ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
and Geometry Lab

Motion Capture in Uncontrolled Environments

Olga Diamanti

Master Thesis
September 2010

Prof. Dr. Marc Pollefeys

Abstract

The goal of a markerless motion algorithm is to recover the pose of a person from a set of images or videos. This is typically addressed in two ways: global optimization, used primarily when pose detection is the main interest, and/or local optimization, when the focus is on pose tracking. While the latter can provide very accurate results in relatively little time, it suffers from sensitivity to local minima. On the contrary, the former approach ensures a correct estimation of the pose, even without any prior pose information, at the cost of much higher computational complexity.

This thesis will address the pose detection problem in both these ways by enhancing an already existing tracking approach so as to obtain a robust motion capture algorithm. This will allow us to track people also in not controlled environments, such as in the case of outdoor scenarios. Additionally, we incorporated appearance modeling (by means of a texture map) with the aim to improve the tracking results.

Zusammenfassung

Das Ziel einer Motion-Capture Algorithmus ist die Rückforderung der Pose von einer Person aus einer Reihe von Bildern oder Videos. Dieser ist in der Regel auf zwei Arten angegangen: globale Optimierung, die hauptsachlich verwendet wird, falls das Hauptinteresse bei Pose-Erkennung ist, und/oder lokale Optimierung, wenn der Fokus auf Pose-Tracking ist. Während bei der lokale Optimierung sehr genaue Ergebnisse in relativ wenig Zeit geben können, leidet sie von Sensibilität gegenüber lokalen Minima. Im Gegenteil, die globale Optimierung sorgt eine korrekte Schätzung der Pose, auch ohne vorherige Information, auf Kosten der viel höheren Rechenaufwand.

Diese Arbeit wird sich mit dem Pose Erkennung Problem mit beiden diesen Methoden beschäftigen, durch die Verbesserung eines bereits bestehenden Tracking-Systems, um eine robuste Motion-Capture-Algorithmus zu erhalten. Dies wird uns erlauben, die Pose von Menschen auch in nicht kontrollierten Umgebungen bestimmen, wie z. B. im Fall von Outdoor-Szenarien. Darüber hinaus haben wir versucht, das Aussehen auch zu modellieren (durch eine Textur) , so dass wir die Tracking-Ergebnisse verbessern können.

Acknowledgements

I would like to thank Luca Ballan for the supervision and the great collaboration. The many discussions we had led to me gaining an insight into the area of motion capture and to a lot of interesting ideas.

I would also like to thank Gabriel Brostow for helping me getting me started in this field of research.

Last but not least I would like to thank my supervisor Prof.Dr. Marc Pollefeys for giving me the opportunity to work as part of his group and get acquainted with many of the ideas and applications of computer vision.

Contents

List of Figures

List of Tables

1

Introduction and Related Work

1.1. Overview

Motion capture, motion tracking, or mocap are terms used to describe the process of recording movement and translating that movement onto a digital model. It is used in military, entertainment, sports, and medical applications. In film making it refers to recording actions of human actors, and using that information to animate digital character models in 2D or 3D computer animation. When it also includes capture of the face and/or fingers and captures subtle expressions, it is often referred to as performance capture. During a motion capture session, movements of one or more actors are sampled many times per second; with most techniques, only the movements of the actor are recorded, not his/her visual appearance. This animation data is then mapped onto a 3D model so that the model performs the same actions as the actor.

A mocap system can generally be categorised as either **optical** or **non-optical**. **Optical** systems utilise data captured from image sensors to triangulate the 3D position of a subject between one or more cameras, calibrated in a way that ensures overlapping projections. Data acquisition is traditionally implemented using special markers attached to an actor; however, more recent systems are able to generate accurate data by tracking surface features identified dynamically for each particular subject. Tracking a large number of performers or expanding the capture area is accomplished by the addition of more cameras. These systems produce data with 3 degrees of freedom for each marker, and rotational information must be inferred from the relative orientation of three or more markers; for instance shoulder, elbow and wrist markers provide the angle of the elbow.

In **non-optical** systems, the actor uses complex devices, capable of revealing their positions inside the capture environment. A performer wears markers near each joint to identify the motion by the positions or angles between the markers. Acoustic, inertial, LED, magnetic or reflective markers, mechanical exoskeletons, or combinations of any of these are tracked, optimally at a frequency at least two times the frequency rate of the desired motion, so as to obtain submillimeter precision.

In both optical and non-optical cases, the acquired data is processed by a central unit which is responsible for actually estimating the motion. While the non-optical systems can provide high accuracy, fast computations, and more direct measurements, they can be quite invasive and uncomfortable for the actors to wear. Optical methods, on the other hand, are completely noninvasive and do not require special hardware; however, specific algorithms have to be developed to extract the results and make the system robust to problems such as body part occlusions.

The system described in this thesis belongs to the optical category. It is a multiview markerless motion capture system able to capture the shape, appearance and motion of people.

1.2. Related Work

A very good survey on existing motion capture systems can be found in [10]. Most of the existing markerless mocap algorithms consist of three basic steps: **initialization**, where the type of model is defined and its initial characteristics (shape, pose etc.) are set, **tracking**, which refers to the detection of the moving people throughout the entire duration of the scene, and **pose estimation**, which is the estimate of the actual body configuration of the person at each frame of the video sequence. Sometimes an extra step of **recognition** of the scenes follows.

A popular class of methods reconstructs the pose at time t from the pose at time $t-1$ based on an explicit representation of the kinematic structure of the human subjects. The analysis-by-synthesis approach is typically adopted in order to optimise a functional representing the similarity between observed and estimated data [2]. The **optimization** is generally performed by gradient descent techniques. Other approaches use stochastic tracking techniques like **particle filtering** ([7], [8]), which allow handling of abrupt pose changes; these methods have the drawback of a considerably increased computational complexity. A faster approach, called the **Hybrid Monte Carlo** filter ([13],[5]), uses sets of particles, instead of single particles; instead of relying on the frequently used schemes of resampling and mutating, each of the particles in the set evolves by performing a few gradient descent steps to produce a Markov chain of states. This allows the filter to cover higher dimensional state spaces with much fewer particles.

The works in the above described class of methods can be further classified according to the type and the domain of the used motion cues. The most used motion cue is **silhouette** information ([19]), but also **stereo** and/or **optical flow** are exploited. The motion cue

domains found in the literature can either be 3D or 2D. Methods belonging to the first class use the input images in order to get a coarse estimate of the 3D shape of the viewed human, i.e. the 3D cues. They then try to fit their own subject model within such a reconstruction. The model's pose which best fits the reconstruction will be the solution to the pose problem. Typically, these methods use visual hull techniques ([11]). Since the visual hull is an upper bound of the real surface, the reconstruction quality critically depends on the number of cameras: the fewer views available, the higher the probability that there will be discrepancies between the visual hull and the model to be estimated. Therefore, sometimes stereo information is additionally incorporated ([12]), in order to deal with concavities and to increase the 3D reconstruction quality. Other methods ([19]) use 3D motion field as additional information. The use of temporal motion models ([20]) is also an option. On the other hand, the methods based on 2D motion cues ([2]) minimise an objective function without directly inferring the 3D of the scene. The aforementioned reasons behind an imperfect match between the observed data and the estimated model, typical of 3D cues, do not exist in the case of 2D motion cues.

1.3. Our Approach

The system described in this thesis is an enhancement of the system described in [2], with the addition of a Hybrid Monte Carlo / particle filtering step as for example in [6] and [5]. It uses silhouette cues, texture and optical flow information in the form of an energy functional to be minimised, while at the same time employing the distance transform to evaluate the quality of the estimate.

1.3.1. The existing system

The existing motion capture system, which was the starting point of this thesis, models the moving human as a time-varying 3D mesh supported by an inner skeleton, as shown in Fig. 1.1. The skeleton is a kinematic tree where each node is a bone of the skeleton. The configuration (or state) of a skeleton (tree) with N bones is represented by a vector θ of dimension $6N$ where each dimension (an axis in $\mathbb{R}^{6N}$) represents a degree of freedom of a bone in the tree. Note that each bone has six degrees of freedom, three for rotation and three for translation. Rotations are parameterized by exponential maps. It should be noted that the actual skeleton has many more bones than the ones shown in Fig. 1.1, as is described in [2], with the option of disallowing some of them to move, or setting limits to their possible displacements. The model shown here is the one after applying these constraints, and is the one used for most of the experiments in this the thesis. Note that there is no specific bone for the feet and thus the mesh vertices that would normally be assigned to a foot bone are now assigned to the calf.

The method assumes that the 3D mesh of the human to be tracked is available at the starting position. For frame t, it estimates the motion state $\theta(t)$ of the articulated deformable model that best describes the human body in the scene, given the previous

motion state $\theta(t-1)$ and the set of images of the body's action taken at time t and at time $t-1$ from a set of C cameras. The algorithm first pre-processes these images in order to extract motion cues, then uses this information in order to define an objective function $g(\theta)$ having its minimum at the current motion state $\theta(t)$. The objective function also accounts for the non-rigid deformations of the body skin. In the end, it minimises $g(\theta)$ using as starting point for the minimisation $\theta(t-1)$. The minimisation is performed using the Levenberg - Marquardt algorithm ([9]).

For more details about the model, the mathematical formulation of the problem and the error functions used, see [2].

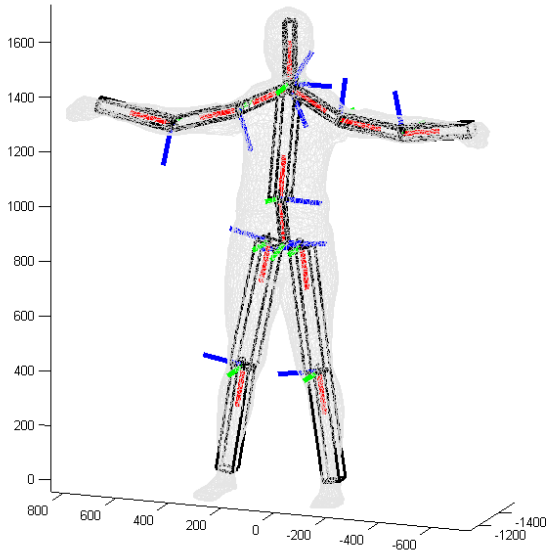


Figure 1.1.: *The 3D model used for the experiments. The figure shows both the 3D mesh and the 3D skeleton, where the bones are represented by boxes. The coordinate systems of the bones are also shown (red, green and blue colors correspond to the local x,y,z axes respectively). The dimensions of the model are the ones used for the HUMANEVA experiments.*

1.3.2. Video Sequences Used

For the purposes of the thesis we used two video sequences, one filmed under controlled indoor conditions by 4 static and precalibrated cameras and one filmed outdoors by 6 moving handheld cameras. The former (which we shall call the HUMANEVA sequence) was provided by the HUMANEVA dataset ([15]) and shows a person walking, then running, and then lifting his feet sideways in the interior of a room. The latter (which we shall call the JUGGLER ORANGE sequence) shows a juggler, performing outside an ETHZ building. Details about the synchronization, camera calibration and segmentation of the JUGGLER ORANGE sequence can be found in [1]. The camera setups are shown in Fig. 1.3 and Fig. 1.2 and some typical images are shown in Fig. 1.4.

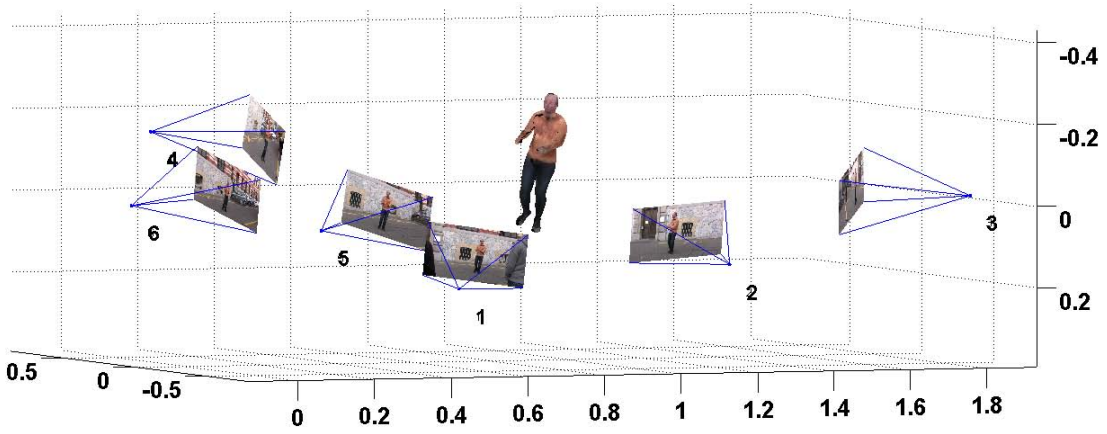


Figure 1.2.: *The camera setup for the JUGGLER ORANGE sequence. The locations of the cameras, their corresponding images as well as the position estimate for the mesh is shown.*

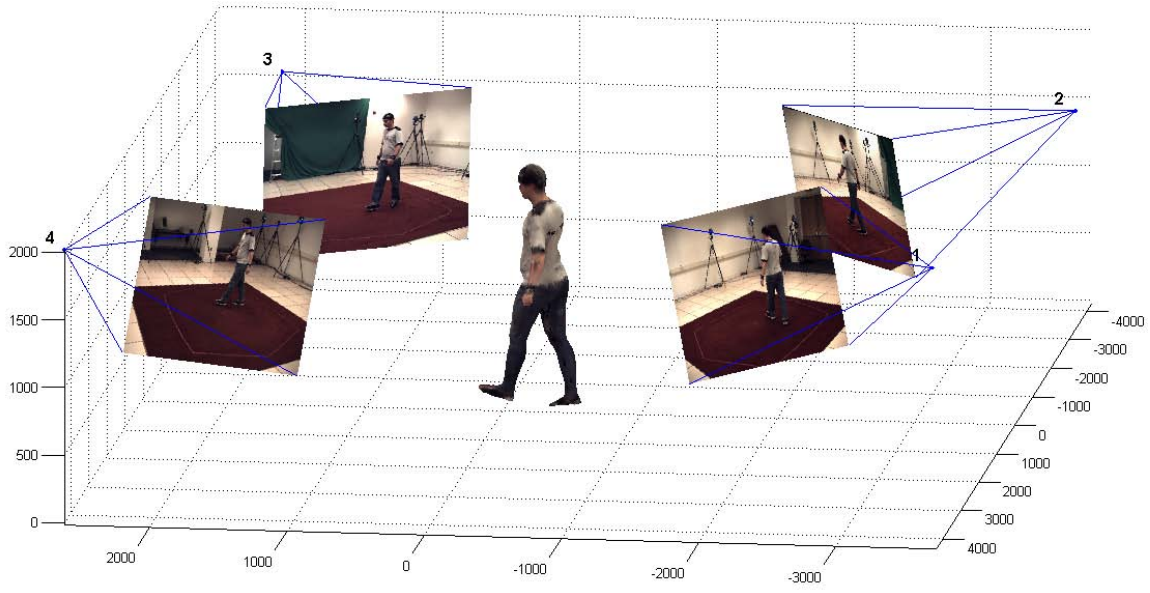


Figure 1.3.: *The camera setup for the HUMANEVA sequence. The locations of the cameras, their corresponding images as well as the position estimate for the mesh is shown.*

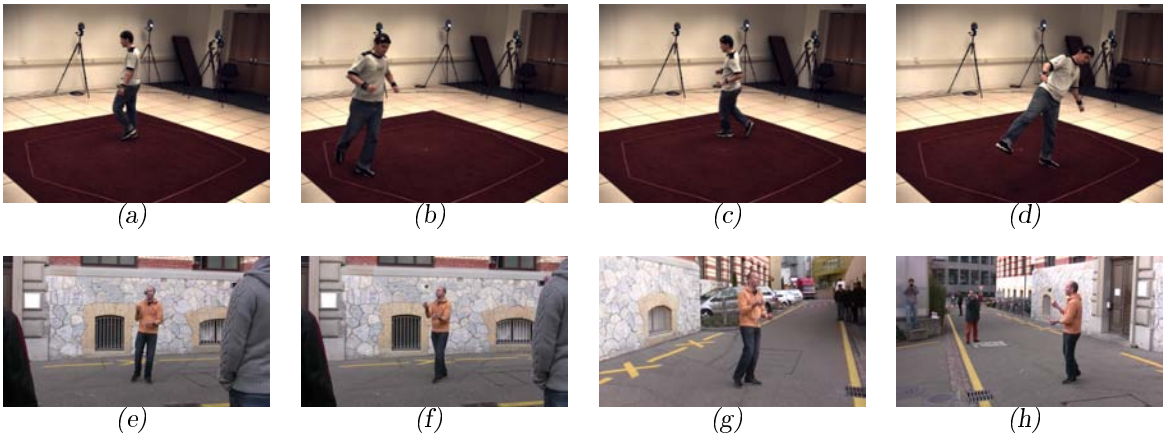


Figure 1.4.: *Some Images for the sequences used.* (a)-(d) *HUMANEVA Sequence.* (e)-(h) *JUGGLER ORANGE Sequence.*

The JUGGLER ORANGE sequence is significantly more challenging than the HUMANEVA one due to various reasons:

- The cameras are moving together with the subject, as is common in cases where people are filming outdoor events. Thus the cameras have to calibrated at each frame. Errors in the estimates of the camera positions and rotations in space greatly affect the mesh projections and therefore the performance of the system.
- Even though we have more cameras than in the HUMANEVA case, they are more unevenly distributed in space. In the HUMANEVA setup, the four cameras are carefully positioned so that they cover (almost) all of the room, while in the JUGGLER ORANGE setup the cameras are mostly positioned in a half circle around the subject, with the back of the juggler hardly ever visible.
- Due to the juggling action of the subject, his hands stay mostly close to the body. Given the camera setup, it is often hard to discern the locations of the hands in the segmentation, as the juggler appears as a single smooth blob. In these cases it is difficult to estimate the pose of the hands.
- The movement in the JUGGLER ORANGE scene is much faster and jittery. This affects greatly the accuracy of the image segmentation, due to the induced motion blur.
- The segmentation is further impaired by the presence of other moving objects and people in the scene, and by the cluttered background. When the moving person is in the region near the juggler, the segmentation often yields a single blob containing both persons. In that case, estimating the positions of the hands (if not the overall body) is extremely difficult. Also, the current segmentation algorithm is unable to distinguish between foreground and background regions when the two have the same or similar colors.

2

Global Optimization

In this chapter we will describe the methods implemented to deal with some of the problems of the local optimization system described previously.

2.1. Motivation

The local optimization performed in the system [2], while successful in many cases, is prone to local minima. Furthermore, once a pose is estimated wrongly for frame t, it is highly likely that the estimations for the next frames will also be wrong, since the starting point for the minimisation is incorrect.

An example of this problem is demonstrated in Fig. 2.1. In this case, while the estimation starts correctly, around frame 300 of the HUMANEVA sequence the pose estimation problem becomes ambiguous, as the four camera setting does not allow for discriminating which leg is moving forward and which stays back. The system then accidentally picks the wrong solution for that frame, and from then on the solution becomes problematic. The system does not recover and in the rest of the sequence the estimate for the body pose faces the wrong way.

The situation is even worse in the JUGGLER ORANGE sequence. As can be seen in Fig. 2.2, for the first 500 frames, the poor segmentation, where the hands are barely visible in the segmentation images in too many frames, leads to both hands being wrongly positioned in the same instead of two discrete regions in 3D space in these frames. Since the system only locally searches for matching silhouette points, it is unlikely to recover and guess the correct pose in the next frames. Instead, it searches for a pose very close

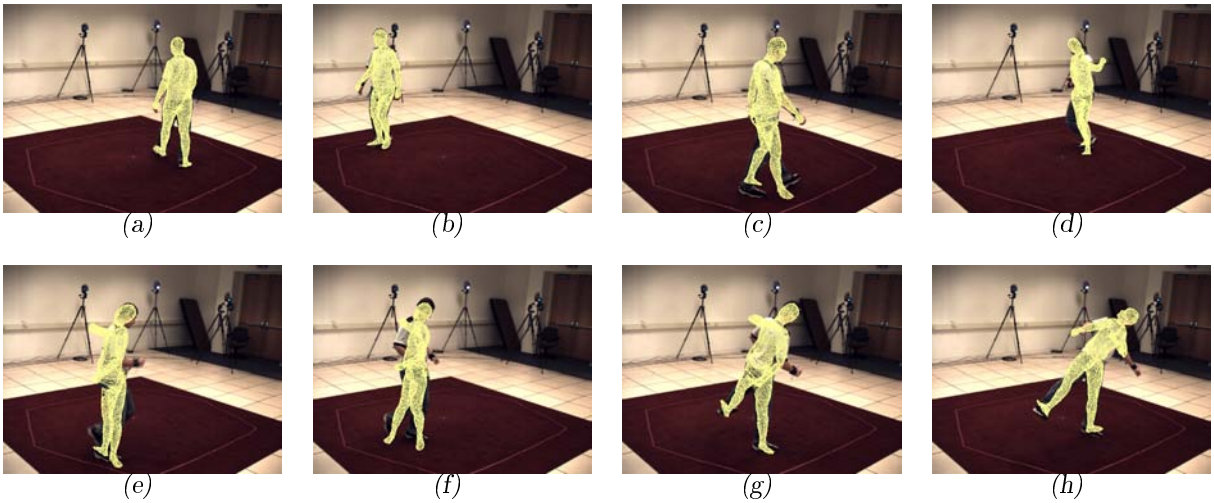


Figure 2.1.: *Failure of the local optimization in the HUMANEVA sequence. The figure shows frames 4, 146, 312, 430, 571, 797, 855 and 997.*

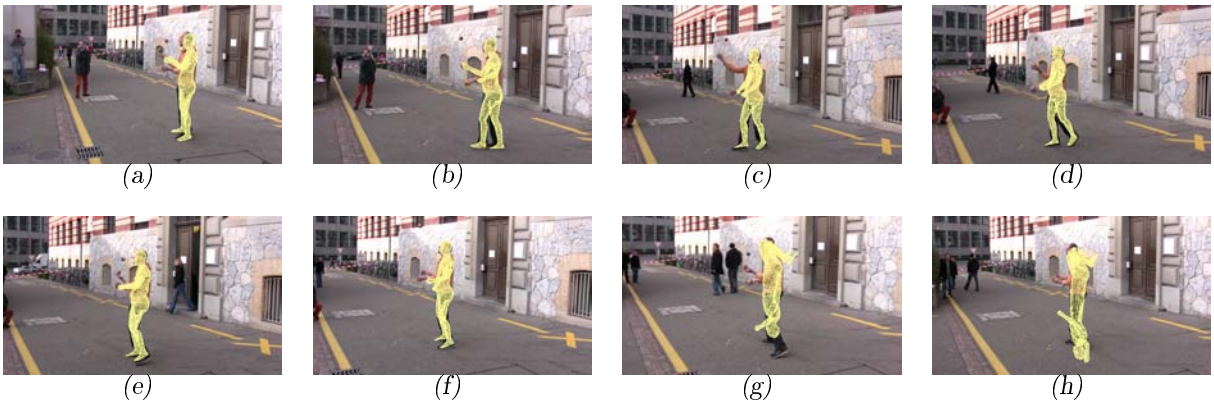


Figure 2.2.: *Failure of the local optimization in the JUGGLER ORANGE sequence. The figure shows frames 3, 216, 430, 441, 529, 762, 1090 and 1188.*

to the previous estimate, one that minimises the silhouette distances. This in the end leads to the estimate being inverted by 180^o degrees around frame 1050, and remaining upside down for the rest of the sequence. The system then still tries to minimise the pixel discrepancies in the silhouette images locally, which it does by placing the head in the region occupied by the right foot, and "raising" the left hand to where the left foot should be.

The above examples already hint towards a possible solution of the problem. The two main drawbacks of the existing system is the restriction to local search only, which renders the system unable to recover from a previous error in the next frames, and its inability to perform well in cases of not ideal segmentation (as in the JUGGLER ORANGE case, where it looses track of the hands when they are not clearly outlined in the segmentation). Thus, broadening the search space and attempting to enhance the information provided

by the segmentation might be the way to go. The former can be accomplished by global optimization methods, like particle filtering, which search throughout the whole state space for a global minimum, at the cost of a much higher running time. The latter can be addressed by using some alternative to the simple pixel distance in the silhouette image, one that would be more sensitive to small changes in the body part poses. One example is the distance transform function, which takes into account not only the number of pixels that fall within/out of the silhouette, but also of the shape of the silhouette. Thus the effect of small changes in the silhouette shape, such as protrusions that correspond to parts of the arms, is magnified and these changes are more likely to be detected.

We will start by an overview of the methods that provided the starting point for our algorithm, and proceed to describe our selected procedure.

2.1.1. Particle Filtering

As explained already, it is generally a good idea to perform a global search for the optimum. The approach implemented here is similar to the one found in [7], where, after each local optimization step, a test is performed where each of the bones of the model is checked to see whether it is correctly aligned to its silhouette. In cases of misalignment, a particle filtering step is performed. The dimension of the state space is reduced by fixing the degrees of freedom of the well-aligned bones to their correctly estimated values, and a search is performed throughout the space defined by the degrees of freedom of all the misaligned bones. The particles are initially uniformly distributed inside the reduced search space, but at each iteration of the particle filter they are weighted according to an evaluation function, and resampled so that only the best particles are kept for the next iteration. The evaluation function takes into account the differences between the distance transforms of the correct silhouettes (computed from the initial segmentation) and the silhouettes computed by using the particle's state as the mesh state. For view c the evaluation function is shown in Eq. 2.1.

$$E_c(\tilde{\theta}) = \frac{1}{area(S_c^p)} \sum_{p \in S_c^p} \left| D_c^p(\tilde{\theta})(p) - D_c(p) \right| + \frac{1}{area(S_c)} \sum_{q \in S_c} \left| D_c^p(\tilde{\theta})(q) - D_c(q) \right| \quad (2.1)$$

where : S_c^p and S_c are the estimated and correct silhouette images respectively, D_c^p and D_c are their distance transforms, and $\tilde{\theta}$ is the state of the mesh after using the particle state for the degrees of freedom of the misaligned bones, and using the estimates from the local optimization for the rest of the degrees of freedom (i.e., after projecting the state vector θ to a lower dimensional space by keeping some of its coordinates constant). The sums over p and q in Eq. 2.1 are to show that the differences are only added over the silhouette areas of S_c^p and S_c respectively, not over the background areas. The weight of a particle whose configuration is $\tilde{\theta}$ at iteration k of the particle filter is given by

$$w(\tilde{\theta}) = exp\left\{ -k^{0.7} \frac{\sum_{c=1}^{C} E_c(\tilde{\theta})}{C} \right\} \quad (2.2)$$

so that a "better" particle is assigned a bigger weight. The weights are then normalised so that they sum up to one. This is done so that a particle's weight better represents a probability for that particle's state; the choice of the particles during the resampling is done according to these probabilities.

After the resampling, a mutation step takes place, which spreads the resampled particles a bit so that in the end they can cover a bigger part of the search space. For more details, see [6].

The evaluation function that we use here is similar to the above, only the misalignment test is optional, as explained later.

2.1.2. Hybrid Monte Carlo

As shown in [5], particle filtering can be very computationally expensive when the whole multidimensional search space needs to be covered, as a great lot of particles need to be used. In order to bypass this issue, the authors of [5] propose to use chains of particles instead of single particles. Each particle has its own state (here, the set of degrees of freedom for all the bones, typically of dimension 25-35), and a weight, as given by an evaluation function (similar to the one described in the previous section). At each iteration the particle generates a new state, by performing a set of gradient descent steps on the evaluation function. More precisely, the state $\vec{s}$ of each particle is enhanced with a momentum variable $\vec{p}$ of equal dimension (initially sampled from a unit Gaussian) , and both of these variables undergo a set of evolution steps as in eq. 2.3.

$$\begin{aligned} \frac{\partial \vec{p}}{\partial t} &= -\nabla E(\vec{s}) \\ \frac{\partial \vec{s}}{\partial t} &= \vec{p} \end{aligned} \tag{2.3}$$

,where t is used to denote the evolution steps. In practise this is implemented numerically by a set of leapfrog steps (see [5]). It can be proved that the above evolution equation simulates a physical system evolving according to Hamiltonian dynamics; that is, a system where the sum of the potential energy (as defined by the evaluation function, which depends on the state $\vec{s}$) , and the kinetic energy (which depends on the momentum variable $\vec{p}$) remains constant.

In our approach we shall use a slightly different version of this, as the gradient descent step is inherent to the local optimization used in our system.

2.2. Our Approach

Instead of the normal HMC evolution steps, we combine the particle filter with the optimization present in the HMC filter by having each of the particles perform a few of the Levenberg-Marquardt optimization iterations found in [2] (based on feature and silhouette

correspondences) to update its state. This approach is similar to the HMC in the sense that instead of relying on a large number of particles to search through the whole state space for the global optimum, it makes use of much fewer particles that evolve according to the gradient of an evaluation function. No momentum variables are defined, though, therefore the Hamiltonian is not preserved.

Furthermore, instead of relying entirely on the solution of the previous frame as the starting point for the optimization, we initially sample N particles from a Gaussian centred on a predicted solution $\tilde{\theta}(t)$ for the current frame. We illustrate how the algorithm functions with a simple one-dimensional example shown in Fig. 2.3. Let's assume that the function to be estimated is one dimensional (for example, a single angle for one bone). Its true values, which we hope to be able to detect with our system, are shown as a curved line in Fig. 2.3(a), where the frame index is shown on the x axis and on the y axis the function values. Ideally, the error function on which the minimisation is performed (eg. the sum of feature and silhouette errors used by our system) attains its global minimum for each frame on points of that curve. These points are shown with black "x" signs. Apart from these minima, a number of other equally small minimizing points for the error function may exist for the same frame, for example due to ambiguous situations that occur in a multi-camera acquisition setup (eg. when the legs cross and the projected silhouettes are identical regardless of which leg is in front). These are also shown with black "x"-s in Fig. 2.3(a). However, since we are optimizing locally, a number of local minima of the error function might also exist, which may not lie on the curve, as shown with the orange "x"-s in Fig. 2.3(a).

The global minima of the error function correspond to the best solutions our system can find with this error function. This means that our system cannot directly discriminate between sets of global minima for a given frame, which in turn means that a possible solution path could consist of points other than the true pose, as shown in Fig. 2.3(b). This case cannot be directly avoided in our framework unless some sort of filtering is applied to all the results after the full solution for all the frames is calculated. On the contrary, the local minima can be avoided by our system, as the following figures demonstrate.

Fig. 2.3(c) shows the beginning of the method. Here, a prediction for the current frame is performed by extrapolating from the solution of the previous frame. This prediction may be close to the true pose, or it may be closer to another global or local minimum instead. If we only perform local optimization only, starting from this prediction, then the system will converge to the closest local minimum, which can be different from the true pose, as shown by the circled points in Fig. 2.3(d). Instead, we choose to sample some points from this prediction and guess some starting points for the optimization, as shown by the green circles in Fig. 2.3(e). By optimizing locally each of those guesses, we end up with a set of local or global minima, depending on which point is closest to each of the guesses is closest. Thus we find the set of solutions depicted with the red circles in Fig. 2.3(f). If we then compare the error functions of the detected minima, we can discard the local minima and keep only the global ones. A useful step might also be to compare the solution to the solution of the previous frame, to also discard any global minima far

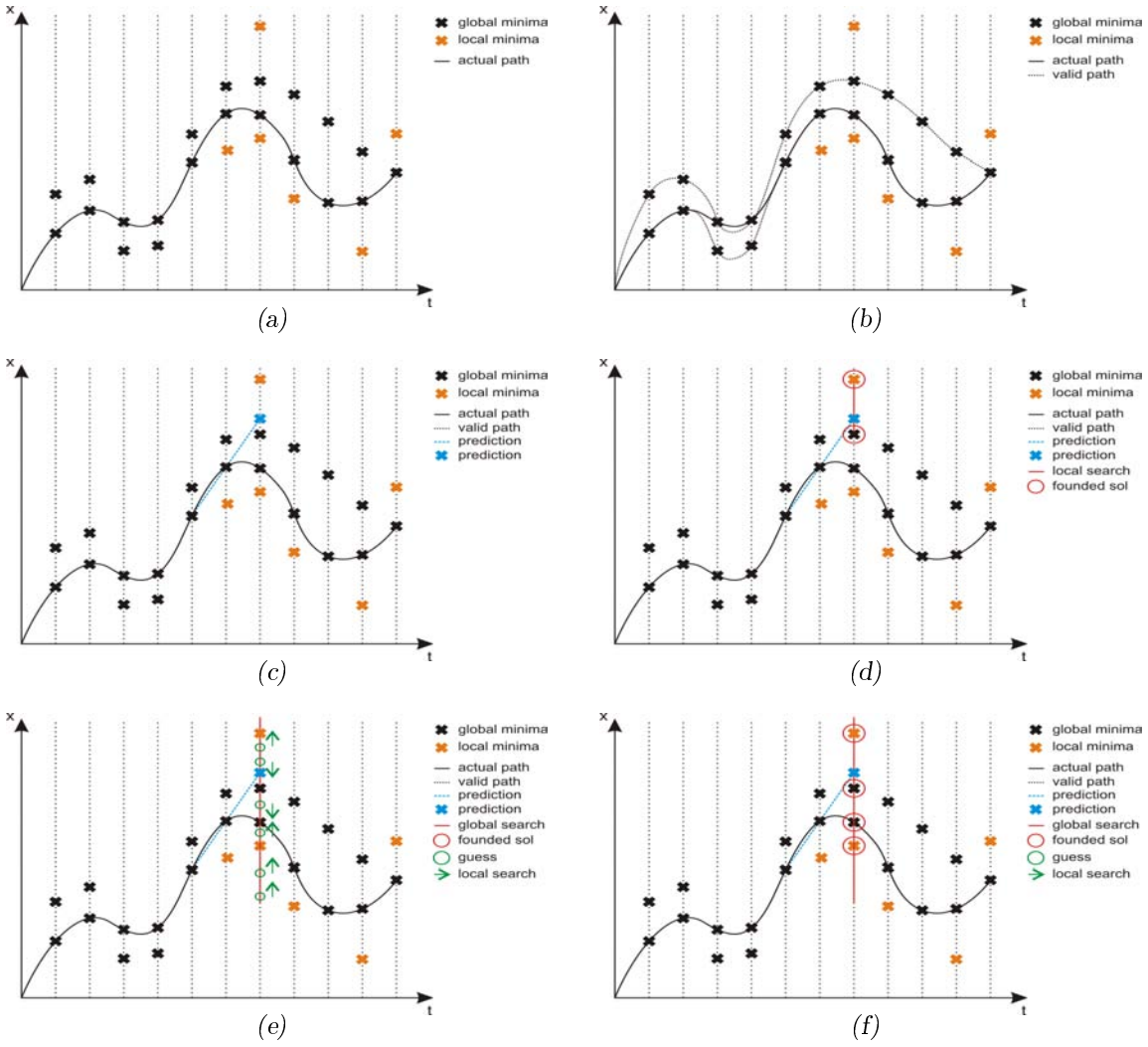


Figure 2.3.: *Illustration of our algorithm in 1D. See text for more information.*

away from the desired curve.

Our algorithm is a direct extension of the above in multiple dimensions. The overall approach is summarised in the form of a block diagram in Fig. 2.4.

The prediction for the current frame can be the solution of the previous frame itself, or an extrapolated estimate based on the solution of the two previous frames. Linear extrapolation is used if the degree of freedom is translational (Eq. 2.4), while SLERP quaternion interpolation is used for a rotational degree of freedom (Eq. 2.5).

$$\tilde{\theta}(t) = \theta(t-2) + 2 * (\theta(t-1) - \theta(t-2)) \quad , \text{ if } \theta \text{ is translational} \tag{2.4}$$

$$\tilde{\theta}(t) = \left(\theta(t-2)\theta(t-2)^{-1}\right)^{1.5} \theta(t-2) \quad , \text{ if } \theta \text{ is quaternion} \tag{2.5}$$

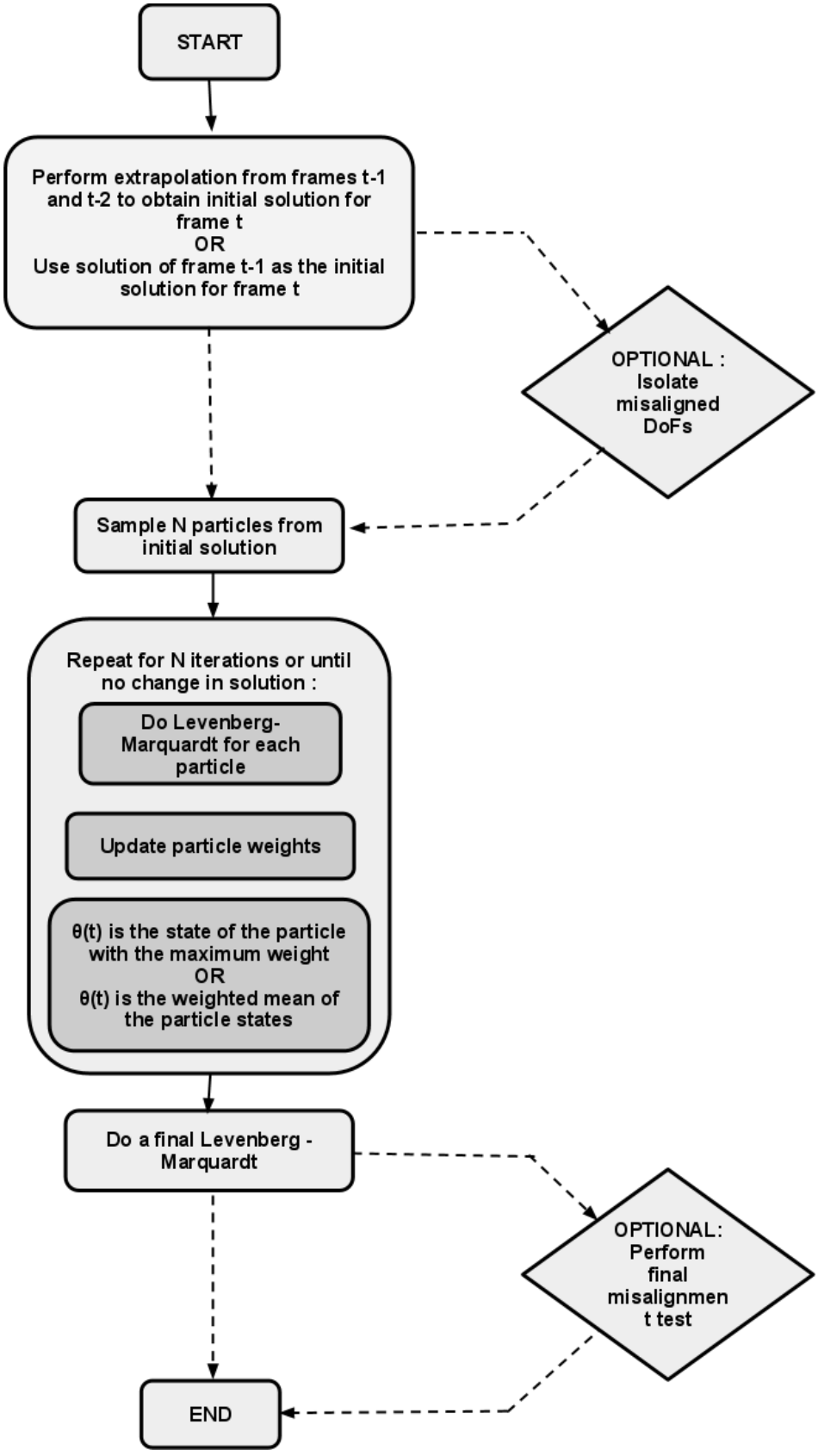


Figure 2.4.: *Flow chart of the algorithm.*

It should be noted here that SLERP works well only for interpolating **between** values and not for extrapolating, i.e., for interpolation parameter values between 0 and 1. This is the reason why an interpolation parameter of 1.5 instead of the expected 2 is used for the extrapolation of the rotation. Another option for the extrapolation is to keep the axis of rotation (for each bone) the same as in the previous frame, and extrapolate only the rotation angle linearly from the two previous frames. The extrapolated version of the prediction sometimes helps for the system to avoid getting stuck at ambiguous situations, e.g. when more than one solutions are equally possible due to there not being enough cameras to capture the correct pose.

The Gaussian used for the initialization of the particles should be relatively wide, so that the true solution can always be found, even if the solution for the previous frame is wrong. A standard deviation of 55 degrees for the rotational degrees of freedom and 0.1 times the room radius for the translational degrees of freedom proved to be usable values.

Each of these N particles starts from its initial state and performs a set of iterations of the Levenberg-Marquardt optimization to update its state. The error function for this optimization is based on the 3D-2D correspondences as described in [2]. After all particles are updated, a resampling step takes place, to discard the particles that are far from the correct solution. The particles' weights are evaluated according to Eq. 2.1, and normalised to a sum of 1 (see [6]).

With regards to the resampling step, there exist mainly two options. First, there is the option of resampling according to the scheme described in [6]; in this case, N particles are sampled with replacement from the initial set, using the particles' weights directly as probabilities, and afterwards a mutation step is performed. In this mutation step, the particles are spread around by a Gaussian with deviation proportional to the covariance matrix of the particle states. That is, as the iterations increase and the distribution of the particles becomes more and more centred on the final set of solutions, the spreading imposed by the mutation step becomes smaller. Alternatively, another resampling option is to pick the n best particles out of the N particles in the initial set, and sample $N - n$ more particles from these n. During this second sampling, a particle is selected with replacement from the n, again using those n particles' weights as probabilities, and then its state is mutated by a Gaussian; i.e., to generate one of the $N - n$ last particles in the set, we select one of the n best particles, place a Gaussian with its centre on this selected particle's state, and sample one particle from that Gaussian. The standard deviation of this Gaussian starts from the sigma value of the initialization Gaussian, but drops as the iterations increase, so that the particle set becomes more and more concentrated around a few good states, but some variation is still present in the particle set (alternatively, a mutation as in [6] can be applied in the $N - n$ particles). In addition to the above, the particles' mean state is added to the particle set. The various resampling schemes were found not to play an important role in the overall solution, but the system still provides an option for choosing one or the other.

The above procedure (evolution of the particles, weight update and resampling) is performed a number (4-10) of times (we will call them the "outer" iterations), or until the maximum change in the solution in below 4.5 degrees.

At each “outer” iteration (update of the particle set in a whole), the solution $\theta(t)$ for the current frame is updated and set equal to either the state of the particle with the maximum weight or as the weighted mean of the particles, depending on which of the two possibilities has a lower distance transform error. Thus the final solution $\theta(t)$ for the current frame can come from any of the “outer” iterations. This is done because the local optimization, since it minimises silhouette distances and not the overall distance transform error, might result in some particles getting “worse" (in terms of the distance transform error) and thus the estimates from previous iterations should also be kept.

After the “outer” iterations are completed, we perform a number of extra Levenberg - Marquardt iterations with silhouette and feature correspondences to the final solution, so as to better align it to the segmentation images. We found that this step helps in the cases where eg. the number of particles was not enough and the particle filter solution contained misaligned bones. These extra iterations in the end also help to give a smoother result; the initial HMC result appeared a bit noisy, since, contrary to the local optimization only case, there is no temporal continuity requirement.

The above steps assume that the particles' state is of dimension equal to the total number of degrees of freedom of the mesh. However, we added an option for the system to only update the misaligned degrees of freedom, as described in Section 2.1.1. In that case, the check for misalignment is performed at the beginning, right after the prediction for the current frame, and only the bones for which the prediction causes misalignment are used in the evolution of the particles. Additionally, there exists an option for doing a second misalignment test after the estimate, in which case if there exists misalignment, a normal particle filtering step is performed as in Section 2.1.1. However, this last test rarely detected misaligned bones, and this step was almost never used.

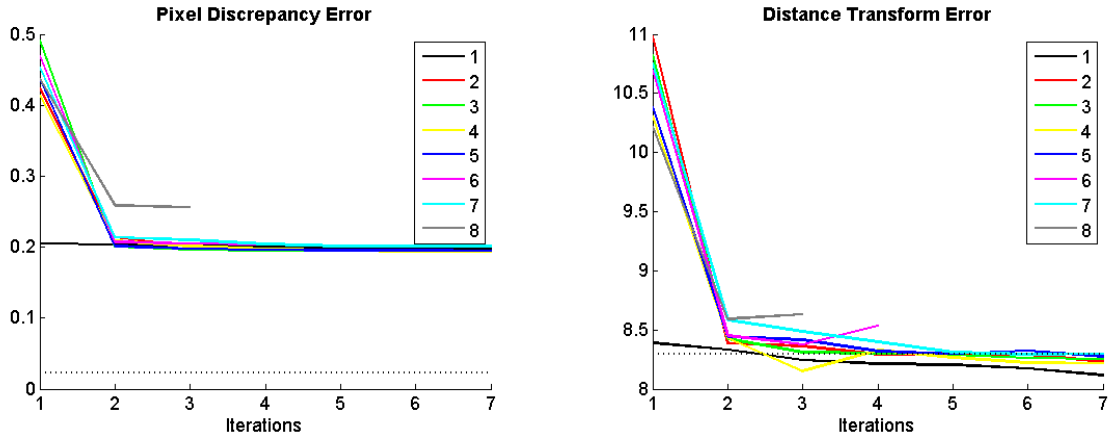
As can be seen from Fig. 2.4, the systems has a total of 5 parameters:

- the number of particles, N
- the number of best particles to keep at each resampling step, n
- the maximum number of updates to perform to the particle particle set
- the number of "inner" Levenberg - Marquardt(LM) iterations for each particle
- the number of final Levenberg - Marquardt(LM) iterations to perform in the end

plus a set of boolean parameters that lets the user choose

- whether all of the degrees of freedom will participate in the particle filtering (i.e., whether the initial bone misalignment test will take place)
- whether the final misalignment step will take place to correct the solution
- whether extrapolation should be used to predict an initial solution
- the resampling scheme.

In order to visualise the results of the evolution of the particles, we made the following experiment: We used $N = 15$ particles, and 7 iterations of the particle update step



(a) (b)

Figure 2.5.: *Pixel Discrepancy and Distance Transform error for each of the particles, for frame 4 of the HUMANEVA sequence. Each of the lines 1-8 corresponds to each of the 8 best particles. The dotted line is the error of the final solution found after the final LM step.*

(instead of 4). At each iteration we picked the $n = 8$ best particles and discarded the rest. The silhouette and distance transform errors for the best 8 particles at each iterations (frame 4 of the HUMANEVA sequence) are shown in Fig. 2.5. The silhouette error is the pixel discrepancy error described in [2] (basically the number of pixels in the XNOR between the estimated and the true silhouette, divided by the number of pixels in the true Silhouette), and the distance transform error is calculated as in Eq. 2.1.

In Fig. 2.5, notice that particles 6 and 8 are eliminated from the list of the best 8 after a few iterations, and are replaced by other particles. This is why they disappear in the graph. In Fig. 2.6, we can see the evolution of particle 7 in the first four iterations. Notice that after that practically nothing changes, and the states of the particle converges. This figure serves to illustrate the results of applying Levenberg - Marquardt optimization on a single particle. Fig. 2.7 illustrates the evolution of the mean particle state and the state of the best particle for the first 4 iterations of this experiment. The numerical value shown on the image is the exponential of the mean distance transform error for that specific mesh pose (Equation 2.2, for k=1). Notice that this error does not necessarily decrease at each iteration, as mentioned before, thus explaining the need for keeping the solution found at previous iterations until the end. The actual particle states and their evolution are shown in Fig. 2.8. Notice the convergence after a few iterations.

Results with regards to the error in the estimated poses are presented in Fig. 2.9. The error of the estimate of the particle filtering approach is clearly smaller, while the error of the local optimization is practically the same, with or without prediction, which means that prediction does not completely ensure that the system won't get stuck in ambiguous situations. Notice the shape of the curve of the error of the LO case; as we have seen in Fig. 2.1, the system starts to deviate from the correct solution around frame 300 and does

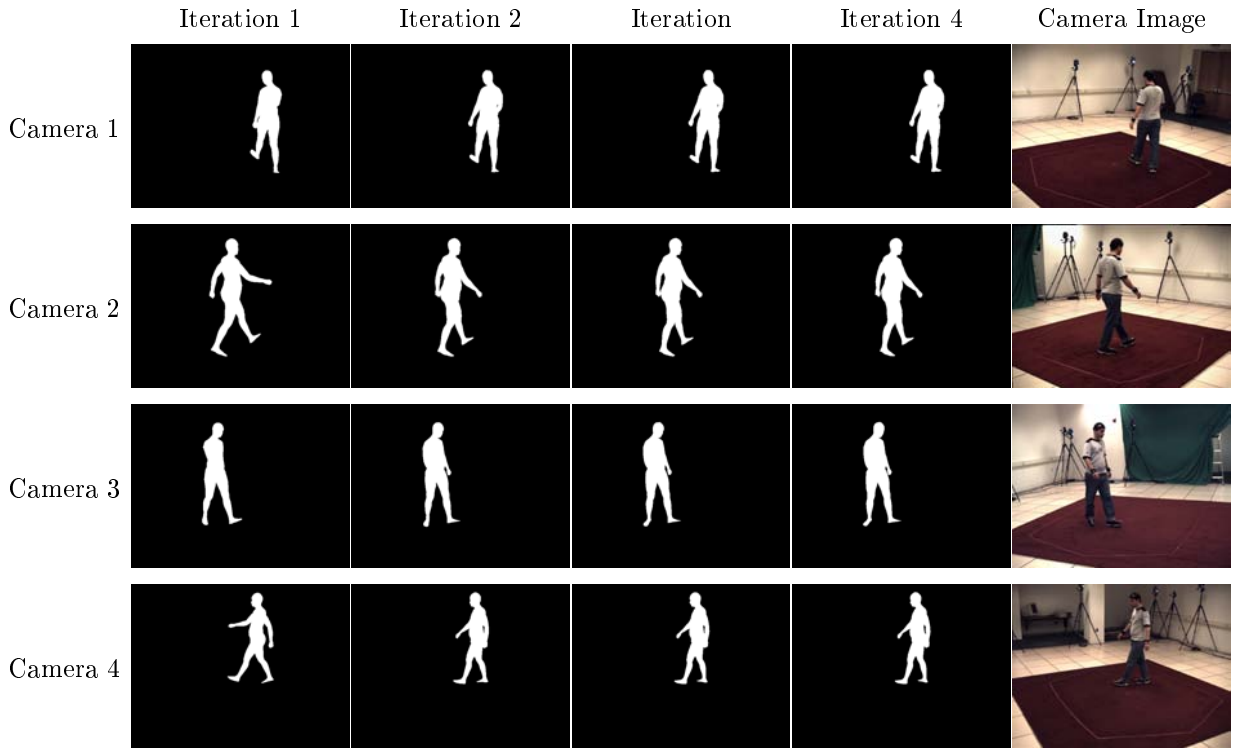


Figure 2.6.: *Evolution of particle 7, for frame 4 of the HUMANEVA sequence. Each row corresponds to a camera view and each column to an iteration of the particle update step.*

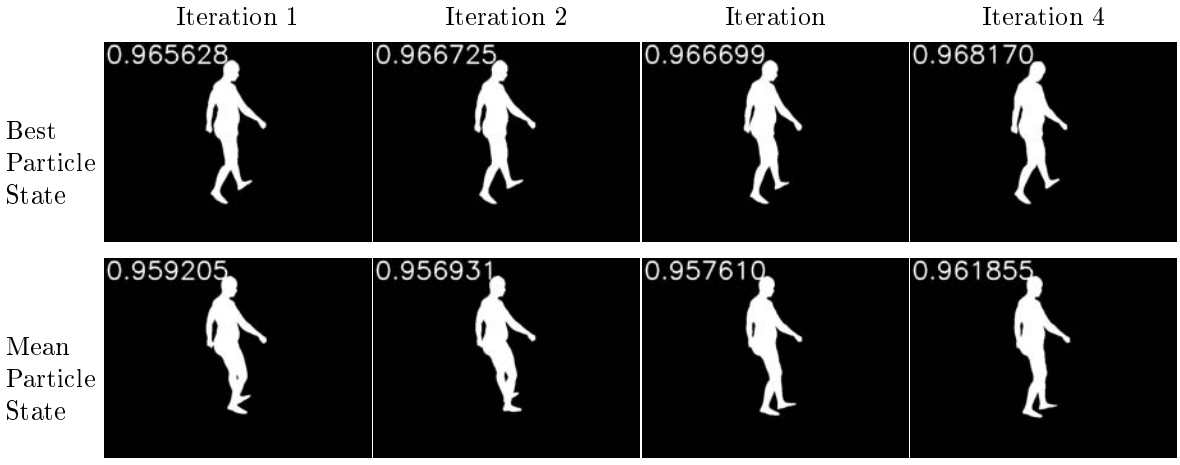
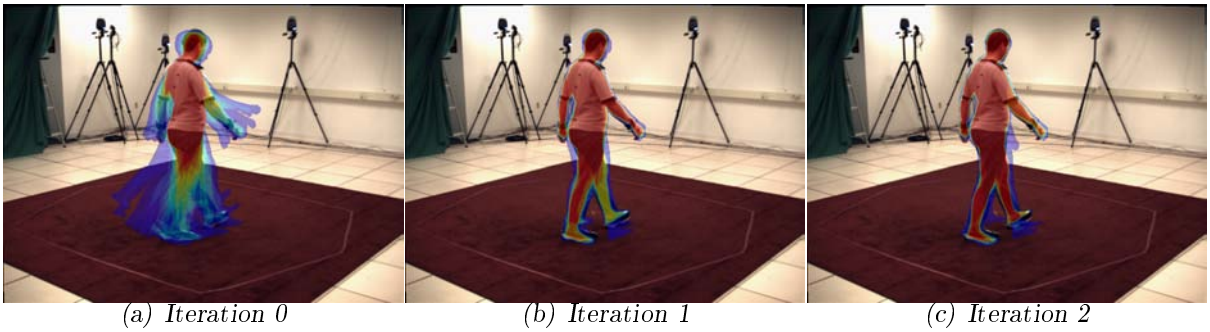
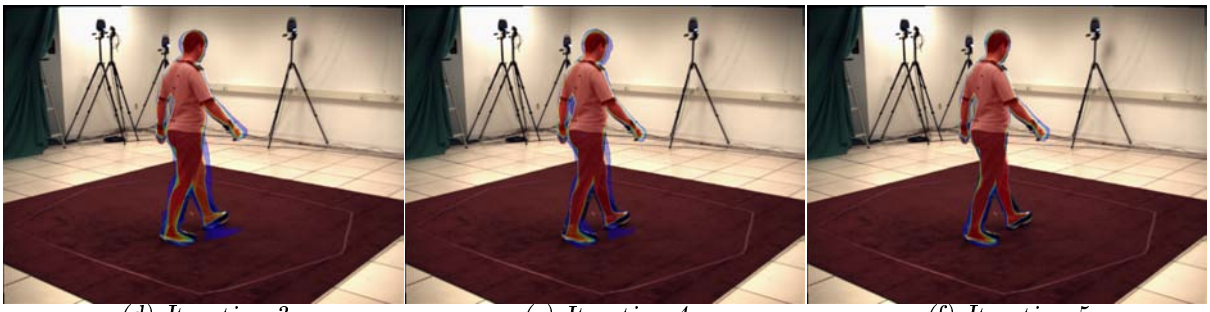


Figure 2.7.: *Evolution of the mean particle state and the state of the best particle, for frame 4 of the HUMANEVA sequence, as viewed from camera 2.*



(a) Iteration 0 (b) Iteration 1 (c) Iteration 2



(d) Iteration 3 (e) Iteration 4 (f) Iteration 5

Figure 2.8.: *Evolution of the particle set, for frame 4 of the HUMANEVA sequence, as viewed from camera 2.*

not recover, thus from that frame on there exists an increase in the error. Fig. 2.9(b) and Fig. 2.9(d), show the respective differences in the errors.

We should note that whenever the difference of the distance transform error is between -1 and 1, the total difference in the error in the whole silhouette is less than a pixel; this might be due to round-off errors in the rendering of the projections and should not taken into account. Also, it should be mentioned that even though the pose estimated in the LO case is very wrong, as shown in Fig. 2.1, the pixel discrepancy error in Fig. 2.9, and even the distance transform error are not that larger in the LO case when compared to the 30Part case, although the estimate in the latter case is much closer to the true pose. This reflects the fact that the local optimization system really does a good job in minimizing the silhouette error (and thus the pixel discrepancy error), but this error function is, in many cases, not ideal to distinguish between the correct and the wrong pose estimate.

Since the HUMANEVA database also contains ground truth information from a motion capture system that makes use of markers to estimate the bone locations and orientations, we also plotted the error with respect to those ground truth values. This is shown for all the frames in Fig. 2.10. The figure shows the errors before and after the final Levenberg-Marquardt optimization that is used to align the pose better to the segmentation masks. The average errors are shown with the dashed lines. As can be seen from Fig. 2.10(b), the addition of the final local optimization iterations also improves the errors by roughly 2-3 cm.

Fig. 2.12 shows actual results for some of the frames of the HUMANEVA sequence, with

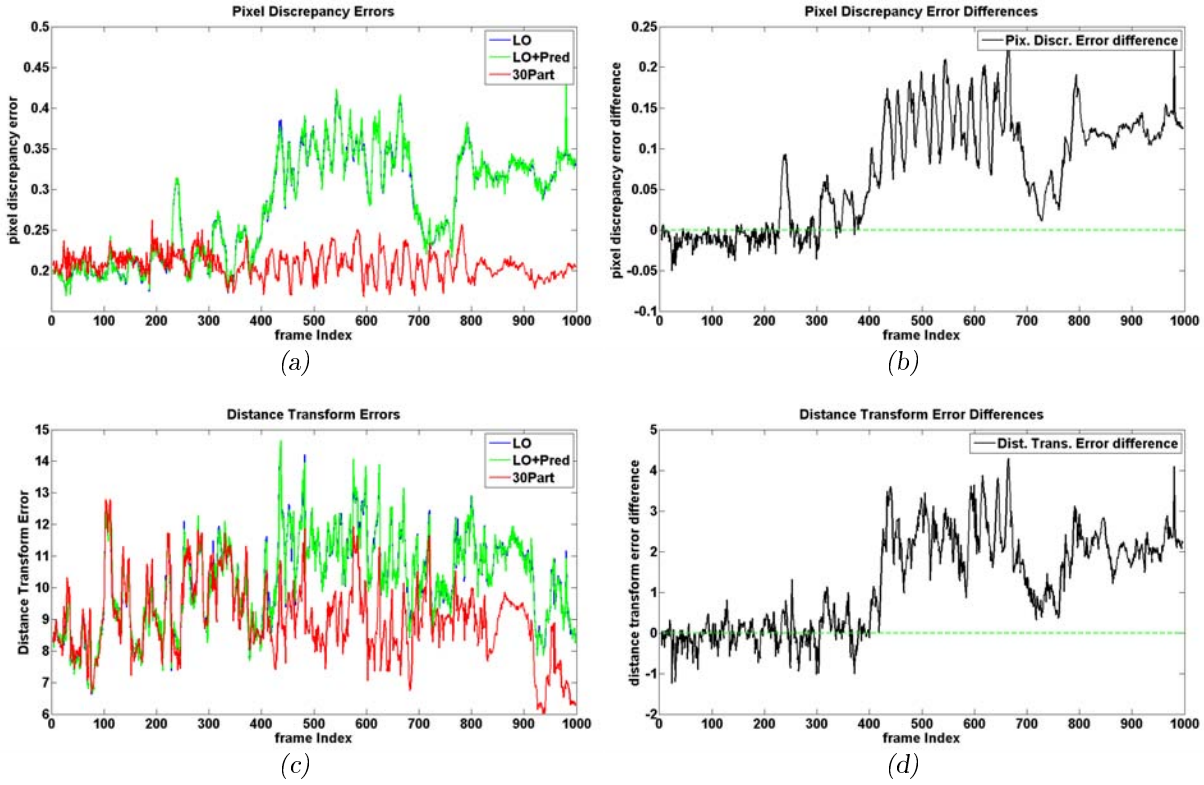
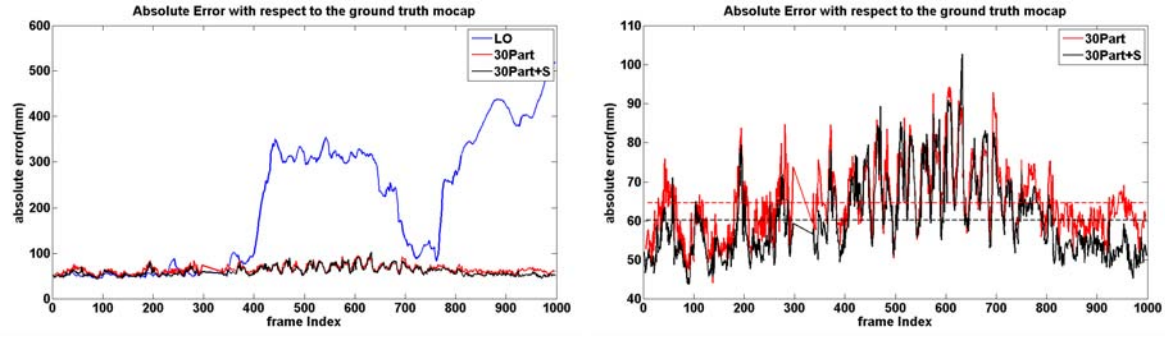


Figure 2.9.: *Comparison of the pixel discrepancy and distance transform errors. LO: local optimization only, LO+Pred: local optimization with prediction of the next solution by extrapolation, 30Part: particle filtering with 30 particles (a): Pixel Discrepancy errors, (b) Differences in the pixel discrepancy errors between the LO case and the 30Part case, (c) Distance Transform errors, (d) Differences in the distance transform errors between the LO case and the 30Part case.*

the estimated projection of the mesh overlaid onto the camera image. 30 Particles were used, for a total of 25 degrees of freedom:

- Pelvis (root) : 3 rotational + 3 translational degrees of freedom
- Thighs : 2x 3 rotational degrees of freedom
- Calves: 2x 1 rotational degree of freedom
- Head: 3 rotational degrees of freedom
- Upper arms: 2x 3 rotational degrees of freedom
- Lower arms: 2x 1 rotational degrees freedom

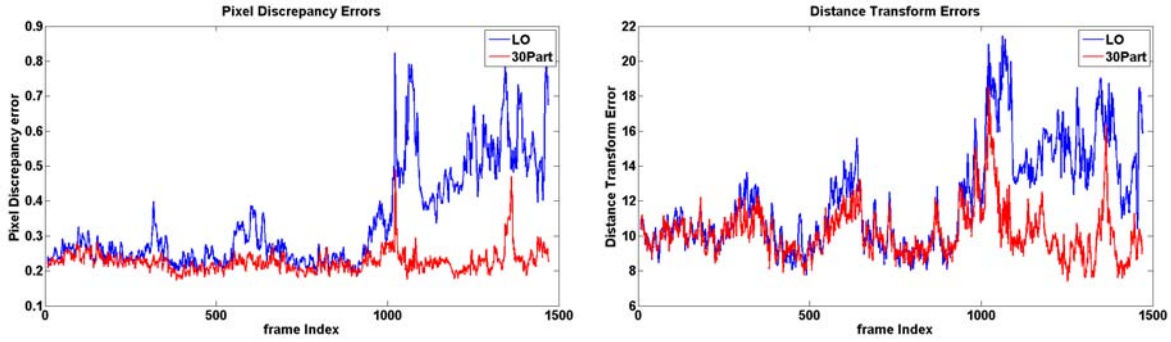
One thing that is worth mentioning about the HMC results for the HUMANEVA sequence is that the head of the model appears to be moving in a very abrupt and random way in the final result. This is because the silhouettes that we have from the segmentation provide virtually no information about the exact angles of the head, since almost all angles



(a) (b)

Figure 2.10.: *The average error per frame with respect to the ground truth for the marker positions, in millimetres. LO: the result of the local optimization only, 30Part: the result of the particle filtering with 30 particles, 30Part+S: the same result after application of the final Levenberg - Marquardt step. Frames 298-330 contain invalid data and are not used for comparisons, thus the straight parts of the curve. Fig. 2.10(b) is simply a zoom-in of Fig. 2.10(a) so that the particle filtering result is more clear. The dashed lines show the overall mean.*

produce the same silhouette. We will see in the next chapter how we can improve the result for the head a bit.



(a) (b)

Figure 2.11.: *Pixel Discrepancy and Distance Transform errors for the JUGGLER ORANGE sequence.*

For the results of the JUGGLER ORANGE sequence, see Fig. 2.13. Again, the particle set consisted of 30 particles. This time, however, the total number of degrees of freedom was 37. We added the following degrees of freedom:

- Spine: 3 translational + 3 rotational degrees of freedom. As the model was not as well aligned to the juggler as to the HUMANEVA subject, we added these 3 degrees to allow the mesh to shrink and extend around the torso region to match the juggler's silhouette. No remeshing was done in this case (i.e. the same weights were used during skinning), which explains the sometimes unnatural and "crooked" look of the mesh around the waist and the torso.

- Clavicles: 2x 3 rotational degrees of freedom. This is targeted at enabling the system to handle the more complex juggling motion.

Note that the system recovers the pose more or less successfully, even though the number of particles is actually smaller than the number of degrees of freedom, i.e. less than one particle is assigned to each degree. The respective pixel discrepancy and distance transform errors are shown in Fig. 2.11. Note that these are far from ideal error metrics to assess the quality of the result in this case, since the segmentation is not exactly perfect. These results are provided only for comparisons.

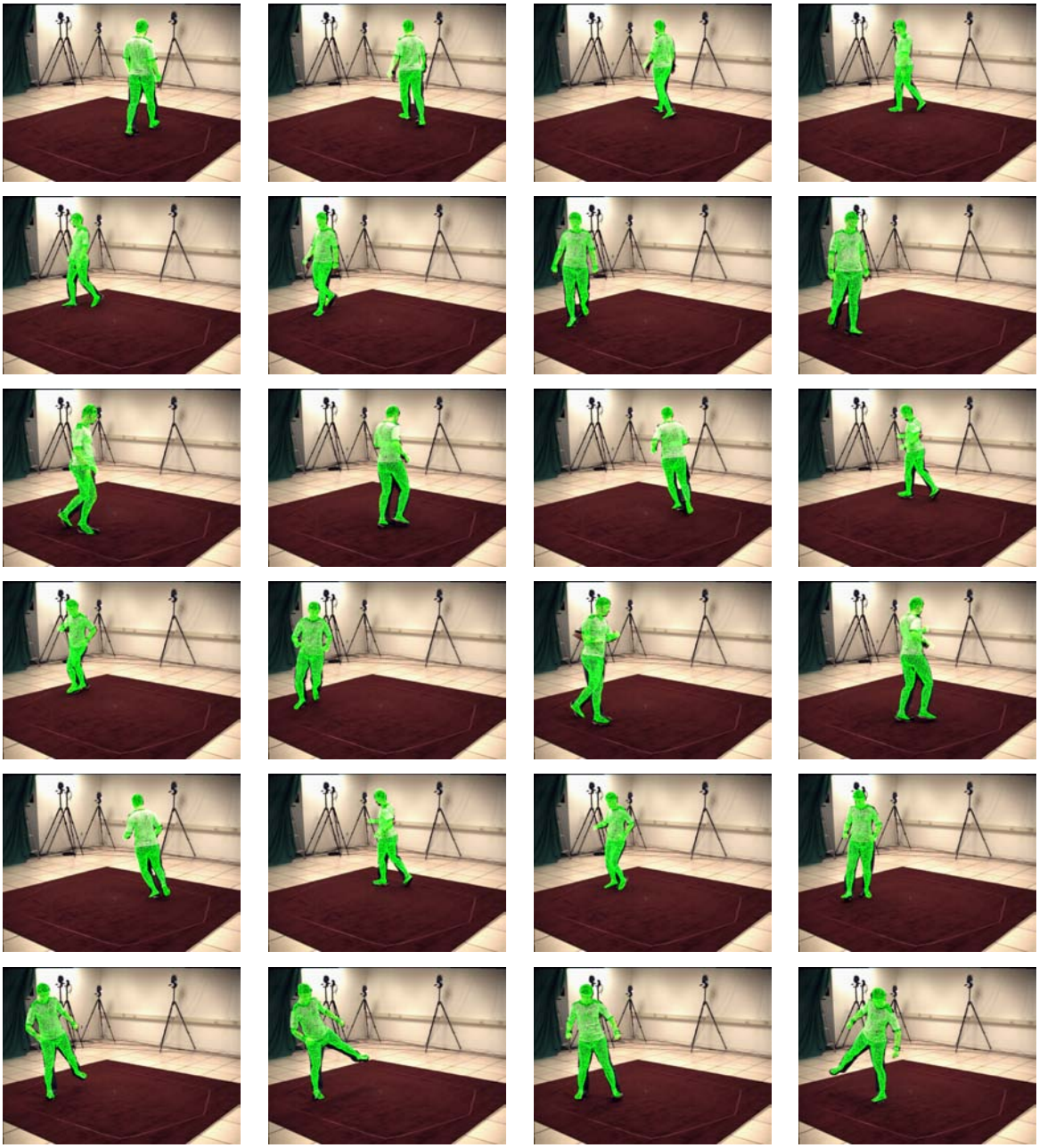


Figure 2.12.: *Final result for the HUMANEVA sequence, approximately every 40 frames (from left to right and top to bottom).*

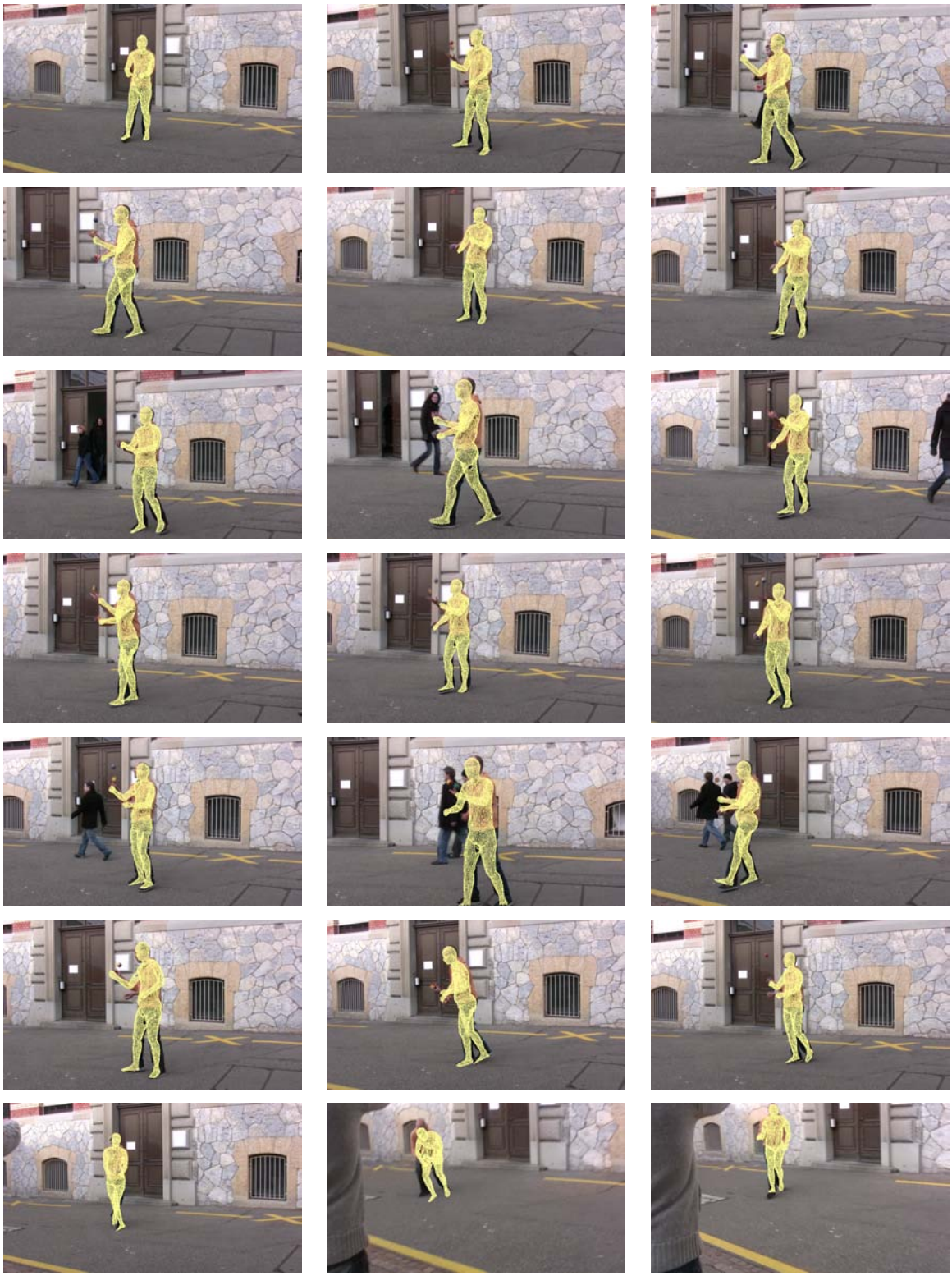


Figure 2.13.: *Final Result for the JUGGLER ORANGE sequence, approximately every 64 frames (from left to right and top to bottom).*

3

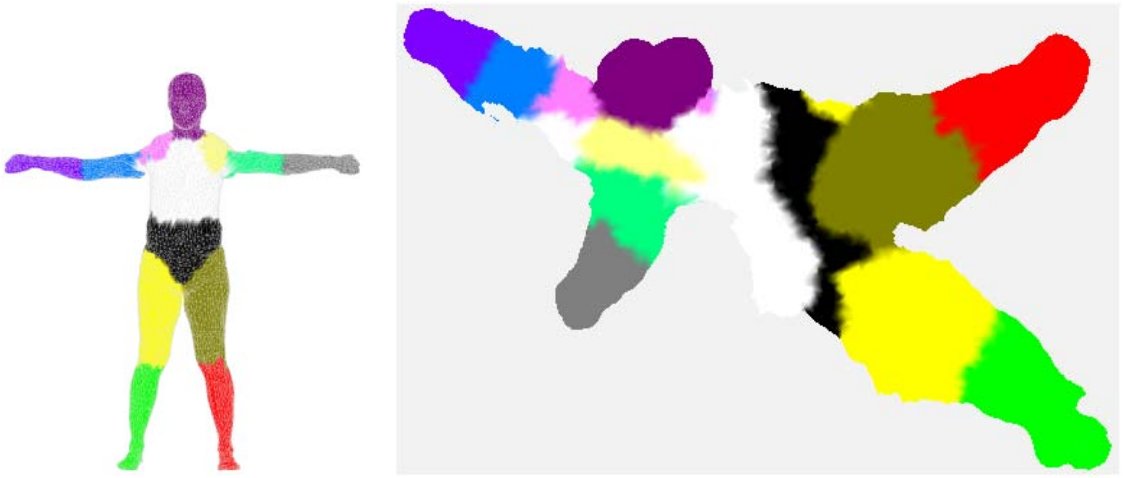
Texture

So far the information used in the system consists of silhouette-related information (either in the form of true silhouette distances, or in the form of distance transform differences), and optical flow on the image plane. In this chapter we will describe the use of appearance as an additional source of information. In order to capture the appearance of the subject in the video, we utilise a texture map. The creation of the texture map and its use in the existing system are described in the sections that follow.

3.1. Creation of the texture map

The first step to create a texture map is to map each of the vertices to a set of texture coordinates (u, v). This mapping is an internal property of the 3D model and remains constant for the entire video sequence. This essentially means that while the 3D location of a vertex v may change over time, its texture coordinates never change. The texture coordinates are unique for every vertex and no two vertices map to the same texture coordinates, i.e., the mapping is one-to-one and there are no overlaps in the texture space. This mapping between vertices and texture coordinates is illustrated in Fig. 3.1, where the reader can see where the different regions of the mesh are located on the texture map.

To compute the texture for frame t, we start by projecting each point of the mesh onto each of the cameras from which the point is visible. The visibility of a point from a camera is determined by the dot product between the outward triangle normal at that point and the vector from the camera to the point. Whenever the dot product is positive,



(a) (b)

Figure 3.1.: *The mapping between the 3D mesh and the texture map. (a): The 3D mesh. Each colored region roughly corresponds to a human bone. (b): The texture domain.*

the corresponding point is visible from the respective camera; also, the magnitude of the dot product indicates how reliable the camera's view of the point is. The camera pixels at the 2D projection are then copied to the corresponding 2D point in the texture domain. The resulting image is the camera's contribution to the texture map. Copying the values of the dot product for each visible mesh point onto the texture domain yields the visibility mask for that camera. For the camera setup and the position estimate shown in Fig. 1.3, the texture contributions as well as the visibility masks of each camera are shown in Fig. 3.2. Note that the points that are more visible from each camera are assigned bigger (whiter) values in the visibility mask.

The final texture map can be found by combining the contributions from all cameras. Since the visibility mask for each camera is a measure of how well the camera views each mesh vertex at the given configuration (and therefore how reliable the color that it assigns to the corresponding texture coordinates is), the visibility masks can be used to weigh the contribution of the cameras. The way to do this is to normalise the weights-visibility values (once for every point on the texture domain), and perform a weighted sum of the contributions of the cameras. The result will be a full texture map, with holes only in the regions that are not visible from any camera. This weighted average result can be seen in Fig. 3.3.

However, since the cameras are not radiometrically calibrated, it can happen that two different but neighboring regions of the mesh, visible from eg. two different cameras, are assigned very different colors from each. The use of weights does not entirely resolve this problem, since it might happen that both regions are equally visible from their respective cameras. This will result in discontinuities in the colors of the texture map, as can be seen in Fig. 3.3. One approach to resolve the issue is to use pyramidal blending ([4]). This approach is usually applied to only pairs of images and with binary blending masks, not

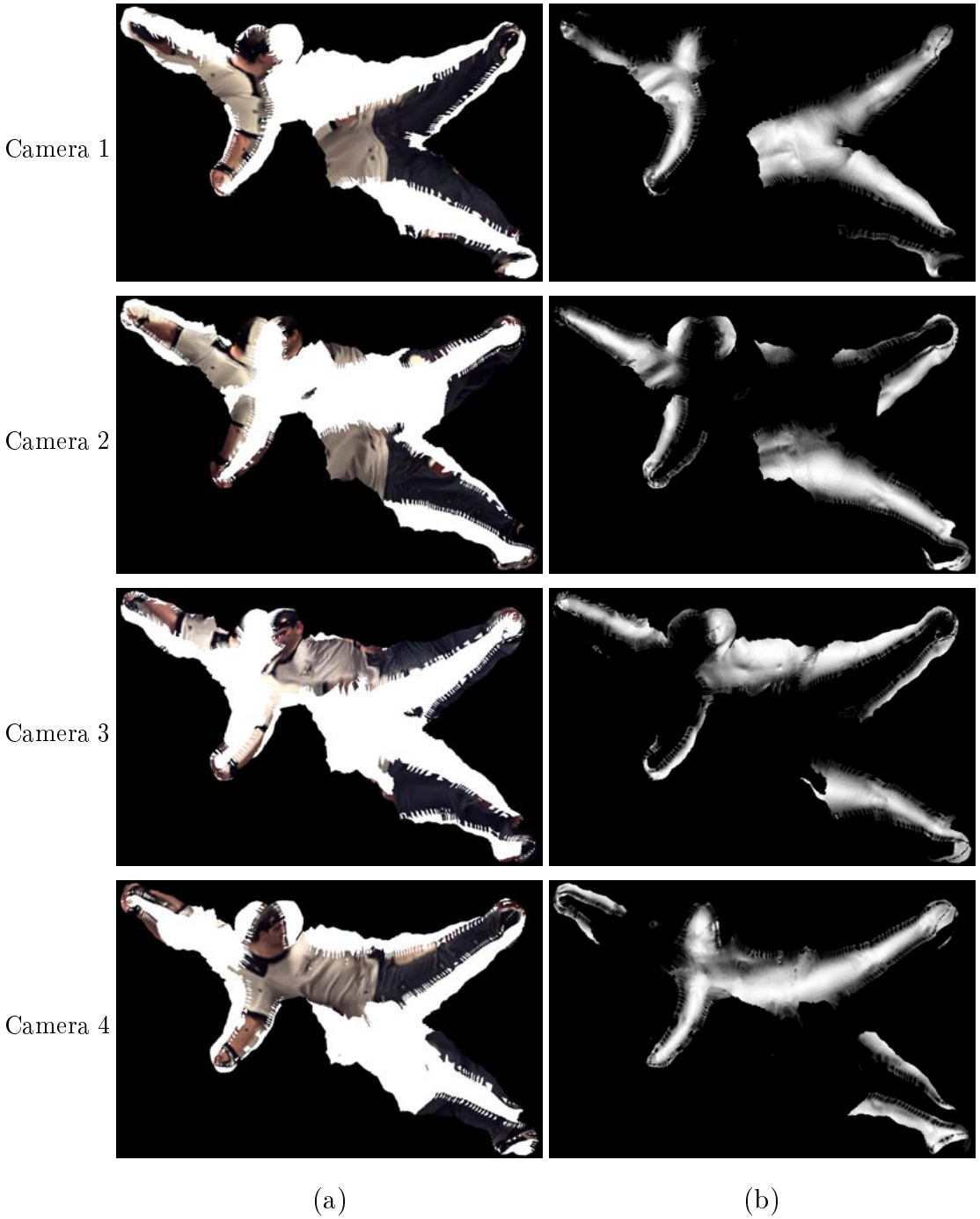


Figure 3.2.: *HUMANEVA: The texture contributions (a) from each camera, and the corresponding visibility masks (b). Each of the rows corresponds to a different camera, where the cameras are located as in Fig. 1.3.*

Figure 3.3.: *HUMANEVA texture map result, when only weighted averaging of the camera contributions is performed.*

grayscale, as is our case. When applied to a pair of images, the algorithm builds Laplacian pyramids for both the images and the blending masks, and performs weighted averaging of the Laplacians at each level of the pyramids to obtain a new composite Laplacian pyramid. Adding up the images at each level of the pyramid after the averaging yields the final result. While this works very well for binary masks, the results for our texture images proved to be poor. We then opted for a variant of this classical pyramidal blending approach, which is described in [14]. In brief, the algorithm proceeds as follows:

1. Create an alpha mapped image for each of the texture images, one from each camera. Here, the alpha map for each image is the visibility mask for the respective camera.
2. Create a weight map for each texture image:
   a) Threshold the visibility mask, so that the visible vertices for that camera (on the texture domain) are set to true and the rest to false. The resulting image is a binary mask that indicates the valid (visible) regions in the texture map for that camera.
   b) Perform a distance transform on the binary mask, to obtain the weight map. Each point of the weight map thus has a value proportional to its distance from the contour of the binary mask. Intuitively, this means that points in the inner part of the valid texture region for a camera are assigned larger weights, which reflects what is often the case in practise: Since each continuous region of the 3D mesh that is visible from the camera is normally mapped to a continuous blob in the texture domain, the interior of this blob is usually better visible

from that camera than the outer contour of the blob.

3. Create an alpha premultiplied version for each of the texture images. This is done by multiplying each of the red, blue and green channels of the texture image with the alpha map (the visibility mask).
4. Create a band pass Laplacian pyramid from each alpha premultiplied image. This means that Laplacian pyramids are constructed from all the channels of the premultiplied image, as well as from the alpha map itself.
5. Construct a low-pass Gaussian pyramid from the weight map (the distance transform) associated with each alpha premultiplied image.
6. Compute a new composite band-pass Laplacian pyramid using each band-pass Laplacian pyramid and its associated low-pass Gaussian pyramid for each pyramid level. Essentially this step performs a weighted average of the Laplacians at each level of the pyramid, using the corresponding level of the Gaussian pyramid of the weights (the distance transforms). This is performed for both the colour and the alpha pyramids.
7. Construct a new, blended, alpha premultiplied image by expanding and summing all the levels of the composite band-pass Laplacian pyramid.
8. Clip the red, green and blue values for each pixel of the blended image so that they fall within the normal limits of $[0, 255]$. Also crop the alpha channel so that it falls within the range $[0, 1]$.
9. Divide the red, green and blue channels by the alpha map at each pixel of the blended image.
10. Set the value of the blended image pixels to zero whenever these pixels are outside of the binary visibility masks of all the cameras.

The result can be seen in Fig. 3.4. Notice the much smoother transitions between the different regions of the texture map.

To facilitate the extraction of texture information (as will be described in the next sections), the camera images are sharpened prior to the creation of the texture map. This is done by smoothing the initial image with a 5x5 Gaussian kernel, taking the difference of the smoothed image from the initial image, and adding the result to the initial (prior to the smoothing) image. The result is a sharper version of the initial image. This is then used to calculate the camera's contribution to the texture map, and the texture map creation algorithm is then applied. Also, one more texture rendering is performed, this time using the silhouette masks instead of the original camera images as input to create the texture contributions from each camera. The result of this rendering indicates regions in the texture map where the 3D mesh (based on the current estimate of its pose) is located out of the convex hull created from the segmentation images. These regions are then cropped out, as they come from regions in the background and we are only interested in the projections 3D mesh which should normally lie within the silhouette images.

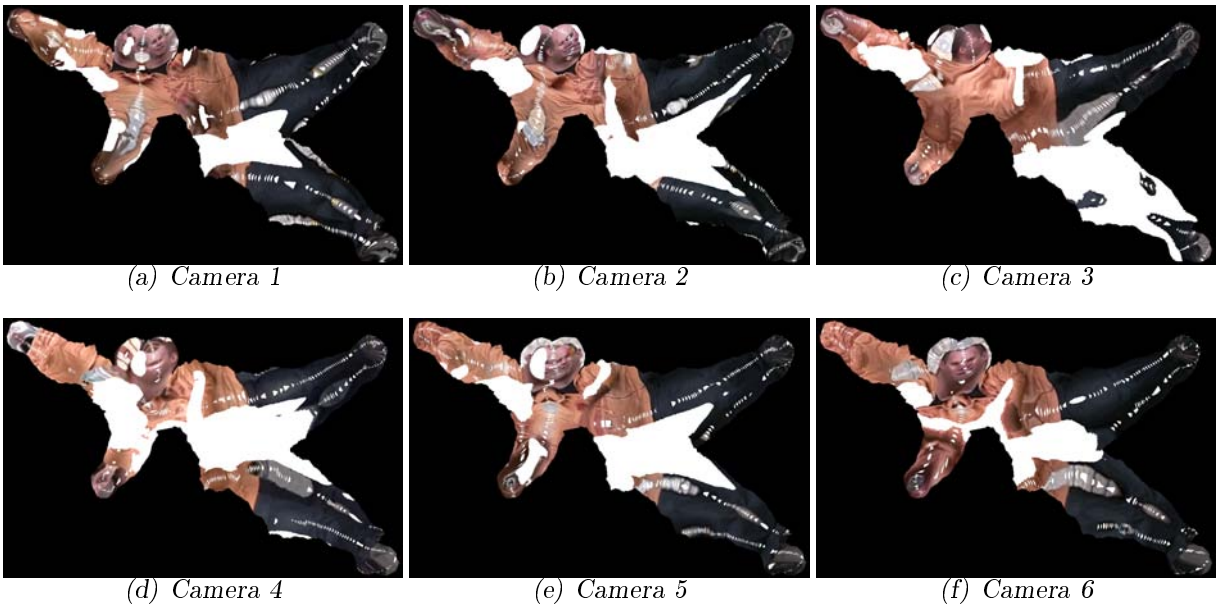
Figure 3.4.: *HUMANEVA texture map results, after pyramidal blending with alpha maps.*

The texture results for the JUGGLER ORANGE sequence, as can be seen from Fig. 3.5 and Fig. 3.6, are unfortunately not so good. This is mainly due to the fact that the mesh projections are not perfectly aligned with the true silhouette of the human, due to either incorrect segmentation, camera calibration errors or the fact that the 3D model does not match the subject exactly. Thus the contributions of the 6 cameras view different colors for certain regions of the mesh (eg. the head regions in the images of Fig. 3.5) and the result comes out blurred and uninformative.

3.2. Extracting Texture Information

In this section we show how the texture could be used to create sets of 2D correspondences on the image space, suitable to be used as input to the local optimization in the exact same way that optical flow and silhouette correspondences are used. The approach, similar to the ones found in [19] and [3], is based on calculating the optical flow on the texture map, and then projecting the optical flow back to the 3D mesh and then to the camera images themselves. This hopefully results in a set of good 2D correspondences between the projections of the vertex locations, as derived from the current model estimate, and the projections of the desired (true) vertex locations. We will demonstrate this by using a synthetic example.

Fig. 3.7 shows a set of 4 images from which the mesh pose is to be estimated. For the rest of the section we assume that a good estimate for the true texture map of the mesh is available, which can be seen in Fig. 3.8(a). In real cases, the true texture map could be



(a) Camera 1 (b) Camera 2 (c) Camera 3

(d) Camera 4 (e) Camera 5 (f) Camera 6

Figure 3.5.: *JUGGLER ORANGE: The texture contributions from each camera. Each of the rows corresponds to a different camera, where the cameras are located as in Fig. 1.2.*

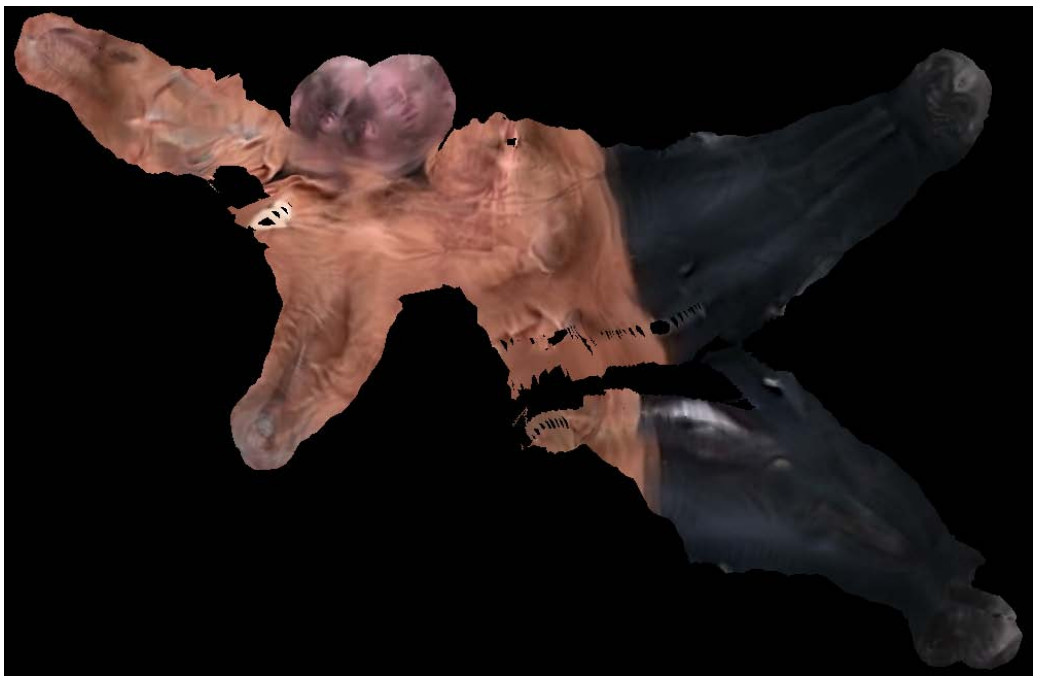
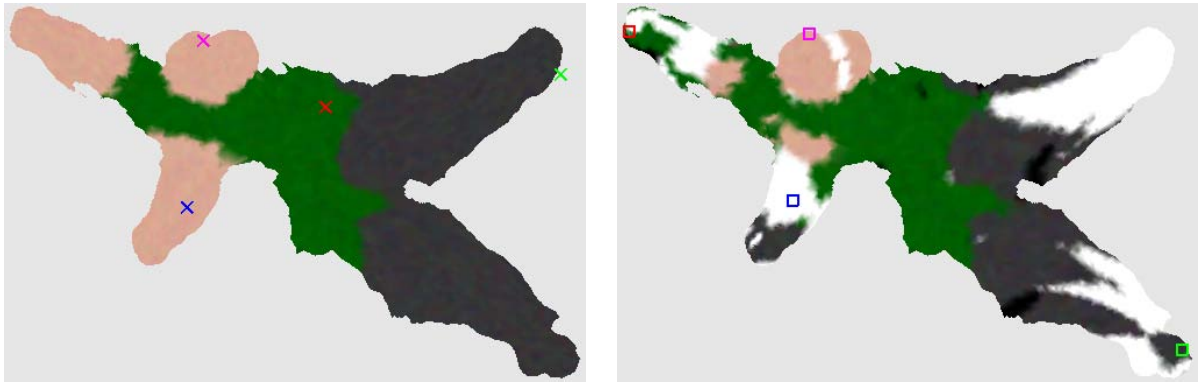


Figure 3.6.: *JUGGLER ORANGE texture map results, after pyramidal blending with alpha maps.*

(a) (b) (c) (d)

Figure 3.7.: *Four synthetic images from which the pose is to be derived.*



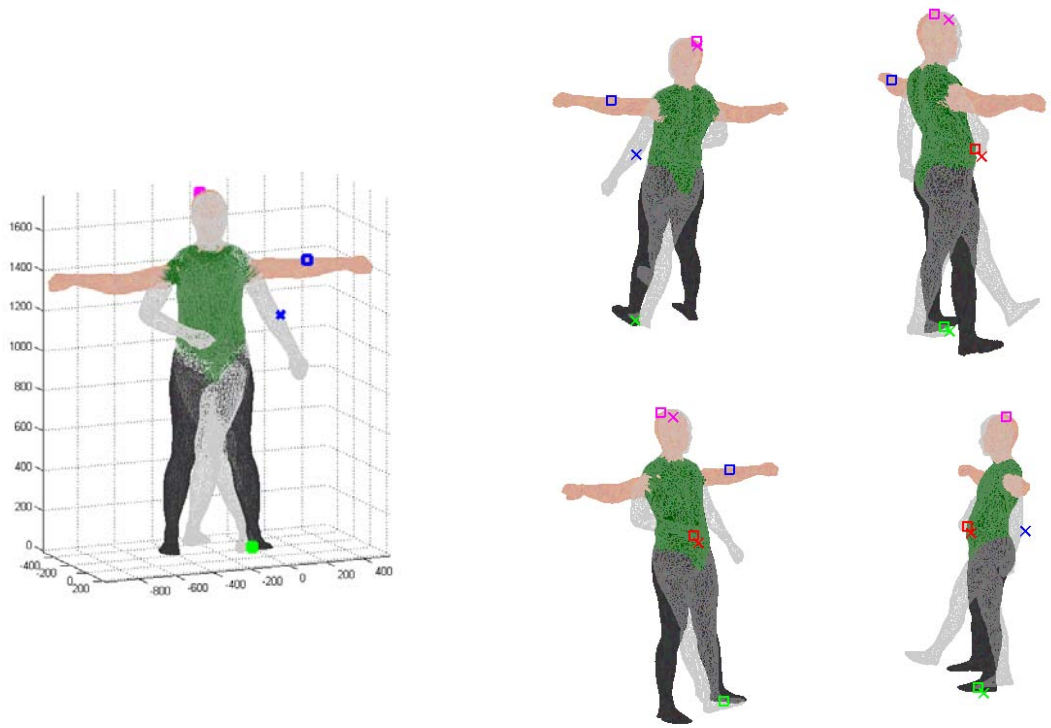
(a) (b)

Figure 3.8.: *Original texture map (a) and the texture map derived from the current pose estimate (b).*

derived from a frame where our pose estimate is deemed accurate, for example by using a good result of the particle filter.

Suppose that we have an estimate for the pose of the mesh, as is shown in gray in Fig. 3.9(a). This estimate may, for example, come from a prediction step, or it could be the pose estimate from the previous frame, or the estimate at any "outer" or "inner" Levenberg-Marquardt iteration of the algorithm described in the previous chapter. The situation in Fig. 3.9(a) is exaggerated, since the estimate rarely differs that much from the true pose, but it serves to show some interesting cases, which will be described shortly. Cases like that, where large differences between the estimate and the true pose exist, might also occur when sampling poses from a prediction, if the standard deviations used during sampling are large. Fig. 3.9(b) shows the projections of the pose estimate, overlaid onto the original camera images. The camera setup for this example is the same as in the HUMANEVA sequence.

Based on the current estimate of the mesh pose, we proceed in estimating the texture map, as described in the previous section. The texture map generated by the camera images of Fig. 3.7 and the (wrong) pose estimate of Fig. 3.9(a) is shown in Fig. 3.8(b). Notice that the texture is more or less correct inside the torso region, where the pose estimate is close to the correct pose, but in the hand and leg regions the texture values



(a) (b)

Figure 3.9.: *The true pose of the mesh, shown with the true mesh colors, and the current estimate, shown in gray. (a) Mesh location in 3D space, (b) Projections of the true pose and the pose estimate as seen from the 4 cameras.*

are wrong, as the color on the texture map comes from either the white background (eg. inside the leg regions of the texture map), or from another body part (eg. in the left hand, where the texture detected comes from the leg).

The algorithm then proceeds in calculating optical flow correspondences between the true texture map and the estimated texture map. Since the mapping between vertices and texture coordinates is one-to-one, texture correspondences map, almost directly, to pairs of matching mesh vertices (where “almost" refers to the fact that, since the resolution of the texture map is larger than the number of vertices, and the colors on the texture map are interpolated from the colors of the vertices, there might not be a corresponding vertex for every point on the texture map; its corresponding vertex is then taken as the one whose texture coordinates are closest to the point). By projecting the mesh vertices for which a texture correspondence was found onto the camera images, we obtain 2D correspondences in the image domain.

Let’s focus on the magenta coloured points in Fig. 3.8 and Fig. 3.9, which represent points on the top of the head of the 3D model. Assume that a correct optical flow match is found between the “x” shaped point and the square shaped point (we will call it the “o” point). Since we assume that each vertex has its own fixed color, this means that the color found

on the image projections of the "o" vertex is the color that should be found on the image projections of the "x" vertex. This in turn means that the "x" vertex should be located in 3D on the current location of the "o" vertex, instead of its current estimated location. This gives us a correspondence for the "x" vertex in 3D (see Fig. 3.9(a)), the source location being the 3D location of the "x" vertex and the target location being the 3D location of the "o" vertex, both based on the current pose estimate. By projecting these correspondences onto the image space, we obtain 2D correspondences, shown in Fig. 3.9(b).

The 2D correspondences are recorded for the 3 cameras that have the best view of the vertex at hand, i.e., for each match in the texture domain, 3 correspondences are stored and used during the optimization. Computing the optical flow on the texture domain functions similarly to finding optical flow correspondences in the image domain, except that now we can find correspondences between two pixels regardless of their visibility from a specific camera (essentially like computing 3D optical flow on the mesh vertices directly, like in [19]).

To assess how this algorithm might help with correcting the pose estimate, it is useful to distinguish between the following possible cases of correspondence:

1. **Correct Correspondence** : In this case, a correspondence is found on the texture map between two vertices belonging to the same body part. This is the case demonstrated in the previous paragraph. In this case, the algorithm indeed helps with locally correcting the head pose. This case appears mostly when the estimate for a body part is relatively close to the true part's pose.

2. **Occlusion with Body Part of Different Color** : This case is demonstrated by the red coloured points in Fig. 3.9 and Fig. 3.8. Here, the arm is located very close to the torso, and the four camera setup does not manage to discriminate between the two, since in all the camera images the projection of the arm is located entirely inside the projection of the torso. Thus the texture assigned to the arm is the same as the texture assigned to the torso. Now, if a texture correspondence is found between the "x" point on the torso and the "o" point on the hand, as shown in Fig. 3.8, then the resulting 3D and 2D correspondences will try to move the torso vertex towards the location of the arm vertex, which is correct in this case (actually in this case it is redundant, since the torso vertex is already located where it should be, i.e. near the arm; but it might help more if the location estimate for the torso vertex was also a bit off).

3. **Occlusion with Body Part of Similar Color** : This is the case with the green points in Fig. 3.9 and Fig. 3.8. This is similar to the previous case in the sense that it is also a case of occlusion, but now the occluding body parts are of similar color. This case might occur when for example the subject crosses their feet. Here this approach will actually worsen the estimate, because the optimization will try to keep the "x" vertex close to the current location of the "o" vertex, which means that it will try to keep the leg in its current(wrong)location. In the case of the crossing legs, a wrong correspondence, eg. one between two vertices, one belonging to the right and the other to the left foot, might lead to the crossing not being detected

by the system.

4. **Missing Optical Flow Target** : This is the case of the blue points in Fig. 3.9 and Fig. 3.8. Here, the projections of the estimated mesh fall entirely outside the correct projection, and therefore the texture captured at these points belongs to the background (hence the white regions in the estimated texture). The true texture provides no information about these points, since the true texture map only contains the colors of the mesh and not the background. Therefore the algorithm is of no use in this case. However, if we run the algorithm more than one times, then the estimate for the regions close to the border between foreground and background (in the example, the regions near the shoulders) might start to improve, which might lead to the estimate for the regions further away (eg. the arms) to also be gradually corrected after some time. To avoid having texture correspondences between mesh vertices and points in the background, which might mess up the estimate, we chose to use the segmentation masks to discard texture regions that are generated by image parts outside the segmentation; however, since the segmentation is only coarse, a few background pixels will almost certainly still be included during the texture generation.

After the texture correspondences have been found and projected to the image space, an outlier detection is performed to ensure that only correct matches are included in the optimization. This is done by calculating, for each camera view, the average 2D motion induced on each bone by the correspondences, i.e. the mean 2D distance between the target and the source location of any vertex of a bone for which a correspondence was found in that view. Then, any correspondence whose distance between source and target is larger than 112% of the mean motion is discarded. Furthermore, to prevent cases where a well aligned bone is forced to change its (correct) pose due to a correspondence that was found for its parent bone, even though no correspondence for that bone itself was found, it is a good idea to also include silhouette correspondences instead of only the texture ones. This ensures that any bone that is initially well aligned with its silhouette will not deviate from its position. Of course this means that the quality of the output will also depend on whether the silhouette correspondences are reliable; as we have seen they might sometimes lead the system to failure. The relative weights of the texture vs. the silhouette correspondences inside the optimization is a parameter that has to be tested against beforehand, to ensure a good tradeoff.

Some results are shown in Fig. 3.11 and Fig. 3.10, for frame 95 of the HUMANEVA sequence. Here, the pose estimate from the particle filter was used as the initial estimate, and 10 iterations of Levenberg - Marquardt with texture correspondences were applied. Every 3 iterations, a set of silhouette correspondences was also used in the optimization. Fig. 3.11 shows some (10% of the total number) of the optical flow correspondences found in the texture domain, between the true texture map (the second column in the figure) and the current estimate of the texture map (the first column). The correspondences are colored based on the texture region to which they belong in the true texture map (where each texture region roughly corresponds to a single bone, see Fig. 3.1). We have omitted the straight lines connecting pairs of optical flow matches for clarity; however, we notice

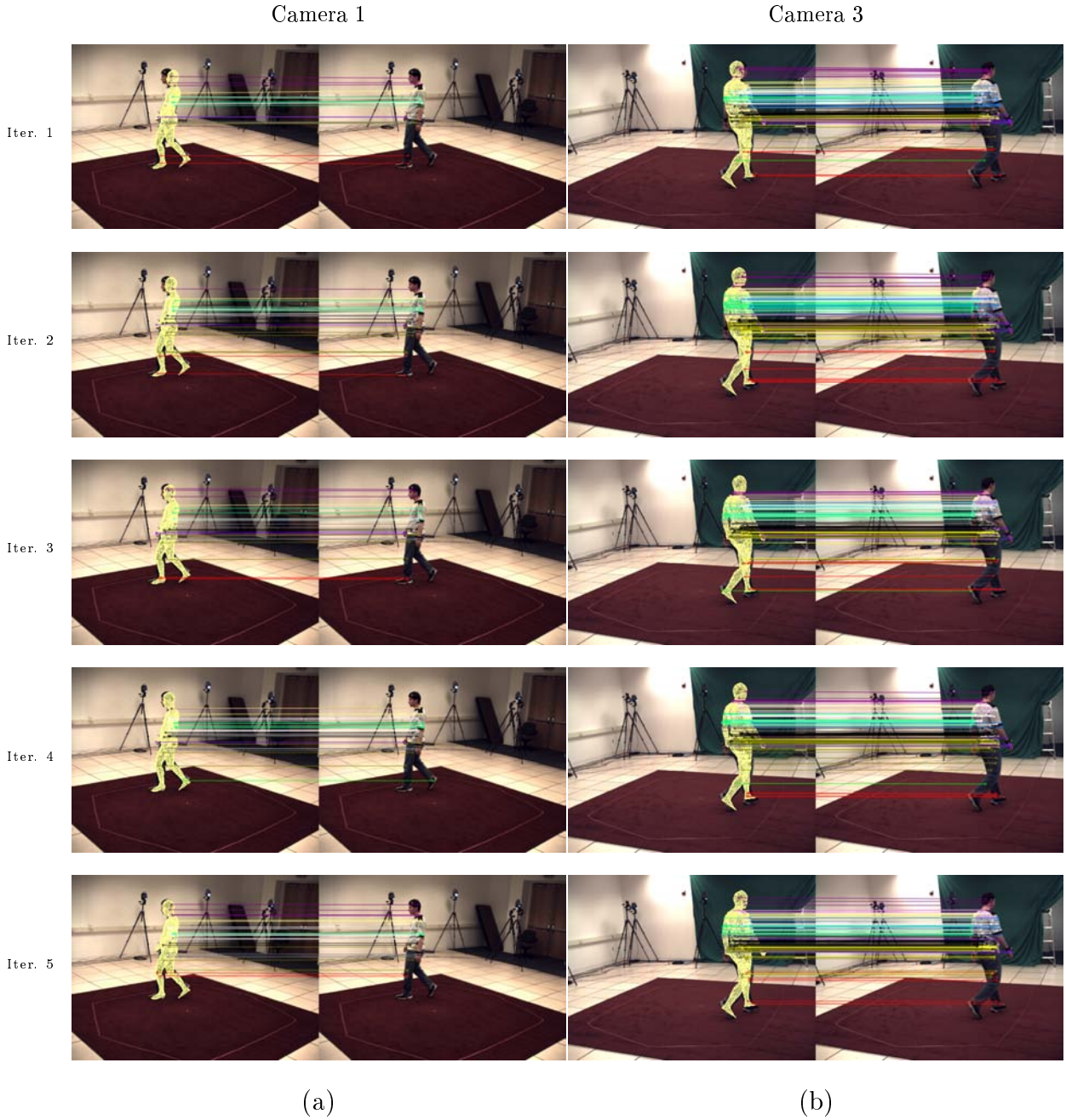


Figure 3.10.: *HUMANEVA: Texture Correspondences on the image domain for frame 95, for 5 consecutive iterations of the optimization. The correspondences are colored according to the bone to which they belong in the true texture map (see 3.1). (a): Point of view of Camera 1, (b): Point of view of Camera 3*

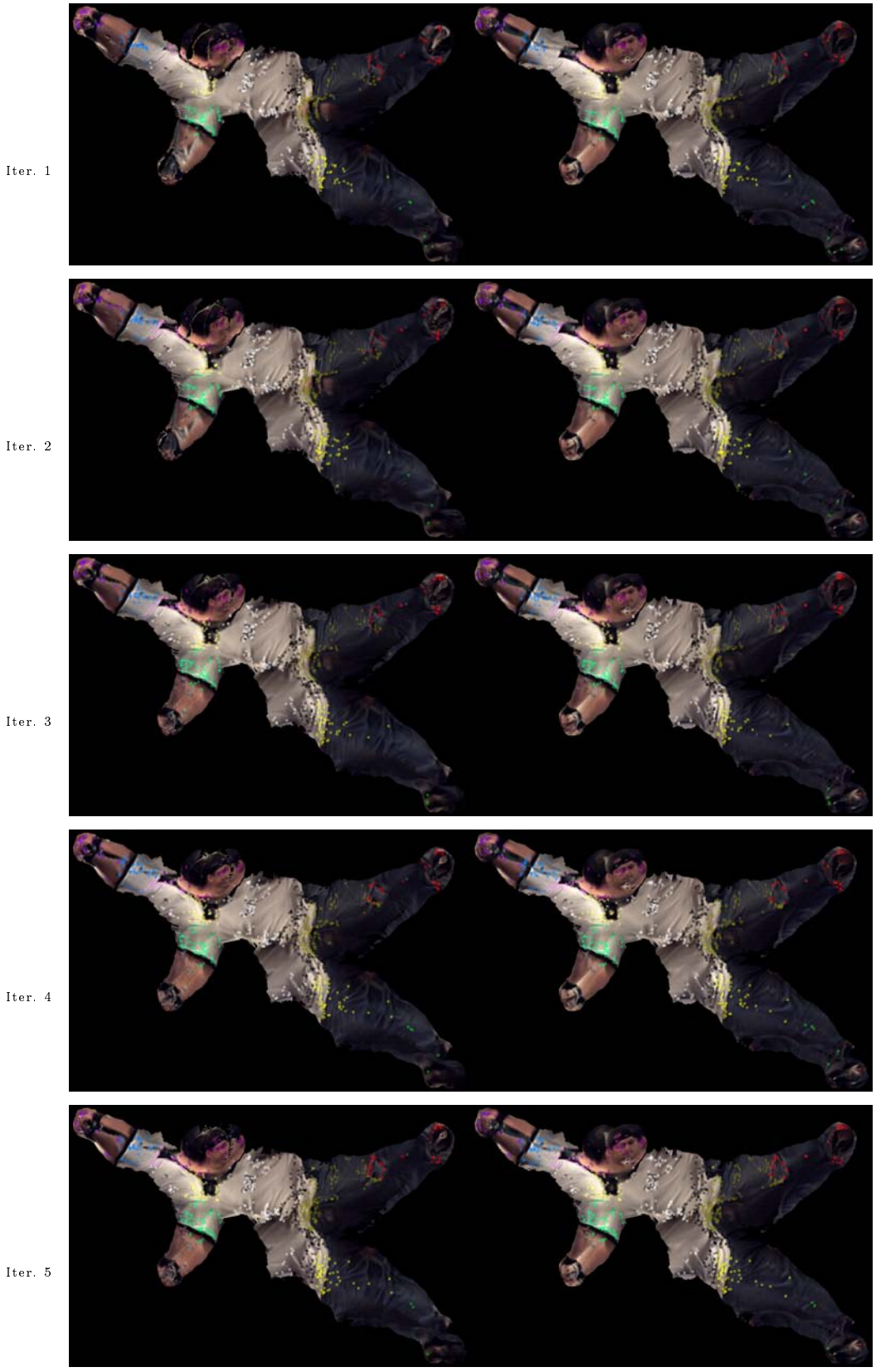


Figure 3.11.: *HUMANEVA: Texture Correspondences of Fig. 3.10, this time on the texture domain. (a): Current Estimate for the texture map, (b): True Texture map. The lines have been omitted to avoid excessive clutter in the images.*

Ground Truth Errors (mm)					
	Mean	Median	Std	Min	Max
LO	210.51	233.46	143.26	45.61	520.98
LO + T	127.36	137.79	61.28	44.97	229.99
30Part	64.67	63.39	8.90	44.09	94.24
30Part + S	61.36	58.18	9.97	45.80	98.55
30Part + ST	60.18	57.62	9.94	43.69	102.66

Table 3.1.: *The ground truth errors for the HUMANEVA sequence (in mm). LO: local optimization with silhouette correspondences only, LO+T: local optimization with silhouette and texture correspondences, 30Part: particle filtering with 30 particles, 30Part+S: particle filtering + final Lev.-Mar. with silhouette correspondences, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

that most of the points belonging to a certain bone in the true texture map end up in the same bone in the estimated texture map, which hopefully means that the procedure will actually try to align that bone a bit better according to its appearance. Notice the "evolution" of the texture map as the iterations of the optimization increase, with the texture map in the final row being more similar to the true texture map than the one in the first row.

Fig. 3.10 shows the correspondences of Fig. 3.11 projected on the images of the cameras that have the best view of each correspondence. These are the actual 2D correspondences used in the optimization framework. We are showing two sets of pairs of images; one is from the point of view of camera 1 and the other is from the point of view of camera 3. Both of the camera images in each pair are the same; the first image of the pair shows the current mesh pose estimate, as it evolves during the 5 iterations, while the second image of the pair shows the 2D targets to which the mesh vertices will move. Notice that the final mesh pose estimate is better aligned to the image than the initial one.

The effect of the texture is more obvious in the case where only local optimization is used. This can be seen by the ground truth errors in Fig. 3.13, and also in Fig. 3.12, which shows some of the cases where the silhouette correspondences lead the system to failure, but the incorporation of the texture correspondences yields a much better (though not entirely correct) pose estimate. The aforementioned tradeoff between silhouette and texture becomes much more obvious and important in this case.

Another impact of the texture is the fact that it helps to slightly smooth out the movement of the head. As we mentioned in the previous chapter, the HMC result for the head was a bit noisy. Using the texture, we can get a smoother motion and more accurate position angles. This is shown in Fig. 3.14. The blue curve is the HMC result where only silhouette

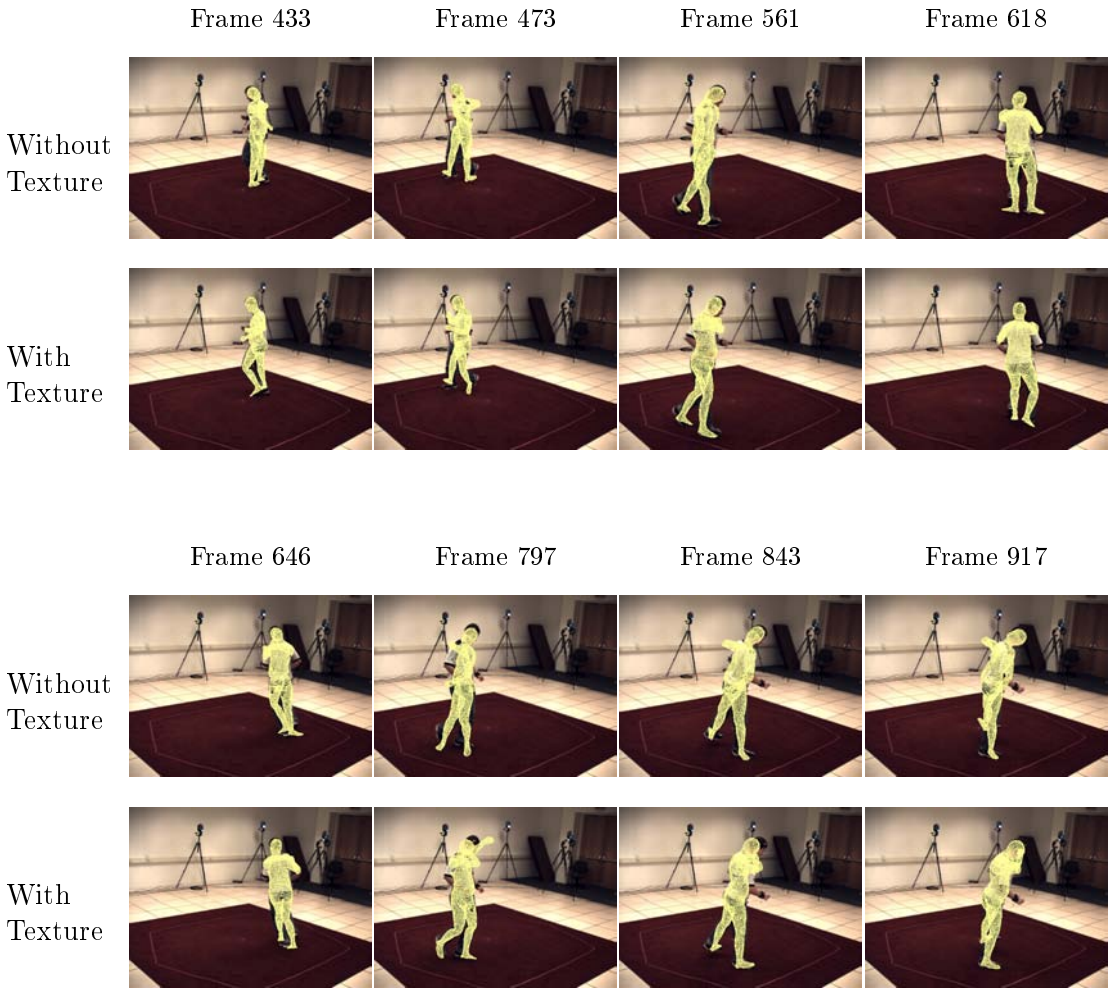
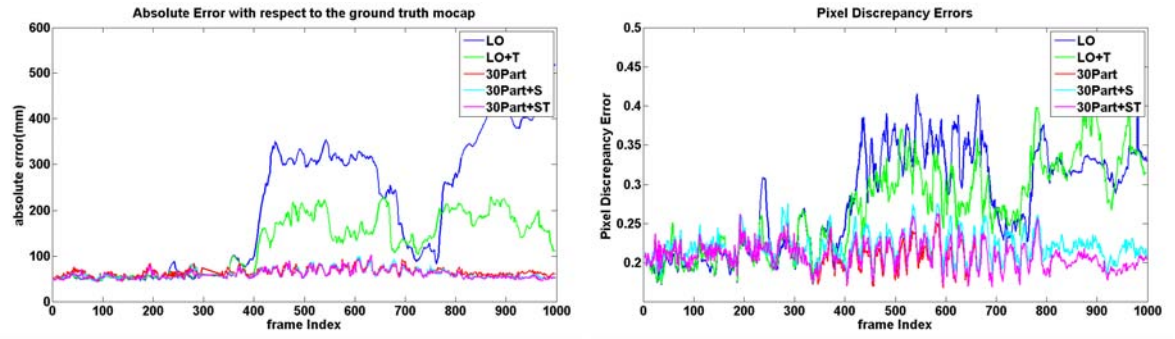


Figure 3.12.: *Comparison of the results of the local optimization with and without texture correspondences, in the frames where the silhouette fails (after frame 300). The figure shows frames 433, 473, 562, 618, 646, 797, 843 and 917.*



(a) (b)

Figure 3.13.: *(a): The average error per frame with respect to the ground truth for the marker positions, in millimetres, for all the used methods. (b): The pixel discrepancy errors. LO: local optimization with silhouette correspondences only, LO+T: local optimization with silhouette and texture correspondences, 30Part: particle filtering with 30 particles, 30Part+S: particle filtering + final Lev.-Mar. with silhouette correspondences, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

Pixel Discrepancy Errors					
	Mean	Median	Std	Min	Max
LO	0.2787	0.2899	0.0639	0.1719	0.4277
LO + T	0.2694	0.2673	0.0578	0.1728	0.4103
30Part	0.2066	0.2062	0.0157	0.1677	0.2623
30Part + S	0.2191	0.2180	0.0178	0.1751	0.2759
30Part + ST	0.2086	0.2073	0.0166	0.1689	0.2630

Table 3.2.: *The pixel discrepancy errors for the HUMANEVA sequence. LO: local optimization with silhouette correspondences only, LO+T: local optimization with silhouette and texture correspondences, 30Part: particle filtering with 30 particles, 30Part+S: particle filtering + final Lev.-Mar. with silhouette correspondences, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

Distance Transform Errors					
	Mean	Median	Std	Min	Max
LO	10.1818	10.2700	1.4188	6.6260	14.2000
LO + T	9.9118	9.9500	1.3873	6.5010	13.7500
30Part	8.9620	8.9520	1.2125	6.0230	12.7900
30Part + S	9.1923	9.1570	1.1947	6.1900	12.9600
30Part + ST	8.9864	8.9815	1.2154	6.0230	12.7900

Table 3.3.: *The distance transform errors for the HUMANEVA sequence. LO: local optimization with silhouette correspondences only, LO+T: local optimization with silhouette and texture correspondences, 30Part: particle filtering with 30 particles, 30Part+S: particle filtering + final Lev.-Mar. with silhouette correspondences, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

correspondences were used in the final optimization iterations, while the black curve is the HMC result where both silhouette and texture are present during the final optimization iterations. Notice that the black curve is a bit smoother than the blue one.

Fig. 3.13, Table 3.1, Table 3.2 and Table 3.3 show the ground truth, pixel discrepancy and distance transform errors for the various methods used, for the HUMANEVA sequence. Although small (since the particle filter estimate with the final silhouette iterations was already pretty good), the use of texture succeeds in reducing those errors a little bit. It is interesting to note that although the ground truth reduces from the 30Part case to the 30Part+S or the 30Part+ST cases, the distance transform and pixel discrepancy errors remain practically constant or increase a bit. This verifies the fact that, due to imperfect segmentation, these two error metrics are not ideal in assessing the quality of a result. Also, it happens that in ambiguous cases (with regards to the sihlouette), such as the crossing of the legs around frame 300 of the HUMANEVA sequence the silhouette correspondences tend to resolve the ambiguity by "averaging", e.g., by placing both feet in the middle region,while the particle filtering approach yields a more correct solution. This also partly explains the slight increase in the pixel discrepancy/ distance transform.

The same results for the JUGGLER ORANGE sequence are shown in Table 3.4 and Table 3.5. As expected from the low quality of the "true" texture map, the improvement in the results is almost negligible. Apart from the fact that most of the texture map is already too blurred, the skin regions of the subject do not differ a lot in color from his shirt. This leads to only a few (if any) correct correspondences being found for the head or the arms, as can be seen in Fig. 3.15 : of the few correspondences found for the arms, only 1 or 2 are correct, and do not lead to the correction of the arm pose. Instead, most of the valid correspondences are located in the torso region. Thus the pose of the body extremities is not corrected, and since the torso region is almost always relatively

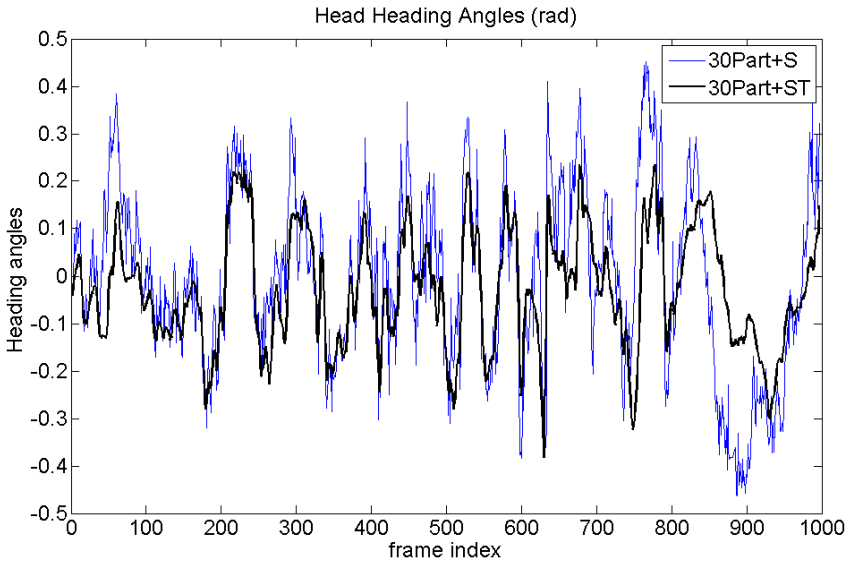


Figure 3.14.: *The heading angles for the head, as estimated with and without texture in the final optimization after the HMC.*

well aligned, the impact of the texture correspondences in the result is very small. The situation is exacerbated by the fact that the 3D model is not perfectly matched to the subject. The slight drop in the errors is due to some slight improvement of the overall pose, mostly caused by changing the translation of the torso a little bit.

Table 3.6 summarises the running times for the various versions of our system. The local optimization only case corresponds to 20 iterations of Levenberg - Marquardt with silhouette and optical flow correspondences, and the same number of iterations was used for the local optimization + texture experiments. The particle filter result corresponds to 30 particles with 3 Levenberg - Marquardt iterations each and 5 outer iterations. The particle filter + texture result is the same as the previous with the addition of 10 more

Pixel Discrepancy Errors					
	Mean	Median	Std	Min	Max
LO	0.3397	0.2721	0.1400	0.1871	0.8231
30Part	0.2429	0.2286	0.0316	0.1769	0.6369
30Part + ST	0.2259	0.2222	0.0301	0.1728	0.6333

Table 3.4.: *The pixel discrepancy errors for the JUGGLER ORANGE sequence. LO: local optimization with silhouette correspondences only, 30Part: particle filtering with 30 particles, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

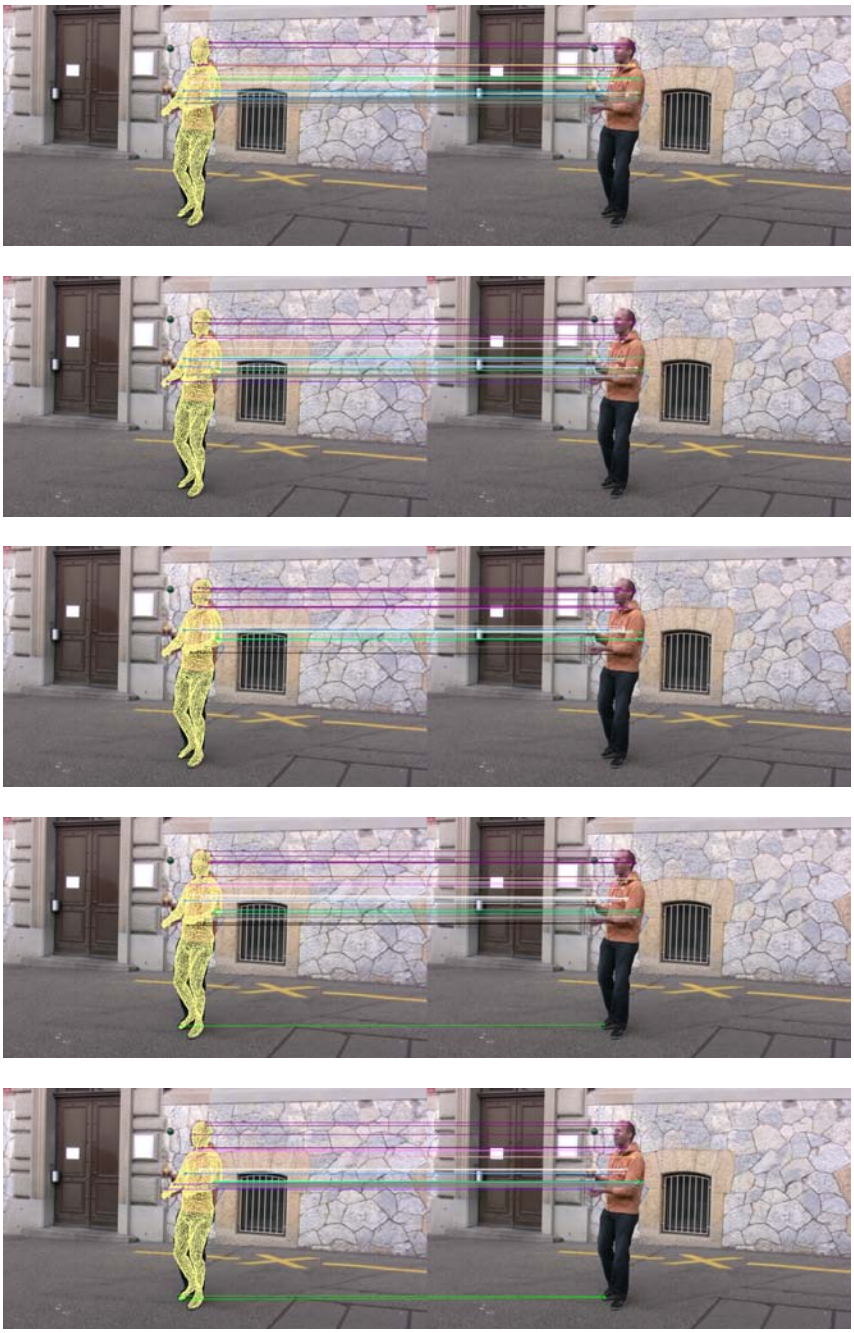


Figure 3.15.: *Texture Correspondences on the image domain, for 5 consecutive iterations of the optimization, for frame 8 of the JUGGLER ORANGE sequence.*

Distance Transform Errors					
	Mean	Median	Std	Min	Max
LO	12.0027	11.1900	2.8519	7.7670	21.4600
30Part	10.1943	9.9930	1.4081	7.4330	19.2700
30Part + ST	9.8919	9.8740	1.1947	7.3990	18.5500

Table 3.5.: *The distance transform errors for the JUGGLER ORANGE sequence. LO: local optimization with silhouette correspondences only, 30Part: particle filtering with 30 particles, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

iterations of Levenberg - Marquardt with silhouette and texture correspondences. The texture correspondences were not used for the updates of the particle states, due to the much longer running time that this would incur.

3.3. Minimizing the Texture Differences by Optimization

Another option when it comes to exploiting the information provided by the texture map is to directly use the dependence between the mesh pose and the generated texture, and try to minimise the difference between the estimated texture map and the true texture map. This section explains this approach.

Assuming, like before, that a good estimate for the true texture map is available, we would like the mesh at the current frame to have the same vertex colors as the ones specified by that true texture map. If we denote the set of mesh pose parameters (angles and translations) by θ, then this can be accomplished by minimizing the following function:

$$F(\theta) = \sum_{v=1}^{V} \{T_{estimated}(\vec{uv}_v) - T_{true}(\vec{uv}_v)\}^2 \tag{3.1}$$

where v is the vertex index, V the total number of vertices in the mesh and $\vec{uv}_v$ are the fixed texture coordinates associated with vertex v. We can minimise this function by using optimization (eg. the Levenberg-Marquardt procedure used before), but we will need the derivative of the above function with respect to the mesh pose θ.

We will use the simplified weighted average version of the texture, since the blending version involves calculating distance transforms as the weights and therefore not differentiable. If we denote the camera index by c, the total number of cameras by C, the camera

Running Times per Frame		
	HUMANEVA	JUGGLER ORANGE
LO	0'9"	0'20"
LO + T	0'14"	0'32"
30Part	5'40"	6'50"
30Part + S	5'48"	7'03"
30Part + ST	5'52"	7'10"

Table 3.6.: *The running times of the various algorithms, for an Intel Core i5 CPU 750 @2.67 GHz, with 8 GB of RAM. LO: local optimization with silhouette correspondences only, LO+T: local optimization with silhouette and texture correspondences, 30Part: particle filtering with 30 particles, 30Part+S: particle filtering + final Lev.-Mar. with silhouette correspondences, 30Part+ST: particle filtering + final Lev.-Mar. with silhouette and texture correspondences.*

image by I_c (any of the R, G, B channels ; the Jacobian of the texture with respect to a single degree of freedom is a 3×1 vector) and the 2D projection of vertex v as viewed from camera c as $p_v^c(\theta)$, then we have

$$T_{estimated}(\vec{uv}_v) = \sum_{c=1}^{C} w_c I_c\left(\vec{p}_v^c(\theta)\right) \tag{3.2}$$

where w_c are the respective visibility weights, which as before are proportional to the angle between the mesh normal at vertex v and the vector from the camera centre to the 3D location of the vertex, normalised so that they add up to one:

$$w_c = \frac{\vec{n}_v(\theta)^T \frac{\vec{O}_c - \vec{P}_v(\theta)}{\|\vec{O}_c - \vec{P}_v(\theta)\|}}{\sum_{c'=1}^{C} \frac{\vec{n}_v(\theta)^T (\vec{O}_{c'} - \vec{P}_v(\theta))}{\|\vec{O}_{c'} - \vec{P}_v(\theta)\|}} \tag{3.3}$$

where $\vec{O}_c$ is the camera centre, $\vec{P}_v(\theta)$, $\vec{n}_v(\theta)$ respectively are the 3D location and normal of mesh vertex v.

Both the mesh normal and the 3D vertex locations and their projections are differentiable with respect to the pose. The derivation of the Jacobian of the texture difference function (3.1) is straightforward but quite lengthy and is shown in Appendix A. Note that for the computation of the texture Jacobian we omit the fact that whenever a weight w_c is negative, it should be set to zero; the true expression for the weights is

$$w_c = \frac{A_c}{\sum_{c'=1}^{C} A_{c'}}, \qquad A_c = \max\left(0, \vec{n}_v(\theta)^T \frac{\vec{O}_c - \vec{P}_v(\theta)}{\|\vec{O}_c - \vec{P}_v(\theta)\|}\right) \tag{3.4}$$

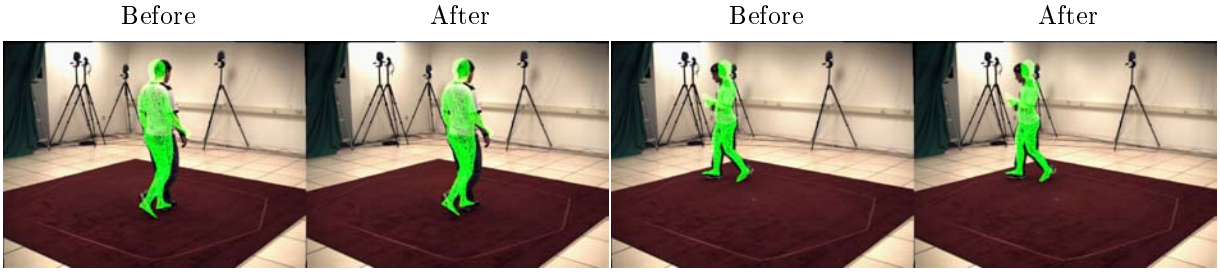


Figure 3.16.: *Failure of the optimization with the texture Jacobian to align the 3D model. The starting position was far from the correct one in the region of the head, and there was obviously a local minima among the colors of the background. (a): Frame 23, (b): Frame 515*

This, however, is a not differentiable form, so instead we use the form in equation (3.3). Equation (3.4) is however taken into account when using the pose estimate to calculate the texture colors at the vertices, a function also needed by the Levenberg - Marquardt method. After the Jacobian is calculated, the only thing needed for the Levenberg - Marquardt method to work is the desired color values at the vertices; these are derived from the true texture map. Using the Jacobian and the target R, G, B values at the vertices, the Levenberg-Marquardt will (hopefully) converge to the minimum of the function (3.1) .

One problem with this method is its susceptibility to local minima. Due to the fact that the human clothing and skin mostly consists of regions of the same or very similar colors (roughly, one region corresponding to the blouse, two regions for the trousers and a few skin regions), all of the poses that place the mesh projections inside these regions in the images have roughly the same textures, and therefore roughly the same (small) differences with respect to the true texture map. This means that the algorithm can easily get stuck in an incorrect estimate (a local minimum of the texture difference) if it produces relatively small differences. In the indoor scene example, the same problem exists also with the background, which also consists of regions of more or less the same color. This local minima sensitivity also means that the initial condition from which we start the optimization has to be relatively close to the true minimum, otherwise we might bump into a local minimum within the background region. This is illustrated in Fig. **??**. In our experiments, we use the HMC result (without the final local optimization iterations) and tried to see whether using the Jacobian of the texture could help to better align some of the body parts.

Problems can also occur if our true texture map is not entirely accurate, which might well be the case since we are only using a good particle filter pose estimate to generate the true texture, and not, for example, a pretextured mesh. In that case, the optimization will tend to move the mesh not to the most correct pose, but to the pose that yields the texture map that is closest to the true texture; thus, parts that were a bit misaligned in

that first pose estimate from which the true texture was generated (for example, parts of the head that are assigned white-ish background colors in the texture map of in image Fig. 3.4) , will continue to be deliberately misaligned in the rest of the sequence.

Some results are shown in Fig. 3.17. In these experiments, the output of the particle filter was used as an initial solution, and a few iteration of Levenberg - Marquardt were applied, where only texture information was used, by supplying the desired vertex colors to the system and evaluation the texture estimation equation (3.2) and the Jacobian as found in Appendix A. Notice the better alignment of some of the body extremities, such as the head or the hands. This is quite useful especially in the case of the head since the segmentation is rather coarse segmentation and it is difficult to extract the exact pose of the head, since the projection of the head looks roughly the same from most angles.

This is perhaps more obvious in Fig. 3.18. Here, we have colored the mesh vertices with the colors from the texture map, and then projected the mesh to the image space. Notice that the texture on the head and the hands look better in the second column than in the first, which means that the projections of the mesh are better aligned to the image. The overall result with the texture correction was also found to be smoother than the HMC result.

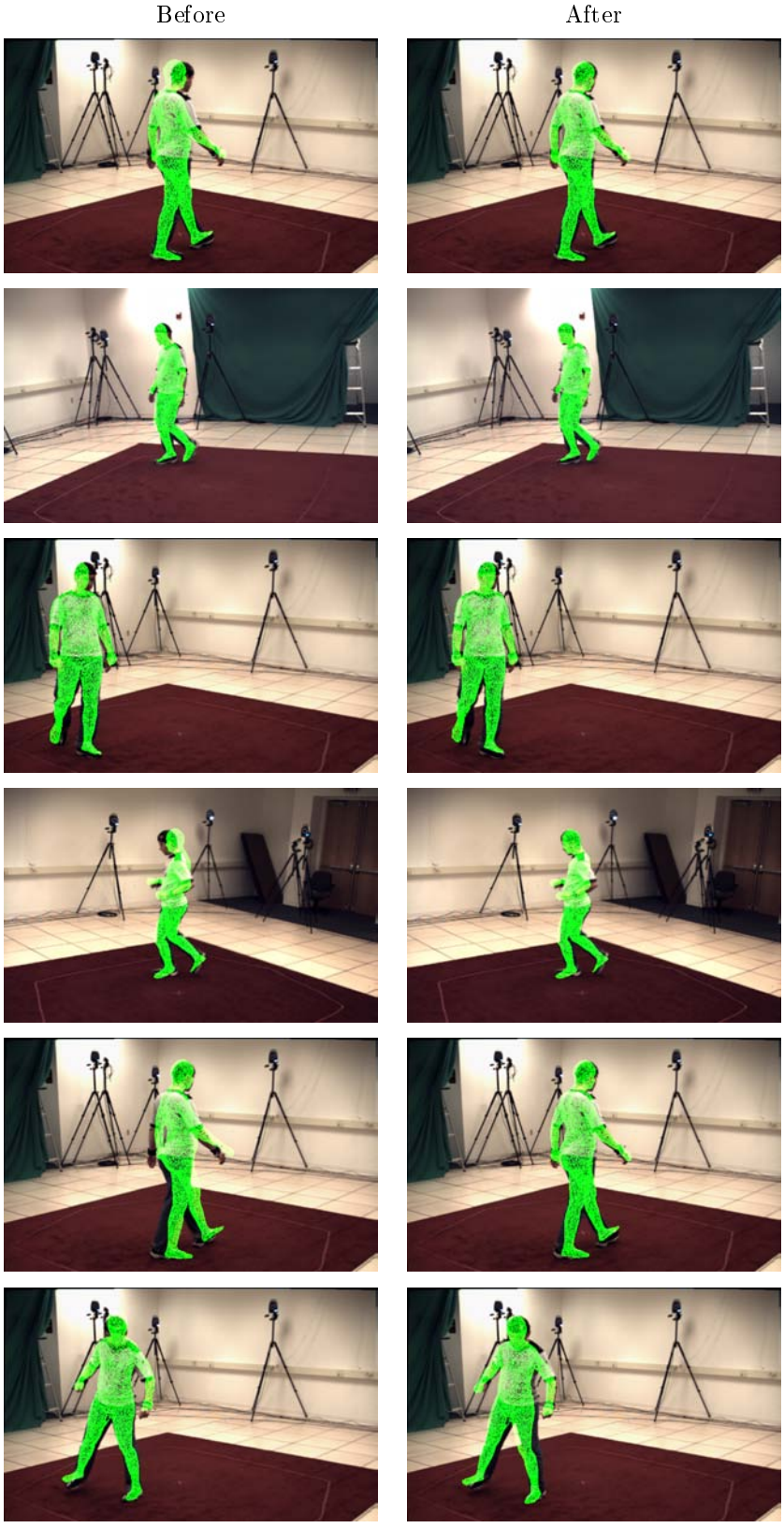


Figure 3.17.: *Using the Jacobian of the texture. (a): Results of the HMC filter, (b): Results after a few iterations of Lev.-Mar. optimization with the texture Jacobian.*

Figure 3.18.: *The 3D mesh colored with the colors from the estimated texture map. (a): Results of the HMC filter, (b): Results after a few iterations of Levenberg-Marquardt optimization with the texture Jacobian.*

4

Conclusion and Outlook

In this thesis we tackled the problem of markerless optical motion capture, in both controlled and uncontrolled environments. As shown in the results, the latter is a much more complex case to handle, since the motions and the setup can make it difficult to extract useful information from the scene that can be used to estimate the pose of the moving subject.

The major goal of this work was to try and incorporate multiple streams of information in a unified way. Starting from an existing system, which took into account not only the skeletal motion but also the non-rigid deformations of the human body and considered both optical flow and silhouette information, we extended it so as to be able to handle more complex cases of faster motion, camera movement and imperfections in the segmentation input. We also showed two ways in which the appearance of the moving subject can be used to add robustness to the system.

The experimentation with a controlled, indoor scene showed that the use of particle filtering, in combination with local Levenberg - Marquardt optimization can be crucial in finding a solution in cases where the segmentation is not perfect or there is an ambiguity in the pose caused by the lack of enough cameras in the setup, or a local minimum in the error function used by the optimization. This sequence also helped to evaluate the merits of using texture as an additional source of information; we showed that although the particle filter can handle the indoor scenario pretty well, the texture helps to better align some of the moving parts. It also helps to correct some of the errors cause by the silhouette correspondences.

Testing the system with the much more challenging outdoor scene, we concluded that even a small number of particles can handle a large number of degrees of freedom with

relative success (30 particles for a system with 34 degrees of freedom). The use of the distance transform error helps to evaluate the possible solution and discriminate between local and global minima of the silhouette errors. However, the running time of the system rose from a few seconds (30) to a few minutes (6).

Apart from the drawback of the much longer execution time, another drawback of the particle filtering system remains its great dependence on the quality of the segmentation. If the segmentation is too blurry, and no information is provided with regards to the detailed pose of the body extremities (hands and legs) then the system can get lost. The use of texture partly alleviates this problem, however it is itself very sensitive to local minima and can get some of the body parts mixed up, if they have similar colors. An additional concern with the use of the texture is its sensitivity with respect to what we consider the "true" texture map. Unless a pretextured mesh is used, it is difficult to find a perfectly good pose estimate to extract the "true" texture from. Even in the case of a pretextured mesh there might be problems in outdoor scenarios due to changes in the illumination conditions.

Future work to improve the system might be related to (but not restricted to) the following directions

- Combine the texture Jacobian inside the particle filtering framework. This has not yet been tested and might help to reduce the number of particle update iterations required.
- Use the texture as an evaluation function in the same way as the distance transform is currently used, i.e. assess the quality of the solution by computing the difference of the texture derived by this solution and the "true" texture.
- Try to incorporate the distance transform into the optimization framework, in the same way as was done for the texture. For this, the "true" distance transform should be estimated from the segmentation and evaluated at the mesh vertices, and then optimization should be performed as with the texture. Of course, the distance transform itself is not differentiable, therefore some sort of differentiable approximation could be used (eg. an approximation of the "true" distance transform with a Gaussian mixture).
- Try to improve the estimate of the "true" texture by updating it every time a significant change is detected. This is fairly easy to do by updating the color on the texture map whenever the majority of the cameras see that a certain mesh point has a color vastly different from the color it seems to have in the "true" texture map. This update would also be particularly useful in the cases where there is no "true" texture map available, or there are regions in the texture map that are initially invisible from all cameras, but become visible later in the sequence. Also, in this way we ensure that the cameras see the same colors for the same region of the mesh, thus we can obtain less blurry texture maps.
- Use the texture Jacobian approach to improve the "true" texture, especially for the outdoor sequence. Assume that cameras 1 and 2 disagree about the colors that

they see for the same region (eg. the head). Then, by using the texture Jacobian to minimise the pairwise differences between the two texture images (one for each camera) we can find that pose that makes the two texture images more similar in the head region. By minimising the pairwise difference, we will be able to correct the pose of the faulty regions and therefore also improve the texture.

- Improve the prediction of the pose for the next frame by using something more efficient and robust than similar linear/quaternion extrapolation, eg. a Kalman filter.

A

The Texture Jacobian

A.1. Notation

- θ : mesh pose
- v : vertex index, V : number of vertices
- c : camera index, C : number of cameras
- K_c: projection matrix of camera c
- $\vec{P}_v(\theta)$: 3D point location of vertex v for configuration θ
- $\vec{p}_v^c(\theta)$: 2d projection of $\vec{P}_v(\theta)$ for camera c
- $I_c(\vec{p})$: image taken from camera c, evaluated at 2d location $\vec{p}$
- $\vec{O}_c$: center of camera c
- $\vec{uv}_v$: texture coordinates for vertex v
- $\vec{n}_v(\theta)$: mesh outward normal at vertex v
- $T_{true}(\vec{uv}_v)$: correct texture map, evaluated at the texture coordinates of vertex v

A.2. Function to minimize

$$F(\theta) = \sum_{v=1}^{V} \left\{ \sum_{c=1}^{C} w_c I_c\left(\vec{p}_v^{\,c}(\theta)\right) - T_{true}(u\vec{v}_v) \right\}^2 \tag{A.1}$$

$$w_c = \frac{\vec{n}_v(\theta)^T \frac{\vec{O}_c - \vec{P}_v(\theta)}{\|\vec{O}_c - \vec{P}_v(\theta)\|}}{\vec{n}_v(\theta)^T \sum_{c'=1}^{C} \left(\frac{\vec{O}_{c'} - \vec{P}_v(\theta)}{\|\vec{O}_{c'} - \vec{P}_v(\theta)\|} \right)} \tag{A.2}$$

where I_c is any of the R,G,B channels of the texture image (we will have 3 entries per vertex)

For the Levenberg-Marquardt method we specify the target vertex colors, and we want to ensure that

$$f(v,\theta) = \sum_{c=1}^{C} w_c I_c\left(\vec{p}_v^{\,c}(\theta)\right) = T_{true}(u\vec{v}_v) \quad , \qquad \forall v \in \{1,2,...,V\} \tag{A.3}$$

Thus we need to calculate the derivative $\frac{\partial f(v,\theta)}{\partial \theta}$ for all v and θ. Stacking up these derivatives horizontally for the N degrees of freedom and vertically for the 3 channels and the V vertices yields the $3V \times N$ Jacobian matrix of the function f.

A.2.1. Jacobian

The product rule gives

$$\frac{\partial f(v,\theta)}{\partial \theta} = \sum_{c=1}^{C} \frac{\partial \left[w_c I_c\left(\vec{p}_v^{\,c}(\theta)\right) \right]}{\partial \theta} = \sum_{c=1}^{C} \frac{\partial w_c}{\partial \theta} I_c\left(\vec{p}_v^{\,c}(\theta)\right) + w_c \frac{\partial I_c\left(\vec{p}_v^{\,c}(\theta)\right)}{\partial \theta} \tag{A.4}$$

Starting from the second term of the sum, we can write

$$\frac{\partial I_c\left(\vec{p}_v^{\,c}(\theta)\right)}{\partial \theta} = \frac{\partial I_c}{\partial \vec{p}_v^{\,c}(\theta)} \frac{\partial \vec{p}_v^{\,c}(\theta)}{\partial \theta} = \nabla I_c|_{\vec{p}_v^{\,c}(\theta)} \frac{\partial \vec{p}_v^{\,c}(\theta)}{\partial \theta} \tag{A.5}$$

Let $\tilde{p}_v^{\,c}(\theta) = K_c P_v(\theta)$, and . Then

$$\vec{p}_v^{\,c}(\theta) = \frac{1}{\tilde{p}_v^{\,c}(\theta)[2]} \begin{bmatrix} \tilde{p}_v^{\,c}(\theta)[0] \\ \tilde{p}_v^{\,c}(\theta)[1] \end{bmatrix} \tag{A.6}$$

$$\frac{\partial \vec{p}_v^{\,c}(\theta)}{\partial \theta} = \frac{1}{\left(\tilde{p}_v^{\,c}(\theta)[2]\right)^2} \begin{bmatrix} \tilde{p}_v^{\,c}(\theta)[2] \frac{\partial \tilde{p}_v^{\,c}(\theta)[0]}{\partial \theta} - \tilde{p}_v^{\,c}(\theta)[0] \frac{\partial \tilde{p}_v^{\,c}(\theta)[2]}{\partial \theta} \\ \tilde{p}_v^{\,c}(\theta)[2] \frac{\partial \tilde{p}_v^{\,c}(\theta)[1]}{\partial \theta} - \tilde{p}_v^{\,c}(\theta)[1] \frac{\partial \tilde{p}_v^{\,c}(\theta)[2]}{\partial \theta} \end{bmatrix} \tag{A.7}$$

and

$$\frac{\partial \tilde{p}_v^{\,c}(\theta)}{\partial \theta} = K_c \frac{\partial \vec{P}_v(\theta)}{\partial \theta} \tag{A.8}$$

and the latter is found by differentiating the transformation matrices of the respective joints. This covers the computations needed for the second term of equation (A.4).

The weights w_c can be written as

$$w_c(\theta) = \frac{A_c}{\sum_{c'=1}^{C} A_{c'}} \tag{A.9}$$

where

$$A_c = \vec{n}_v(\theta)^T \frac{\vec{O}_c - \vec{P}_v(\theta)}{\|\vec{O}_c - \vec{P}_v(\theta)\|} \tag{A.10}$$

Thus for the derivatives of the weights in the second term of equation (A.4) we can write

$$\frac{\partial w_c}{\partial \theta} = \frac{\frac{\partial A_c}{\partial \theta}}{\sum_{c'=1}^{C} A_{c'}} - \frac{A_c}{\left(\sum_{c'=1}^{C} A_{c'}\right)^2} \sum_{c'=1}^{C} \frac{\partial A_{c'}}{\partial \theta} = \frac{\frac{\partial A_c}{\partial \theta} \sum_{c'=1}^{C} A_{c'} - A_c \sum_{c'=1}^{C} \frac{\partial A_{c'}}{\partial \theta}}{\left(\sum_{c'=1}^{C} A_{c'}\right)^2} \tag{A.11}$$

The derivative of the term A_c is as follows

$$\begin{aligned} \frac{\partial A_c}{\partial \theta} &= \frac{\partial \vec{n}_v(\theta)^T}{\partial \theta} \frac{\vec{O}_c - \vec{P}_v(\theta)}{\|\vec{O}_c - \vec{P}_v(\theta)\|} \\ &- \frac{\vec{n}_v(\theta)^T}{\|\vec{O}_c - \vec{P}_v(\theta)\|} \cdot \frac{\partial \vec{P}_v(\theta)}{\partial \theta} \\ &+ \left(\frac{\partial \vec{P}_v(\theta)}{\partial \theta}^T (\vec{O}_c - \vec{P}_v(\theta)) \right) \vec{n}_v(\theta)^T \frac{\vec{O}_c - \vec{P}_v(\theta)}{\|\vec{O}_c - \vec{P}_v(\theta)\|^3} \end{aligned} \tag{A.12}$$

where also the term $\frac{\partial \vec{P}_v(\theta)}{\partial \theta}$ can be found in a straightforward way by differentiating the joint transformation matrices.

The only unknown in the above equation is the derivative of the mesh normal $\frac{\partial \vec{n}_v(\theta)^T}{\partial \theta}$. In order to permorm this differentiation , we approximate the normal by a form that uses the derivatives of the 3D locations of the neighboring bertices. If we assume that vertex v is the shared vertex of its N_v neighboring faces, then the vertex normal at vertex v can be written as an average of the (outward) normal directions of the neighboring faces. These directions can be found by using cross products of those faces' edge directions. Thus

$$\vec{n}_v(\theta) = \frac{\vec{t}}{\|\vec{t}\|}, \qquad \vec{t} = \sum_{i=1}^{N_v} (\vec{P}_{v_i^2} - \vec{P}_v) \times (\vec{P}_{v_i^3} - \vec{P}_v) \tag{A.13}$$

where (v_i^2, v_i^3) are the 2 vertices of face i different from vertex v (arranged in such order that the above cross product is actually the outward and not the inward normal at vertex v).

Therefore, we can write

$$\frac{\partial \vec{t}(\theta)}{\partial \theta} = \sum_{i=1}^{N_v} \left[\left(\frac{\partial \vec{P}_{v_i^2}}{\partial \theta} - \frac{\partial \vec{P}_v}{\partial \theta} \right) \times \left(\vec{P}_{v_i^3} - \vec{P}_v \right) + \left(\vec{P}_{v_i^2} - \vec{P}_v \right) \times \left(\frac{\partial \vec{P}_{v_i^3}}{\partial \theta} - \frac{\partial \vec{P}_v}{\partial \theta} \right) \right] \tag{A.14}$$

where all the terms can be found straightforwardly. Finally,

$$\frac{\partial \vec{n}(\theta)}{\partial \theta} = \frac{1}{\|\vec{t}(\theta)\|} \frac{\partial \vec{t}(\theta)}{\partial \theta} - \left(\frac{\partial \vec{t}}{\partial \theta}^T \vec{t} \right) \frac{\vec{t}}{\|\vec{t}(\theta)\|^3} \tag{A.15}$$

Combining equations (A.15), (A.12), (A.11), (A.8), (A.7), (A.5) and substituting them into equation (A.4) yields the term $\frac{\partial f(c,\theta)}{\partial \theta}$.

Bibliography

[1] L. Ballan, G. J. Brostow, J. Puwein, and M. Pollefeys, *Unstructured video-based rendering: Interactive exploration of casually captured videos*, 29 (2010), p. to appear.

[2] L. Ballan and G. M. Cortelazzo, *Marker-less motion capture of skinned models in a four camera set-up using optical flow and silhouettes*, in 3DPVT, Atlanta, GA, USA, June 2008.

[3] D. Bradley, W. Heidrich, T. Popa, and A. Sheffer, *High resolution passive facial performance capture*, ACM Trans. on Graphics (Proc. SIGGRAPH), 29 (2010).

[4] P. J. Burt and E. H. Adelson, *A multiresolution spline with application to image mosaics*, ACM Trans. Graph., 2 (1983), pp. 217–236.

[5] K. Choo and D. J. Fleet, *People tracking using hybrid monte carlo filtering*, in ICCV, 2001, pp. 321–328.

[6] J. Gall, B. Rosenhahn, T. Brox, and H.-P. Seidel, *Optimization and filtering for human motion capture*, Int. J. Comput. Vision, 87 (2010), pp. 75–92.

[7] 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, Miami, USA, 2009, pp. 1746–1753.

[8] N. Hasler, B. Rosenhahn, T. Thormählen, M. Wand, J. Gall, and H.-P. Seidel, *Markerless motion capture with unsynchronized moving cameras*, in IEEE CVPR, Miami, USA, June 2009, pp. 224–231.

[9] M. Lourakis, *levmar: Levenberg-marquardt nonlinear least squares algorithms in C/C++*. [web page] `http://www.ics.forth.gr/~lourakis/levmar/`, Jul. 2004. [Accessed on 31 Jan. 2005.].

[10] T. B. Moeslund, A. Hilton, and V. Krüger, *A survey of advances in vision-based human motion capture and analysis*, Comput. Vis. Image Underst., 104 (2006), pp. 90–126.

[11] L. Mundermann, S. Corazza, and T. Andriacchi, *Accurately measuring human movement using articulated icp with soft-joint constraints and a repository of articulated models*, IEEE Conference on Computer Vision and Pattern Recognition, (2007), pp. 1 –6.

[12] R. Plankers and P. Fua, *Articulated soft objects for multiview shape and motion capture*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (2003), pp. 1182 – 1187.

[13] E. Poon and D. J. Fleet, *Hybrid monte carlo filtering: Edge-based people tracking*, in MOTION '02: Proceedings of the Workshop on Motion and Video Computing, Washington, DC, USA, 2002, IEEE Computer Society, p. 151.

[14] H.-Y. Shum and R. S. Szeliski, *Inverse texture mapping using weighted pyramid blending*, US Patent, October 2002.

[15] L. Sigal, A. O. Balan, and M. J. Black, *Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion*, Int. J. Comput. Vision, 87 (2010), pp. 4–27.

[16] L. Sigal, S. Bhatia, S. Roth, M. J. Black, and M. Isard, *Tracking loose-limbed people*, in IEEE CVPR, 2004, pp. 421–428.

[17] L. Sigal and M. Black, *Measure locally, reason globally: Occlusion-sensitive articulated pose estimation*, in IEEE CVPR, 2006, pp. II: 2041–2048.

[18] L. Sigal and M. J. Black, *Predicting 3d people from 2d pictures*, in AMDO, 2006, pp. 185–195.

[19] C. Theobalt, J. Carranza, M. A. Magnor, and H.-P. Seidel, *Enhancing silhouette-based human motion capture with 3d motion fields*, in PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, Washington, DC, USA, 2003, IEEE Computer Society, p. 185.

[20] R. Urtasun, D. J. Fleet, and P. Fua, *Temporal motion models for monocular and multiview 3d human body tracking*, Comput. Vis. Image Underst., 104 (2006), pp. 157–177.