This CVPR2015 paper is the Open Access version, provided by the Computer Vision Foundation.
The authoritative version of this paper is available in IEEE Xplore.

Real-time 3D Head Pose and Facial Landmark Estimation from Depth Images Using Triangular Surface Patch Features

Chavdar Papazov
chavdar.papazov@gmail.com

Tim K. Marks
tmarks@merl.com

Michael Jones
mjones@merl.com

Mitsubishi Electric Research Laboratories (MERL)
201 Broadway, Cambridge, MA 02139

Abstract

We present a real-time system for 3D head pose estimation and facial landmark localization using a commodity depth sensor. We introduce a novel triangular surface patch (TSP) descriptor, which encodes the shape of the 3D surface of the face within a triangular area. The proposed descriptor is viewpoint invariant, and it is robust to noise and to variations in the data resolution. Using a fast nearest neighbor lookup, TSP descriptors from an input depth map are matched to the most similar ones that were computed from synthetic head models in a training phase. The matched triangular surface patches in the training set are used to compute estimates of the 3D head pose and facial landmark positions in the input depth map. By sampling many TSP descriptors, many votes for pose and landmark positions are generated which together yield robust final estimates. We evaluate our approach on the publicly available Biwi Kinect Head Pose Database to compare it against state-of-the-art methods. Our results show a significant improvement in the accuracy of both pose and landmark location estimates while maintaining real-time speed.

1. Introduction

Head pose estimation and facial landmark localization are challenging computer vision problems with important applications in various fields such as human-computer interaction [22], face recognition [25, 1], and character animation [5], just to name a few. Recent advances in real-time 3D geometry acquisition have led to a growing interest in methods that operate on 3D data. Furthermore, in contrast to 2D image-based approaches, the 3D methods do not have to deal with changes in illumination and viewpoint. This makes them, as shown in [22], more accurate and robust.

In this paper, we present an algorithm for 3D head pose estimation and facial landmark localization using a commodity depth sensor such as Microsoft's Kinect. The proposed method consists of an offline training and an online testing phase. Both phases are based on a new triangular surface patch (TSP) descriptor. Our method is robust to noise, is rotation and translation invariant, and runs on a frame-by-frame basis without the need for initialization. It can process 10 to 25 frames per second, depending on the desired accuracy, on a single CPU core, without using parallel hardware such as GPUs.

We test the proposed algorithm using the data from the Biwi Kinect Head Pose Database [9]. However, our method is trained on synthetically generated 3D head models [21], rather than on depth maps from a Kinect sensor. As a result, our method should be easily adaptable to other types of 3D sensors, sensor locations, and resolutions, without the need to re-train it on new sensor-specific or situation-specific data. The test results show that the proposed method achieves better accuracy than state-of-the-art approaches [2, 9], while maintaining real-time processing.

2. Related Work

Previous approaches to head pose estimation include both 2D image-based and 3D depth-based approaches. We will not discuss 2D image-based methods here except to note that previous work has found them to be less accurate than 3D depth-based algorithms [22].

One of the earliest methods for real-time head pose estimation from depth data was the one proposed by Seeman et al. [22]. They used a stereo camera pair to compute depth and then detected the head using skin color. A 3-layer neural network estimated the pose angles given the scaled depth map of the head region. Breitenstein et al. [7] presented a more accurate and faster system for head pose estimation that took advantage of less noisy depth map acquisition and the speed of GPUs. First, candidate 3D nose positions were detected in the high quality depth map. Then, the GPU was used to quickly find the best match between the input depth map and a number of stored depth maps from an average head model located at each candidate nose position. Their

system estimated poses at almost 56 frames per second (not counting the time to acquire the depth maps).

Following this work, Fanelli and colleagues published several related papers [10, 11, 9] about head pose estimation and facial landmark localization from depth maps. In [10], high quality 3D head scans were used as input. Their approach was based on random regression forests and learned mappings from a patch of the depth image to pose angles or facial landmark locations. The regression forests were trained on example depth patches rendered from synthetic head models. The results in [10] were more accurate than those in [7], and depth maps were processed at about 25 frames per second (not including acquisition time) on a standard CPU. In particular, real-time performance was achieved with no GPU. The follow-up paper [11] used a Kinect sensor, which provides significantly noisier data than the high quality scans used in the previous work. Similar to [10], the approach is based on a regression forest with the added feature of classifying whether or not each depth patch belonged to the face. This allowed the system to work on scenes in which torsos and hands were in view in addition to the face. The training was performed on patches from actual Kinect depth maps and not from synthetic data. The resulting systems processed from 11 to 130 frames per second on a current CPU, depending on the trade-off between speed and accuracy. However, it was not as accurate as the previous system due to the noisier input depth maps.

The Constrained Local Model framework [8] was extended by Baltrusaitis et al. [2] to use depth as well as intensity information to fit a 3D shape model to RGB-D data. This was combined with a rigid head pose tracker based on a Generalized Adaptive View-based Appearance Model [17] and led to more accurate pose angle estimates than Fanelli et al. [9]. Another relevant paper, by Padeleris et al. [20], estimated the pose of an input Kinect sensor depth map by finding the 3D rotation of a template that best matched the input. Excellent results were reported, however, the method requires an initial person-specific template in a known pose. This makes it impractical for many applications.

In addition to the above cited methods, a number of systems have been proposed in the computer graphics literature [5, 27, 28, 6, 31] for tracking heads in RGB-D video streams to create high resolution face models or to animate avatars. However, such tracking systems need some sort of initialization. Furthermore, papers in this line of research do not include any evaluation of head pose estimation accuracy.

Other tracking-based approaches to 3D head pose estimation include Bleiweiss and Werman [4] and Smolyanskiy et al. [24]. These approaches suffer from the problem of drift. Also, neither paper reports pose error values.

Our approach is more similar to Fanelli et al. [11] than to other previous work. Like in [11], a Kinect depth sensor provides the input data which we process in real-time without the need for a GPU. However, [11] uses simple rectangular difference features in depth map space, which are viewpoint-dependent and require the training data to contain viewpoints similar to those present in the test data. In contrast, we propose more elaborate triangular surface patch features which are computed from the 3D point cloud reconstructed from a depth map. As a result, our features are viewpoint-independent and robust to changes in pose, scale (distance to camera), and resolution. Furthermore, we train our system on synthetic head models. These are much easier to obtain in large numbers than real-world depth maps of a variety of heads (such as the training data used in [11]). Our experimental results demonstrate better accuracy than previous work [11, 9, 2], while performing at a comparable speed using a single core of a standard CPU.

3. Overview of our Technique

Our method consists of two phases. The first is an offline training phase, which is executed on a set of synthetically generated 3D head models. It only needs to be performed once, since the model representation learned in this phase does not depend in any way on the test data or test scenario. The second phase is the online test phase which consists of real-time pose estimation and facial landmark localization. It is performed on real-world depth images and makes use of the model representation learned in the training phase.

Both phases are based on our new TSP feature, which we now introduce before giving more detail on the two phases.

3.1. Triangular Surface Patches (TSPs)

Definition of a TSP Let $\mathbf{S} \subset \mathbb{R}^3$ be a point cloud representing a human face. In the training phase, the points $\mathbf{S}$ are the vertices of a synthetic 3D head model, while in the testing phase, each point in $\mathbf{S}$ is the 3D location of a pixel from a depth image. Assume we are given an equilateral triangle $\mathbf{T} = (\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2)$ with vertices $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2 \in \mathbb{R}^3$ and side length l. Furthermore, assume that the triangle's vertices lie on the surface of the face, meaning that the distance between each $\mathbf{q}_i$ and its closest point from $\mathbf{S}$ is smaller than a given threshold d. (In our implementation, we use $d = 3$ mm.) In this case, we say that the triangle $\mathbf{T}$ is sampled from $\mathbf{S}$. Details on how to perform the sampling will be given in Section 4.1. Figure 1(a) provides an example.

Our triangular surface patch $\mathbf{P}$ consists of those points from $\mathbf{S}$ that are above or below the sampled equilateral triangle $\mathbf{T}$. In other words, $\mathbf{P} \subset \mathbf{S}$ is the set of points from $\mathbf{S}$ contained within an infinitely tall triangular prism with base $\mathbf{T}$. Figure 1(b) shows a TSP.

Having a prism of infinite extent could lead to TSPs containing not only points close to the base triangle but also points belonging to unrelated parts of the face, as shown in Figure 1(c). To avoid this, we only consider points to be in the TSP if they are both in the interior of the prism

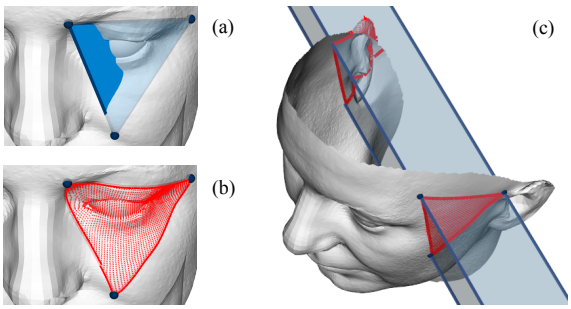

Figure 1. (a) An equilateral base triangle, $\mathbf{T}$, sampled from the vertices, $\mathbf{S}$, of a 3D head model. The light blue part of the base triangle is occluded by the face's surface, while the dark blue part occludes the face. (b) The corresponding triangular surface patch (TSP), $\mathbf{P}$ (shown in red), consists of the points in $\mathbf{S}$ that lie above or below the base triangle. (c) Using an infinite prism could lead to undesired points (in this example, the right ear) being incorrectly included in the TSP.

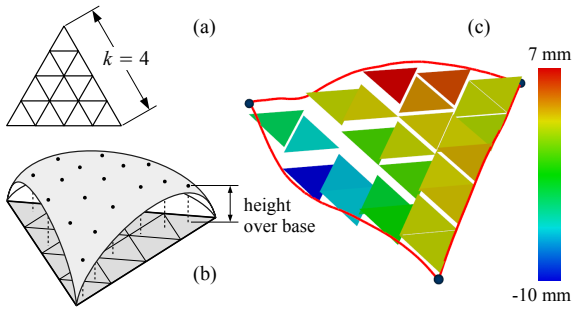

Figure 2. (a) Subdivision of the base triangle into $k = 4$ sub-triangles per side results in a total of k^2 sub-triangles. (b) Each TSP point (black dot) belongs to the sub-triangle in which it would lie if it were projected perpendicularly onto the base triangle. (c) Visualization of the descriptor, $\mathbf{v}$, for the TSP shown in Figure 1(b), using $k = 5$ sub-triangles per side. Each sub-triangle is displaced above or below the base triangle and colored according to the mean height of the points it contains.

and inside of the circumsphere centered at the centroid of the base triangle and passing through its vertices. Besides solving the problem illustrated in Figure 1(c), this also improves computational performance, since we use a fast radius search from the highly optimized FLANN library [18, 19] to retrieve the points inside the circumsphere of the base triangle. We then need to perform a prism containment test only on the retrieved points.

For general surface geometries, the circumsphere of a base triangle might cut off surface parts that rise too much above or below the base triangle. In practice, however, this does not happen, because human faces have limited height variation and we use a triangle side length that is large enough (see Section 4.3).

TSP descriptor Given a base triangle $\mathbf{T}$ and the corresponding triangular surface patch $\mathbf{P}$, we compute a simple and robust geometric descriptor $\mathbf{v}$ as follows. The base triangle is partitioned into k^2 equilateral sub-triangles, as shown in Figure 2(a). If it were projected perpendicularly onto the base triangle, each point in $\mathbf{P}$ would lie within one of the sub-triangles. We say that the point *belongs* to the sub-triangle, or equivalently that the sub-triangle *contains* the point. Each point in $\mathbf{P}$ has some (positive or negative) height over the base triangle, as illustrated in Figure 2(b). The descriptor of each sub-triangle is the mean height of the points it contains. The descriptor of the whole TSP is a vector $\mathbf{v} \in \mathbb{R}^{k^2}$ that is simply a concatenation of the descriptors of all of its sub-triangles. The TSP descriptor can be thought of as a piecewise-constant triangular surface patch, defined in the coordinate system of the base triangle, that approximates the TSP. This is illustrated in Figure 2(c).

Using the average height of all points within a sub-triangle makes the proposed descriptor robust to noise and to variations in the data resolution. This is confirmed in the experimental part of the paper, in which synthetic, noise-free high-resolution models are used for training, but then the detection is successfully performed on real-world, noisy, lower-resolution data from a commodity RGB-D sensor.

A further challenge of real-world depth images is the presence of holes due to scan device limitations and self-occlusions of the imaged object. This can lead to empty sub-triangles and thus to TSP descriptors with undefined components. To handle this, we fill in the missing data by propagating the height information of the full sub-triangles across the empty ones, using a simple iterative procedure. In each iteration, the empty sub-triangles are populated by assigning them the average height of their full neighboring sub-triangles. Sub-triangles that have no populated neighbors are left unchanged in the current iteration. This process is repeated until all sub-triangles are populated. Finally, a fixed number, s, of smoothing iterations (simple averaging of the values in the neighboring sub-triangles) are applied only to the newly populated sub-triangles (i.e., without changing the original full ones). In our implementation, we use $s = 3$. This leads to a smooth distribution of height information across any holes in the original data.

The base triangle side length, l, and the number k of sub-triangles per side are important parameters of the TSP descriptor. To select an optimal combination of parameter values, we collected and labeled a small data set of Kinect depth image sequences and employed an automatic parameter selection procedure, presented in Section 4.3.

A number of 3D geometric descriptors have been proposed in the literature. A short list includes Spin Images (SI) [14], 3D shape context (SC) [12], SHOT [26], and MeshHOG [30, 29]. All these descriptors are defined with respect to a local coordinate frame and rely heavily on surface normals, which can be inaccurate for noisy

data. In contrast, our triangular base is defined using 3 well-separated surface points, making it stable and robust to noise. We use normals only for rough initialization of sampling (Section 4.1). Furthermore, in contrast to the other descriptors, ours enables simple, efficient, and robust hole filling, as described above.

In practice, point cloud data exhibit variations in sampling density that can be both global (e.g., due to changes in object distance or sensor resolution) and local (e.g., due to viewpoint changes). Unlike SI, SC, and SHOT, our descriptor is not based on histograms. In ours, a bin stores a mean height rather than a point count. Since mean height is not sensitive to the number of points in a bin, our descriptor is robust to both local and global density variations. In contrast, the other descriptors need complicated normalization procedures to try to compensate for local density changes.

Unlike our descriptor, each SI does not uniquely define a 6-DOF pose for the point described. Thus, a single SI descriptor cannot vote for face pose nor landmark locations, which our voting system needs. MeshHOG is more expensive to compute, making it less suitable for real-time processing. [30] reports a timing of "under 1 s" to compute 706 features on a 2.4 GHz CPU. On the same CPU, our method uses only 122 ms for 706 features.

Furthermore, our descriptor is low-dimensional, which reduces memory consumption and speeds up nearest-neighbor search. With optimal parameter settings (Section 4.3), its dimensionality is just 25. This compares favorably to the dimensionality of SHOT: 32, MeshHOG: 96, SI (uncompressed, as used nowadays): 200, and SC: 1980.

3.2. Training

The training is performed on high-resolution meshes, which we generated using the Basel Face Model (BFM) [21]. The BFM is a publicly available PCA-based 3D morphable model that was created based on high-resolution 3D scans of 200 people. By sampling PCA coefficient values from a normal distribution, the BFM can be used to generate any number of random faces in the form of surface meshes (see Figure 3(a)). Moreover, semantically corresponding mesh vertices have the same index in all meshes. For example, the vertex at the tip of the nose has the same index number in all of the training models. Consequently, the facial landmarks need to be manually annotated only once on a single BFM face, in order to know the 3D landmark positions on all models.

From each mesh in the training set, we randomly sample n equilateral base triangles $\mathbf{T}_1, \ldots, \mathbf{T}_n$. Details on the sampling procedure will be given in Section 4.1. In our experiments, we use $n = 10,000$, which densely covers the mesh in overlapping base triangles. Next, we determine the TSP corresponding to each $\mathbf{T}_i$ and compute its descriptor $\mathbf{v}_i \in \mathbb{R}^{k^2}$. Furthermore, we associate with each base triangle $\mathbf{T}_i$ the vectors that connect $\mathbf{T}_i$'s centroid to certain points of interest, which are the model centroid and the facial landmarks of that model (e.g., top of the nose, tip of the nose, and eye centers). These vectors are used in the online testing phase (see Section 3.3) to estimate the head pose and to localize the facial landmarks.

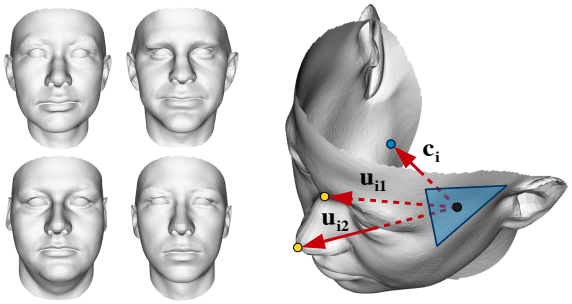

Figure 3. (a) A few of the synthetic 3D head models in our training set. (b) A base triangle $\mathbf{T}_i$ (shown in blue) sampled from a training head model, along with the vectors $\mathbf{c}_i$, $\mathbf{u}_{i1}$, and $\mathbf{u}_{i2}$, which originate at the centroid of $\mathbf{T}_i$ and point to the head model's centroid (blue dot) and two facial landmarks (yellow dots), respectively.

Thus, for each training model, we generate and save n samples $\mathcal{T}_1, \ldots, \mathcal{T}_n$, each one consisting of a base triangle $\mathbf{T}_i$ along with its associated data. More precisely,

$$\mathcal{T}_i = \{\mathbf{T}_i, \mathbf{v}_i, \mathbf{c}_i, \mathbf{u}_{i1}, \ldots, \mathbf{u}_{iq}\}. \quad (1)$$

Here, $\mathbf{v}_i$ is the TSP descriptor, $\mathbf{c}_i$ is the vector from the centroid of $\mathbf{T}_i$ to the model centroid, and $\mathbf{u}_{ij}$ is the vector from the centroid of $\mathbf{T}_i$ to the position of the jth facial landmark (see Figure 3(b)).

We store all samples from all of the training face models in a single library. It is organized in a way that allows rapid retrieval of samples that are most similar to a given query sample. Similarity is measured by the Euclidean distance between the TSP descriptors of the samples. To obtain nearest neighbors of TSP descriptors efficiently, we use the FLANN library [18, 19] to store and retrieve them. The TSP descriptors, $\mathbf{v}_i$, are saved as row vectors in a large matrix, and the other components of $\mathcal{T}_i$ are stored in corresponding order in an array. Given a query descriptor, FLANN operates on the matrix and provides the row indices of the TSP descriptors that are (approximately) most similar to the provided query descriptor. Using these, we then retrieve the corresponding base triangle and its associated facial landmark information from the array.

Because the centroid $\mathbf{c}_i$ of the face model and the facial landmark vectors $\mathbf{u}_{i1}, \ldots, \mathbf{u}_{iq}$ are all defined relative to the base triangle $\mathbf{T}_i$, our model representation can be used to estimate pose and facial landmark positions for heads in arbitrary poses and, in particular, at arbitrary distances from the depth sensor. Furthermore, it does not depend on the

sensor used in the testing phase. This is not the case for the method of Fanelli et al. [9]. For their method to succeed on a test depth map representing a head in a particular orientation and at a particular distance from the sensor, their training data must have been sampled from heads in similar poses. Moreover, the training data must be generated for a particular depth sensor. Thus, in contrast to our approach, the one presented in [9] cannot handle arbitrary head positions and orientations and uses sensor specific training data.

3.3. Online Testing

The online testing phase is performed on a 3D point cloud $\mathbf{S}$ that has been reconstructed from a depth map, which in our experiments is provided by a Kinect sensor. The head orientation and the locations of facial landmarks are estimated as follows.

Voting Consider an equilateral base triangle $\mathbf{T}'$ that has been sampled from $\mathbf{S}$. We compute the corresponding TSP and its descriptor $\mathbf{v}'$, as described in Section 3.1. Next, $\mathbf{v}'$ is used as a key to retrieve the most similar descriptor, $\mathbf{v}_i$, from our training library. Recall from (1) that $\mathbf{v}_i$ is associated with the library sample $\mathcal{T}_i$, which also contains a base triangle $\mathbf{T}_i$ as well as $\mathbf{c}_i$, the face centroid relative to $\mathbf{T}_i$.

We need to transform $\mathbf{c}_i$ into the coordinate system of $\mathbf{S}$. This is done by the rigid transformation whose translation is $\boldsymbol{\mu}'$, the centroid of $\mathbf{T}'$, and whose rotation matrix R is found as follows. Translate $\mathbf{T}_i$ and $\mathbf{T}'$ to respectively obtain $\widetilde{\mathbf{T}}_i$ and $\widetilde{\mathbf{T}}'$ whose centroids are at the origin, then find R that satisfies $R\widetilde{\mathbf{T}}_i = \widetilde{\mathbf{T}}'$. Thus, based on its match to the test triangle $\mathbf{T}'$, the library sample $\mathcal{T}_i$ votes for:

- head orientation R, and
- head centroid location $\boldsymbol{\mu}' + R\mathbf{c}_i$.

Since we are dealing with real-world data, the input point cloud is corrupted by noise and may contain non-face objects. Furthermore, the faces to be detected are not present in our training library. To make our method robust to these types of variations, we sample m query base triangles and retrieve for each one of them the h training samples whose descriptors are most similar to the query triangle's descriptor. This results in a total of mh rotation matrix votes for the head orientation, and mh positional votes for the centroid location. Note that these votes live in two different spaces. The rotational votes lie in the group of rotation matrices SO(3), and the positional votes lie in $\mathbb{R}^3$. Before the facial landmark locations are estimated, we eliminate inconsistent votes using a simple but effective filtering strategy.

Vote Filtering Recall from (1) that in the training phase, the facial landmark locations are stored (like the model centroids) as vectors relative to the sampled base triangle. Thus, voting for facial landmark locations can be performed in the same way that voting for the model centroid was performed. This leads to multiple voting spaces: SO(3) for the head orientation, and a separate voting space $\mathbb{R}^3$ for the head centroid and each of the landmark locations. Finally, the cluster center in each voting space can be detected independently. This is similar to the voting strategy employed in [9]. However, we found that this sometimes leads to incorrect results. Figure 4 illustrates the voting process in the centroid (b) and orientation (c) voting spaces. The blue dots in the centroid space represent the cluster that would be (incorrectly) detected if clustering were done independently in the centroid space.

To account for this, we filter out inconsistent votes by performing *joint* clustering in the rotation and centroid voting spaces. The red dots in Figure 4(b) and (c) represent the clusters correctly detected using this procedure. The reason why we don't cluster facial landmark votes is the following. Tests with independent clustering in each space showed that the rotation estimates are more stable than those of the facial landmarks, meaning that the landmark positions were more often wrongly estimated than the head orientation. To get the missing translational degrees of freedom of the head, we use the centroid votes. Note that the error of a facial landmark estimate increases with increasing distance between the base triangle and the landmark position. Since the model centroid roughly minimizes the average distance to all base triangles, it is the most appropriate among all positional votes.

Our joint clustering works as follows. Recall that each library sample $\mathcal{T}_i$ votes for both a head orientation, call it R_i, and a head centroid location, call it $\mathbf{t}_i$. We say that $R_i \in SO(3)$ and $\mathbf{t}_i \in \mathbb{R}^3$ are the votes of $\mathcal{T}_i$. To estimate the cluster center in both spaces simultaneously, we count the number of neighbors of $\mathcal{T}_i$ in each voting space. For another library sample $\mathcal{T}_j$ to count as a neighbor of $\mathcal{T}_i$, *both* the rotational and centroid votes of $\mathcal{T}_j$, which we call R_j and $\mathbf{t}_j$, have to be within a predefined distance to the votes of $\mathcal{T}_i$. That is, both conditions $\mathrm{d}(R_i, R_j) < d_r$ and $\|\mathbf{t}_i - \mathbf{t}_j\| < d_t$ have to hold, where $\mathrm{d}(\cdot,\cdot)$ is an appropriate distance function in SO(3). In our implementation, we use $d_r = 15^\circ$ and $d_t = 25$mm.

The set of winning library samples, shown in red in Figure 4(b) and (c), consists of the library sample that has the most neighbors as well as all neighbors of that sample. Assume there are N winning samples, $\mathcal{T}_1, \ldots, \mathcal{T}_N$. Recall that each $\mathcal{T}_i$ was allowed to vote because its TSP descriptor matched that of a test triangle $\mathbf{T}'_i$, with centroid $\boldsymbol{\mu}'_i$, that was sampled from $\mathbf{S}$. Our estimate of the head orientation is the average, $\overline{R}$, of the winning library samples' rotation votes. The facial landmark location estimates are computed by averaging the landmark vectors of the winning library samples after transforming them into the coordinate system of $\mathbf{S}$. The vectors $\mathbf{u}_{1k}, \ldots, \mathbf{u}_{Nk}$ corresponding to the kth

landmark (say, the tip of the nose) are transformed and averaged to yield the estimate $\widetilde{\mathbf{u}}_k$ of that landmark's position:

$$\widetilde{\mathbf{u}}_k = \frac{1}{N}\sum_{i=1}^{N}\left(\boldsymbol{\mu}_i' + \overline{R}\mathbf{u}_{ik}\right), \tag{2}$$

This procedure always produces consistent facial landmark estimates, since all of them stem from base triangles that vote for similar head orientations and centroid locations. Details on the distance in SO(3) that we use and how we average rotation matrices are provided in Section 4.2.

4. Implementation Details

4.1. Equilateral Triangle Sampling

Fast equilateral triangle sampling from 3D point clouds is needed in both phases of our method. The problem can be formulated as follows. Given a point cloud $\mathbf{S} \subset \mathbb{R}^3$ and two positive numbers d and l, generate an equilateral triangle $\mathbf{T} = (\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2) \subset \mathbb{R}^3$ with side length l such that the distance between each $\mathbf{q}_i$ and the closest point from $\mathbf{S}$ is smaller than d. Obviously, for certain point clouds and values for d and l, no such triangle exists. However, our point clouds are dense enough, and both d and l have appropriate values ($d = 3$mm leads to good results, and an optimal l is computed in Section 4.3).

First, we sample a seed point $\mathbf{p}$ uniformly from $\mathbf{S}$. If $\mathbf{S}$ is the set of vertices of a mesh, the seed point is sampled uniformly inside a randomly chosen mesh triangle, where the probability of choosing a triangle is proportional to its area. Next, we compute a normal direction $\mathbf{n}$ at $\mathbf{p}$. In the case of a mesh, we simply take the normal of the mesh triangle that $\mathbf{p}$ was sampled from. If $\mathbf{S}$ is a point cloud reconstructed from a depth image, we compute $\mathbf{n}$ as the average of the normals of the planes passing through $\mathbf{p}$ and pairs of its 4-connected neighbors. The neighborhood structure of $\mathbf{S}$ is the one imposed by the rectangular grid of the depth image.

Now that we have a seed point $\mathbf{p}$ and a normal $\mathbf{n}$, we generate an equilateral triangle $\mathbf{T}$ with side length l and transform it such that it lies in the plane defined by $\mathbf{p}$ and $\mathbf{n}$ and its centroid coincides with $\mathbf{p}$. This defines $\mathbf{T}$ up to a rotation about $\mathbf{n}$ by an angle which we select randomly from the interval $[0, 2\pi)$.

This yields a randomly generated triangle that meets all requirements except being close enough to the point cloud. To achieve this, we transform $\mathbf{T}$ to $\mathbf{S}$ using ICP [3]. An ICP iteration computes for each triangle vertex $\mathbf{q}_i$ the closest point from $\mathbf{S}$, call it $\mathbf{q}_i'$. Next, $\mathbf{T}$ is rigidly transformed such that the sum of squared distances between each $\mathbf{q}_i$ and $\mathbf{q}_i'$ is minimized. (see [3] for more details). Initially, $\mathbf{T}$ is not too far away from $\mathbf{S}$, so ICP typically needs no more than three iterations to converge. After that, we test if each triangle vertex is indeed within a distance of d units from $\mathbf{S}$. If not, the triangle is rejected and the whole procedure is repeated.

This sampling method generates triangles which uniformly cover the input point cloud. Note that if the depth image is too noisy and the normal $\mathbf{n}$ cannot be reliably computed, we can simply set $\mathbf{n}$ to be the negative of the depth sensor viewing direction (usually $[0, 0, -1]^{\mathrm{T}}$). In this case, the initial triangle is not as well aligned to the point cloud, and ICP is likely to need additional iterations to converge.

4.2. Processing in the Group of Rotations SO(3)

An essential part of our joint clustering approach presented in Section 3.3 are the notions of distance in the group of rotations and averaging of rotation matrices. Recall that the product $R_1^T R_2$ of two rotation matrices R_1 and R_2 is also a rotation matrix. Thus, $R_1^T R_2$ is equivalent to a single rotation by an angle θ about an axis $\mathbf{n} \in \mathbb{R}^3$. The function $\mathrm{d}_R(R_1, R_2) = |\theta|$ is known as the Riemannian distance [16] or the geodesic metric in SO(3) [13].

There is a connection between d_R and the Frobenius distance $\|R_1 - R_2\|_F$ given by the following equation [16]:

$$\|R_1 - R_2\|_F = 2\sqrt{2}\left|\sin\frac{\theta}{2}\right|, \tag{3}$$

where $\mathrm{d}_R(R_1, R_2) = |\theta|$. Thus, to determine if R_1 is within a given (Riemannian) distance θ of R_2, as required in Section 3.3, we check whether $\|R_1 - R_2\|_F < 2\sqrt{2}\left|\sin\frac{\theta}{2}\right|$.

The problem of computing the average rotation matrix $\overline{R}$ of $R_1, \ldots, R_n \in$ SO(3) can be formulated as a minimization problem, once a distance function $\mathrm{d}(\cdot, \cdot)$ is chosen:

$$\overline{R} = \operatorname*{arg\,min}_{R \in \mathrm{SO(3)}} \sum_{i=1}^{n} \mathrm{d}^2(R_i, R). \tag{4}$$

Unfortunately, (4) cannot be solved in closed form for the Riemannian distance. In our case, however, the input matrices are close to each other because they all lie in the neighborhood of the same rotation matrix. In this case, solving (4) using the Frobenius distance $\mathrm{d}(R_i, R) = \|R_i - R\|_F$ is a good approximation and can be done in closed form:

$$\overline{R} = WV^T, \tag{5}$$

where $W\Sigma V^T$ is the singular value decomposition of the arithmetic mean $\frac{1}{n}\sum_{i=1}^{n} R_i$. See [16] for a proof.

4.3. Parameter Optimization

The most important parameters of our method are the base triangle side length l, the number k of sub-triangles per side (both introduced in Section 3.1) and the number h of nearest neighbors retrieved from the library of training samples (see Section 3.3). In order to avoid a time-consuming manual tuning, we used a simple automatic procedure which computes an optimal combination.

We chose a range of values for each of the three parameters such that the lower and upper bounds of each range

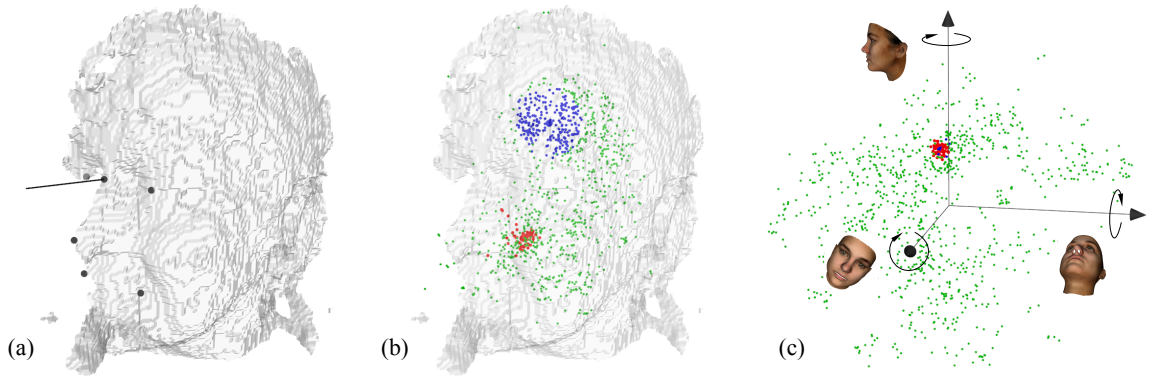

Figure 4. Our results on a frame from the Biwi Database [9]. (a) Estimated head orientation (black line) and landmark locations (black dots at nose top, eyes, nose tip, and mouth corners). (b) Head centroid voting space. All votes are shown as dots. If we clustered only in centroid voting space, the blue dots would be the winning votes, leading to an incorrect estimate. But since we cluster jointly in the centroid and orientation voting spaces, the red dots are the winning votes which give a correct estimate. (c) Orientation voting space. Each dot shows the axis-angle representation for one orientation vote. Joint clustering yields the winning votes shown in red. (For this frame, the winning orientation votes are almost the same whether clustering jointly or in orientation space alone.)

are extremes which lead either to suboptimal results or to too long computation time. In this way, a reasonable combination is expected to be in the interior of the tested parameter space: $l \in \{40, 60, 80, 100, 120\}$, $k \in \{4, 5, 6\}$ and $h \in \{1, 3, 5, 7\}$. The optimization was performed using 50 head models for training and 100 base triangles for online sampling. These parameters were not included in the optimization since the way they affect the accuracy is obvious: the bigger the numbers the better.

We recorded three depth image sequences of the same person moving his head from left to right and from top to bottom and waving with his hands while sitting in front of a Kinect sensor. We labeled 6 facial landmarks in all frames, namely, the bridge of the nose, the tip of the nose, the eye centers, and the mouth corners.

For a particular parameter combination, the facial landmark localization was performed on all frames and the root mean squared (RMS) error of the estimated locations was saved. The combination with the smallest RMS error, namely, $(l, k, h) = (80\text{mm}, 5, 5)$ was selected as the optimal one. However, other values led to very similar results, so we expect this particular parameter combination to be stable across different sensors and testing scenarios.

5. Experimental Results

All experiments use a single CPU core of a Windows 64-bit laptop with 12GB RAM and 2.4GHz Intel Core i7 CPU.

As already mentioned in Section 3.2, the training was performed on high-resolution BFM head meshes [21]. In addition, [21] used the BFM to fit 3D models to images of all 68 subjects in the CMU-PIE face image database [23]. Our training set includes these 68 3D head models based on the CMU-PIE subjects plus other 3D head models randomly generated using the BFM. The depth maps used for testing are from the Biwi Kinect Head Pose Database [9]. It consists of 24 sequences of RGB-D images from 20 different subjects moving their heads over a range of roughly $\pm 75°$ for yaw, $\pm 60°$ for pitch and $\pm 50°$ for roll.

To estimate the head pose and localize facial features given an RGB-D image, we first use a 2D multi-view face detector that roughly locates the head region in the RGB image. 2D face detection is now a well developed technology available in many commercial products. We use a Viola-Jones type detector very similar to the one described in [15] that takes 75 ms to process a single VGA image. This is a general multi-view face detector that detects all faces from left profile to right profile in a cluttered image. This is in contrast to the detector learned in Fanelli et al.'s work which was trained on, and thus very specific to, Biwi data.

After detecting the face, only those depth pixels that are within the 2D bounding box returned by the face detector are used to reconstruct the 3D point cloud. After that, the head pose estimation and facial feature localization are performed as described in Section 3.3. Figure 4(a) provides an example.

Our results are shown in Table 1. Each RGB-D image is processed independently, which implies that our method does not need any kind of initialization. If the 2D detector fails to detect a face, that frame is counted in the *Missed* column of the results table, but no pose or facial landmark position estimate is given and that frame is not included in the error estimates. The same protocol was used in [9].

We compare with Fanelli et al. [9] for their most accurate parameter setting (stride = 5). Two parameters that we can increase to improve accuracy (but reduce speed) are number of head models used in training and number of triangles sampled from each test depth image. We choose values for training set size (250 models) and number of triangles

Method	Nose tip (mm)	Rotation (°)	Direction (°)	Yaw (°)	Pitch (°)	Roll (°)	Missed	Time
Ours ($\triangle = 200$)	6.8 ± 14.2	4.5 ± 13.7	3.2 ± 7.9	2.5 ± 8.3	1.8 ± 4.3	2.9 ± 12.8	8.1%	75.1 ms
Ours ($\triangle = 100$)	8.6 ± 21.3	6.4 ± 19.2	4.4 ± 11.8	3.5 ± 12.6	2.5 ± 6.3	4.2 ± 17.2	8.1%	38.9 ms
[9] Trained on Biwi	12.2 ± 22.8	NA	5.9 ± 8.1	3.8 ± 6.5	3.5 ± 5.8	5.4 ± 6.0	6.6%	44.7 ms
[9] Synthetic Training	19.7 ± 46.5	NA	8.5 ± 12.9	6.0 ± 11.5	4.8 ± 7.1	5.8 ± 6.8	9.3%	44.0 ms

Table 1. Results on Biwi data set in "detection" mode

Method	Nose tip (mm)	Rotation (°)	Direction (°)	Yaw (°)	Pitch (°)	Roll (°)	Missed	Time
Ours ($\triangle = 200$)	8.4 ± 22.2	5.5 ± 16.5	3.9 ± 10.3	3.0 ± 9.6	2.5 ± 7.4	3.8 ± 16.0	0.0%	75.9 ms
Ours ($\triangle = 100$)	10.6 ± 28.1	7.8 ± 22.7	5.4 ± 14.3	4.3 ± 14.7	3.2 ± 8.8	5.4 ± 20.8	0.0%	37.9 ms
Baltrusaitis et al. [2]	NA	NA	NA	5.7	7.5	8.0	0.0%	NA

Table 2. Results on Biwi data set in "tracking" mode

($\triangle = 100$) so that our speed roughly matches that of [9].

We report the 3D error (in mm) of the nose tip landmark position (mean $\pm$ std, across all Biwi frames). Our training 3D models (from BFM) are well aligned with respect to orientation (frontal pose is consistent across all models), but the ground truth orientation values given in the Biwi test set are not aligned with the BFM models (ground truth frontal pose for each subject is not consistent). To compensate for this, for each Biwi subject we compute a single rotation matrix, by which we premultiply our orientation estimate for every frame of that subject's sequence, to minimize the error of the final estimates. Rotation error is the average Riemannian distance between the estimated and ground truth orientations. We also compute Direction error as in [9]: separately rotate the frontal vector $[0, 0, -1]^{\mathrm{T}}$ by the estimated and ground truth rotations, then measure the angle between the resulting two vectors. Yaw, pitch, and roll errors are the absolute difference between the Euler angles for ground truth orientation and those for estimated orientation. We also report missed detection rate and running time (not including RGB-D image acquisition and 2D face detection).

Our method is significantly more accurate than Fanelli et al. [9] for all pose angle and landmark position estimates. By increasing the number of sampled triangles to $\triangle = 200$, we further improve our accuracy while still maintaining a speed of over 13 frames per second. It is also interesting to note that Fanelli et al. got much better results when training on part of the Biwi database and testing on the rest (3rd row of Table 1). When they trained using synthetic heads for positive examples (but still Biwi data for negative examples), their accuracy declined significantly (4th row of Table 1). Our method was only trained on synthetic head models but still improves on the results of Fanelli et al. even when they trained using the Biwi database.

For the results in Table 1, we discard 8.1% of the input images due to our multi-view face detector failing to detect a face. This is a slightly higher than Fanelli et al.'s missed detection rate of 6.6% when they train on Biwi data, and slightly lower than their missed detection rate (9.3%) when they train on synthetic positive examples. Our missed detection rate could be improved by simply using a better 2D face detector. Another way to reduce the missed detection rate is to use a tracking approach. We did this in a very simple manner. When the 2D face detection failed on the current frame, we just used the 2D bounding box from the most recent previous frame in which a face was detected. In effect, this assumes that the face has moved little since the last detection, which is roughly true for the Biwi database. Using this "tracking" approach, our missed detection rate is 0. We show results for this case in Table 2 when sampling 100 or 200 base triangles per image. We compare against Baltrusaitis et al. [2], who also show results on the Biwi data sets using a much different tracking approach. As can be seen, our results are significantly better than Baltrusaitis et al. [2]. In fact, our results with 0 missed detections and slightly faster speed are comparable to the best results of Fanelli et al. [9] with 6.6% missed detection rate.

6. Conclusions

We presented a fast, accurate, and practical new technique for estimating 3D head pose and facial landmark positions from an RGB-D image. Our technique is based on a new TSP feature, which represents a surface patch of an object, along with a fast approximate nearest neighbor lookup to find similar surface patches from training heads. The correspondence between a testing and a training TSP feature yields an estimate for the head pose and the facial landmark locations. Since the estimate from a single TSP may be inaccurate, we sample many such patches, whose cluster mean gives robust estimates of pose and landmark positions.

Because we rely on synthetic 3D head models for training instead of real depth images of human heads, our training data are easy and inexpensive to generate. Furthermore, the resulting training library of feature descriptors is independent of a particular sensor and therefore more general than methods that rely on actual depth images for training.

A comparison with other methods demonstrate a significant improvement over the current state of the art in accuracy while maintaining real-time speed.

References

[1] A. Asthana, T. Marks, M. Jones, K. Tieu, and R. M.V. Fully automatic pose-invariant face recognition via 3d pose normalization. In *ICCV*, pages 937–944, 2011. 1

[2] T. Baltrusaitis, P. Robinson, and L.-P. Morency. 3d constrained local model for rigid and non-rigid facial tracking. In *CVPR*, 2012. 1, 2, 8

[3] P. Besl and N. D. McKay. A method for registration of 3-d shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2), 1992. 6

[4] A. Bleiweiss and M. Werman. Robust head pose estimation by fusing time-of-flight depth and color. In *IEEE International Workshop on Multimedia Signal Processing*, 2010. 2

[5] S. Bouaziz, Y. Wang, and M. Pauly. Online modeling for realtime facial animation. In *ACM Trans. on Graphics (SIGGRAPH)*, 2013. 1, 2

[6] D. Bradley, W. Heidrich, T. Popa, and A. Sheffer. High resolution passive facial performance capture. In *ACM Trans. on Graphics (SIGGRAPH)*, 2010. 2

[7] M. Breitenstein, D. Kuettel, T. Weise, and L. V. Gool. Real-time face pose estimation from single range images. In *CVPR*, 2008. 1, 2

[8] D. Cristanacce and T. Cootes. Feature detection and tracking with constrained local models. In *BMVC*, 2006. 2

[9] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. V. Gool. Random forests for real time 3d face analysis. *Int. J. of Computer Vision*, 101:437–458, 2013. 1, 2, 5, 7, 8

[10] G. Fanelli, J. Gall, and L. V. Gool. Real time head pose estimation with random regression forests. In *CVPR*, 2011. 2

[11] G. Fanelli, T. Weise, J. Gall, and L. V. Gool. Real time head pose estimation from consumer depth cameras. In *Proc. of the German Assoc. for Pattern Rec. (DAGM)*, 2011. 2

[12] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik. Recognizing Objects in Range Data Using Regional Point Descriptors. In *ECCV*, 2004. 3

[13] R. Hartley, J. Trumpf, Y. Dai, and H. Li. Rotation averaging. *International Journal of Computer Vision*, 103(3):267–305, 2013. 6

[14] A. Johnson and M. Hebert. Using Spin Images for Efficient Object Recognition in Cluttered 3D Scenes. *IEEE TPAMI*, 21(5):433–449, 1999. 3

[15] M. Jones and P. Viola. Fast multi-view face detection. Technical Report TR2003-96, Mitsubishi Electric Research Labs, 2003. 7

[16] M. Moakher. Means and averaging in the group of rotations. *SIAM Journal on Matrix Analysis and Applications*, 24(1):1–16, 2002. 6

[17] L.-P. Morency, J. Whitehill, and J. Movellan. Generalized adaptive view-based appearance model: Integrated framework for monocular head pose estimation. In *Face and Gesture*, 2008. 2

[18] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009. 3, 4

[19] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014. 3, 4

[20] P. Padeleris, X. Zabulis, and A. Argyros. Head pose estimation on depth data based on particle swarm optimization. In *CVPR Workshop on Human Activity Understanding from 3D data*, 2012. 2

[21] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter. A 3d face model for pose and illumination invariant face recognition. In *IEEE Intl. Conf. Advanced Video and Signal based Surveillance (AVSS)*, 2009. 1, 4, 7

[22] E. Seeman, K. Nickel, and R. Stiefelhagen. Head pose estimation using stereo vision for human-robot interaction. In *Face and Gesture*, 2004. 1

[23] T. Sim, S. Baker, and M. Bsat. The cmu pose, illumination, and expression database. *IEEE Trans. Pattern Anal. and Machine Intelligence*, 25(12), 2003. 7

[24] N. Smolyanskiy, C. Huitema, L. Liang, and S. Anderson. Real-time 3d face tracking based on active appearance model constrained by depth data. In *Image and Vision Computing*, 2014. 2

[25] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014. 1

[26] F. Tombari, S. Salti, and L. Di Stefano. Unique Signatures of Histograms for Local Surface Description. In *ECCV*, 2010. 3

[27] L. Valgaerts, C. Wu, A. Bruhn, H.-P. Seidel, and C. Theobalt. Lightweight binocular facial performance capture under uncontrolled lighting. In *ACM Trans. on Graphics (SIGGRAPH)*, 2012. 2

[28] T. Weise, S. Bouaziz, H. Li, and M. Pauly. Real-time performance-based facial animation. In *ACM Trans. on Graphics (SIGGRAPH)*, 2011. 2

[29] A. Zaharescu, E. Boyer, and R. Horaud. Keypoints and Local Descriptors of Scalar Functions on 2D Manifolds. *International Journal of Computer Vision*, 100(1), 2012. 3

[30] A. Zaharescu, E. Boyer, K. Varanasi, and R. Horaud. Surface Feature Detection and Description with Applications to Mesh Matching. In *CVPR*, 2009. 3, 4

[31] L. Zhang, N. Snavely, B. Curless, and S. Seitz. Spacetime faces: high resolution capture for modeling and animation. In *ACM Trans. on Graphics (SIGGRAPH)*, 2004. 2