3D Face Detection and Pose Estimation in PCL

Aitor Aldoma

September 27, 2012

1 Introduction

This document summarizes the work realized during the PCL code sprint sponsored by Honda. The task of the sprint was to integrate in the library methods to classify and localize any face present in the depth images such as those obtained from OpenNI compatible sensors. The progress of the sprint has been regularly reported in the code sprint blog[1].

During the first months, we tried to approach face detection as a general object recognition problem. Specifically, we tried a *Bag of Words* approach (see [5]) based on 3D local features, such as SHOT [6] or FPFH [4]. For the generation of the codebook we relied on the Reciprocal Nearest Neighbour (RNN [3]) algorithm. Even though the approach showed some potential we dismissed it early due to the high computational cost of computing the 3D features over the whole depth image. Also, in order to obtain the pose of the face after detection, additional steps would have been required, thus increasing the computational cost further.

Therefore, we decided to continue with real-time face detection approaches that are based on simpler features and machine learning techniques. The most prominent being the one proposed by Fanelli et al. that rapidly estimates wether faces are present in the scene and simultaneously retrieve their location and orientation [1,2]. One of the advantages of this method is that it relies solely on geometrical information and therefore is able to perform under different light conditions. Due to recent contributions in PCL regarding machine learning, we used and extend the machine learning module to integrate the training and detection stages from Fanelli's method into PCL. The following sections present the method in more detail as well as its usage within the Point Cloud Library.

2 Real-time 3D Face Detection

The method we describe is based on *regression random forests* which extend random forests to perform classification as well as regression and therefore, in our scenario, allows to simulatenously detect faces in a given depth image as well

[1]`http://www.pointclouds.org/blog/hrcs/aaldoma/index.php`

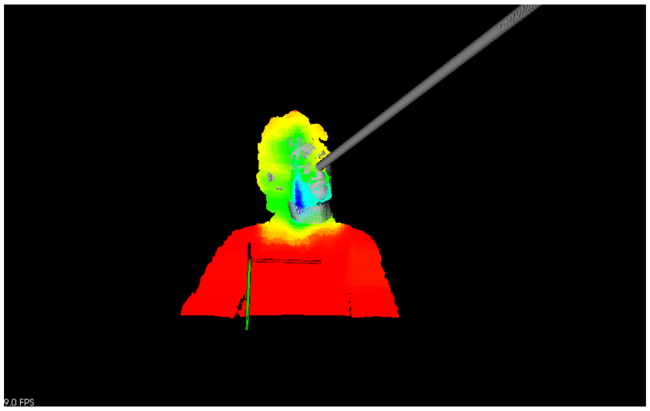


Figure 1: An example detection using the method implemented in PCL. The color palette represents how many times a specific point was voted as belonging to a face, going from dark blue to red. The gray cylinder indicates the direction of the nose starting from the center location of the head.

as estimate their position and orientation. Figure 1 shows an example of such a detection. As usual in this cases, the method consists of an off-line training stage during which the random forest is built and an on-line detection stage where the patches extracted from the current frame are classified using the trained forest. A leaf of the trees composing the forest stores the ratio of *face* patches that arrived to it during training as well as two multi-variate gaussian distributions *voting* for the location (μ_1, Σ_1) and orientation of the head (μ_2, Σ_2).

2.1 Training

In order to successfully train a regression random forest labeled training (*face*, *no-face*) data is needed. Additionaly, for regression, the location and orientation of the training heads is required as well. Figure 2 exemplifies a labeled frame obtained with Kinect from the *Biwi Kinect Head Pose Database*[2] showing *face* labels in green, red otherwise. With each frame, the head center location and orientation is as well provided. The dataset in PCL format can be download from[3].

Hence, a training example consists of the following elements:

- A patch of size wxw with its location (u, v) in the depth image.
- A label (1 for face patch, 0 for non-face)

and for positive samples (label=1):

[2]`http://www.vision.ee.ethz.ch/~gfanelli/head_pose/head_forest.html#db`
[3]`http://svn.pointclouds.org/data/biwi_face_database/pcl_biwi_version.tar.bz2`

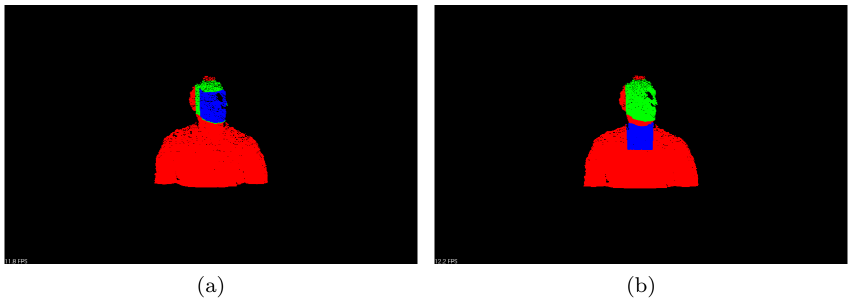


Figure 2: Training labeled data: a) a positive patch extracted on the face area b) a negative patch extracted somewhere else. The green points represent the face points, the red points are non-face points and blue areas represent a window patch used as training example.

- A vector $v_t = (x_t, y_t, z_t)$ representing the translation between the center of the patch and the head center location.
- A vector r_t representing the yaw, pitch and roll values of the head.

For each Kinect frame used during training, 20 positive and negative sample patches are extracted. Randomness is introduced in the forest in two ways: (i) bagging on the training data (each tree sees only a subset of the whole training set) and (ii) each split node generates a number N of random tests to grow the tree. The random tests at each split node are based on the difference of the respective average depth sum of two rectangles inside a patch and a threshold τ (see [1] for more details regarding the training procedure).

In order for the forest to perform classification and regression, we adopt a linear scheme that first concentrates in improving classification and regression gets more and more weight as the ratio of face patches in a specific branch of the tree becomes higher. At each split node, the test maximizing classification of face and non-face patches combined with regression accuracy is selected to split the training set at the node. Again, we refer to the reader to [1] for more details as well as the mathematical aspects behind the training procedure.

This process results in a ensemble of trees each leave storing the ratio between positive and total patches as well as gaussian distributions, respectively voting for the location and orientation of the head. The resulting forest, $\mathcal{F}$ is used during detection to generate face poses.

2.2 Detection

During detection, the depth frame is processed using a sliding window approach. The sliding window has the same size as the one used to extract positive and negative patches during training, w. A stride parameter for the sliding window is introduced to speed up the computation. As the window slides through the

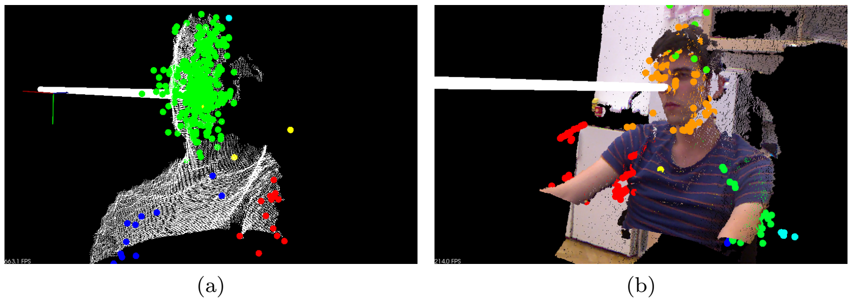


Figure 3: The coloured big points represent the votes obtained by the detection stage on two different point clouds (one in the training set, one outside of it). Each color represents the results of the first grouping stage.

image, each patch is analyzed by the forest $\mathcal{F}$ resulting in a set of leaves (as many as trees in the forest, n). Because each leaf is associated with a location and orientation, the patch results in n possible locations and orientations. At this point, votes coming from leaves with a low ratio of positive faces or with a high variance regarding location and orientation, are dismissed in order to improve robustness.

The remaining votes (those that according to the trees belong to faces and have an accurate pose prediction) are clustered together in a two step process. First, using the average head diameter a rough clustering is made that groups together votes that come from the same head. Figure 3 illustrates this first clustering stage. As it can be seen from the images, this first stage contains several outliers regarding the central location of the head. The second stage aims at removing this outliers in order to obtain a more accurate head location. To do so, 10 iterations of mean-shift cluster with a radius of approximately 4cm is used that better estimate the center location of the head.

After clustering, a face/head is said to be present in the scene if the clusters after mean-shift consists of more than min_votes votes. The location and orientation of the detected faces is obtained by averaging the corresponding votes (location and orientation) in the cluster. Figure 5 show some detection results and Figure 6 depicts some problems you might encounter.

2.2.1 6DOF pose refinement

We have extended the method with a pose refinement stage based on the PCL registration API that iteratively allows to improve the detected pose. Figure 4 shows an example of the improvement in pose obtained with such a stage. The pose refinement stage is performed between the current scene, $\mathcal{S}$ (downsampled to a 5mm) and a generic face model $\mathcal{M}$ that is initially transformed with the pose obtained by the real-time method previously presented. Roughly it works as follows:

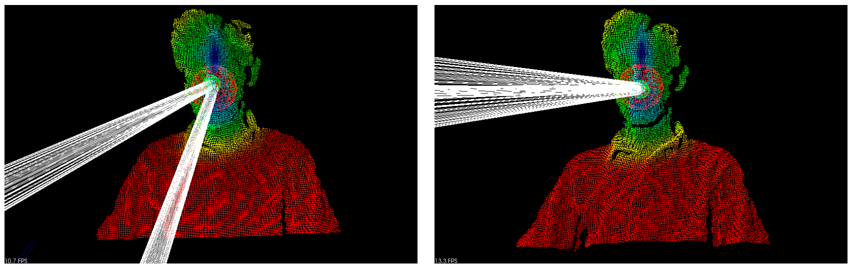

(a) (b)

Figure 4: A frame from the Kinect with the detected face as well as the ground truth pose. (a) Without pose refinement (b) With pose refinement.

- Establish correspondences between $\mathcal{S}$ and $\mathcal{M}$ using *NormalShooting* as correspondence estimation method.
- Reject correspondences with a RANSAC based method.
- Estimate the incremental transformation with *point to plane* Linear Least Squares optimization.

Unfortunately, activating this stage, results in a slower performance loosing the real-time capabilities of the aforementioned method. However, the motivation for this additional stage is to obtain an accurate as possible pose in a calibration stage that might be used to initialize a face tracker (visual or geometry based) to follow the face throughout a sequence.

3 Contributions to PCL

The contributions to PCL have been mainly realized in the following three modules:

- **pcl_ml**: The initial random forest implementation has been modified in order two increase randomness capabilities as well as to do *bagging* on the training data by means of a data provider interface allowing to train on bigger datasets. Regarding the latest aspect, a generic data provider interface has been added to the random forest implementation that is called before training a tree in order to request a subset of the training set. This subset will be generated by the user's data provider specialization and used to train the specific tree. This allows to keep in memory only the subset used to train the tree and therefore reduce the memory footprint of the training stage.
- **pcl_recognition**: The classes implementing the logic of the method covered in this document have been added in the recognition module. The

classes allow the user to train a random forest as well as perform face detection on an input point cloud. Please note, that due to the nature of the method, it can only be applied in the so called *organized* point clouds.

- **pcl_apps**: Finally, we have include several applications to show the performance of the method. These applications interface with the recognition module and allow detection from several sources (OpenNI streams, filesystem) as well as training. The next section covers the usage of the applications in order to perform training and recognition.

3.1 Usage

You can find a sample forest[4] and a face cloud for the pose refinement stage[5] in the PCL data repository.

3.1.1 pcl_openni_face_detector

The most important parameters are:

- *-forest_fn*: Path to the trained forest.
- *-max_variance*: The maximum variance allowed for a vote to be considered.
- *-face_threshold*: A threshold (between 0 and 1) indicating the minimum ratio of positive patches for a vote to be considered.
- *-min_votes_size*: Minimum number of votes in a cluster after mean-shift to accept a head hypothesis.
- *-stride_sw*: The stride for the sliding window. Trade-off between speed and number of votes generated.

If you would like to activate the pose refinement stage, the following parameters need to be set:

- *-pose_refinement*: Boolean value indicating wether pose refinement stage is activated (1) or not (0).
- *-model_path*: Path to the face model used for refinement.
- *-icp_iterations*: Number of pose refinement iterations.

Usage example:

```
./bin/pcl_openni_face_detector -face_threshold 0.99
-max_variance 2400 -min_votes_size 300 -stride_sw 4
-heat_map 0 -show_votes 1 -pose_refinement 1 -icp_iterations 5
-model_path model.pcd -forest_fn forest_example.txt
```

[4] http://svn.pointclouds.org/data/biwi_face_database/forest_example.txt
[5] http://svn.pointclouds.org/data/biwi_face_database/model.pcd

3.1.2 pcl_fs_face_detector

Parameters in this case are almost the same than the previous application except for:

- *-directory*: A directory containing PCD files that will be processed in a sequence.
- *-filename*: If -directory is not set, a single PCD file can be specified.
- *-video*: With -directory set, wether visualization should stop at each frame or not.

Usage examples:

```
./bin/pcl_fs_face_detector -face_threshold 0.99
-max_variance 2400 -min_votes_size 300 -stride_sw 4
-heat_map 0 -show_votes 1 -pose_refinement 1 -icp_iterations 5
-model_path model.pcd -forest_fn forest_example.txt -filename scene.pcd
```

```
./bin/pcl_fs_face_detector -face_threshold 0.99
-max_variance 2400 -min_votes_size 300 -stride_sw 4
-heat_map 1 -show_votes 0 -pose_refinement 0 -icp_iterations 5
-model_path model.pcd -forest_fn forest_example.txt -directory /home/aitor/faces/seq1/
-video 1
```

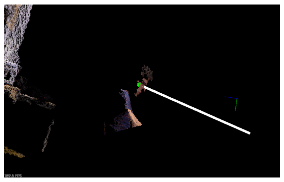
If you would like to use face detection in your code, you can have a look at the source code of this application in order to see how to instantiate the face detector.

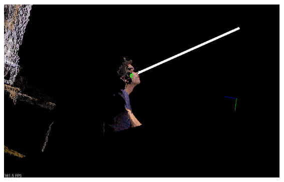
3.1.3 pcl_face_trainer

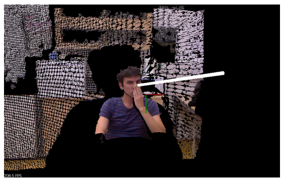
Finally, this application allows to train a new forest. The parameters are:

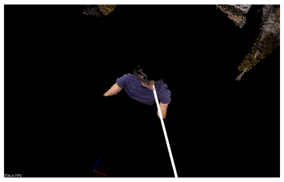
- *-ntrees*: Number of trees to be trained.
- *-n_features*: The number of random features to be generated at each split node.
- *-directory*: A directory containing labeled training data.
- *-num_images*: Number of images (depth scenes) to extract training samples.
- *-forest_fn*: Path in your compter where the forest will be saved after training.

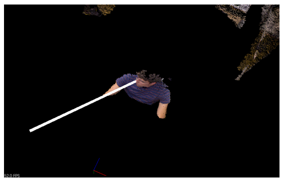
Be aware that training a huge forest might take several hours.

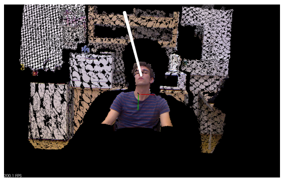

(a)

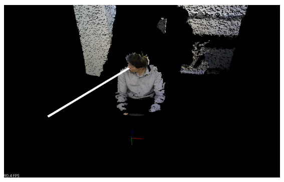

(b)

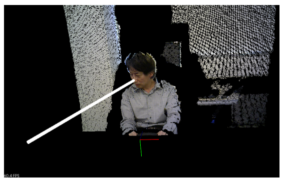

(c)

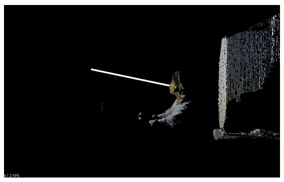

(d)

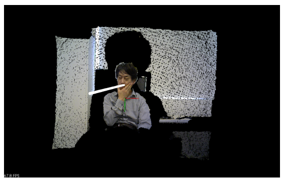

(e)

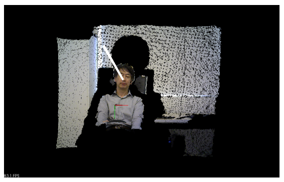

(f)


(g)


(h)


(i)


(j)


(k)

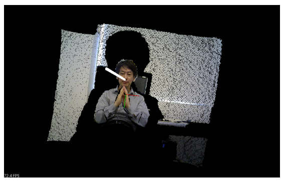
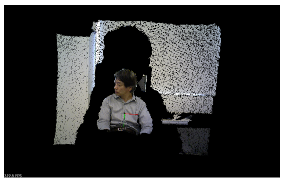
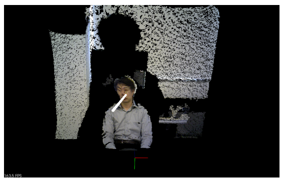

(l)

Figure 5: Some correct detection under different conditions of yaw and pitch as well as partial occlusions (c, j). Results obtained without pose refinement. You can find some other results in form of video in the code sprint blog.


(a)


(b)

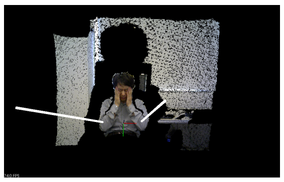

(c)

Figure 6: Some situations where the head is not detected, RGB image is blurry or there are false positives.

4 Limitations and future improvements

4.1 Limitations and problems

- The trained forest was trained mainly on adult caucasian subjects and therefore, for subjects belonging to another ethnic group or children, the results might not be so good. This indicates that the training set should be extended to better and uniformly cover this variability within human faces. Another reason causing this might be related to over-fitting.
- Due to the computation of integral images, the method works only on organized data such as that obtained with Kinect, Asus Xtion, etc.

4.2 Future improvements

- As mentioned before, a possible improvement would be related to the integration of a visual or geometrical tracker that could run in parallel to the detection method to increase robustness. The tracker would be initialized with the face poses obtained by the detector.
- An in-deep review of the training process should be performed to address some situations that are not completely clear at the moment.
- Improvements in the **pcl_ml** module regarding training speed as well as out-of-core solutions allowing to train on even bigger datasets.
- Right now, the presence of a face or not is decided by the *min_votes_size* threshold. In this aspect, might be interesting to investigate the possibility of including an hypotheses verification stage. Such a stage is a very important piece in 3D object recognition allowing to keep correct hypotheses with low support (votes in this case) while still removing hypotheses than geometrically or visually do not fit to the scene.
- Related to face variability and to the pose refinement stage, a morphable face model with non-rigid pose refinement would be an interesting aspect to further research. Such an approach might deal with variations in scale as well as shape that are obviously not covered with the current rigid and scale dependant approach.

4.3 Acknoweledgements

The work leading to these results has been achieved during the PCL HRI Code Sprint, funded by Honda. Special thanks to Gabriele Fanelli for his research as well as for allowing us to publish the *Biwi Kinect Head Pose Database* on the PCL data repository.

References

[1] Gabriele Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool. Random forests for real time 3d face analysis. *Int. J. Comput. Vision*, 2012.

[2] Gabriele Fanelli, Thibaut Weise, Juergen Gall, and Luc Van Gool. Real time head pose estimation from consumer depth cameras. In *Proceedings of the 33rd international conference on Pattern recognition*, DAGM'11, 2011.

[3] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *Int. J. Comput. Vision*, 77(1-3):259–289, 2008.

[4] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D Registration. In *ICRA*, 2009.

[5] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, October 2003.

[6] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of Histograms for local surface description. In *Proc. 11th ECCV*, 2010.