Project	TAMS Feature	**Page**	1/**??**
Purpose	Feature based on SEI inherited from Feature Class of PCL	**Updated**	05/01/2015
Version	3.0	**Application**	Scan Registration/ Object Recognition
Library	PCL	**Designer**	Bo Sun

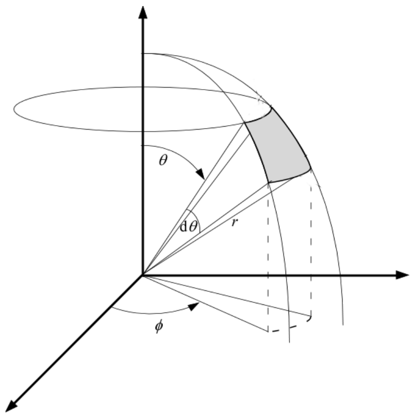

Figure 1: 3D patch when compute SEI

1 Theoretical Background

1.1 State-of-the-art

1.2 Spherical Entropy Image (SEI)

The original scan is divided into bins by equally spaced boundaries in the azimuth and elevation dimensions. As shown in Fig. 1, the 3D patch is an analogous square pyramid taking the origin of the scan as the vertex, but the bottom surface is not a plane but the spherical grid. The depth of points belonging to the same 3D patch could be interpreted as the observations of a random variable P, then the information entropy $E(p)$ of the variable P is regarded as the value of structural representation of the patch:

$$E(P) = -\sum_{d \in D} p(P = d) \cdot log\{p(P = d)\} \tag{1}$$

where D is the set of possible values of P, and p is its Probability Density function (PDF).

The process of calculating entropy is:

1. divide the 3D point cloud into several patches according to the polar angle and azimuth angle of points;
2. for each patch, considering the depth of points as the observations of a random variable P, normalize the depth of points, build the histogram and compute the PDF of variable P;
3. compute the entropy of variable P based on the probability distribution.

The structural representation of the 3D point cloud is achieved by computing the entropy of patches in a dense manner. We name this type of representation as Spherical Entropy Image (SEI) [**?**].

1.3 Local Reference Frame

The SEIs contain the orientation information of the original point clouds. In order to obtain the rotation invariant versions of SEIs, we have two directions:

- firstly, project SEI into its rotation invariant representation, just like what TAMSFeature2.0 does;
- or secondly, build a repeatable & unambiguous local reference frame related to the internal structure of the original point clouds the SEIs will describe, and compute SEIs in the new local reference frame.

In TAMSFeature3.0, we implement the second method.

With regard to the local reference frame of the point clouds, the Principle Component Analysis(PCA) is traditionally the first choice to achieve this. PCA is mathematically defined as an orthogonal linear transformation that transforms the data to a new coordinates system such that the greatest variance by some projection of the data comes to lie on the first coordinate(called the first principal component), the second greatest variance on the second coordinate, and so on. PCA can be thought as revealing the internal structure of the data in a way that best explains the variance in the data.

PCA can be done by singular value decomposition(SVD) of a data matrix: $X = U\Sigma V^T$. Σ is a diagonal matrix of positive numbers called the singular values of X; the columns of U are orthogonal unit vectors called the left singular vectors of X; and the columns of V are orthogonal unit vectors called the right singular vectors of X. If the X is a *symmetric matrix*, the decomposition $X = U\Sigma U^T$ is also a singular value decomposition.

The conventional way of building the coordinate system based on PCA is: define the covariance matrix M of the k-nearest neighbours p_i of the query point: $M = \frac{1}{k}\sum_{i=0}^{k}(p_i - \hat{p})(p_i - \hat{p})^T$, $\hat{p} = \frac{1}{k}\sum_{i=0}^{k} p_i$; compute SVD of M and then build the coordinate system based on the eigenvectors of M. By the way, the normal estimation in PCL is based on PCA of a covariance matrix created from the nearest neighbours of the query point, and the eigenvector according to the smallest eigenvalue is regarded as the normal of the query point. But the SVD itself provides no means for assessing the sign of each singular vector.

[**?**] provides a solution to the sign ambiguity problem. It is suggested that the sign of the singular vector should be similar to the sign of the majority of vectors it is representing. So the sign of singular vector could be determined from the sign of the inner product of the singular vector and the individual data vectors. The data vectors may have different orientation but it makes intuitive as well as practical sense to choose the direction in which the majority of the vectors point. The author of [**?**] adopt this sign-disambiguation technique to develop a new algorithm to estimate a repeatable local RF of local surface as follows:

- search the points laying within the spherical support (of radius R)
- compute matrix M as a weighted linear combination
  $M = \frac{1}{\sum_{i:d_i<R}} \sum_{i:d_i<R}(R-d_i)(p_i-p)(p_i-p)^T, d_i = \|p_i - p\|_2$
- refer to the three eigenvectors in decreasing eigenvalue order as the x^+,y^+ and z^+ axis respectively. With x^-,y^- and z^-, we denote instead the opposite vectors.
- the final disambiguated x axis is defined as follows. The same procedure is used to disambiguate the z axis. Finally, the y axis is obtained as $z \times x$.

$$S_x^+ \doteq \{i : d_i < R \wedge (p_i - p)\dot{x}^+ \geq 0\}$$
$$S_x^- \doteq \{i : d_i < R \wedge (p_i - p)\dot{x}^- > 0\}$$
$$x = \begin{cases} x^+, & |S_x^+| \geq |S_x^-| \\ x^-, & otherwise \end{cases}$$

1.4 Polar and Azimuth in new coordinate system

When the Cartesian coordinate system with:

$$x\_axis = [1.0, 0.0, 0.0]$$
$$y\_axis = [0.0, 1.0, 0.0]$$
$$z\_axis = [0.0, 0.0, 1.0]$$

the azimuth(θ) and polar(ϕ) could be easily computed by following:

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = tan^{-1}\left(\frac{y}{x}\right)$$
$$\phi = cos^{-1}\left(\frac{z}{r}\right)$$

But how to compute the azimuth and polar of the point when the axis of the coordinate system are not the normal ones?

- project the point onto the xy plane of the coordinate system, compute the vector $\overrightarrow{P}$ from origin to projection and normalize it
- compute the angle between the normalized $\overrightarrow{P}$ and the x_axis to achieve azimuth θ
- compute the angle between the vector from origin to point and z_axis to achieve polar ϕ

1.5 Algorithm in this package

In this package, we add a new point type to PCL: TAMSFeatureType, and define a new rotation invariant feature based on SEI and the repeatable local reference frame.

We define a new class named TAMSFeatureEstimation which is inherited from class Feature of PCL.

✗ Public Types

```
typedef boost::shared_ptr
<TAMSFeatureEstimation<PointInT, PointOuT> > Ptr ;
typedef boost::shared_ptr
<const TAMSFeatureEstimation<PointInT, PointOuT> > ConstPtr ;
```

✗ Public Member Functions

```
TAMSFeatureEstimation ()

inline void setEntropyBinDim (size_t tams_entropy_bin_);
inline void setSEIAzimuthDim (size_t tams_sei_azimutch_dim_);
inline void setSEIPolarDim (size_t tams_sei_polar_dim_);
inline int getEntropyBinDim ();
inline int getBandwidth ();
inline int getSEIAzimuchDim ();
inline int getSEIPolarDim ();
```

```
void computeFeature (PointCloudOut &output);
```

```
/** computeFeature is a virtual function of Class Feature
  * which means you have to define a computeFeature function
  * for the class inherit from Class Feature
  * Usually, this is the critical function of
  * FeatureEstimation Class
  */
```

✗ Protected Member Functions

```
void tams_cart2sph (float x, float y, float z,
                        float& azimuth, float& polar);
```

```
/** cart2sph(X,Y,Z,azimuth,polar) transforms Cartesian coordinates stored in
  * corresponding elements of X, Y, and Z into spherical coordinates.
  * azimuth and polar are angular displacements in radians.
  * azimuth(longitudinal) is the counterclockwise angle in the x-y plane
  * measured from the positive x-axis.
  * polar(colatitudianl) is the polar angle measured from the positive z axis.
  * 0 < azimuth < 2*M_PI; 0 < polar < M_PI
  */
```

```
    void tams_vector_normalization (std::vector<float> &tams_vector);
```

```
/** tams_vector_normalization normalize the input vector
  * Parameters:
  * [in]     tams_vector   the input vector
  * [out]    tams_vector   the normalized vector
  *                              (values are in range [0,1])
  */
```

```
    void tams_vector2entropy(const std::vector<float> & tams_vector,
                             float &entropy);
```

```
/** tams_vector2entropy compute the entropy of a vector
  * Parameters:
  * [in]   tams_vector     the input vector
  * [in]   hist_bin        the size of histogram in entropy computation
  * [out]  entropy         the resultant entropy
  */
```

```
void tams_sph (int l, int m, double polar, double azimuth,
                  gsl_complex &spharmonics);
```

```
/** tams_sph compute the spherical harmonics given the l, m, polar and azimuth
  * according to the following formula:
  * Y_{l}^{m}(polar,azimuth) =
  *      Sqrt[(2*l+1)/4*Pi]
  *  *Sqrt[Fractorial[l-m]/Fractorial[l+m]]
  *  *P_{l}^{m}(Cos[polar])
  *  *Exp[m*azimuth*I]
  * Please note that m should be positive, which is the requirement of GSL functions.
  * For the m<0, we adopt the relationship between the spherical harmonics with
  * the plus and minus sign of m:
  * Real(Y_{l}^{m} (polar,azimuth)) = (-1)^m * Real(Y_{l}^{-m} (polar, azimuth))
  * Imaginary (Y_{l}^{m} (polar,azimuth)) = (-1)^(m+1) Imaginary (Y_{l}^{-m} (polar, azimuth))
  */
```

✗ Private Attributes

```
size_t tams_sei_azimutch_dim_,
       tams_sei_polar_dim_,
       tams_entropy_bin_;
float tams_sei_azimutch_spa_, tams_sei_polar_spa_;
```

Next, we discuss what the critical function computeFeature does. For each keypoint:

A. search the neighbours of keypoints

B. compute the local reference frame with the keypoint as origin

C. compute the SEI of the neighbours

2 Package Structure

✗ CMakeLists.txt
used by **CMake**, the cross-platform, open-source build system. Make this package be easily used under other OS.

✗ FindFFTW3.cmake
.cmake file used by **CMake** to tell compiler the location of **FFTW3** library. Copy it to the **Modules** folder of CMake, then it is unnecessary to set the **FFTW3** library location by hand.

✗ `tams_feature_type.hpp`
Define a new point type tams::TAMSFeatureType and register it to PCL.

✗ `tams_feature.h`
Define a Class named tams::TAMSFeatureEstimation.

✗ `tams_feature.hpp`
Implementation of the functions of tams::TAMSFeatureEstimation.

✗ `tams_feature.cpp`
Example shows how to use the class tams::TAMSFeatureEstimation.

3 Package Dependency

The package was developed and tested in the GNU/Linux environment. But we give the *CMakeLists.txt* which is used by CMake, the cross-platform, open-source build system. We believe the code could be used under other OS. Some modifications might be required(e.g., tell the complier the locations of dependencies), but I do not think anything drastic should be necessary.

This package depends on **PCL**.

✓ **PCL** is free available on `http://www.pointcloud.org`. Normally for **PCL**, *PCLConfig.cmake* and *PCLConfigVersion.cmake* are placed in **Share** folder automatically when install **PCL**.

4 Application Example

`tams_feature.cpp` shows an example of how to use the our class. It is quite similar to use Feature Class of PCL.

5 Discussion