ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

NESNE TABANLI PROGRAMLAMA I
PROJE RAPORU -2

Nokta Bulutu Bilgileri İle Şekil/Nesne Yorumlayıcı

152120151039 - Can ÇETİNER
152120161066 - Evren RAHİMOĞLU
152120151009 - Nail Emre KAYAPINAR
152120161092 - Tarık COŞKUN

~~Burak CABAŞ~~

İÇİNDEKİLER

Giriş

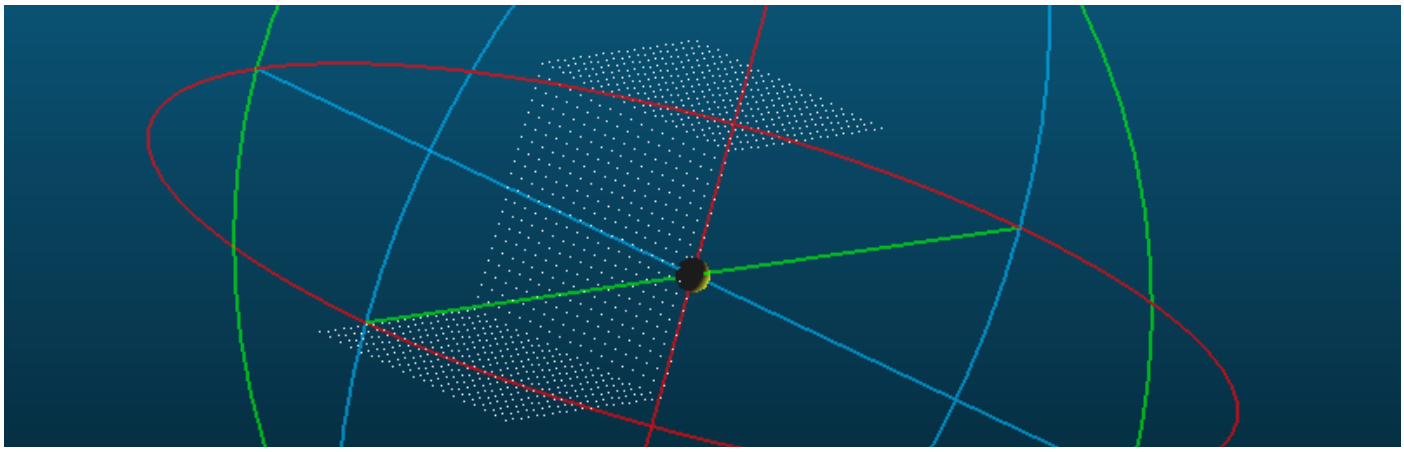
Proje, dış kaynaklardan 3 boyuttaki kordinatlarını okuduğu noktaları kullanarak ve gerekli bazı işlemleri uygulayarak çıktı olarak yine 3 boyutta noktalardan oluşturulmuş olan bir görüntü vermektedir.

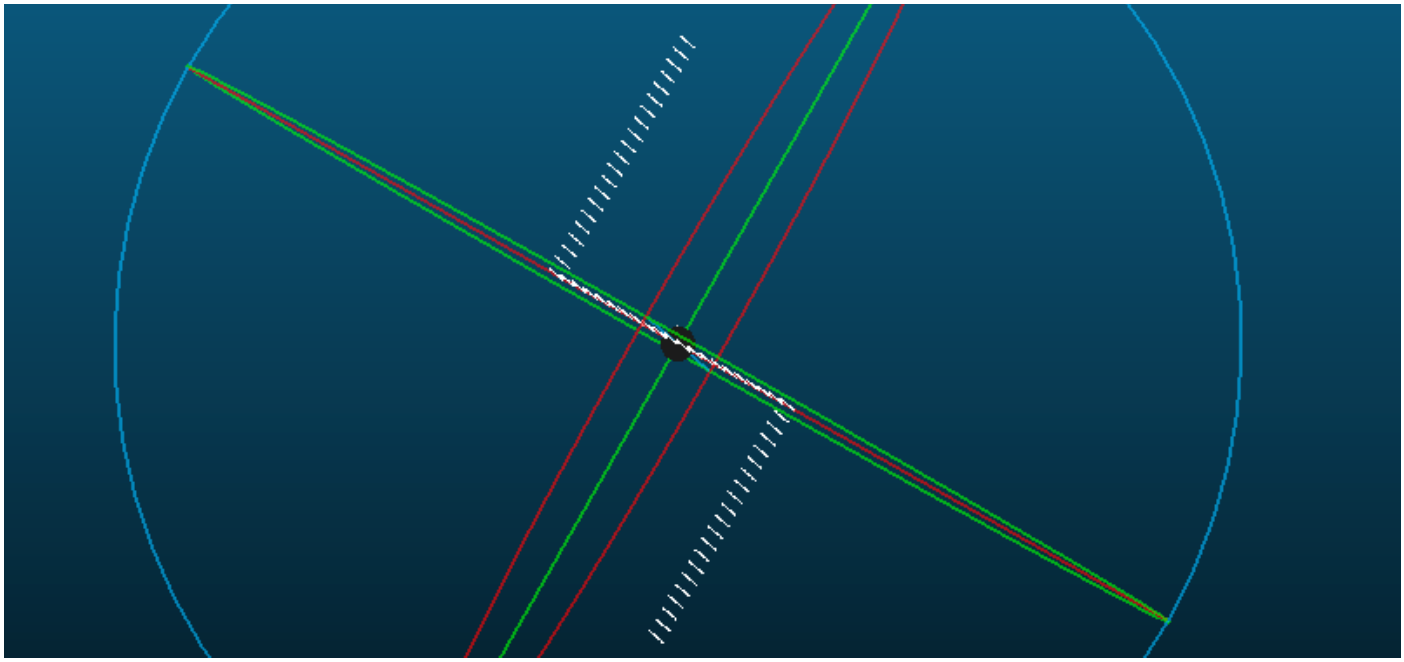
Kendi taban kordinat sistemlerine göre nokta bilgileri mevcut olan 2 adet kameramız vardır **(camera1.txt ve camera2.txt)**. Öncelikle kullanılcak filtreler bir araya toplanır **(FilterPipe Sınıfı)** ve dönüşüm yapılabilmesi için gerekli değerler ile bir transform nesnesi yaratılır **(Transform Sınıfı)**. Daha sonra bu filtreler ve dönüşüm konfigürasyonlarını ilgili kamera için set edilerek **(DepthCamera Sınıfı)** interface içerisine eklenir **(PointCloudInterface)**.

Yazılım içerisindeki **PointCloudGenerator**' dan kalıtılmış sınıflardan hangisi kullanılacak ise uygun değerler ile set edilerek aynı şekilde interface içerisine set edilir. Bu uygulamada 2 adet kamera olduğu için uygun değerler ile set edilerek kullanıma hazırlanır.

Kayıt için gerekli olan recorder **(PointCloudRecorder sınıfı)**, uygun değerler ile yaratılır ve yine interface içerisine set edilir. Böylece interface görevini yerine getirebilmek için tüm parametler ile set edilmiş olur.

Son adımda ise interfacesin "generate" fonksiyonu çalıştırılarak tüm kameraların "captureFor" fonksiyonu tetiklenir ve işlenmiş nokta bulutları bir araya getirilirerek kayıt işlemi başlatılır.

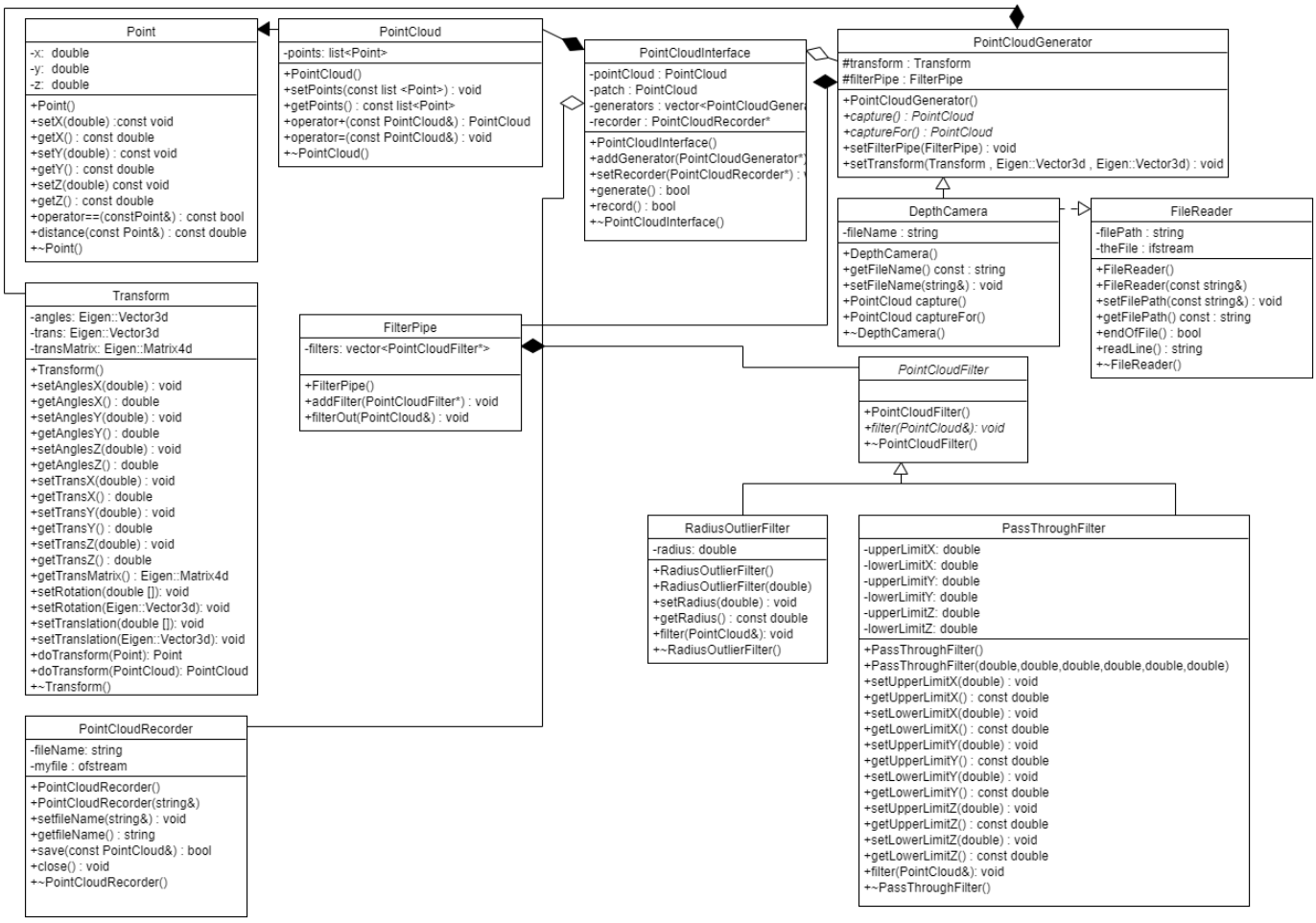




Şekil 1 ve Şekil 2
En son elde edilecek çıktı

Sınıfların Gösteriminde UML Diyagramı

Projenin nihai olarak ulaştığı noktayı gösteren UML çizimi aşağıda detaylandırılmıştır.

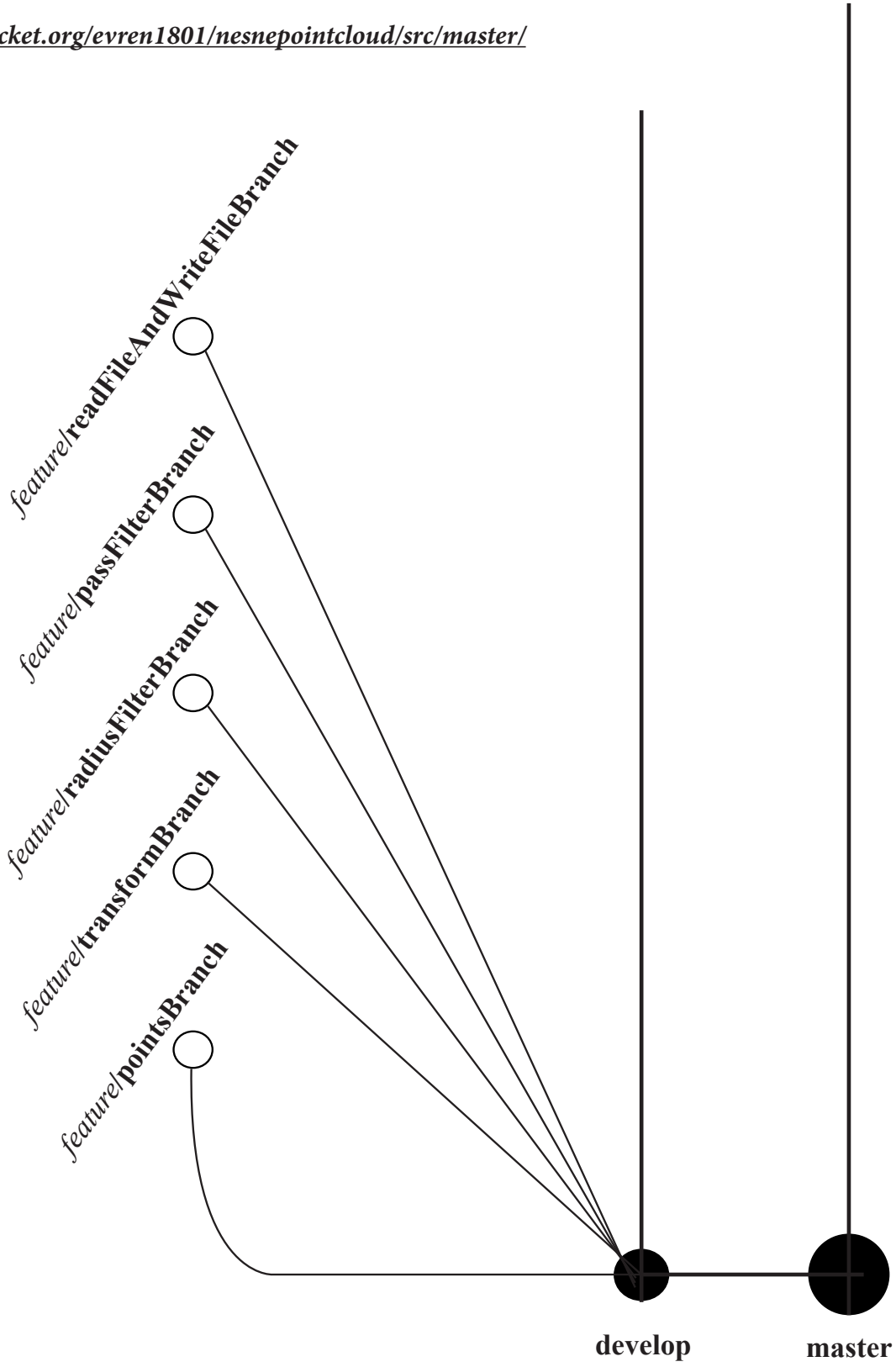


Şekil 3
Projenin UML diyagramı

Bitbucket/Git Organizasyonu

Projenin başında, yapılan görev dağılımlarına göre uygun git/branch yapısı oluşturulup çalışma alanları belrlenmiştir.

https://bitbucket.org/evren1801/nesnepointcloud/src/master/



Şekil 4
Proje başlangıcında repository üzerinde oluşturulan branchler

Sınıflar

Nokta ve Nokta Kümelerinin Tanımlanması:

”Point” Sınıfı: Her bir noktanın yazılımda nasıl tanımlanacağı burada belirlenmiştir. Noktanın koordinatları tutulur.
Bu bilgileri kontrol eden setter/getter fonksiyonları mevcuttur. Ayrıca, noktanın “distance” fonksiyonu ile başka bir noktaya olan uzaklığı kontrol edilebilir. “= =“ operatörü ile de eşitlik durumları sorgulanabilir.

”PointCloud” Sınıfı: Her bir nokta bulutunun nasıl tanımlanacağı burada belirlenmiştir. İçerisinde “Point“ tipinde list tutar. İlgili setter/getter fonksiyonları mevcuttur.
“+“ operatörü ile nokta bulutlarının birleşim kümesi yaratılır ve geri döndürülür. “=” atama operatörü ise bir nokta bulutunu başka bir nokta bulutuna kopyalarken kullanılır.

```
 * \param ctrlDisPoint: ikinci nokta
 * \return distance
 */
double Point::distance(const Point& ctrlDisPoint) const {
    double difX = this->x - ctrlDisPoint.x;
    double difY = this->y - ctrlDisPoint.y;
    double difZ = this->z - ctrlDisPoint.z;

    double sumOfSquares = pow(difX, 2) + pow(difY, 2) + pow(difZ, 2);

    double distance = sqrt(sumOfSquares);

    return distance;
}



/**
 * \brief <h2><b><i>2 nokta kiyaslama</i></b></h2>
 *
 * <p>  Nokta, kendisi ile kendisine verilen baska bir noktanin esitligini kiyaslayip geriye true/false dondurur.</p>
 *
 * \param cmpPoint: ikinci nokta
 * \return true/false
 */
bool Point::operator == (const Point& cmpPoint) const {
```

```
for (it = secondPointCloud.points.begin(); it != secondPointCloud.points.end(); it++) { // (Ikinci PointCloud icin) ...
    for (it2 = next(it, 1); it2 != secondPointCloud.points.end(); it2++) {
        if ((*it) == (*it2)) {
            same = true;
            break;
        }
        else {
            same = false;
        }
    }

    if (same == true && it != prev(secondPointCloud.points.end(), 1)) { continue; }
    secondPointLIST.push_back(*it);
}


secondPointCloud.points = secondPointLIST;


for (it = thisPointLIST.begin(); it != thisPointLIST.end(); it++) { // Bu iki dongu, temizlenmiş noktlardan oluşan vektörleri tek bir vektörd
    joinedPointLIST.push_back(*it);
}
for (it = secondPointLIST.begin(); it != secondPointLIST.end(); it++) { // Bu iki dongu, temizlenmiş noktlardan oluşan vektörleri tek bir vek
    joinedPointLIST.push_back(*it);
}



PointCloud joinedPointCloud; //exit(0);

joinedPointCloud.points = joinedPointLIST;

return joinedPointCloud; // Birleştirilmiş nokta bulutu döndürülüyor
```

Şekil 5 ve Şekil 6
Point ve PointCloud içerisinden bir kaç satır kod

Filtrelerin Tanımlanması:

”FilterPipe” Sınıfı: “addFilter” fonksiyonuna “PointCloudFilter” dan kalıtım alınmış sınıflardan oluşturulan nesnelerin adresleri parametre geçirilerek “filters“ ismindeki bir vektöre upcast yapılır. Diğer üye fonksiyon olan “filterOut“ ise parametre aldığı nokta bulutunu vektördeki tüm filtrelerden geçirerek filtreleme işlemlerini gerçekleştirir.

”PointCloudFilter” Sınıfı: Soyut bir sınıftır. Kendinden kalıtılmış filtre sınıflarına arayüz görevi görmektedir.

```cpp
#include "FilterPipe.h"

void FilterPipe::addFilter(PointCloudFilter* filter) {

    filters.push_back(filter);

}
void FilterPipe::filterOut(PointCloud& pc) {

    for (int i = 0; i < filters.size(); i++)
    {
        filters[i]->filter(pc);
    }


}
```

```cpp
#pragma once
#include "PointCloud.h"

class PointCloudFilter {

public:

    PointCloudFilter(){}
    virtual void filter(PointCloud& pc) = 0;
    ~PointCloudFilter(){}
};
```

Şekil 7 ve Şekil 8
FilterPipe ve PointCloudFilter içerisinden bir kaç satır kod

”RadiusOutlierFilter” Sınıfı: Bir radius değeri belirterek ve ardından “filter“ fonksiyonuna filtreleme yapılacak nokta bulutu geçirilerek nesne başlatılır. Radius değerine göre noktaların birbirine olan uzaklıkları kontrol edilirek filtreleme yapılır. İlgili setter/getter fonksiyonları mevcuttur. Fonksiyon parametreyi adres değeriyle aldığı için nesnenin zaten orjinali üzerinde işlem yapmaktadır.

”PassThroughFilter” Sınıfı: X, Y, Z değerleri için üst ve alt limitler belirterek ve ardından “filter“ fonksiyonuna filtreleme yapılacak nokta bulutu geçirilerek nesne başlatılır. Bu limit değerlerine göre taşma durumları filtrelenir. İlgili setter/getter fonksiyonları mevcuttur. Fonksiyon parametreyi adres değeriyle aldığı için nesnenin zaten orjinali üzerinde işlem yapmaktadır.

```
void RadiusOutlierFilter::filter(PointCloud& pc) {

    vector <Point> clearRadiusPointVec;
    //cout << pc.getPointNumber() << endl;
    //system("pause");

    for (int i = 0; i < pc.getPointNumber(); i++) {
        for (int j = 0; j < pc.getPointNumber(); j++) {
            if ((pc.getPoints()[i].distance(pc.getPoints()[j])) <= radius) {
                if (i == j) { continue; }
                //cout << endl << "<if> " << i << " - " << j << "  ***  " << pc.getPoints()[i].distance(pc.getPoints()[j]) << " <
                clearRadiusPointVec.push_back(pc.getPoints()[i]);
                break;
            }
        }
    }
    //cout << "filter for bitti." << endl;
    //system("pause");

    int newPointNumber = clearRadiusPointVec.size();
    PointCloud pcTMP(newPointNumber);
    Point* pointsTMP = new Point[newPointNumber];

    for (int i = 0; i < newPointNumber; i++) {
        pointsTMP[i] = clearRadiusPointVec[i];
    }
```

```
void PassThroughFilter::filter(PointCloud& pc) {

    vector <Point> clearPassPointVec;

    for (int i = 0; i < pc.getPointNumber(); i++)
    {
        if ((pc.getPoints()[i].getX() <= upperLimitX && pc.getPoints()[i].getX() >= lowerLimitX)
            && (pc.getPoints()[i].getY() <= upperLimitY && pc.getPoints()[i].getY() >= lowerLimitY)
            && (pc.getPoints()[i].getZ() <= upperLimitZ && pc.getPoints()[i].getZ() >= lowerLimitZ))
        {
            clearPassPointVec.push_back(pc.getPoints()[i]);


        }
    }


    int newPointNumber = clearPassPointVec.size();
    PointCloud pcTMP(newPointNumber);
    Point* pointsTMP = new Point[newPointNumber];

    for (int i = 0; i < newPointNumber; i++) {
        pointsTMP[i] = clearPassPointVec[i];
```

Şekil 9 ve Şekil 10
RadiusOutlierFilter ve PassThroughFilter içerisinden bir kaç satır kod

Dönüşümün Tanımlanması:

"Transform" Sınıfı: Rotasyon matrisi ve koordinat sistemleri arasındaki uzaklık bilgilerini kullanarak 4x4 lük bir dönüşüm matrisi yaratır ve dönüşümü buna göre yapar. Bu işlemler için **"Eigen Kütüphanesi"** kullanılmıştır. Dönüşüm sonunda geriye nokta bulutu cinsinden değer döndürür. İlgi setter/getter fonksiyonları mevcuttur.

```
void Transform::setRotation(Eigen::Vector3d anglesEigen) {



    transMatrix(0,0) = (cos(anglesEigen(2) * PI / 180) * cos((anglesEigen(1)) * PI / 180));
    transMatrix(0,1) = (cos(anglesEigen(2) * PI / 180) * sin(anglesEigen(1) * PI / 180) * sin(anglesEigen(0) * PI / 180)) - (sin(anglesEigen
    transMatrix(0,2) = (cos(anglesEigen(2) * PI / 180) * sin(anglesEigen(1) * PI / 180) * cos(anglesEigen(0) * PI / 180)) + (sin(anglesEigen
    transMatrix(1,0) = (sin(anglesEigen(2) * PI / 180) * cos(anglesEigen(1) * PI / 180));
    transMatrix(1,1) = (sin(anglesEigen(2) * PI / 180) * sin(anglesEigen(1) * PI / 180) * sin(anglesEigen(0) * PI / 180) + (cos(anglesEigen(
    transMatrix(1,2) = (sin(anglesEigen(2) * PI / 180) * sin(anglesEigen(1) * PI / 180) * cos(anglesEigen(0) * PI / 180)) - (cos(anglesEigen
    transMatrix(2,0) = (-sin(anglesEigen(1) * PI / 180));
    transMatrix(2,1) = (cos(anglesEigen(1) * PI / 180) * sin(anglesEigen(0) * PI / 180));
    transMatrix(2,2) = { cos(anglesEigen(1) * PI / 180) * cos(anglesEigen(0) * PI / 180) };
```

```
void Transform::setTranslation(Eigen::Vector3d transEigen) {


    transMatrix(0,3) = (transEigen(0));
    transMatrix(1,3) = (transEigen(1));
    transMatrix(2,3) = (transEigen(2));
    transMatrix(3,0) = (0);
    transMatrix(3,1) = (0);
    transMatrix(3,2) = (0);
    transMatrix(3,3) = (1);


}



void Transform::setTranslation(double transGelen[]) {
```

```
Point Transform::doTransform(Point p) {

    Point pointTMP;

    double xTMP = p.getX();
    double yTMP = p.getY();
    double zTMP = p.getZ();

    Eigen::Vector4d resultPoint;
    Eigen::Vector4d thePoint= { xTMP , yTMP , zTMP , 1 };

    resultPoint = transMatrix * thePoint;

    pointTMP.setX(resultPoint(0));
    pointTMP.setY(resultPoint(1));
    pointTMP.setZ(resultPoint(2));

    return pointTMP;
}
/**
 * \brief <h2><b><i>doTransform fonksiyonu</i></b></h2>
 *
 * <p>  icine aldigi PointCloud tan olusturulmus bir nokta bulutunu alip transform islemine tabi tuta
 *
```

Şekil 11, Şekil 12 ve Şekil 13
Transform içerisinden bir kaç satır kod

Nokta Bulutu Yaratıcılarının Tanımlanması:

”PointCloudGenerator” Sınıfı: Soyut bir sınıftır. Kendinden kalıtılmış nokta bulutu sağlayıcıları sınıflarına arayüz görevi görmektedir.

”DepthCamera” Sınıfı: Kameraların kordinatlarının bulunduğu dosyaları okuma ve bu degerlere uygun olarak nokta bulutu yaratma bu sınıfın görevidir. İlgili setter/getter fonksiyonları mevcuttur. Dosya yolu set edilir ve daha sonrasında “capture” veya “captureFor“ fonksiyonu tetiklenerek “FileReader” sınıfı üzerinden satır satır okuma yapılır.

```
void PointCloudGenerator::setFilterPipe(FilterPipe filterPipe) {

    this->filterPipe = filterPipe;

}
/**
 * \brief <h2><b><i>setTransform fonksiyonu</i></b></h2>
 *
 * <p>  Transform icin gerekli olan setRotation ve setTranslation fonksiyonlari burda calistirili
 *
 * \param transform: Transform
 * \param rotation: Eigen::Vector3d
 * \param translation: Eigen::Vector3d
 */
void PointCloudGenerator::setTransform(Transform transform , Eigen::Vector3d rotation , Eigen::Vector3d tr

    this->transform = transform;
    this->transform.setRotation(rotation);
    this->transform.setTranslation(translation);

}
```

```
 * \brief <h2><b><i>Dosyadaki noktalari algilar</i></b></h2>
 *
 * <p>  Dosya okuma sinifi ile dosyayi satir satir okur ve uygun sekilde nokta bulutunu yara
 *
 * \return pointCloudTMP
 */
PointCloud DepthCamera::captureFor() {
    PointCloud pcTMP = this->capture();

    filterPipe.filterOut(pcTMP);

    pcTMP = transform.doTransform(pcTMP); // pcTMP için dönüşüm başlar.


    return pcTMP;
}
```

Şekil 14 ve Şekil 15
PointCloudGenerator ve DepthCamera içerisinden bir kaç satır kod

”FileReader” Sınıfı: Dosya okuma bu class üzerinde tanımlanmıştır. İlgili setter/getter fonksiyonları mevcuttur. Dosya yolu verilerek nesne başlatılır. “readLine“ fonksiyonu ile sırayla bir satır okur ve o satırı geriye döndürür. En son kaldığı yeri unutmaz. Dosya sonu okunduğunda dosya kapatılır.

```
void FileReader::setFilePath(const string& filePath) {
    if (theFile.is_open()) {
        theFile.close();
    }

    this->filePath = filePath;
    theFile.open(filePath);

    try {
        if (!theFile.is_open()) {
            throw errorReadFile();
        }
    }
    catch (errorReadFile error) {
        error.message(filePath);
        theFile.clear();
    }
}
```

```
 */
string FileReader::readLine() {
    string line;

    while (getline(theFile, line)) {
        return line;
    }

    if (theFile.eof()) {
        theFile.close();
    }
}
```

Şekil 16
FileReader içerisinden bir kaç satır kod

Kaydedicinin Tanımlanması:

”PointCloudRecorder” Sınıfı: Bir dosya adı belirtilere başlatılır ve ardından “save“ fonksiyonuna nokta bulutu geçirilerek dosyay kayıt edilmesi sağlanır. İlgili setter/getter fonksiyonları mevcuttur.

```
void PointCloudRecorder::setfileName(string& fileName) {
    if (myfile.is_open()) {
        myfile.close();
    }

    this->fileName = fileName;
    myfile.open(fileName, ios::app);

    try {
        if (!myfile.is_open()) {
            throw errorWriteFile();
        }
    }
    catch (errorWriteFile error) {
        error.message(fileName);
    }
}
/**
 * \brief <h2><b><i>getfileName fonksiyonu</i></b></h2>
```

```
bool PointCloudRecorder::save(const PointCloud& pc) {

    list <Point> PointLIST = pc.getPoints();

    list <Point>::iterator it;

    for (it = PointLIST.begin(); it != PointLIST.end(); it++)
    {
        try {
            myfile << it->getX() << "  " << it->getY() << "  " << it->getZ() << endl;
            if (myfile.fail()) {
                throw errorWriteLine();
            }
        }
        catch (errorWriteLine error) {
            error.message();
            myfile.clear();
            return false;
        }
    }

    return true;
```

Şekil 17
PointCloudRecorder içerisinden bir kaç satır kod

Arayüzün Tanımlanması:

”PointCloudInterface” Sınıfı: Uygun parametrelerin bir arada bulunduğu durumlarda -generator ve recorder istenilen biçimde set edildiğinde- tüm işlemlerin doğru biçimde çalışmasını yönetecek olan arayüzdür. İş adımlarının daha sonra da doğru yürütlmesi açısından ve sürekli aynı akışı kodlama durumunu ortadan kaldırmak için önemli bir konumdadır.

```
bool PointCloudInterface::generate() {

    PointCloud tmpPC;
    int totalSize = 0;

    for (int i = 0; i < generators.size(); i++)
    {
        tmpPC = generators[i]->captureFor();
        pointCloud = pointCloud + tmpPC;
        totalSize += tmpPC.getPoints().size();

    }


    if (pointCloud.getPoints().size() == totalSize)
        return true;

    return false;
```

```
 *
 * <p>  pointCloud uyesi generate fonksiyonuyla beraber son haline geldikten
 *
 * \return true/false
 */
bool PointCloudInterface::record() {


    if (recorder->save(pointCloud)) {
        recorder->close();
        return true;
    }

    return false;
}
```

Şekil 18
PointCloudRecorder içerisinden bir kaç satır kod

Projedeki Görev Dağılımları

NOT1: Projeye başlamadan önce toplantı yapılarak, projenin akışı tartışılmıştır. Akabinde görev paylaşımları yapılarak bireysel çalışma pozisyonuna geçilmiştir.

152120151039 - Can ÇETİNER

- "FilterPipe" Sınıfı
- "PointCloudFilter" Sınıfı
- Önceki filtreler için inheritance

- Rapor Hazırlığı
- Proje Genel Testi (pointCloudCamerasApp)
- Git-Bitbucket Kullanımı

152120161066 - Evren RAHİMOĞLU

- "Transform" Sınıfı için Eigen Kütüphanesi
- "PointCloudGenerator" Sınıfı
- "PointCloudInterface" Sınıfı

- UML Çizimi
- Doxygen Kontrolü ve Düzenlemeler
- Git-Bitbucket Kullanımı

152120151009 - Nail Emre KAYAPINAR

- "FilterPipe" Sınıfı
- "PointCloudFilter" Sınıfı
- Önceki filtreler için inheritance
-
- Rapor Hazırlığı
- Proje Genel Testi (pointCloudCamerasApp)
- Git-Bitbucket Kullanımı

152120161092 - Tarık COŞKUN

- "PointCloud" Sınıfı için <list> yapısı
- "DepthCamera" Sınıfı "captureFor" fonksiyonu
- "PointCloudGenerator" Sınıfı
- "PointCloudInterface" Sınıfı

- Rapor Tasarımı
- Git-Bitbucket Organizasyonu
- Git-Bitbucket Kullanımı

Sonuç

Projenin ikinci aşamasında, ilk aşamadan çıkardığımız dersler sebebi ile zaman yönetimi çok daha etkiliydi. Aynı şekilde her bir takım üyesinin yetenekleri ve proje adaptasyonu göz önüne alınarak daha makul bir görev paylaşımı yapıldı. Dolayısı ile daha az sorunla karşılaştık ve daha rahat çözümler ürettik. Büyük resme bakarak konuşacak olursak, grup halinde çalışmanın gerekliliklerini öğrenmiş, zorluklarını yaşamış, doğru iş paylaşımının iyi bir ieltişimin sonucu oduğunu görmüş olduk. Ek olarak projenin ikinci kısmında “Pair Programming” tekniğini de daha etkin ve yoğun kullandık. Teknik olarak projeye bakarsak da, soyut sınıf nedir ve nasıl kullanılır, kalıtım nasıl kullanılır, minimum degişiklik-meximum genişleme kodda nasıl sağlanır, daha karmaşık bir UML diyagramı nasıl okunur gibi sorularımıza tecrübe ederek yanıt bulduk.