/\*\*

\* **\brief** NARF: Normal Aligned Radial Feature

\* **\param** rangeImg

\* **\param** keyPts

\*/

void pclFeaturePt\_NARF(pcl::RangeImage& rangeImg, pcl::PointCloud<pcl::PointXYZ>::Ptr keyPts, pcl::PointCloud<int>::Ptr keypoint\_indices)

{

pcl::RangeImageBorderExtractor rangeImgBorderExtra;

pcl::NarfKeypoint narfKeyptDete(&rangeImgBorderExtra); //NARF首先需要探测深度图像的边缘

narfKeyptDete.setRangeImage(&rangeImg);

narfKeyptDete.getParameters().support\_size = 5;

//narf\_keypoint\_detector.getParameters ().add\_points\_on\_straight\_edges = true;

//narf\_keypoint\_detector.getParameters ().distance\_for\_additional\_points = 0.5;

narfKeyptDete.compute(\*keypoint\_indices);

std::cout << "Found " << keypoint\_indices->points.size() << " key points.\n";

//在距离图像显示组件内显示关键点

//for (size\_ti=0; i<keypoint\_indices.points.size (); ++i)

//range\_image\_widget.markPoint (keypoint\_indices.points[i]%range\_image.width,

//keypoint\_indices.points[i]/range\_image.width);

//关键点点云

keyPts->points.resize(keypoint\_indices->points.size());

for (size\_t i = 0; i < keypoint\_indices->points.size(); ++i)

{

keyPts->points[i].getVector3fMap() = rangeImg.points[keypoint\_indices->points[i]].getVector3fMap();

}

}

void showPointCloudVisual(cv::Mat originBGRMap, cv::Mat disparityMap, std::string cameraPairParaPath)

{

if (originBGRMap.empty())

{

return;

}

if (disparityMap.empty())

{

return;

}

if (cameraPairParaPath.empty())

{

return;

}

disparityMap.convertTo(disparityMap, CV\_32F);

originBGRMap.convertTo(originBGRMap, CV\_32F);

double u0, v0, fx, fy, Tx, doffs;

Mat cameraInnerPara\_left, cameraInnerPara\_right;

Mat cameraDistPara\_left, cameraDistPara\_right;

Mat matrixR, matrixT;

FileStorage fn(cameraPairParaPath, FileStorage::READ);

fn["Left\_CameraInnerPara"] >> cameraInnerPara\_left;

fn["Left\_CameraDistPara"] >> cameraDistPara\_left;

fn["Right\_CameraInnerPara"] >> cameraInnerPara\_right;

fn["Right\_CameraDistPara"] >> cameraDistPara\_right;

fn["R2L\_Rotation\_Matrix"] >> matrixR;

fn["R2L\_Translate\_Matrix"] >> matrixT;

fn.release();

u0 = cameraInnerPara\_left.at<double>(0, 2);

v0 = cameraInnerPara\_left.at<double>(1, 2);

fx = cameraInnerPara\_left.at<double>(0, 0);

fy = cameraInnerPara\_left.at<double>(1, 1);

Tx = abs(matrixT.at<double>(0, 0));

doffs = 1;

pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZRGB>);

int rowNum = originBGRMap.rows;

int colNum = originBGRMap.cols;

cloud->height = rowNum;

cloud->width = colNum;

cloud->points.resize(cloud->width \* cloud->height);

Mat rangeImage(originBGRMap.size(), CV\_32FC1);

pcl::RangeImagePlanar pclRangeImg;

pclRangeImg.setDisparityImage((float\*)disparityMap.data, colNum, rowNum, fx, Tx);

//pcl::visualization::RangeImageVisualizer range\_image\_widget("Range Image");

//range\_image\_widget.showRangeImage(\*pclRangeImg);

//pcl::PolygonMesh triangles;

//pclMesh\_OrganizedFastMesh(pclRangeImg, triangles);

//boost::shared\_ptr<pcl::visualization::PCLVisualizer> viewer\_mesh(new pcl::visualization::PCLVisualizer("PCL Mesh"));

//viewer\_mesh->setBackgroundColor(0.5, 0.5, 0.5);

//viewer\_mesh->addPolygonMesh(triangles, "tin");

//viewer\_mesh->addCoordinateSystem();

//while (!range\_image\_widget.wasStopped() && !viewer\_mesh->wasStopped())

//{

// range\_image\_widget.spinOnce();

// boost::this\_thread::sleep(boost::posix\_time::microseconds(100));

// viewer\_mesh->spinOnce();

//}

pcl::PointCloud<pcl::PointXYZ>::Ptr keyPt\_NARF(new pcl::PointCloud<pcl::PointXYZ>);

pcl::RangeImage::Ptr rangeImg\_(&pclRangeImg);

pcl::PointCloud<int>::Ptr keypoint\_indices(new pcl::PointCloud<int>);

pclFeaturePt\_NARF(\*rangeImg\_, keyPt\_NARF, keypoint\_indices);

for (unsigned int u = 0; u < rowNum; u++)

{

unsigned int num\_rows = u \* colNum;

for (unsigned int v = 0; v < colNum; v++)

{

unsigned int num = num\_rows + v;

double Xw = 0, Yw = 0, Zw = 0;

Zw = fx \* Tx / ((double)disparityMap.at<float>(u, v) + doffs);

Xw = (v + 1 - u0) \* Zw / fx;

Yw = (u + 1 - v0) \* Zw / fy;

cloud->points[num].b = originBGRMap.at<Vec3f>(u, v)[0];

cloud->points[num].g = originBGRMap.at<Vec3f>(u, v)[1];

cloud->points[num].r = originBGRMap.at<Vec3f>(u, v)[2];

cloud->points[num].x = Xw / 100;

cloud->points[num].y = Yw / 100;

cloud->points[num].z = Zw / 100;

originBGRMap.at<float>(u, v) = Zw;

}

}

pcl::io::savePCDFileASCII("res.pcd", \*cloud);

}