Chapter 5

Human Pose Retrieval System

5.1 Introduction

In this chapter, we describe the real time retrieval of human poses from a large collection of videos. The last decade has seen considerable progress in 2D human pose (layout) estimation on images taken in uncontrolled and challenging environments. There are now several algorithms available [5, 27, 80, 103] that can detect humans and estimate their poses in typical movie frames, where humans can appear at any location, at any scale and wear clothing of varying sizes, colors and texture; the illumination can vary and the backgrounds can be cluttered.

There are numerous reasons why detecting humans and obtaining their pose is useful. A fundamental one is that often the pose, or a pose sequence, characterizes a person’s attitude or action. More generally, applications range from video understanding and search for suspicious activity detection in surveillance videos to search for particular sport shots or dance poses. In this work, we enable, for the first time, the real time retrieval of poses in a scalable manner. Being able to retrieve video material by pose provides another access mechanism to video content, complementing previous work on searching for shots containing a particular object or location [87], person [6, 61, 84], or action [11, 59].

The real time system we propose is applicable on top of most 2D pose estimation algorithms. The only condition is that their output can be mapped into the simple pose representation, which enables rapid Euclidean distance based nearest-neighbor matching. We demonstrate our system here on the output of two existing pose estimation algorithms, Eichner & Ferrari [27] and Yang & Ramanan [103]. The functionality of the system is illustrated in figure 5.1, which shows the three query modalities that we have developed and the type and quality of the output.

While an early system for pose search was first proposed by [40], in this chapter we extend the idea and methods substantially. First, we introduce a low dimensional pose representation and approximate nearest neighbor matching, making our system much more efficient both in terms of computational cost and memory consumption. This enables real-time search in a large scale database (over 3 million frames). Second, our system has a better retrieval engine: by combining independent pose estimation

\[\text{http://zeus.robots.ox.ac.uk/posesearch/index.html}\]
Figure 5.1: **Pose retrieval overview.** An illustration of the three query modalities: interactive stickman GUI, pose from Kinect, and pose estimated from an image. The output ranked list is obtained instantaneously as the query is varied. This can be tailored in various ways, for example to select only poses in different shots or within a particular movie. The results of the query pose are shown here at the video level.

algorithms, we can reject frames where the pose estimate is likely to be incorrect. This leads to higher precision in the top returns (but not higher recall). Third, our approach allows for new query modes: query-by-stickman (with an interactive GUI), and query-by-Kinect, in addition to the query-by-image of [40].

The following section overviews the design and functionality of the system and the rest of the chapter then gives details of the various stages and their experimental performance.

### 5.2 System Overview

We overview here the query modes and output of the retrieval system. The system is able to search over fifty thousand poses from 3 million frames of 18 movies in 5 ms, running on a single 2.33 GHz machine. To enable such real time performance all the processing steps, query and presentation of retrieved results need careful consideration. In particular much of the processing is carried out off-line with only the pose matching carried out at run time.

The following subsections describe the querying and output of the system and the principal processing blocks.
5.2.1 Query modes

For a user to query the system, we have developed three querying modes (figure 5.1). (i) The first is a so-called stickman. The interface is intuitive where the user can move the joints of a iconic stick figure freely and the pose retrieval system responds to these movements in real-time by retrieving the matching frames from the database. Imagine that the user is moving the stickman from the famous ‘Titanic’ pose (arms stretched side ways and parallel to the ground) to a ‘Hands up’ pose, passing through a series of intermediate poses. Then the pose retrieval system follows this sequence of poses, and continuously updates the results. This mode is very useful as a web-based application and can be deployed anywhere with a basic computer and an Internet connection. (ii) In the second mode, the user himself can move by waving his/her arms around and making different poses. The pose retrieval system tracks his movements using the Kinect sensor and displays the retrieved results in real-time. This mode is suitable for entertainment applications. (iii) Finally in the third mode, the user can upload an image of a person and then request the pose retrieval system to display similar poses in the database. For mobile application, this mode is very useful.

5.2.2 Output Presentation

Ranked thumbnails of poses from the database are displayed according to their similarity to the query pose. Hovering over the thumbnail shows the full frame, and clicking on the thumbnail plays the video starting from that shot.

The user can choose the level at which to perform retrieval among (i) frame level, (ii) shot level, or (iii) video level retrieval. For example, shot level means that only a single result from each shot is displayed, whereas for frame level a number of results could be from the same shot. Thus shot and video level give more diverse results in terms of actors, scenes etc. The user can also choose whether to search the whole database or a particular movie. Retrieving from the whole database and at shot level are the default choices.

5.2.3 Off-line Processing Stages

Video processing: Frames are grouped into shots, and an upper body detector is run on all the frames to detect people. The upper body detections are then grouped through the temporal sequence into a track. Finally two human pose estimation algorithms [27, 103] are run on each upper body detection in all the tracks, to estimate a stickman pose (section 5.3).

Improving pose estimation: Unfortunately the precision of these pose estimation algorithms is not high and this significantly affects the perceived quality of the pose retrieval system. To improve precision, we propose to use the human pose evaluator which takes into account the human pose estimate and auxiliary information to decide if the pose estimate is correct or not. Several ways of using the pose evaluator are described in section 4.4. Poses which survive this filtering step are then considered for retrieval.
Pose representation and matching: For representing the pose we propose to use the the low dimensional 12D vector pose representation described in section 4.5. The pose representation is crucial to the performance of the retrieval system as it simultaneously affects both the accuracy and speed of the system. The 12D vectors are recorded in a forest of randomized K-D trees for fast approximate nearest neighbor retrieval (section 5.5).

5.2.4 On-line Processing Stages

Pose retrieval Each query mode provides a 12D vector specifying the query pose. Nearest neighbor pose matches in the database are then obtained using the approximate nearest neighbor algorithm. This returns \( K \) approximate nearest neighbors, which are then sorted according to their Euclidean distance from the query vector. Depending on the chosen level of retrieval, information about the frames, shots and videos respectively are returned.

Client-server architecture: For the retrieval system we use a standard client-server architecture (section 5.6). While the client interfaces with the user, the server performs the back-end operations of fast search and ranking, stores the randomized K-D tree structure, and serves the thumbnails, frames and videos.

5.3 Video Processing

The videos are processed off-line over a series of three stages: (i) shot detection using a standard color histogram method [62]; (ii) upper-body detection to localize the people in each video frame and track them within the shot; (iii) human pose estimation (HPE) algorithms to determine the human pose within the upper-body detection area. In the current system we have 18 videos, 17 are Hollywood movies and one is a season of the TV serial ‘Buffy the Vampire Slayer’. In all, there are about 3 million frames. The statistics are given in table 5.1.

5.3.1 Upper Body Detection and Tracking

An upper body detector (UBD) is run on every frame of the video. An UBD algorithm detects and gives a bounding box around the people in the image. These are often a prerequisite for human pose estimation algorithms [5, 27, 80].

Here, we use the publicly available detector of [28]. It combines an upper-body detector based on the model of Felzenszwalb et al. [34] and the OpenCV face detector [98]. Both detectors run in a sliding window fashion followed by non-maximum suppression, and output a detection window \((x, y, w, h)\). To avoid detecting the same person twice, each face detection is then regressed into the coordinate frame of the upper-body detector and suppressed if it overlaps substantially with any upper-body detection. As shown in [31] this combination yields a higher detection rate at the same false-positive rate, compared to using either detector alone.
Table 5.1: Dataset statistics. We consider 18 movies for the retrieval system. For each video, the number of images, shots, tracks and human pose estimates are reported. Movies with * are abbreviations for ‘Buffy the Vampire Slayer’, ‘Four weddings and a Funeral’, ‘Living in oblivion’, ‘Lost in Translation’, ‘My Cousin Vinny’, ‘Desperately Seeking Susan’ and ‘A fish called Wanda’.

In order to reduce the false positives, the detections in a shot are grouped into tracks and short tracks are discarded. These tracks are obtained using the temporal association strategy described in [40]. Visually, the tracks form a continuous stream of a person’s bounding box in the shot.

5.3.2 Human Pose Estimation and Representation

We use here the algorithms of Eichner and Ferrari [27] and Yang and Ramanan [103]. Both of these have publicly available implementations. The algorithm [27] is run on each upper body detection in a track. The upper body detection is used to determine the scale of the person. The algorithm [103] is run over the whole frame to obtain multiple pose estimates. As many of the pose-estimates are false-positives, we use the upper body detections to filter them out. In detail, all detections returned by [103] which overlap less than 50% with any UBD detection are discarded. Overlap is measured using the standard “intersection over union” criterion [33]. Each of the pose estimate is then passed through pose evaluator described in section 4.4. Poses which survive this filtering step are then considered for retrieval.
For representing the pose, the $12D$ vector pose representation described in section 4.5 is used. The representation is constructed by taking the $sine$ and $cosine$ of the six body parts obtaining $12D$ vector.

### 5.4 Pose Matching

Consider any part $i$ of the poses A and B. Let the angles ($0^\circ \leq \theta \leq 360^\circ$) be $\theta_A^i$ and $\theta_B^i$ respectively. We measure the dissimilarity between the poses of the parts of A and B as the negative cosine of $|\theta_A^i - \theta_B^i|$. The negation ensures that the dissimilarity monotonically increases with $\Delta \theta = |\theta_A^i - \theta_B^i|$.

In this work, we are interested in large scale nearest neighbor search. Most standard algorithms for this task are based on the Euclidean distance and require a feature vector for each sample. By encoding the angles as the vectors $v_A = (\cos(\theta_A^i), \sin(\theta_A^i))$ and $v_B = (\cos(\theta_B^i), \sin(\theta_B^i))$, $-\cos(\theta_A^i - \theta_B^i)$ is obtained as the squared distance between the vectors i.e. $(v_A - v_B)^2 = 2(1 - \cos(\theta_A^i - \theta_B^i))$.

### 5.5 Pose Retrieval

After the database has been pre-processed off-line with the stages detailed above, it is ready to be searched by the user. The user enters a query (section 5.2.1) which is converted to the same 12-D representation as the poses in the database (section 4.5).

For searching the query in the database we use the approximate nearest neighbor (ANN) method proposed by Silpa-Anan et al. [83]. The method has one of the best recall rates among algorithms that index high dimensional data with significant search speed gain over exhaustive search [65]. The method organizes the database as a collection of randomized K-D trees. To construct a K-D tree, the data is split at the median value of a pre-assigned dimension $d$ and the two halves are passed down to the left and right subtrees. This procedure is recursively followed to further split the data. In [83] the splitting dimension is randomly chosen among the $T$ dimensions with the largest variances. The technique constructs a set of randomized trees by running the random selection multiple times.

Given the randomized trees, the $K$ nearest neighbors of a query are obtained as follows. The query point is sent to all of the trees. Each tree then returns a different set of approximate nearest neighbors (due to the randomness in the construction of each tree). First the query is passed down through all the trees to obtain the initial set of nearest neighbors. Then the search is systematically expanded into other nodes by backtracking from the leaf nodes and exploring the nearby alternative paths. This operation is efficiently performed by maintaining a list of nodes from all the trees in a priority queue. The node which is closest to the query is explored first. This procedure is repeated until $K$ nearest neighbors are obtained. The nearest neighbors returned by the algorithm (with duplicates removed) are then sorted based on the Euclidean distance to the query.

**Implementation details** We discuss the important parameters of the number of trees used and the number of nearest neighbors returned from each below in the evaluation. To construct a forest of two
Figure 5.2: **Screen-shot of the pose retrieval system.** The elements of the web page such as the three query modes, options for selecting the database and the level of retrieval and video player, amongst other things, are indicated by text annotations and pointers in light red color. The interactive stickman can be moved around and the results are instantly updated as a ranked list. Clicking on a thumbnail plays a video of that shot.

K-D trees over a database of 54,000 pose estimates requires 350 MB of memory and a retrieval time of 5ms for 2000 poses.

### 5.6 Client-Server Architecture

The system is implemented using a standard client-server architecture. The client interfaces with the user and the server performs the back-end operations. The functionalities of these components are described in detail for the stickman query mode, and summarized for the other two query modes.

#### 5.6.1 Client

The client provides an interface for the user to interact with the system. It is responsible for taking the query from the user, sending it to the server and displaying the retrieved results returned by the
server. Figure 5.2 shows the pose retrieval web demo that we have built. The upper-left corner shows the tabbed interface for three query modes query-by-stickman, query-by-Kinect and query-by-image respectively. In the query-by-stickman mode, the user can choose any pose by moving the arms of the interactive stickman in the image. As the user moves the stickman, the client continuously sends coordinates of stickman as queries to the server. The server responds in real time with the database poses that best match the query. This provides a gratifying and interactive pose search experience. In the query-by-Kinect mode as the user moves, the skeleton of the user output by the Kinect sensor is uploaded to the server. We observe that the Kinect’s pose estimation is stable when the user is in full view of the Kinect. To further improve the stability, the skeleton is uploaded to the server only when the Euclidean distance between successive poses is greater than 5 pixels. In the query-by-image mode, the query image is uploaded to the server. The server then detects human poses in the image and relays back the best human pose to the client. The user is also given the option select the database and level (frame/shot/video) to retrieve at. The thumbnail of the matching result, the position of the shot and the movie to which it belongs are displayed. If the user clicks on any thumbnail, the corresponding video shot is played.

The client is implemented as a web application using the AJAX framework with javascript as the language. The client independently handles the user interaction, communication with the server and the results display. The input given by the user is passed onto the server as an XML request. Then the incoming XML data, a ranked list, from the server is processed. The corresponding thumbnails are requested from the server and displayed. To view the video clip of any retrieved result, the user is given an option to click on the corresponding thumbnail.

5.6.2 Server

The server processes a request from the user and returns the top ranked results that best match with the query. After receiving a query from the client, the server: (i) constructs the 12D pose representation, and (ii) retrieves the best matching results from the database and returns relevant information. Upon further request, it (iii) returns the video clip of the requested shot.

If the input is a stickman, the 12D representation is simply computed by measuring the orientation of the body parts. If the input is an image, the server applies the human pose estimation algorithms and then derives the 12D descriptor just as if it were a database video frame. If the input is a Kinect skeleton, the locations of the neck, head, base of the spine, two wrists, two elbows and two shoulders are used to construct the stickman (see the Kinect example in figure 5.1) The stick corresponding to the torso, for example, is constructed by forming a line segment with the end points as location of the neck and the base of the spine.

The server then retrieves 100 best approximate nearest neighbors. For each element in the ranked list the thumbnail path, the video name, the position of the shot to which it belongs to are sent back as result. If the user has selected shot or video level retrieval, then the nearest neighbors are grouped (on shot or video), and the single pose estimate most similar to the query in each group is returned.
Table 5.2: Approximate nearest neighbor search vs Exhaustive search. The recall of the top 100 ground truth matches is averaged over 1000 queries.

<table>
<thead>
<tr>
<th>Experiment A: Recall, for ((m \times 100)) NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple (m)</td>
</tr>
<tr>
<td>Recall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment B: Recall using (N) trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num trees</td>
</tr>
<tr>
<td>Recall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment C: Recall vs database size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataBase size</td>
</tr>
<tr>
<td>Recall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment D: Search speed ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataBase size</td>
</tr>
<tr>
<td>Search speed ratio</td>
</tr>
</tbody>
</table>

5.7 Experimental Evaluation

We evaluate the real time pose search system on a large video collection consisting of 18 video and about 3 million frames. Of these 18 videos, 17 are Hollywood movies and one is a season of the TV serial ‘Buffy the Vampire Slayer’. The statistics are given in table 5.1.

We first evaluate how accurately does the ANN method retrieve the K nearest neighbors, with exhaustive search as reference. The number of trees used and number of nearest neighbors to be returned are also discussed. We then evaluate how many of the samples retrieved by the pose search system are indeed a true match to the query.

5.7.1 Approximate Nearest Neighbor

To evaluate the ANN algorithm’s performs compared to exhaustive search, 1000 random pose estimates are sampled from the whole movie database as queries. Each query is searched in the database using both exhaustive and approximate nearest neighbor search. In the exhaustive search, the query pose is compared using the Euclidean distance to all the elements in the database and the best 100 pose estimates are retained. Next, the search is repeated with the ANN algorithm. These ANN algorithms suffer from low recall. To address this, the standard practice is to retrieve more points and retain the desired number of nearest neighbors closest to the query. In this experiment, the desired number of nearest neighbors is 100 and multiples of 100, \(100 \times m\) where \(m > 1\), are retrieved using ANN. The performance is measured using recall at 100, i.e. the proportion of the ground-truth nearest neighbors that are in the top 100 neighbors returned by the ANN algorithm.

**Experiment A:** In the first experiment, the recall of the ANN is observed while varying the multiple \(m\). The multiple \(m\) is varied with values \(\{1, 5, 10, 20\}\), but the number of trees is fixed to 2. As shown in table 5.2 (‘Experiment A’), the recall at 100 rapidly improves with \(m\).
Figure 5.3: Pose retrieval evaluation: The poses displayed in the second row are used to evaluate the performance of the system. The corresponding numbers below are the precision values of the top 12 results. The third and fourth row respectively are the precision values using the ‘Filter II (Fl2)’ and ‘Hybrid pose estimates (Hpm)’ methods. The mean precision over the ten pose queries is 31.7% and 32.5% respectively.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fl2</td>
<td>91.7</td>
<td>83.3</td>
<td>41.7</td>
<td>41.7</td>
<td>25</td>
<td>16.7</td>
<td>8.3</td>
<td>8.3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hpm</td>
<td>91.7</td>
<td>75.0</td>
<td>8.3</td>
<td>50.0</td>
<td>41.7</td>
<td>8.3</td>
<td>25</td>
<td>8.3</td>
<td>8.3</td>
<td>8.3</td>
</tr>
</tbody>
</table>

Experiment B: In the second experiment the number of trees, $N$, is varied, but $K$ is fixed at 2000. Thus on average, $K/N$ neighbors are requested from each tree. As shown in table 5.2 (‘Experiment B’), the performance is best for two trees.

Experiment C: In the third experiment the size of the database is varied, for $K = 2000$ and $N = 2$. As shown in table 5.2 the recall is largely unaffected by the database size.

Experiment D: In the fourth experiment the size of the database is varied, for $K = 2000$ and $N = 2$. As shown in table 5.2 (‘Experiment D’), the speed gain over exhaustive search is orders of magnitude better and significantly improves with the size of the database.

5.7.2 Retrieval

The pose retrieval system is quantitatively evaluated by posing ten queries to the system and measuring the precision of the retrieved results on the first page of the web application, i.e. the top 12 returns. The queries are chosen to cover a diverse set of poses that people often make.

For this evaluation, we use all the 18 videos listed in table 5.1 and process them as described in section 5.3. We then employ the two best methods (section 4.4), ‘Filter II’ and ‘Hybrid pose estimation’, separately to improve the precision. Figure 5.3 shows the queries posed to the system and the corresponding precisions for both the methods, and figure 5.4 shows the first five retrieved results for the three top performing queries.

Analyzing the precision values in figure 5.3 we see that for the best query both ‘Filter II (Fl2)’ and ‘Hybrid pose estimation (Hpm)’ methods have 91.7%. Similarly, the mean precision is nearly the same at 31.7% and 32.5% respectively. But a closer look at the precision values of individual queries reveals the differences between both the methods. While ‘Filter II’ method retrieves high precision results for many queries, particularly for whom many examples in the database, it fails to retrieve any pose for the last two queries. ‘Hybrid pose estimation’ manages to both retrieve poses at high precision and also retrieve the less frequent ones (queries 9 and 10). In fact for the queries 9 and 10, as far as we can tell there is only one example each in the collection.
Note how these include diverse poses, with arms next to the torso, overlapping with the torso, and stretched out from the torso. This illustrates the versatility of the system. The major impediment to the performance of the system at the moment is the failures of the pose estimation algorithms, which both the methods ameliorate to the extent possible.

<table>
<thead>
<tr>
<th>Query Pose</th>
<th>Top retrievals</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Query Pose" /></td>
<td><img src="image" alt="Top retrievals" /></td>
</tr>
</tbody>
</table>

Figure 5.4: **Pose retrieval examples**: For the query poses displayed in the first row, five retrieved results are displayed.

### 5.8 Summary

In this chapter, we described the prototypical and large scale pose retrieval system. We have designed a general, scalable, real time pose retrieval system for accessing frames, shots and videos. Even with the current state of pose estimation algorithms, with 50k examples there are many interesting poses that can be discovered and retrieved. We expect the performance of the system to improve over time as the performance of pose estimation algorithms improves.

Although we have demonstrated the idea for videos, a similar system could equally well be developed for large scale image datasets.