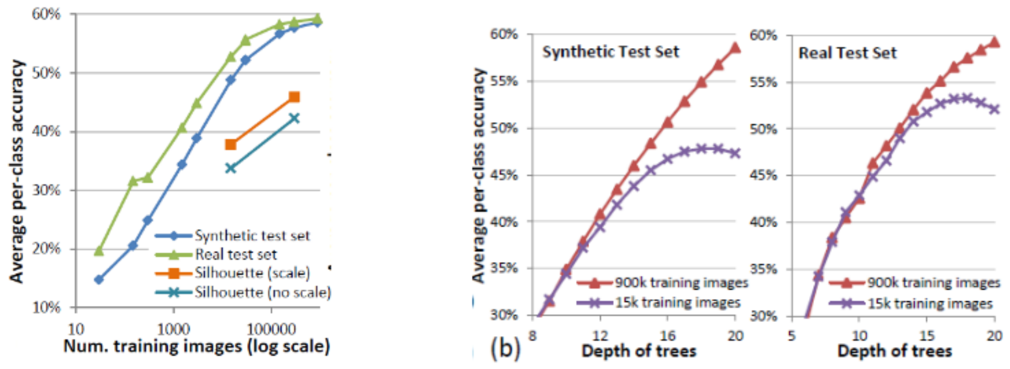


Figure 3.10: Left: Random Forest body detection classification accuracy as a function of the training set size. Right: Classification accuracy as a function of tree depth and training set size.

detection algorithm that has unfortunately been discontinued is the NiTE tracker of OpenNI Project. It has similar quality to the Microsoft implementation although it suffers from initial delays when first observing a new body since it requires calibration for a few seconds before starting to provide detections. This fact makes the method ill-suited for rapid detections, i.e. when observed persons enter and exit the scene very fast. A second problem with the method is the very big training set that requires extensive computation time along with the general inability to include poses that have not been considered in the initial training.

3.3.3 Dynamic programming

Finally the last category of 3D body detectors are the detectors that use dynamic programming and classic image processing techniques in order to perform detection. A method that demonstrates this is [58] where there are also extensive comparisons with the Random Forest method mentioned in the previous section. The input 3D Depth map image is processed through a series of filters which simplify it and then an iterative algorithm fits a human model using the extracted information. Spatial and foreground segmentation can be easily performed from the RGBD frame pair since we have depth information as well as camera intrinsics. Thus the background as well as small disconnected regions can be easily wiped out. After this initial preprocessing most of the remaining image is foreground and after the application of erosion filters, corner detection as well as Hough line transforms, basic shapes of arms, hands and feet can be derived. A model of the human body along with admissible limb sizes is iteratively fitted until the segmented volume is sufficiently explained by the detector. The detector can even detect arms with a fully occluded body and the head is not visible. Overall the method showcases incredible performance does not need specialized GPGPU hardware and detections are robust enough for it to be also used as a gesture detector. Instead of the aggregated training procedure of the other methods we can specify hard constraints on detection parameters (body dimensions) and fine tune it for specific tasks but the only down side is that there is no formal proof of the optimality of the resulting detector since it does not rely on statistics and mathematic formulations like random forests or back propagation but instead on a hand crafted rules based on human observation heuristics and classic pattern recognition primitives.

Chapter 4

3D Tracking

Having covered the fundamentals for 3D human model acquisition and methods we can use to get a rough estimation of the position of an observed human, the next step is to delve into 3D tracking. 3D Tracking can be thought as the logical extension of the detection problem seen in Chapter 3. However instead of dealing with a very large space where potential humans may be, our search space is much more limited since we typically know its previous configuration. The second difference is that instead of just wanting to have an approximation of the position of a detected subject we are interested in a very precise and fine grained solution. Lastly as the tracking title suggests the accuracy of the estimations must persist across all of the visible motions in the view-port of the camera regardless of movement that may also contain temporary self-occlusions.

4.0.1 Direct Linear Transform (DLT)

Starting with the simplest case of a rigid object with no articulations, and as an immediate thought when any sort of frame to frame registration via computer vision comes to mind, the most direct and purely mathematical way to solve the tracking problem is by using direct linear transformation (DLT). It can sometimes be more plainly referred as the 8 point algorithm or PnP (Perspective-aNd-Point problem). Assuming that we have some accurate feature points in the visible image, DLT is a thoroughly researched and well documented method of obtaining the pose of a known object. It can work using both 2D as well as 3D constraints (since 3D constraints simplify the resulting system of equations). The features used are typically SIFT [50], SURF [4], FAST [94] or ORB [96]. The different feature sets come with different performance/accuracy treadeoffs and are subject to different licensing restrictions. The quality and distinctiveness of the extracted features is of paramount importance to enable the method to produce acceptable results. The features extracted are matched using a nearest neighbors strategy like [63]. After having a set of corresponding features the next step is to perform an iterative RANSAC [23] optimization where error minimization occurs to establish a good pose estimation. Finally a Kalman filter [77] can be used as a way to

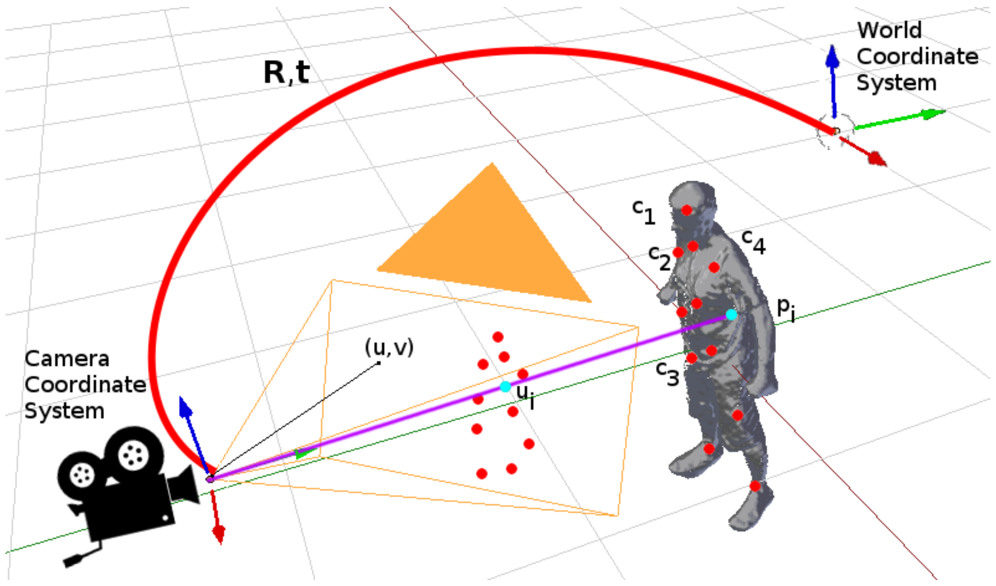


Figure 4.1: Equation 4.1 visualized assuming a rigid human model. In our case however the model is fully articulated and so the DLT method does not suffice as a method to perform tracking.

discard erroneous poses and improve overall tracking accuracy.

Assuming we know the intrinsic parameters of the camera (f_x, f_y, c_x, c_y) and having point to point correspondences the basic equation governing a 3D scene projected on a 2D surface is the following. What we are trying to extract is the pose of the camera compared to the tracked object (the matrix with the R and t rotation and translation components). Getting the inverse of this matrix we can derive the transformation of the object in the coordinate frame of our camera.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.1}$$

Since the method described in the previous paragraph is one of the most fundamental computer vision building-blocks and the reader is probably already very familiar with it there is no reason to go into further details. The book *Multiple View Geometry in Computer Vision* [29] is an excellent source for more information for an interested reader and in our case we do not deal with rigid well-textured objects so it does not offer a powerful enough tool for our generic human tracking case. It is however a good introductory method for this tracking chapter, it defines a baseline for a single object problem and it could be extended using inverse kinematics.

Unfortunately since we are not dealing with rigid geometry but work with moving and articulated deformeable humans, this mathematical formulation is not strong enough to cover us. So we must resort to a better way to track the observed motions, and that way is offered by generative optimization methods.

4.1 Generative Methods

As previously stated in the introductory chapter at sections 1.3 and 1.4.1, generative model based methods are very well suited for 3D tracking. The most relevant generative method tackling a similar use-case as our work that also slightly predates the submission of our paper in WACV18 [82] was VNect [55] and established a single view RGB based 3D regressor for 2D neural network output. In this section we will cover the basics of various prevalent generative techniques that can be used to perform model based regression and fill in the role of our optimization function.

4.1.1 Iterative Closest Point (ICP)

The iterative closest point algorithm [5] and its variants [97] are iterative methods that are easy to implement and that can be used to align two 2D or 3D point clouds and return the transformation matrix that leads from one to the other. As the DLT method that was mentioned in Section 4.0.1 they where originally intended for rigid objects and are widely used for robot navigation and localization as well as 3D reconstruction.

The vanilla ICP algorithm consists of the following discrete steps :

1. For each of the points in the source 3D cloud we find a match with the closest 3D point in the target point cloud.
2. We calculate a transformation (rotation/translation) that minimizes point to point distance using a root mean square (RMS) metric.
3. We perform the transformation calculated by the previous step to all of the source points.
4. We check if average point distance between matched points is less than a threshold, or we have exceeded our time limit. If not we repeat the process from step 1 with our new source cloud.

Of course there are many improvements and intermediate steps that can be added that improve results by rejecting outliers, using different strategies of penalization, including point weights and many other ideas that boost efficiency and that can be found here [97].

Although the ICP algorithm is much closer to our 3D tracking target compared to the DLT method and a version of it is actually used by our offline reconstruction module. More specifically during the second step of reconstruction when the rigid 3D model is skinned. Using a derivative of this method correspondances are established between the rigid human mesh and the deformable SCAPE [1] skeleton seen in [21]. Although it is a useful method for offline computations it still suffers from many shortcomings, mainly its performance since it lacks parallelization and lacks provisions for articulated meshes (although modifications that handle this case exist). What is especially problematic is that in comparison to other methods like PSO (Section 4.1.3) and even if we where interested in just tracking rigid objects, ICP is both slower and less accurate overall for reasonable optimization times as seen in this work [123] for noisy depth sources like our low cost RGB+D sensor.

4.1.2 Particle Filters (PF)

Particle filtering, also known as the condensation algorithm or hierarchical particle filtering (in its more popular implementation geared towards high dimensional spaces), is a technique that has been successfully applied for motion tracking of articulated objects that include humans [19] and human hands [53]. These cited works offer a very detailed explanation of the algorithm. The method is relevant to our problem although out of the scope of this thesis, thus we will just give a brief synopsis in this section.

Particle Filters perform tracking via the use of an array of models that are called auxiliary models. These models store and leverage probabilistic information about the state of the tracking problem. Their state is in turn used to update a main model which actually holds the tracking result we are searching for. By integrating incoming frames and re-estimating the probability density as well as maintaining a likelihood model that compares the current frame we want to calculate with history, the method manages to respond well both in terms of accuracy as well as performance being able to achieve very fast frame-rates (90fps for a 27dof hand-tracking problem [53]) and be robust to noise. However the dual component objective function we planned to use in order to balance the depth and neural network detections as well as the even bigger parameter size of our detailed 3D scanned model where the main reasons not to choose Particle Filters as our optimization engine.

4.1.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) [15] is a stochastic method that performs optimization by iteratively improving a candidate solution with respect to an error term characterizing its quality (objective function). PSO has been applied successfully to a number of vision problems such as object detection [108], head pose estimation [75], 3D hand tracking [70, 71], 3D tracking of hands in interaction with objects [45] as well as 3D human pose tracking [117, 57]. The popularity of PSO as an optimization strategy for articulated motion tracking comes from its ability to handle large search spaces and noisy, multi-modal, non-differentiateable objective functions. Moreover, as shown in Section 6.6.2, PSO is ideal for parallel implementation on modern GPU architectures, permitting interactive framerates and a 100× speedup compared to the serial implementation.

PSO maintains a population of candidate solutions, called particles, that have a position p and a velocity V in the search space. The movement of each particle p_i is influenced by the best position P_i this particle has ever visited up to the current iteration/generation, and simultaneously guided towards the globally best known position G in the search space (i.e., the best of all P_is). Both these positions are updated as better ones are found by other particles. The update to the k-th generation is described by:

$$V_{i,k} = r_1 c_1 \left(P_i - p_{i,k-1}\right) + r_2 c_2 \left(G - p_{i,k-1}\right) + \omega V_{i,k-1} \tag{4.2}$$

$$p_{i,k} = p_{i,k-1} + V_{i,k}, \tag{4.3}$$

where $p_{i,k}$ and $V_{i,k}$, respectively, denote the position and velocity of the particle p_i at the k-th generation, r_i are samples of the uniform distribution $U(0,1)$, and c_1, c_2 and ω are parameters controlling the convergence speed of PSO. The particles are allowed to move within predefined ranges along each dimension of the search space (in our problem, these ranges are shown in Table 5.1). To enforce this constraint whenever it is violated, the respective velocity $V_{i,k}$ is reduced up to the point that the constraint is again satisfied. These steps are followed iteratively, until a fixed upper bound of generations is reached. Parameters c_1, c_2 and w, are set as proposed in [15], that is, $c_1 = 2.8$, $c_2 = 1.3$ and $\omega = 2/\left|2 - \psi - \sqrt{\psi^2 - 4\psi}\right|$, where $\psi = c_1 + c_2$.

For our 3D human pose estimation problem, PSO is used to minimize the objective function of Eq.(5.2) over candidate solutions h. For each incoming frame, particles are initialized around the solution for the previous frame. The space around that solution is made large enough to include the J^e estimation for the current frame. This, together with the E_J term in the objective function, are the elements that permit to the method to perform without initialization and to recover from potential tracking drifts. Perturbations [69] are used for particles during the optimization procedure in order to help them escape local minima towards the global best solution. We also keep a history of detections and the last 5 estimated poses are also considered as particles in every new frame to help us recover faster from low quality OpenPose estimations that may cause a momentary drift. In order to evaluate fairly the solution proposed in this paper, no motion prediction model is used to initialize PSO particles and no smoothing is being performed in the sequence of results. However, such techniques, are expected to improve pose estimation accuracy when employed.

4.2 3D Tracking with PSO

We selected Particle Swarm Optimization as our optimizer between all other available options and its detailed examination covered the way it works. What remains before proceeding to the Chapter where we will recount the full body tracking framework is to provide information on the use of PSO on simpler 3D tracking problems.

4.2.1 The single object case

The simplest case for model based 3D tracking using PSO and an RGBD camera is the case of a single object. The parametric space for a single object is incredibly constrained when compared to our articulated body as we will see later. An object can be described using 7 parameters that determine its position and orientation. Positions are encoded using 3 coordinates (X, Y, Z) that position the object in our world frame and orientations are encoded using a quaternion that consists of four parameters (qX, qY, qZ, qW). The use of quaternions as a rotation representation has multiple benefits. The first and most important is that random sampling of the solution space may easily lead to rotations that contain gimbal locks. Quaternion representation does not exhibit this problem. A second positive feature of quaternions is that their 4 parameters offer a more smooth way to sample intermediate orientations since,once again, we consider random samples of quads around a previous state.

For each combination of 7 numbers we can come up we can render the model in a virtual depth buffer and retrieve its depth profile. This can be then easily differentiated to the observation and so we can tag each combination of 7 numbers with a score. A score of 0 means that the combination fits perfectly to the observation. Our problem thus is a minimization problem that in the case of a single object has only 7 degrees of freedom.

4.2.2 Articulated objects and occlusions

Each extra object introduced in the scene increments the parameter space by 7 more degrees of freedom. Ideally we could try to solve the problem by solving in parallel two instead of one single object problems trying not to invest a lot of resources to the problem but unfortunately occlusions and the curse of dimensionality do not allow us to get away with this. We cannot treat the two objects tracking problem as two smaller $7dof$ problems but have to face them as one $14dof$ which is exponentially bigger. The reason why is that when we consider multiple objects in our scenes but our optimizer only fits a single model hypothesis every time, occluded objects will never fit correctly to our hypothesis unless the occluder is also taken in consideration. We thus need to always render all objects at the same time since only joint combinations of all the objects will render similarly to our observations. This however means that in order to deal with occlusions we have to accept an exponentially growing parameter space. The problem becomes even worse when we consider articulated objects since each of them will further penalize performance by increasing the parameter space much more than the $7dof$ of a simple rigid object. 3D Hands can be encoded using 27 parameters and our detailed human body model has 36. Methods such as [45] and [85] attempt to improve the situation but nevertheless the human body tracking problem can be roughly thought as simultaneously tracking 5 different objects in terms of computational complexity, certainly not a trivial task.

Chapter 5

Our proposed tracking framework

The proposed optimization framework consists of well defined and self-contained modules that are limited in scope and are dedicated to solving one problem each. The details of each of them have been covered in the previous chapters and with that in mind we can now proceede to their combination. Working in tandem they provide a powerful and robust tool that can facilitate 3D Human Tracking tracking. A schematic overview of the framework can be seen in Fig. 5.1 and the proposed method is summarized in Fig. 1.1.

5.1 Overview of proposed framework parts

3D Reconstruction As explained in detail in Chapter 2 the 3D reconstruction algorithm used to recover human models is [102]. The acquired models are initially rigid and are converted to a fully skinned and deformable model that can take poses using [21]. All the resulting data is stored using the .tri file format also described in subsection 2.4.4.

2D Joint estimation Our problem formulation as seen in Figure 5.1 requires a detector to prvoide seeds that will become the solutions for each frame. From all of the options considered in Chapter 3, method [13] was picked as the most robust and generic detector.

3D Tracking From the optimization frameworks discussed in Chapter 4, PSO was chosen as it both has the ability of dealing with the large parameter space we are considering as well as the desirable features of escaping local maxima, very good parallelization qualities that can be leveraged using a GPGPU implementation and the ability to extend the body model at will as well as incorporate additional soft limits that can be useful in an extended version of the problem.

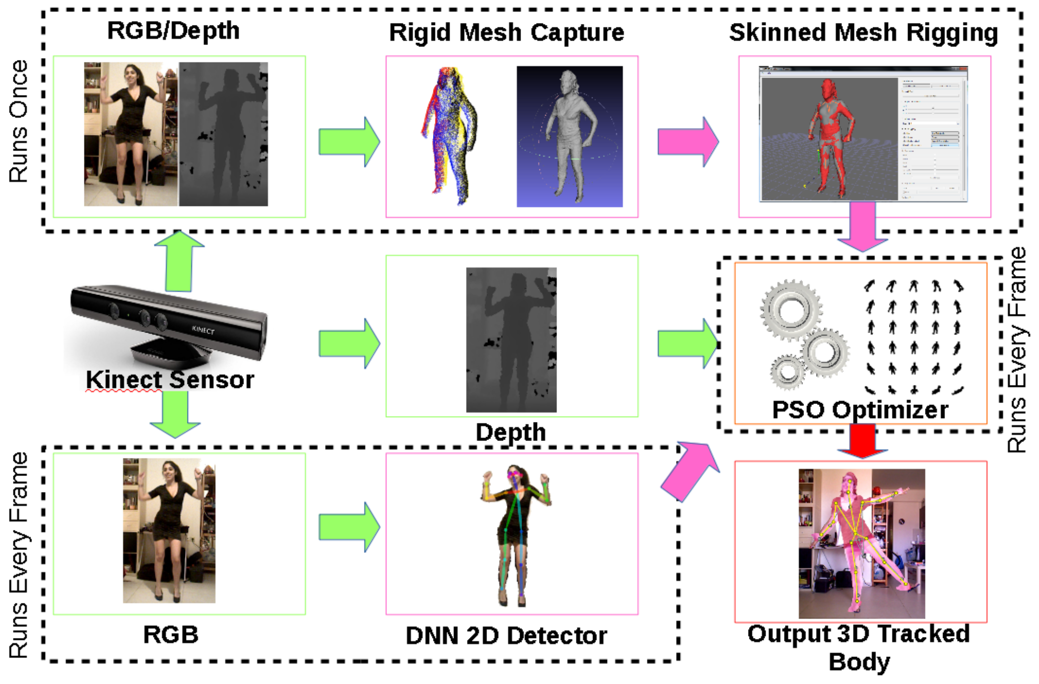


Figure 5.1: A block diagram that gives a connectivity overview for the various modules of the proposed framework. Green arrows are raw input from our sensor, pink arrows symbolize intermediate input provided by a module of the framework, the red arrow is the final output pose. Our input source is an RGBD Kinect sensor that provides us with a stream of VGA color and depth images. The DNN 2D Detector, that is described in Chapter 3, only needs the Color stream and works as a black box that provides us with 2D estimations about joint locations. Both the color and depth stream are used by the reconstruction module that is described in detail in Chapter 2. The 4 views of the person are first combined to a rigid mesh that is then, in a subsequent step, converted to a fully skinned articulated body model. This procedure only needs to be performed once. Finally our 3D Tracker receives as input a skinned model, a depth frame and an array of 2D joints estimations and outputs a 3D tracking hypothesis that matches the observation of the Kinect sensor.

An RGBD frame is denoted as $o = (c^o, d^o)$, where c^o and d^o stand for the RGB and depth frames, respectively. The proposed method capitalizes on a parametric skinned model of the human body (Section 5.1.1). Human pose estimation in a frame amounts to estimating the parameters of the model that is most compatible with visual observations. The discrepancy between the model and the observations is quantified by an objective function (Section 5.1.3) that has two terms. The first (Section 5.1.3.1), compares the 3D structure of the observed human with the 3D structure of the rendered model. The second (Section 5.1.3.2) compares the locations of the joints as they were estimated by a neural network (Section 5.1.2) to the locations of the joints of the model hypothesis. The optimization of the defined objective function is performed effectively and efficiently using Particle Swarm Optimization [15] (Section 4.1.3).

5.1.1 Human body model

The proposed method can operate with a human body model that consists of a set of appropriately assembled geometric primitives as in [57], or with a skinned model. In this work, we are particularly interested in personalized, automatically acquired human body models. The acquisition of such a model **H** is performed in two steps (a) human body scanning and (b) model rigging. Body scanning is performed as described in [102]. Model acquisition requires only that the human subject stands with the T-pose in front of an RGBD camera, in 4 different orientations. These views are then registered automatically[1] to form a reconstruction of the human body. Articulating the scanned model through model rigging is performed automatically[2] as described in [21]. From a practical point of view, given the mentioned automated tools, the acquisition of a body model **H** can be performed in less than 2 min.

A body model **H** acquired this way has a total of 94 bones which also account for parts of the face, hand fingers etc. For the large-scale body tracking scenario we are interested in, we restrict ourselves to a subset of those body parts and the respective joints as listed in Table 5.1.

The 3D position of **H** is represented with three parameters. Four more parameters encode a quaternion-based representation of its global orientation. Joints are represented with roll/pitch/yaw angles. Thus, a pose of **H** is represented as a $3 + 4 + 29 = 36$D parameter vector h. The limits of 29 of these parameters are shown in Table 5.1. Setting these limits excludes several physically implausible poses. However, certain combinations of valid (i.e., within limits) joint angles still result in impossible body configurations.

In order to grasp the complexity of the 36D parametric space of body motions we can devise a simple thought experiment. For our experiment we will assume that we somehow have a 36D vector that exactly describes the configuration a person at a particular time. We can also assume that the person will move very

[1]Human body reconstruction software: `https://goo.gl/jFFj6a`
[2]Human body model rigging software: `https://goo.gl/AEtX96`

Joint	*roll* *min*	*roll* *max*	*pitch* *min*	*pitch* *max*	*yaw* *min*	*yaw* *max*	*weight* w_i
Neck	-0.4	0.4	-0.3	0.3	-	-	1.0
Spine	-0.5	0.5	-0.2	0.5	-1.5	1.5	-
LC	-0.1	0.1	-	-	-0.15	0.1	-
RC	-0.1	0.1	-	-	-0.1	0.15	-
LS	-4	4	-1.8	1.57	-2	1.4	1.0
RS	-4	4	-1.8	1.57	-1.4	2	1.0
LE	-	-	-	-	-2.5	0	1.4
RE	-	-	-	-	0	2.5	1.4
LW	-	-	-	-	-	-	1.8
RW	-	-	-	-	-	-	1.8
LH	-1.57	1.57	-0.78	2	-1.2	0.5	0.4
RH	-1.57	1.57	-0.78	2	-0.5	1.2	0.4
LK	-	-	-2.8	0.1	-	-	1.4
RK	-	-	-2.8	0.1	-	-	1.4
LA	-	-	-0.6	-0.6	-0.7	0.7	1.8
RA	-	-	-0.6	-0.6	-0.7	0.7	1.8

Table 5.1: Limits (in radians) of joint angles of the human body model **H**. Besides Neck and Spine, the joints are coded with two letters. The first (L, R) stands for Left/Right and the second (C, S, E, W, H, K, A) for Collar, Shoulder, Elbow, Wrist, Hip, Knee and Ankle. The last column is the weight of the particular joint (w_i) when calculating scores in the objective function E_J

slowly from frame to frame in order to help our case. In this theoretical situation each of the parameters can remain numerically the same from frame to frame, it can increase by d or decrease by d, where d is a slow movement distance constant. This means we have 3 possible outcomes for each of the joint states from frame to frame. When we combine all of them this means that the number of states is $3^(36)$ or 150.094.640.000.000.000 cases that need to be checked and discarded until we can find the correct one. To futher simplify the situation for us and assuming that we will not examine any non essential body parts like feet or the head, we can decrease the parameters to 22 which leads to $3^(22)$ or 31.381.059.609 distinct events. Our hardware allows a maximum of 4096 renderings which can just explore a fraction of all possible states with acceptable performance. This thought process can be very enlightning and very indicative of the effectiveness of our optimizer that gracefully handles this huge parameter space.

Given an instantiation h of the model **H** and camera calibration parameters, we can render **H** to the view of the camera, obtaining color and depth maps $r = (c^h, d^h)$ that are comparable to the observations.

5.1.2 Localizing body joints

Given a color frame c^o, we employ the OpenPose neural network [13, 120] which computes the 2D locations j^e of human body joints. Any source of 2D/3D joint locations can be used, however OpenPose has been used because of its accuracy and robustness. We expect the 2D estimations of the joints not to be perfectly accurate but we also expect the detected joints to have some consistency in the temporal domain and in relation to each other. Given j^e, we can sample the depth information d^o to get a coarse estimate of the 3D locations J^e of the joints. Inaccuracies come from the fact that these 3D points lie on the surface of the body, while joints do not. Such problems are even more pronounced for occluded joints, i.e., for the back shoulder in a side view of a body or in cases of crossed arms or legs. Despite such inaccuracies, a coarse estimation of the 3D locations J^e proves useful during optimization. A minor improvement we can do for this inaccurate 2D to 3D uplifting as seen in Subsection 3.2.5 is to push back every joint 5-10 centimeters inside the body depending on the joint type, this is still a very inaccurate 3D estimation but it is more than enough as a building block for the objective function to be initialized and then guide us close to the solution where the 3D mesh will in turn dominate the scoring function and yield the accurate estimation we desire.

5.1.3 Objective function

An objective function $E(h, o)$ has been designed to quantify the discrepancy between a model hypothesis h and the actual observations o. Estimating the human pose at a certan frame amounts to finding the model parameters h^* of $\mathbf{H}$ that minimize $E(h, o)$. In notation,

$$h^* \triangleq \arg\min_h E(h, o). \tag{5.1}$$

$E(h, o)$ consists of two terms, E_D and E_J. Specifically,

$$E(h, o) = w_D E_D(h, d^o) + w_J E_J(h, o). \tag{5.2}$$

The first term measures the discrepancy between the observed depth map and the depth map resulting from the rendering of $\mathbf{H}$ according to h. The second term measures the displacement between the locations (both 2D and 3D) of the human body joints. The first term is weighted by a constant w_D whose value is determined experimentally in Section 6.6.1. The weight w_J is set to 1.

5.1.3.1 The depth term E_D

For a given hypothesis h, we render $\mathbf{H}$ to obtain a color image c^h and a depth map d^h. d^h is comparable to the actual, observed depth map d^o and their similarity is a strong indication for a correct hypothesis h. This motivates the following definition of the error term E_D:

$$E_D\left(h, d^o\right) = \frac{1}{N_P} \sum_{p \in B} C\left(|d_p^h - d_p^o|, T\right). \tag{5.3}$$

In Eq.(5.3), E_D sums the absolute depth differences $|d_p^h - d_p^o|$ for all points p that belong to a bounding box B containing the human figure. The clamping function $C(x, T)$ returns x if $x \leq T$ and T otherwise. This is used to robustify the error term and prevent spurious points/outliers from affecting it too much. In our implementation we set $T = 30$cm. It is worth noting that other types of information which are more stable could also very well be used in the generative part, not necessarily from the visible spectrum but from an infrared or thermographic camera or another sensor working in a different part of the em-spectrum.

In RGBD camera depth readings, a value of 0 represents lack of measurement due to e.g., reflective materials or infrared interference. Such points are ignored in Eq.(5.3).

The normalization term N_p requires special attention. Setting N_p equal to the number of rendered model points is a bad choice, because this promotes hypotheses that project to only a few pixels in the image, instead of hypotheses that explain all the available observations. To avoid this, N_p is set equal to the number of points inside B that are close to the depth profile of the previous solution. This way, all different hypotheses for a certain frame share the same normalization. We also maintain normalization which is again important for correctly handling scale changes. Another helpful step we perform while creating the sum N_b is to remove pixels that where not included in the count and are also not part of the rendered hypothesis from the bounding box b. This way the sum 5.3 will exclude background pixels and will ideally only contain information about our foreground and all hypothesis pixel making the gradient of the objective function closer to the truth and easier to traverse for PSO.

5.1.3.2 The joints location term E_J

As discussed in Section 5.1.2, OpenPose provides estimations j^e of the 2D locations of human joints, which can then be lifted to 3D estimations J^e by exploiting the depth information d^o. Additionally, given a hypothesis h of $\mathbf{H}$, we can estimate the 3D joint locations noted as J^h and also render them in the camera view to obtain j^h. Thus, we define an error term E_{J2D} that penalizes discrepancies between the hypothesized (j^h) and estimated (j^e) 2D locations of joints:

$$E_{J2D}\left(h, c^o\right) = \frac{1}{W} \sum_{i=1}^{n_J} w_i ||j_i^h - j_i^e||_2. \tag{5.4}$$

Similarly, we define a term E_{J3D} that penalizes discrepancies between the hypothesized (J^h) and estimated (J^e) 3D locations of joints:

$$E_{J3D}(h,o) = \frac{1}{W}\sum_{i=1}^{n_J} w_i ||J_i^h - J_i^e||_2. \tag{5.5}$$

Some types of joints are localized by OpenPose more accurately than others. As an example, the head and the knees are more accurately localized compared to hips. In order to compensate for this behavior of the detector, we employ a weighting scheme in both the 2D (Eq.(5.4)) and 3D (Eq.(5.5)) terms of the objective function that prioritizes more accurate joints. We use $w_i = 0.2$ for hips and $w_i = 1.8$ for ankles and wrists. In Eqs.(5.4) and (5.5), $W = \sum_{i=1}^{n_j} w_i$, where n_J is the number of all considered joints.

The aggregated joints location term E_J is defined as

$$E_J(h,o) = \alpha E_{J2D}(h,c^o) + (1-\alpha)E_{J3D}(h,o), \tag{5.6}$$

where $\alpha = 0.97$ is a constant that balances the contributions of the 2D and 3D error terms and which was set experimentally (Section 6.6.1). It appears that this value of α gives a dominant role to E_{J2D}. However, this is not the case, as α needs to compensate also for the different arithmetic scales in which E_{J2D} and E_{J3D} are measured.

For each x parameter vector we can easily go back to our H mesh and recursively calculate the transformations of each of the joints and also get their 2D projections since we know the intrinsics of the observation camera. This way for each joint j we can get a direct euclidean distance in both 2D and 3D space that is our 2D and 3D Neural Network error estimation.

5.1.3.3 Incorporating heuristics to the objective function

Since the main focus of this work is assessing model quality and neural network importance this objective function serves as a very clean middle-ground solution where by setting factor values to zero we can switch off the neural (5.5 5.4 or depth 5.3) terms by zeroing W_{2D} , W_{3D} and/or W_{depth} and clearly identify their impact in tracking quality.

Increasing the value of W_{depth} makes the role of the high quality model more important, while reducing W_{depth} close to zero results to the model stops making any difference since we are just fitting the kinematic skeleton joints to the 2D proposal of the neural network, A value of zero for W_{depth} make this method perform similarly to Vnect [55] and LCR-Net [93].

It is worth mentioning that object tracking can be easily included by adding a further 7 dof per object (XYZ and a quaternion) in the parameter vectors and by supplying a mesh of the tracked object [45]. Although c^h is chromatically accurate, color observations are very susceptible to illumination changes, shadows and since depth information is also present in this work, color comparison is omitted from the generative component and only used from the CNN detector. The neural network 2D joint output will never have a zero distance to our model (since neural network detections are very good overall but not so accurate), but this term 5.6 of the objective function H is really very valuable as x gets more distant from the optimal solution and can guide the solution process back to where $Depth_{Error}$ becomes dominant.

5.1.3.4 CPU/GPU acceleration for the objective function

The implementation of our framework relies on the MBV toolkit [68] which powers the rendering and optimization engine utilized in this work. A very detailed technical report on its inner workings can be found here [46] and will answer most questions. In the context of our work and trying not to go off-topic a brief overview will also be given here.

Modern GPU cards are built to perform real-time rendering of immense volumes of polygons. The workload we offload to the GPU for our task is hardly challenging, novel or surprising. However the main differentiating factor between this and other methods is the way that hypothesis evaluations are performed since they turn out to be the most computationally demanding part of any generative method.

During the development of this method the first and easiest approach that was implemented to evaluate rendered hypothesis was to render each of them serially in the GPU and then retrieve them and score them using the CPU in order to be able to debug the objective function weights and better study and log its behavior. However this had many negative performance penalties with the most pronounced being the serial evaluation, the PCI-Express bus bottlenecks between main system memory and GPU memory and bad resource utilization due to poor concurrency.

The best way to perform the said tasks is to use a technique called deferred-shading that performs tiled rendering and stores results on GPU texture memory. These rendered results can be directly scored against our observations using the CUDA GPGPU architecture and the resulting difference maps can be scanned and reduced into a numeric score that can be very efficiently communicated back to system memory to seed the next PSO generation. This way we only use the GPU bus intensely one time during the initialization of the application when we first upload our mesh geometry. All subsequent GPU queries just contain the incoming frame and a list of appropriate transformations for each of the renderings/hypothesis/PSO particles. Concurrency is maximized and every PSO generation is computed in a very fast parallel step.

The typical speed-up of applications ported from a CPU to a GPU architecture in the literature is typically 10x but in our case framerates increase from 0.7fps to 10fps which is an incredible improvement. However reducing the model vertex count, as well as using less PSO generations as well as better performance from the DNN 2D joint detector could further improve framerates.

The following code segment is the Thrust/C++ objective function implemented at the core of the presented work to enable the objective function to perform depth scoring in a GPU Accelerated fashion.

```
//Shorthands to make code easier to read
#define YES 1
#define SMALLEST_FLOAT 0.0001

//Structure that keeps all relevant pixel information
struct sample
{
 float difference;
 unsigned short isCorrect;
 unsigned short isExisting;
};

//Structure that keeps all data results
struct ObjectiveHelperGPU::PrivateData
{
 thrust::device_vector<sample> perPixelAllInOne;
 thrust::device_vector<sample> sumPerPixelAllInOne;
 thrust::device_vector<int> keys;
};


//This is the code segment that performs
//reduction to sum up our calculations
struct SampleReductor :
 public thrust::binary_function
 <const sample &,const sample &,sample>
{
  __host__ __device__
  sample operator()(
                   const sample & lhs,
                   const sample & rhs
                 ) const
  {
    sample summed;

    summed.difference = lhs.difference + rhs.difference;
    summed.isCorrect  = lhs.isCorrect  + rhs.isCorrect;
    summed.isExisting = lhs.isExisting + rhs.isExisting;
    return summed;
  }
};


//This is the function that performs depth difference
//calculation and thresholding
struct SampleCalculator:
public thrust::binary_function
<float,float,sample>
{
 SampleCalculator(
                    float depthSampleVariance,
                    float depthClampMillimeters
                                   ):
 depthSampleVariance(depthSampleVariance),
 depthClampMillimeters(depthClampMillimeters) {}


 //We want absolute distance of a haystack(hypothesis)
 //sample to the needle(observation) sample.
 //The next function is called in parallel for every
 //pixel combination and calculates our results.
 __device__
 sample operator()(float haystack,float needle) const
 {
   float result=0.0;
   float depthDifference = glm::abs(needle-haystack);

   //Result storage for this sample
   sample output={0};

   //If we have exceeded the max Variance
   //we clamp the distance to the maximum
   //allowed score
   if (needle>SMALLEST_FLOAT)
    {
     output.isExisting = YES;
     result=depthClampMillimeters;
     if (
          (depthDifference<=depthSampleVariance) ||
          (haystack>SMALLEST_FLOAT)
        )
        {
          output.isCorrect = YES;
          result=depthDifference;
         }
     }

    //We also perform thresholding and clamp results
     if (result>depthClampMillimeters)
      {
       result=depthClampMillimeters;
      }

     //We return our calculated value..
     //Third output is depth difference
     output.difference = result;

     return output;
   }

 //Fields of our SampleCalculator struct
 float depthSampleVariance;
 float depthClampMillimeters;
};
```

This means that after rendering our geometry to textures through OpenGL and uploading our observation to the GPU we just need to perform one Thrust transformation to compute all results.

```
thrust::transform
(
 haystack.begin(),   //InputIterator1 first1 ,
 haystack.end(),     //InputIterator1 last1
 thrust::make_permutation_iterator
 (
  //InputIterator2 first2
  needle.begin(), //Element to permutate
  //Index Iterator
  thrust::make_transform_iterator
   (
    thrust::make_counting_iterator(0), //Iterator
    nIdx //AdaptableUnaryFunction
   )
  ),
 m_data->pixels.begin(),
 SampleCalculator(depthSampleVariance,depthClampMillimeters)
);
```

After the transform operation performs all calculations, we end up with an array (m_data→pixels) that has each pixel depth difference stored in a separate field.We want to both sum depth differences in order to assess how compatible each hypothesis is to our observation as well as to calculate the number of rendered pixels and the number of rendered pixels that correspond to the observation needed as seen in Section 5.1.3.1. So the last step we need to perform is a reduction operation that sums up these 3 distinct results. By combining all the relevant information in to one structure (struct sample) we manage to only perform one reduction (instead of 3) and further optimize performance. After the reduction we get a result vector that has three numeric values associated to each of the hypothesis.

```
thrust::reduce_by_key
(
 // InputIterator1 keys_first
 thrust::make_transform_iterator
 (
  thrust::make_counting_iterator(0), // Begin
  hid
 ),
 //InputIterator1 keys_last
 thrust::make_transform_iterator
 (
  thrust::make_counting_iterator(cloudSize), //End
  hid
 ),
 m_data->pixels.begin(),     // InputIterator2 Values First
 m_data->keys.begin(),       // OutputIterator1 Keys Output
 m_data->pixelSums.begin(),// OutputIterator2 Values Output
 thrust::equal_to<int>(),    // BinaryPredicate
 SampleReductor()            // BinaryFunction
);
```

Finally we retrieve the results from the GPU to the CPU and compute the final depth score E_D as proposed in Equation 5.3. The N_P normalization factor is computed via an F-Measure formula (Section 5.1.3.1) and it is a very important feature of the objective function. Without it the optimizer can minimize scores by reducing hypothesis surfaces instead of fitting them to observations since smaller surfaces produce smaller numerical score values. Instead of getting good fits the optimizer can perform score minimization by just pushing the model further and further away. So we constrain this behavior by keeping track of “correct” (pixels that are both part of the hypothesis and the observation) and “existing” pixels (pixels that have been hypothesized, although not necessarily corresponding to the observation) to find an overall best match. This normalization makes scores directly comparable regardless of the hypothesis distance from the camera. The final score scaling variable "depthFactor" seen in the following code segment is what we use to balance the importance of the depth term E_D compared to the neural network term E_J and corresponds to the w_D variable in Equation 5.2.

```
//We copy our hypothesis (h) results to CPU
thrust::host_vector<sample>  h = m_data->pixelSums;

//We return results after dividing pixel distances
//with the populated pixel count..
for (unsigned int i=0; i<h.size(); i++)
{
 if (
     (h[i].isExisting >0) &&
     (h[i].isCorrect >0)
    )
 {
  a=0.5;//This can be changed to influence precision/recall
  oneMinusA=1.0-a;
  precisionBalanced = (float) a/h[i].isCorrect;
  recallBalanced    = (float) oneMinusA / h[i].isExisting;

  //Our F-Measure or Np
  F =  precisionBalanced + recallBalanced;
 }
 score[i] = (float) (h[i].difference*depthFactor*F);
}
```

The Thrust code given is the heart of the framework and has the strongest impact in performance from all the pipeline. Performance Tables 5.2 and 5.3 provide a detailed breakdown of computation times across our pipeline. The $Thrust_{gpu}$ column corresponds to the time consumed by the CUDA/Thrust code. The carefully crafted and massively parallel implementation proposed, does all pixel computations in one step and only uses one reduce operation ensures very fast performance. Computation times for our objective function range from 874μs for calculating scores on a 512x512 pixel batch of 64 particles/hypothesis to 5130μs for a 1600x1200 surface for each PSO generation. The speed-up compared to a CPU implementation is very big since we achieve more than 10 times faster performance on computations while also completely circumventing the PCI-Bus bottlenecks of retrieving the rendered images to the CPU and back and also have minimal CPU load. The only PCI-Bus usage of our framework is when we first upload our geometry during initialization. After initialization for each frame we just upload our rescaled observation and a small array of joint configurations for each hypothesis. The GPU renders our configurations, applies the objective function and then we just retrieve back the appropriate scores as seen in the above code segments. The neural network 3D Joint part of the objective just consists of a sum of the euclidean distances of 2D and 3D points as stated in Section 5.1.3.2 so it is trivial and skipped for brevity. Figure 5.3 offers a visualization of the different rendering sizes for comparison.

5.1.3.5 GPU rendering surface optimizations

As one can easily predict and experimentally measure in Tables 5.2 and 5.3 the polygon count of our 3D model as well as the rendering size selected play a very important role in achieved framerates. The fact that our optimizer requires at least 64 particles and 20 generations to perform optimizations motivates further thought about optimizations. Of course higher optimization budget definitely contributes to better accuracy as seen in the Experiments section 6 and more specifically Figure 6.13. We have to accept that we will need at least 1280 renderings to explore our 36 Dimensional space. The CUDA/Thrust code segment cannot be further simplified or optimized although a pure CUDA implementation could maybe marginally further reduce times. 3D rendering speeds can only be improved by simpler geometry, see Figure 5.2 or a better graphics card. On the other hand a smaller rendering surface both improves rendering speeds while also reducing the number of samples that the CUDA/Thrust objective function has to work on. It is thus a setting that gives us two-fold benefits. However reducing the rendering size also makes the tracking resolution decrease so once again we reach an impasse.

A final optimization that can be done is to “zoom” and crop the projection matrix so instead of making an image that fully corresponds to our entire camera view, we can crop our view and just render to the interesting part of it that contains our mesh plus a small margin around it. This technique is illustrated in Figure 5.3 where we can see the space saved without significant resolution loss. The discarded insignificant empty areas of the scene lead to better utilization of our resources for important parts of the scene instead of summing zeros for 2/3 of our image.

A very detailed book on 3D Graphics that gives information on Projective geometry is [24]. We will just briefly outline the Projection matrices used in our framework for completeness in the rest of this section. The camera we want to emulate with our 3D renderer has focal lengths f_x, f_y and focal center c_x, c_y, with an original VGA viewport (Left,Bottom,Right,Top) where $L = 0$, $B = 0$, $R = 640$, $T = 480$. The F and N variables correspond to the Far and Near planes we want to use depending on the graphics card Z-Buffer precision which is typically 16 or 24 bit, the target scene and the model scale. In our 3D Body Tracking case where we work on millimeters $N = 400$ and $F = 10000$ are good values. We can thus use the following Projection matrix P presented in column-major OpenGL compatible form.

$$P = \begin{vmatrix} -2 * f_x/(R-L) & 0 & 0 & 0 \\ 0 & 2 * f_y/(T-B) & 0 & 0 \\ 2 * c_x/(R-L-1) & 2 * c_y/(T-B) - 1 & -1 * (F + N/F - N) & -1 \\ 0 & 0 & -2 * F * N/(F-N) & 0 \end{vmatrix}$$

Smaller tile sizes are much more lightweight as seen in Tables 5.2 and 5.3. We can thus rescale the incoming observed array from a 640x480 VGA image to 160x120 QQVGA and just use $L = 0$, $B = 0$, $R = 160$, $T = 120$ to match this smaller observation size. In this case we will only have to process 19200 pixels instead of 307200 per hypothesis but even this reduction is still wasteful due to more than half of the image being empty as already stated. In order to crop and zoom it to yield maximum performance we can adjust our P matrix with the following P_c matrix. Assuming we have a 160x120 QQVGA image but we no longer want to target the range from 0,0 to 160,120 but instead (30,10) to (130,110) we first calculate X_c, Y_c, W_c, H_c by substituting our original viewport and zoomed-viewport measurements. We can thus have $L_o = 0$, $B_o = 0$, $R_o = 160$, $T_o = 120$ and $L_z = 30$, $B_z = 10$, $R_z = 100$, $T_z = 110$ where with O we have denoted the original viewport and with Z the zoomed one.

$X_c = (L_z - 0.5 * R_o - L_o)/R_o$
$Y_c = -(B_z - 0.5 * T_o - B_o)/T_o$
$W_c = R_z/R_o$
$H_c = T_z/T_o$

After we calculate these intermediate values we then proceed to calculate the P_c matrix

$$P_c = \begin{vmatrix} 1/W_c & 0 & 0 & -2 * (X_c + W_c/2) * (1/W_c); \\ 0 & 1/H_c & 0 & -2 * (Y_c - H_c/2) * (1/H_C); \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

And with it we can then calculate the final zoomed projection matrix P_z. The resulting 100x100 tile will only contain 10000 pixels almost half of the 19200 pixel QQVGA tile while also exhibiting higher resolution of our subject. Please note that matrix order, model scale as well as right vs left hand coordinate systems can cause problems in computations and one should have a very good understanding of all these details since in the event of a row-major matrix order for example the following matrix multiplication will also have to change order.

$$P_z = P_c * P$$

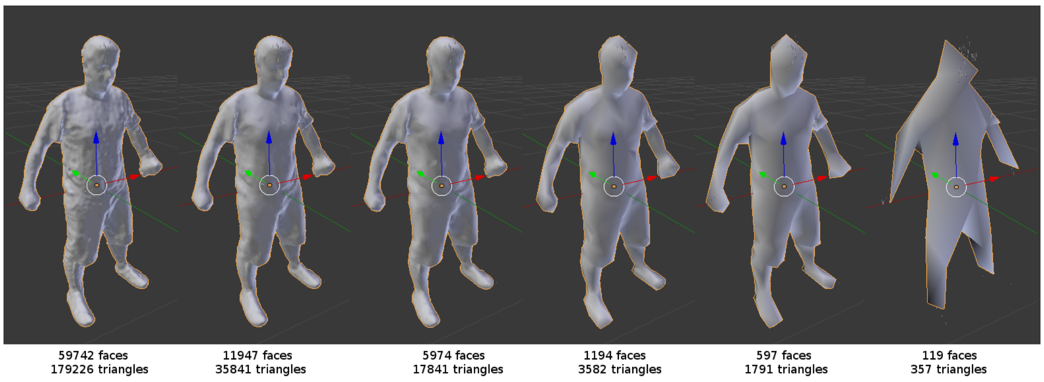


Figure 5.2: The level of detail of the meshes rendered in the optimization loop will dramatically increase optimization times, especially on lower end graphics cards. By simplifying the mesh through Quadratic Edge Collapse [34],[49] a method that is readily available on 3D modeling tools like Blender and Meshlab, we can reduce triangle counts and yield significant performance gains. In this figure we can see gradual simplified versions of our original 119484 faces model at 50%,10%,5%,1%,0.5% and 0.1% face count. We can see that our model is over-sampled and over-tessellated especially in the context of our QVGA(320x240) kinect sensor and our QQVGA(160x120) renderings and comparisons. Even when discarding 99.5% of the original vertex count and using only 597 faces (0.5%) the volume occupied by the model is still accurate while much easier to render. However further reducing the face count to 0.1% of the original model the collapsing algorithm starts to impact useful features of the model like the head and feet and devolves to something that barely even resembles a human. A good performance to quality ratio selection is the 1194 face (1%) model. Further tweaking the model in the 0.1-0.5% face range could yield highest performance benefits but at serious expense of tracking accuracy.

Performance table for a medium-quality (1194 faces/3582 triangles) model

Columns 4-7 depict time measurments in μs, Columns 8-9 depict framerates in fps

Tile *width*	*Tile* *height*	*PSO* *particles*	*Render* *gpu/sync*	*Render* *gpu/async*	*Thrust* *gpu*	*Neural* *cpu*	*Framerate* *serial−neural*	*Framerate* *parallel−neural*
64	64	64	2585μs	475μs	874μs	1009μs	10.12 fps	12.71 fps
64	64	81	3272μs	796μs	872μs	1289μs	8.03 fps	10.12 fps
64	64	100	4003μs	948μs	928μs	1580μs	6.70 fps	8.50 fps
100	100	64	2567μs	1016μs	1993μs	1021μs	7.58 fps	8.97 fps
100	100	81	3238μs	1245μs	2407μs	1277μs	6.12 fps	7.26 fps
100	100	100	4006μs	1513μs	2958μs	1578μs	4.97 fps	5.90 fps
120	120	64	2563μs	1271μs	2713μs	1028μs	6.60 fps	7.64 fps
120	120	81	3246μs	1571μs	3306μs	1262μs	5.33 fps	6.16 fps
120	120	100	4008μs	1932μs	3974μs	1583μs	4.35 fps	5.04 fps
160	120	64	2594μs	1553μs	3414μs	1020μs	5.83 fps	6.61 fps
160	120	81	3280μs	1936μs	4267μs	1264μs	4.65 fps	5.27 fps
160	120	100	4026μs	2370μs	5130μs	1584μs	3.81 fps	4.34 fps

Table 5.2: Average performance measurements of the various code segments of the optimization framework when using a medium quality polygon model. The first three columns refer to the optimizer configuration. Bigger tile sizes and particle numbers ensure better quality at the expense of computing time. Render columns in the table refer to the rendering times which are split to the setup and uploading of the articulated model matrices and the actual rendering. The thrust column refers to the time consumed by CUDA/Thrust to perform scoring. Finally the Neural column refers to the time it takes to score the neural-network part of the objective function (Eq:5.4). It can be concurrently calculated on the CPU while the GPU handles the rendering or happen serially after rendering is done. These two different pipelining options are shown on the last 2 columns. We observe that greater particle numbers mean more sphere comparisons while GPU loads also influence thread scheduling. Neural network computation times also appear to be slightly impacted on larger tile sizes despite not directly performing more computations. Each of the PSO generations takes the sum of the times presented in the last four columns. For each received observation frame we repeat the optimization step depending on the number of PSO generations desired. So rendering tiles sized 64x64 on a 64 particle 20 generation (1280 renderings) configuration we will consume 20 * (2585 + 475 + 874 + 1009) = 98860 microseconds achieving performance of roughly 10 fps. Running the neural network computations on a separate thread can further boost the framerate to 12 fps.

Performance table using the raw scanned model (119486 faces/358458 triangles)

Tile *width*	*Tile* *height*	*PSO* *particles*	*Render* *gpu/sync*	*Render* *gpu/async*	*Thrust* *gpu*	*Neural* *cpu*	*Framerate* *serial−neural*	*Framerate* *parallel−neural*
64	64	64	2652μs	16829μs	863μs	1064μs	2.34 fps	2.46 fps
64	64	81	3345μs	21541μs	870μs	1320μs	1.85 fps	1.94 fps
64	64	100	4099μs	26631μs	997μs	1666μs	1.50 fps	1.58 fps
100	100	64	2647μs	17425μs	1989μs	1053μs	2.16 fps	2.27 fps
100	100	81	3360μs	22108μs	2476μs	1379μs	1.71 fps	1.79 fps
100	100	100	4079μs	27157μs	2957μs	1619μs	1.40 fps	1.46 fps
120	120	64	2636μs	17725μs	2712μs	1076μs	2.07 fps	2.17 fps
120	120	81	3343μs	22425μs	3304μs	1348μs	1.64 fps	1.72 fps
120	120	100	4108μs	27669μs	3999μs	1671μs	1.34 fps	1.40 fps
160	120	64	2648μs	18061μs	3410μs	1086μs	1.98 fps	2.07 fps
160	120	81	3352μs	22820μs	4261μs	1347μs	1.57 fps	1.64 fps
160	120	100	4071μs	28135μs	5136μs	1604μs	1.28 fps	1.34 fps

Table 5.3: Average performance measurements of the various code segments of the optimization framework when using the raw high polygon model acquired by scanning. As stated in Figure 5.2, the Neural column can be concurrently calculated on the CPU while the GPU handles the other three columns. Each of the PSO generations takes the sum of the times presented in the last four columns. The neural time can be subtracted if we run it on a separate thread instead of a serial execution. For each frame we repeat these steps depending on the number of PSO generations selected so for a 64x64 sized 64 particle 20 generation configuration we will achieve 20 * (2652 + 16829 + 863 + 1064) = 428160 microseconds or roughly 2.3 fps. Running the neural network computations on a separate thread does not produce a meaningful speed-up as in the case of the optimized medium-polygon model since the bottleneck in this case is the time to render the geometry. Of course rendering the raw model on a 64x64 tile is wasteful especially since we know that our camera sensor cannot output observations of good enough quality. The way we achieve this quality during model capture with the same sensor is with extended exposure times for each frame to reduce noise as elaborated in Section 2. An off-line 3D tracker application interested in a very high level of detail should also use a larger tile size to harness the high quality renderings as well as a bigger number of PSO generations to ensure enough optimization budget to converge to a good solution.

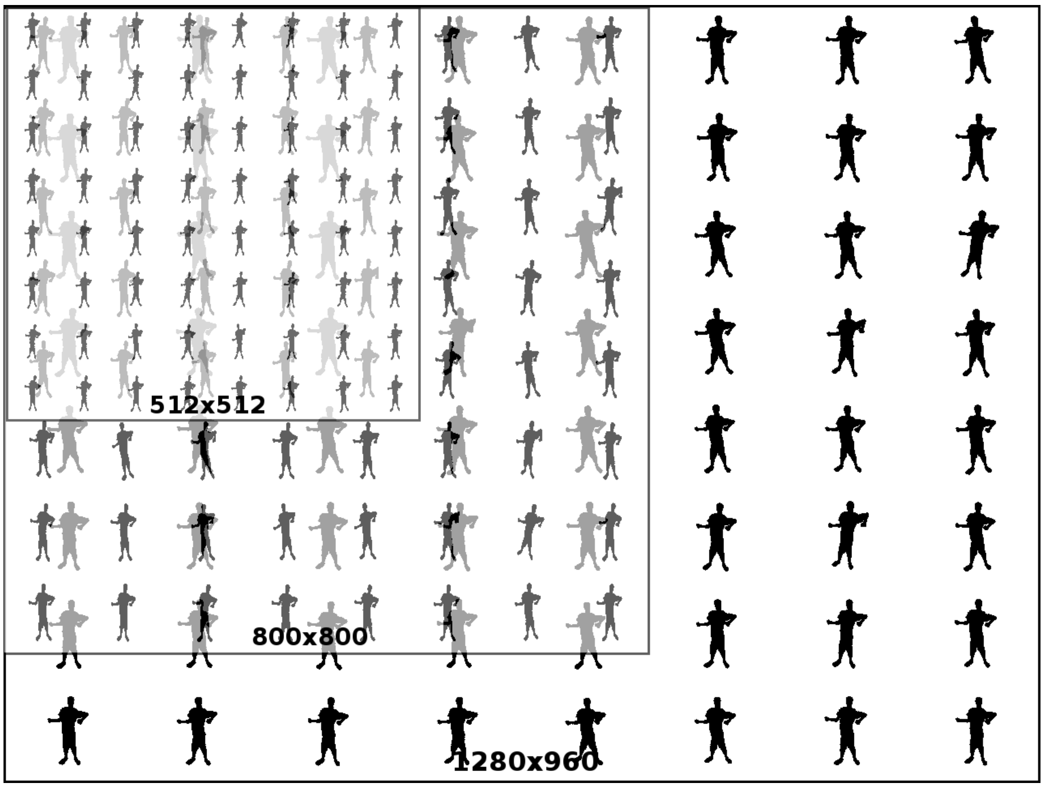


Figure 5.3: Different rendering sizes of a 64 (8x8) PSO particle configuration , displayed overlayed for comparison. Input depth observations come encoded in frames of 640x480 pixels, a resolution which is internally upscaled from the 320x240 depth sensor of the ASUS Xtion/Kinect. However performing comparisons in the original resolution proves very demanding for contemporary GPUs and thus an important technique that enable interactive framerates is to subsample and work with more manageable image dimensions. The largest 1280x960 tile-batch consists of tiles sized 160x120 pixels each that share the same aspect ratio with the input depth frame. However we can further reduce sizes by using "zoomed" projection matrices that clip the rendering viewport and that can also alter the aspect ratio by squeezing it. A 100x100 "zoomed" projection matrix has more resolution than the 160x120 scaled one since almost two thirds of the 160x120 image are empty. This way and combined to a lower polygon mesh we can drastically improve rendering times as well as perform fewer CUDA reduction operations and gain significant speed-ups reaching 9fps framerates when including the neural network times.

Chapter 6

Experimental Evaluation

The proposed framework consists of high quality modules and this fact immediately gives the resulting method a positive outlook. However in order to quantify its precision and overall accuracy we needed to perform a large number of experiments to prove that the method is robust and will work in practice as well as its architecture suggests.

We thus needed to perform experiments and meticulously log every execution detail to take care of the following issues :

1. Investigate the importance of scanned model quality for accurate tracking. This means comparing skinned models with parametric models.
2. Investigate the personalization aspect of models. That means having each subject be tracked with multiple different skinned models to test how model variability affects tracking quality.
3. Investigate the robustness of the tracker when we have different movements, occlusions and movement speeds.
4. Compare tracking quality when using different optimization budgets

In the next sections of this chapter we will start addressing the issues mentioned above. We will begin by discussing the source of ground truth to accurately compare our method results to ideal results, we will continue by explaining the rationale behind the models selected for the experiments, the experimentation system and optimization parameters and the final part of the chapter will conclude with quantitative and qualitative results.

6.1 Ground truth and the need for a synthetic dataset

The quantitative evaluation of our method requires a dataset containing RGBD frames of moving humans, together with ground truth regarding their 3D motion as well as their scanned and skinned 3D model. To the best of our knowledge and at the time of writing this thesis, datasets that meet all of the mentioned criteria were not publicly available.

That beeing said there were many public RGBD datasets that partly fulfilled our requirements. Most of them featured humans performing actions with ground truth from a Motion Capture system but all of them lacked the 3D model information of the actors. For example, the Berkeley Multi modal Human Action Database (MHAD) [116] contains RGBD data and ground truth motion information, but no detailed skinned models of the actors. The CMU datasets [115] contain more challenging motions than MHAD, but contains no actual RGB information. Finally, the TNT15 dataset [119] lacks depth information. Moreover, although the lack of RGB data makes it already problematic as a candidate dataset the 3D models of subjects are supplied but they are laser-scanned and therefore much more accurate and noise-free compared to the ones we employ, adding another obstacle to their use for a fair comparison. Geometric approximation models are the norm in the literature and they just require one initialization frame to match the height and thickness of the human body to be tracked. One or more frames of the subject doing a neutral T-pose were provided for initialization by most public datasets but these were not enough to achieve the "super resolution" of the model required to bootstrap the model acquisition methods used by this work [102], [21]. The model we use [102] is acquired by 8 key poses every one of which requires sampling 200 frames in order for the Kinect Fusion [38] algorithm to refine input and discard artifacts.

Having no dataset that could readily be used to facilitate our experimentation we had to resort to constructing our own. We recruited 15 different subjects shown in Figure 6.1 of all genders, heights and body types in order to acquire a diverse set of high quality body models using [102]. Having a big pool of subjects to perform experiments was our first step but there where two further options to choose on how to proceede.

The first option was to record datasets using a full body motion capture system that would provide accurate measurements for each joint along with the captured RGB+D data. The recruited persons would all need to wear a MOCAP suit and then try to reproduce the same scripted motions in front of a multicamera calibrated system in an effort to produce a new dataset that could be tracked and objectively measured. The drawbacks of this dataset generation method would be that the subjects would not perform exactly the same motions so there would be deviations in the "difficulty" of the dataset from person to person. A second drawback would be that although the markers would track very accurately it would be impossible for them to be placed in the same positions as the joints of the 3D scanned models. Although this could be remedied by triangulating the joint positions using multiple markers on the same limbs, this fact could still introduce a slight bias when measuring results. Finally the biggest obstacle was the cost and general unavailability of a MOCAP system which was the reason of dismissing this method of dataset generation.

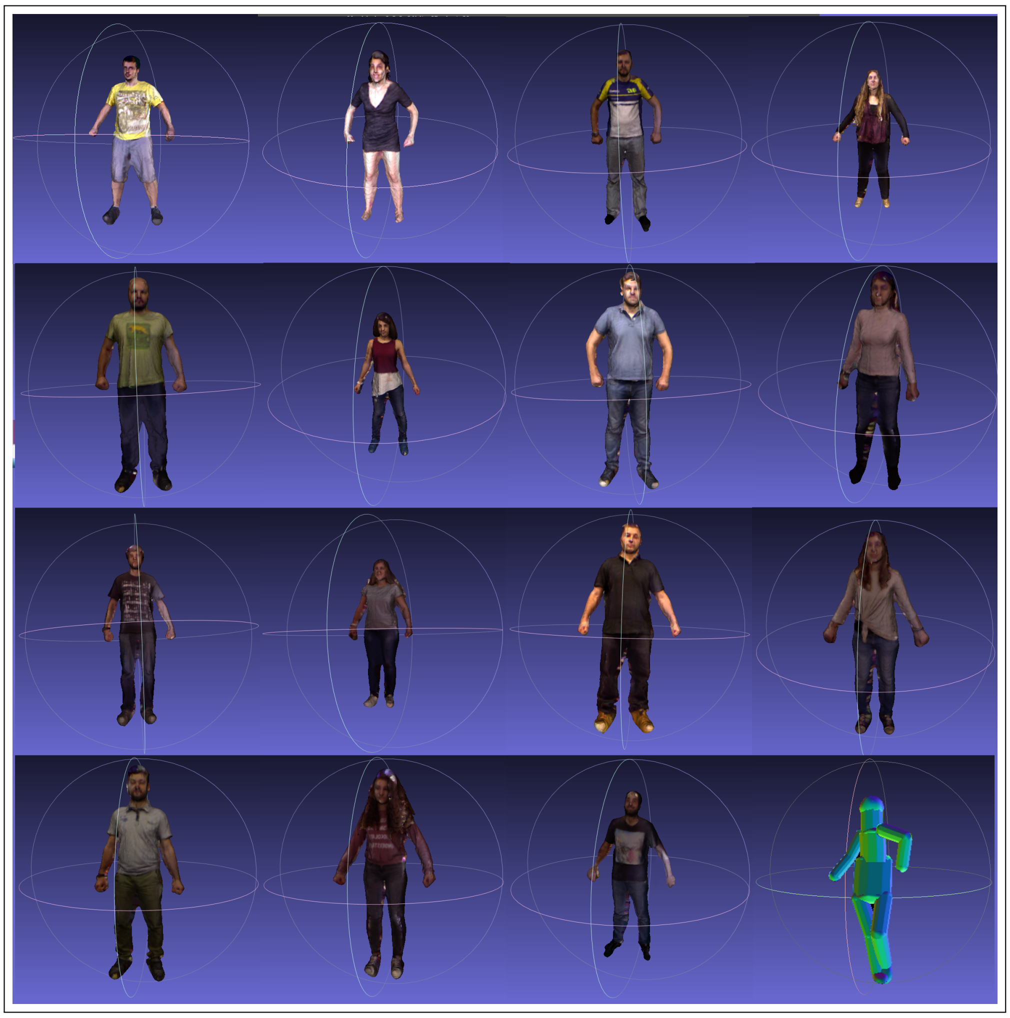


Figure 6.1: Snapshots of the 15 models captured using [102] posing in the default t-pose. The parametric model is also included for comparison in the bottom right. The 4 subjects selected for the experiments are seen in the first row. The selection was based on maximizing their BMI difference to make experiments more representative of the importance of body type. Their BMI measurements are the following from left to right. Male 1 (M_1) is 1.78cm tall and 85kg (26.8BMI), Female 1 (F_1) is 1.58m and 47kg (18.8BMI), Male 2 (M_2) is 1.90cm and 80kg (22.2BMI) and Female 2 (F_2) 1.71m, 67kg (22.9BMI).

The second option was to generate a synthetic dataset using the scanned subjects model data and a 3D renderer. This way we could programatically render each one at every desireable pose while also knowing exactly what the tracked pose should be. This way a variety of motions that where acquired from the motion capture data of the CMU dataset [115] and the rendered datasets would be represented and have 1:1 correspondance between all subjects as well as allow unbiased scoring of the tracker since we would know the true position of the joints for each of the rendered frames. The only downside would be that any unmodeled behavior of the body would not be reflected by the dataset, but given our options this proved to be the only way to have high quality ground truth.

Thus, we constructed our own dataset using the synthetic dataset option which proved to be a very practical method and accomodated the experiments very well.

Having 15 scanned models plus one for a parametric model and 15 ground truth datasets means that we would have to perform tracking for every combination of dataset/model for each of the tested configurations bringing them to a total of 240 experiments to cover all combinations per experimental configuration.

From the 15 persons scanned using [102] we thus selected the 4 persons with the most diverse body mass indicators (BMI), 2 from each gender. This way by cross-examining 4 very diverse subjects plus the customizable parametric "tin-man" model $\mathbf{P}$, the computational load was much more manageable (20 experiments per configuration) and the results more indicative since we had much less pronounced similiarities between scanned models.

Had we selected all subjects, then running each experimental configuration would take weeks instead of hours something that would make the study impossible to be completed in a reasonable timeframe. Moreover since many models share very similar proportions, heights and weights their cross-examination would reveal less pronounced errors something that would make the metrics extracted confusing and not allow clear conclusions on the importance of body models.

The male subjects are noted using ($\mathbf{M}$, M_1 for the low BMI and M_2 for the high male BMI) and the two females using ($\mathbf{F}$, F_1 for low BMI and F_2 for high female BMI). We also employed a primitives-based model ($\mathbf{P}$) whose dimensions can be adapted to best approximate a given human model. These models are illustrated in Fig. 6.12 (top). The collected motion capture data from the CMU datasets [115], included a variety of motions like bending, jumping jacks, simultaneous twisting of torso and limbs, etc. Finally, we rendered the $\mathbf{M}$ and $\mathbf{F}$ models in the laboratory environment of the MHAD [116] datasets. As stated before, the reason why we did not employ MHAD per se was the unavailability of skinned human models. Moreover, MHAD contains repetitive motions of lower complexity compared to that of the employed CMD dataset. As stated in section 2.4.1, dual quaternion blending [39] was used to realize the skin deformations of the $\mathbf{M}$ and $\mathbf{F}$ models to avoid the "candy-wrapping" artifacts produced by standard linear blending. Thus, we obtained RGBD frames of known human models performing known, complex

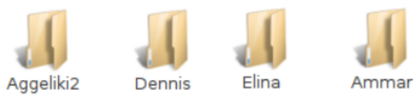


Figure 6.2: The datasets provided in the public archive are the four used in the comparitive analysis of this work. $M1$ is Ammar, $M2$ is Dennis, $F1$ is Elina and $F2$ is Aggeliki2

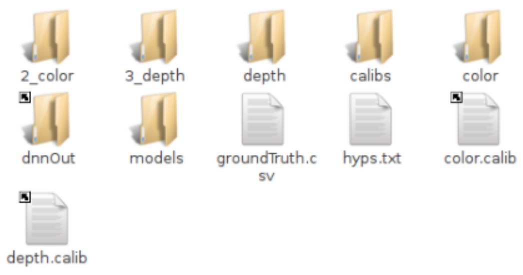


Figure 6.3: Each of the dataset subfolders hold all of the relevant information to allow someone to use the stored data to perform his own experiments.

motions in a realistic environment. The final result is two RGBD sequences[1] of 720 frames each, of the same motions, performed by a male and a female subject. We refer to these sequences as MS and FS, respectively.

6.2 Ground truth public dataset

The generated dataset is provided to the public domain in a compressed archive [83]. After downloading the archive which is 844MB long and extracting it each of the extracted sub-folders will contain the data of a separate subject. The correspondence of the filenames of the subjects to the models of this thesis are seen in Figure 6.2.

Inside each of the subjects subfolders we will find their highly detailed skinned models as well as the ground truth image data and text files that hold the correct configurations of the persons for each of the observed frames.

6.2.1 Color and depth data

Folders color and 2_color hold JPEG image files that are colored renderings of the skeleton in each of the poses in a frontal and back view. The frontal view also contains a simulated background taken from the MHAD dataset in order to

[1]These sequences, together with more that are under development, are publicly available at http://users.ics.forth.gr/~argyros/research.html#datasets

dnnOut 000000.jpg 000001.jpg 000002.jpg 000003.jpg 000004.jpg
000005.jpg 000006.jpg 000007.jpg 000008.jpg 000009.jpg 000010.jpg
000011.jpg 000012.jpg 000013.jpg 000014.jpg 000015.jpg 000016.jpg
000017.jpg 000018.jpg 000019.jpg 000020.jpg 000021.jpg 000022.jpg
000023.jpg 000024.jpg 000025.jpg 000026.jpg 000027.jpg 000028.jpg
000029.jpg 000030.jpg 000031.jpg 000032.jpg 000033.jpg 000034.jpg
000035.jpg 000036.jpg 000037.jpg 000038.jpg 000039.jpg 000040.jpg
000000.jpg 000001.jpg 000002.jpg 000003.jpg 000004.jpg 000005.jpg
000006.jpg 000007.jpg 000008.jpg 000009.jpg 000010.jpg 000011.jpg
000012.jpg 000013.jpg 000014.jpg 000015.jpg 000016.jpg 000017.jpg
000018.jpg 000019.jpg 000020.jpg 000021.jpg 000022.jpg 000023.jpg
000024.jpg 000025.jpg 000026.jpg 000027.jpg 000028.jpg 000029.jpg
000030.jpg 000031.jpg 000032.jpg 000033.jpg 000034.jpg 000035.jpg
000036.jpg 000037.jpg 000038.jpg 000039.jpg 000040.jpg 000041.jpg

Figure 6.4: The datasets folders color (left) and 2_color (right) that hold the two views of our simulated color renderings of our ground truth.

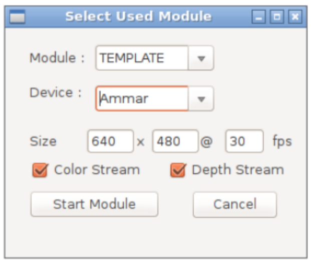
make the dataset more realistic and challenging. The back view only contains foreground.

Folders depth and 3_depth hold 16-bit PNG depth image files. They will typically appear as completely black frames in file managers and most image editors that expect 8-bit RGB encodings. In order to read them correctly you can use OpenCV and its imread command (i.e. imread(filename,IMREAD_ANYDEPTH)) . If you want to directly use the libpng library the following code snippet[2] might be useful which is a part of a larger codec toolkit of an RGBDAcquisition tool [84] that can directly handle accessing these datasets.

When a pair of images from the same view are opened correctly (color/xxxxxx.jpg corresponds to depth/xxxxxx.png) and (2_color/xxxxxx.jpg to 3_depth/xxxxxx.png) you should be able to get a snapshot of the experiment. In order to visualize the frame pairs the code snippets in the RGBDAcquisition repository[3] [84]. might be useful although they will require modifications to be decoupled from the RGBDAcquisition library. Each of the pixels of the 1 channel 16-bit depth frame is an unsigned short integer which contains a sensor reading in millimeters for the specific pixel. Each of the pixels of the 3 channel 8-bit color frame is a Red Green and Blue value. Please note that in order to properly visualize the depth frame it needs to be converted to an 8-bit RGB, this can be done using the OpenCV convertTo calls, and because OpenCV works in a BGR color space instead of RGB a convertTo call needs to be done for the RGB frame as well.

000000.png 000001.png 000002.png 000003.png 000004.png 000005.png
000006.png 000007.png 000008.png 000009.png 000010.png 000011.png
000012.png 000013.png 000014.png 000015.png 000016.png 000017.png
000018.png 000019.png 000020.png 000021.png 000022.png 000023.png
000024.png 000025.png 000026.png 000027.png 000028.png 000029.png
000030.png 000031.png 000032.png 000033.png 000034.png 000035.png
000000.png 000001.png 000002.png 000003.png 000004.png 000005.png
000006.png 000007.png 000008.png 000009.png 000010.png 000011.png
000012.png 000013.png 000014.png 000015.png 000016.png 000017.png
000018.png 000019.png 000020.png 000021.png 000022.png 000023.png
000024.png 000025.png 000026.png 000027.png 000028.png 000029.png
000030.png 000031.png 000032.png 000033.png 000034.png 000035.png

Figure 6.5: The datasets folders depth (left) and 3_depth (right) that hold the two views of our simulated 16-bit depth renderings of our ground truth. File managers typically expect 8-bit png images so the 16-bit depths may appear black until processed by a proper PNG reader.


Select Used Module

Module : TEMPLATE

Device : Ammar

Size 640 x 480 @ 30 fps

✓ Color Stream ✓ Depth Stream

Figure 6.6: Opening a dataset using the RGBDAcquisition Editor [84]

6.2.2 3D rigged model data

The 3D Models are included in 3 different file formats and are all inside the models subdirectory of each of the subject folders.

1. For each of the models there is a .dae file that is a Collada rigged mesh that can be imported using various tools such as Blender, Meshlab, SmartBody suite or other compatible 3D editor tools. The .dae files are provided as they were output by the Autorigger and Reshaper tool [21] and require a resize operation in the armature. You can consult the blender.ogv video file provided here[4] that shows how to open a .dae model file using Blender, and perform this operation as well as to perform a decimate operation to their geometry. You can also use blender to convert the model to other file formats

[2]`https://github.com/AmmarkoV/RGBDAcquisition/blob/master/tools/Codecs/pngInput.c`
[3]`https://github.com/AmmarkoV/RGBDAcquisition/blob/master/viewer/main.cpp#L232`
[4]`http://cvrlcode.ics.forth.gr/web_share/wacv18/blender.ogv`

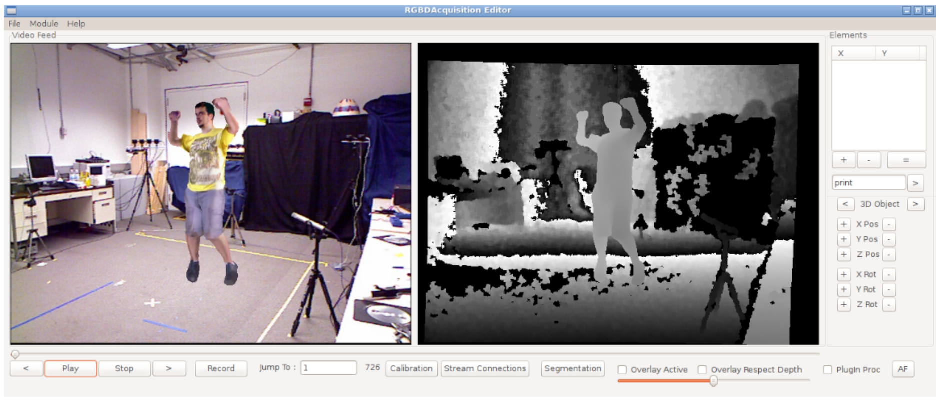


Figure 6.7: Viewing the $M1$ dataset using the RGBDAcquisition Editor [84]

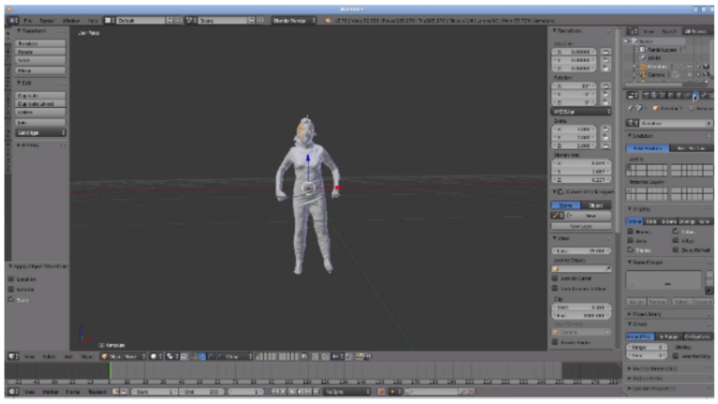


Figure 6.8: Viewing the $F1$ model using the Blender 3D Editor in its original .dae OpenCollada container as outputed by [21].

so the .dae file can be a very useful tool.

2. A second file format provided is the .tri file format which is a simple C/C++ oriented 3D mesh container. You can use this code snippet[5] and the loadModelTri call that can load a filename to a struct TRI_Model[6] which can hold all of the geometry in an accessible way from a native C/C++ program. In order to animate the skeleton through your program you can use

[5]https://github.com/AmmarkoV/RGBDAcquisition/blob/master/opengl_acquisition_shared_library/opengl_depth_and_color_renderer/src/ModelLoader/model_loader_tri.c

[6]https://github.com/AmmarkoV/RGBDAcquisition/blob/master/opengl_acquisition_shared_library/opengl_depth_and_color_renderer/src/ModelLoader/model_loader_tri.h

the transformation code provided here[7] that performs the vertex transformation operations in CPU although a better implementation should use a GPU shader based transformation renderer. The OpenGL renderer provided here[8] can be easily built using CMake and render using high level .scene scripts such as this script[9]. Further details can be found in the project‘s github page since they are beyond the scope of this thesis. In order to perform a conversion from the .dae file to the .tri format yourself you can use the tool provided here[10] that can perform the conversion using ./assimpTester –convert source.dae target.tri

3. A third file format provided is .xml / .bm file pairs that can be opened using the MBV Tracker suite [68]. They contain both the high quality rigged model described in the paper as well as a parametric version of it with the prefix tinBodyXXX.xml This file format is proprietary and property of ICS-FORTH and governed by this license[11].

6.2.3 DNN 2D joint tracking output

Instead of demanding the dataset user to manually compile the neural network 2D joint source we have recorded its output and stored it to the dnnOut subdirectory that contains a 2D estimation for bodies detected for each of the color frames provided. The body detector used [13] is implemented in Caffe and generates .json file output using the COCO format. For more information you can see the declarations in this file [12]. Parsing the files can be trivially done using many different tools but it can also be done using this provided code [13] that with the parseJsonCOCOSkeleton(const char * filename, struct skeletonCOCO * skel) call parses the file to a struct SkeletonCOCO that is directly accessible through a C/C++ program but also has many service functions some of which can also convert it to CSV (printCOCOSkeletonCSV), generate .scene files that can be viewed with the RGBDAcquisition renderer(visualize3DSkeletonHuman) or even generate SVG files (visualize2DSkeletonHuman).

[7]`https://github.com/AmmarkoV/RGBDAcquisition/blob/master/opengl_acquisition_shared_library/opengl_depth_and_color_renderer/src/ModelLoader/model_loader_transform_joints.c`

[8]`https://github.com/AmmarkoV/RGBDAcquisition/tree/master/opengl_acquisition_shared_library/opengl_depth_and_color_renderer`

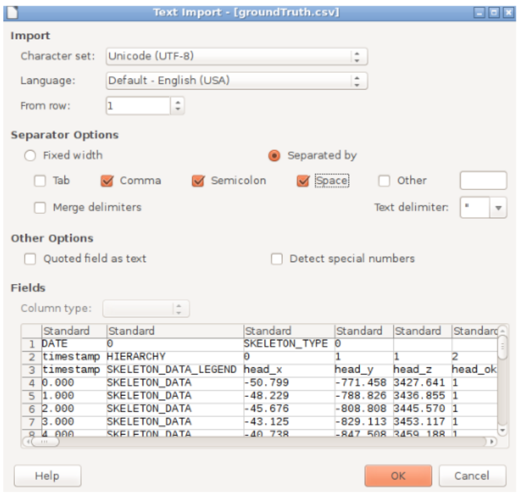
[9]`https://github.com/AmmarkoV/RGBDAcquisition/blob/master/opengl_acquisition_shared_library/opengl_depth_and_color_renderer/Scenes/mbvHumanNew.conf`

[10]`https://github.com/AmmarkoV/RGBDAcquisition/tree/master/opengl_acquisition_shared_library/opengl_depth_and_color_renderer/submodules/Assimp`

[11]`https://github.com/FORTH-ModelBasedTracker/HandTracker/blob/master/license.txt`

[12]`https://github.com/CMU-Perceptual-Computing-Lab/caffe_rtpose/blob/master/src/rtpose/modelDescriptorFactory.cpp`

[13]`https://github.com/AmmarkoV/RGBDAcquisition/blob/master/tools/Primitives/jsonCocoSkeleton.c`


Text Import - [groundTruth.csv]

Import

Character set: Unicode (UTF-8)

Language: Default - English (USA)

From row: 1

Separator Options

( ) Fixed width [✓ Separated by]

☐ Tab [✓ Comma] [✓ Semicolon] [✓ Space] ☐ Other

☐ Merge delimiters Text delimiter: "

Other Options

☐ Quoted field as text ☐ Detect special numbers

Fields

Column type:

	Standard	Standard	Standard	Standard	Standard	Standard
1	DATE	0	SKELETON_TYPE	0		
2	timestamp	HIERARCHY	0	1	1	2
3	timestamp	SKELETON_DATA_LEGEND	head_x	head_y	head_z	head_ok
4	0.000	SKELETON_DATA	-50.799	-771.458	3427.641	1
5	1.000	SKELETON_DATA	-48.229	-788.826	3436.855	1
6	2.000	SKELETON_DATA	-45.676	-808.808	3445.570	1
7	3.000	SKELETON_DATA	-43.125	-829.113	3453.117	1
8	4.000	SKELETON_DATA	-40.738	-847.508	3459.188	1

Figure 6.9: Opening a groundtruth .csv file using LibreOffice Calc, notice that Semicolon is checked as a seperator option

6.2.4 Ground truth data

Finally the most important aspect of the dataset the ground truth data are provided in a .csv file format in the groundTruth.csv files that are very easy to parse. They can be directly opened by a very big selection of applications including office tools Microsoft Excel or LibreOffice Calc are semicolon separated and have labels for each column that make it easy to understand the assigned values. Timestamps correspond to frame numbers in color and depth subdirectories.

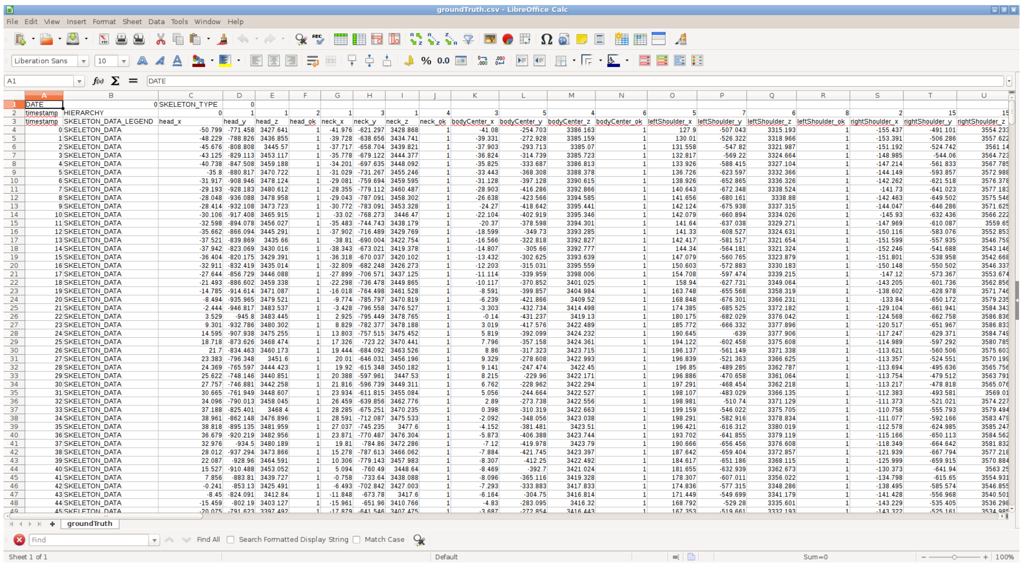
In order to parse the groundtruth .csv files using a program you can use this simple Input Parser for C [14] that can be fed strings and tokenize them using the semicolon as a delimiter. An example client that will parse the csv files can be found here [15].

6.2.5 Camera intrinsics

The camera intrinsics configurations are stored in the color.calib and depth.calib (which are identical since the RGB and depth streams are registered). There are no intrinsic distortions since the groundtruth is rendered, and front views have no Translation or Rotation vectors but rather have an identity transform matrix. The information for the back views is stored in calibs/02.calib and calibs/03.calib. The calibration files are very easy to parse since they try to follow a Matlab

[14] `https://github.com/AmmarkoV/InputParser/blob/master/InputParser_C.c`

[15] `https://github.com/AmmarkoV/RGBDAcquisition/blob/master/tools/Primitives/CompareBody/groundTruthParser.cpp`


groundTruth.csv - LibreOffice Calc
File Edit View Insert Format Sheet Data Tools Window Help
Liberation Sans
10
A1
DATE
groundTruth
Find
Find All
Search Formatted Display String
Match Case
Sheet 1 of 1
Default
Sum=0
100%

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	DATE	0	SKELETON_TYPE	0																	
2	timestamp	HIERARCHY	0	1	1	2	1	3	1	4	3	5	4	6	5	7	6	8	2	15	15
3	timestamp	SKELETON_DATA_LEGEND	head_x	head_y	head_z	head_ok	neck_x	neck_y	neck_z	neck_ok	bodyCenter_x	bodyCenter_y	bodyCenter_z	bodyCenter_ok	leftShoulder_x	leftShoulder_y	leftShoulder_z	leftShoulder_ok	rightShoulder_x	rightShoulder_y	rightShoulder_z
4	0	SKELETON_DATA	-50.799	-771.458	3427.641	1	-41.976	-621.297	3428.868	1	-41.08	-254.703	3386.163	1	127.9	-507.043	3315.193	1	-155.437	-491.101	3554.233
5	1	SKELETON_DATA	-48.229	-788.826	3436.855	1	-39.728	-638.656	3434.741	1	-39.331	-272.928	3385.159	1	130.01	-526.322	3318.966	1	-153.391	-506.286	3557.622
6	2	SKELETON_DATA	-45.676	-808.808	3445.57	1	-37.717	-658.704	3439.821	1	-37.903	-293.713	3385.07	1	131.558	-547.82	3321.987	1	-151.192	-524.742	3561.14
7	3	SKELETON_DATA	-43.125	-829.113	3453.117	1	-35.778	-679.122	3444.377	1	-36.824	-314.739	3385.723	1	132.817	-569.22	3324.664	1	-148.985	-544.06	3564.723
8	4	SKELETON_DATA	-40.738	-847.508	3459.188	1	-34.201	-697.635	3448.092	1	-35.825	-333.687	3386.813	1	133.926	-588.415	3327.104	1	-147.214	-561.833	3567.785
9	5	SKELETON_DATA	-35.8	-880.817	3470.722	1	-31.029	-731.267	3455.246	1	-33.443	-368.308	3388.378	1	136.726	-623.597	3332.366	1	-144.149	-593.857	3572.988
10	6	SKELETON_DATA	-31.917	-908.946	3478.124	1	-29.081	-759.694	3459.595	1	-31.128	-397.128	3390.615	1	138.926	-652.865	3336.326	1	-142.262	-621.518	3576.378
11	7	SKELETON_DATA	-29.193	-928.183	3480.612	1	-28.355	-779.112	3460.487	1	-28.903	-416.286	3392.866	1	140.643	-672.348	3338.524	1	-141.73	-641.023	3577.183
12	8	SKELETON_DATA	-28.048	-936.088	3478.958	1	-29.043	-787.091	3458.302	1	-26.638	-423.566	3394.585	1	141.656	-680.161	3338.88	1	-142.463	-649.502	3575.546
13	9	SKELETON_DATA	-28.414	-932.108	3473.723	1	-30.772	-783.091	3453.328	1	-24.27	-418.642	3395.441	1	142.124	-675.938	3337.315	1	-144.047	-646.286	3571.625
14	10	SKELETON_DATA	-30.106	-917.408	3465.915	1	-33.02	-768.273	3446.47	1	-22.104	-402.919	3395.346	1	142.079	-660.894	3334.026	1	-145.93	-632.436	3566.222
15	11	SKELETON_DATA	-32.598	-894.078	3456.027	1	-35.483	-744.743	3438.179	1	-20.37	-378.598	3394.301	1	141.64	-637.038	3329.271	1	-147.969	-610.087	3559.65
16	12	SKELETON_DATA	-35.662	-866.094	3445.291	1	-37.902	-716.489	3429.769	1	-18.599	-349.73	3393.285	1	141.33	-608.527	3324.631	1	-150.116	-583.076	3552.853
17	13	SKELETON_DATA	-37.521	-839.869	3435.66	1	-38.81	-690.004	3422.754	1	-16.566	-322.818	3392.827	1	142.417	-581.517	3321.654	1	-151.599	-557.935	3546.759
18	14	SKELETON_DATA	-37.942	-823.069	3430.016	1	-38.343	-673.021	3419.378	1	-14.807	-305.66	3392.777	1	144.34	-564.181	3321.324	1	-152.246	-541.688	3543.146
19	15	SKELETON_DATA	-36.404	-820.175	3429.391	1	-36.318	-670.037	3420.102	1	-13.432	-302.625	3393.639	1	147.079	-560.765	3323.879	1	-151.801	-538.958	3542.668
20	16	SKELETON_DATA	-32.911	-832.419	3435.014	1	-32.809	-682.248	3426.273	1	-12.203	-315.031	3395.559	1	150.603	-572.883	3330.183	1	-150.148	-550.502	3546.337
21	17	SKELETON_DATA	-27.644	-856.729	3446.088	1	-27.899	-706.571	3437.125	1	-11.114	-339.959	3398.006	1	154.708	-597.474	3339.215	1	-147.12	-573.367	3553.674
22	18	SKELETON_DATA	-21.493	-886.602	3459.338	1	-22.298	-736.478	3449.865	1	-10.117	-370.852	3401.025	1	158.94	-627.731	3349.064	1	-143.205	-601.736	3562.856
23	19	SKELETON_DATA	-14.785	-914.614	3471.087	1	-16.018	-764.498	3461.528	1	-8.591	-399.857	3404.984	1	163.748	-655.568	3358.319	1	-138.602	-628.978	3571.746
24	20	SKELETON_DATA	-8.494	-935.965	3479.521	1	-9.774	-785.797	3470.819	1	-6.239	-421.866	3409.52	1	168.848	-676.301	3366.231	1	-133.84	-650.172	3579.235
25	21	SKELETON_DATA	-2.444	-946.817	3483.537	1	-3.428	-796.558	3476.527	1	-3.303	-432.734	3414.498	1	174.385	-685.525	3372.182	1	-129.104	-661.941	3584.343
26	22	SKELETON_DATA	3.529	-945.8	3483.445	1	2.925	-795.449	3478.765	1	-0.14	-431.237	3419.13	1	180.175	-682.029	3376.042	1	-124.568	-662.758	3586.836
27	23	SKELETON_DATA	9.301	-932.786	3480.302	1	8.829	-782.377	3478.188	1	3.019	-417.576	3422.489	1	185.772	-666.332	3377.896	1	-120.517	-651.967	3586.833
28	24	SKELETON_DATA	14.595	-907.938	3475.255	1	13.803	-757.515	3475.452	1	5.819	-392.099	3424.232	1	190.645	-639	3377.906	1	-117.247	-629.371	3584.749
29	25	SKELETON_DATA	18.718	-873.626	3468.474	1	17.326	-723.22	3470.441	1	7.796	-357.158	3424.361	1	194.122	-602.458	3375.608	1	-114.989	-597.292	3580.785
30	26	SKELETON_DATA	21.7	-834.463	3460.173	1	19.444	-684.092	3463.526	1	8.86	-317.323	3423.715	1	196.137	-561.149	3371.338	1	-113.621	-560.506	3575.603
31	27	SKELETON_DATA	23.383	-796.348	3451.6	1	20.01	-646.031	3456.196	1	9.329	-278.608	3422.993	1	196.839	-521.363	3366.625	1	-113.357	-524.551	3570.199
32	28	SKELETON_DATA	24.369	-765.597	3444.423	1	19.92	-615.348	3450.182	1	9.141	-247.474	3422.45	1	196.85	-489.285	3362.787	1	-113.694	-495.636	3565.756
33	29	SKELETON_DATA	25.622	-748.146	3440.851	1	20.388	-597.961	3447.53	1	8.215	-229.96	3422.171	1	196.886	-470.658	3361.064	1	-113.754	-479.512	3563.791
34	30	SKELETON_DATA	27.757	-746.881	3442.258	1	21.816	-596.739	3449.311	1	6.762	-228.962	3422.294	1	197.291	-468.454	3362.218	1	-113.217	-478.818	3565.076
35	31	SKELETON_DATA	30.665	-761.949	3448.607	1	23.934	-611.815	3455.084	1	5.056	-244.664	3422.527	1	198.107	-483.029	3366.135	1	-112.383	-493.581	3569.01
36	32	SKELETON_DATA	34.096	-790.013	3458.045	1	26.459	-639.856	3462.776	1	2.89	-273.738	3422.556	1	198.981	-510.74	3371.129	1	-111.373	-521.021	3574.227
37	33	SKELETON_DATA	37.188	-825.401	3468.4	1	28.285	-675.251	3470.235	1	0.398	-310.319	3422.663	1	199.159	-546.022	3375.705	1	-110.758	-555.793	3579.494
38	34	SKELETON_DATA	38.961	-862.148	3476.896	1	28.591	-712.087	3475.533	1	-2.092	-348.056	3423.038	1	198.291	-582.916	3378.834	1	-111.077	-592.166	3583.479
39	35	SKELETON_DATA	38.818	-895.135	3481.959	1	27.037	-745.235	3477.6	1	-4.152	-381.481	3423.51	1	196.421	-616.312	3380.019	1	-112.578	-624.985	3585.247
40	36	SKELETON_DATA	36.679	-920.219	3482.956	1	23.871	-770.487	3476.304	1	-5.873	-406.388	3423.744	1	193.702	-641.855	3379.119	1	-115.166	-650.113	3584.562
41	37	SKELETON_DATA	32.976	-934.5	3480.189	1	19.81	-784.86	3472.286	1	-7.12	-419.978	3423.79	1	190.666	-656.456	3376.608	1	-118.349	-664.642	3581.832
42	38	SKELETON_DATA	28.012	-937.294	3473.866	1	15.278	-787.613	3466.062	1	-7.884	-421.745	3423.397	1	187.642	-659.404	3372.857	1	-121.939	-667.794	3577.218
43	39	SKELETON_DATA	22.087	-928.96	3464.591	1	10.306	-779.143	3457.983	1	-8.307	-412.25	3422.492	1	184.617	-651.186	3368.115	1	-125.999	-659.915	3570.884
44	40	SKELETON_DATA	15.527	-910.488	3453.052	1	5.094	-760.49	3448.64	1	-8.469	-392.7	3421.024	1	181.655	-632.939	3362.673	1	-130.373	-641.94	3563.25
45	41	SKELETON_DATA	7.856	-883.81	3439.727	1	-0.758	-733.64	3438.088	1	-8.096	-365.116	3419.328	1	178.307	-607.011	3356.022	1	-134.798	-615.65	3554.931
46	42	SKELETON_DATA	-0.241	-853.13	3425.491	1	-6.493	-702.842	3427.003	1	-7.293	-333.883	3417.833	1	174.836	-577.315	3348.286	1	-138.495	-585.574	3546.855
47	43	SKELETON_DATA	-8.45	-824.091	3412.84	1	-11.848	-673.78	3417.6	1	-6.164	-304.75	3416.814	1	171.449	-549.699	3341.179	1	-141.428	-556.968	3540.501
48	44	SKELETON_DATA	-15.459	-802.19	3403.127	1	-15.961	-651.96	3410.766	1	-4.83	-283.095	3416.32	1	168.792	-529.28	3335.601	1	-143.229	-535.405	3536.298
49	45	SKELETON_DATA	-20.075	-791.623	3397.492	1	-17.879	-641.546	3407.475	1	-3.687	-272.854	3416.443	1	167.353	-519.661	3332.193	1	-143.322	-525.161	3534.985

Figure 6.10: Viewing the ground truth data in Libre Office Calc

compatible format. A code snippet that you can find here [16] provides a call RefreshCalibration(const char * filename,struct calibration * calib) which can open a file and populate a struct calibration with the intrinsic values.

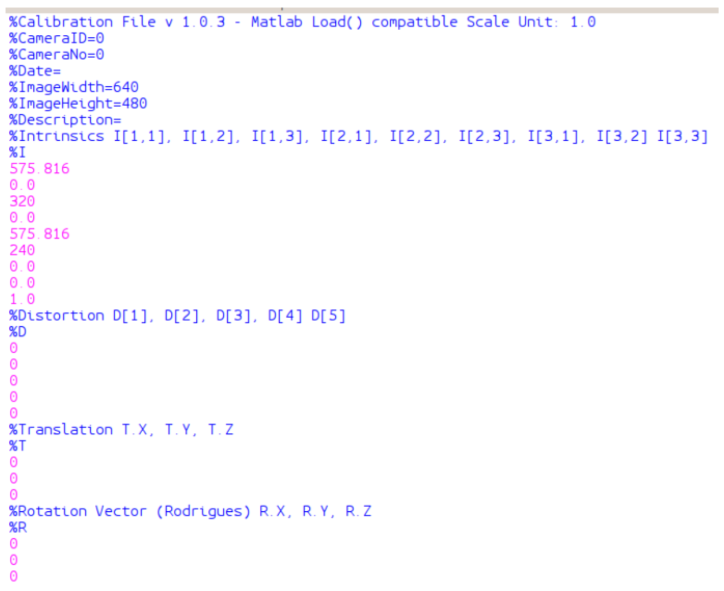
All of the files at the time of writing use a fx=575.816, fy=575.816, cx=320, cy=240 values to mirror our experimental camera, so one could even omit parsing the .calib files and hardcoding these values. However we plan to add more subjects to the datasets and it would be best to parse the .calib files in case of future additions that use different camera settings.

6.3 Optimization parameters

Having got all of the implementation details covered in Chapter 5 and having provided our ground truth details, before proceeding to the discussion of the experimental results we must first list and define the optimization parameters that needed investigation in order to study the behavior of the proposed optimization framework.

Since PSO (Section 4.1.3) is used as the optimizer that governs regression, its configuration parameters are also the most important for accurate and fast 3D tracking. These basic parameters are the number of **Generations**, **Particles** and the possiblity of generated **Pertrubations** as a mechanism to escape local minima during the optimization process. As mentioned in Chapter 4, multiplying Generations times Particles will give us the total number of renderings performed

[16] `https://github.com/AmmarkoV/RGBDAcquisition/blob/master/tools/Calibration/calibration.c#L145`



```
%Calibration File v 1.0.3 - Matlab Load() compatible Scale Unit: 1.0
%CameraID=0
%CameraNo=0
%Date=
%ImageWidth=640
%ImageHeight=480
%Description=
%Intrinsics I[1,1], I[1,2], I[1,3], I[2,1], I[2,2], I[2,3], I[3,1], I[3,2] I[3,3]
%I
575.816
0.0
320
0.0
575.816
240
0.0
0.0
1.0
%Distortion D[1], D[2], D[3], D[4] D[5]
%D
0
0
0
0
0
%Translation T.X, T.Y, T.Z
%T
0
0
0
%Rotation Vector (Rodrigues) R.X, R.Y, R.Z
%R
0
0
0
```

Figure 6.11: Contents of .calib file

to regress an input frame, this is the total computational budget afforded for the specific frame. Since PSO particles can be efficiently computed in parallel using a GPU implementation the Generations variable roughly controls the optimization loop time while the Particles number controls GPU utilization. The number of particles is very important for large parameter spaces, such as the one we work on, and we need a large population of PSO particles to effectively scan the multi-dimensional problem. A very big number of particles combined with a very small number of generations will not be able to perform adequately since the velocities and positions of particles will not have enough iterations to converge. A very big number of Generations is also not a good idea since it will have an adverse effect on achieved optimization framerates and after a specific point more computation time ceases to benefit output quality. Finally the Pertrubations parameter enables select parts of the solution vector to be perturbed in order to escape local minima. We chose not to perturb the first 7 parameters of the solution vector (i.e. the position and orientation of the human) since the big surface of the body ensures a smooth track across frames. Instead we chose to perturb only the limbs due to their small surface and much less pronounced contribution to the overall optimization score. The fact that limbs do not generally create a big gradient was also combatted by normalization of the depth volume as seen in Section 5.1.3.1 but pertrubations where also very useful in that regard. In our experiments we wanted to both strike a good balance between the number of generations and particles used but also study the role of pertrubations so all of these variables are a main part of our investigation.

A second category of imporant variables to tune where the Depth (W_D) and Joint (W_J) objective function weights that where described in Section 5.1.3. Having a comparatively big W_D value compared to W_J would make the body fit better to the observed depth but in case of tracking drift results would start deteriorating fast. An inverse configuration with a big W_J value and small W_D would make the tracker react well to sudden motions but the overall fit would never be good since the depth information would be largely ignored. Other than the experiments to strike a good balance between the W_D and W_J what was also very interesting as an experiment was to perform tracking with one of the two parameters beeing iteratively eliminated (beeing set to 0). This way we could roughly measure the role of the two parts of our objective function and their contribution to the final solution.

A third group of methods that where provisioned and could be switched on or off by command-line parameters, where heuristics that performed smoothing. The first and simplest smoothing option was to enable direct smoothing of the optimization output. This setting in fact slightly hurt average percision but it also eliminated jitter from the augmented visualization making results more appealing to the human eye. Although it could be a useful tool in cases where visualization is very important this idea was in the end not used at all and is included here for documentation purposes. Another imporant heuristic was the temporal continuity hypothesis. Having fast tracking rates a reasonable assumption is that subsequent frames will be close and share a similar solution vector. Thus remembering the last frame solution means it can be used as a hint that can effectively reduce the optimization search space. This is a very good idea and was used on at all experiments. Finally a last heuristic was to inject seed particles based on predictions. Enabling this meant that the tracker would keep a history of past states and monitor velocities and accelerations of limbs. When a new frame arrived and for the first generation a set of guesses would be generated to seed the first batch of particles. The implementation of this idea had a negligible impact on tracking percision since after the first generation PSO would replace these initial guesses and manage to converge with or without the guessed particles, and so it was not used at all during the experiments. However better statistical analysis of observed movements in conjunction with a different strategy of optimization potentially be used to improve tracking accuracy.

Another group of parameters that are important but are used as constants during the experimentations where the foreground estimation parameters. Having the previous rendering and the neural network detection we created a bounding box that covered both 2D areas plus a 1% margin around it. We then sampled incoming depths and thresholded all incoming depths using a bounding box of 1.2 meters of depth when compared to the previous frame solution. This ensured that background artifacts as well as the floor would be segmented out of the incoming depth image and while we would match depth volumes they would not cause any interference. As stated in Section 5.1.3.1 further thresholding happened using the $T = 30$cm final threshold during depth comparisons. All these numbers where

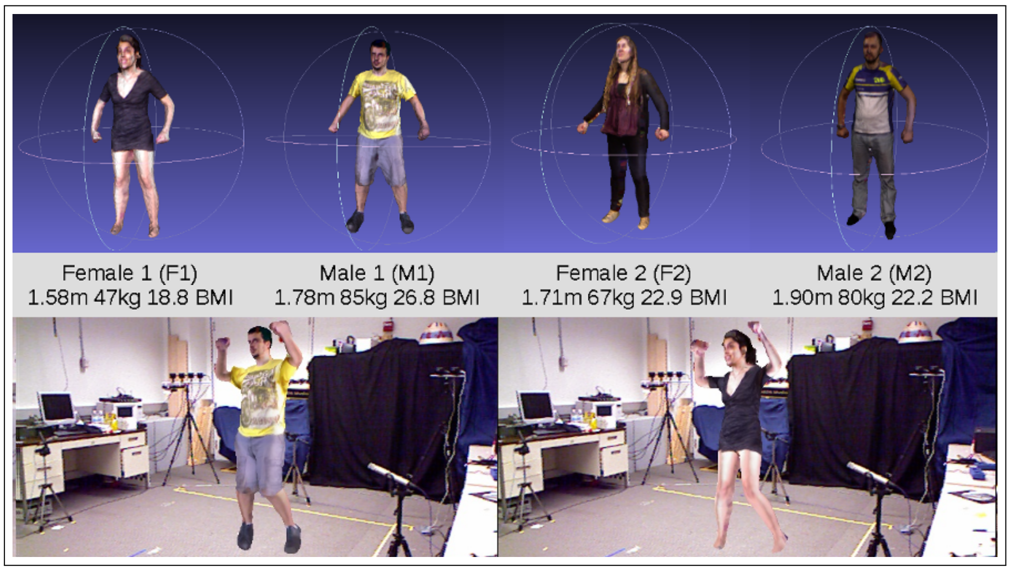


Figure 6.12: Snapshots of the four scanned models used in the experiments. Top row: The **M**ale and **F**emale models that were acquired automatically based on [102]. Bottom row: sample RGB frames that result from the rendering of the **M** and **F** models in the MHAD [116] dataset background, based on the CMU [115] mocap data.

experimentally found to work but they have to do with the foreground estimation. Assuming having only foreground input none of these techniques would need to be used.

The last group of optimization parameters are the joint weights for 2D estimations that have been listed in Table 5.1 and that define joint scoring using the E_J objective function joint term described at Section 5.1.3.2. They are also used as constants since they just serve to guide the tracker to pay more attention to the parts of the body that have less surface and are more important. These joint weights conclude the optimization parameters used and we are now ready to proceede to the experiments.

6.4 Evaluation metrics

The proposed method has been evaluated quantitatively on a synthetic data set (Section 6.1) annotated with ground truth. A first set of experiments investigated the effect of various parameters and design choices in the accuracy of the proposed method. In another set of experiments we compared the obtained performance to that of a baseline method [57]. The experimental evaluation is concluded with indicative qualitative results in RGBD sequences.

Preliminary qualitative tests of the proposed method with real RGBD camera

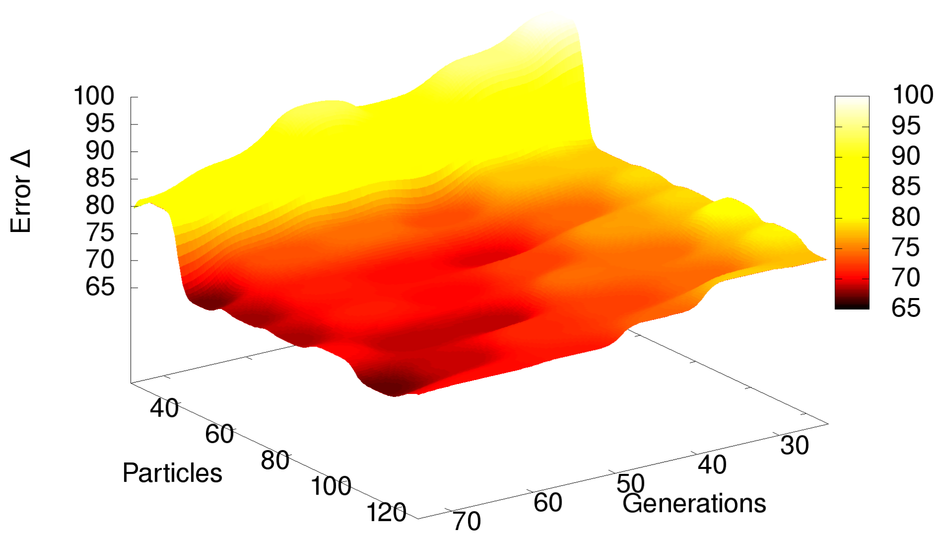
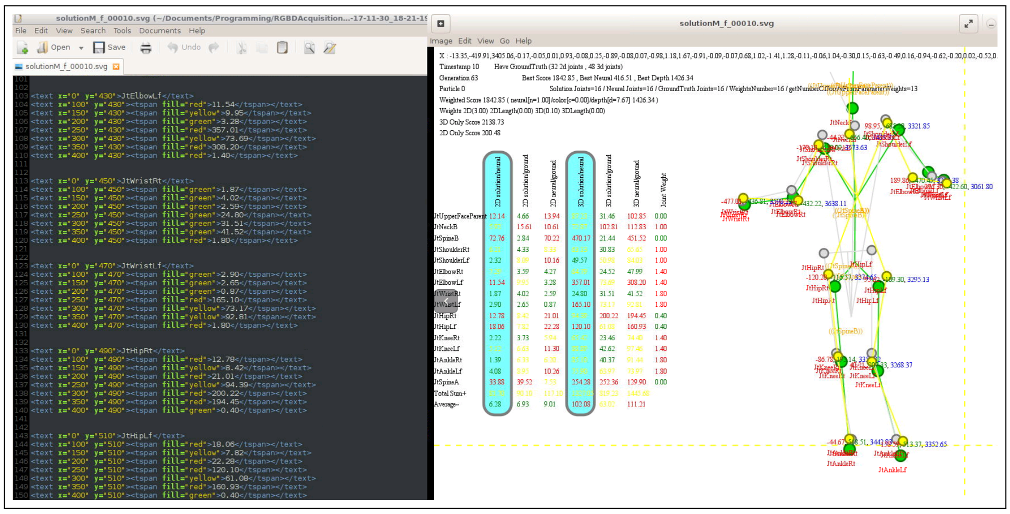


Figure 6.13: The error Δ as a function of the number of particles and generations in PSO optimization. See text for details.

input, after scanning an actor and then tracking him looked very promising but a systematic testing approach was needed in order to assess the accuracy of the proposed method. The main question to be answered was how much would a skinned human model benefit the quality of a generative body tracker compared to a generic model consisting of primitives, or a scanned model of a different person. Other significant questions where how much could a discriminative detector affect tracking quality, what would be good objective function tuning factors in order to achieve a balance between depth correspondence and neural network output. A final question what would be the best depth correspondence with a PSO solution if we had more information (dual camera) [57].

To quantify the error in body pose estimation, we adopt the metric used in [28] which involves the Euclidean distances of skeleton joints in the ground truth and the corresponding points in the estimated body model. The average of all these distances over all the frames of the sequence constitutes the resulting error estimate Δ.

Another metric reports the percentage $A(t)$ of these distances that are within a distance t from their true location. We will refer to this metric as pose estimation accuracy. For example, an accuracy of $A(80) = 70\%$ for a sequence means that in all frames of the sequence, 70% of the joints were estimated within 80mm from the ground truth.


solutionM_f_00010.svg (~/Documents/Programming/RGBDAcquisition...-17-11-30_18-21-1

```
<text x="0" y="430">JtElbowLf</text>
<text x="100" y="430"><tspan fill="red">11.54</tspan></text>
<text x="150" y="430"><tspan fill="yellow">9.95</tspan></text>
<text x="200" y="430"><tspan fill="green">3.28</tspan></text>
<text x="250" y="430"><tspan fill="red">357.01</tspan></text>
<text x="300" y="430"><tspan fill="yellow">73.69</tspan></text>
<text x="350" y="430"><tspan fill="red">308.20</tspan></text>
<text x="400" y="430"><tspan fill="red">1.40</tspan></text>


<text x="0" y="450">JtWristRt</text>
<text x="100" y="450"><tspan fill="green">1.87</tspan></text>
<text x="150" y="450"><tspan fill="green">4.02</tspan></text>
<text x="200" y="450"><tspan fill="green">2.59</tspan></text>
<text x="250" y="450"><tspan fill="green">24.80</tspan></text>
<text x="300" y="450"><tspan fill="green">31.51</tspan></text>
<text x="350" y="450"><tspan fill="green">41.52</tspan></text>
<text x="400" y="450"><tspan fill="red">1.80</tspan></text>


<text x="0" y="470">JtWristLf</text>
<text x="100" y="470"><tspan fill="green">2.90</tspan></text>
<text x="150" y="470"><tspan fill="green">2.65</tspan></text>
<text x="200" y="470"><tspan fill="green">0.87</tspan></text>
<text x="250" y="470"><tspan fill="red">165.10</tspan></text>
<text x="300" y="470"><tspan fill="yellow">73.17</tspan></text>
<text x="350" y="470"><tspan fill="yellow">92.81</tspan></text>
<text x="400" y="470"><tspan fill="red">1.80</tspan></text>


<text x="0" y="490">JtHipRt</text>
<text x="100" y="490"><tspan fill="red">12.78</tspan></text>
<text x="150" y="490"><tspan fill="yellow">8.42</tspan></text>
<text x="200" y="490"><tspan fill="red">21.01</tspan></text>
<text x="250" y="490"><tspan fill="yellow">94.39</tspan></text>
<text x="300" y="490"><tspan fill="red">200.22</tspan></text>
<text x="350" y="490"><tspan fill="red">194.45</tspan></text>
<text x="400" y="490"><tspan fill="green">0.40</tspan></text>


<text x="0" y="510">JtHipLf</text>
<text x="100" y="510"><tspan fill="red">18.06</tspan></text>
<text x="150" y="510"><tspan fill="yellow">7.82</tspan></text>
<text x="200" y="510"><tspan fill="red">22.28</tspan></text>
<text x="250" y="510"><tspan fill="red">120.10</tspan></text>
<text x="300" y="510"><tspan fill="yellow">61.08</tspan></text>
<text x="350" y="510"><tspan fill="red">160.93</tspan></text>
<text x="400" y="510"><tspan fill="green">0.40</tspan></text>
```

solutionM_f_00010.svg

X : -13.35,-419.91,3405.06,-0.17,-0.05,0.01,0.93,-0.08,0.25,-0.89,-0.08,0.07,-0.98,1.18,1.67,-0.91,-0.09,-0.07,0.68,1.02,-1.41,1.28,-0.11,-0.06,1.04,-0.30,0.15,-0.63,-0.49,0.16,-0.94,-0.62,-0.20,0.02,-0.52,0.

Timestamp 10 Have GroundTruth (32 2d joints , 48 3d joints)

Generation 63 Best Score 1842.85 , Best Neural 416.51 , Best Depth 1426.34

Particle 0 Solution Joints=16 / Neural Joints=16 / GroundTruth Joints=16 / WeightsNumber=16 / getNumberOfJointsFromParameterWeights=13

Weighted Score 1842.85 (neural[n=1.00]/color[c=0.00]/depth[d=7.67] 1426.34)

Weights 2D(3.00) 2DLength(0.00) 3D(0.10) 3DLength(0.00)

3D Only Score 2138.73

2D Only Score 200.48

	2D solution/neural	2D solution/ground	2D neural/ground	3D solution/neural	3D solution/ground	3D neural/ground	Joint Weight
JtUpperFaceParent	12.14	4.66	13.94	87.23	31.46	102.85	0.00
JtNeckB	5.92	15.61	10.61	72.87	102.81	112.83	1.00
JtSpineB	72.76	2.84	70.22	470.17	21.44	451.52	0.00
JtShoulderRt	6.21	4.33	8.33	63.13	30.83	65.65	1.00
JtShoulderLf	2.32	8.09	10.16	49.57	50.98	84.03	1.00
JtElbowRt	7.29	3.59	4.27	64.79	24.52	47.99	1.40
JtElbowLf	11.54	9.95	3.28	357.01	73.69	308.20	1.40
JtWristRt	1.87	4.02	2.59	24.80	31.51	41.52	1.80
JtWristLf	2.90	2.65	0.87	165.10	73.17	92.81	1.80
JtHipRt	12.78	8.42	21.01	94.39	200.22	194.45	0.40
JtHipLf	18.06	7.82	22.28	120.10	61.08	160.93	0.40
JtKneeRt	2.22	3.73	5.94	65.42	23.46	74.40	1.40
JtKneeLf	5.12	6.63	11.30	88.89	42.62	97.46	1.40
JtAnkleRt	1.39	6.33	6.20	85.10	40.37	91.44	1.80
JtAnkleLf	4.08	8.95	10.26	75.90	63.97	73.97	1.80
JtSpineA	33.88	39.52	7.53	254.28	252.36	129.90	0.00
Total Sum+	81.70	90.10	117.10	1327.85	819.23	1445.68	
Average~	6.28	6.93	9.01	102.08	63.02	111.21	

Figure 6.14: Snapshot of an SVG log file generated to document a specific frame of a tracking attempt. The svg format proved to be very useful choice for output since it offered both an easily parseable XML file (left), as well as an immediately visualizeable format with highlighted errors and all relevant information conviniently placed. SVG files are natively supported by all graphics programs in Linux and all internet browsers.

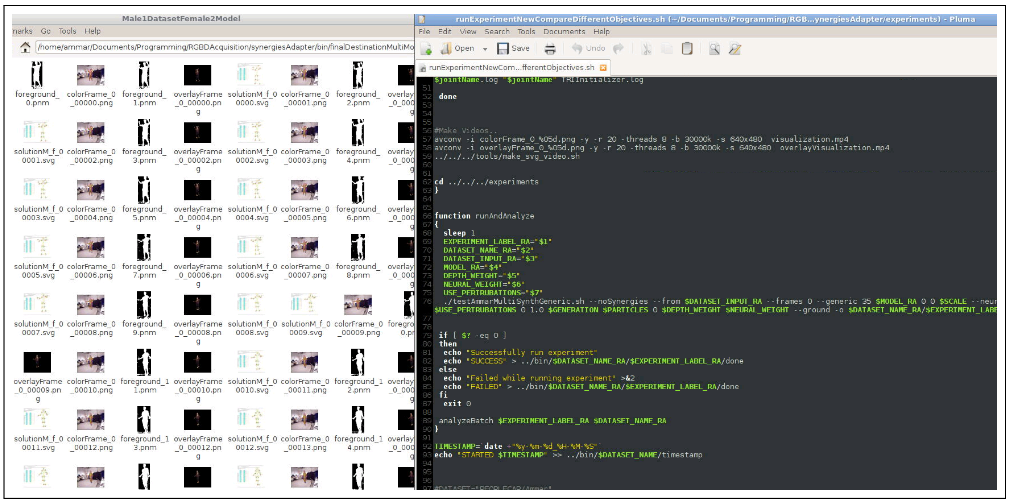
6.5 The experiment system

The experiment system consists of a very well organized optimization framework that can be seen in Figure 5.1. It can be thought as a black box that receives a skinned 3D Mesh, an RGB+D image pair as well as 2D Joint estimations for the frame pair. Its output consists of a full 3D pose solution vector as well as a 3D rendering of the solution and a series of detailed logs that contain every hypothesis formulated by the algorithm until reaching the solution, along with comparisons to ground truth. As seen in Section 6.3 this black box also has a lot of configuration parameters that change it's optimization strategy and resource consumption .

These configurations are provided to the executable via command-line arguments at the start of each run. Each log-file also contains the configuration setup that produced it in order for multiple logs to be directly compareable. Every specific combination of parameters for a specific model/dataset generated over 700MB of data something that made a full run for all combinations of models/datasets for a specific parameter coniguration reach 11GB+.

As seen in Figure 6.14 an SVG file format was used to hold XML based files with the output of the method. This had the added benefit of beeing also automatically rendered by graphics programs and also giving a very high-level to check individual frames.

Bash scripts where used to run and control experiments as well as post process


Male1DatasetFemale2Model
runExperimentNewCompareDifferentObjectives.sh (~/Documents/Programming/RGB...ynergiesAdapter/experiments) - Pluma
File Edit View Search Tools Documents Help
Open
Save
Undo
runExperimentNewCom...fferentObjectives.sh
/home/ammar/Documents/Programming/RGBDAcquisition/synergiesAdapter/bin/finalDestinationMultiMo

```
$jointName.log "$jointName" TRIInitializer.log

 done



#Make Videos..
avconv -i colorFrame_0_%05d.png -y -r 20 -threads 8 -b 30000k -s 640x480  visualization.mp4
avconv -i overlayFrame_0_%05d.png -y -r 20 -threads 8 -b 30000k -s 640x480  overlayVisualization.mp4
../../../tools/make_svg_video.sh


cd ../../../experiments
}


function runAndAnalyze
{
  sleep 1
  EXPERIMENT_LABEL_RA="$1"
  DATASET_NAME_RA="$2"
  DATASET_INPUT_RA="$3"
  MODEL_RA="$4"
  DEPTH_WEIGHT="$5"
  NEURAL_WEIGHT="$6"
  USE_PERTRUBATIONS="$7"
  ./testAmmarMultiSynthGeneric.sh --noSynergies --from $DATASET_INPUT_RA --frames 0 --generic 35 $MODEL_RA 0 0 $SCALE --neur
$USE_PERTRUBATIONS 0 1.0 $GENERATION $PARTICLES 0 $DEPTH_WEIGHT $NEURAL_WEIGHT --ground -o $DATASET_NAME_RA/$EXPERIMENT_LABE


 if [ $? -eq 0 ]
 then
  echo "Successfully run experiment"
  echo "SUCCESS" > ../bin/$DATASET_NAME_RA/$EXPERIMENT_LABEL_RA/done
 else
  echo "Failed while running experiment" >&2
  echo "FAILED" > ../bin/$DATASET_NAME_RA/$EXPERIMENT_LABEL_RA/done
 fi
  exit 0

 analyzeBatch $EXPERIMENT_LABEL_RA $DATASET_NAME_RA
}

TIMESTAMP=`date +"%y-%m-%d_%H-%M-%S"`
echo "STARTED $TIMESTAMP" >> ../bin/$DATASET_NAME/timestamp



#DATASET="PEOPLECAP/Ammar"
```

Figure 6.15: Right: Snapshot of an experiment script and the parameters passed to the optimization framework. Left: A sample log directory output for an experiment . For each optimization frame we get an svg file as described in Figure 6.14 as well as the derived foreground for the person, an overlay frame with the proposed pose and finally a color frame where the skeletons are all overlayed on top of the image to better visualizize what happened .

the log files generated. The log files where processed using the R programming language to acquire statistics and plotted automatically using gnuplot.

Of course all of the configurable values of the internal implementation of the tracking framework could not be modeled by the exposed configuration parameters. What complicated things even more was that chages to the optimizer and fine tuning of its source code, invalidated previous results. Every time the calculation method changed this meant that new results were no longer compareable with old ones. It is worth noting that all of the results presented in this thesis have been acquired using the same final codebase version and changing the specified configuration parameters for each experiment. One of the challenges of the experimental phase, especially during the early stages of the project where the optimization framework was not finalized and good values for the optimization parameters where not yet known was to find stable configuration values. Thankfully the scripting system along with the generated statistics and plots did all the heavy lifting. After scanning a lot of configuration parameter combinations with an initial version of the code the configuration parameters could be frozen and work could be done on improving the optimization engine. Once the optimization engine worked in a better way a narrower search for good configuration parameters could be performed. This gradually led to the final optimization configurations presented in the following sections.

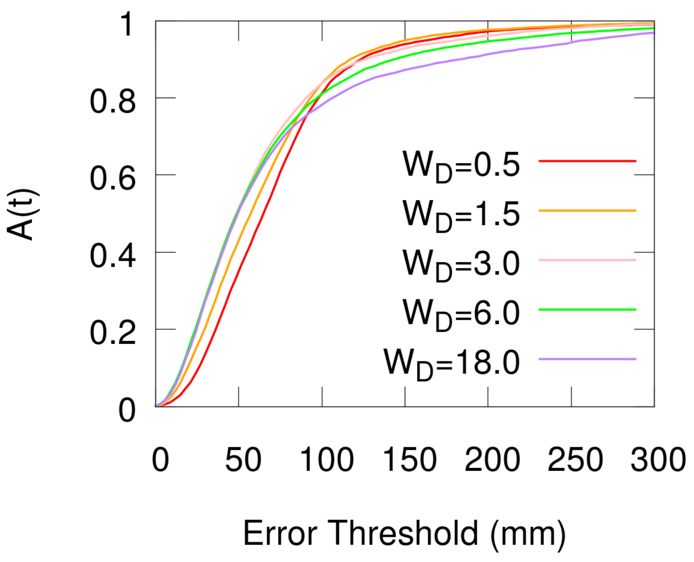


Figure 6.16: Tracking accuracy $A(t)$ for different weights w_D of the depth term E_D in the objective function.

6.6 Experimental results

6.6.1 Quantitative results

Determining the PSO budget: PSO optimizes its objective function by evolving p particles in g generations (see Section 4.1.3). The proper selection of p and g is crucial because it influences the accuracy and the computational requirements of the method. We set p, g based on the following experiment. We tracked the MS sequence for all combinations of $p \in [40, 120]$ with a step of 10 and $g \in [30, 70]$ with a step of 10. For each particles/generations combination, we measured and averaged the error Δ (see Section 6.4) in 5 runs. Figure 6.13 shows Δ as a function of p and g. It can be verified that a budget of 64 particles and 64 generations balances the error/computational resources tradeoff. Therefore, in all subsequent experiments we set $p = g = 64$.

Tuning the objective function: The objective function of Eq.(5.2) consists of two terms, one that depends on the registration of the depth values of the hypothesis and the observation and another that depends on the alignment of the observed and estimated body joints. The weights w_D and w_J control the relative contribution of the two terms. We performed experiments on the MS sequence to investigate the influence of the weights w_D and w_J in the objective function of Eq.(5.2) that control the relative contribution of the depth and the joints terms. We investigated different values of w_D, maintaining $w_J = 1$. Figure 6.16 shows $A(t)$ for values $w_D \in \{0.5, 1.5, 3.0, 6.0, 18.0\}$. For a very broad range of errors t,

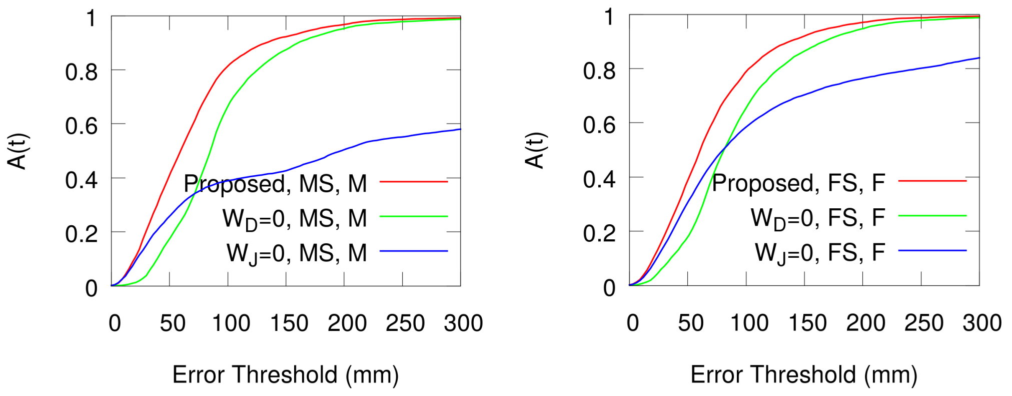


Figure 6.17: The accuracy $A(t)$ of the proposed method for the cases of balanced depth and joints localization terms (red curves), joints-localization-only term (green curves) and depth-only term (blue curves) for the MS (left) and the FS (right) sequences.

$w_D = 3.0$ achieves, overall, the best $A(t)$. Around 10cm, the plots exhibit a switch point, i.e., lower weights become preferable. This is attributed to the fact that for larger allowed errors, the significance of the depth term is less pronounced. With similar experiments, we determined experimentally the values of $\alpha = 0.97$ (Eq.(5.6)) as well as the values for the weights w_i (Eqs.(5.4) and (5.5)).

Proposed vs depth-only vs joints-only: In another experiment, we compared the performance of the proposed method ($w_D = 3.0$, $w_J = 1.0$) with the case where the joints localization term E_J is ignored ($w_J = 0.0$, $w_D = 1.0$) and the case where the depth term E_D is ignored ($w_J = 1.0$, $w_D = 0.0$). Figure 6.17 illustrates the accuracy $A(t)$ obtained in the three cases for the MS (left) and the FS (right) sequences. It can be verified that the depth alone performs the worst. Optimization only with the OpenPose proposals performs better than optimization only with depth. However, fusing and balancing the two terms achieves better performance than any of the terms alone.

Table 6.1 shows the error Δ in these experiments. It can be verified that Δ is significantly lower when the two terms in the objective function are properly balanced.

Two-phases optimizazation: Having a rather accurate estimation of human pose, we performed another experiment to check whether, starting from this solution, we can further reduce Δ by employing a second, refinement phase that optimizes an objective function that consists only of the depth term. This yielded a reduction of Δ of less than 0.5cm. This is in strong support of the objective function of Eq.(5.2).

The initial formulation for the problem used a 2 step optimization procedure. The first step used the previous solution as a starting point and utilizing only the

Sequence, model	$w_D = 1.0$, $w_J = 0.0$	$w_D = 0.0$, $w_J = 1.0$	$w_D = 3.0$, $w_J = 1.0$
MS, **M**	140.2	98.9	**67.3**
FS, **F**	152.1	98.7	**75.6**

Table 6.1: Error Δ (in mm) for experiments isolating different terms of the objective function. Rows correspond to the sequences MS, FS, each of which is tracked with the proper model (**M**, **F**, respectively). Columns correspond to different weighting schemes (depth only, joints only, balanced). Boldface indicates best results.

neural network input produced a solution. The second step utilized the neural network estimation (result of the first step) and with depth input tried to refine it. For the initialization of the second step optimizer we again tried different techniques. Most important variants where the one which included PSO particles with the Neural Network Output while basing all samples around the last solution and the one that based all particles around the solution of the neural network. The latter strategy proved better but this still required twice the processing time, performed worse than the formulation in equation 5.2 since at each step we did not have all the information available in the swarm. This 2 step formulation practically made little difference compared to running just using the depth step so the two steps had to be combined. Having decided to combine the neural and depth terms in the same objective function, there had to be some weighting to make them comparable in magnitude and combinable in a single score. Ideally we would like the neural network term to act like a spring that will cause limbs to gravitate towards their respective places where depth should dominate locally and lead them to their true position as described in the Method section of this work.

Other experiments conducted to study the impact of the parameters W_{depth}, W_{2D}, W_{3D} and Wj as mentioned in the equations 5.1 to 5.6 was to perform tracking using the same model, same dataset, same PSO budget and only change one of the variables at a time to identify their very important role for properly combining all terms.

Baseline vs perturbed PSO: As described in Section 4.1.3, we employ a variant of the PSO in which particles are perturbed during the optimization procedure in order to help them escape local minima. In the experimental setting of Section 6.6.1, we investigated the difference in performance of this perturbed version of PSO relative to the baseline, canonical version [15]. Table 6.2 summarizes the obtained results for different w_Ds. For all tested values, the perturbed PSO provides more accurate pose estimates.

The impact of the personalized human model: We performed experiments to showcase the impact of using a personalized human body model. In that direction, we measured the pose estimation accuracy $A(t)$ when different models are used for

w_D	Baseline PSO (Δ)	Perturbed PSO (Δ)
0.5	82.8	77.0
1.5	67.5	70.0
3.0	84.1	67.3
6.0	93.7	73.5
18.0	108.3	81.2

Table 6.2: Error Δ for the canonical, baseline PSO versus perturbed PSO for different values of the weighting factor w_D.

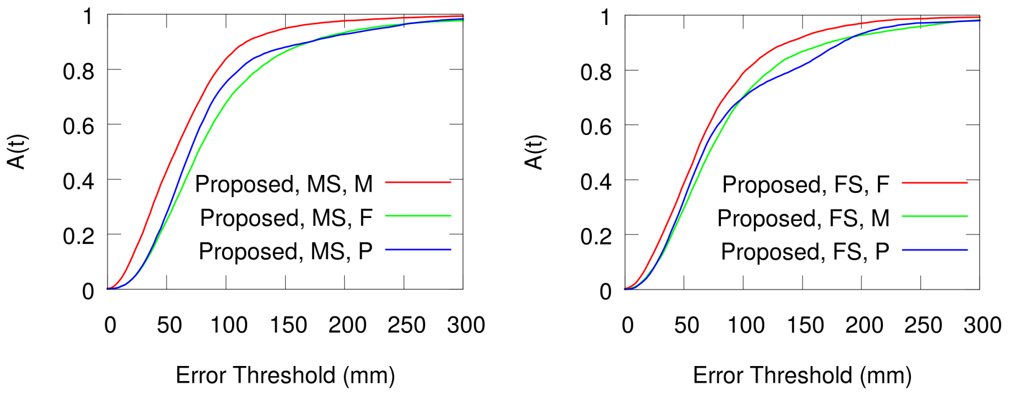


Figure 6.18: The accuracy $A(t)$ when a sequence is tracked with the proper model (red curves), with a scaled model of the opposite sex (green curves) and with a scaled primitives-based model (blue curves). Results are shown for the MS (left) and the FS (right) sequences. See text for details.

tracking. We identified three scenarios: (a) tracking with the correct model (i.e., **M** for MS, **F** for FS), (b) tracking with the model of the opposite sex (i.e., **F** for MS, **M** for FS) and (c) tracking with the primitives model (i.e., **P** for both MS, FS). It should be stressed that in cases (b) and (c) the models have been adjusted to fit the dimensions of the actual person. Thus, differences in performance should be attributed to the individual human body shape differences and not to their absolute scale difference. Figure 6.18 illustrates the obtained results. It can be verified that the best performance is obtained when the personalized model is used for tracking. Interestingly, the performance of a scaled primitives-based model is quite similar to the performance of a scaled model of another person. Experiments using different models as seen in table 6.4 reveal a better behavior that averages at 2cm for each of the joints when using the 3D scanned models, something which comes as no surprise since hypothesis are much closer to observations. Something however that was initially not foreseen is the rival relationship of the neural network

Sequence	**P**rimitives (Δ)	**M**ale (Δ)	**F**emale (Δ)
MS	90.2	**67.3**	97.1
FS	93.3	92.0	**75.6**

Table 6.3: The error Δ when the sequences MS, FS are tracked with models **M**, **F**, **P**. Boldface font indicates best results.

Sequence \ model	$\mathbf{M_1}$	$\mathbf{M_2}$	$\mathbf{F_1}$	$\mathbf{F_2}$	**P**
MS_1	**67.3**	103.7	85.4	87.5	95.1
MS_2	83.3	**76.9**	84.3	94.7	112.2
FS_1	90.2	96.9	**75.6**	99.4	100.2
FS_2	84.7	103.2	92.8	**79.7**	108.1

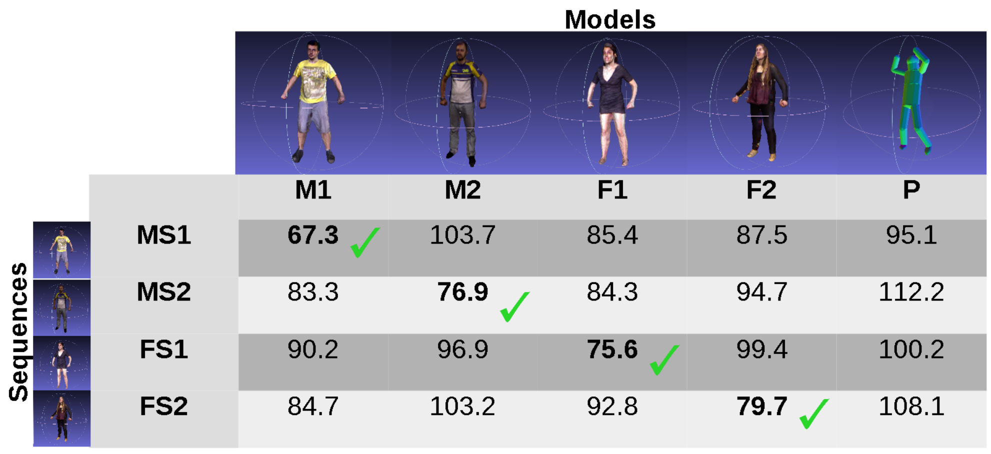
Table 6.4: The error Δ when the sequences MS_i, FS_i are tracked with models $\mathbf{M_i}$, $\mathbf{F_i}$, **P**. Boldface font indicates best results.

term to the generative term. Since the two do not necessarily agree we need to be very careful adjusting W_{depth} since low values there and a high reliance on the neural network detector term makes the model less important. There is a delicate balance to be struck between the neural and depth term and only when the depth term is sufficiently large do we take full advantage of our very precise body model.

Table 6.4 shows the error Δ in these experiments. Using the proper model results in an error reduction at the level of 2.0 cm.

Table 6.4 shows the error Δ in these experiments. The minimal values appearing on the diagonal shows that a sequence is tracked more accurately when the proper (personalized) model is used. Moreover, the personalized model always outperforms the on based on primitives (**P**).

Proposed vs [57]vs [74]: We compare the performance of the proposed method with that of [57], referred to as "Dual RGBD" because it employs synchronized input from two, extrinsically calibrated, wide baseline RGBD cameras. This was possible because of the synthetic nature of the developed datasets that permit the rendering of a scene from different views. Figure 6.20 summarizes the accuracy $A(t)$ of the Dual RGBD (continuous curves) and the proposed (dashed curves) methods for the MS (red) and the FS (green) sequences. The Dual RGBD method achieves higher accuracy than the proposed method. It appears that a simple body model and two wide baseline camera views are preferable to an accurate body model and a single view. However, the Dual RGBD method requires more complex hardware setup (two synched cameras), is computationally more intensive and requires initialization for the first frame. It should also be noted that It should also This partly happens because of utilizing two different RGBD views instead of one and because of a larger budget of 65 generations and 200 particles compared


Models
Sequences

	M1	M2	F1	F2	P
MS1	**67.3** ✓	103.7	85.4	87.5	95.1
MS2	83.3	**76.9** ✓	84.3	94.7	112.2
FS1	90.2	96.9	**75.6** ✓	99.4	100.2
FS2	84.7	103.2	92.8	**79.7** ✓	108.1

Figure 6.19: Table 6.4 error Δ measurments visualized along with the models used.

to the 64 particles 64 generations used in this work. This is useful however as an approximation of the theoretical best behavior of the pPSO generative component. The second is the widely deployed OpenNI method [74]. Figure 6.20 summarizes the accuracy $A(t)$ of the Dual RGBD (red), the OpenNI (blue) and the proposed (green) methods aggregated over all four sequences. The Dual RGBD method achieves higher accuracy than the proposed method. It appears that a simple body model and two wide baseline camera views are preferable to an accurate body model and a single view. However, the Dual RGBD method requires more complex hardware setup (two synchronized and extrinsically calibrated cameras), is computationally more intensive and requires initialization for the first frame. Figure 6.20 also shows that the proposed method outperforms clearly the OpenNI method.

6.6.2 Qualitative results

Figures 6.21, 6.22, 6.23, 6.24, 6.25 and 6.26 show indicative qualitative tracking results. Figures 6.21, 6.22, 6.23, 6.24 show results on the synthetic sequences MS and FS and Figures 6.25, 6.26 results on real data. The estimated skinned models are rendered on the RGB frames. It can be verified that there is a good fit between the estimated body models and the observed human figures.

6.6.3 Computational performance

The computational performance of the method has been measured in Section 5.1.3.4. All experiments presented here were performed on an Intel i7-4790 16GB RAM, NVIDIA Geforce GTX 970 GPGPU and executed in a single thread. The objective function was evaluated in a serial fashion something mandatory to properly log and store all the intermediate results. The Open Pose 2D body tracker

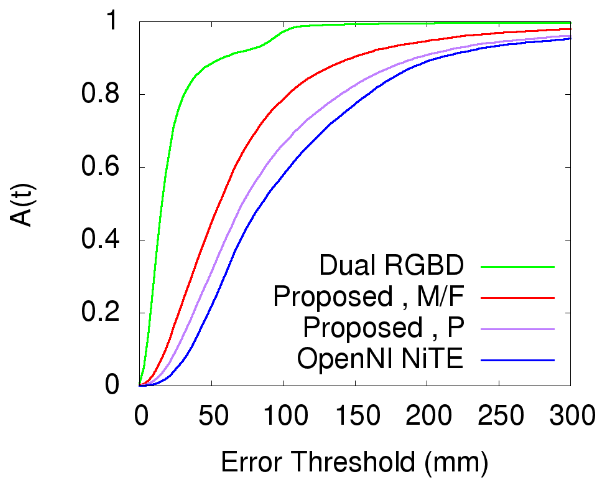


Figure 6.20: The accuracy $A(t)$ of the Dual RGBD method [57] (green), the OpenNI method [74] (blue) and of the proposed one (red) in all four sequences of the dataset.

runs at 10 fps. The version of the proposed method that does not employ the depth term requires no human body rendering, so it runs at 1.68 fps in CPU. The primitives-based model with the full objective function ran at 0.76 fps and the skinned models ($\mathbf{M}$, $\mathbf{F}$) at a little slower 0.72 fps which is expected because of the higher complexity of each rendering. However, as stated in Section 5.1.3.4 the proposed method is very well suited for GPU implementation since all particles in a PSO generation can be computed in parallel. An optimized implementation achieve interactive framerates of $6 - 12$ fps depending on the quality settings.

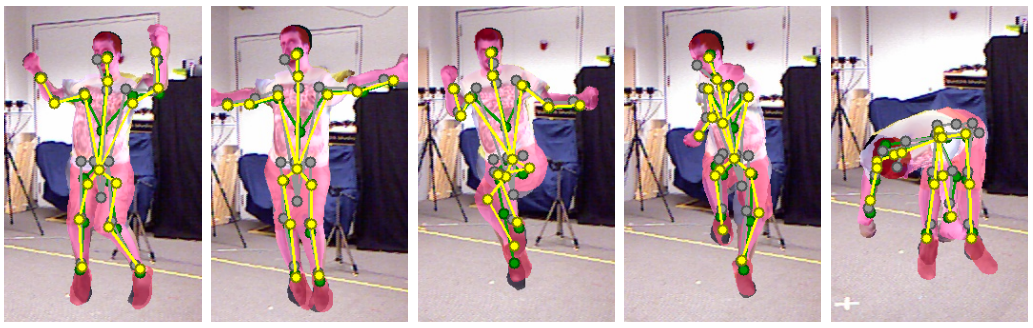


Figure 6.21: Sample qualitative tracking results on the synthetic sequences MS_1. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth.

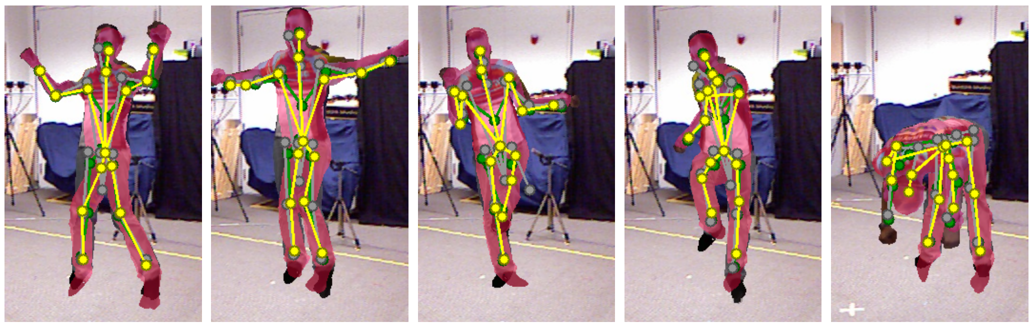


Figure 6.22: Sample qualitative tracking results on the synthetic sequences MS_2. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth.

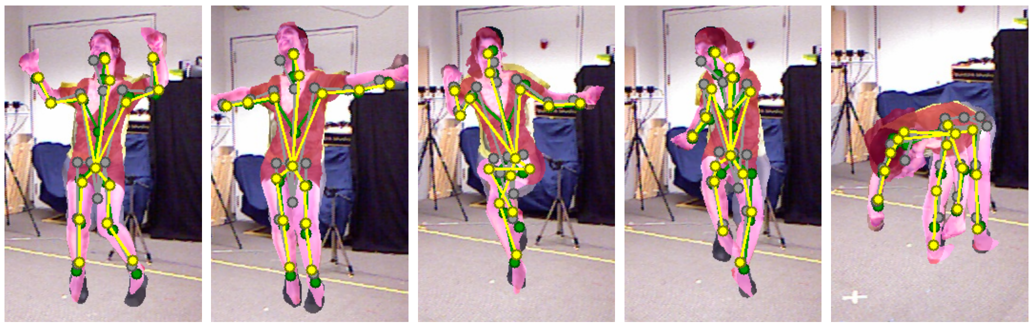


Figure 6.23: Sample qualitative tracking results on the synthetic sequences FS_1. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth.

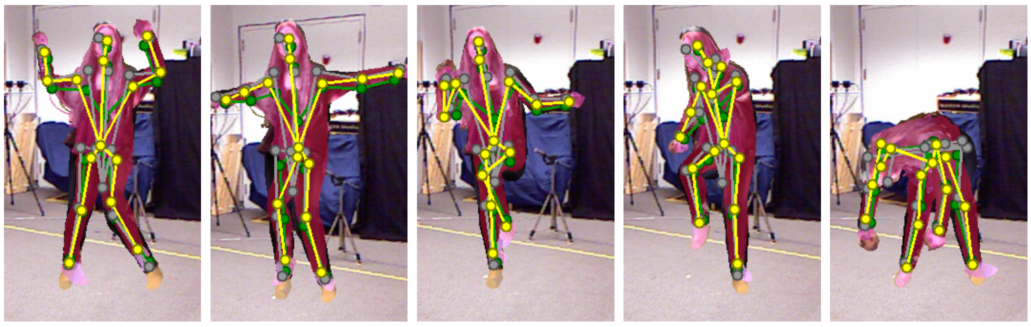


Figure 6.24: Sample qualitative tracking results on the synthetic sequences FS_2. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method, green skeletons: ground truth.

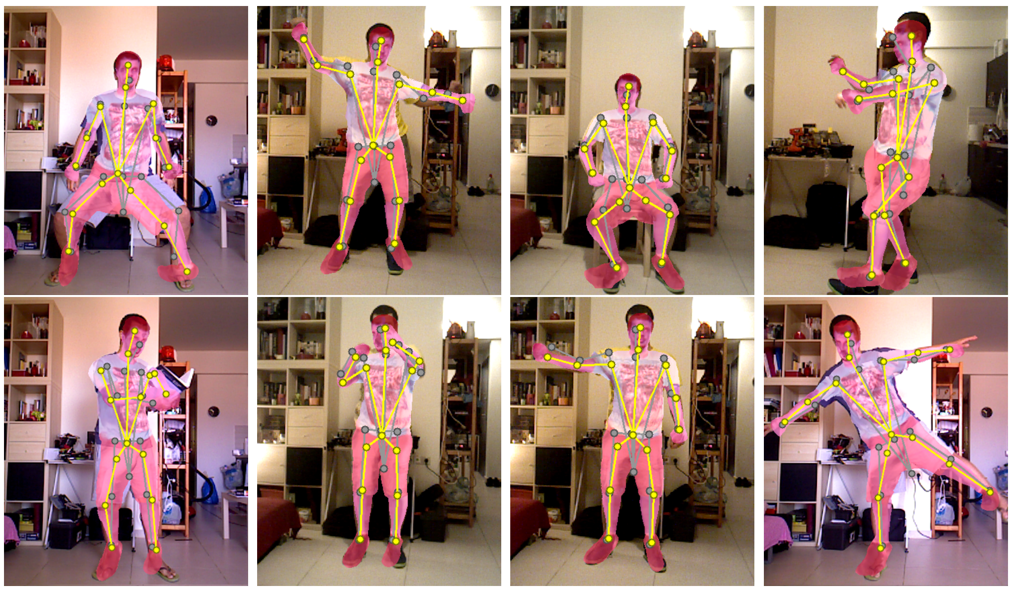


Figure 6.25: Sample qualitative tracking results on real sequences with MS_1 subject. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method.

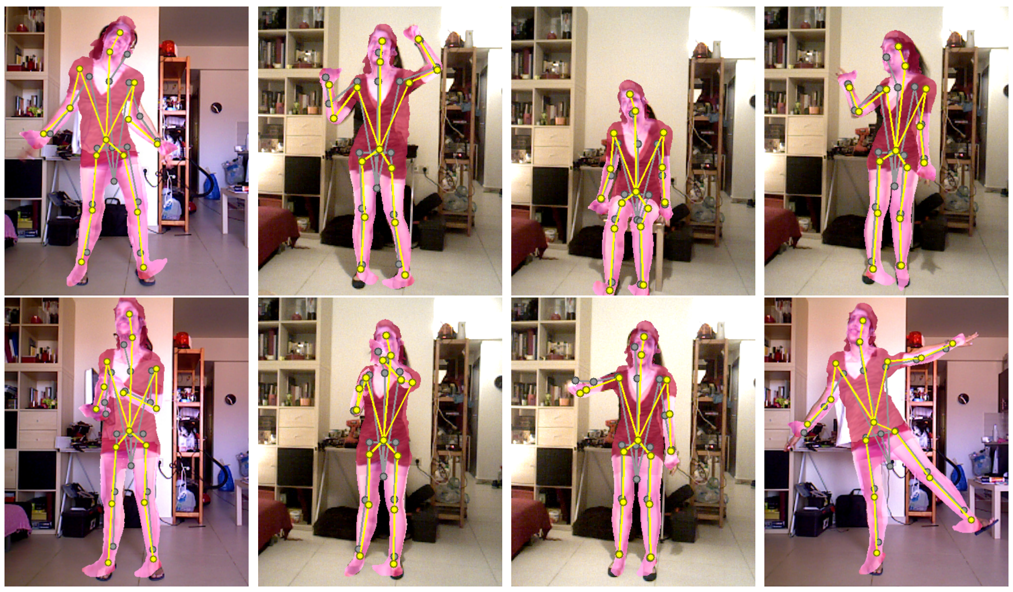


Figure 6.26: Sample qualitative tracking results on real sequences with FS_1 subject. Grey skeletons: OpenPose proposals, yellow skeletons: proposed method.

Chapter 7

Discussion

This chapter concludes the presented human body tracking framework. The resulting framework can be considered successful since it met its intended design goals. Its publication [82] in WACV 2018 makes it one of the first published works to extend a neural network body detector to a fully personalized 3D human tracker, and the first in the case of RGB+D cameras. That being said, and as is the case with most computer vision methodologies, there is still space to further improve it and there is also great potential for its use as a tool to power a method that will tackle even harder problems. In this final chapter and after completing the detailed description of the method and measuring its accuracy experimentally, we will take a step back to re-examine it and propose a summary of incremental upgrades and additions in the spirit of further improving it.

7.1 Our contribution

The focus of this work has been to extend convolutional neural network based 2D human body detectors. With this work the 2D output of a neural network can be converted to a fully 3D estimation. However generative tracking methods have been also extended with this work due to the fact that two of their most important drawbacks, namely initialization and tracking loss/drift have been remedied by the proposed joint estimation component of the objective function. With the simple formulation of the objective function we can now recover from tracking loss since the 2D hints will act as magnets that will recover limbs that might stop registering to the observed motions. We also no longer have the problem of initialization since new detections automatically shift the 3D skeleton close to the 2D estimations and initiate tracking gracefully.

7.2 What can still be improved

We will briefly list possible improvements to the proposed framework which can be considered as future work for researchers that might wish to further extend the method and fill in the gaps unaddressed by the proposed framework.

7.2.1 Live human reconstruction

The first thing that comes to mind that would be an immediate improvement of this work would be merging the currently discrete step of 3D reconstruction into 3D tracking. Of course the method can currently work by skipping reconstruction and using a generic 3D model, but this defeats much of its purpose since as seen in the experimental results (Table 6.4) accuracy suffers from a significant drop when using a parametric model. However we could tolerate to start tracking using a less accurate parametric model and after measuring enough depth samples to perform a fully automatic transition to a reconstructed model. This theoretical single step method will have all the benefits of the current method and not require the separate initialization currently performed by [102],[21]. It could prove to be a very useful strategy that would make the method applicable to a variety of new tracking scenarios. What is more is that by directly acquiring volume slices that when augmented with a parametric model can convert it to the observed person we could quantize and enumerate the observed body and perform 3D human recognition as well. The implementation could be difficult and have unforeseen complications but overall live reconstruction will be a substantial improvement.

7.2.2 Body/Clothes/Face and Hair separation

A related topic of future work to the live reconstruction improvements mentioned in the previous section is to split the now “monolithic” scanned human model to a split representation where human body, human head and clothes are separate entities. The SCAPE [1] model along with [21] do a very good job of creating a composite animation of all these different parts fused together that fits our initial requirements. However by taking a look on Figure 6.1 what we might notice is that human body volumes follow strict patterns while hair styles along with clothes have a much more pronounced effect on the observed appearance of each subject as well as its volume. Human heads and faces in particular are much more unique and have an especially important role in our cognitive system that makes them unfit for generic parametrization. Splitting the human body in the above mentioned part-models could help simplify the live reconstruction procedure and make tracking more modular. Hair and cloth physics simulation could also help with more realistic rendering of the subjects that will in turn maximize accuracy while tracking.

7.2.3 Multiple person tracking

Despite the neural network 2D joint estimator being able to scale to multiple persons without effort, this is not the case for our generative body tracker. Adding a second tracked person in the scene will double the parameter space (from 36 to 72 dimensions). If we treat the dual person tracking problem as a single problem and not two smaller ones this will mean working in an exponentially larger search space, the situation will further deteriorate by adding even more persons. There has been work on this domain in the similar problem of hand-tracking [72] and although there is a method of mitigating the dimensionality problems using an ensemble of collaborative trackers (ECT) [45], the different scale of observed bodies when compared to hands as well as the nature of their movement and the extent of occlusions between them could be an interesting case-study.

7.2.4 Person plus objects tracking

3D human tracking is a very interesting problem, but it is ultimately limited in scope since humans do not exist in a vaccum but instead interact with objects and their surroundings. One of the strongest aspects of the proposed model based method is that it can also accommodate 3D object tracking by a simple extension of the parametric space. This is a big differentiating factor from other methods since a PSO based optimizer can also reason about occlusions between objects and persons in a unified way as seen in [45]. Extending the work presented here to a person+objects tracker would be a very interesting task and will enable a whole new range of applications. The proposed objective function could be directly used to incorporate SURF [4],ORB [96] or other sets of features to facilitate object initialization and tracking in the same hybrid fashion it is performed for human tracking. Convolutional neural network detectors [87], [88] could also be used as a detection source that will dynamically instantiate 3D object tracking and make the method robust to scenes where the viewer is encountered with unexpected objects and persons.

7.2.5 Person plus hands tracking

The tracker presented utilizes an overall very high quality mesh. However the level of detail of the hands is limited since they are 3D scanned as a rigid fist. It is true that the resolution of the sensor we are working with combined with the distance that we observe the body, means that hands can be abstracted as fists. However when a person approaches the camera and especially when he has a transaction with the viewer this will not be the case. For robotic applications and for a potential scenario where the camera/viewer is an autonomous robot that needs to manipulate objects presented by a user, hand tracking could be a very interesting addition to our human tracking problem. It will be straightforward to add a skinned articulated human hand to the 3D body model used here and the kinematics chain provided by [21] already provides joints for fingers, albeit not skinned. Human-Robot interaction scenarios could be an especially compelling case for an extension of our method with PSO based hand tracking since PSO hand tracking has been proved to be very accurate, even allowing computers to infer manipulation forces [78].

7.2.6 RGB/Stereo Camera support

Another potential improvement for the method is adapting it to a different camera system than the RGB+D sensor currently used. Although RGB+D sensors are widely adopted and considered cheap commodity mass marketed devices, they are still not as ubiquitous as RGB devices like webcameras that exist in all mobile phones and laptops. Another drawback they exhibit is their sensitivity to sunlight (due to their infrared projector) which makes them unsuitable for outdoor environments. A final limitation is their short range and low resolution which can be a serious limiting factor for the applicability of the method. Using a Stereo Camera instead of an RGB+D sensor these drawbacks are resolved. Methods like [76] utilize stereo cameras while maintaining a depth objective function component similar to the depth formulation in this work. Using the dual rendering in [76] we could extend a similar PSO optimizer to stereo cameras. Finally future work could even attempt to work using just a single camera with a single view/color only input. Our model fidelity is very high, the single RGB view is suitable as input for our 2D joint detector and on top of that we could use a color instead of a depth objective. Sobel edge detection operators and corner detectors could also be used as tools to mitigate lighting and shadows changes that could negatively affect tracking.

7.2.7 Performance Improvements

Computational performance is a serious concern for computer vision algorithms. Generative methods are typically more computationally intensive than discriminative methods and PSO is considered a demanding optimization scheme. However in our context PSO turns out to perform favorably since it is significantly sped up by leveraging GPGPU technology. During the course of this work performance has been a secondary concern (with the primary concern being accuracy) that has been mostly addressed by relying on the Mobile Based Vision (MBV) software framework [68] which is highly optimized for tracking workloads. Until very late in the project most of the code used single threaded CPU only objective function calculations which was a big bottleneck but was mandatory in order to log and study the objective function behavior as presented in the graphs of our experimental evaluation (Chapter 6) However and even after implementing the thrust optimizer presented in Section 5.1.3.4 I firmly believe that there is still room for further performance optimizations. Earlier work in potential speed ups [85] of generative PSO based trackers managed to significantly reduce resource consumption by focusing optimization budget on parts of the scene that needed it. The same techniques could be applied for the human tracking problem tackled by this work. Further optimizations could reduce the datapath of the application by optimizing computations. Using a renderer based on the Vulcan API that combines rendering with computations or a shader based renderer/optimizer could provide even better performance than the current OpenGL/CUDA implementation. Many such optimizations are technicalities that have little scientific interest but nevertheless they are important polish that is essential for a production-grade software solution.

7.2.8 Different Optimization Schemes

Particle Swarm Optimization is a robust and proven method that has been successfully deployed as an optimization scheme to tackle many problems, as well as our 36D problem. In the context of this work where we wanted to experiment on a hybrid combination of heterogeneous data sources it performed very well as seen from the overall accuracy achieved during experimentation. Its straightforward formulation and its ability for parallelization of rendering and simultaneous evaluation was a very desirable feature. The fact that the gradient of our objective function did not need to be differentiable or follow any other constraints was also very important. However it would be an interesting future endeavor to test the formulation we concluded with using other optimization schemes. Hierarchical PSO, Particle Filters and other methods would be the first candidates for our task. Altering the proposed objective function to better accommodate different optimizers that have specific constraints would also be an interesting endeavor to further explore the topic of hybrid personalized 3D tracking. Decomposition of the problem by introducing a body hierarchy could also be beneficial since assuming we can disect it to 4 distinct 9D problems this would yield a much smaller workload than the originl 36D one due to partial prevention of the combinatorial explosion.

7.3 Applications that can benefit from the framework

The main workload of this thesis focused on proving the proposed method accuracy and testing it quantitatively and qualitatively. In retrospect this left very little time for testing it in the context of the various applications that can be powered by it. Since this method produces tracking output both in the form of 3D limb positions but it also provides a full solution of limb orientations this means it is ideal for applications that rely on inverse kinematics. A brief list of potential applications was offered in the introductory chapter. During the development time we explored the virtual avatar/gaming case where we could animate a model by tracking a different person. A similar, although special case of matching an observed body to a different one is the teleoperation scenario in the context of a humanoid robot. We can directly use the joint angles returned by the method instead of deriving an approximate angle configuration through 3D points as shown in Figure 7.1. We tested the conversion of our tracking results to humanoid robot poses (in the case of a NAO humanoid) although we never proceeded with live tests due to many complications such as the physical stability problems of the NAO platform caused by the inertia of moving arms, or the possible damage to the motors by rapid motions. We briefly experimented with possible medical applications in the scenario of physiotherapy/ergo-therapy for people with moving disabilities after

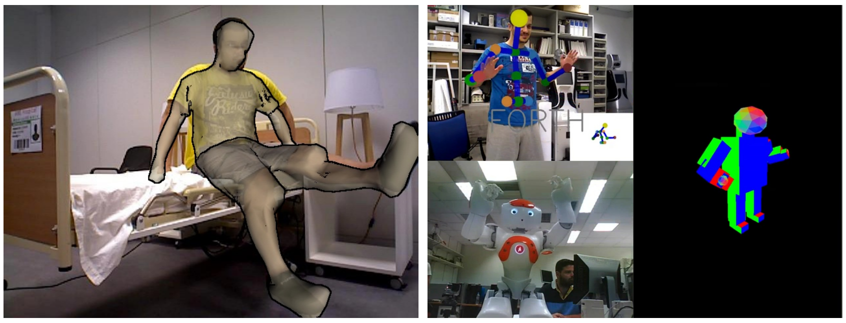


Figure 7.1: The presented framework is very well suited for applications that require detailed tracking. Left: Physiotherapists could use the tracking output to track patient rehabilitation progress after injuries. The parameter space of the problem can be easily constrained to only track the limbs focused by the doctor to better facilitate measurements. Right: Humanoid robot teleoperation could also benefit from the proposed method. The list of angles returned by the tracker can be directly mapped to robot motors and be used to control humanoid robots.

accidents. The tracker not only can provide direct measurements on the range of motions executed by the patient, but the developer can also restrict the parameter space in order to better suit measurements and just contain the limbs a doctor is interested in increasing both the performance as well as the accuracy of the specified estimation as seen in 7.1. These use-cases and tests of the tracker where very limited in scope and time budget and thus did not warrant a special chapter in this thesis. Instead they are briefly mentioned here for the sake of completeness.

7.4 Epilogue

Having thoroughly documented all of the design process, the fundamental theory and literature behind the chosen framework modules, the implementation details, experimental results and thoughts on future research, this method has been fully recorded in print and will hopefully stand the test of time. This text is a source of information on our attempts to tackle the human perception problem in a time where machine learning and neural networks are gaining traction sometimes replacing traditional computer vision methodologies. Our hybrid approach could provide a useful tool as well as insight and inspiration for better ideas to come.

Bibliography

[1] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: shape completion and animation of people. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 408–416. ACM, 2005.

[2] Andreas Baak, Meinard Muller, Gaurav Bharaj, Hans-Peter Seidel, and Christian Theobalt. A data-driven approach for real-time full body pose reconstruction from a depth camera. In *IEEE ICCV*, pages 1092–1099. IEEE, 2011.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[5] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[6] Paul J Besl, Neil D McKay, et al. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[7] Osameh Biglari, Reza Ahsan, and Majid Rahi. Human detection using surf and sift feature extraction methods in different color spaces. *Journal of Mathematics and Computer Science*, 11(11):111–122, 2014.

[8] A. Bisacco, Y. Ming-Hsuan, and S. Soatto. Fast human pose estimation using appearance and motion via multi-dimensional boosting regression. In *IEEE CVPR*, 2007.

[9] Cynthia Breazeal. Jibo, the world's first social robot for the home, 2014.

[10] CG Broyden. A new method of solving nonlinear simultaneous equations. *The Computer Journal*, 12(1):94–99, 1969.

[11] Nikhil Buduma and Nicholas Locascio. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms.* O'Reilly Media, Inc., 1st edition, 2017.

[12] Laïka by CamToy. Laïka: An interactive companion for you and your dog, 2017.

[13] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1611.08050*, 2016.

[14] Lulu Chen, Hong Wei, and James Ferryman. A survey of human motion analysis using depth imagery. *Pattern Recognition Letters*, 34(15):1995 – 2006, 2013. Smart Approaches for Human Action Recognition.

[15] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[16] S. Corazza, L. Mundermann, E. Gambaretto, G. Ferrigno, and T. Andriacchi. Markerless motion capture through visual hull, articulated icp and subject specific model generation. *IJCV*, 87(1-2):156–169, 2010.

[17] Jeff Debrosse. Nixie - redefining the personal robot, 2018.

[18] J. Deutscher and I. Reid. Articulated body motion capture by stochastic search. *IJCV*, 61(2):185–205, 2005.

[19] Yuanqiang Dong and Guilherme N DeSouza. A new hierarchical particle filtering for markerless human motion capture. In *Computational Intelligence for Visual Intelligence, 2009. CIVI'09. IEEE Workshop on*, pages 14–21. IEEE, 2009.

[20] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 61(1):55–79, January 2005.

[21] Andrew Feng, Dan Casas, and Ari Shapiro. Avatar Reshaping and Automatic Rigging Using a Deformable Model. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, pages 57–64, Paris, France, November 2015. ACM Press.

[22] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Trans. Comput.*, 22(1):67–92, January 1973.

[23] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[24] N. Platis T. Theoharis G. Papaioannou, N. M. Patrikalakis. *Graphics and Visualization: Principles & Algorithms.* A K Peters, ISBN-10: 1568812744, ISBN-13: 9781568812748, first edition, 2007.

[25] J. Gall, B. Rosenhahn, T. Brox, and H-P. Seidel. Optimization and filtering for human motion capture. *IJCV*, 87(1-2):75–92, 2010.

[26] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H. P Seidel. Motion capture using joint skeleton tracking and surface estimation. In *IEEE CVPR*, pages 1746–1753, 2009.

[27] Varun Ganapathi, Christian Plagemann, Sebastian Thrun, and Daphne Koller. Real time motion capture using a single time-of-flight camera. In *IEEE CVPR*, 2010.

[28] H. Hamer, K. Schindler, E. Koller-Meier, and L. Van Gool. Tracking a hand manipulating an object. In *IEEE ICCV*, 2009.

[29] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[30] Rodolphe Hasselvander. Buddy robot, 2015.

[31] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[32] Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Christian Theobalt. Real-time body tracking with one depth camera and inertial sensors. In *IEEE ICCV*, pages 1105–1112. IEEE Computer Society, 2013.

[33] HONDA. Honda 3e concept robots 2018, 2018.

[34] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *Visualization '99. Proceedings*, pages 59–510, Oct 1999.

[35] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, Apr 2013.

[36] Ingen Dynamics Inc. Aido robot, 2016.

[37] Ingen Dynamics Inc. Aido robot, 2016.

[38] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11. ACM, 2011.

[39] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 39–46, New York, NY, USA, 2007. ACM.

[40] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, 27(4):105:1–105:23, November 2008.

[41] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Trans. Graph.*, 32(3):29:1–29:13, July 2013.

[42] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106, Nov 2013.

[43] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[45] Nikolaos Kyriazis and Antonis A Argyros. Scalable 3d tracking of multiple interacting objects. In *IEEE CVPR*, pages 3430–3437, Columbus, Ohio, USA, June 2014. IEEE.

[46] Nikolaos Kyriazis, Iason Oikonomidis, and Antonis A Argyros. A gpu-powered computational framework for efficient 3d model-based vision. Technical Report TR-420, 2011.

[47] LG. Lg cloi, 2018.

[48] S. Li, W. Zhang, and A. Chan. Maximum-margin structured learning with deep networks for 3d human pose estimation. In *IEEE ICCV*, pages 2848–2856, 2015.

[49] P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *Visualization '98. Proceedings*, pages 279–286, Oct 1998.

[50] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[51] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[52] Machisto.com. Robit, the world's most affordable home robot, 2017.

[53] Alexandros Makris, Nikolaos Kyriazis, and Antonis A Argyros. Hierarchical particle filtering for 3d hand tracking. In *IEEE Computer Vision and Pattern Recognition Workshops (HANDS 2015 - CVPRW 2015)*, pages 8–17, Boston, USA, June 2015. IEEE.

[54] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Netw.*, 16(5-6):555–559, June 2003.

[55] Dushyant Mehta, Srinath Sridhar, Oleksandr Sotnychenko, Helge Rhodin, Mohammad Shafiei, Hans-Peter Seidel, Weipeng Xu, Dan Casas, and Christian Theobalt. Vnect: Real-time 3d human pose estimation with a single rgb camera. *ACM Transactions on Graphics*, 36(4), 2017.

[56] D. Michel and A. Argyros. Apparatuses, methods and systems for recovering a 3-dimensional skeletal model of the human body, March 2016.

[57] D. Michel, C. Panagiotakis, and A. Argyros. Tracking the articulated motion of the human body with two rgbd cameras. *Machine Vision Applications*, 26(1):41–54, 2015.

[58] Damien Michel, Ammar Qammaz, and Antonis A Argyros. Markerless 3d human pose estimation and tracking based on rgbd cameras: an experimental evaluation. In *International Conference on Pervasive Technologies Related to Assistive Environments (PETRA 2017)*, pages 115–122, Rhodes, Greece, June 2017. ACM.

[59] I. Mikic, M. Trivedi, E. Hunter, and P. Cosman. Human body model acquisition and tracking using voxel data. *IJCV*, 53(3):199–223, 2003.

[60] Krystian Mikolajczyk, Cordelia Schmid, and Andrew Zisserman. Human detection based on a probabilistic assembly of robust part detectors. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 69–82, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[61] Japan MJI Inc. Tokyo. Tapia: Ai robot companion learning your lifestyle, 2017.

[62] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(2):90 – 126, 2006. Special Issue on Modeling People: Vision-based understanding of a person's shape, appearance, movement and behaviour.

[63] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[64] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[65] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436, June 2015.

[66] C. V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pages 524–530, Oct 2012.

[67] Matthias Niessner, Michael Zollhofer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6):169:1–169:11, November 2013.

[68] Computer Vision Robotics Lab Foundation of Research and Technology Hellas. Forth modelbasedtracker, 2011.

[69] I. Oikonomidis, N. Kyriazis, and A. Argyros. Efficient Model-based 3D Tracking of Hand Articulations using Kinect. In *BMVC*, Dundee, UK, 2011.

[70] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Efficient model-based 3d tracking of hand articulations using kinect. In *BMVC*, pages 1–11. BMVA Press, BMVA, 2011.

[71] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Tracking the articulated motion of two strongly interacting hands. In *IEEE CVPR*, pages 1862–1869. IEEE, June 2012.

[72] Iason Oikonomidis, Nikolaos Kyriazis, and Antonis A Argyros. Tracking the articulated motion of two strongly interacting hands. In *IEEE Computer Vision and Pattern Recognition (CVPR 2012)*, pages 1862–1869, Providence, Rhode Island, USA, June 2012. IEEE.

[73] Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 2018. https://distill.pub/2018/building-blocks.

[74] OpenNI. *OpenNI User Guide*. OpenNI organization, November 2010.

[75] Pashalis Padeleris, Xenophon Zabulis, and Antonis A Argyros. Head pose estimation on depth data based on particle swarm optimization. In *IEEE Computer Vision and Pattern Recognition Workshops (CVPRW 2012)*, pages 42–49. IEEE, June 2012.

[76] Paschalis Panteleris and Antonis A Argyros. Back to rgb: 3d tracking of hands and hand-object interactions based on short-baseline stereo. In *IEEE International Conference on Computer Vision Workshops (HANDS 2017 - ICCVW 2017). Also available at arxiv.*, pages 575–584, Venice, Italy, October 2017. IEEE.

[77] Katharina Pentenrieder. Sensor fusion using the kalman filter, March 2005.

[78] Tu-Hoa Pham, Abderrahmane Kheddar, Ammar Qammaz, and Antonis A Argyros. Towards force sensing from vision: Observing hand-object interactions to infer manipulation forces. In *IEEE Computer Vision and Pattern Recognition (CVPR 2015)*, pages 2810–2819, Boston, USA, June 2015. IEEE.

[79] Christian Plagemann, Varun Ganapathi, Daphne Koller, and Thrun Sebastian. Real-time identification and localization of body parts from depth images. In *ICRA*, 2010.

[80] G. Pons-Moll, L. Leal-Taixe, T. Truong, and B. Rosenhahn. Efficient and robust shape matching for model based human motion capture. In Rudolf Mester and Michael Felsberg, editors, *Pattern Recognition*, volume 6835 of *LNCS*, pages 416–425. Springer, 2011.

[81] David Poole. *Linear Algebra: A Modern Introduction.* Cengage Learning, ISBN10: 1-285-46324-2, ISBN13: 978-1-285-46324-7, forth edition, 2015.

[82] A. Qammaz, D. Michel, and Antonis A Argyros. A hybrid method for 3d pose estimation of personalized human body models. In *IEEE Winter Conference on Applications of Computer Vision (WACV 2018)*. IEEE, March 2018.

[83] A. Qammaz, D. Michel, and Antonis A Argyros. Public rgbd dataset : A hybrid method for 3d pose estimation of personalized human body models, 2018.

[84] Ammar Qammaz. Rgbdacquisition, a uniform library wrapper for input from v4l2,freenect,openni,openni2,depthsense,intel realsense,opengl simulations and other types of video and depth input, 2013.

[85] Ammar Qammaz, Nikolaos Kyriazis, and Antonis A Argyros. Boosting the performance of model-based 3d tracking by employing low level motion cues. In *British Machine Vision Conference (BMVC 2015)*, pages 144–1, Swansea, UK, September 2015. BMVA.

[86] Deva Ramanan. Learning to parse images of articulated bodies, 2006.

[87] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[88] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[89] Olly Robot. Olly robot, 2017.

[90] Aeolus Robotics. Aeolus robot, 2018.

[91] AMY Robotics. Amy robot: Your thoughtful assistant., 2018.

[92] G. Rogez and C. Schmid. Mocap-guided data augmentation for 3d pose estimation in the wild. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 3108–3116. Curran Associates, Inc., 2016.

[93] Gregory Rogez, Philippe Weinzaepfel, and Cordelia Schmid. Lcr-net: Localization-classification-regression for human pose, 07 2017.

[94] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[95] Henry Roth. Moving volume kinectfusion. 01 2012.

[96] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[97] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[98] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[99] Iasonas Kokkinos Rıza Alp Guler, Natalia Neverova. Densepose: Dense human pose estimation in the wild. *arXiv preprint arXiv:1802.00434*, 2018.

[100] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H.J. Kelly, and Andrew J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[101] Juergen Schmidhuber. Deep learning in neural networks: An overview. *arXiv preprint arXiv:1404.7828*, 2014.

[102] Ari Shapiro, Andrew Feng, Ruizhe Wang, Hao Li, Mark Bolas, Gerard Medioni, and Evan Suma. Rapid avatar capture and simulation using commodity depth sensors. *Computer Animation and Virtual Worlds*, 25(3-4):201–211, 2014.

[103] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. In *IEEE CVPR*, 2011.

[104] Jamie Shotton, Andrew Fitzgibbon, Andrew Blake, Alex Kipman, Mark Finocchio, Bob Moore, and Toby Sharp. Real-time human pose recognition in parts from a single depth image. IEEE, June 2011.

[105] L. Sigal, M. Isard, H. Haussecker, and M. Black. Loose-limbed people: Estimating 3d human pose and motion using non-parametric belief propagation. *IJCV*, 98(1):15–48, 2012.

[106] C. Sminchisescu, A. Kanaujia, Zhiguo Li, and D. Metaxas. Discriminative density propagation for 3d human motion estimation. In *IEEE CVPR*, volume 1, pages 390–397, 2005.

[107] SONY. Sony aibo 2018, 2018.

[108] Stefanos Stefanou and Antonis A Argyros. Efficient scale and rotation invariant object detection based on hogs and evolutionary optimization techniques. In *Advances in Visual Computing (ISVC 2012)*, pages 220–229. Springer, July 2012.

[109] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *CoRR*, abs/1710.08864, 2017.

[110] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[111] Jing Tong, Jin Zhou, Ligang Liu, Zhigeng Pan, and Hao Yan. Scanning 3d full human bodies using kinects. 18:643–50, 04 2012.

[112] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *IEEE CVPR*, pages 1653–1660. IEEE Computer Society, 2014.

[113] Greg Turk. The ply polygon file format. *Georgia Institute of Technology*, 1998.

[114] M. A. Turk and A. P. Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591, Jun 1991.

[115] Carnegie Mellon University. Cmu graphics lab motion capture database. `http://mocap.cs.cmu.edu/`. Accessed: 2017-06-01.

[116] Rene Vidal, Ruzena Bajcsy, Ferda Ofli, Rizwan Chaudhry, and Gregorij Kurillo. Berkeley mhad: A comprehensive multimodal human action database. In *Proceedings of the 2013 IEEE Workshop on Applications of Computer Vision (WACV)*, WACV '13, pages 53–60. IEEE Computer Society, 2013.

[117] John Vijay, Emanuele Trucco, and Spela Ivekovic. Markerless human articulated tracking using hierarchical particle swarm optimisation. *Image and Vision Computing*, 28(11):1530–1547, 2010.

[118] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[119] Timo von Marcard, Gerard Pons-Moll, and Bodo Rosenhahn. Human pose estimation from video and imus. *Transactions on Pattern Analysis and Machine Intelligence*, 38(8):1533–1547, January 2016.

[120] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[121] Xiaolin Wei, Peizhao Zhang, and Jinxiang Chai. Accurate realtime full-body motion capture using a single depth camera. *ACM Trans. Graph.*, 31(6):188:1–188:12, November 2012.

[122] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, Thomas Whelan, John Mcdonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J. Leonard. Kintinuous: Spatially extended kinectfusion, 2012.

[123] P. Koutlemanis X. Zabulis, M. Lourakis. Correspondence-free pose estimation for 3d objects from noisy depth data. *The Visual Computer*, 2:193–211, 2016.

[124] H. Yasin, U. Iqbal, B. Kruger, A. Weber, and J. Gall. A dual-source approach for 3d pose estimation from a single image. In *IEEE CVPR*, 2016.

[125] Mao Ye, Wang Xianwang, Ruigang Yang, Ren Liu, and Marc Pollefeys. Accurate 3d pose estimation from a single depth image. In *IEEE ICCV*, pages 731–738, 2011.

[126] Ming Zeng, Jiaxiang Zheng, Xuan Cheng, and Xinguo Liu. Templateless quasi-rigid shape modeling with implicit loop-closure. In *Proceedings*

/ CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 145–152, 06 2013.

[127] L. Zhang, J. Sturm, D. Cremers, and D. Lee. Real-time human motion tracking using multiple depth cameras. In *IROS*, Oct. 2012.

[128] Michael Zollhöfer, Matthias Niessner, Shahram Izadi, Christoph Rehmann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, and Marc Stamminger. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Trans. Graph.*, 33(4):156:1–156:12, July 2014.