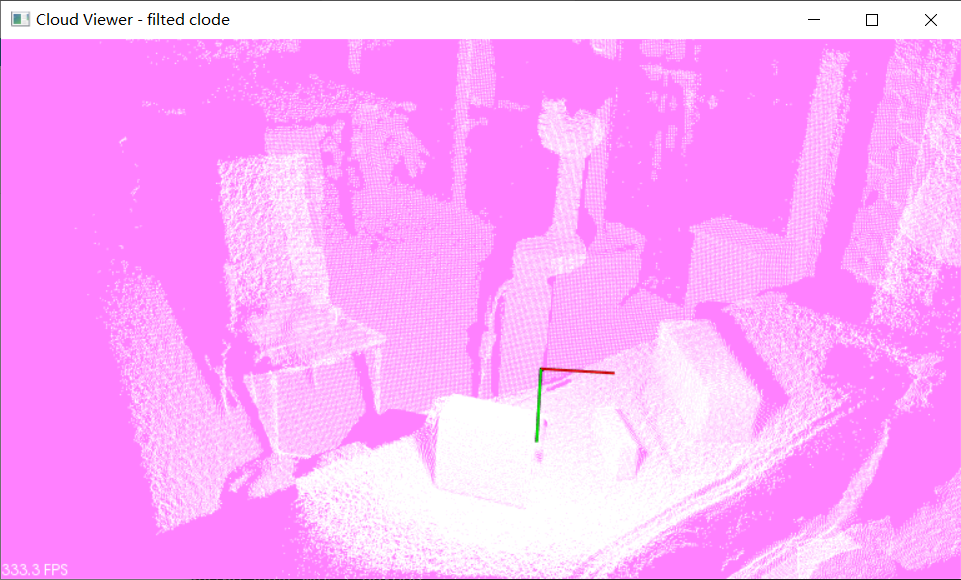
作业思路：

自己用VC2015结合写了一个代码，但是没有完成。通过代码看图，可以看出：

Pc0~ pc10： 传送带上没有箱子

Pc15 ~pc505: 每隔5一张点云图



1. 先利用前10张图求传送带背景，求出旋转平移矩阵H，经过旋转平移矩阵，传送带边缘与y轴重合。
   1. 人为限定一个空间范围 （x1 , y1 ,z1）- （x2，y2, z2） 区间，该区间内应只含有传送带，传送带平面为一个和X,Y，Z轴倾斜的平面
   2. 对pc0 ~pc10的图进行裁剪，只保留人为限定空间范围内部的点。
   3. Pc0~pc10每张图调用离群点滤波函数去除离群点。
   4. 对滤波后的pc0~pc10点云进行合并得到pc0sum10
   5. 通过边沿检测算法找到传送带边缘并拟合传送带边缘，得到传送带边缘，线方程。
   6. 通过边缘线方程算出一个旋转平移矩阵，该旋转平移矩阵H，可以让传送带边缘与y轴重合。
   7. 对pc0sum10中所有点乘以旋转平移矩阵H
   8. 对旋转平移后的矩阵，通过RANSAC算法拟合平面S

H=[ R t

0 1]

R 为3x3旋转矩阵， t为3x1平移矩阵

1. 对pc15~ pc405 点云中每个点乘以H矩阵,得到pcH15~pcH405
2. 再对旋转平移后的点云pcH15~pcH405，人为设定一个空间范围（x3,y3,z3）~(x4,y4,z4)，该空间范围的选择应保证对于所有旋转平移后的点云图，空间范围内只有传送带和箱子。
3. 对旋转平移后的pcH15~pcH405用空间范围 (x3,y3,z3)~(x4,y4,z4)进行裁剪，只保留限定范围内的点云，得到pchl15~pchl405。
4. 对pchl15~pchl405进行离群点滤波。
5. 去掉pchl15~pchl405中在S平面（传送带平面）上的点（或去掉距离S平面较近的点）。
6. 去除了出传送带背景的点云，用KNN对点云图进行分割
7. 对分割的点云块进行拟合，从而算出一个个分离的箱子的尺寸。

该算法针对生产线运作设计，而非仅仅是为了解一个题目，做好了旋转平移，限定范围，求取传送带平面后，可以随着传送带的运动每张图都给出在传送带上面箱子的尺寸。对于半个箱子也能有效算出半个箱子的尺寸。当然针对半个箱子如果箱子放置与传送带平行那么不好区分出来，但是如果箱子与传送带放置有一定角度，还可以加入更为复杂的算法形状判断算法，来判断出箱子没有完全送上传送带。

关于算法中的两次人为选取范围：其实生产线一旦固定，3D相机相对生产线的位置一旦固定，箱子的种类尺寸范围已知，那么可以人为限定计算范围，而且范围一旦设定，不需要经常更改。可以提前计算好，一直使用，而不需要每过一个箱子都计算一次。

// Dep\_to\_PCL\_VC14.cpp : 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include<iostream>//标准C++库中的输入输出类相关头文件。

#include <fstream>

#include <vector>

#include <string>

#include "\_DepthToPCL.h" //自己写的转换类

#include<pcl/visualization/cloud\_viewer.h>

#include<pcl/io/io.h>

#include<pcl/io/pcd\_io.h>//pcd 读写类相关的头文件。

#include<pcl/io/ply\_io.h>

#include<pcl/point\_types.h> //PCL中支持的点类型头文件。

#include <iostream>

#include <boost/thread/thread.hpp>

#include <pcl/common/common\_headers.h>

#include <pcl/range\_image/range\_image.h> //关于深度图像的头文件

#include <pcl/visualization/range\_image\_visualizer.h> //深度图可视化的头文件

#include <pcl/visualization/pcl\_visualizer.h> //PCL可视化的头文件

#include <pcl/console/parse.h>

#include <pcl/filters/statistical\_outlier\_removal.h>

#include "\_CreatePlyFile.h"

int user\_data;

using *std*::*cout*;

void viewerOneOff(*pcl*::*visualization*::*PCLVisualizer*& viewer) {

viewer.*setBackgroundColor*(1.0, 0.5, 1.0); //设置背景颜色

}

int main()

{

//图像数据内存申请

int img\_w, img\_h;

img\_w = 640\*480;

img\_h = 1;

float \*m\_p\_pcl\_x = new float[img\_w\*img\_h];

float \*m\_p\_pcl\_y = new float[img\_w\*img\_h];

float \*m\_p\_pcl\_z = new float[img\_w\*img\_h];

//读取点云数据

char str\_fn[255];

*FILE* \*fp;

int j, i, a, b;

int k;

int n = 0;

int m\_np = 0;

float z\_max = -380000.0;

float z\_min = 99900000000.0;

// pcl::PointCloud<pcl::PointXYZRGB> cloud;

*pcl*::*PointCloud*<*pcl*::*PointXYZ*>::*Ptr* cloud(new *pcl*::*PointCloud*<*pcl*::*PointXYZ*>);

//对点云进行滤波

*pcl*::*PointCloud*<*pcl*::*PointXYZ*>::*Ptr* cloud\_filtered(new *pcl*::*PointCloud*<*pcl*::*PointXYZ*>);

// 创建滤波器，对每个点分析的临近点的个数设置为10 ，并将标准差的倍数设置为1 这意味着如果一

//个点的距离超出了平均距离一个标准差以上，则该点被标记为离群点，并将它移除，存储起来

*pcl*::*StatisticalOutlierRemoval*<*pcl*::*PointXYZ*> sor; //创建滤波器对象

//数据显示

*pcl*::*visualization*::*CloudViewer* v2("Cloud Viewer - filted clode"); //创建viewer对象

int m\_nfile = -1;

while (m\_nfile < 505)

{

if (m\_nfile<10)

{

m\_nfile += 1;

}

else

{

m\_nfile += 5;

}

*sprintf*(str\_fn, "pc%d.csv", m\_nfile);

fp = *fopen*(str\_fn,"r");

*printf*(str\_fn);

if (m\_nfile<2)

{

*getchar*();

}

// getchar();

n = 0;

m\_np = 0;

while (!*feof*(fp))

{

*fscanf*(fp, "%f,%f,%f,", m\_p\_pcl\_x +n, m\_p\_pcl\_y+n, m\_p\_pcl\_z+n);

n++;

m\_np++;

// printf("pix%d=%f\n", n, \*(m\_p\_src + n));

// getchar();

}

img\_w = m\_np;

img\_h = 1;

*fclose*(fp);

z\_max= -380000.0;

z\_min = 99900000000.0;

n = 0;

for (n=0;n<img\_w\*img\_h;n++)

{

if (z\_max<\*(m\_p\_pcl\_z + n))

{

z\_max = \*(m\_p\_pcl\_z + n);

}

if (z\_min > \*(m\_p\_pcl\_z + n))

{

z\_min = \*(m\_p\_pcl\_z + n);

}

}

*printf*("z\_max=%f\n",z\_max);

*printf*("z\_min=%f\n", z\_min);

//显示点云

// cloud->width = img\_w\*img\_h;

// cloud->height = 1;

cloud->*empty*();

cloud->*width* = img\_w;

cloud->*height* = img\_h;

cloud->*is\_dense* = false;

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

n = 0;

for (n = 0; n < img\_w\*img\_h; n++)

{

cloud->*points*[n].*x* = m\_p\_pcl\_x[n];

cloud->*points*[n].*y* = m\_p\_pcl\_y[n];

cloud->*points*[n].*z* = m\_p\_pcl\_z[n];

}

*sprintf*(str\_fn, "pc%d.ply", m\_nfile);

\_PlyFileCreator c1;

c1.CreatePlyFile(str\_fn, img\_w, img\_h, m\_p\_pcl\_x, m\_p\_pcl\_y, m\_p\_pcl\_z, 0, *NULL*, *NULL*, *NULL*, 0);

// pcl::visualization::CloudViewer viewer("Cloud Viewer"); //创建viewer对象

// viewer.showCloud(cloud, "cloud");

// viewer.runOnVisualizationThreadOnce(viewerOneOff);

/\*

//对点云进行滤波

// pcl::PointCloud<pcl::PointXYZ>::Ptr cloud\_filtered(new pcl::PointCloud<pcl::PointXYZ>);

// 创建滤波器，对每个点分析的临近点的个数设置为10 ，并将标准差的倍数设置为1 这意味着如果一

//个点的距离超出了平均距离一个标准差以上，则该点被标记为离群点，并将它移除，存储起来

// pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor; //创建滤波器对象

sor.setInputCloud(cloud); //设置待滤波的点云

sor.setMeanK(10); //设置在进行统计时考虑查询点临近点数

sor.setStddevMulThresh(1.0); //设置判断是否为离群点的阀值

sor.filter(\*cloud\_filtered); //存储

printf("width=%d highth=%d", cloud\_filtered->width, cloud\_filtered->height);

//pcl数据提取

int fw, fh;

fw = cloud\_filtered->width;

fh = cloud\_filtered->height;

float\* m\_p\_pcl\_xf = new float[fw\*fh];

float\* m\_p\_pcl\_yf = new float[fw\*fh];

float\* m\_p\_pcl\_zf = new float[fw\*fh];

for (n = 0; n < fw\*fh; n++)

{

m\_p\_pcl\_xf[n] = cloud\_filtered->points[n].x;

m\_p\_pcl\_yf[n] = cloud\_filtered->points[n].y;

m\_p\_pcl\_zf[n] = cloud\_filtered->points[n].z;

}

//数据存ply

\_PlyFileCreator c2;

c2.CreatePlyFile("pc\_out.ply", fw, fh, m\_p\_pcl\_xf, m\_p\_pcl\_yf, m\_p\_pcl\_zf, 0, NULL, NULL, NULL, 0);

//存储point cloud csv

fp = fopen("pc\_out.csv", "w");

n = 0;

for (n = 0; n < fw\*fh; n++)

{

fprintf(fp, "%f,%f,%f\n", m\_p\_pcl\_xf[n], m\_p\_pcl\_yf[n], m\_p\_pcl\_zf[n]);

// printf("%f,%f,%f\n", m\_p\_pcl\_x[n], m\_p\_pcl\_y[n], m\_p\_pcl\_z[n]);

}

fclose(fp);

\*/

//数据显示

// pcl::visualization::CloudViewer v2("Cloud Viewer - filted clode"); //创建viewer对象

v2.*showCloud*(cloud, "cloud");

v2.*runOnVisualizationThreadOnce*(viewerOneOff);

// system("pause");

}

*system*("pause");

//删除数据

return 0;

}

/\*

#include <iostream>

#include <pcl/io/pcd\_io.h>

#include <pcl/point\_types.h>

#include <pcl/filters/statistical\_outlier\_removal.h>

int

main(int argc, char\*\* argv)

{

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

pcl::PointCloud<pcl::PointXYZ>::Ptr cloud\_filtered(new pcl::PointCloud<pcl::PointXYZ>);

// 定义读取对象

pcl::PCDReader reader;

// 读取点云文件

reader.read<pcl::PointXYZ>("table\_scene\_lms400.pcd", \*cloud);

std::cerr << "Cloud before filtering: " << std::endl;

std::cerr << \*cloud << std::endl;

// 创建滤波器，对每个点分析的临近点的个数设置为50 ，并将标准差的倍数设置为1 这意味着如果一

//个点的距离超出了平均距离一个标准差以上，则该点被标记为离群点，并将它移除，存储起来

pcl::StatisticalOutlierRemoval<pcl::PointXYZ> sor; //创建滤波器对象

sor.setInputCloud(cloud); //设置待滤波的点云

sor.setMeanK(50); //设置在进行统计时考虑查询点临近点数

sor.setStddevMulThresh(1.0); //设置判断是否为离群点的阀值

sor.filter(\*cloud\_filtered); //存储

std::cerr << "Cloud after filtering: " << std::endl;

std::cerr << \*cloud\_filtered << std::endl;

pcl::PCDWriter writer;

writer.write<pcl::PointXYZ>("table\_scene\_lms400\_inliers.pcd", \*cloud\_filtered, false);

sor.setNegative(true);

sor.filter(\*cloud\_filtered);

writer.write<pcl::PointXYZ>("table\_scene\_lms400\_outliers.pcd", \*cloud\_filtered, false);

return (0);

}

\*/