Implementation of Fast 3D Recognition and pose using Viewpoint Feature Histogram using Kinect cameras

***Abstract*—The present report show a procedure that was follow in order to analize a powerful meta-local descriptor and implement a demo that can show the usage of such descriptor. This descriptor is called Viewpoint Feature Histogram and was created for the purpose of recognition of 3D objects and its pose estimation. This kind of methods are widely used to develop capabilities in the area of perception for robots object manipulation. The demo develop and presented in this report show the produce on how the information is obtained from stereo data in real time and used to identify and describe 3D objects and its pose.**

***Index Terms*—Meta-local descriptor, 3D object recognition, pose estimation, Point Feature Histogram.**

I. Introduction

In order to manipulate an object, robots must reliably identify it and estimate its 6 degree-of-freedom pose. Even though there are many developed methods for this kind of task, they are still dealing with some problems such us reflection or transparency in objects, occlusion, real time processing,etc.

Most of them are not accurate enough for robot manipulation or computationally low for real time operation. However, the Viewpoint Feature Histogram can manage these problems and show good results. [3] The structure of this report is as follows: We describe Viewpoint Feature Histogram in Section II, the experimental results and demo construction are presented in Section III. Finally, conclusions are discussed in Section IV.

II. Viewpoint Feature Histogram VFH

This descriptor is based on two previous descriptors, Point Feature Histogram PFH [1] and the Fast Point Feature Histogram FPFH [2]. So in order to understand VFH, we will go deep into these to descriptors first.

A. Point Feature Histogram PFH

The PFH is an histogram that collects the pairwise pan, tilt and yaw angles between every pair of normals on a surface patch, Thus for a pair of 3D points (p_i, p_j), and their respective estimated surface normals n_i, n_j the set of normal angular deviations can be estimated as:

$$\alpha = \mathrm{v} \cdot \boldsymbol{n}_j$$
$$\phi = \mathrm{u} \cdot \frac{(\boldsymbol{p}_j - \boldsymbol{p}_i)}{d}$$
$$\theta = \arctan(\mathrm{w} \cdot \boldsymbol{n}_j, \mathrm{u} \cdot \boldsymbol{n}_j)$$

Fig. 1: Pan, tilt and Yaw angles estimation.

where u, v, w represent a Darboux frame coordinate system chosen at p_i. Then, the Point Feature Histogram at a patch of points $P = pi$ with $i = 1n$ captures all the sets of (α, ϕ, θ) between all pairs of (p_i, p_j), and bins the results in a histogram. Figure 2 shows the relation between a pair of points that is taken to estimate the three angles at a point p_s.

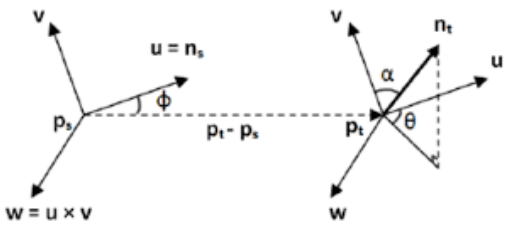


Fig. 2: PFH

Because all possible pairs of points are considered, the computation complexity of a PFH is $O(n^2)$ in the number of surface normals n [3]. In order to make a more efficient algorithm, the Fast Point Feature Histogram [2] was developed and we will check it in the next subsection.

B. Fast Point Feature Histogram FPFH

The FPFH measures the same angular features as PFH, but estimates the sets of values only between every point and its k nearest neighbors. The theoretical computational complexity of the Point Feature Histogram for a given point cloud P with n points is $O(nk^2)$, where k is the number of neighbors for each point p in P. For real-time or near real-time applications, the computation of Point Feature Histograms in dense point neighborhoods can represent one of the major bottlenecks.

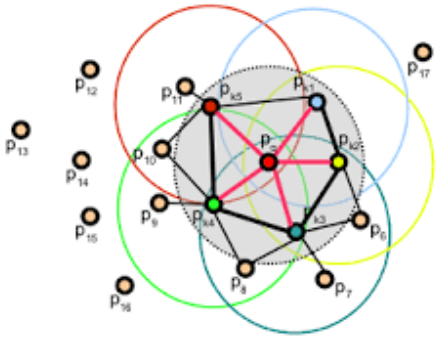


Fig. 3: K-nearest neighbors considered FPFH.

C. Viewpoint Feature Histogram

Calculating the FPFH help with the object identification task, however recognizing its six degree of freedom pose for grasping is also necessary. FPFH is invariant both to object scale (distance) and object pose and so the second task is not yet achieve. To do this, we used the key idea of mixing the viewpoint direction directly into the relative normal angle calculation in the FPFH.

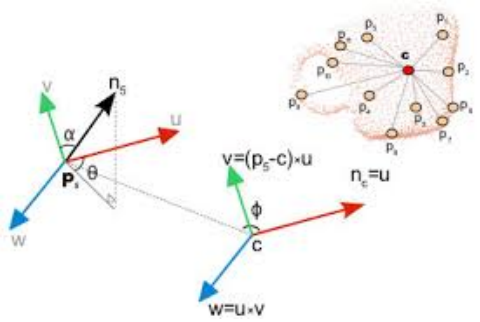


Fig. 4: VFH

The viewpoint component is computed by collecting a histogram of the angles that the viewpoint direction makes with each normal. Note, we do not mean the view angle to each normal as this would not be scale invariant, but instead we mean the angle between the central viewpoint direction translated to each normal.

The second component measures the relative pan, tilt and yaw angles as described in Fast Point Feature Histograms (FPFH) descriptors but now measured between the viewpoint direction at the central point and each of the normals on the surface. [3].

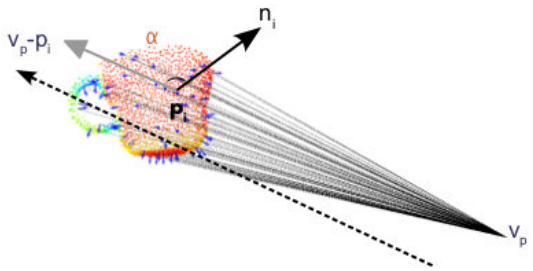


Fig. 5: Viewpoint direction at the central point

The major difference between the PFH/FPFH descriptors and VFH, is that for a given point cloud dataset, only a single VFH descriptor will be estimated, while the resultant PFH/FPFH data will have the same number of entries as the number of points in the cloud.

Finally the figure 6 shows the histogram calculated using a VFH, in the image we clearly can identify two parts:
- A viewpoint direction component and
- A surface shape component comprised of an extended FPFH.

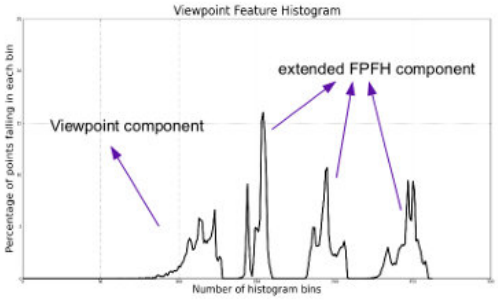


Fig. 6: VFH Histogram

III. Experimental results

In this section, we will present our results using simple object (see Fig.7), we choose this one because is a common object and easy to manipulate and move in several angles or point of view. Following to the work that we based this implementation,

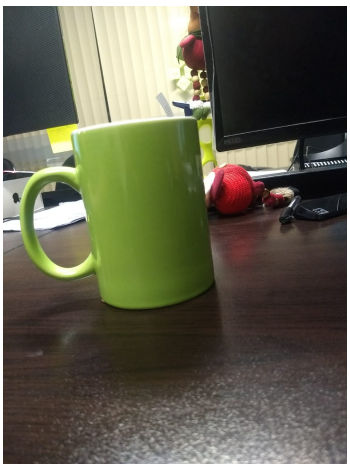


Fig. 7: A cup, our object to study

We use Point cloud Library [5] and libfreekinect [6] library to manipulate kinect camera.
Our first step was calibrated kinect camera, once we have those parameters, we get RGB image, Depth image and also IR image (this is provided by kinect) as you see in Fig.8

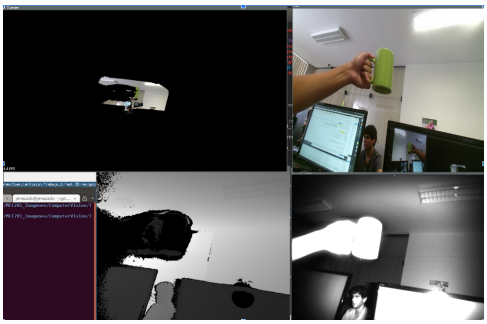
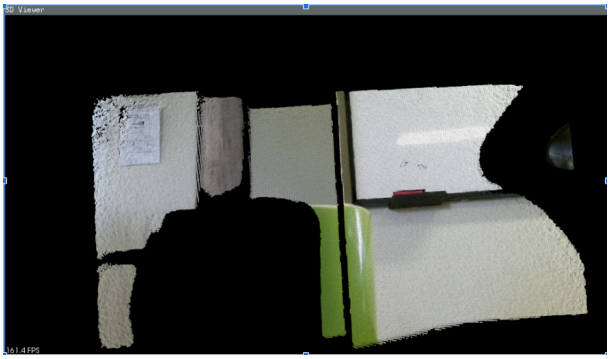


Fig. 8: Depth, RGB and Ir images from kinect

Next step was extract from kinect (using RGB and Depth images) points clouds to generate our point surface, in this step, we use FPFH and some methods to get pt and finally convert that into point clouds data (PCD) (See fig. 9).



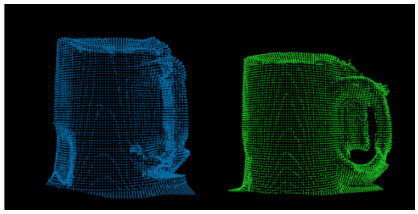
(a) Pint cloud obtained

(b) Point cloud normalize

Fig. 9: Point cloud and point cloud normalize.

Next step, was how to identify object in middle of all noise (or other objects), we use FLANN (Fast Library for Approximate Nearest Neighbours) included in PCL library to normalize the contour of the object.

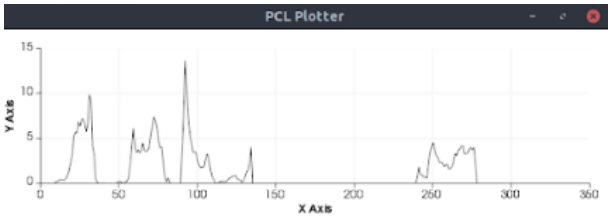
As well as, a set functions that can be used to normalize point cloud data. The author of this work also recommends to use this libraries.



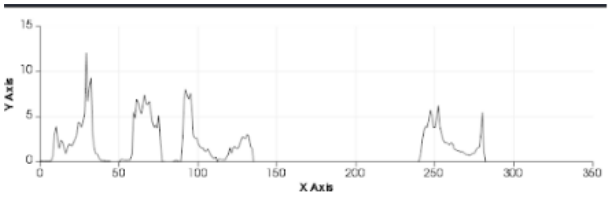
(a) Position 1 (b) Position 2

Fig. 10: Point cloud of cup in different positions.

Final step, once we get object with point clouds normalized we generate the respective VFH for any position of the object, VFH it varies depends of positions and how many points we select to do normalization, bellow you can see our VFH generated from our 2 positions showed above.



(a) Position 1 VFH



(b) Position 2 VFH

Fig. 11: VFH of cup in different positions.

IV. Conclusions

We implement Real time VFH using our kinect camera, our method is a bit computationally cost which depends of how many points (neighbors) to analyze. Our real time method provide an iteractive way to extract RGB, Depth from image and get point clouds from them.

Also, the VFH provides information for both tasks, identification of objects and recognition of approximate pose and distance from the point of view.

References

[1] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, Learning Informative Point Classes for the Acquisition of Object Model Maps, in In Proceedings of the 10th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2008.

[2] R. B. Rusu, N. Blodow, and M. Beetz, Fast Point Feature Histograms (FPFH) for 3D Registration, in ICRA, 2009.

[3] R. B. Rusu, G. Bradsky, R. Thibaux and Jhon Hsu. "Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram" in ICRA, 2009.

[4] http://pointclouds.org/documentation

[5] http://pointclouds.org/

[6] https://github.com/OpenKinect/libfreenect2/