PCL :: Features

Michael Dixon and Radu B. Rusu

July 1, 2011

pointcloudlibrary

Outline

1. Introduction
2. Feature Estimation
3. Keypoint Detection
4. Keypoints + Features

pointcloudlibrary

Outline

1. Introduction
2. Feature Estimation
3. Keypoint Detection
4. Keypoints + Features

pointcloudlibrary

Local Features

What is a feature?

In vision/perception, the word “feature” can mean many different things. In PCL, “feature estimation” means:

- computing a feature vector based on each points local neighborhood
- or sometimes, computing a single feature vector for the whole cloud

pointcloudlibrary

Local Features

What is a feature?

Feature vectors can be anything from simple surface normals to the complex feature descriptors need for registration or object detection.

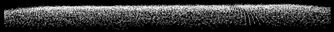
Today, we'll look at a couple of examples:

- Surface normal estimation (`NormalEstimation`)
- Point Feature Histogram estimation (`PFHEstimation`)

Local Features (1-2/5)

Surface Normal Estimation. Theoretical Aspects: Basic Ingredients

- Given a point cloud with x,y,z 3D point coordinates

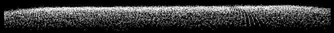


pointcloudlibrary

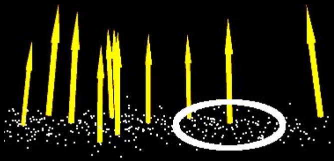
Local Features (1-2/5)

Surface Normal Estimation. Theoretical Aspects: Basic Ingredients

- Given a point cloud with x,y,z 3D point coordinates



- Select each point's *k*-nearest neighbors, fit a local plane, and compute the plane normal



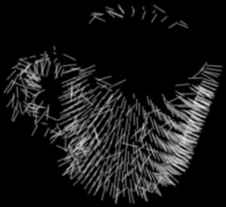
Local Features (3/5)

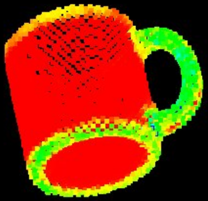
Surface Normal and Curvature Estimation

$$\boxed{C_j = \sum_{i=1}^{k} \xi_i \cdot (p_i - \overline{p_j})^T \cdot (p_i - \overline{p_j}),\ \overline{p} = \frac{1}{k} \cdot \sum_{i=1}^{k} p_i}$$

$$\xi_i = \begin{cases} e^{-\frac{d_i^2}{\mu^2}}, & p_i \ \mathit{outlier} \\ 1, & p_i \ \mathit{inlier} \end{cases}$$

$$\sigma_p = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$$





pointcloudlibrary

NormalEstimation example

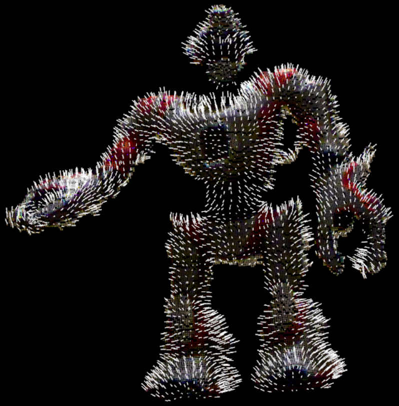
Estimating surface normals

```
void compute_surface_normals
  (pcl::PointCloud<pcl::PointXYZRGB>::Ptr &points, float normal_radius
   pcl::PointCloud<pcl::Normal>::Ptr &normals_out)
{
  pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> norm_est;

  // Use a FLANN-based KdTree to perform neighborhood searches
  norm_est.setSearchMethod
    (pcl::KdTreeFLANN<pcl::PointXYZRGB>::Ptr
      (new pcl::KdTreeFLANN<pcl::PointXYZRGB>));

  // Specify the size of the local neighborhood to use when
  // computing the surface normals
  norm_est.setRadiusSearch (normal_radius);

  // Set the input points
  norm_est.setInputCloud (points);

  // Set the search surface (i.e., the points that will be used
  // when search for the input points' neighbors)
  norm_est.setSearchSurface (points);

  // Estimate the surface normals and store the result in "normals_out
  norm_est.compute (*normals_out);
}
```

pointcloudlibrary

NormalEstimation results

```
$ ./features_demo ../data/robot1.pcd normals
```



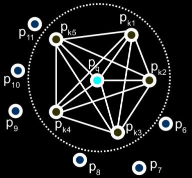
Now let's look at a more complex feature, like...

- Point Feature Histograms (PFH)

Point Feature Histograms

A 3D feature descriptor

- For every point pair $\langle (p_s, n_s); (p_t, n_t) \rangle$, let $u = n_s,\ v = (p_t - p_s) \times u,\ w = u \times v$

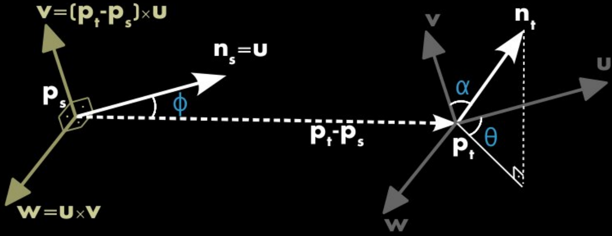


$$\left.\begin{aligned} f_0 &= \langle v, n_j \rangle \\ f_1 &= \langle u,\ p_j - p_i \rangle / ||p_j - p_i|| \\ f_2 &= ||p_j - p_i|| \\ f_3 &= atan(\langle w, n_j \rangle, \langle u, n_j \rangle) \end{aligned}\right\} \quad i_{hist} = \sum_{x=0}^{x \leq 3} \left\lfloor \frac{f_x \cdot d}{f_{x_{max}} - f_{x_{min}}} \right\rfloor \cdot d^x$$

pointcloudlibrary

Point Feature Histograms

A 3D feature descriptor



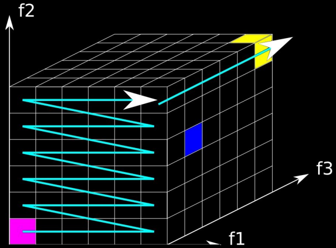
$$\left.\begin{aligned} f_0 &= \langle v, n_j \rangle \\ f_1 &= \langle u,\ p_j - p_i \rangle / ||p_j - p_i|| \\ f_2 &= ||p_j - p_i|| \\ f_3 &= atan(\langle w, n_j \rangle, \langle u, n_j \rangle) \end{aligned}\right\} \quad i_{hist} = \sum_{x=0}^{x \leq 3} \left\lfloor \frac{f_x \cdot d}{f_{x_{max}} - f_{x_{min}}} \right\rfloor \cdot d^x$$

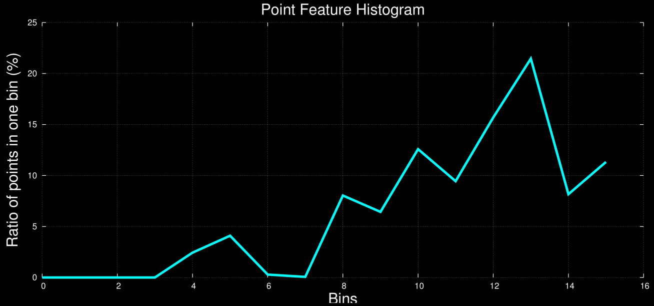
pointcloudlibrary

Point Feature Histograms

A 3D feature descriptor

- For every point pair $\langle(p_s, n_s); (p_t, n_t)\rangle$, let $u = n_s,\ v = (p_t - p_s) \times u,\ w = u \times v$

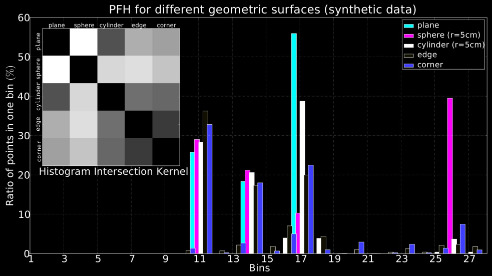


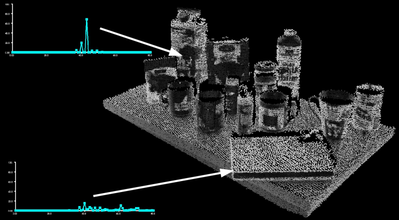


pointcloudlibrary

Point Feature Histograms

Points lying on different geometric primitives





pointcloudlibrary

PFHEstimation example

PFH is a more complex feature descriptor, but the code is very similar

```
using namespace pcl; // Let's save ourselves some typing

void compute_PFH_features
  (PointCloud<PointXYZRGB>::Ptr &points, PointCloud<Normal>::Ptr &norm
   float feature_radius, PointCloud<PFHSignature125>::Ptr &descriptors
{
  // Create a PFHEstimation object
  PFHEstimation<PointXYZRGB, Normal, PFHSignature125> pfh_est;

  // Set it to use a FLANN-based KdTree to perform its neighborhood se
  pfh_est.setSearchMethod (KdTreeFLANN<PointXYZRGB>::Ptr
    (new KdTreeFLANN<PointXYZRGB>));

  // Specify the radius of the PFH feature
  pfh_est.setRadiusSearch (feature_radius);

  // Set the input points and surface normals
  pfh_est.setInputCloud (points);
  pfh_est.setInputNormals (normals);

  // Compute the features
  pfh_est.compute (*descriptors_out);
}
```

pointcloudlibrary

PHFEstimation example

...actually, we'll get back to this one later. For now, let's move on to keypoints.

pointcloudlibrary

Outline

1. Introduction
2. Feature Estimation
3. Keypoint Detection
4. Keypoints + Features

pointcloudlibrary

Keypoints

What is a keypoint?

- A keypoint (also known as an “interest point”) is simply a point that has been identified as a relevant in some way.
- Whether any given point is considered a keypoint or not depends on the *keypoint detector* in question.

pointcloudlibrary

Keypoints

What is a keypoint?

There’s no strict definition for what constitutes a keypoint detector, but a good keypoint detector will find points which have the following properties:

- Sparseness: Typically, only a small subset of the points in the scene are keypoints
- Repeatiblity: If a point was determined to be a keypoint in one point cloud, a keypoint should also be found at the corresponding location in a similar point cloud. (Such points are often called "stable".)
- Distinctiveness: The area surrounding each keypoint should have a unique shape or appearance that can be captured by some feature descriptor

pointcloudlibrary

Keypoints

Why find keypoints?

What are the benefits of keypoints?

- Some features are expensive to compute, and it would be prohibitive to compute them at every point. Keypoints identify a small number of locations where computing feature descriptors is likely to be most effective.
- When searching for corresponding points, features computed at non-descriptive points will lead to ambiguous feature corespondences. By ignoring non-keypoints, one can reduce error when matching points.

pointcloudlibrary

Keypoints

Keypoint detection in PCL

There are a couple of keypoint detectors in PCL so far.

- NARFKeypoint
  - Requires range images
  - Bastian will talk more about this later
- SIFTKeypoint
  - A 3D adaptation of David Lowe's SIFT keypoint detector

Now let's look at an example of how to compute 3D SIFT keypoints

pointcloudlibrary

SIFTKeypoints example

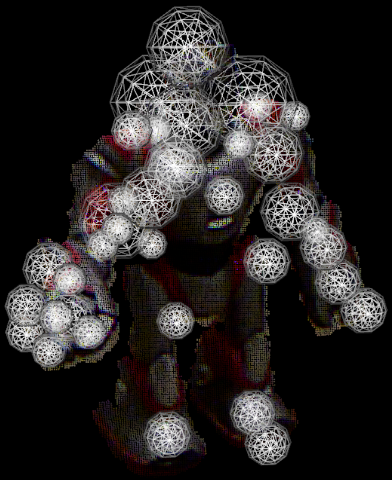
Computing keypoints on a cloud of PointXYZRGB points

```
using namespace pcl;

void detect_keypoints
  (PointCloud<PointXYZRGB>::Ptr &points, float min_scale,
   int nr_octaves, int nr_scales_per_octave, float min_contrast,
   PointCloud<PointWithScale>::Ptr &keypoints_out)
{
  SIFTKeypoint<PointXYZRGB, PointWithScale> sift_detect;

  // Use a FLANN-based KdTree to perform neighborhood searches
  sift_detect.setSearchMethod
    (KdTreeFLANN<PointXYZRGB>::Ptr
      (new KdTreeFLANN<PointXYZRGB>));

  // Set the detection parameters
  sift_detect.setScales (min_scale, nr_octaves, nr_scales_per_octave);
  sift_detect.setMinimumContrast (min_contrast);

  // Set the input
  sift_detect.setInputCloud (points);

  // Detect the keypoints and store them in "keypoints_out"
  sift_detect.compute (*keypoints_out);
}
```

pointcloudlibrary

SIFTKeypoint results

```
$ ./features_demo ../data/robot1.pcd keypoints
```



pointcloudlibrary

Outline

1. Introduction
2. Feature Estimation
3. Keypoint Detection
4. Keypoints + Features

pointcloudlibrary

Keypoints + features

Keypoints and feature descriptors go hand-in-hand.

Let's look at how to compute features only at a given set of keypoints.

pointcloudlibrary

Keypoints + features (1)

Computing feature descriptors for a set of keypoints

```
using namespace pcl;

void
compute_PFH_features_at_keypoints
  (PointCloud<PointXYZRGB>::Ptr &points,
   PointCloud<Normal>::Ptr &normals,
   PointCloud<PointWithScale>::Ptr &keypoints, float feature_radius,
   PointCloud<PFHSignature125>::Ptr &descriptors_out)
{
  // Create a PFHEstimation object
  PFHEstimation<PointXYZRGB, Normal, PFHSignature125> pfh_est;

  // Set it to use a FLANN-based KdTree to perform its
  // neighborhood searches
  pfh_est.setSearchMethod
    (KdTreeFLANN<PointXYZRGB>::Ptr (new KdTreeFLANN<PointXYZRGB>));

  // Specify the radius of the PFH feature
  pfh_est.setRadiusSearch (feature_radius);

  // continued on the next slide ...
```

pointcloudlibrary

Keypoints + features (2)

```
  // ... continued from the previous slide

  // Convert the keypoints cloud from PointWithScale to PointXYZRGB
  // so that it will be compatible with our original point cloud
  PointCloud<PointXYZRGB>::Ptr keypoints_xyzrgb
    (new PointCloud<PointXYZRGB>);
  pcl::copyPointCloud (*keypoints, *keypoints_xyzrgb);

  // Use all of the points for analyzing the local structure of the cl
  pfh_est.setSearchSurface (points);
  pfh_est.setInputNormals (normals);

  // But only compute features at the keypoints
  pfh_est.setInputCloud (keypoints_xyzrgb);

  // Compute the features
  pfh_est.compute (*descriptors_out);
}
```

pointcloudlibrary

What can you use this for?

Here's an example of how you could use features and keypoints to find corresponding points between two point clouds.

```
int correspondences_demo (const char * filename_base)
{
  // Okay, this is a bit longer than our other examples,
  // so let's read the code in another window.
  // (See "features_demo.cpp", lines 308-368 )
}
```

pointcloudlibrary

Correspondence results

```
$ ./features_demo ../data/robot
correspondences
```