Robust Real-Time Extreme Head Pose Estimation

Sergey Tulyakov, Radu-Laurenţiu Vieriu, Stanislau Semeniuta, and Nicu Sebe
Department of Information Engineering and Computer Science
University of Trento, Italy
{sergey.tulyakov, stanislau.semeniuta}@unitn.it, {vieriu, sebe}@disi.unitn.it

***Abstract*—This paper proposes a new framework for head pose estimation under extreme pose variations. By augmenting the precision of a template matching based tracking module with the ability to recover offered by a frame-by-frame head pose estimator, we are able to address pose ranges for which face features are no longer visible, while maintaining state-of-the-art performance. Experimental results obtained on a newly acquired 3D extreme head pose dataset support the proposed method and open new perspectives in approaching real-life unconstrained scenarios.**

I. Introduction

Head pose estimation (HPE) has been at the center of attention of many research groups in the past decade. This is mostly because it plays an important role in decoding human behavior, offering a valuable proxy for the gaze direction. Many applications can benefit from reliable HPE spanning domains such as human computer interaction, human behavior understanding, driving monitoring. Furthermore, HPE has become a necessary intermediate step for face registration or facial expression recognition, especially under large poses.

We investigate the HPE problem in the context of aiding decision-making process for elderly people while they move in complex unconstrained environments. Our scenario involves using a wheeled frame enhanced with a depth sensor oriented towards the subject. The novelty of our work comes directly from the placement of the camera, enforcing a point of view (see Fig. 6 for some examples) which guarantees to capture only the lower part of the user's face, while the upper part may be completely occluded (e.g., in case of up-tilt). In order to treat such extreme head poses, we propose fusing two competitive processing pipelines, combining their strengths and overcoming their weaknesses: one uses a personalized template created offline which ensures smooth and accurate real-time tracking performance but it is prone to losing track in some cases; the other is a frame independent head pose estimator that needs no initialization but its performance is poorer with respect to the tracking module. In this work, we consider a pose to be extreme when the rotation of the head results in occluded facial features.

The paper is structured as follows: Section II discusses relevant state-of-the-art methods. We describe our proposed system and explain how fusion is performed in Section III. Experimental results are presented in Section IV. Finally, we conclude the paper in Section V.

II. Related Work

While many approaches on HPE rely on 2D information (see [9] for a survey), they are seriously influenced by illumination changes, facial appearance etc., making it difficult to find simple enough features to meet real-time constraints. In contrast, depth or range images are not sensitive to illumination changes and therefore, are more suitable for analysis under various conditions. In this paper, we focus on 3D approaches, since they are closer in relevance to our work.

Fanelli et al. [5] describe an approach for head pose estimation based on random regression forests. To train a classifier they generate a dataset of 50K images using a 3D morphable model to randomly generate poses. The percentage of testing examples that were correctly identified within the threshold of 10° degrees is 90.4%, which is a very promising result. However, the case of extreme head orientations is not covered in their study. In [2] the authors propose a novel shape signature to help identifying nose position in range images. Using parallel computing, they evaluate many pose hypotheses reaching a hit rate of 97.8% corresponding to an error threshold of 15° at 55.8 *fps*. In the same context, in [10] a particle swarm optimization search results in remarkably small uncertainty when predicting head pose (around 2° standard deviation for all angles), but, similar to [2], they also resort to massive parallel resources coming from GPU.

Another group of methods for head pose estimation in 3D treats the task as a mesh registration problem. An output of a depth sensor is seen as a set of points or as a mesh. Weise et al. [14] present a method for transferring facial expressions from a user to an animated avatar. To accomplish the transfer they first create a person-specific model for a performer by manually marking correspondences between the model and the user. This operation is done automatically in [7], eliminating the requirement to do offline tuning of the system to a particular human performer. Methods like [3], [8], [13], [14] still rely on landmarks detection, such as eyes, nose, mouth and other features. When facial features are only partially visible in the scene due to an extreme head pose (e.g., just one eye, or a part of the nose), accurate correspondence estimation is no longer possible and the accuracy of the system decreases.

In order to handle large pose variations and therefore being able to process non-frontal views of a face, one needs a different method of initial correspondence estimation. Under these conditions, the fusion between two independent components seems an attractive and elegant solution. To the best of our knowledge there are no works in the literature that tackle the problem of real-time head pose estimation from such extreme views as we do.

III. From detection to tracking

To be able to determine a head pose from non-frontal views we fuse the results of two independent components every time instant. The first component is a head pose detector that analyses frames independently and provides an estimate for the head center and orientation simultaneously. At the first frame, these estimates are used to initialize a person-specific tracker. After the tracker is initialized the two components process each frame in parallel. The fusion of these two independent components allows us to improve the estimate of the head pose. These methods are further detailed in the following sections.

A. Head Pose Detector

The goal of the head pose detector is predicting the coordinates of the head center projected onto the camera plane (depth coordinate can be easily obtained by taking the pixel value at (x, y) in the depth image) and two of the 3DOF angles that describe the head orientation: yaw and tilt. Roll is omitted since for the given scenario one expects insignificant variations of this component.

Inspired by the work of [6], we address head pose estimation in 3D, using a patch majority voting scheme. There is one major aspect that differentiates our system from the one in [6] though. Instead of considering a random forest approach, we employ a cascade of specialized trees, each solving a simple classification/regression problem. Learning consists of generating depth patches from each frame, feature extraction and tree growing.

1) Patch data construction: We randomly sample each scene from the training set with a predefined number of squared patches. The patches are then selected based on the amount of depth information they contain and then saved along with the associated labels. The size of the patch was fixed to 100 pixels experimentally, by computing the correlation between considered features and the classification/regression labels.

2) Feature Extraction: Similarly to [6], we considered as features the simple differences between regions of pixels, computed on top of depth data. Each feature is obtained by subtracting the mean values of two rectangular regions located inside a given patch. Since we are looking at training individual trees instead of random ensembles, we need to ensure that our trees are strong enough by themselves to cast reliable predictions. Ideally, one would initially build the feature space for all training samples, making it available at the root node. However, this approach is limited by the available memory resources, especially when working with massive quantities of data. Instead, we generate a considerable amount of predefined pairs of pixel regions (one million in our experiments) and introduce a trace of randomness in the training stage, by choosing a feature subset on which the classification/regression measure will be optimized at each split. This way, we statistically ensure that all features will be *visited* at least once, while keeping memory requirements to practical values.

3) Learning the trees: We start by training a classification tree to differentiate between head patches and non-head ones. Growing the classification tree is driven by maximizing the information gain for each split. Next, we train regression trees only on patches coming from the head region, one for each regression label (i.e., offset on X, offset on Y, yaw and tilt). In this case, each split minimizes the weighted sum of standard deviations of the specific regression coordinate for the resulted statistical populations sent to children nodes. This way, moving to deeper levels inside the tree, the uncertainty associated to a particular prediction should continuously decrease. Training stops when a predefined maximum depth is reached, or there are fewer samples to split than a given threshold.

Leaf nodes store statistical information about the classification measure (e.g., the probability of a patch to belong to the head region) and Gaussian model parameters for regression case (mean and standard deviation).

At test stage, we focus on isolating the patches coming from the head area, as they are the only ones that hold information about the head orientation. We use two mean shift algorithms, one for clustering the votes around the head center and another for inferring the head angles.

To motivate our adopted scheme, we trained a cascade of 5 trees (CT) on the publicly available Biwi Kinect dataset [5]. We used for comparison the already trained forest (RF) along with the testing code provided by the same authors. In Table I we report the mean and the standard deviation of the angular error for all 15K samples from Biwi. We can see that, for yaw angle our proposed cascade is as competitive as the random forest model, while for tilt our method is not so accurate. However, CT requires less than half the number of comparisons when processing a patch. In addition, we reduce the processing time even further by feeding to the regression trees only the patches that belong, with a high confidence, to the head region. In the experimental section, we will show that our approach can also generalize well for unseen subjects.

TABLE I: Comparison between cascade of specialized trees (CT) and random forests (RF) on Biwi Kinect Dataset

	mean(yaw)	std(yaw)	mean(tilt)	std(tilt)
RF [5]	4.55	7.09	3.49	5.38
CT	4.72	7.68	5.26	9.12

B. Head Pose Tracker

The head pose detector described in Section III-A analyses each frame independently. In contrast, tracking allows us to represent head pose as a sequence of changes depending on the previous position and orientation. In our solution the initial position is determined by the head pose detector, while the tracker is required to refine these estimates and track changes.

Having initial estimates of head position and orientation we perform tracking by rigidly fitting a person-specific template, obtained in an offline template recording session for every individual. The term ”offline” here means that the template is required prior to the tracking stage. This is not a constraint for our application, because template creation is performed in real time, and the user has only to move their face from left to right passing through frontal position.

1) Person-Specific Template: A person-specific template represents a 3D surface learned for a particular individual. This surface is used for head pose tracking, as described in the next section. To create a person-specific template we use frontal recordings from the dataset described in Section IV-A, where each person is asked to rotate the head in horizontal plane from left to right. Each time instant t we receive a

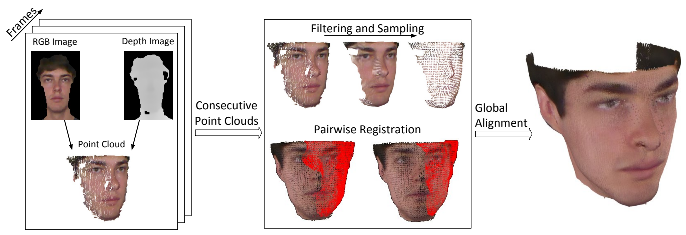


Fig. 1: Person-specific template creation pipeline. For each subject we take the first session from the dataset described in Section IV-A. Prior to creating the point cloud, we filter out noise by convolving the depth image with a Gaussian kernel. Voxel-grid algorithm on the smooth cloud is used to obtain a cloud with fewer points. Pairwise registration is performed on the consecutive clouds.

pair of depth and RGB images (D_t, I_t). Having the optical parameters of the capturing device we are able to restore the scene in the world coordinates. This scene represents a so called point cloud $P_t = \{p_i : i = 1, \dots, m\}$, where each point $p_i = \{x, y, z, r, g, b\}$ contains information about the location and the color taken from the corresponding (D_t, I_t) pair.

We treat the person-specific template creation as a registration problem, where we want to find a transformation $T_{i\to j}$ that transforms every point of a cloud P_i to a coordinate system associated with P_j, $i \neq j$. Having consecutive point clouds $(j = i + 1)$ we are able to find an accurate transformation, since the overlap of these clouds is large.

Given a set of K clouds, in order to find a set of pairwise transformation matrices $T = \{T_{i\to j} : j = i+1, i = 1, \dots, K-1\}$, one needs to have exact correspondences between clouds. To estimate these transformations we use the Iterative Closest Point (ICP) algorithm [1]. A generic ICP algorithm starts from a rough initial alignment of two objects, then it repeatedly determines correspondences, weighs them and estimates a transformation by minimizing a selected error metric. After a transformation estimate is determined and applied, a new set of correspondences is generated. The process stops when a difference between consecutive transformations is below a desired threshold or an iterations limit is reached. There are many variants of the ICP algorithm depending on a particular strategy used at each step [12]. A particular interest for us is the weighting step, where every correspondence receives a weight according to its importance (see Section III-B2).

Fig. 1 shows the pipeline to obtain a person-specific template. The sensor provides us with consecutive pairs of depth and color images (D_t, I_t). To filter out high frequency noise, we convolve the depth image with a Gaussian kernel before it is being used to create a point cloud. This makes the resulting point cloud smoother. Moreover, this convolution is able to fill holes that have smaller size than the Gaussian kernel. To make personalized template creation faster and more robust we use a voxel-grid approach, which imposes a 3D grid consisting of cubic cells with a predefined edge length. All points belonging to the same cell are averaged and, thus, a new point cloud is created. This subsampling not only allows for faster convergence, due to reduced number of points, but also helps removing ambiguity when determining corresponding pairs of points at the next step.

Consider two consecutive point clouds P_i and P_j, where $j = i+1$. Let k and l be the numbers of points in these clouds correspondingly. A corresponding point $p^j_m \in P_j$ for a point $p^i_n \in P_i$ is determined in the following way:

$$C_{i,j}(p^i_n) = \{p^j_m : m = \underset{h}{\operatorname{argmin}} \|p^i_n - p^j_h\|, \|p^i_n - p^j_h\| < \epsilon, h = 1, \dots, l\} \quad (1)$$

Then a set of all corresponding points for a cloud P_i in a cloud P_j is defined as:

$$C(P_i, P_j) = \{C_{i,j}(p^i_n), n = 1, \dots, k\}. \quad (2)$$

If a distance between points is greater than a predefined threshold ϵ, then this pair of points is not considered as a correspondence. During template creation stage, the exact value of this distance is used to assign a weight for each correspondence. In Section III-B2 we use a different way of weighting correspondences.

Having the correspondences we seek for a transformation that minimizes the distances between corresponding points. To do that one needs two select a proper error metrics. We used a point-to-plane metrics since it allows minimization algorithm to converge faster [11].

We associate the world coordinates with the coordinate system of the last cloud in a sequence. Therefore, to obtain a global registration transformation for a cloud P_i in the world coordinates $T^w_i = T_{i\to K}$ having consecutive transformations one needs to propagate required pairwise transformations for each cloud except the last one in the following way:

$$T^w_i = \prod_{j=i}^{K-1} T_{j\to j+1}$$

where $T_{j\to j+1}$ is a consecutive pairwise transformation. An example of a person-specific template is given in Fig. 1.

2) History-Based Weighted Tracking: Inspired by [14], we study whether template-based tracking of the head pose and orientation can be applied to extreme poses. Initial head position and orientation T_0 is provided by the detector component described in Section III-A. A person-specific template is initialized with this orientation and translation and the pose of the template is refined by registering the template to a new scene.

To allow for real time tracking we introduce a history-based weighted scheme, that significantly increases speed. The motivation behind this scheme comes from the observation that when fitting a template to extreme head poses not all correspondences should be weighted equally. Fig. 2 shows an example of template registration to an extreme head pose. Since not all points of the face are present in the scene, only a small set of template points should be used for registration.

When a template $\mathcal{T}$ is registered with a scene at a time $t-1$, the most valuable points of the template $\mathcal{T}^*_{t-1}$ are determined as correspondences between a template and a scene in the following way:

$$\mathcal{T}^*_{t-1} = C(\mathcal{T}, P_{t-1}).$$

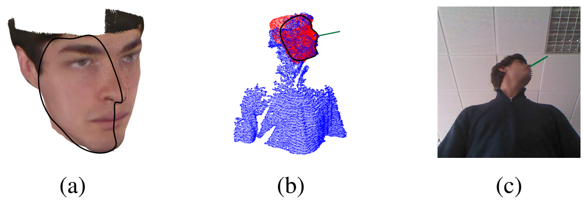


Fig. 2: A person-specific template (a) is registered to an extreme view of a person (b, c). Not all correspondences are equal in registering a template in this situation. The highlighted part of the template is more important than the rest.

Having a history of the most valuable points up to iteration $t-k$ and assuming that the two consecutive point clouds do not have significant differences, we highlight the point p_i of a template at a time t according to a weight ω_i in the following way:

$$\omega_i = \alpha + \beta \cdot k_i, \tag{3}$$

where α is a weight that every point receives, β is a highlighting factor for a point p_i, and k_i is number of times p_i appears in the history $\{\mathcal{T}^*_{t-1}, \ldots, \mathcal{T}^*_{t-k}\}$. To use the equation (3) one needs to decide upon values of α, β and k_{max} which is a maximum size of the history. Our experiments to determine these values are given in Section IV-C. Some examples of important points for difficult head poses are given in Fig. 6.

C. Detector-Tracker Fusion

At this stage, for every frame we obtain two predictions for head orientation: one from the head pose detector and one from the person-specific template tracker. Having these two measurements one needs to decide which measurement to use or come up with a fusion strategy. We have chosen the later, since we are interested in combining both measurements. One simple way to do this is to average both predictions. However, by setting equal weights to both measurements we do not consider that the variances of two predictions can be different. Therefore, a measurement with a smaller variance will be affected by the measurement with the bigger variance. One way to utilize the variances of both measurements is to use a Kalman Filter framework.

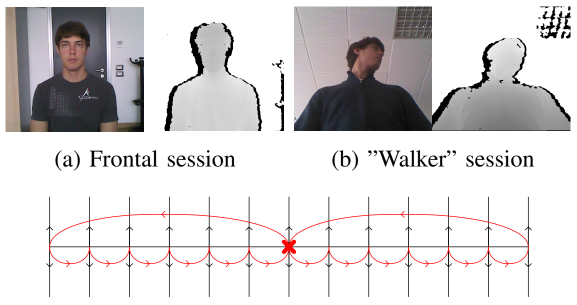
To use this framework we consider our predictions as two measurements z_1 and z_2 performed by two independent sensors having observation noise matrices $\mathbf{\Sigma_1}$ and $\mathbf{\Sigma_2}$. To be able to use a group sensor method one needs to specify an observation model for the two sensor case:

$$\mathbf{H} = \left[\mathbf{H}_1^T, \mathbf{H}_2^T\right]^T, \tag{4}$$

where $\mathbf{H}_1$ and $\mathbf{H}_2$ are the observations models for each sensor, and provide the observation noise covariance which is a block-diagonal matrix, where each block is equal to the observation noise matrix for a particular sensor:

$$\mathbf{R} = \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_2 \end{bmatrix}. \tag{5}$$

The rest of the Kalman Filter framework remains unchanged. Observation noise for each sensor ($\mathbf{\Sigma_1}$, $\mathbf{\Sigma_2}$) is computed prior to fusion with the use of the data that is then removed from testing set. The obtained result are discussed in the Section IV-C.



(a) Frontal session (b) ”Walker” session

(c) Head movement pattern

Fig. 3: Colour and depth images for the first recording session (a). Images from the second session recorded using the walker (b). Head movement pattern (c). The red cross in the middle indicates start and end position of the head

IV. Experiments

In this section we introduce the newly acquired Dali3DHP dataset and present the experimental output of our system.

A. Dali3DHP Dataset

Since we are treating HPE under extreme pose variations, we need a database to comply with the task at hand. One solution would have been to consider existing synthetic 3D datasets and apply geometric transformations in order to get the desired view points. However, the literature [6] shows that moving from high resolution scans to noisy real-life Kinect data may result in performance drops. Therefore, we proceeded at recording a dataset that contains various head poses, captured from the walker's view point. This dataset consists of two sessions of range and RGB images of 33 individuals. During the first session each person is asked to show their face frontally and then to perform a full head movement to the left, right, up and down (see Fig. 3a).

The second session was captured with the use of the walker (see Fig. 3b). A person is standing holding the handles of the walker while following with the head a special movement pattern marked on the wall in front (see Fig. 3c). Each subject starts looking ahead (red cross in Fig. 3c) with all head angles approximately equal to zero. Next, the subject moves the head to the left keeping the tilt angle equal to zero and does up-down movement for each vertical line. Transitions between the vertical lines are performed with zero tilt angle. After the subject has performed all the necessary movements, he/she returns to the starting point with all angles equal to zero.

Ground-truth labels were recorded via a Shimmer sensor[1], fixed on the head. The sensor transfers accelerometer and gyroscope readings at 100Hz frequency via a Bluetooth protocol to a computer. We used a Kalman filter based algorithm [4] to transform sensor readings into head orientation angles. Since

[1]http://www.shimmersensing.com/

no wires are required, the subject is not constrained and, thus, can perform movements in a natural way. Our experiments show, that the Shimmer sensor is not able to accurately measure roll angle. Therefore, head movement pattern was designed so that the actual movements are performed with roll angle close to zero. This is acceptable for many practical scenarios that aim at determining the gaze direction of a person. For the head pose detector, we manually annotated the center of the head in each frame using specialized assisted software.

Our database contains more than 60K depth/color pairs coming from 33 individuals covering the following head angles: tilt $[-65.76°, 52.60°]$, roll $[-29.85°, 27.09°]$, and yaw $[-89.29°, 75.57°]$. The dataset will be made available to the research community.

B. Testing the trees

In order to assess the performance of our cascade approach, we performed a leave-one-out cross-validation analysis on the entire dataset. For each iteration, we built a cascade of five trees, as described in III-A. We pushed the maximum depth of the trees responsible for the head angles to 18, in order to better fit the distributions of these label dimensions. Table II reveals the mean and the standard deviation of the error for yaw and tilt, computed for all 60K samples from Dali3DHP dataset using three stride values for exploring the scene with patches. As in [6], the stride parameter controls the trade-off between speed and accuracy. The detector experiences more difficulties for the tilt angle, especially at extreme values. This behavior can be explained by the rather unbalanced tilt distribution over the range of values.

TABLE II: Cross-validation results obtained on Dali3DHP Dataset

	mean(yaw)	std(yaw)	mean(tilt)	std(tilt)
Stride 5	4.73	6.89	7.69	9.81
Stride 10	5.37	7.69	7.84	10.05
Stride 20	6.28	10.14	8.20	10.62

C. Tracking and Fusion

In this section we analyze the results of Full Template Tracker, Weighted Template Tracker, Detector with Stride 5, Fusion by Averaging and Fusion by Kalman Filter of the detector with weighted template tracker. To asses the performance of the described approaches we use the Dali3DHP dataset. To start the tracking we first determine initial head position and orientation by using the detector described in Section III-A. A person-specific template, then, is initialized at the predicted location and tracking starts. Thereafter both components work in parallel, with their outputs being fused by a Kalman Filter.

Prior the comparison, we need to decide upon the values of the weighting parameters α, β and k_{max}. We have not found that the accuracy depends significantly on these parameters, whereas the processing speed does (see Fig. 4). To asses the speed we measure the average number of ICP iterations required to converge, since it is a hardware-independent metric. The average number of ICP iterations is positively correlated with α, which is the default weight each point receives, while

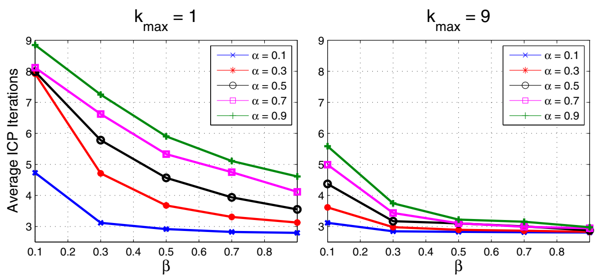


Fig. 4: Dependence of number of ICP iterations required to converge on the weighting parameters α, β and k_{max}.

the correlation with β and k_{max}, on the contrary, is negative. These results suggest that when the relative weights of the important points grow the speed increases. The intuition behind this is that by weighting we increase the gradient into the promising direction and the optimization algorithm makes bigger steps, and hence converges faster.

Out of all possible combinations of the weighting parameters we selected $\alpha = 0.1$, $\beta = 0.5$ and $k_{max} = 9$. For the remaining tables in this section we use this combination of the weighting parameters.

TABLE III: Comparison between Weighted and Full Template Tracking in computational time

	Average ICP Iterations	Frames per second
Full Template	14.64	10.05
Weighted Template	**3.16**	**38.87**

Table III compares Weighted Template Tracker with Full Template Tracker in average ICP iterations. These results show that the Weighted Template Tracker requires 4.63 times less iterations than the Full Template Tracker, being able to process more than 38 frames per second. Table IV compares the means and the standard deviations of the error of all described approaches.

TABLE IV: Results obtained on Dali3DHP Dataset

	mean(yaw)	std(yaw)	mean(tilt)	std(tilt)
Full Template	4.06	5.89	8.21	11.45
Weighted Template	3.93	5.23	8.21	11.31
Detector - Stride 5	4.73	6.89	7.69	9.81
Fusion by Averaging	3.51	4.92	6.09	7.77
Kalman Filter Fusion	**3.18**	**4.28**	**5.91**	**7.46**

Although Weighted Template Tracker shows on average a better accuracy than Full Template Tracker, this does not reflect the main advantage of using the weights. Processing speed is a key issue we gain in this case. Fusion by Kalman Filter allows us to decrease both the means of the error and their standard deviations. To prove that Fusion by Kalman Filter outperforms Fusion by Averaging and the difference is statistically significant we report the results of left-tailed t-test for means and left-tailed F-test for the standard deviations against two sets of hypotheses. The p-values are given in Table V, where subscript k denotes Fusion by Kalman Filter and a denotes Fusion by Averaging. Since p-values are very small we are able to reject H_0 in favour of H_1 proving that the observed differences in the results are statistically significant.

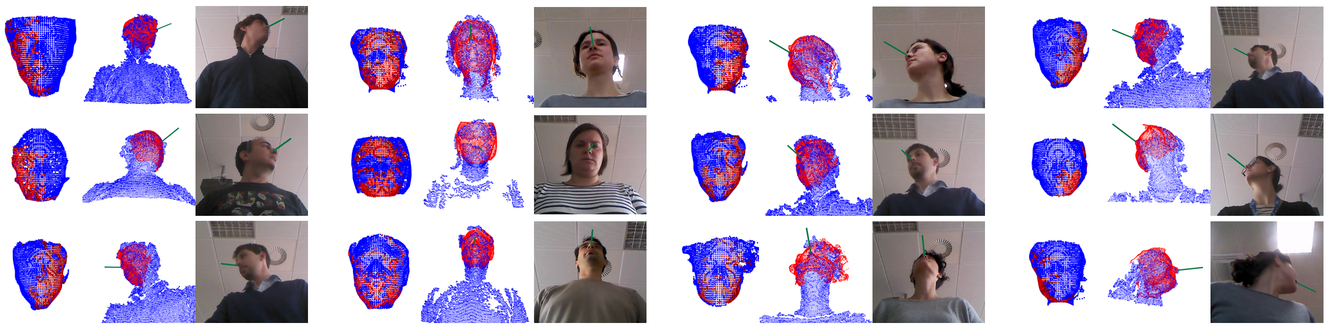


Fig. 6: Some examples of extreme head poses correctly recognized by our approach. For every subject three images are given. The left one represents a template with the most important points marked in red. The image in the middle shows the template fitted to the point cloud. The right image shows the view from the walker. Note that for some subjects the face is almost completely hidden.

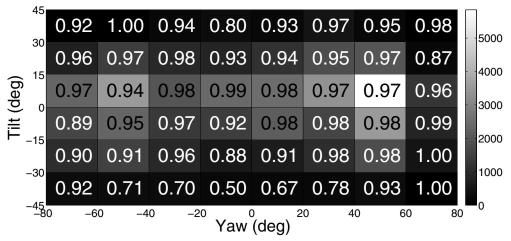


Fig. 5: Fusion success rate for error smaller than 15° on all Dali3DHP Dataset. Note that the tilt is measured with respect to the horizontal plane, not with the camera coordinate system. Tilt equal to +40° represents a head pose with almost completely unseen face.

TABLE V: A set of hypothesis with corresponding p-values for yaw and tilt angles

H_0	H_1	yaw	tilt
$\mu_k = \mu_a$	$\mu_k < \mu_a$	1.73e-72	4.54e-11
$\sigma_k = \sigma_a$	$\sigma_k < \sigma_a$	3.13e-244	8.77e-23

Fig. 5 shows the success rate of the Kalman fusion on the Dali3DHP dataset. The dataset was split into 20° × 15° intervals for yaw and tilt respectively. Success is defined for errors smaller than 15° both for yaw and tilt. The color map encodes the number of test samples for each cluster. Finally, Fig. 6 gives some examples with correctly recognized extreme head poses.

V. Conclusions

In this paper we propose a fusion approach to address real-time head pose estimation under extreme head orientations. By combining a frame independent decision tree based estimator with a personalized template tracker, we constructed a reliable real-time system able to recover easily in case it loses track. In addition, we recorded a 3D head pose database containing more than 60K pairs of depth/color frames along with the associated orientation labels, which will be released to the community.

References

[1] P. Besl and N. D. McKay. A method for registration of 3-D shapes. *PAMI*, 14(2):239–256, 1992.

[2] M. D. Breitenstein, D. Kuettel, T. Weise, L. Van Gool, and H. Pfister. Real-time face pose estimation from single range images. In *CVPR*, pages 1–8, 2008.

[3] K. I. Chang, W. Bowyer, and P. J. Flynn. Multiple nose region matching for 3d face recognition under varying facial expression. *PAMI*, 28(10):1695–1700, 2006.

[4] A. Colombo, D. Fontanelli, D. Macii, and L. Palopoli. A wearable embedded inertial platform with wireless connectivity for indoor position tracking. In *I2MTC*, pages 1–6, 2011.

[5] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Gool. Random Forests for Real Time 3D Face Analysis. *IJCV*, 101(3):437–458, 2012.

[6] G. Fanelli, J. Gall, and L. Van Gool. Real Time Head Pose Estimation with Random Regression Forests. In *CVPR*, pages 617–624, 2011.

[7] H. Li, J. Yu, Y. Ye, and C. Bregler. Realtime facial animation with on-the-fly correctives. *ACM TOG*, 32(4):42, 2013.

[8] X. Lu and A. K. Jain. Automatic feature extraction for multiview 3d face recognition. In *AFGR*, pages 585–590, 2006.

[9] E. Murphy-Chutorian and M. M. Trivedi. Head Pose Estimation in Computer Vision: A Survey. *PAMI*, 31(4):607–626, 2009.

[10] P. Padeleris, X. Zabulis, and A. A. Argyros. Head pose estimation on depth data based on particle swarm optimization. In *CVPRW*, pages 42–49, 2012.

[11] K. Pulli. Multiview registration for large data sets. *3DDIM*, pages 160–168, 1999.

[12] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3DDIM*, pages 145–152, 2001.

[13] Y. Sun and L. Yin. Automatic pose estimation of 3d facial models. In *ICPR*, pages 1–4, 2008.

[14] T. Weise, S. Bouaziz, H. Li, and M. Pauly. Realtime performance-based facial animation. *ACM TOG*, 30(4):77–85, 2011.