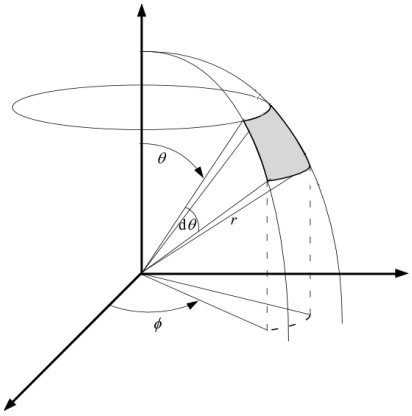

Figure 1: 3D patch when compute SEI

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 Spherical Harmonics

Fourier Analysis is extremely significant in signal processing and pattern recognition, since it decomposes the function into a linear combination of sinusoidal basis functions. In other words, the Fourier Analysis maps a function to a set of coefficients of basis functions. Admittedly, there are infinite ways to decompose the signals, and the reason why sinusoids are adopted is that they are eigenfunctions of the *Laplacian operator*, hence they maintain fidelity to most real systems. The basis functions of traditional Fourier Analysis are induced by the Laplacian operator in Cartesian coordinate system. By the same token, the Laplacian operator also has effective forms in other coordinate systems, e.g. polar and spherical coordinates. *The Spherical Fourier Transform is connected with Cartesian Fourier Transform by the Laplacian operator*.

The angular part of spherical Laplacian operator's eigenfunctions are named *spherical harmonic* functions $Y_l^m : S^2 \mapsto \mathbb{C}$, where S^2 stands for the unit 2D sphere and $\mathbb{C}$ symbolizes the set of complex number.

$$Y_l^m(\vartheta, \varphi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(cos\vartheta) e^{im\varphi} \tag{2}$$

where (ϑ, φ) are the spherical coordinates; P_l^m is an associated Legendre polynomial; l, m are integers, $l > 0, |m| < l$. The l is called the degree of spherical harmonics. For the l-th degree, there are $2l+1$ spherical harmonics basis functions indexed in the range of $-l \leq m \leq l$.

NOTE

- ✓ P_l^m is associated Legendre polynomial, and $\sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(x)$ is called normalized associated Legendre polynomial.
- ✓ $|Y_l^{-m}(\vartheta, \varphi_1)| = |Y_l^m(\vartheta, \varphi_2)|$ which means that the complex modulus of Y_l^m is unrelated to the azimuth angles.
- ✓ Some care must be taken in identifying the notational convention being used. In Y_l^m, ϑ is taken as the polar (colatitudinal) coordinate with $\vartheta \in [0, \pi]$, and φ as the azimuthal (longitudinal) coordinate with $\varphi \in [0, 2\pi)$.

Properties

In the following, we list some important properties of Y_l^m

1. $P_l^{-m}(x) = (-1)^m \frac{(l-m)!}{(l+m)!} P_l^m(x)$ Validated based on Mathematica.

2. Based on Property 1, we could deduce:
   $Y_l^{-m}(\vartheta,\varphi) = (-1)^m \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(cos\vartheta) e^{i(-m)\varphi}$
3. Based on SUPER Property 2 we could get:
   (a) $Y_l^{-m}(\vartheta,\varphi) = (-1)^m \overline{Y_l^m(\vartheta,\varphi)}$
   (b) $Real\{Y_l^{-m}(\vartheta,\varphi)\} = (-1)^m Real\{Y_l^m(\vartheta,\varphi)\}$
   (c) $Imaginary\{Y_l^{-m}(\vartheta,\varphi)\} = (-1)^{m+1} Imaginary\{Y_l^m(\vartheta,\varphi)\}$
   (d) $|Y_l^{-m}(\vartheta,\varphi)| = |Y_l^m(\vartheta,\varphi)|$

Let $L^2(S^2)$ denote the space of square integrate functions defined on S^2. The spherical harmonics of degree l span a $(2l+1)$ dimensional subspace of $L^2(S^2)$. Spherical harmonics of different degrees are orthogonal to each other. Furthermore, the spherical harmonic functions provide an complete orthonormal basis for $L^2(S^2)$. In other words, any function $f(\vartheta,\varphi) \in L^2(S^2)$ could be expanded as a linear combination of spherical harmonics:

$$f(\vartheta,\varphi) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} f_l^m Y_l^m(\vartheta,\varphi) \quad (3)$$

$$f_l^m = \int_{S^2} f(\omega)\overline{Y_l^m(\omega)}d\omega \quad (4)$$

where the overline stands for the complex conjugate. Equation (3) is named Spherical Fourier expansion or inverse Spherical Fourier Transform, and f_l^m is commonly called the spherical harmonic coefficients of $f(\vartheta,\varphi)$. In Cartesian Fourier Transform, the translation of signal does not change the magnitude of Fourier coefficients. Combine with equation (2), we could obtain that the change of φ in function $f(\vartheta,\varphi)$ does NOT change the magnitude of Spherical Harmonic Coefficients.

For function with bandlimit B, the number of coefficients is B^2

1.4 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 its SHC.

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 setBandwidth (size_t tams_bandwidth_);
inline void setEntropyBinDim (size_t tams_entropy_bin_);
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

```
// I/O to private attributes
    inline void setSEIAzimuthDim (size_t tams_sei_azimutch_dim_);
    inline void setSEIPolarDim (size_t tams_sei_polar_dim_);
```

```
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_,
       tams_bandwidth_;
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 SEI of the neighbours

C. compute the Spherical Harmonics Coefficients (SHC) of SEI

D. compute the magnitude of SHC of SEI

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.

✗ lib1
subdirectory contains the libraries used for computing the Spherical Harmonics Coefficients of functions defined on sphere.

✗ `tams_s2_semi_memo_for.h`
Declaration of function `tams_s2_semi_memo_for`, which is used to compute the forward spherical Fourier transform

```
tams_s2_semi_memo_for(TAMS_sei_real,
                      tams_bandwidth_,
                      TAMS_sh_real,
                      TAMS_sh_imag);
```

✗ `tams_s2_semi_memo_for.cpp`
Implementation of function `tams_s2_semi_memo_for`.

✗ `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** and **FFTW3**.

✓ **FFTW3** is free available on `http://www.fftw.org`. You could tell the compiler the location of **FFTW3** by copying the *FindFFTW3.cmake* to the **Modules** folder of cmake, if you want. Or you could tell the compiler the location of **FFTW3** by hand.

✓ **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**.

For example, in our laptop (Ubuntu 12.04/14.04, cmake-2.8), **Modules** folder of cmake is under */usr/share/cmake-2.8/Modules*

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