Chapter 5

Articulated Tracking

This chapter describes two state-of-the-art probabilistic articulated tracking systems that rely on the pose estimation framework presented in Chapter 4. What distinguishes these systems from most other approaches to tracking is the use they make of task-specific similarity models, and specifically of the embeddings learned to facilitate detection of and search for similarity under these models. The similarity embeddings are an essential component of the systems from which they derive the ability to establish and evaluate hypotheses more efficiently.

In Section 5.1 we give a brief introduction to articulated tracking and discuss the role played by single-frame pose estimation. The first system, described in Section 5.2, is aimed at motion-driven swing dancing animation. That is a joint work with L. Ren, J. Hodgins, P. Viola and H. Pfister, published in [93]. Our contribution in that work has focused on the design of an example-based approach, based on similarity learning, to evaluating the hypotheses arising in the context of tracking constrained motion with a discrete dynamics model. The second system, that offers a new approach to the task of general articulated tracking, is a joint work with D. Demirdjian, L. Taycher, K. Grauman and T. Darrell. It was published in [35], and is described here in Section 5.3. In contrast to the first system, here our similarity learning framework is responsible for generating hypotheses based on estimated similarity between stored examples and the current observation.

5.1 Articulated tracking

The task of articulated tracking is to recover a *sequence* of articulated poses from a video (sequence of images) showing a moving articulated object, in most cases a human. The definition of pose, and the desired form of the output for each frame remain the same as in the static pose estimation task introduced in the previous chapter. What makes the tracking task qualitatively different is the presumed dependencies between consecutive poses. Tracking algorithms attempt to take advantage of such dependencies by framing the problem as that of probabilistic, usually Bayesian, inference. Consequently, the tracking task is often reformulated as estimating, at every frame, the *posterior distribution* of the pose parameters given the observations

so far.

5.1.1 Probabilistic tracking framework

Let $\theta^{(t)}$ be the articulated pose at time t, and $\mathbf{x}^{(t)}$ the observation recorded at that time–that is, an image, or a set of images received by a multi-camera system. We will denote $\mathbf{x}^{(1,\ldots,t)}$ the entire sequence of observations recorded up to, and including, time t. A standard step in probabilistic training is to model the *dynamics* of the motion by a distribution $p(\theta^{(t+1)}|\theta^{(1)},\ldots,\theta^{(t)})$. Usually folded into this model is the assumption that the current pose is independent of the pose in most of the previous frames, given the last few ones, so that the *prior* distribution of the pose given the "history" is

$$p(\theta^{(t+1)}|\theta^{(1)},\ldots,\theta^{(t)}) \;=\; p(\theta^{(t+1)}|\theta^{(t-\tau)},\ldots,\theta^{(t)}). \tag{5.1}$$

The relationship between the observation and the image is modeled by the *likelihood* function $p(\mathbf{x}_t|\theta^{(t)})$. The posterior distribution of the pose in the current frame, given all the observations so far, can be written as

$$\begin{aligned} p(\theta^{(t+1)}|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t+1)}) \;\propto\; & p(\mathbf{x}^{(t+1)}|\theta^{(t+1)}) \\ & \times \int_{\theta^{(t-\tau)},\ldots,\theta^{(t)}} p(\theta^{(t+1)}|\theta^{(t-\tau)},\ldots,\theta^{(t)}) p(\theta^{(t-\tau)},\ldots,\theta^{(t)}|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t)}), \end{aligned} \tag{5.2}$$

using the Bayes rule to invert the likelihood and normalized to integrate to unity. The first factor in the integrand in (5.2) is computed according to the dynamics model (5.1); the second factor is expanded, recursively, using the same equation. An important consequence of this approach is that it is necessary to have an estimate for some initial frames in the sequence (or a reliable narrow estimate of the posterior in those frames) known. Providing such initialization is one of the main roles played by single-frame pose estimation, which is not dependent on the past estimates and observations.

In practical applications of tracking it is usually necessary to commit to a specific point estimate, that is, to produce a set of deterministic values of the model parameters. The commonly used principal way to form such an estimate in a probabilistic framework is to compute the *maximum a-posteriori* (MAP) value of the pose, that is, one that maximizes (5.2).

5.1.2 Models of dynamics

The form of the prior depends on the assumptions about the dynamics in the system, and on the "depth of the horizon" τ.

Continuous models

One popular model is the Gaussian *diffusion model* under which $\tau = 1$, and the prior is $p(\theta^{(t+1)}|\theta^{(t)}) = \mathcal{N}(\theta^{(t)}; \mathbf{0}, \mathbf{\Sigma})$. This essentially means that the model constrains the magnitude of the motion. This is the model used in the system described in

Section 5.3. More complex models that use higher values of τ may be able to model higher order properties of the motion such as velocity or acceleration. Such models are either analytical, implementing linear filtering mechanism like the Kalman filter and its variants [110], or non-parametric and learned from the data [72]. In these models the information about the past is entirely contained in the prior distribution. The prior distribution may be represented in a parametric form, or in a semi-parametric form, that is, by a set of samples [66]. The latter approach leads to the *particle filtering* framework, among the most popular ones in articulated tracking.

Discrete models

In some applications, the motion is highly structured, in the sense that only a finite, and relatively small, number of transitions are considered possible from any given pose. Furthermore, the number of attainable qualitatively different poses is also finite.[1] This is usually the case with activities that follow certain well-defined rules, like many sports or dancing. Under these constraints, it is possible to write down the dynamics model by explicitly enumerating the possible poses as nodes of a graph, and possible pose transitions as directed edges in this graph, weighted by the probability of the transition. This leads to the *motion graph* [72], the model used in the system described in Section 5.2.

5.1.3 Likelihood and similarity

The form of the likelihood term in (5.2) is typically determined by a generative model of image given pose. In most algorithms, maximizing the likelihood is achieved by a gradient-based optimization, in which the model is iteratively used to predict an observation, and the mismatch between the prediction and the actually observed data is used to improve the model. Two major problems with this approach are its computational complexity (likelihood computations are often the computational bottleneck of a tracking algorithm) and the dependence on the starting location [108].

In the context of motion graph tracking, computing the likelihood is reduced to evaluating the match between the input frame and a finite set of hypotheses, namely all the poses θt_i in the notation of (5.3). In other words, this is an instance of the similarity detection problem formulated in Section 1.1, and we will approach it using the tools developed in Chapter 3.

The likelihood model in the systems described in this chapter is example-based: likelihood is evaluated implicitly, by comparing the input frame to stored examples for which the pose is known:

$$p(\theta_i|\mathbf{x}^{(t)}) \sim \mathcal{S}_{pose}(\mathbf{x}^{(t)}, \mathbf{x}_i).$$

More precisely, the likelihood is assumed to be high for poses whose associated observations are similar to the current observation. This is made possible by applying the similarity learning framework developed in Chapter 3. The assumption underlying

[1] That is, the number of poses qualitative different up to some tolerance.

this approach is that examples similar to the input under $\mathcal{S}_{pose}$ correspond, with high probability, to peaks of the likelihood.

5.2 Case I: Motion-driven animation

This section describes a system for motion-driven animation in the domain of swing dancing. The application here is to allow a user to perform a swing dance in front of a multi-camera system, parse the motion into an admissible sequence of swing steps, and render it on the screen along with a matching action by a "virtual partner". The details of the entire system are available in [93]; here we will be focusing on the motion parsing task, which is essentially an articulated tracking task constrained to a specific *dictionary* of poses defined by the rules of swing dance.[2]

5.2.1 The setup and training data

The visual input to the system consists of three silhouettes extracted[3] from synchronized, calibrated consumer-grade digital cameras, mounted so that the fields of view overlap over a working area of approximately 8'×24'. These are concatenated to form a single three-view observation $\mathbf{x}$. The 62 parameters of the articulated model θ can be divided into the pose parameters ξ, describing the configuration of the limbs in the articulated tree (as joint angles in person-centered coordinate system), and the parameters specifying the orientation and location of the entire articulate tree in the world coordinate system. The orientation β can be encoded by a single number, the *yaw* angle, as we can assume that a swing dancer's body is generally in an upright posture. Specifying location involves additional two degrees of freedom λ for the coordinates on the ground plane where the center of mass is projected. In the remainder of this section we will refer to ξ as pose, to make this distinction clear.

The dynamics of the swing dance are modeled by a motion graph, constructed from 5,120 frames of motion capture data recorded with a professional dancer. Transitions are modeled using the distance between poses and ignoring the global orientation and location. In addition to these transitions, a small number of transitions are added manually to ensure compliance of the graph with the choreographic rules of swing.

The graph constructed in this way encodes a rather rigid constraint on the movement speed; if the user moves significantly slower or faster than the dancer in the motion capture recording session, no transitions will match his or her motion. To alleviate that, the motion graph is *augmented* by adding a few nodes on each edge to allow for slower motion and extra edges (from each node to its grandchildren and grand-grandchildren in the graph) to allow for faster motion.

[2]The contribution to [93] by the thesis author was primarily in this component.

[3]The silhouette is extracted by applying a simple color-based foreground/background segmentation model, finding the bounding box of the foreground pixels and resizing it to 60×60 pixels.

Tracking with motion graph

The prior distribution (5.1) corresponding to the motion graph is a discrete probability mass function, assigning values according to the weights on the out-going edges from $\theta^{(t)}$ and putting zero weights on absent edges. Consequently, the posterior is also discrete and can be described by listing all the paths $\theta^{(1)} \dots, \theta^{(t)}$ in the graph. For computational reasons, it is possible to maintain only paths of certain maximal *buffer length* b (to avoid combinatorial explosion of the number of paths.) Furthermore, depending on the application we may afford to maintain a small *look-ahead* buffer, that is, we can defer inferring the pose in frame t until we have seen the frame $t+a$.[4]

Suppose that at the time $t+a$ we maintain n_t paths

$$\{\theta_i^{(t-b)}, \dots, \theta_i^{(t+a)}\}_{i=1}^{n_t}.$$

The expression for the posterior is

$$p(\theta^{(t)}|\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t+a)}) \propto p(\mathbf{x}^{(t)}|\theta^{(t)}) \times \sum_{i=1}^{n_a} p(\theta^{(t)}|\theta^{(t-b)}, \dots, \theta^{(t-1)}, \theta^{(t+1)}, \dots, \theta^{(t+a)}) p(\theta^{(t-\tau)}, \dots, \theta^{(t)}|\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}), \tag{5.3}$$

5.2.2 Two-stage architecture

In principle, one could attempt to build a similarity classifier which would be invariant to the external parameters of the model (location and yaw). However, this makes for a very complex problem, since the appearance of the silhouettes depends greatly both on yaw and on the pose and, albeit to a much lesser extent, on the location. Instead, we will follow a divide-and-conquer approach which breaks the estimation into a two-stage process.

At the first stage, the yaw of an observation is estimated and quantized into one of fixed yaw "bins". At the second stage, the pose of the observation is estimated using a similarity model that "specializes" on a particular yaw bin.[5] Each of these stages deals with an input subspace in which severe variations of appearance are largely accounted for by the relevant parameters to be recovered. Below we describe the role played by similarity detectors in the design of both stages.

The training data

As mentioned in the previous chapter, an example-based approach requires a representative database of pose-labeled observations. The human data itself in this case is

[4]The resulting dynamic model could in principle be modeled as a hidden Markov model, however estimation in this framework is not practical due to the complexity of the state space together with the requirements from the application: it has to be real-time and, importantly, have no "jitter" in the resulting animation.

[5]This architecture resembles the mixture of experts architecture[67], with the yaw classifier acting as a gating function.

representative of the relevant body configurations, but is not inclusive of all possible angles and locations in the working space (the latter affects the input significantly due to perspective distortion in the cameras). Thus, a database in this case is built following the same paradigm as in Section 4.5: by using human motion capture data described above to generate a larger set of synthetic examples.

For each recorded frame of the human data, an artificial set of silhouettes is rendered, using a computer graphics package, for every combination of 36 yaw bins and five fixed locations. Each yaw bin c is associated with the angle $c\pi/36$, and covers all yaw values $c\pi/36 - \pi/72 \leq \beta \leq c\pi/36 + \pi/72$; the five locations are in the center and four corners of the work area. This procedure yields a database of 921,600 example observations (triple silhouettes).

These observations are represented in the space similar to the EDH (see Chapter 4) but more appropriate for binary silhouette data: the concatenated set of responses of *box filters*. These filters can be visualized, as in Figure 5-1, by boxes placed over an image region and divided into black and white segments. The value of a filter is computed by subtracting the sum of the pixel values within the black portions of the box from the sum of the pixels within the white portions. These filters were introduced in [115] in a similar context, where the set of responses of such filters was used as a feature space for ensemble face detectors. It was also shown in [115] that each response can be computed very efficiently using the integral image transform (in which the value in each position is the sum of the pixels above and to the left from it.) We used three types of such filters, all seen on Figure 5-1: a two-part (vertical or horizontal), a three-part (vertical or horizontal), and a four-part "checkerboard". Filters of each type are slid, at multiple scales, through the image similarly to the multi-scale EDH. The resulting representation for a 180×60 observation contains more than 200,000 values.

Pose-invariant yaw classification

We treat the problem of estimating the yaw as a regression problem where the range of the target function is the discrete set of yaw bin centers $c\pi/36$. The precision (bin width) of ten degrees was chosen by examining its potential effect on the second stage of the process–the pose classification. We have observed that, on the one hand, finer divisions of the yaw range do not seem to significantly reduce appearance variation between images rendered for the same body pose for angles within the same bin, and on the other hand, coarser divisions seem to significantly increase such variation.

Once formulated as a regression with a set precision threshold, this problem naturally fits the framework outlined in Section 2.2.1. We therefore define a yaw-sensitive similarity between two images $\mathcal{S}_{yaw}$ to be $+1$ if the underlying yaws fall in the same bin–ignoring the pose and the location! Examples of a similar and of a dissimilar pair under $\mathcal{S}_{yaw}$ are shown in Figures 5-2 and 5-1.

Using this definition we form a training set of similar and dissimilar pairs, and apply the Boosted SSC algorithm. To make sure the training examples are representative of the poses while keeping them of manageable size, we apply k-means clustering [38] to find 50 centers of pose clusters. We then construct each of the 4,000

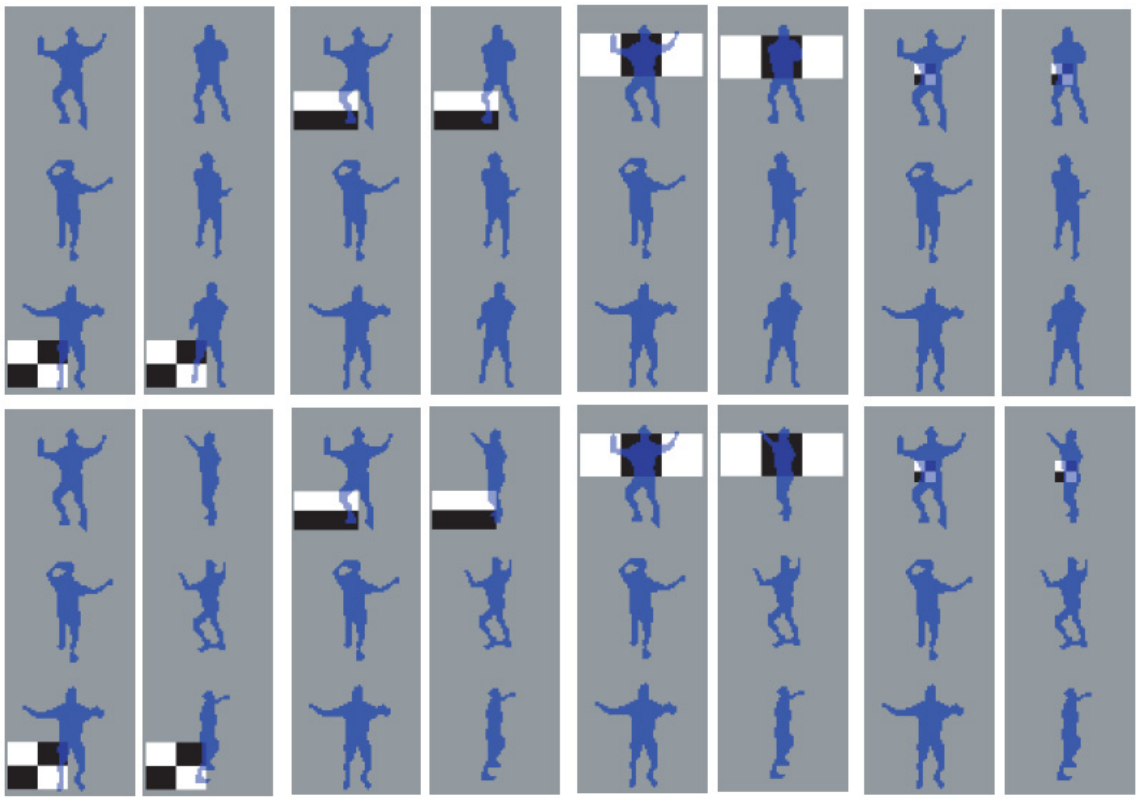


Figure 5-1: Some of the projections (filters) learned by Boosted SSC for yaw similarity. Top row: positive examples. Bottom row: negative examples. A projection corresponding to a box filter is computed by subtracting the sum of pixels under the rark regions of the box from the sum of pixels under the white regions. From [93].

Figure 5-2: Positive (left pair) and negative (right pair) training examples for yaw similarity; pose and location are ignored in determining $\mathcal{S}_{yaw}$. From [93].

positive examples by choosing a random bin, selecting two of the cluster centers, and selecting a random location (out of five possible) for each. 6,000 negative examples are formed by a similar process, pairing cluster centers across bins. Increasing the number of training examples beyond 10,000 is impractical (recall that with the finite number of projections, as in this case, each iteration of Boosted SSC involves examination of the value of each projection on each pair.)

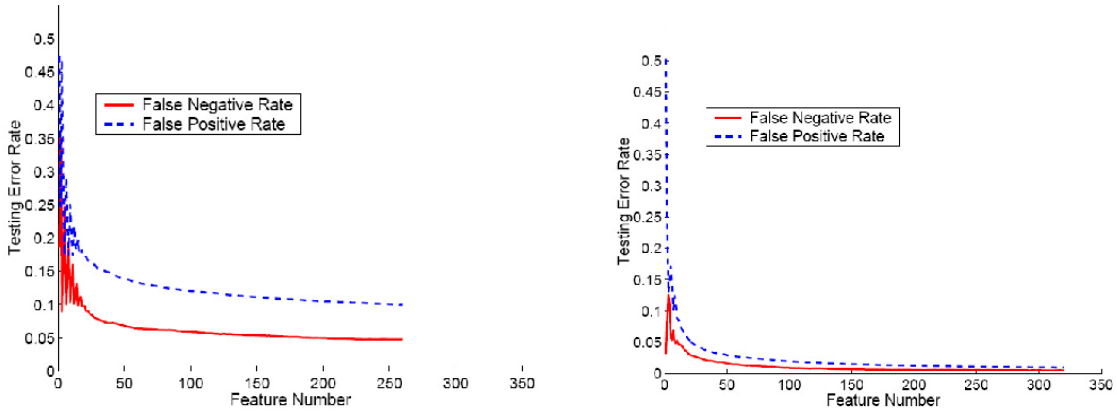
We have also set up an additional set of labeled pairs (300,000 positive and 37,000,000 negative), not used in training, to serve as validation set. Testing on this set reveals that 10,000 examples may not be sufficient to cover the space adequately. Therefore, we follow the *resampling* approach [106]. We maintain a much larger set that serve as a "pool" of training examples (800,000 positive and 2,000,000 negative). Every 40 iterations of boosting, we resample a 10,000-example training set of pairs by the following procedure:

1. The response $y_i = \sum_m \alpha_m c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$of the current ensemble classifier is computed for each example pair in the pool.

2. Each example is assigned a weight $\exp(-l_i y_i)$.

3. The weights are normalized to form a distribution.

4. A new 4,000+6,000-strong training set is sampled according to that distribution; the AdaBoost distribution is set to uniform.



(a) Yaw similarity classifier

(b) Pose similarity classifier

Figure 5-3: Test errors for yaw (top) and pose (bottom) similarity classifiers, as a function of the number of features collected by AdaBoost. From [93].

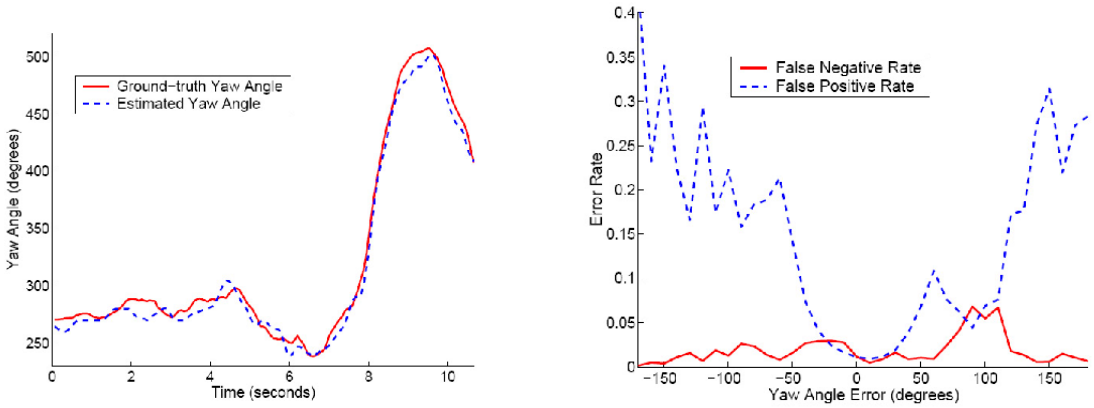
Note that the computational cost of this algorithm is quite high, even with this relatively small fraction of examples explicitly examined. Fortunately, most of the steps in each iteration, including the resampling steps, can be trivially parallelized, by dividing the data and the features among processes. Most of the computation is done in parallel, and only modest amount of inter-process communication is required at each iteration to combine the error estimates, select the winning weak classifier and distribute the updated parameters such as α to each process. We have implemented such a parallelized version of the algorithm, and ran it on a Beowulf cluster, using between 80 and 120 processes.

Figure 5-3(a) shows the behavior of the error on the validation set as boosting proceeds. As can be seen, the error is steadily decreasing until it levels off, and the algorithm is stopped after 260 iterations. We thus have obtained a weighted 260-bit encoding that corresponds to $\mathcal{S}_{yaw}$. the estimation of the yaw for a new observation follows the paradigm presented in Section 2.2.1, using a 20-NN classifier with L_1 distance in the embedding space. Due to the real time speed requirement of the application, we use LSH (Section 2.4.2) to perform the search for neighbors. To deal with potential failures due to mirror symmetry of the silhouettes, the actual estimate during tracking is made more robust by the following procedure:

1. Instead of taking a simple majority vote, we build a histogram of the labels (yaw values) of the 20 NN and smooth is with a low-pass filter.

2. Two highest peaks in the histogram are located.

3. We form a set of four candidates from those peaks and their mirror reflection (by adding π to each).



(b) Yaw similarity classifier

Figure 5-4: 5-4(a): performance of the yaw estimator on a real video sequence of swing dancer. Solid: ground truth from motion capture; dashed: estimate. 5-4(b): effect of error in yaw on the resulting pose similarity classifier. The correct yaw in this case is zero. From [93].

4. Out of these four candidates, we choose the one closest to the estimated yaw in the previous frame.

Figure 5-4(a) shows an example performance of the final yaw estimator on a real video recorded simultaneously with the motion capture. For 73% of the frames in this sequence, the error in yaw estimate is below 10 degrees - i.e., a correct yaw bin assignment under our definition of sensitivity. In 92.5%, the error is below 20 degrees, and in 98% below 30 degrees. To understand how tolerable are errors of different magnitude, we tested the effect of incorrect yaw estimation on the accuracy of the resulting pose similarity classifiers (we describe learning these classifiers below). The results for the case of zero degrees yaw, shown in Figure 5-4(b), imply that errors of up to 30 degrees do not cause significant increase in the error of pose similarity classifier. We believe that this is explained by the relatively smooth transitions between the appearances of the same pose in neighboring yaw bins. It is interesting to consider the difference between false negative and false positive rates: the latter is significantly higher, in particular for yaw estimates with large error, since the responses of the similarity classifier in that regime become more random, and as we have noted in Chapter 3 it is "easier" to misclassify a dissimilar pair by a random set of projections/thresholds.

Yaw-specific pose estimation

For each yaw bin β, we learn a classifier of the pose similarity $\mathcal{S}^{\beta}_{pose}$ defined according to (4.1), with $\mathcal{D}_{\theta}$ being the L_2 in the pose parameter space. The learning proce-

Figure 5-5: Positive (left pair) and negative (right pair) training examples for pose similarity for a fixed yaw. From [93].

dure closely follows the one described above for the yaw similarity learning, with the following differences:

- For positive examples, all the 5,120 poses, rather than 50 cluster centers, are used, since the similarity rate for the pose is significantly lower than for yaw, and the total number of potential positive pairs is much lower. For negative examples, we use clustering in the way described above.

- There are 3,000 positive and 5,000 negative examples in the training set, and 200,000/10,000,000 examples in the resampling pool.

- Resampling occurs every 80 iterations.

- The number of boosting iterations is 320.

As can be seen in Figure 5-3(b), the behavior of the error of a typical pose similarity classifier follows the same trend as for the yaw classifier. However, the absolute level of the error is much lower. This reveals that the difficulty of the yaw estimation task significantly exceeds that of the pose classifier. We relate that to two factors. One is the larger visual diversity among the examples seen by the yaw similarity classifier, versus any of the 36 pose similarity classifiers. The other factor is the much smaller, in absolute numbers, set of potential positive examples in the case of pose

estimation; as a result, it is possible to represent those better in the training set provided to the learning algorithm, whereas in the case of yaw classifier we must resort to clustering as a pre-processing step.

Once the pose similarity classifiers are learned, we could follow the same approach as with yaw estimation, i.e. convert the classifier to an embedding, and build a NN-based regression mechanism using the L_1 in the embedding space. However, recall that under the motion graph model we need, for frame t, to consider a finite number of examples - the nodes $\theta_1^{(t)}, \ldots, \theta_{n_t}^{(t)}$ in terms of Equation 5.3. The number of currently maintained paths n_t is typically small due to relatively low branching factor of the motion graph. Therefore, we can afford to use the similarity classifier explicitly! In other words, we can form n_t pairs of observations $(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t)})$ and compute the response[6] $\sum_{m=1}^{M} \alpha_m c_m(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t)})$.

5.2.3 Performance

The system described in this section has been successfully tested on a number of real video sequences. To obtain a quantitative measure of error, an evaluation was done on one such sequence for which motion capture data, recorded simultaneously with the video, was available. On that sequence, performance of our system was compared to the performance of a system using a commonly used set of features–Hu moments [63]. Hu moments are based on seven moment invariants of a binary shape 9the silhouette in this case). The purpose of this evaluation was to try to ascertain the effect of learning similarity and of using the box filter representation.

Figure 5-6 shows the histogram of error *differences*; the superior performance of the learned similarity representation over Hu moments is clear. In 86% percent of the frames, the error with Hu moments exceeded the error of our system. The effect of the learned embedding (or equivalently the learned distance measure) is further illustrated in Figure 5-7. The figure shows some typical examples of poses retrieved by the nearest neighbor search with respect to the ground truth, the embedding distance, and the distance in Hu moment space (the results shown were obtained with no temporal information to emphasize the effect of the space/distance on the retrieval.) The average per frame $\mathcal{D}_\theta$ error of the embedding NN was 44cm, compared to 25cm with the "hindsight" NN (finding the best match knowing the correct ground truth value) and to 76cm with Hu moments.

Finally, in addition to measuring the accuracy of pose estimation on a per-frame basis, extensive evaluation was done to compare the quality of the resulting animation to that obtained with state-of-the-art methods in the field; see [93] for details.

[6] We found that ignoring the weights α_m speeds up the computation, which is then reduced to calculating Hamming distance, with no significant effect on the results.

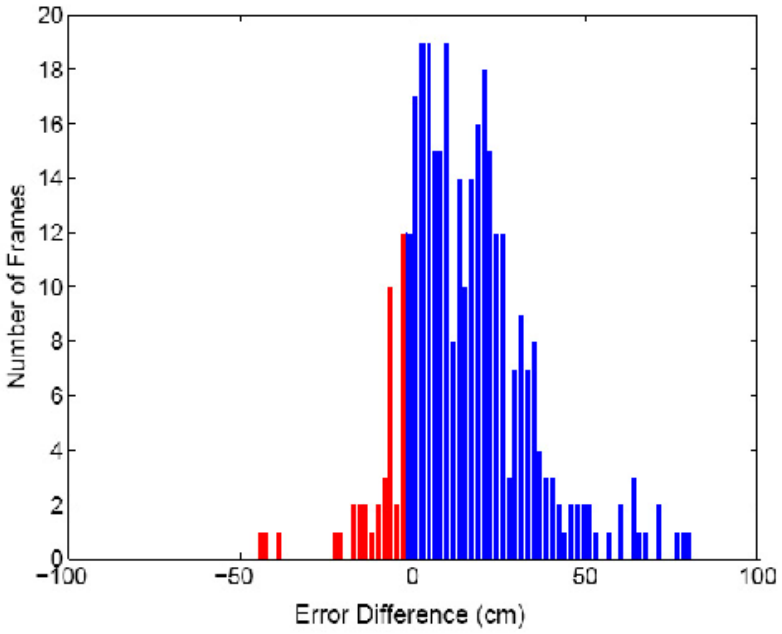


Figure 5-6: Histogram of differences between the $\mathcal{D}_\theta$ error of 1-NN in Hu moments and the 1-NN error in the learned similarity embedding; positive values mean smaller error for our system. Obtained on one real dance sequences with measured ground truth. From [93].

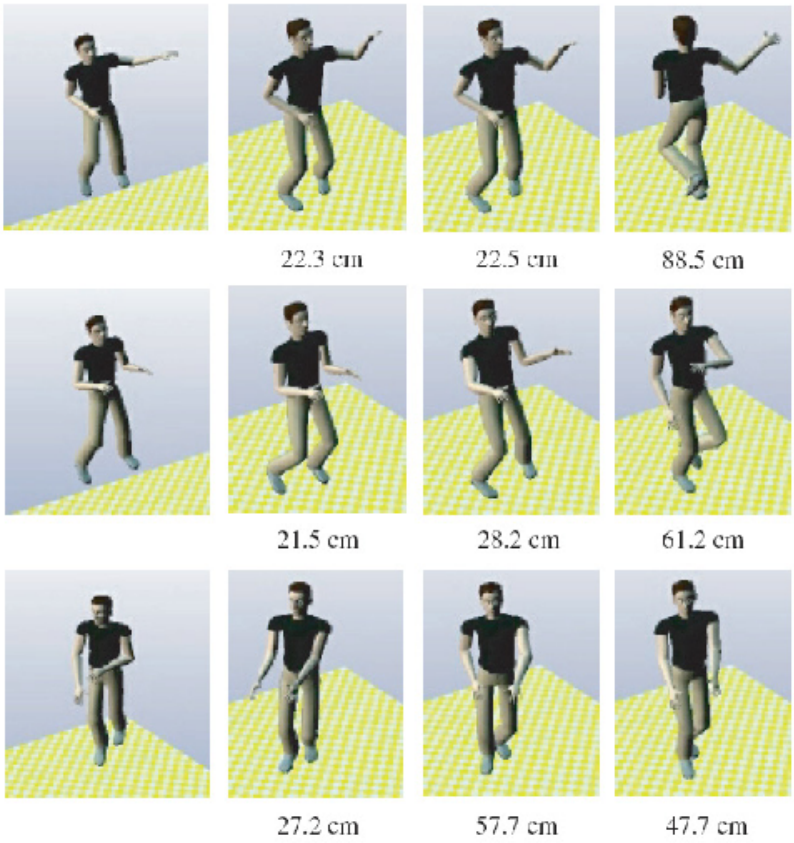


Figure 5-7: Representative examples of pose retrieval accuracy with various methods. Each row corresponds to one frame. Columns, from left to right: Ground truth; best match in the database by brute search; best match in the embedding space; best match with Hu moments. Numbers: L_2 error in the pose space. From [93].

5.3 Case II: General pose tracking with likelihood modes

If the goal of the tracking application is not restricted to a specific domain with constrained and structured motion, continuous models of dynamics are more appropriate. The system described in this section implements such a model. It is however unusual in that it avoids the problem shared by many state-of-the-art tracking algorithms: being led astray by an inaccurate prior. This is achieved by using a weak prior, and using similarity search in an embedding space to obtain the peaks of the likelihood function directly from the observation. Since such search can be implemented extremely fast using LSH, the resulting tracking system is also very fast.

5.3.1 Pose similarity as likelihood sampling

As mentioned in Section 5.1.3, we adopt the assumption that the examples closest to the current observation in the similarity embedding space are, with high probability, close to peaks of the likelihood. More precisely, we assume that at least one example among those returned by a K-NN search in the embedding space will correspond to each high peak in the likelihood.

Following this assumption, we can think of searching a large database of labeled images under a learned pose similarity as a very fast evaluation and sampling of approximate likelihood. The examples not returned by the search (i.e., the vast majority of the database) are considered to have been pruned away by this procedure, which is taken to mean that their likelihood is low. The returned examples are treated as candidates for high likelihood. However, since the search is approximate, we do not directly use these examples in estimation, but rather use them as starting points for local optimization.

The likelihood optimization in our system uses the Iterative Closest Point (ICP) algorithm [15] to find a local optimum of the match between a set of points on the surface constructed for the articulated model and a corresponding set of points on the estimated surface of the observed body (the latter is obtained by applying a standard stereo matching algorithm.)

5.3.2 Tracking with likelihood modes

The main motivation behind the algorithm described here is to avoid a typical failure mode of tracking algorithms that rely on the prior: being led astray and losing track due to inaccurate, and overly confident, prior. One reason for using a strong prior is that it provides a starting point for an optimization step in which the model is improved with the objective to increase the likelihood (and thus the posterior), typically by means of an iterative gradient ascent algorithm. This strategy often is reasonable, however it may fail in cases of occlusion, unusually fast or slow motion, and simply when the tracker made significant mistakes in the previous few frames. Applying

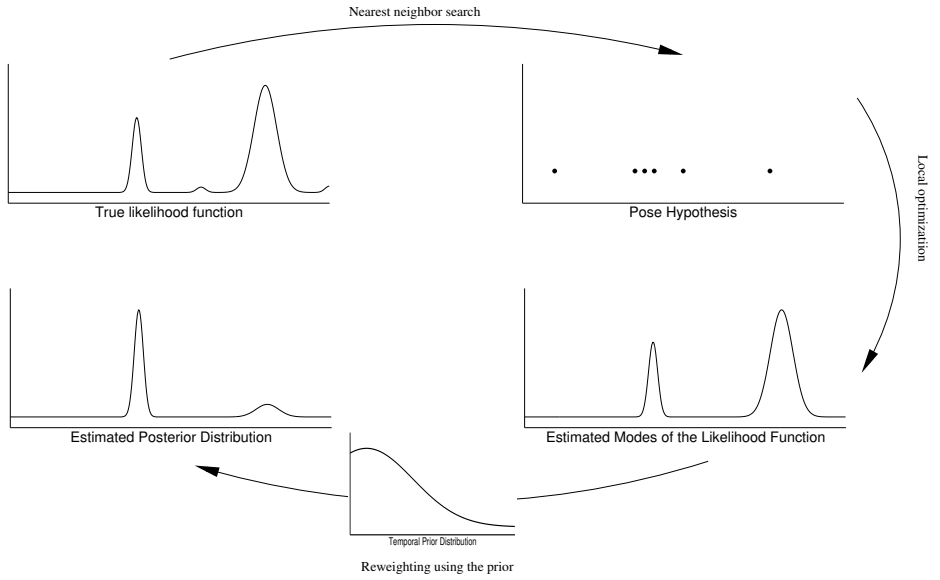


Figure 5-8: Schematic illustration of the estimation flow in ELMO. The temporal prior is obtained by multiplying the posterior from the previous frame by a high-bandwidth Gaussian. From [35].

strong prior model in such cases results in a bad starting point for the gradient descent, which may have critical effect on the quality of the eventual result [108].

The operation of our tracking algorithm (called ELMO: Tracking with Likelihood MOdes) is schematically shown in Figure 5-8. At every frame, we maintain a parametric estimate of the pose posterior in the previous frame in the form of a mixture of Q Gaussians. A temporal prior is obtained from this estimate by multiplying the posterior by a high-bandwidth Gaussian window, in accordance with the diffusion dynamics model.

Given the new observation, we compute its embedding into a space H and retrieve the poses corresponding to K top matches in H. As mentioned above, our assumption is that the poses associated with the retrieved examples have, with high probability, high likelihood given the current observation. Each of these K candidate poses is then used as a starting point for an ICP search to find the adjacent local maximum of the explicit likelihood. Once these modes of the likelihood are found, we multiply them by the prior obtained, as mentioned above, by smoothing the posterior from the previous frame. The result is the estimate of the posterior in the current frame.

5.3.3 Implementation and performance

In our system we use the concatenated multi-scale edge direction histogram (EDH) as the feature space, as described in the previous chapter. Using a combination of color background model with stereo volume information we extract a bounding box of the silhouette and normalize it to 200×200 pixels. With 3 scales (8, 16 and 32 pixels) and with location step size of half the scale, the EDH yields $N = 13,076$

bins. Applying SSC , and specifying the minimum TP-FP gap of .1, we obtained $M = 3547$ embedding dimensions.

Having encoded the data (described in Section 4.5) according to the learned embedding, we constructed LSH with $l = 50$ hash tables with $k = 18$ bit keys. At the run time, we retrieve for every frame $K = 50$ training examples and use their poses to initialize the ICP.

Using six synthetic sequences generated based on human motion capture data, obtained from [41], we compared the performance of ELMO to that of two alternative algorithms: the state-of-the-art Condensation particle filtering approach [36], with 1,000 particles, and the ICP differential tracker [34]. We also measured the per-frame error resulting from retrieving the NN under the SSC embedding directly. The sequences are rendered with significant perspective distortion, relatively wide range of motions, much self-occlusion and are therefore challenging for a tracking algorithm.

The summary of the results of this comparison appears in Figure 5-9. As can be seen, the average error with ELMO is the lowest. Figure 5-10 shows the per-frame account for segments from two of the synthetic sequences. The error with ELMO is the lowest in most of the frames. The performance of Condensation was the worst among the tracking algorithms (and only better than the per-frame static estimation with SSC that ignores dynamics). It is possible that with additional particles Condensation would improve its performance. However, with 1,000 particles it is already two orders of magnitude slower than the other methods.

It is also interesting to note that although the average performance of the single-frame pose estimation with SSC is inferior to that of proper tracking algorithms, it fares relatively well, with the error being less than a standard deviation above the average errors of Condensation and ICP trackers. Looking at the sequence data we can see that in fact there are segments in which it does better than those algorithms. This provides some insight into the success of ELMO: relying on the robust single-frame estimator allows it to recover from some severe errors, since at most after a few frames the matches found with LSH in the embedding space become again close to the likelihood peaks. This is also related to an additional advantage not explicit in these figures: ELMO does not need manual initialization, instead using the NN search from the very first frame.

In addition to the tests with synthetic data, where exact error evaluation is possible, we evaluated the performance of ELMO on a number of real video sequences. Figures 5-11 and 5-12 show a few frames from two of such sequences. Notably, the frame rate in these sequences was only four frames per second, creating significant inter-frame pose differences. With tracking algorithms strongly depending on the prior, this would normally pose much difficulty. However, ELMOdeals with this gracefully.

5.4 Discussion

In both of the tracking systems described in this chapter, an example-based pose estimation algorithm is a crucial component. In the swing dance animation system it

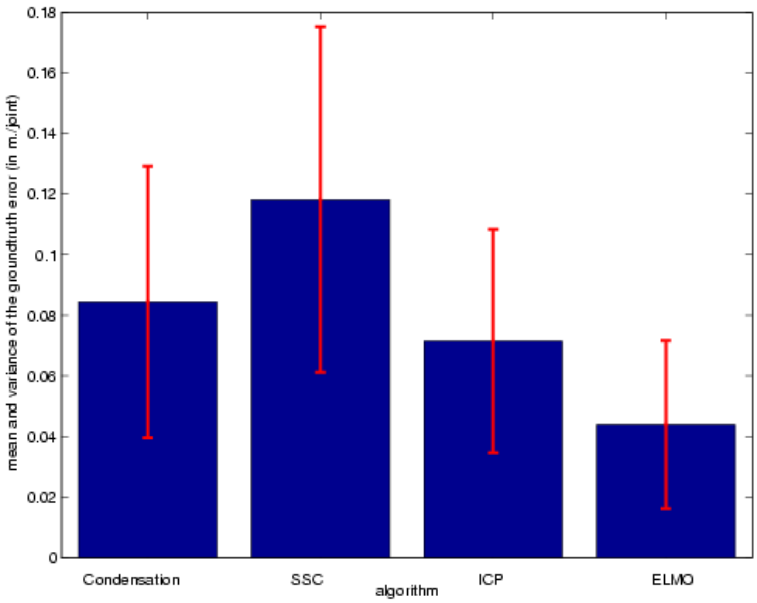


Figure 5-9: Average and standard deviation of the ground truth error obtained using Condensation, SSC (PSH, in terms of [35]), ICP, and ELMO on six sequences of 1000 images each. ELMO outperforms the Condensation, ICP and SSC algorithms. The average error per joint for ELMO is less than 5 cm. From [35].

allows for efficient evaluation of multiple hypotheses. In ELMO, it provides a means of automatic initialization and re-initialization, and in combination with the weak temporal model for the prior makes the tracker qualitatively less vulnerable to losing track.

Two key properties of our similarity learning approach that make it possible are the classifier/embedding duality, that enables us to reduce the problem of evaluating a large hypothesis space to the problem of search in a database for neighbors under the L_1 distance, and the setup for learning the similarity specific to the task at hand. The latter, as exemplified in the swing animation system described in this chapter, allows us to break down complex estimation problems into much simpler ones, leading to a winning divide-and-conquer architecture.

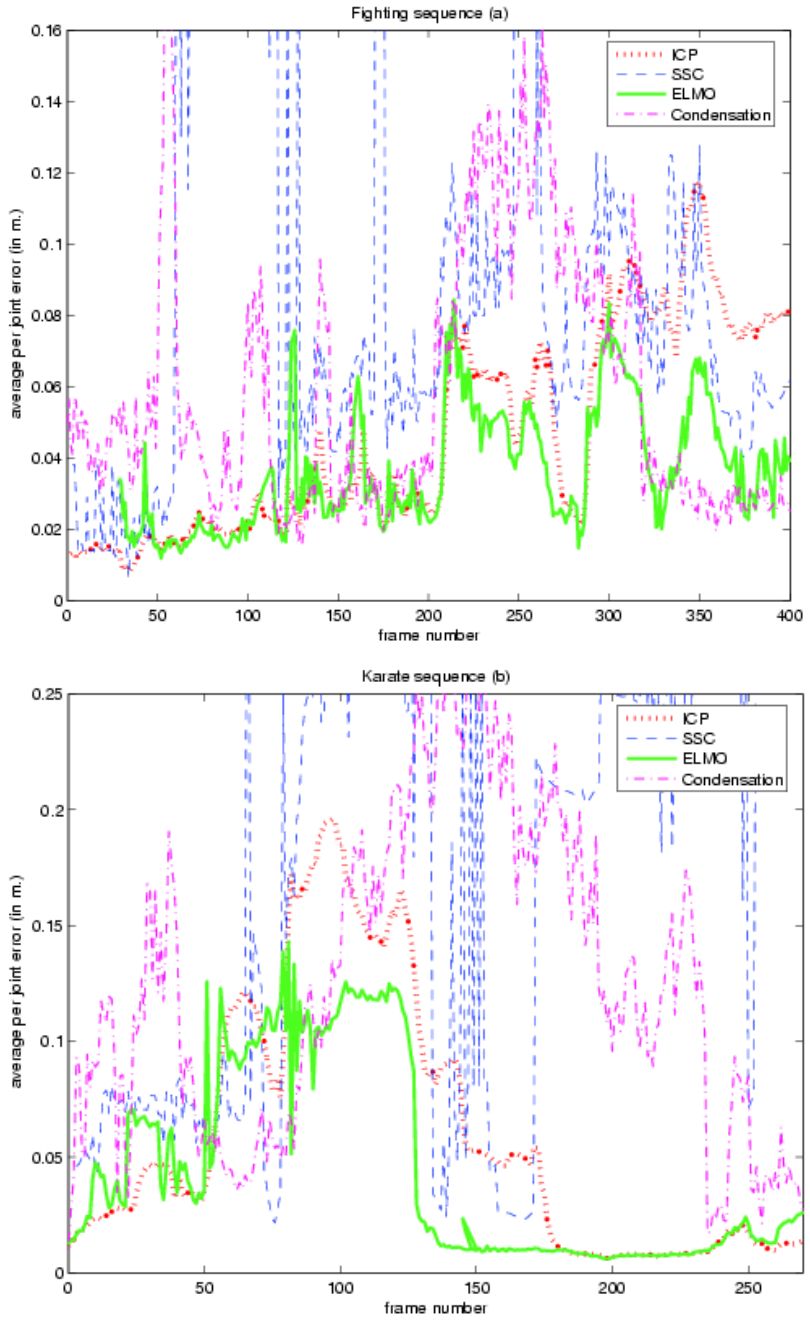


Figure 5-10: Tracking results on two of the six test sequences (for better clarity, only segments of the sequences are shown). The graphs report the ground truth error (vs. frame number) corresponding to Condensation, SSC, ICP and ELMO. (a) Fighting sequence, (b) Karate sequence. The error corresponding to the ELMO algorithm is almost always the smallest. From [35].

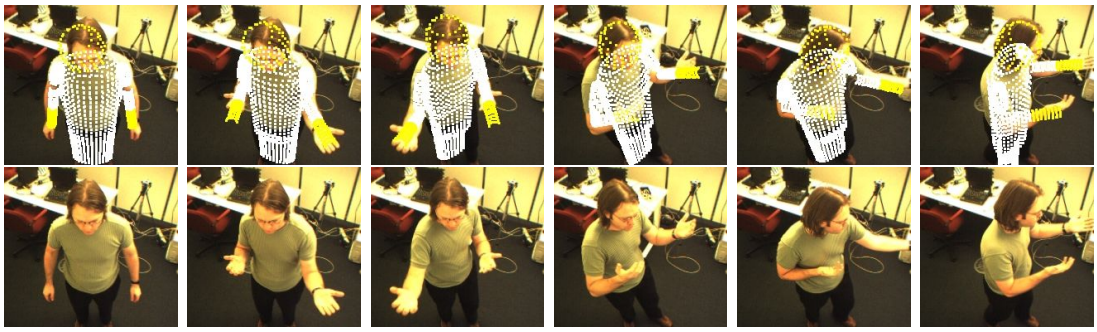


Figure 5-11: Tracking results extracted from the *dance* sequence. From [35].

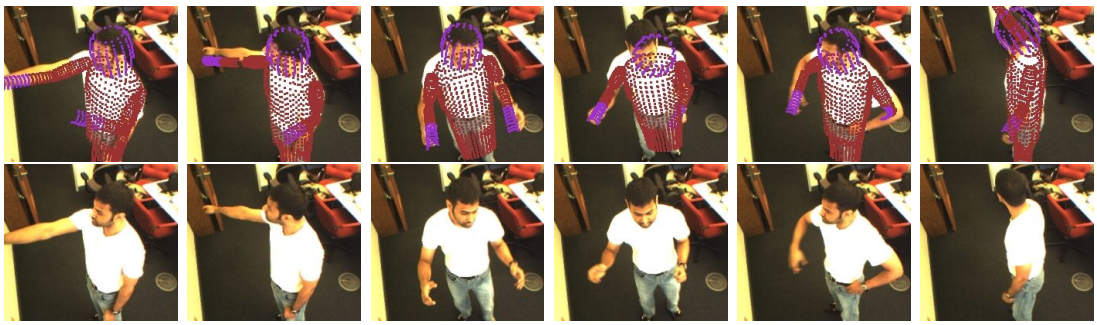


Figure 5-12: Tracking results extracted from the *whiteboard* sequence. From [35].