Chapter 5
A Data-Driven Approach for Real-Time Full Body Pose Reconstruction from a Depth Camera

Andreas Baak, Meinard Müller, Gaurav Bharaj, Hans-Peter Seidel, and Christian Theobalt

Abstract The 3D reconstruction of complex human motions from 2D color images is a challenging and sometimes intractable problem. The pose estimation problem becomes more feasible when using streams of 2.5D monocular depth images as provided by a depth camera. However, due to low resolution of and challenging noise characteristics in depth camera images as well as self-occlusions in the movements, the pose estimation task is still far from being simple. Furthermore, in real-time scenarios, the reconstruction task becomes even more challenging since global optimization strategies are prohibitive. To facilitate tracking of full-body human motions from a single depth-image stream, we introduce a data-driven hybrid strategy that combines local pose optimization with global retrieval techniques. Here, the final pose estimate at each frame is determined from the tracked and retrieved pose hypotheses which are fused using a fast selection scheme. Our algorithm reconstructs complex full-body poses in real time and effectively prevents temporal drifting, thus making it suitable for various real-time interaction scenarios.

A. Baak (✉)
MPI Informatik & Saarland University, Campus E1.4, 66123 Saarbrücken, Germany
e-mail: abaak@mpi-inf.mpg.de

M. Müller
Bonn University & MPI Informatik, Römerstraße 164, 53117 Bonn, Germany
e-mail: meinard@mpi-inf.mpg.de

G. Bharaj · H.-P. Seidel · C. Theobalt
MPI Informatik, Campus E1.4, 66123 Saarbrücken, Germany

G. Bharaj
e-mail: gbharaj@mpi-inf.mpg.de

H.-P. Seidel
e-mail: hpseidel@mpi-inf.mpg.de

C. Theobalt
e-mail: theobalt@mpi-inf.mpg.de

A. Fossati et al. (eds.), *Consumer Depth Cameras for Computer Vision*,
Advances in Computer Vision and Pattern Recognition, DOI 10.1007/978-1-4471-4640-7_5,

5.1 Introduction

In recent years, several approaches for marker-less human pose estimation from multiple video streams have been presented [5, 9, 13, 17, 51]. While multi-view tracking already requires solving challenging non-linear optimization problems, monocular pose estimation puts current technology to its limits since, with intensity images alone, the problem is considerably underconstrained [8, 31, 37]. In order to have a chance to reconstruct human movements, non-trivial inference or optimization steps are needed in combination with strong priors. In general, real-time reconstruction of complex human motions from monocular intensity-image sequences can still be considered an open problem.

New depth sensors, such as time-of-flight (ToF) cameras or the Microsoft Kinect sensor, provide depth images at video frame rates. In such images, each pixel stores a depth value instead of a color value. Since this representation of a scene stands somewhere in the middle between a pure 2D color-based representation and full 3D scene geometry, depth images are also referred to as 2.5D data [25]. It turns out that with depth cameras the 3D pose reconstruction of human poses from a single viewpoint becomes more feasible as shown in [7, 15, 18, 24, 35, 48, 56], see also Sect. 5.2. In this chapter, we present a tracking framework that yields robust pose estimates from monocular depth-image sequences. Moreover, our framework enables significant speed-ups of an order of magnitude compared to most of the previous approaches. In fact, we reach similar run time behavior as the algorithm implemented in the Microsoft Kinect [48], whereas we do not need multicore or GPU implementations.

Our procedure follows a hybrid strategy combining generative and discriminative methods, which is an established paradigm for pose estimation and tracking problems. While local optimization strategies [24] have proven to yield high frame rates, such techniques tend to fail for fast motions. Algorithms using global optimization provide more reliable pose estimates, but are typically slow and prohibitive for real-time scenarios. Various data-driven approaches have also been suggested to overcome some of these issues, enabling fast yet robust tracking from intensity-image streams, see [34, 40, 46, 52]. These approaches rely on databases that densely cover the range of poses to be tracked, and fail on poses that are not contained in the database. Moreover, due to the high variability of general human motion, constructing such a database might become intractable. Hybrid strategies that combine generative and discriminative methods have proven to be a suitable methodology for pose estimation and tracking procedures, see [4, 12, 18, 41, 43, 50, 55]. In this work, the main idea is to stabilize generative optimization algorithms by a discriminative component based on a database lookup or a classification scheme. Using this strategy, the risk of getting stuck in local minima is significantly reduced, while time-consuming global optimization methods are avoided.

In our approach, we employ a data-driven hybrid strategy conceptually similar to [12], where local optimization is combined with global retrieval techniques, see Fig. 5.1 for an overview. In our scenario, an actor may perform even complex and fast motions in a natural environment facing a single depth camera at a reasonable

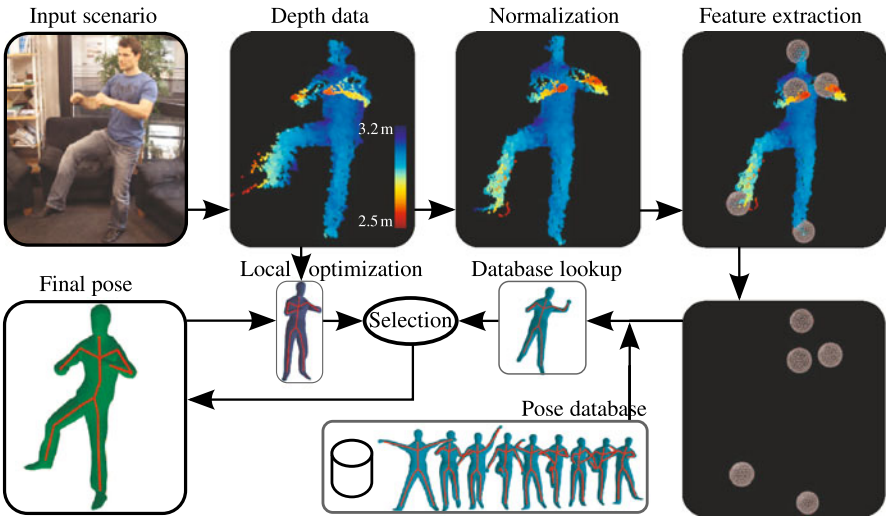

Fig. 5.1 Overview of our proposed pose estimation framework. Using features extracted from the depth data, we retrieve a pose candidate using a database lookup scheme. An additional candidate is obtained by running a local optimization algorithm that is initialized with the final pose of the previous frame. Then, a hypothesis selection decides for the either of the candidates

distance. Similar to [12], we retrieve a pose hypothesis from a large database of 3D poses using sparse features extracted from the depth input data. Additionally, a further hypothesis is generated based on the previously tracked frame. After a local optimization of both hypotheses, a late-fusion selection approach combines the hypotheses to yield the final pose. While the overall procedure is inspired by previous work [12, 18], we explain a number of novel techniques which add robustness and significantly speed up computations at various stages including efficient feature computation, efficient database lookup, and efficient hypothesis selection. In our experiments, we also compare our pose estimation results to previous work using the publicly available benchmark data set [18]. We gain significant improvements in accuracy and robustness (even for noisy ToF data and fast motions) while achieving frame rates of up to 100 fps (as opposed to 4 fps reported in [18]).

Contributions In this chapter, we present a system for full-body pose estimation from monocular depth images that requires only 10 to 16 milliseconds per frame on a standard single-core desktop PC, while being able to track even fast and complex full-body motions. Following a data-driven hybrid strategy that combines local pose estimation with global retrieval techniques, we introduce several technical improvements. Firstly, in the feature extraction step, we introduce a variant of Dijkstra's algorithm that allows for efficiently computing a large number of geodesic extrema. Secondly, in the retrieval step, we employ an efficient database lookup scheme where semantic labels of the extrema are not required. Thirdly, we describe a novel late-fusion scheme based on an efficiently computable sparse and

symmetric distance measure. It is the combination of all these techniques that avoids computational bottlenecks while providing robust tracking results.

The remainder of this chapter is organized as follows. After giving an overview about related work in Sect. 5.2, we describe the input data and preprocessing steps in Sect. 5.3. The steps of the pose reconstruction framework are defined in Sect. 5.4. We describe our extensive experiments in Sect. 5.5, before we conclude in Sect. 5.6.

5.2 Related Work

Intensity-Image-Based Tracking Monocular 3D human pose tracking from intensity images has become an important research topic. In order to deal with challenges coming from occlusions and missing 3D information, different approaches have been pursued making use of statistical body models [21], physical constraints [53], object interaction [39], or motion capture data [41]. For example, Guan et al. [21] fit a statistical model of human body pose and shape to a single image using cues such as silhouettes, edges, and smooth shading. In a similar vein, Hasler et al. [22] present a method for estimating human body pose and shape from single images using a bilinear statistical model. Physics-based constraints are used in Brubaker et al. [10], where a physically based modeling of the lower body helps to track walking motions from monocular images. In [53], the authors propose to annotate parts of image sequences with 2D joint positions, bone directions, and environmental contacts. From such annotations and the image data, they compute physically realistic human motion. As a different type of constraint, the interaction with objects can be exploited as demonstrated in [39]. Having a database of admissible poses, approaches such as [33, 34, 40, 46, 52] compute a mapping from image features to database poses. With such discriminative approaches, poses that are not contained in the database are difficult to recover. By combining generative and discriminative approaches, robust and smooth pose estimates from a single view can be achieved [14, 41, 43, 50].

Depth-Image-Based Tracking 3D human pose tracking based on a single depth-image stream has received increasing attention in the last years. While it appears like a simpler problem at first sight, one still has to deal with noise in the input data, low resolution sensors, lack of color information, and occlusion problems. Nowadays, commercial packages [7] or software libraries exist that can compute joint positions from depth images for multiple people in real time (Microsoft Kinect SDK [30], Primesense NITE middleware [38]). While the algorithm behind the NITE middleware is not revealed to the public, Microsoft published the approach that is implemented in the Kinect SDK in [48]. As explained in the previous chapter, the authors use randomized decision forests trained on a huge set of various body poses and shapes in order to hypothesize joint locations from features on the raw depth input data. The approach was recently combined with a regression scheme to predict joint locations more accurately [19]. Also, positions of occluded joints can be estimated.

Some pose reconstruction approaches use *global optimization* methods in order to solve the pose reconstruction task. For example, Friborg et al. [15] use a GPU-accelerated particle filter to fit a surface mesh consisting of rigidly connected generalized cylinders to stereo depth data. However, even with GPU implementations, such approaches are often not real-time capable. Pure *local optimization* strategies have also been explored, which are implemented as variants of the iterative closest points (ICP) method [6]. For example, Pekelny and Gotsman [35] simultaneously track and reconstruct the shape of limbs through depth images. Knoop et al. [24] show that a combination of ToF and stereo data enables full-body pose estimation at real-time frame rates. Also using ICP, Grest et al. [20] combine depth and silhouette data to capture articulated motion with ten degrees of freedom in real time. Although yielding high frame rates, such methods often fail due to noise and motion blur present in the depth data. In particular with fast motions, local optimization easily gets stuck in erroneous poses which are hard to recover from.

In order to yield a more robust tracking, many approaches stabilize the optimization algorithms using additional *prior knowledge*. For example, Schwarz et al. [44] include a database of motions as prior knowledge for a particle filter optimization method. However, motions not present in the database cannot be tracked. As a complementary technique for stabilization, many approaches *detect features in a bottom-up fashion* directly from the depth input data. Here, geodesic distances are used in some work in order to detect anatomic landmarks. Following such a scheme, Ganapathi et al. [18] classify geodesic extrema features extracted from depth images according to the class labels 'hand', 'foot', and 'head'. With these detections, the search space of a particle filter is constrained. Integrating constraints into similar optimization techniques, anatomical landmarks are identified using feature tracking or heuristics in [1, 28, 49]. Using object detectors to estimate the position of the head and the hands, Gall et al. [16] stabilize a local optimization-based algorithm for tracking the upper body from depth data. Also, constrained inverse kinematics has been used on anatomical landmarks in [45, 57]. In our approach, we also make use of bottom-up detected features in order to stabilize a local optimization approach. As for the feature extraction, we build on the idea of accumulative geodesic extrema [36] and contribute with an efficient feature computation strategy.

Depth cameras seem to be an ideal type of sensor to facilitate intuitive human computer interaction based on full-body motion input. Therefore, many approaches focus on achieving *real-time* performance and try to find efficient algorithms for the pose reconstruction task. Although efficiency is clearly one of the key aspects to make pose estimation applicable for home use, most approaches with a focus on robustness reach only interactive run times around 10 fps [18, 20, 57]. Only recently, methods have been published that perform robust pose estimation within just a couple of milliseconds per frame [19, 48], see also the previous chapter. Such approaches for pose estimation are interesting from a practical point of view since they leave enough CPU cycles free for applications or games that use the reconstructed pose as input. Exceeding the performance of most published methods, we can report nearly 100 fps for full body pose estimation. Apart from the methodology of combining discriminative and generative models, the key to our efficient and stable

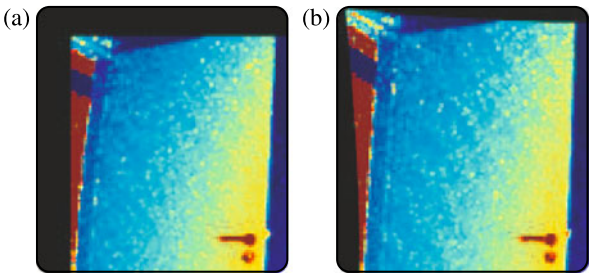

Fig. 5.2 Point cloud obtained by a ToF camera (**a**) without and (**b**) including a method for removing lens distortion effects. Note the straightening effect on the left edge of the door

pose estimation procedure is a compound of efficient feature computation, efficient database lookup, and an efficient selection strategy.

5.3 Acquisition and Data Preparation

5.3.1 Depth Data

Before ToF cameras and the Kinect sensor became popular, stereo cameras were predominantly used to obtain depth data in real time. In order to compute depth values, such passive stereo systems need to identify corresponding features in the two images captured at every time step. However, computing and matching such features is computationally expensive and often fails for objects without texture.

Current depth cameras based on active illumination with infrared light overcome these limitations. Moreover, current ToF cameras are robust to background illumination and yield stable distance values independent of the observed textures. In principle, ToF cameras capture depth/distance data at video frame rates by measuring the round trip time of infrared light emitted into and reflected from the scene. Several successive measurements have to be made in order to estimate the phase shift of the infrared light from which the round trip time is derived [25]. Moreover, further measurements are taken over a longer period of time in order to reduce the amount of noise. For static scenes, this process leads to measurements with high accuracies in the range of millimeters. For dynamic scenes with moving objects, however, this process can lead to errors in the estimation of depth values. Problematic are edges that separate an object from another, more distant object, resulting in strongly corrupted depth measurements, also called *mixed pixels*. Furthermore, low resolution, strong random noise and a systematic bias [25] lead to data that is difficult to handle.

A depth camera returns a distance image $I := \mathbb{Z}^2 \rightarrow \mathbb{R}$ with $\mathbb{Z}^2$ being the pixel domain. Since the camera also produces an amplitude image in the infrared domain, we use a standard pattern-based camera calibration [29] to recover the camera matrix and parameters for the lens distortion. To remove lens distortion effects, we apply the method of [23] which yields stable and accurate metric distance values, see Fig. 5.2 as an example. We do not calibrate for systematic bias of the camera,

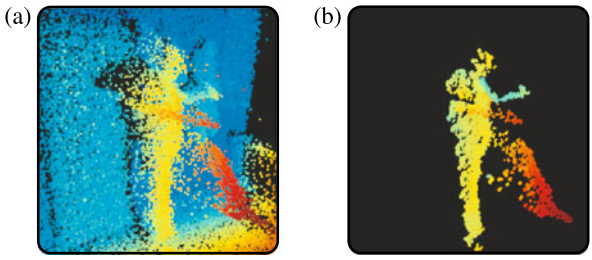

Fig. 5.3 (**a**) Original depth point cloud. (**b**) Point cloud after background subtraction and filtering. Some mixed pixel artifacts remain, see, e.g., the left leg

since for full-body pose estimation slight constant deviations in the measurements do not play an important role. For a recent method that calibrates for systematic bias using an intensity-based approach we refer to [27]. Using the calibration information, we transform the per-pixel distances into a metric 3D point cloud $\mathcal{M}_I \subseteq \mathbb{R}^3$ for every input frame of our online pose estimation framework, see Fig. 5.3(a). We then perform background subtraction using a static prerecorded background model and delete contour pixels to remove the influence of mixed pixels. Finally, a 3×3 median filter is used to reduce noise in the measurements, see Fig. 5.3(b).

In contrast to a ToF camera, the Microsoft Kinect depth sensor uses an active stereo approach. More specifically, a camera records an image of a projected structured light pattern in the infrared domain. Then, from the recorded pattern, a depth map is derived. In contrast to a ToF camera, only one image is analyzed in every time step. Thus, the Kinect camera is less susceptible to mixed pixels in dynamic scenes. However, the data also exhibits significant noise. In particular, artifacts like holes in the data appear when the projected pattern cannot be recognized. Moreover, the coarse depth quantization limits the accuracy in the far field from the camera, where, for example, a 2.5 cm quantization gap occurs at 3 meters distance to the camera. The presented algorithms in this chapter have been applied to depth data coming from ToF cameras as well as data coming from the Microsoft Kinect. Without changing or tuning the proposed algorithms, the final pose estimates with each of the cameras are qualitatively very similar as shown in the accompanying video [2].

5.3.2 Model of the Actor

The body of the actor is modeled as a kinematic chain [32]. We use $J = 20$ joints that are connected by rigid bones, where one distinguished joint is defined as the root of the kinematic chain, see Fig. 5.4(a). A pose is fully determined by the configuration of a kinematic chain specified by a pose parameter vector χ containing the position and orientation of the root joint as well as a set of joint angles. Through forward kinematics [32] using χ, 3D joint positions represented by a stacked vector $P_\chi \in \mathbb{R}^{3J \times 1}$ can be computed. Using linear blend skinning [26], we attach a surface mesh with a set of 1170 vertices $\mathcal{M}_\chi$ to the kinematic chain to model the deforming body geometry, see Fig. 5.4(b). Initializing the body model to the shape of a specific actor is beyond the scope of this chapter. Methods exist to solve this task using

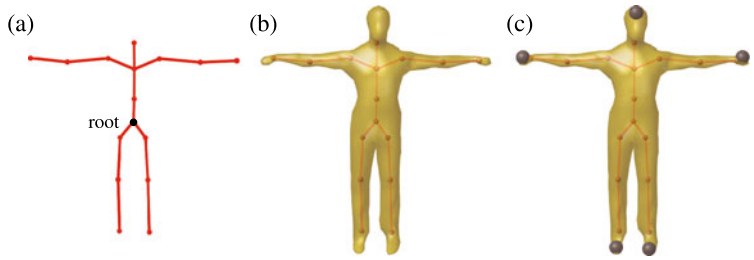

Fig. 5.4 (**a**) Skeletal kinematic chain. (**b**) Rigged mesh. (**c**) Highlighted end effectors (hands, feet, and head)

image data and a large database of scanned humans, see, e.g., [21, 22]. Recently, Weiss et al. [54] have shown that the body shape of a person can be determined using depth images from four different views, see also the following chapter for details. As shown in our experiments (Sect. 5.5), even with a fixed body model we can track people for a range of different body sizes.

5.3.3 Pose Database

Pose reconstruction approaches based on local optimization update the model parameters (in our case the joint angles) by optimizing a specified cost function, where convergence only to a near local minimum can be guaranteed. Although such methods typically run very fast, they fail when the initialization is too far away from the actual pose. For such a failure case we say that the algorithm loses track. This is often the case for fast motions where body parts can move far from frame to frame. One strategy to overcome such limitations is to reinitialize the local optimization when the track is lost. In the proposed algorithm, we use global pose estimates derived from database knowledge for such reinitializations. To this end, we create a database of human full body poses obtained with a marker-based motion capture system. The actor performs a variety of motions including hand gestures and foot motions to span a large range of different poses. To enable invariance under global transformations, the obtained poses χ_i are then normalized according to the positions of the root joint and the viewing direction. Furthermore, to maximize the variety and minimize the number of poses in the database, we select a subset of the recorded poses using a greedy sampling algorithm [52]. In this algorithm, the distance of two poses specified by χ_1 and χ_2 is measured by taking the distance of the corresponding joint positions into account:

$$d_P(\chi_1, \chi_2) := \frac{1}{J} \cdot \|P_{\chi_1} - P_{\chi_2}\|_2. \tag{5.1}$$

In contrast to [52], we truncate the sampling as soon as the minimal distance between all pairs of selected poses reaches a certain threshold. Using the truncated

sampling, we obtain roughly 25 000 poses in which any two selected poses have a pose distance d_P larger than 1.8 cm. For each selected pose, we then consider end effector positions of the left/right hand, the left/right foot, and the head, modeled as $E_\chi^5 := (e_\chi^1, \ldots, e_\chi^5) \in (\mathcal{M}_\chi)^5$, see Fig. 5.4(c).

The following three reasons motivate the use of end effector positions as features. Firstly, end effector positions can be efficiently estimated for a large set of different poses even from depth data alone, see Sect. 5.4.2. Secondly, for many poses these positions are characteristic, thus yielding a suitable representation for cutting down the search space. Thirdly, they lead to low-dimensional feature vectors which facilitates the usage of efficient indexing methods. Thus, end effector positions constitute a suitable mid-level representation for full-body poses that on the one hand abstract away most of the details of the noisy input data, yet on the other hand retain the discriminative power needed to cut down the search space in the pose estimation procedure.

For indexing, we use a kd-tree [11] on the 15-dimensional stacked vectors E_χ^5 since they provide logarithmic search time in the size of the database and have turned out to be an efficient search structure for low-dimensional feature vectors. Since the size of the skeleton (e.g., body height or arm span) varies with different actors, the pose database has to be adapted to the actor. While not implemented in the presented system, this task can be solved using a retargeting framework. Even without retargeting, by manipulating the depth input point cloud $\mathcal{M}_I$ we are able to track motions of people if body proportions are not too far off the database skeleton, see Sect. 5.5.

5.3.4 Normalization

In the proposed tracking framework, we allow the actor to move freely within the field of view of the camera, while we restrict variations of the viewing direction to the range of about $\pm 45°$ rotation around the vertical axis with respect to the frontal viewing direction. Recall that in our database all poses have been normalized with regard to the position of the root joint and the viewing direction. Thus, in order to query the database in a semantically meaningful way, we need to cope with variations in global position and orientation of the actor. We normalize $\mathcal{M}_I$ with respect to global position by means of a 3D ellipsoid fit around $\mathcal{M}_I$ using a mean-shift algorithm similar to [52]. To cope with global rotations, one could augment the database to contain pose representations from several viewing directions [12, 46, 52]. In this case, the retrieval time as well as the risk of obtaining unwanted poses would increase. Instead, in our framework, we normalize the depth input point cloud according to an estimated viewing direction. To this end, we compute a least-squares plane fit to points corresponding to the torso, which we assume to be the points that are closer than 0.15 m to the center of $\mathcal{M}_I$, see Fig. 5.5. The normal to the plane, as indicated by the cyan arrow in Fig. 5.5, corresponds to the eigen-vector with the smallest eigen-value of the covariance matrix of the points. The

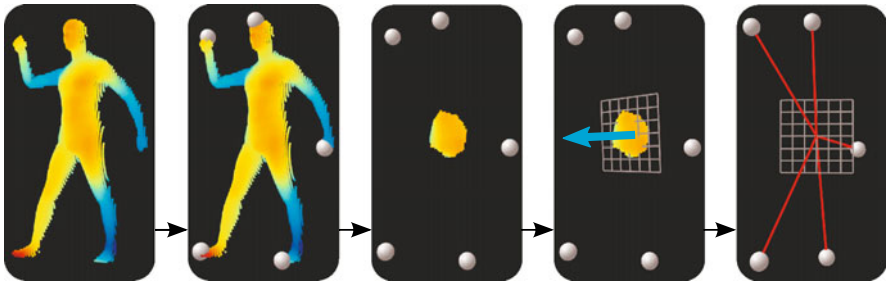

Fig. 5.5 Normalization of the geodesic extrema with respect to a computed viewing direction

viewing direction is its projection onto an imagined horizontal ground plane. We then rotate the positions of the geodesic extrema about the vertical axis through the center such that the normal of the rotated plane points to front. To cope with frames in which the viewing direction cannot be estimated because, e.g., the torso is occluded, we adaptively smooth the estimated directions over time. We detect whether the torso is occluded by inspecting the eigen-values of the above mentioned covariance matrix. Here, occluding body parts often lead to a stronger curvature in the regarded points (smallest eigen-value is relatively large) or a less circular fit (largest eigen-values are not similar). Then, we minimize the influence of the estimated normal. As a consequence, the detected viewing direction remains stable even if the arms occlude the torso or the center of $\mathcal{M}_I$ does not correspond to the torso. The proposed strategy might yield inaccurate viewing directions for persons with a very roundish belly, where the regarded points might always posses a strong curvature and thus the influence of the estimated normals might be continuously down-weighted. Moreover, the depth data of a side view of such a belly might appear similar to the depth data of a front view which could lead to invalid normals.

5.4 Pose Reconstruction Framework

As explained in the previous section, in the offline preprocessing phase, the camera matrix is obtained and the background model is created. We now describe our proposed online framework, see also Fig. 5.1. At a given frame t, the first steps are to compute the point cloud $\mathcal{M}_I$ from the distance image I, to perform background subtraction, to filter out noise and to normalize according to the viewing direction. Let χ^*_{t-1} be the final pose estimate of the previous frame $t-1$. From χ^*_{t-1}, we obtain a pose hypothesis χ_t^{LocOpt} by refining χ^*_{t-1} with a local optimization procedure that takes the input depth data into account (Sect. 5.4.1). A second pose hypothesis is obtained as follows. We extract a 15-dimensional feature vector from $\mathcal{M}_I$, representing the 3D coordinates of the first five geodesic extrema (Sect. 5.4.2). Being a low-dimensional yet characteristic pose representation, the features permit rapid retrieval of similar full-body poses from a large pose database (Sect. 5.4.3). From the set of retrieved poses we choose a single pose hypothesis χ_t^{DB} using a distance

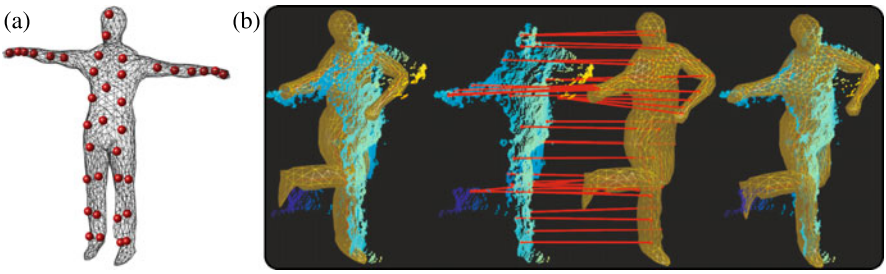

Fig. 5.6 (**a**) Subset of vertices $\mathcal{C}_\chi \subseteq \mathcal{M}_\chi$. (**b**) From pose χ (*left*), correspondences for mesh vertices in $\mathcal{C}_\chi$ are estimated (*middle*). Local optimization using the correspondences yields an updated pose χ' (*right*)

function which takes the influence of the estimated pose of the previous frame χ^*_{t-1} into account. Based on a late-fusion selection scheme that combines two sparse distances measures, our algorithm decides between χ_t^{DB} and χ_t^{LocOpt} to find the final pose χ_t^*, see Sect. 5.4.4.

5.4.1 Local Optimization

In our local pose optimization, we follow a standard procedure as described in, e.g., [42]. Here, the goal is to modify an initial pose χ such that the modified pose χ' fits to the point cloud $\mathcal{M}_I$ more accurately. To this end, we seek correspondences between vertices in $\mathcal{M}_\chi$ and points in $\mathcal{M}_I$.

Finding correspondences for all $v \in \mathcal{M}_\chi$ is not meaningful for three reasons. Firstly, many vertices do not have semantically meaningful correspondences in $\mathcal{M}_I$, e.g., the vertices of the back of the person. Secondly, the number of correspondences for the torso would be much higher than the number of correspondences in the limbs, which would disregard the importance of the limbs for pose estimation. Thirdly, the computation time of local optimization increases with the number of correspondences.

Therefore, we use a predefined set $\mathcal{C}_\chi \subseteq \mathcal{M}_\chi$ of mesh vertices as defined in Fig. 5.6(a). Here, we make sure that we select a couple of vertices for each body part. Using a local kd-tree built up in every frame, we efficiently obtain the ℓ nearest neighbors in $\mathcal{M}_I$ of each vertex $v \in \mathcal{C}_\chi$ and claim correspondence of v to the median of its ℓ nearest neighbors to reduce the influence of noise. Using these correspondences, we obtain updated pose parameters χ' by applying an optimization framework similar to the one in [42].

5.4.2 Feature Computation

To obtain a sparse yet expressive feature representation for the input point cloud $\mathcal{M}_I$, we revert to the concept of geodesic extrema as introduced in [36]. Such ex-

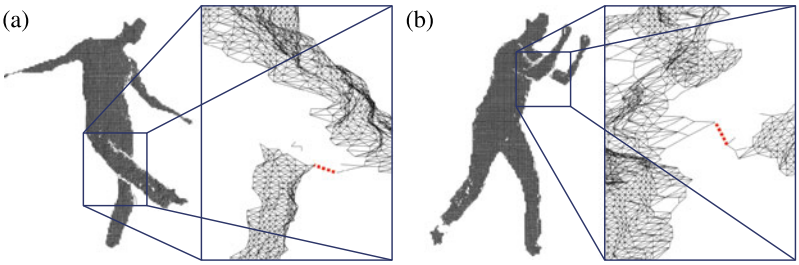

Fig. 5.7 Graph obtained from the depth image (*black lines*) and a zoom-in from a more lateral viewpoint for two poses with self-occlusions. The initially disconnected graph is automatically connected using edges indicated by the *red dashed lines*, respectively

trema often correspond to end effector positions, yielding characteristic features for many poses as indicated by Fig. 5.10. Following [36], we now summarize how one obtains such features. Furthermore, we introduce a novel variant of Dijkstra's algorithm that allows for efficiently computing a large number of geodesic extrema. We model the tuple of the first n geodesic extremal points as

$$E_I^n := \left(e_I^1, \ldots, e_I^n\right) \in (\mathcal{M}_I)^n. \tag{5.2}$$

To compute E_I^n, the point cloud data is modeled as a weighted graph where each point in $\{p_1, \ldots, p_L\} := \mathcal{M}_I$ represents a node in the graph. We refer to a node by its index $\ell \in [1 : L]$. To efficiently build up the edge structure of the graph, we exploit the neighborhood structure in the pixel domain $\mathbb{Z}^2$ of the underlying distance image. For a given $p_\ell \in \mathcal{M}_I$, we consider the 8-neighborhood in the domain of the underlying image. For each such neighboring pixel $p_m \in \mathcal{M}_I$, we add an edge between m and ℓ of weight $w = \|p_m - p_\ell\|_2$ if w is less than a distance threshold τ. This way, we obtain a weighted edge structure in form of an adjacency list

$$\mathcal{E}(\ell) := \left\{(m, w) \in [1 : L] \times \mathbb{R}_+ \mid p_m \text{ and } p_\ell \text{ share an edge of weight } w\right\} \tag{5.3}$$

for $\ell \in [1 : L]$. Here, note that when building up the edge structure, the distance between any two points in $\mathcal{M}_I$ has to be evaluated only once.

In our approach, in contrast to the method in [36], we need to obtain a fully connected graph with only one connected component in order to obtain meaningful geodesic extrema. In practice, however, the graph computed as described above is not fully connected if, for example, the depth sensor misses parts of the thin limbs, or due to occlusions, see Fig. 5.7. To cope with such situations, we use an efficient union-find algorithm [47] in order to compute the connected components. To reduce small artifacts and noise pixels, we discard all components that occupy a low number of nodes. Furthermore, we assume that the torso is the component with the largest number of nodes. All remaining components are then connected to the torso by adding an edge between the respective closest pair of pixels if the edge weight is less than 0.5 m, see the red dotted lines in Fig. 5.7. This allows us to find meaningful geodesic extrema even if the initial graph splits into separate connected

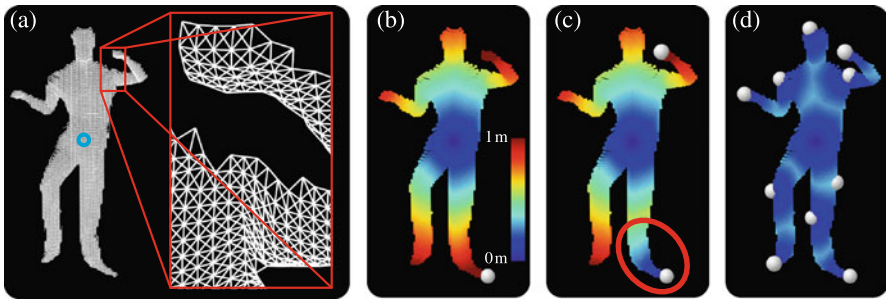

Fig. 5.8 Computation of geodesic extrema using a variant of Dijkstra's algorithm. (**a**) Graph structure and source node (*cyan circle*). (**b**) Geodesic distances and first geodesic extremum. (**c**) Updated geodesic distances and second geodesic extremum. (**d**) The first ten geodesic extrema

components, see Fig. 5.10(b)–(h) for the resulting geodesic extrema of the graphs shown in Fig. 5.7. Still, the estimated connections might lead to unwanted geodesic extrema in some poses. In this case, the database lookup (as will be explained in Sect. 5.4.3) might yield an inaccurate pose hypothesis. However, the influence of such a pose on the final pose estimate will be minimized by means of a hypothesis selection, as will be explained in Sect. 5.4.4.

We now show how a large number of extrema can be computed efficiently. Basically, we follow an iterative computation strategy. In each iteration, we use Dijkstra's algorithm [11] to compute the geodesic distances from a centroid node ℓ_0 (referred to as *source node*) to all other nodes in the graph. We then pick the node with the maximal distance as the corresponding extremal point. The efficiency of our algorithm is based on the observation that only in the first iteration of our algorithm, a full pass of Dijkstra has to be computed. In all remaining iterations one needs to consider only a small fraction of the nodes. As another observation, we only need to obtain the geodesic distances of each node and do not need to store the actual shortest path information which is usually saved in a predecessor array in Dijkstra's algorithm [11]. Therefore, we save additional time in each iteration by omitting the predecessor array.

As input to Algorithm 1, we use the graph structure with nodes, edges, and the designated source node ℓ_0, see Fig. 5.8(a). Additionally, we use a priority queue Q that stores tuples $(m, w) \in [1 : L] \times \mathbb{R}_+$ of nodes and weights sorted by increasing weight. The priority queue allows us to extract the tuple with the minimal weight by the Q.getMin() operation. To keep track of the distance values of each node, we use an auxiliary array Δ having L entries.

We start the algorithm by initializing Δ, see Lines 1–3. Then, we insert the source node into the previously empty priority queue Q in Line 4. We then iterate over all geodesic extrema to be computed. The first pass of Dijkstra (Lines 6 to 13) stores the shortest geodesic distances from the source node to any other node in the graph in the array Δ, see Fig. 5.8(b). Then, the point corresponding to the node ℓ^* with the largest distance in Δ is taken as the first geodesic extremum e_I^1 (Lines 14 to 15). Note that if there are still nodes which are not reachable from the source node ℓ_0,

Algorithm 1 Geodesic Extrema Computation

Input: $\{p_1 \dots p_L\} := \mathcal{M}_I$: 3D point cloud with points $p_\ell \in \mathbb{R}^3$ and nodes $\ell \in [1:L]$
$\mathcal{E}(\ell) := \{(m, w) \in [1:L] \times \mathbb{R} | p_m$ and p_ℓ share an edge of weight $w\}$:
edge adjacency list defined on $\mathcal{M}_I$
$\ell_0 \in [1:L]$: index of the designated source node
Q: priority queue for elements $(m, w) \in [1:L] \times \mathbb{R}$
n: number of geodesic extrema to be computed
Output: $(e_I^1, \dots, e_I^n) \in (\mathcal{M}_I)^n$: the n first geodesic extrema of G

1: **for** $\ell \leftarrow 1$ **to** L **do**
2: $\Delta[\ell] \leftarrow \infty$ ▷ Initialize distance array
3: $\Delta[\ell_0] \leftarrow 0$ ▷ Distance to source
4: Q.insert$((\ell_0, 0))$
5: **for** $i \leftarrow 1$ **to** n **do** ▷ Compute the first n extrema
6: **while** $Q \neq \emptyset$ **do**
7: $\ell \leftarrow Q$.getMin() ▷ Get entry with minimal weight
8: Q.removeMin() ▷ Remove the entry from Q
9: **for each** $(m, w) \in \mathcal{E}(\ell)$ **do** ▷ For all nodes connected by an edge to p_ℓ
10: **if** $\Delta[\ell] + w < \Delta[m]$ **then**
11: $\Delta[m] = \Delta[\ell] + w$ ▷ A shorter path has been found
12: Q.insert$((m, \Delta[m]))$
13: **end while** ▷ Δ now contains the geodesic distances
14: $\ell^* \leftarrow \arg\max_{\ell \in [1:L]} \Delta[\ell]$ ▷ Note: the arg max must ignore nodes that were not reached
15: $e_I^i \leftarrow p_{\ell^*}$ ▷ Store ith extremum
16: $\Delta[\ell^*] \leftarrow 0$ ▷ Simulates edge insertion between p_{ℓ_0} and p_{ℓ^*}
17: Q.insert$((\ell^*, 0))$ ▷ Let ℓ^* act as new source
18: **end for**

they bear the same distance values ∞ as set in the initialization. Of course, such unreachable nodes should not be considered as geodesic extrema. Therefore, the arg max operator in Line 14 must ignore these nodes in order to recover the true geodesic extremum. In Fig. 5.8(b), the detected extremum e_I^1 is indicated by the gray sphere on the left foot. According to [36], the next step is to add a zero-cost edge between ℓ_0 and ℓ^* and then to restart Dijkstra's algorithm to find e_I^2, and so on. This leads to a run time of $O(n \cdot D)$ for n extrema with D being the run time of Dijkstra's algorithm for the full graph.

Note that the second run of Dijkstra's algorithm shows a high amount of redundancy: the entries in the array Δ corresponding to all nodes in the graph that are geodesically closer to ℓ_0 than to the node of e_I^1 will not change in the second run. For example, in Fig. 5.8(c), only the distance values of the nodes within the highlighted area have changed.

Therefore, to compute the second pass, we keep the distance values of the first pass and let the node ℓ^* corresponding to e_I^1 act as the new source, see Lines 16 and 17. This way, the second iteration will be by an order of magnitude faster than

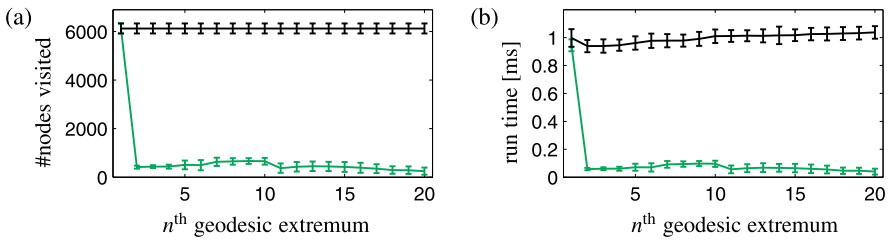

Fig. 5.9 (**a**) Number of nodes visited and (**b**) run time in milliseconds to find the nth geodesic extremal point for the baseline (*black*) and our optimized algorithm (*green*). Average values and standard deviation bars for a sequence of 400 frames from the data set of [18] are reported

the first iteration, as also confirmed by our experiments described in the subsequent paragraphs. Using ℓ^* as the new source, we update Δ with a pass of Dijkstra's algorithm and pick e_I^2 as the point with the maximal distance in the updated Δ, see Fig. 5.8(c). For the third pass we let the node corresponding to e_I^2 act as the new source by setting the corresponding value in Δ to 0, and run Dijkstra again. This way, in the third pass, only nodes in the graph that are closer to the node of e_I^2 than to all other previously used source nodes are touched. We proceed iteratively to compute the subsequent extremal points, see Fig. 5.8(d) for the resulting distance values and extrema after 10 iterations.

Our computational strategy leads to drastic improvements in the run time for each pass. To experimentally verify this, we evaluated the algorithm on a depth input sequence of 400 frames taken from the data set of [18]. We computed the first 20 geodesic extrema for each of the 400 frames using both a baseline algorithm that runs a full Dijkstra pass in each iteration and our optimized algorithm. We traced the number of nodes visited in each iteration as well as the actual run time for each iteration. Figure 5.9 shows that in the first iteration all reachable nodes in $\mathcal{M}_I$ (on average there were more than 6000 nodes in the graph) were visited. In the second iteration, only 413 ± 61 nodes (average $\pm$ standard deviation over all frames) were visited. This substantial reduction is also reflected by the run time of the algorithm, which drops from 1 millisecond in the first iteration to about 0.058 ± 0.0085 milliseconds in the second iteration, see Fig. 5.9(b). As a result, the overall run time for computing the first 20 geodesic extrema is only slightly higher than the run time of the original Dijkstra algorithm for computing the first geodesic extremum. Thus, the algorithm allows us to efficiently compute a large number of geodesic extrema.

The overall approach enables the detection of semantically meaningful end effector positions even in difficult scenarios. Figure 5.10 shows a number of challenging examples, where legs occlude each other (b)–(c), multiple body parts occlude each other (d)–(f), a fast punching motion with occlusions is performed (g)–(k), a leg is bent to the back (l), and the hands are outstretched to the camera (m). However, in poses where the end effectors are very close to other parts of the body, the topology of the graph may change and the detected extrema may differ from the actual set of end effectors, see Fig. 5.11(a)–(c). In these poses, the left elbow, the left shoulder,

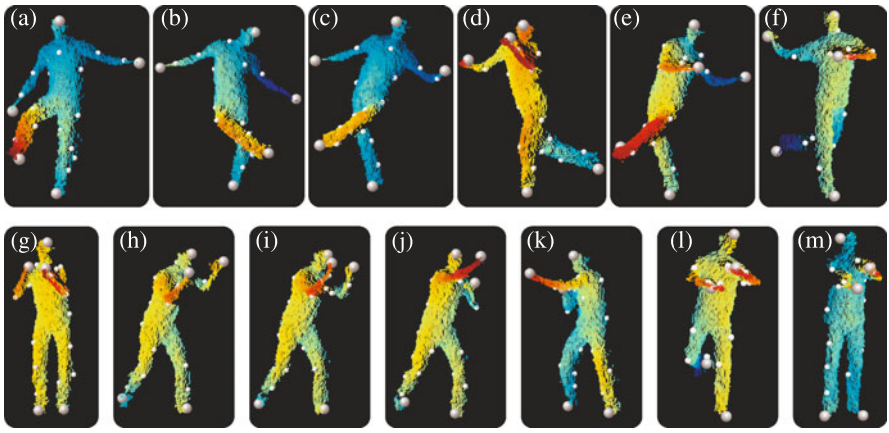

Fig. 5.10 Example poses with the first five geodesic extrema drawn as big spheres, and extrema 6 to 20 drawn as smaller blobs. For many poses, the first five extrema correspond to the end effectors, even when self-occlusions are present.

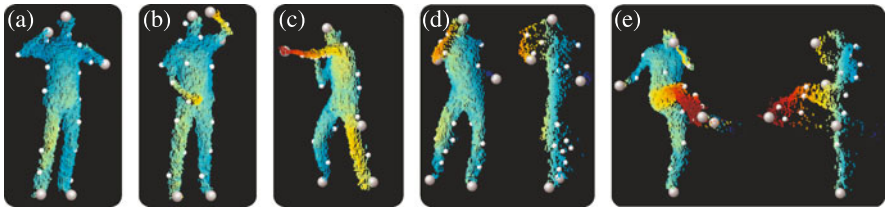

Fig. 5.11 (**a**)–(**c**) Problematic example poses. In particular when the hands come close to the body, end effector detection becomes difficult. (**d**)–(**e**) Flying mixed pixels lead to deviations in the end effector detection

and the left hip are selected as e_I^5, respectively. Also, flying mixed pixels can cause the topology of the graph to change, as depicted in Fig. 5.11(d), where we show a pose once from a frontal view and once from a side view. Note that although the left hand keeps a reasonable distance from the head, mixed pixels build a bridge in the graph from the hand to the head. Thus, the fifth extremum is located at the elbow. Figure 5.11(e) shows a similar situation in which the head is not detected due to mixed pixels. Instead, the fifth extremum is located at the hip.

In the subsequent section, we will explain the discriminative component of our framework, where pose candidates are obtained from the database by using the positions of the first five geodesic extrema as a query. If the end effectors are not revealed by these extrema, however, the obtained pose candidates are often meaningless. As will be explained later, the influence of such meaningless poses on the final pose estimates can be minimized with our combined generative and discriminative framework.

5.4.3 Database Lookup

As for the database lookup component, the goal is to identify a suitable full-body pose χ_t^{DB} from our pose database using the extracted geodesic extrema E_I^5 as the query input. However, as opposed to the database motions where the semantics of the end effector positions are known, the semantic labels of the extrema on the query side are not known. To partially solve for missing semantics, the method [18] uses a classifier trained on 'hand', 'head', and 'foot' patches of distance images. This process, however, is relatively expensive (taking 60 ms per frame according to [36]) and is thus not directly suitable in real-time scenarios. Also, when using depth data alone, misclassification of patches might occur because of missing color information, strong noise, and the low resolution of the measurements. In order to circumvent the classification problem, we propose a query strategy that does not rely on having a-priori semantic labels for the extracted geodesic extrema. Intuitively speaking, we use different queries that reflect different label assignments. As explained in the following, from the retrieved poses, we then choose a candidate pose that most likely corresponds to the correct labeling.

Let $\mathbb{S}_5$ be the symmetric group of all five-permutations. For a permutation σ and a five-tuple E, we denote the permuted tuple by σE. Now, let $\mathcal{S} \subseteq \mathbb{S}_5$ be a subset containing permutations σ such that the positions in σE_I^5 are close to the end effectors of the previous frame χ_{t-1}^*. More specifically, we define

$$\mathcal{S} := \left\{ \sigma \in \mathbb{S}_5 \;\middle|\; \forall n \in [1:5] : \left\| e_I^{\sigma(n)} - e_{\chi_{t-1}^*}^n \right\| < \mu \right\}. \tag{5.4}$$

In our experiments, we use a distance threshold of $\mu = 0.5$ meters to effectively and conservatively prune the search space while still allowing for large jumps in the end effector positions which may be present in fast motions. In frames with clear geodesic extrema, the number of considered permutations $|\mathcal{S}|$ typically drops to one. To further increase robustness to false estimations in the previous frame, we add additional permutations to $\mathcal{S}$ if we detect jumps in the positions of the geodesic extrema. To compute the additional permutations, we assume that the two lowest extrema w.r.t. the vertical axis, say e_I^1 and e_I^2, correspond to the feet. This leads to two possible label assignments where the label 'left foot' is assigned to either e_I^1 or e_I^2. For each of the two assignments, the remaining three extrema can receive $3! = 6$ different labellings. This leads to $2 \cdot 6 = 12$ additional permutations added to $\mathcal{S}$.

By querying the kd-tree of the pose database for K nearest neighbors for each permutation in $\mathcal{S}$, we obtain $K \cdot |\mathcal{S}|$ pose candidates $\chi_{k,\sigma}$ with $k \in [1:K]$ and $\sigma \in \mathcal{S}$. For each (k, σ), we define a distance value between the pose candidate $\chi_{k,\sigma}$ and the permuted E_I^5 by

$$\delta\left(\chi_{k,\sigma}, E_I^5\right) := \frac{1}{5} \cdot \| E_{\chi_{k,\sigma}} - \sigma E_I^5 \|_2. \tag{5.5}$$

Note that to compute the distance $\delta(\chi_{k,\sigma}, E_I^5)$, we stack the tuples $E_{\chi_{k,\sigma}}$ and σE_I^5 into 15-dimensional vectors, respectively. The result of the database lookup χ_{k^*,σ^*}

for frame t is then chosen by also considering temporal consistency using

$$\left(k^*, \sigma^*\right) = \underset{(k,\sigma)}{\arg\min}\, \lambda \cdot \delta\left(\chi_{k,\sigma}, E_I^5\right) + (1-\lambda) \cdot d_P\left(\chi_{k,\sigma}, \chi_{t-1}^*\right) \tag{5.6}$$

with a weighting factor λ that balances out the influence of d_P (defined in Eq. 5.1) and δ. In our experiments, we use $\lambda = 0.5$. Finally, we refine χ_{k^*,σ^*} to the hypothesis χ_t^{DB} using local optimization as described in Sect. 5.4.1.

5.4.4 Hypothesis Selection

At this stage, two alternative pose hypotheses have been derived, namely χ_t^{LocOpt} from the generative and χ_t^{DB} from the discriminative component. In the next step, we need to create a single, final pose χ_t^* taking both hypotheses into account. Recall that the pose hypothesis χ_t^{DB} might be inaccurate when the end effectors are not revealed. Therefore, it is not meaningful to take the average pose of χ_t^{LocOpt} and χ_t^{DB} as final pose. Instead, for this late-fusion step, we propose a novel selection scheme that decides for either χ_t^{LocOpt} or χ_t^{DB} as the final pose χ_t^* based on an efficiently computable sparse and symmetric distance measure. With the proposed selection strategy, the local optimization and database lookup schemes benefit from each other. On the one hand, if the database lookup component fails, then the local optimization component can continue to track the motion. On the other hand, the local optimization might fail to track fast and abrupt motions. In such situations, the database lookup can reinitialize the tracking.

In the proposed selection scheme, we want to avoid a dominant influence of potential errors coming from the feature extraction or from the database lookup. Therefore, we use distance measures that revert to the original input point cloud $\mathcal{M}_I$ rather than to derived data. One possible distance measure could be defined by projecting $\mathcal{M}_\chi$ into a synthetic distance image and comparing it to I. In practice, however, because of the relatively low number of pixels in the thin limbs, such a distance measure is dominated by the torso. For this reason, we propose a novel distance metric that can be computed efficiently and accounts for the importance of the limbs for pose estimation.

To this end, we combine two sparse distances measures. The first distance expresses how well the mesh is explained by the input data:

$$d_{\mathcal{M}\chi \to \mathcal{M}_I} := \frac{1}{|C_\chi|} \sum_{v \in \mathcal{C}_\chi} \min_{p \in \mathcal{M}_I} \|p - v\|_2. \tag{5.7}$$

Here, we revert to only the subset $\mathcal{C}_\chi \subseteq \mathcal{M}_\chi$ of vertices as defined in Sect. 5.4.1. Likewise, the second distance measure expresses how well $\mathcal{M}_I$ is explained by $\mathcal{M}_\chi$:

$$d_{\mathcal{M}_I \to \mathcal{M}_\chi} := \frac{1}{20} \sum_{n \in [1:20]} \min_{v \in \mathcal{M}_\chi} \left\|e_I^n - v\right\|_2. \tag{5.8}$$

To emphasize the importance of the limbs, we take only the first 20 geodesic extrema of the input depth data, which largely correspond to points on the limbs rather than the torso, see Fig. 5.10. Since also for the mesh we take only a subset of vertices, see Fig. 5.6(a), the distance measures are sparse. Both distance measures can be computed efficiently because firstly, geodesic extrema can be extracted very efficiently (Sect. 5.4.2), and secondly, only a small number of points are taken into account. The final pose χ_t^* is then given through

$$\chi_t^* := \underset{\chi \in \{\chi_t^{\mathrm{DB}},\, \chi_t^{\mathrm{LocOpt}}\}}{\arg\min} \; (d_{\mathcal{M}_\chi \to \mathcal{M}_I} + d_{\mathcal{M}_I \to \mathcal{M}_\chi}). \tag{5.9}$$

5.5 Experiments

We implemented the proposed hybrid tracking strategy in C++ and ran our experiments on a standard off-the-shelf desktop PC with a 2.6 GHz CPU. To numerically evaluate and to compare our hybrid strategy with previous work, we used the publicly available benchmark data set of [18]. In this data set, 28 sequences of ToF data (obtained from a Mesa Imaging SwissRanger SR 4000 ToF camera) aligned with ground truth marker positions (obtained from a marker-based motion capture system) are provided. This data set comprises 7900 frames in total. In addition to numerically evaluating on this data set, we demonstrate the effectiveness of the proposed algorithm in a real-time scenario with fast and complex motions captured from a PMD Camcube 2 in a natural and unconstrained environment, see Fig. 5.13 and Fig. 5.14. In the accompanying video [2], we show that the same framework also works with the Microsoft Kinect depth sensor without any further adjustments.

5.5.1 Feature Extraction

First, we evaluate the effectiveness of the proposed feature extractor on the benchmark data set. Not all ground truth markers in all frames are visible, thus, for this evaluation, we use only the 3992 frames in which all five end effector markers are visible. A good recognition performance of the feature extractor is needed for a successful subsequent database lookup. In 86.1 % of the 3992 frames, each of the found five geodesic extrema E_I^5 is less than 0.2 meters away from its corresponding ground truth marker position. This shows that we can effectively detect the end effector positions for most motions contained in the test data set.

5.5.2 Quantitative Evaluation

We run our pose reconstruction algorithm on the benchmark data set. Since the surface mesh of the actor is not part of that data set, we scale the input point cloud data

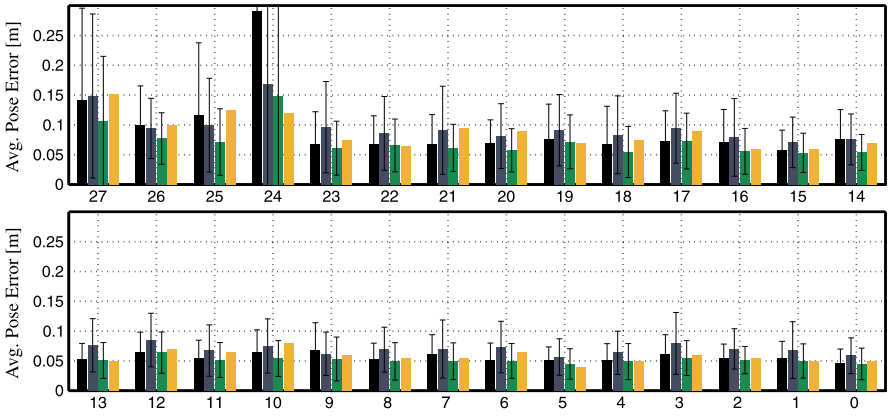

Fig. 5.12 Average pose error and standard deviation of sequences 27 to 0 of the data set of [18]. *Bars left to right*: Using only local optimization, using only the database lookup, results using the proposed fusion scheme, and values reported by [18] (without std. dev.)

so that it roughly fits the proportions of our actor. We manually fix correspondences between each motion capture marker and a mesh vertex. For a test sequence with T frames, let M_t be the number of visible motion capture markers in frame t, let $m_{t,i}$ be the 3D position of the ith visible marker in frame t and $\tilde{m}_{t,i}$ the position of the corresponding mesh vertex of the reconstructed pose. As also used in [18], the average pose error for a sequence is computed as

$$\bar{\epsilon}_{\text{avg}} := \frac{1}{\sum_{t=1}^{T} M_t} \sum_{t=1}^{T} \sum_{i=1}^{M_t} \|m_{t,i} - \tilde{m}_{t,i}\|_2. \tag{5.10}$$

Whereas the overall accuracy of the tracking algorithm is expressed by means of Eq. 5.10, potential local tracking errors can be averaged out. Therefore, we use this evaluation measure to show tendencies in the accuracy by comparing different pose estimation strategies for all benchmark sequences, see Fig. 5.12. To this end, we report how the local optimization component (Sect. 5.4.1) and the database lookup component (Sect. 5.4.3) perform individually, without being combined with the late-fusion hypothesis selection. When using only local optimization (first bar) the method often gets stuck in local minima and loses track. When using only a database lookup (second bar), poses where the end effectors are not revealed by the first five geodesic extrema may cause a false lookup result. Thus, in terms of the average pose error, both methods in isolation do not perform well on all sequences. The third bar shows the result of the proposed hybrid strategy which leads to substantial improvements. Also in comparison to [18] (last bar, std. dev. values were not available), we achieve comparable results for basic motions and perform significantly better in the more complex sequences 20 to 27. Only for sequence 24, the method [18] performs better than our approach. The reason for this is that this sequence contains a 360° rotation around the vertical axis, which cannot be han-

Table 5.1 Average run times in milliseconds over all frames of the benchmark data set

	Total	Preparation	Local optim.	E_I^{20}	Lookup	Selection
Full resolution	16.6 ms	1.2 ms	5.7 ms	6.2 ms	1.2 ms	0.9 ms
	100 %	7 %	34 %	37 %	7 %	5 %
Half resolution	10.0 ms	1.1 ms	4.6 ms	1.5 ms	1.2 ms	0.9 ms
	100 %	11 %	46 %	15 %	12 %	9 %

dled by our framework. However, our system can cope with rotations in the range of $\pm 45°$ since we normalize the input data based on the estimated viewing direction. For the benchmark data set, the hypothesis selection component decided in 22.5 % of the frames for the retrieval component, and in 77.5 % for the local optimization from the previous frame. With our hypothesis selection, we significantly reduced the average pose error of the final pose estimate in comparison to either method ran individually.

5.5.3 Run Time

In Table 5.1, we report the average run time of our pose estimation framework in milliseconds per frame. In [18], the authors report a performance of 4 fps on downsampled input data. By contrast, with our proposed algorithm, we achieve 60.4 fps (16.6 ms per frame) on average on the full resolution input data, and 100 fps (10.0 ms per frame) with half of the resolution which we track with nearly the same accuracy. The run times are in the same order of magnitude than other state-of-the-art approaches like [48] where the authors report 200 fps on a different dataset and "at least 10×" speedup with respect to [18]. As for a more detailed analysis, we also give the run time of each algorithmic component, namely the data preparation phase (Sect. 5.3), the local optimization component (Sect. 5.4.1), the feature extraction (Sect. 5.4.2), the database lookup (Sect. 5.4.3), and the selection (Sect. 5.4.4). Note that our efficient algorithms lead to run times that are well distributed among the different components, such that no clear bottleneck is present. For the full resolution, the run time of local optimization and the feature extraction are approximately the same. The latter benefits most from downsampling the data.

5.5.4 Qualitative Evaluation

In Fig. 5.13, we show example results of fast and complex motions captured in an unconstrained environment. The considered motions are much faster and contain more challenging poses than the ones used in [18]. The leftmost image in each subfigure shows a video frame of the pose captured from a separate video camera

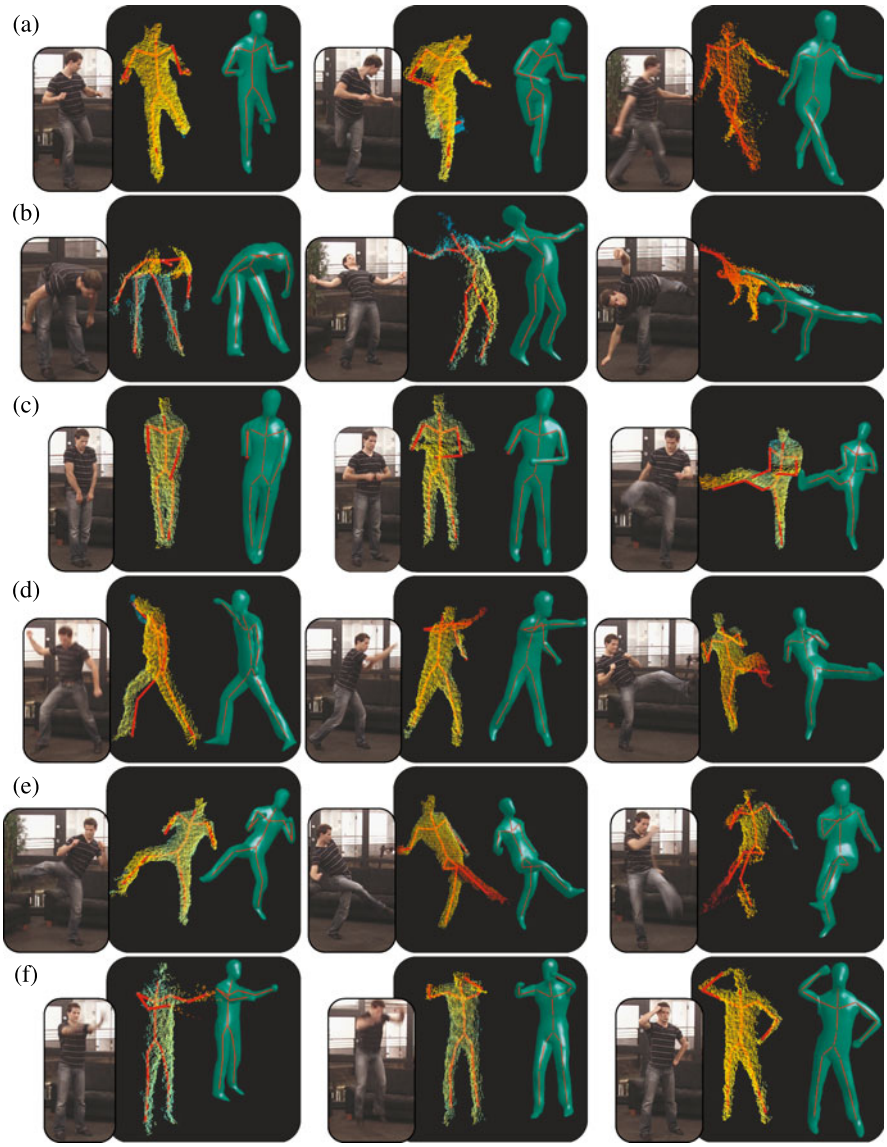

Fig. 5.13 Snapshots of our results on fast and complex motions on data captured with the PMD camera. For every motion we show a video frame of the actor (not used for tracking), ToF data overlaid with the reconstructed skeleton, and a rendering of the corresponding mesh

not used for tracking. In the room where we recorded the data, the video camera was standing to the left of the depth camera. The middle image shows the depth data overlaid with the estimated skeleton of the pose χ_t^*. The rightmost image depicts a rendering of the surface mesh in the corresponding pose.

The first row (Fig. 5.13(a)) shows some frames of a successfully tracked motion sequence. Even though the left foot is bent to the back and is nearly hidden from the depth camera, the 3D geometry of the legs has been recovered correctly. Figure 5.13(b) depicts difficult bending motions. Despite the fact that such poses were not part of our pose database, the motions were tracked successfully. For such motions, the result χ_t^{DB} of the lookup step does not reflect the true pose of the actor. Thus, our selection scheme decided in each frame correctly for the local optimization component which successfully tracked the motions. Figure 5.13(c) contains typical failure cases. The first two images show poses with severe self-occlusion which are still a challenge for pose reconstruction. Nonetheless, the overall pose is reliably captured and arm tracking quickly recovered once the occlusion was resolved. The rightmost image shows a case where the right arm was not visible in the depth input data. Since the proposed method assumes that at least parts of all limbs are still visible in the depth data, the pose of the right arm is not correctly recovered. Figure 5.13(d) shows examples of fast jumping, punching, and kicking motions where the first two motions are additionally rotated to more than $\pm 45°$ around the vertical axis with respect to the frontal viewing direction. The poses in this row are roughly recovered. However, small misalignments of some limbs might occur as visible in the right leg and the right arm, respectively. Also note the inaccuracy in the left leg (third pose). Such minor inaccuracies can locally occur and are typically resolved after a few frames. Figure 5.13(e) shows some poses of a successful tracking of a sequence with fast and complex kicking motions. Note that in the second pose of Fig. 5.13(e) it is difficult to distinguish the left leg from the right leg when having only the depth data of a single frame. However, since the local optimization and the database lookup components use temporal continuity priors, the legs can be tracked successfully. Finally, Fig. 5.13(f) contains a very fast arm rotation motion in a pose where the arms are close to being outstretched to the camera (first image), and a jumping motion in a similar pose (second image). Although only a small part of the arm is visible to the depth camera due to self-occlusions, the 3D geometry of the arm is successfully recovered. The last image shows a pose where the hands touch different parts of the body. Despite the fact that in such poses not all geodesic extrema in E_I^5 correspond to the end effectors, the motion has been tracked successfully since the hypothesis selection decided for the local optimization component. In the accompanying video [2] we show the performance of our prototype implementation also with the Microsoft Kinect depth camera.

First experiments showed that actors with different body proportions can be tracked if they are not too different from our body model. Therefore, we scaled the input data to roughly match the proportions of the model, see Fig. 5.14 and the accompanying video for examples. By scaling the input data, we avoid the recomputation of the pose database for a different model.

Fig. 5.14 Experimental results with a different person (ToF data from a PMD camera)

5.5.5 Limitations

In the proposed framework, we rely on certain model assumptions in several stages of the framework. For example, we use a rigged surface mesh that is assumed to fit the respective actor to be tracked. Therefore, we cannot directly track persons with substantially different body proportions than the ones reflected by the surface mesh. However, as shown in [54], the depth input can be used to estimate the body shape of the actual person. First experiments have shown that one can use the same tracking pipeline after applying a preprocessing step where the pose database is retargeted to correspond to the estimated body shape.

A second limitation arises in situations where only parts of the actor are visible in the field of view of the depth camera. Two assumptions within our framework lead to false pose estimates in these situations. Firstly, in the local optimization component, correspondences between the mesh and the depth data for all body parts of the mesh are established. If some limbs are not visible in the depth data, then the correspondences will inherently be semantically incorrect. Secondly, the geodesic extrema will not correspond to the limbs anymore and retrieved database poses can no longer stabilize the pose estimation. Therefore, the full body of the actor should always be visible in the depth data.

Another problematic situation can occur when the end effectors are not revealed for a longer period of time. Although we run two pose estimation components in parallel, each component in isolation does not give satisfying pose estimates as shown in the accompanying video—it is the combination that facilitates stable and accurate results. Therefore, if one of the components fails for an extended period of time, the results might become unstable. For example, if the end effectors are not revealed by the geodesic extrema for many successive frames, our algorithm continues to track using only local optimization. Then, fast motions lead to unstable pose estimation results, which are resolved as soon as the end effectors are detected again. To overcome this limitation, additional techniques for detecting end effectors could

be employed. For example, a fast method for detecting body parts similar to [48] could be used to identify the end effectors and supplement the geodesic extrema detections.

Finally, we expect the user to face the camera and rotate the body only within a typical range for interaction (±45°). We make use of this assumption in the normalization step in Sect. 5.3.4. By normalizing the depth data with respect to the estimated viewing direction, we can use a comparatively small pose database that contains each pose only in one normalized orientation. However, with our proposed method, the estimates for the viewing direction become unstable once the user leaves the admissible range of rotations. Since the database lookup component relies on a correct normalization of the input data, the retrieved pose hypotheses will not reflect the true pose in such cases. Another problem with strong rotations of the body is that then limbs are more likely hidden behind the body. To meliorate pose estimates in these situations, one could employ a dynamic model for simulating hidden limbs.

5.6 Conclusions

In this chapter, we introduced a combined generative and discriminative framework that facilitates robust as well as efficient full-body pose estimation from noisy depth-image streams. As one main ingredient, we described an efficient algorithm for computing robust and characteristic features based on geodesic extrema. These extracted geodesic extrema are used as query to retrieve semantically meaningful pose candidates from a 3D pose database, where no a-priori semantic labels of the extrema are necessary. Finally, a stable fusion of local optimization and global database lookup is achieved with a novel sparse distance measure that also accounts for the importance of the limbs. For all components of the pipeline, we have described efficient algorithms that facilitate real-time performance of the whole framework. In our experiments we improved on the results of previous work, both in terms of efficiency and robustness of the algorithm, as well as complexity of the tracked motions. As for future work, we plan to integrate a dynamic model for achieving stable pose estimates also for 360°-rotations and for occluded limbs. Furthermore, the fast run time of our method is one main ingredient that could spur further research for capturing several interacting people. Finally, we aim to integrate a method for automatically estimating the surface mesh of the person from depth data only.

Acknowledgements This work was supported by the German Research Foundation (DFG CL 64/5-1) and by the Intel Visual Computing Institute. Meinard Müller has been funded by the Cluster of Excellence on Multimodal Computing and Interaction (MMCI) and is now with the University of Bonn.

References

1. Azad, P., Asfour, T., Dillmann, R.: Robust real-time stereo-based markerless human motion capture. In: IEEE/RAS International Conference on Humanoid Robots, pp. 700–707 (2008)

2. Baak, A., Müller, M., Bharaj, G., Seidel, H.P., Theobalt, C.: Accompanied video to [3]. http://www.youtube.com/watch?v=QWNn01FWUkk (2011)
3. Baak, A., Müller, M., Bharaj, G., Seidel, H.P., Theobalt, C.: A data-driven approach for real-time full body pose reconstruction from a depth camera. In: IEEE International Conference on Computer Vision, pp. 1092–1099 (2011)
4. Baak, A., Rosenhahn, B., Müller, M., Seidel, H.P.: Stabilizing motion tracking using retrieved motion priors. In: IEEE International Conference on Computer Vision, pp. 1428–1435 (2009)
5. Bălan, A.O., Sigal, L., Black, M.J., Davis, J.E., Haussecker, H.W.: Detailed human shape and pose from images. In: IEEE Conference on Computer Vision and Pattern Recognition (2007)
6. Besl, P.J., McKay, N.D.: A method for registration of 3-D shapes. IEEE Trans. Pattern Anal. Mach. Intell. **14**(2), 239–256 (1992)
7. Bleiweiss, A., Kutliroff, E., Eilat, G.: Markerless motion capture using a single depth sensor. In: SIGGRAPH ASIA Sketches (2009)
8. Bo, L., Sminchisescu, C.: Twin gaussian processes for structured prediction. Int. J. Comput. Vis. **87**(1–2), 28–52 (2010)
9. Bregler, C., Malik, J., Pullen, K.: Twist based acquisition and tracking of animal and human kinematics. Int. J. Comput. Vis. **56**(3), 179–194 (2004)
10. Brubaker, M.A., Fleet, D.J., Hertzmann, A.: Physics-based person tracking using the anthropomorphic walker. Int. J. Comput. Vis. **87**, 140–155 (2010)
11. Cormen, T.H., Stein, C., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press, Cambridge (2001)
12. Demirdjian, D., Taycher, L., Shakhnarovich, G., Graumanand, K., Darrell, T.: Avoiding the streetlight effect: tracking by exploring likelihood modes. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 357–364 (2005)
13. Deutscher, J., Reid, I.: Articulated body motion capture by stochastic search. Int. J. Comput. Vis. **61**(2), 185–205 (2005)
14. Fossati, A., Dimitrijevic, M., Lepetit, V., Fua, P.: From canonical poses to 3D motion capture using a single camera. IEEE Trans. Pattern Anal. Mach. Intell. **32**(7), 1165–1181 (2010)
15. Friborg, R., Hauberg, S., Erleben, K.: GPU accelerated likelihoods for stereo-based articulated tracking. In: European Conference on Computer Vision—Workshop on Computer Vision on GPUs (2010)
16. Gall, J., Fossati, A., van Gool, L.: Functional categorization of objects using real-time markerless motion capture. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1969–1976 (2011)
17. Gall, J., Stoll, C., de Aguiar, E., Theobalt, C., Rosenhahn, B., Seidel, H.P.: Motion capture using joint skeleton tracking and surface estimation. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1746–1753 (2009)
18. Ganapathi, V., Plagemann, C., Thrun, S., Koller, D.: Real time motion capture using a single time-of-flight camera. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2010)
19. Girshick, R.B., Shotton, A., Kohli, P., Criminisi, A., Fitzgibbon, A.: Efficient regression of general-activity human poses from depth images. In: IEEE International Conference on Computer Vision, pp. 415–422 (2011)
20. Grest, D., Krüger, V., Koch, R.: Single view motion tracking by depth and silhouette information. In: Proceedings of the Scandinavian Conference on Image Analysis, pp. 719–729. Springer, Berlin (2007)
21. Guan, P., Weiss, A., Bălan, A.O., Black, M.J.: Estimating human shape and pose from a single image. In: IEEE International Conference on Computer Vision, pp. 1381–1388 (2009)
22. Hasler, N., Ackermann, H., Rosenhahn, B., Thormählen, T., Seidel, H.P.: Multilinear pose and body shape estimation of dressed subjects from image sets. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1823–1830 (2010)
23. Heikkila, J., Silven, O.: A four-step camera calibration procedure with implicit image correction. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1106–1112 (1997)

24. Knoop, S., Vacek, S., Dillmann, R.: Fusion of 2D and 3D sensor data for articulated body tracking. Robot. Auton. Syst. **57**(3), 321–329 (2009)
25. Kolb, A., Barth, E., Koch, R., Larsen, R.: Time-of-flight sensors in computer graphics. Comput. Graph. Forum **29**(1), 141–159 (2010)
26. Lewis, J.P., Cordner, M., Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In: Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH, pp. 165–172. ACM/Addison-Wesley, New York/Reading (2000)
27. Lindner, M., Schiller, I., Kolb, A., Koch, R.: Time-of-flight sensor calibration for accurate range sensing. Comput. Vis. Image Underst. **114**(12), 1318–1328 (2010). Special issue on Time-of-Flight Camera Based Computer Vision
28. López-Méndez, A., Alcoverro, M., Pardàs, M., Casas, J.R.: Real-time upper body tracking with online initialization using a range sensor. In: International Conference on Computer Vision Workshops, pp. 391–398 (2011)
29. MATLAB camera calibration toolbox. http://www.vision.caltech.edu/bouguetj/calib_doc (2012)
30. Microsoft: Kinect SDK beta. http://www.microsoft.com/en-us/kinectforwindows (2012)
31. Moeslund, T.B., Hilton, A., Krüger, V.: A survey of advances in vision-based human motion capture and analysis. Comput. Vis. Image Underst. **104**(2), 90–126 (2006)
32. Murray, R.M., Li, Z., Sastry, S.S.: A Mathematical Introduction to Robotic Manipulation. CRC Press, Boca Raton (1994)
33. Okada, R., Soatto, S.: Relevant feature selection for human pose estimation and localization in cluttered images. In: Proceedings of the European Conference on Computer Vision, pp. 434–445 (2008)
34. Okada, R., Stenger, B.: A single camera motion capture system for human–computer interaction. IEICE Trans. Inf. Syst. **E91-D**, 1855–1862 (2008)
35. Pekelny, Y., Gotsman, C.: Articulated object reconstruction and markerless motion capture from depth video. Comput. Graph. Forum **27**(2), 399–408 (2008)
36. Plagemann, C., Ganapathi, V., Koller, D., Thrun, S.: Realtime identification and localization of body parts from depth images. In: IEEE International Conference on Robotics and Automation (2010)
37. Poppe, R.: A survey on vision-based human action recognition. Image Vis. Comput. **28**(6), 976–990 (2010)
38. Primesense: Primesense NITE middleware. http://www.primesense.com (2012)
39. Romero, J., Kjellström, H., Kragic, D.: Hands in action: real-time 3D reconstruction of hands in interaction with objects. In: IEEE International Conference on Robotics and Automation, pp. 458–463 (2010)
40. Rosales, R., Sclaroff, S.: Inferring body pose without tracking body parts. In: IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 721–727 (2000)
41. Rosales, R., Sclaroff, S.: Combining generative and discriminative models in a framework for articulated pose estimation. Int. J. Comput. Vis. **67**, 251–276 (2006)
42. Rosenhahn, B., Schmaltz, C., Brox, T., Weickert, J., Cremers, D., Seidel, H.P.: Markerless motion capture of man–machine interaction. In: IEEE Conference on Computer Vision and Pattern Recognition (2008)
43. Salzmann, M., Urtasun, R.: Combining discriminative and generative methods for 3D deformable surface and articulated pose reconstruction. In: IEEE Conference on Computer Vision and Pattern Recognition (2010)
44. Schwarz, L.A., Mateus, D., Castañeda, V., Navab, N.: Manifold learning for ToF-based human body tracking and activity recognition. In: British Machine Vision Conference (2010)
45. Schwarz, L., Mkhytaryan, A., Mateus, D., Navab, N.: Estimating human 3D pose from time-of-flight images based on geodesic distances and optical flow. In: IEEE Conference on Automatic Face and Gesture Recognition (2011)
46. Shakhnarovich, G., Viola, P., Darrell, T.: Fast pose estimation with parameter-sensitive hashing. In: International Conference on Computer Vision, pp. 750–757 (2003)

47. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall, New York (2002)
48. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from a single depth image. In: IEEE Conference on Computer Vision and Pattern Recognition (2011)
49. Siddiqui, M., Medioni, G.: Human pose estimation from a single view point, real-time range sensor. In: Computer Vision and Pattern Recognition Workshops (2010)
50. Sigal, L., Bălan, A.O., Black, M.J.: Combined discriminative and generative articulated pose and non-rigid shape estimation. In: Advances in Neural Information Processing Systems, pp. 1337–1344 (2008)
51. Stoll, C., Hasler, N., Gall, J., Seidel, H.P., Theobalt, C.: Fast articulated motion tracking using a sums of gaussians body model. In: International Conference on Computer Vision, pp. 951–958 (2011)
52. Wang, R.Y., Popovic, J.: Real-time hand-tracking with a color glove. ACM Trans. Graph. **28**(3) (2009)
53. Wei, X., Chai, J.: Videomocap: modeling physically realistic human motion from monocular video sequences. ACM Trans. Graph. **29**(4), 42:1–42:10 (2010)
54. Weiss, A., Hirshberg, D., Black, M.J.: Home 3D body scans from noisy image and range data. In: IEEE International Conference on Computer Vision, pp. 1951–1958 (2011)
55. Ye, M., Wang, X., Yang, R., Ren, L., Pollefeys, M.: Accurate 3D pose estimation from a single depth image. In: International Conference on Computer Vision, pp. 731–738 (2011)
56. Zhu, Y., Dariush, B., Fujimura, K.: Controlled human pose estimation from depth image streams. In: Computer Vision and Pattern Recognition Workshops (2008)
57. Zhu, Y., Dariush, B., Fujimura, K.: Kinematic self retargeting: a framework for human pose estimation. Comput. Vis. Image Underst. **114**(12), 1362–1375 (2010). Special issue on Time-of-Flight Camera Based Computer Vision