AIV 과제 전형
-Project-
박은석

사용 툴

• visual studio 2022

사용 언어 및 라이브러리

• C++
• PCL Library - 1.12.0 ver
• Eigen Library
• OpenCV Library - 4.6.0 ver

문제점

1. Window환경에서 PCL Library설치를 완료하였으나, pcl visualization이 되지 않는 문제 발생
   - 빌드 오류 발생
   - pcl visualization 함수를 include 하지 않은 상태로 알고리즘 구현

2. ICP 알고리즘을 이용하여 registration을 수행하였으나, 높은 계산량으로 인해, 대응점을 매칭하지 못하는 문제점 발생
   - voxelization을 수행
     ✓ 임계값이 0.9미만일 경우 포인트가 voxelization되지 않음
     ✓ 임계값이 0.9이상일 경우 포인트가 너무 많이 voxelization되어 대응점을 찾지 못함

알고리즘 수행 순서

1. 데이터 load

• master와 real 데이터를 opencv라이브러리를 통해 입력받아 point cloud로 저장

2. point cloud로 저장한 데이터에 대해 Iterative Closest Point(ICP) 수행 및 merge 수행 (${}^{O}T_{r} = {}^{O}T_{I} \times {}^{I}T_{r}$)

• Origin - master

• real - Origin

PointCloudICP.cpp

42 ~ 140행 : bottom, top, side 방향에 대해 Origin-Ideal ${}^{O}T_{I}$ 수행

143 ~ 171행 : real 데이터를 입력 받아 Origin-real ${}^{O}T_{r}$ 수행 및 merge수행

● 제공 받은 master데이터를 load한 후 point cloud로 저장

MasterFile.cpp

```
// Master bottom파일 로드
std::string filePath_bottom = "C://my_job//aiv_project//3D_치수_과제_영상//MASTER_BOTTOM.tiff";
// file read
idealDepthMapBottom_ = cv::imread(filePath_bottom, cv::IMREAD_UNCHANGED);
dataFullSizeBottom_ = idealDepthMapBottom_.total();
auto imgPtr_bottom_ = idealDepthMapBottom_.ptr<cv::Vec3f>(0);

for (auto i = 0; i < dataFullSizeBottom_; ++i) // bottom파일을 point cloud로 저장
{
    pcl::PointXYZ pt_;
    const auto& zyx = imgPtr_bottom_[i];
    pt_.x = zyx[2];
    pt_.y = zyx[1];
    pt_.z = zyx[0];

    idealPointBottom_.push_back(pt_);
}
//std::cout << idealPointBottom_.size() << std::endl;
```

```
// master top파일 로드
std::string filePath_top_ = "C://my_job//aiv_project//3D_치수_과제_영상//MASTER_TOP.tiff";
idealDepthMapTop_ = cv::imread(filePath_top_, cv::IMREAD_UNCHANGED);
dataFullSizeTop_ = idealDepthMapTop_.total();
auto imgPtr_top_ = idealDepthMapTop_.ptr<cv::Vec3f>(0);

for (int j = 0; j < dataFullSizeTop_; ++j) // top파일을 point cloud로 저장
{
    pcl::PointXYZ pt_;
    const auto& zyx = imgPtr_top_[j];
    pt_.x = zyx[2];
    pt_.y = zyx[1];
    pt_.z = zyx[0];

    idealPointTop_.push_back(pt_);
}
```

```
// master side파일 로드
std::string filePath_side_ = "C://my_job//aiv_project//3D_치수_과제_영상//MASTER_SIDE.tiff";
idealDepthMapSide_ = cv::imread(filePath_side_, cv::IMREAD_UNCHANGED);
dataFullSizeSide_ = idealDepthMapSide_.total();
auto imgPtr_side_ = idealDepthMapSide_.ptr<cv::Vec3f>(0);

for (int k = 0; k < dataFullSizeSide_; ++k) // side파일을 point cloud로 저장
{
    pcl::PointXYZ pt_;
    const auto& zyx = imgPtr_side_[k];
    pt_.x = zyx[2];
    pt_.y = zyx[1];
    pt_.z = zyx[0];

    idealPointSide_.push_back(pt_);
}
```

● 제공 받은 real 데이터를 load한 후 point cloud로 저장

RealFile.cpp

```cpp
std::string filePath_bottom = "C://my_job//aiv_project//3D_치수_과제_영상//REAL_BOTTOM.tiff";
realDepthMapBottom_ = cv::imread(filePath_bottom, cv::IMREAD_UNCHANGED);
auto width_bottom = realDepthMapBottom_.cols, height_bottom = realDepthMapBottom_.rows;
auto imgPtr_bottom_ = realDepthMapBottom_.ptr<float>(0);

// vector
std::vector<cv::Point3f> realBottomPC(width_bottom * height_bottom);
auto curldx_bottom = 0;
for (int j = 0; j < height_bottom; ++j)
{
    auto step = width_bottom * j;
    for (int i = 0; i < width_bottom; ++i)
    {
        if (NULLVALUE != imgPtr_bottom_[step + i])
        {
            realBottomPC[curldx_bottom].x = SCALEX * i;
            realBottomPC[curldx_bottom].y = SCALEY * j;
            realBottomPC[curldx_bottom].z = -imgPtr_bottom_[step + i];
            curldx_bottom++;
        }
    }
}

vectorBottom2pointCloud(realBottomPC); // PDF파일에서 제공한 파일을 바탕으로 데이터를 입력받아 point cloud로 저장
```

```cpp
std::string filePath_top = "C://my_job//aiv_project//3D_치수_과제_영상//REAL_TOP.tiff";
realDepthMapTop_ = cv::imread(filePath_top, cv::IMREAD_UNCHANGED);
auto width_top = realDepthMapTop_.cols, height_top = realDepthMapTop_.rows;
auto imgPtr_top_ = realDepthMapTop_.ptr<float>(0);

//vector
std::vector<cv::Point3f> realTopPC(width_top * height_top);
auto curldx_top = 0;
for (int j = 0; j < height_top ; ++j)
{
    auto step = width_top * j;
    for (int i = 0; i < width_top; ++i)
    {
        if (NULLVALUE != imgPtr_top_[step + i])
        {
            realTopPC[curldx_top].x = SCALEX * i;
            realTopPC[curldx_top].y = SCALEY * j;
            realTopPC[curldx_top].z = -imgPtr_top_[step + i];
        }
    }
}
vectorTop2pointCloud(realTopPC); // PDF파일에서 제공한 파일을 바탕으로 데이터를 입력받아 point cloud로 저장
```

```cpp
std::string filePath_side = "C://my_job//aiv_project//3D_치수_과제_영상//REAL_SIDE.tiff";
realDepthMapSide_ = cv::imread(filePath_side, cv::IMREAD_UNCHANGED);
auto width_side = realDepthMapSide_.cols, height_side = realDepthMapSide_.rows;
auto imgPtr_side_ = realDepthMapSide_.ptr<float>(0);

//vector
std::vector<cv::Point3f> realSidePC(width_side * height_side);
auto curldx_side = 0;
for (int j = 0; j < height_side; ++j)
{
    auto step = width_side * j;
    for (int i = 0; i < width_side; ++i)
    {
        if (NULLVALUE != imgPtr_side_[step + i])
        {
            realSidePC[curldx_side].x = SCALEX * i;
            realSidePC[curldx_side].y = SCALEY * j;
            realSidePC[curldx_side].z = -imgPtr_side_[step + i];
        }
    }
}
vectorSide2pointCloud(realSidePC); // PDF파일에서 제공한 파일을 바탕으로 데이터를 입력받아 point cloud로 저장
```

● point cloud로 저장한 데이터들을 이용해 ICP수행

PointCloudICP.cpp

1. 제공받은 TF_matrix를 이용해 Origin frame과 ideal frame간의 변환을 수행

```
*input_bottom = icp_master_.returnIdealPointBottom();
voxel_bottom_.setInputCloud(input_bottom);
voxel_bottom_.filter(*input_filtered_bottom);


//std::cout << input_bottom->size() << std::endl;
pcl::transformPointCloud(*input_filtered_bottom, *output_bottom, tf_master_bottom_);

// icp bottom
pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp_bottom_;
icp_bottom_.setMaxCorrespondenceDistance(1.0);
icp_bottom_.setTransformationEpsilon(0.001);
icp_bottom_.setMaximumIterations(1000);

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

// registration bottom -> Origin - master
icp_bottom_.setInputSource(input_filtered_bottom);
icp_bottom_.setInputTarget(output_bottom);
icp_bottom_.align(*align_bottom);

// set outputs bottom
input2output_bottom = icp_bottom_.getFinalTransformation();
 *input_top = icp_master_.returnIdealPointTop();
 voxel_top_.setInputCloud(input_top);
 voxel_top_.filter(*input_filtered_top);

 pcl::transformPointCloud(*input_filtered_top, *output_top, tf_master_top_);


 // icp top
 pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp_top_;
 icp_top_.setMaxCorrespondenceDistance(0.1);
 icp_top_.setTransformationEpsilon(0.001);
 icp_top_.setMaximumIterations(10);

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

 // registration top  -> Origin - master
 icp_top_.setInputSource(input_filtered_top);
 icp_top_.setInputTarget(output_top);
 icp_top_.align(*align_top);

 // set outputs top
 input2output_top = icp_top_.getFinalTransformation();
```

```
*input_side = icp_master_.returnIdealPointSide();
voxel_side_.setInputCloud(input_side);
voxel_side_.filter(*input_filtered_side);

pcl::transformPointCloud(*input_filtered_side, *output_side, tf_master_side_);


// icp side
pcl::IterativeClosestPoint<pcl::PointXYZ, pcl::PointXYZ> icp_side_;
icp_side_.setMaxCorrespondenceDistance(0.1);
icp_side_.setTransformationEpsilon(0.001);
icp_side_.setMaximumIterations(10);

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

// registration side  -> Origin - master
icp_side_.setInputSource(input_filtered_side);
icp_side_.setInputTarget(output_side);
icp_side_.align(*align_side);

// set outputs side
input2output_side = icp_side_.getFinalTransformation();
```

2. 제공받은 real 데이터를 load하여 real-Origin변환을 수행

```
// real data 가져오기
pcl::PointCloud<pcl::PointXYZ>::Ptr real_top(new pcl::PointCloud<pcl::PointXYZ>);
*real_top= icp_real_.returnRealPointTop();
pcl::PointCloud<pcl::PointXYZ>::Ptr real_side(new pcl::PointCloud<pcl::PointXYZ>);
*real_side = icp_real_.returnRealPointSide();
pcl::PointCloud<pcl::PointXYZ>::Ptr real_bottom(new pcl::PointCloud<pcl::PointXYZ>);
 *real_bottom   = icp_real_.returnRealPointBottom();
```

✓ Real-Origin간의 변환 행렬 계산

```
// 최종 변환 행렬 (origin → real)
Eigen::Matrix4f tf_origin_real_bottom_ = tf_master_bottom_ * input2output_bottom;
Eigen::Matrix4f tf_origin_real_top_ = tf_master_top_ * input2output_top;
Eigen::Matrix4f tf_origin_real_side_ = tf_master_side_ * input2output_side;
```

✓ 계산된 변환 행렬을 실제 point cloud에 적용하여, Origin 좌표계에 정렬

```
// 각각 origin 기준으로 변환
pcl::transformPointCloud(*real_top, *real_top_aligned, tf_origin_real_top_);
pcl::transformPointCloud(*real_side, *real_side_aligned, tf_origin_real_side_);
pcl::transformPointCloud(*real_bottom, *real_bottom_aligned, tf_origin_real_bottom_);
```

✓ 각 시점의 데이터를 하나의 데이터로 merge

```
pcl::PointCloud<pcl::PointXYZ>::Ptr merged(new pcl::PointCloud<pcl::PointXYZ>);
*merged += *real_top_aligned;
*merged += *real_side_aligned;
*merged += *real_bottom_aligned;
```