Who are you? – real-time person identification

Nicholas Apostoloff and Andrew Zisserman
Department of Engineering Science, University of Oxford
{nema,az}@robots.ox.ac.uk

Abstract

This paper presents a system for person identification that uses concise statistical models of facial features in a real-time realisation of the cast identification system of Everingham *et al*. [7]. Our system integrates the cascaded face detector of Viola and Jones with a kernel-based regressor for face tracking, which is trained on-line when new people are detected in the video stream. A pictorial model is used to compute the locations of facial features, which form a descriptor of the person's face. When sufficient samples are collected, identification is performed using a random-ferns classifier by marginalising over the facial features. This confers robustness to localisation errors and occlusions, while enabling a real-time search of the database. These four different processes communicate within a real-time framework capable of tracking and identifying up to 5 people in real-time on a standard dual-core 1.86GHz machine.

1 Introduction

Have you ever been greeted by a person that you have only met briefly, perhaps at a previous BMVC, and wished that you had a magic little device that whispered their name in your ear? The goal of this paper is to implement such a device using a modern multi-core computing architecture, with the only input a video stream obtained from a standard web-cam. To make the problem even more challenging, this must be accomplished in real-time and be able to handle variations in their appearance due to lighting, scale, pose, expressions, image quality, motion blur, etc. The practical goal is the identification of people as they approach the camera, which has the fortunate consequence that we need only consider near-frontal faces. This has important applications in humanoid robotics, surveillance, human-computer interaction, tourism or as an aid for those with impaired vision (or memory).

We build this device using standard off-the-shelf components. The input video stream is captured by a web-cam which may hang around a person's neck or be embedded in their clothing. A standard dual-core laptop, which may be carried in a backpack, is used to process the video stream. The system, which we name *IDU*, stores a repository of people that you have met previously, including their name, the date and location that you met them and a video of that initial meeting. This information is collected automatically by the system when the user chooses to add a new person to the repository. A short 250 frame video is captured and then processed to extract an appearance model of the person.

1.1 Outline

The task of person identification can be broken into three sequential stages: detection (§3.1), tracking (§3.2) and identification (§3.3)—see figure 1. The goal is then to design

BMVC 2007 doi:10.5244/C.21.48

Figure 1: **Overview and example of the real-time IDU system**. The detection, tracking and identification of each person in a web-cam video stream is accomplished in three threads. When a new person is detected by the detection thread, a tracking thread is instantiated to track that person. (top) The tracking thread estimates the pose of the person's face at 15fps and proceeds sequentially (indicated by the dashed lines). (bottom) The feature localization and identification thread depends only on instances of a person's face given by the tracker (indicated by the solid arrows) and runs at 2fps. As 10 facial descriptors are required for reliable recognition, identification takes up to 5s per person. The crosses indicate the facial features used for recognition.

a system that can accurately track multiple people at 15fps and identify them in a timely manner. In our case, it takes approximately 5s to identify a person from the moment they are first detected. An additional design constraint is to ensure that the database search and the entry of new people into the database is scalable.

We build on previous approaches that have used frontal face detections in combination with textual information to automatically identify cast members in TV or film material, in an off-line, batch process [7]. In particular we make three contributions. First, we have implemented a real-time realisation of [7] for person recognition. This in itself, required a significant modification of the original system to guarantee real-time performance. Second, the tracking method used in [7] to group face detections in order to label cast members, does not lend itself to a real-time implementation. To mitigate this computational complexity, a simple kernel based regressor is developed that can be trained for each detected person and is capable of tracking the pose over time. Training the tracker for each individual confers the benefit of greater accuracy while requiring fewer computational resources. Third, a random-ferns classifier is used to identify individuals by marginalizing over a set of facial features, alleviating the effect of localisation errors and occlusions. Random-ferns [15] are a variation on the random-forest classifier first introduced in [1] and developed further in [5]. Their recent popularity owes much to the tracking application of [11]; however, they have also been applied to object recognition in [14, 25, 26]. The advantage of randomized trees/ferns is that they are much faster in training and testing than traditional classifiers (such as an SVM), without a loss in performance.

We show that our real-time classification method outperforms the exhaustive search criteria of the original paper, while requiring far fewer computation resources. A typical search using the random-ferns classifier takes approximately 0.3ms per face in a track, while the exhaustive *min-min* search of [7] takes approximately 200ms per face[1].

[1]The timing results depend on the number of people in the database and in the case of the *min-min* search, the number of instances of each person in the database. In this case, the timing results are presented for the Buffy the Vampire Slayer dataset described in §5 when the system is trained using episode 05-02.

2 Previous work

Automatic person recognition systems have been approached in two distinct strands of research in the literature. A number of authors attempt to identify people in *real-time* systems for surveillance [6] and human-computer interaction [21] using facial appearance [16], gait [17], thermal imagery [2] and even clothing [20], but frequently constrain the environment in which they operate (*i.e.* stationary cameras [17], fully frontal faces and consistent lighting [16, 22], etc.). In a separate strand of research, *off-line batch* processing systems have been used to identify cast members in TV episodes or movies by clustering frontal faces [10], or to retrieve instances of particular characters [3, 19] in a similar fashion to internet search engines. These methods have generally shown good performance over large volumes of difficult data with dramatically varying conditions and we have chosen to adapt these into a real-time application.

Review of Everingham *et al*. [7]. The recent cast identification system of Everingham *et al*. [7] showed excellent performance over the difficult conditions often found in TV or film material and is the basis upon which this real-time system is built. While they consider the more difficult problem of automatically building a database of cast members using both visual and textual information, we review only their contributions relevant to the work presented here.

In [7], an off-line face tracking approach first computes all face detections in the video stream using a cascaded face detector [23] with a high threshold designed to reduce the number of false positives. To reduce the computational burden of identifying each individually detected face[2], face detections linked by features points tracked using the Kanade-Lucas-Tomasi [18] tracker are grouped together. In addition to reducing the volume of data to be processed, this also allows stronger appearance models to be built for each character, as multiple instances of the character are grouped in each track. For this reason, the granularity of the identification procedure is reduced from matching individual faces to matching sets of faces grouped by tracks over the video sequence.

The appearance models are built from 13 local exemplar-based descriptors at the corners and centres of the eyes, nose and mouth. Using local descriptors (in this case, normalized image patches) confers the benefit of robustness to pose variation, lighting and partial occlusion when compared with global descriptors such as eigenfaces [16]. A descriptor for each face $\mathbf{F}$ is then the concatenation of each of the facial feature descriptors $\mathbf{f}_i$. The features are located using a generative model of the feature positions combined with a discriminative model of feature appearance in an extension of the pictorial tree in [9]. Given a database of face tracks, each consisting of a set of facial descriptors $\mathscr{F}_t = \{\mathbf{F}_i\}_t$ with a corresponding label λ_t, a new face track $\mathscr{F}$ is matched against the set of all faces $\mathscr{F}(\lambda_i)$ with the label λ_i using a *min-min* metric

$$d(\mathscr{F}, \lambda_i) = \min_{\mathbf{F}_j \in \mathscr{F}} \min_{\mathbf{F}_k \in \mathscr{F}(\lambda_i)} ||\mathbf{F}_j - \mathbf{F}_k||. \tag{1}$$

For simplicity, we ignore the clothing descriptor they use in combination with the facial descriptors. For a given track $\mathscr{F}$, the likelihood that it corresponds to face λ_i is

[2] A typical episode of a TV series contains over 20,000 face detections, even though these generally arise from a few hundred "tracks" of a particular character.

then $p(\mathscr{F}|\lambda_i) = Z^{-1}\exp\{-d(\mathscr{F},\lambda_i)^2/(2\sigma^2)\}$. The posterior is then obtained by applying Bayes rule to get $p(\lambda_i|\mathscr{F}) = p(\mathscr{F}|\lambda_i)/\sum_j p(\mathscr{F}|\lambda_j)$. A face track is then labelled with the λ_i that maximizes the posterior.

A brief introduction to random forests/ferns. A random forest multi-way classifier consists of a number of trees, with each tree grown using some form of randomization. The leaf nodes of each tree are labelled by estimates of the posterior distribution over the image classes. Each internal node contains a test that splits the space of data to be classified. An image is classified by sending it down every tree and aggregating the reached leaf distributions. Randomness can be injected at two points during training: in sub-sampling the training data so that each tree is grown using a different subset; and in selecting the node tests. In this paper we do only the later.

Suppose that T is the set of all trees, C is the set of all classes and L is the set of all leaves for a given tree. During the training stage the posterior probabilities ($P_{t,l}(\lambda(\mathbf{F}) = c)$) for each class $c \in C$ at each leaf node $l \in L$, are found for each tree $t \in T$. These probabilities are calculated as the ratio of the number of descriptors $\mathbf{F}$ of class c that reach l to the total number of descriptors that reach l. To reduce the effect of unbalanced datasets, the contribution of each descriptor of class c is further normalized by the number of samples of class c in the training dataset. In order to classify a new test descriptor, it is passed down each random tree until it reaches a leaf node. All the posterior probabilities are then averaged over the trees and the arg max is taken as the classification of the input descriptor.

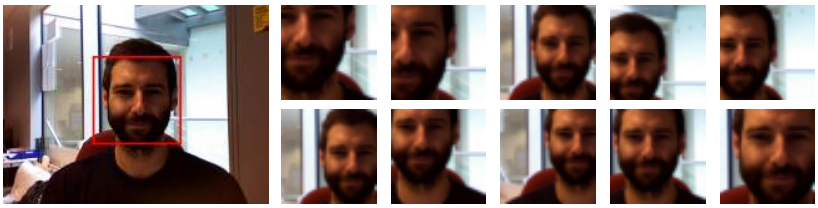
To increase the speed of the random forest Ozuysal *et al*. proposed random-*ferns* classifiers [15]. In the case of ferns, there are an ordered set of nodes, and the node test is applied to the whole training data set. In contrast, in random forests only the data that falls in a child node is taken into account in the test. As in random forests, leaves store the posterior probabilities. At each node in the fern set, a test gives a binary result. The result of each test and the ordering on the set defines a binary code for accessing the leaf node. So if a fern has N nodes, it will have 2^N leaves. The advantage of ferns over forests is that it is not necessary to store the intermediate nodes (of a tree).

3 Visual processing methods

This section describes the three stages of visually identifying a person: detection, tracking and recognition.

3.1 Face detection

The first stage in processing is face detection where a frontal face detector is run once per second to obtain new tracks. This detector is based on the Viola-Jones [23] cascaded frontal face detector and enhancements by Lienhart [12] and is moderately fast, being able to process each frame in ≈ 200 ms. The output of the cascaded face detector is a set of face poses $\mathbf{p}_d = (x_d, y_d, w_d)^\top \in \mathscr{P}_D$ where the pose is a vector containing the spatial coordinates (x_d, y_d) of the centre of the face and its scale w_d. When a new face $\mathbf{p}_d$ is detected, it is compared to all faces currently being tracked $\mathbf{p}_t \in \mathscr{P}_t$ and a new track is only started if $\mathbf{p}_d$ does not overlap any $\mathbf{p}_t$ by more than a quarter of its area.



(a) Face detection (b) Training samples

Figure 2: **Training the regressor from face detections**. Given a set of input faces from the localized face detector, (a), a set of 192 $(\mathbf{v}, \tilde{\mathbf{t}})$ pairs (b) are artificially sampled over the state-space of face poses for training.

3.2 Face tracking

Inspired by the relevance vector machine (RVM) tracker of Williams *et al*. [24] we have implemented a regularized kernel-based face tracker using radial basis functions (RBF). This choice, over that of the RVM, is motivated by the lack of sparsity that is generally present within an RVM trained for face tracking. This is supported by the general regression tracker of Mayol and Murry [13] who also claim that the sparsity induced by the RVM is detrimental to tracking performance because the set of kernels after training is incomplete.

After a new face $\mathbf{p}_d$ has been detected, the tracker enters a training state where it collects samples of the person's face over a number of frames using a localized version of the cascaded face detector, where the search region at frame $i+1$ is centred on the detected face from frame i and has a width equal to $1.2w^{(i)}$. After a preset number of face exemplars are detected, a set of training image/target pairs $\{\mathbf{v}, \tilde{\mathbf{t}}\}$ is generated by sampling the 3D space of possible translations and scales over a uniform grid with $N = 192$ elements (see figure 2). Here, $\tilde{\mathbf{t}} = (dx, dy, dw)$ and lies in the range $(\pm 10, \pm 10, \pm 5)$ in a canonical reference frame where the face image has dimensions 40×40 pixels. Each training image patch $\mathbf{v}$ is then scaled to the reference frame, Gaussian blurred and histogram normalized to reduce the effect of lighting. A kernel-based regressor is then trained for each dimension of the state-space separately.

The kernel-based regressor models the target variable $t(\mathbf{v}, \mathbf{w})$ as a linear combination of a fixed number of $N+1$ nonlinear functions $\phi_j(\mathbf{v})$ of the input variables $\mathbf{v}$

$$t(\mathbf{v}, \mathbf{w}) = \sum_{j=0}^{N} w_j \phi_j(\mathbf{v}) = \mathbf{w}^\top \phi(\mathbf{v}), \tag{2}$$

where $\mathbf{w} = (w_0, \ldots, w_N)^\top$ and $\phi = (\phi_0, \ldots, \phi_N)^\top$ with $\phi_0 = 1$ to account for bias. Here, N is the number of training samples and the kernel function ϕ_j is a radial basis function centred on the training vector $\mathbf{v}_j$. Given a set of N $(t_j, \mathbf{v}_j)$ training pairs, the weights $\mathbf{w}$ can be estimated by placing a quadratic regularizer on the weights and by minimizing the sum-of-squares error function as in *ridge regression* [4] to get $\mathbf{w}_R = (\lambda \mathbf{I} + \Phi^\top \Phi)^{-1} \Phi^\top \mathbf{t}$. Here, the ith and jth element of Φ is the result of applying the jth nonlinear function to the ith input vector $\phi_j(\mathbf{v}_i)$ and $\mathbf{t} = (t_1, \ldots t_N)$, and λ is a system parameter that controls the smoothness of the solution.

The input to each regressor at frame $i+1$ is an image patch extracted from the current image *at the location of the face from frame i*, $\mathbf{p}_i = (x_i, y_i, w_i)$. This is extracted in the same manner used to calculate the training vectors. The output of the three regressors at frame

$i+1$ is then an estimate of the change in the pose $(dx, dy, dw)^\top$ in the canonical reference frame. Hence, the pose at frame $i+1$ is then $\mathbf{p}_{i+1} = (x_i + \hat{dx}, y_i + \hat{dy}, w_i + \hat{dw})$, where $(\hat{dx}, \hat{dy}, \hat{dw})$ is the transformation of (dx, dy, dw) from the canonical reference frame into image space.

The regression framework estimates scale and translational offsets, but does not estimate rotations of the head; however, depending on the variability of the training vectors, it can accurately track a person's face with slight out-of-plane rotations (figure 1). This exhibits similar fail-case scenarios to the RVM tracker where tracks drift off the target when the person's head is rotated more than 30^{o} about the vertical axis. Rotations about the horizontal axis tend to be small and do not effect the results significantly.

3.3 Face recognition

The face recognition system uses the pictorial structure model of Everingham *et al*. [7] to locate 9 facial features at the corners of the eyes, nose and mouth (figure 1). Four additional features are added at the centre of both eyes, mouth and nose. The face region defined by the facial features is normalized with respect to a canonical face to reduce the effects of scale and out-of-plane rotations of the head. An affine transformation is computed between the canonical feature set and the facial features. Using the affine transformation, regions that were circular in the canonical reference frame are extracted from the corresponding elliptical regions in the tracked face. These 13 image patches ($\mathbf{f}_{1\ldots13}$) are then normalized to have zero mean and unit variance, and are concatenated to form a single vector that represents the persons face ($\mathbf{F}$) as in §2. The canonical reference frame has dimensions 80×80 pixels while the extracted image patches have a diameter of 15 pixels.

To classify the faces we use a random-ferns classifier with 40 ferns of 17 levels, where the test at node n is a simple comparison between two elements (p_n and q_n) of the descriptor $\mathbf{F}$, which are chosen at random when building the tree. In this case, elements $\mathbf{F}(p_n)$ and $\mathbf{F}(q_n)$ are compared using the less-than operator $(\mathbf{F}(p_n) < \mathbf{F}(q_n))$.

As mentioned previously, the result of sending a new test descriptor down a random-ferns classifier is a posterior distribution over the classes ($P(\lambda(\mathbf{F}) = c)$). In the simplest case, the label given the test descriptor can be taken as the argmax_c of this distribution; however, in the situation here, we have the additional knowledge of multiple faces in each track $\mathbf{F}_i \in \mathscr{F}$. Hence, two options are available when classifying a track. First, we can take the max over the set of posteriors and faces

$$\lambda_{\text{max-max}} = \underset{c \in C}{\text{argmax}} \max_{i \in \mathscr{F}} P(\lambda(\mathbf{F}_i) = c), \tag{3}$$

which we label the *max-max* result. In the second method, we marginalize over the faces in the data set to get the *max-sum* result

$$\lambda_{\text{max-sum}} = \underset{c \in C}{\text{argmax}} \sum_{i \in \mathscr{F}} P(\lambda(\mathbf{F}_i) = c). \tag{4}$$

In an alternative approach, we train a random-ferns classifier for each facial feature. To classify a new descriptor $\mathbf{F}$, we first marginalize over the facial features $\mathbf{f}_i$ by summing the posteriors returned from each classifier $P(\lambda(\mathbf{f}_i) = c)$, and then apply either the *max-max* or *max-sum* variant. We label these classifiers *max-max-sum* and *max-sum-sum* respectively, and compare the classification performance of these variants in section 5.

4 Architecture

With the introduction of multi-core computing architectures, it becomes possible to process multiple CPU intensive tasks simultaneously, enabling a new branch of real-time applications for video processing. The IDU algorithm takes advantage of these architectures to capacitate real-time person tracking and identification, by segregating the time-sensitive tracking tasks from the slower identification task in a multi-threaded environment. While the kernel-based regressor presented in this paper can efficiently track multiple people in real-time on a single core, the localization of the facial features required for recognition is an expensive process, requiring up to 500ms per frame on a 1.86GHz machine. To mitigate this complexity, the crucial tracking tasks are computed on separate high-priority threads, while relegating the slow feature localization and identification tasks into the background on low-priority threads (figure 1). In live experiments, up to 5 people can be tracked at 15fps and while feature localization can only be computed at 2fps, only 10 faces are required for recognition, and reliable recognition can then be computed in less than 5s per person.

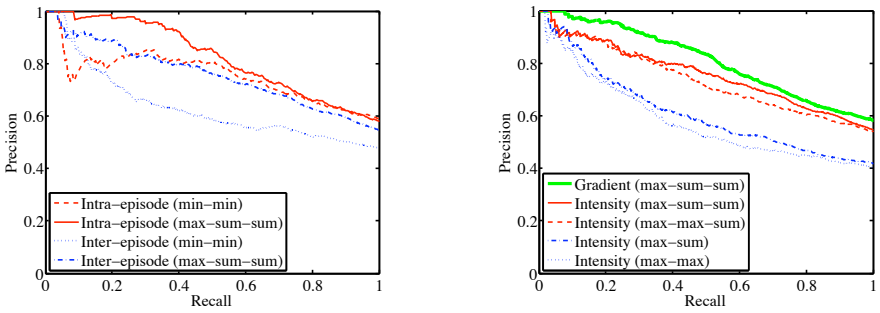
The detection, tracking and subsequent identification of a person proceeds in three phases. First, a global face detection thread using a variant of the Viola-Jones cascaded face detector runs once per second to detect new faces. Second, when a tracking thread has been initialized with a newly detected face, it enters a training state where it collects $N = 12$ face samples by performing a localized cascaded face detection. When enough face samples are collected, it trains a kernel-based regressor for each dimension of the state-space (*i.e*. the pose). Depending on the number of training vectors used, this can track an individual face at a cost of around 13ms per frame. Third, each tracked face is processed by an instance of the facial feature localisation algorithm in a separate thread and the result of this is a set of face descriptors $\mathscr{F}$. When a suitable number of descriptors is collected to make a model of a face (typically more than 10), the database is searched for a matching face model.

Lost tracks are handled directly by the identification system, which returns a score from the pictorial structure used to locate facial features. If subsequent faces over a period of 2 seconds in a track return low scores from the identification module, then the track is deemed lost and terminated.

While, this system has been designed to track and identify multiple people in real-time, it must also provide a satisfying user experience. To accomplish this it is necessary to separate the processing threads from the GUI and hence, there is a total of $2 + 2N$ threads running when tracking N people. This includes one thread for the GUI, one thread for the face detector and a separate tracking and identification thread for each person.

5 Results and discussion

In this section we compare various methods of training and scoring the random-ferns classifier, with the previously used *min-min* classifier of [7], and compare on the data they used. This consists of episodes 2 and 5 from season 5 of Buffy the Vampire Slayer, and contains 2462 tracks over 12 people. The tracks vary in length from 1 to 404 frames, and there are 53032 labelled face detections in the database. Everingham *et al*. provide their tracked ground truth data for these videos at [8]. In the case of processing the Buffy videos, the experimental setup was exactly the same as the IDU system except that the



(a) Comparing the *min-min* search with the random-ferns classifier

(b) Random-ferns classifier using **inter-episode** training

Figure 3: **Precision-recall curves for intra and inter-episode experiments**. Recall is the proportion of tracks assigned a label at a given confidence level, while precision is the proportion of those tracks assigned correct labels. (a) compares the *min-min* distance with the random-ferns classifier for *intra*-episode and *inter*-episode experiments. (b) compares the different configurations of the random-ferns classifier in *inter*-episode experiments and also compares gradient descriptors versus normalized intensity patches for the best configuration (*max-sum-sum*). See text for details.

video stream was read from disk instead of the web-cam.

Exemplar-based descriptors have performed well in experiments using TV or film material where all of the data (*i.e.* all face detections and tracks) is available for categorization [3, 7, 10, 19]. In these cases, many shot transitions are of the form ABAB, where A and B are two different shots and the scene alternates between the two. The remarkable performance of these techniques is partly the result of the repetitive nature of these shot transitions. The repeated shots contain face detections that are almost identical and can be matched accurately using the normalized patch descriptors and simple metrics; however, when trained and tested on separate episodes (or even different parts of the same episode) their performance degrades (see below). Consequently, these descriptors do not generalize well by themselves to our target application where a person's appearance can change significantly from that contained in the database. We mitigate these deficiencies by using a variation of the random-ferns classifier and by marginalizing over the facial features and faces in a track.

Two experimental scenarios are considered. First, we train the classifiers using 160 tracks from episode 05-02 which were obtained in [7] using speaker detection and test the classifier on all other tracks that are at least 10 frames long (*intra*-episode). This is the same experimental scenario examined in [7] except that no clothes descriptor is used and the speaker detected tracks are not included in the precision-recall curves. The speaker detection data contains 9.5% errors, and has 12 characters, one of which is an "other" class containing a mixture of characters. Second, we use all the tracks from a single episode to train the classifiers and then test them on a different episode (*inter*-episode). In figure 3(a) we compare the results of the *min-min* classifier of Everingham *et al*. with the *max-sum-sum* random-ferns classifier. The *min-min* classifier performs well in the intra-episode experiment due partly to the ABAB shots that are present; however, its performance degrades in the inter-episode experiments for larger recalls. In both cases, the *max-sum-sum* classifier outperforms the *min-min* classifier by a significant margin. For example, in the intra-episode experiments, the *max-sum-sum* obtained $97 \pm 2\%$ precision at 20% recall, while the *min-min* classify obtained 81% precision. In Figure 3(b), we compare the dif-

ferent variations of the random-ferns classifier and clearly see that marginalizing over the facial features again improves performance. This may be explained by the stability of the feature localisation algorithm and the effects of occlusion. It was found that while the eyes were quite stable features, those around the mouth and nose were quite unstable. Unstable features would naturally be evenly distributed over the leaf nodes when training the random-ferns classifier resulting in flat posteriors. On the other hand, stable features would frequently take the same paths down the random-ferns and would result in peaks in the posterior distributions. In a similar fashion, occluded features would traverse arbitrary paths in the classifier leading to leaf nodes that were untouched by the training data. The technique was further extended to use gradient descriptors instead of the normalized image patches, which further increased the precision. The gradient descriptors were constructed by concatenating dI/dx and dI/dy computed over each image patch.

Similar trends as found in [11, 15] were observed in the classification performance when increasing the number of ferns and the number of levels in each fern. The results improve significantly as the number of ferns approaches 30, but the improvement tapers off around 40, while a gradual improvement is observed when increasing the number of levels in the ferns above 15. This is limited however, by the prodigious memory requirements of storing the posteriors and 17 levels was found to be a good compromise.

Figure 4 shows a sample set of identification results using the *max-sum-sum* classifier. The first two rows show correct labelling of the data for a wide variety of scenarios with multiple people and varying lighting conditions and head pose. The bottom row shows a number of failure-scenarios which often result from the face detector not firing when there is a non-frontal face present or where there is extremely poor lighting. The final two images depict a scenario where Buffy was initially incorrectly labelled, but where the error was later rectified when additional face samples were collected.

It is emphasized again that while the results presented here were obtained by processing a video stream from disk, they were computed using the same real-time algorithm that is described in section 4 and shown in figure 1.

6 Conclusion

This paper has presented a system for person identification that is a real-time realization of [7]. Integrating a cascaded face detector with a kernel-based regressor that can track up to 5 people in real-time provides the basis for a person identification thread that runs in the background. We have shown that a random-ferns classifier achieves a far better classification performance than the *min-min* classifier of [7].

At this stage, visual feedback for testing is provided on the laptop display, but it is left for future work to implement auditory feedback via headphones. It is envisioned that these standard components could be combined into an ultra-portable device such as a PDA and the input/output embedded directly in the apparel of the person carrying the device. Later versions will automatically add a new person when they do not match anybody currently in the repository.

Acknowledgements. This work was supported by an EPSRC Platform grant.

References

[1] Y. Amit and D. Geman. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588, 1997.

Figure 4: **Sample identifications from episode 05-05**. The top two rows show correctly matched faces, while the bottom row shows some sample failure cases. See text for details.

[2] O. Arandjelović, R. Hammoud, and R. Cipolla. On person authentication by fusing visual and thermal face biometrics. *In Proc. AVSS*, pages 50–56, November 2006.

[3] O. Arandjelovic and A. Zisserman. Automatic face recognition for film character retrieval in feature-length films. In *Proc. CVPR*, pages 860–867, 2005.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., USA, 2006.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] Y.-T. Chien, Y.-T. Huag, S.-W. Jeng, Y.-H. Tasi, and H.-X. Zhao. Real-time surveillance system by use of the face understanding technologies. In *Proc. DICTA*, 2003.

[7] M. Everingham, J. Sivic, and A. Zisserman. Hello! my name is... buffy – automatic naming of characters in tv video. In *Proc. BMVC.*, 2006.

[8] M. Everingham, J. Sivic, and A. Zisserman. Automatic naming of characters in tv video. `http://www.robots.ox.ac.uk/~vgg/research/nface/`, 2007.

[9] P. Felzenszwalb and Huttenlocher D. Pictorial structures for object recognition. *Intl. Journal of Computer Vision*, 61(1):55–79, January 2005.

[10] A. W. Fitzgibbon and A. Zisserman. On affine invariant clustering and automatic cast listing in movies. In *Proc. ECCV*, volume 3, pages 304–320. Springer-Verlag, 2002.

[11] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE PAMI*, 28(9):1465–1479, 2006.

[12] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proc. IEEE ICIP*, volume 1, pages 900–903, September 2002.

[13] W. W. Mayol and D. W. Murray. Tracking with general regression, 2007. In press. See `http://www.cs.bris.ac.uk/~wmayol/research/index.html`.

[14] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *NIPS*, 2006.

[15] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Proc. CVPR*, 2007.

[16] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proc. CVPR*, pages 84–91, 1994.

[17] N. M. Rajpoot and Masood K. Human gait recognition with 3d wavelets and kernel subspace projections. In *Proc. HAREM*, Oxford (UK), September 2005.

[18] J. Shi and C. Tomasi. Good features to track. In *Proc. CVPR*, pages 593–600, 1994.

[19] J. Sivic, M. Everingham, and A. Zisserman. Person spotting: video shot retrieval for face sets. In *Proc. CIVR*, 2005.

[20] J. Sivic, C. L. Zitnick, and R. Szeliski. Finding people in repeated shots of the same scene. In *Proc. BMVC.*, 2006.

[21] K.-T. Song and W.-J. Chen. Face recognition and tracking for human-robot interaction. In *IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume 3, pages 2877–2882, 2004.

[22] M. Turk and A. P. Pentland. Face recognition using eigenfaces. In *CVPR*, pages 586–591, 1991.

[23] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. CVPR*, pages 511–518, 2001.

[24] O. Williams, A. Blake, and R. Cipolla. Sparse bayesian learning for efficient visual tracking. *IEEE PAMI*, 27(8):1292–1304, 2005.

[25] J. Winn and A. Criminisi. Object class recognition at a glance. In *Video Proc. CVPR*, 2006.

[26] J. Winn and J. Shotton. The layout consistent random field for recognizing and segmenting partially occluded objects. In *Proc. CVPR*, 2006.