R E P O R T

Ouster, OctoMap을 활용한 3D 맵 생성

한국기계연구원
스마트산업기계연구실 현장실습생
박영준

1. 실행 환경

가. 하드웨어

구분	제원	보기
PC	Intel NUC	
LiDAR	Ouster OS0-32-G	

나. 소프트웨어

1) Ubuntu 18.04
2) ROS Mellodic
3) Ouster_example[1)]
4) Octomap_mapping[2)]

다. GitHub

1) https://github.com/GadonGadon/OusterOctomap

1) https://github.com/ouster-lidar/ouster_example
2) https://github.com/OctoMap/octomap_mapping

2. Ouster

가. 설치

ouster_example 깃 자료를 받아 사용하였으며 ouster lidar의 hostname은 192.168.0.23, udp_dest는 192.168.0.31을 할당하였다.

나. 클러스터링 결과

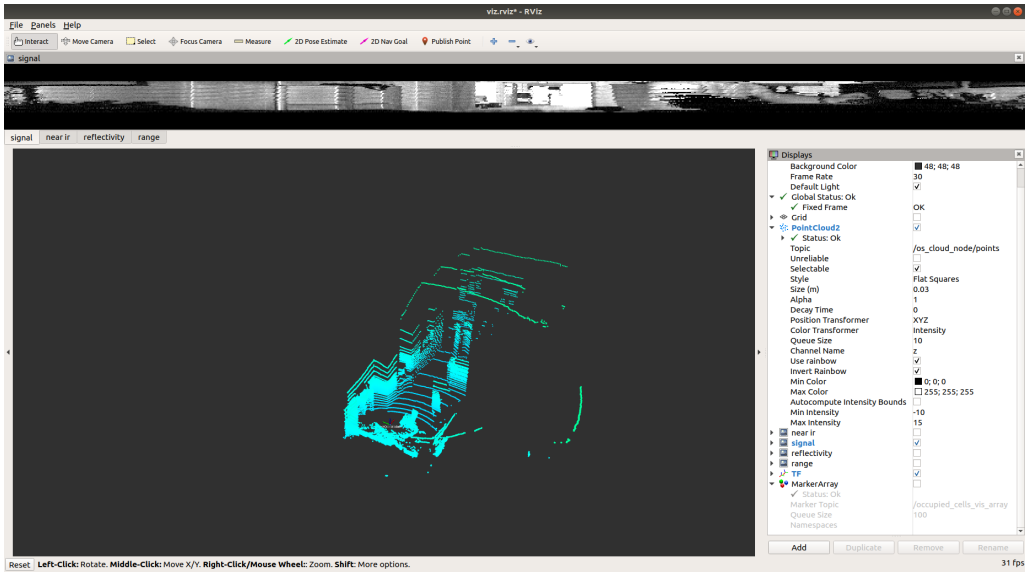


그림 1 OusterLidar 클러스터링 결과

실내에서 Ouster 라이다를 작동시켜 클러스터링 한 결과 [그림 1]과 같이 실내의 모습이 CloudPoint로 나타났다. 다만, 센서를 움직여도 좌표계가 고정이 되는 문제가 발생하였다. 이럴 경우 OctoMap에 매핑을 진행할 때, 센서는 고정되어 있고 주변 환경이 계속 변하는 것으로 인지되어 매핑이 제대로 되지 않는다.

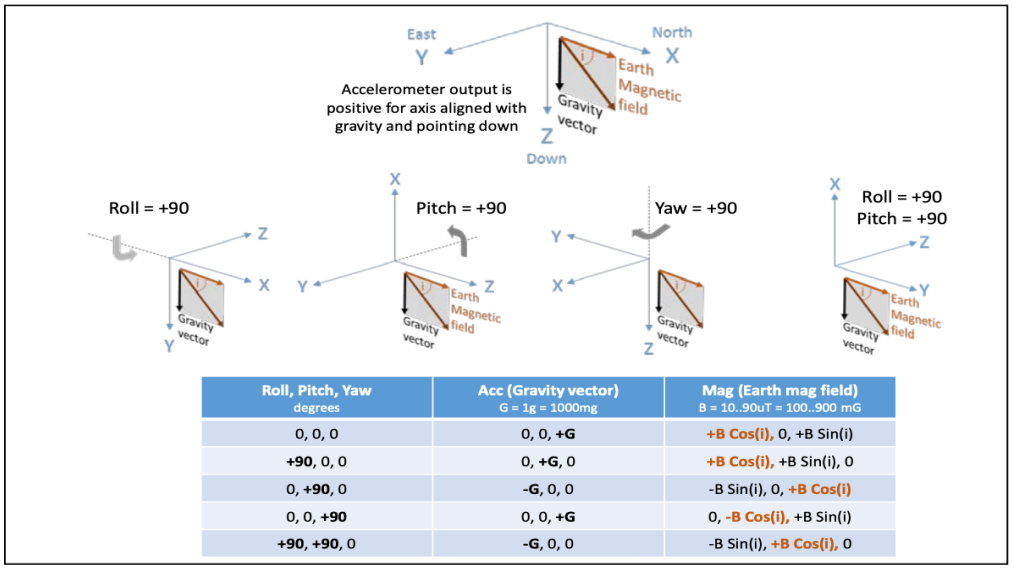
다. 문제 해결 1 - 선형가속도를 이용한 계산

해당 문제를 해결하기 위해 먼저 IMU토픽 메시지의 내용을 확인해 보았다.

```
---
header:
  seq: 3779
  stamp:
    secs: 6956
    nsecs: 972341435
  frame_id: "testWorld/os_imu"
orientation:
  x: 0.0
  y: 0.0
  z: 0.0
  w: 0.0
orientation_covariance: [-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0]
angular_velocity:
  x: 0.0175768631945
  y: -0.00532632218016
  z: 0.00106526443603
angular_velocity_covariance: [0.0006, 0.0, 0.0, 0.0, 0.0006, 0.0, 0.0, 0.0, 0.0006]
linear_acceleration:
  x: -0.0215478149414
  y: 0.00718260498047
  z: 9.73242974854
linear_acceleration_covariance: [0.01, 0.0, 0.0, 0.0, 0.01, 0.0, 0.0, 0.0, 0.01]
---
```

그림 2 /os_cloude_node/imu 토픽 메시지

Ouster에 내장된 IMU는 sensor_msgs/imu타입으로 토픽을 발행하며 각속도(angular_velocity)와 선형가속도(linear_acceleration)의 6축 정보를 1000Hz로 알려준다. 여기서 정지 상태일때의 선형가속도는 중력가속도를 받아 z축으로 약 9.8m/s^2이 가해지고 있는 것을 알 수 있다.



Roll, Pitch, Yaw degrees	Acc (Gravity vector) G = 1g = 1000mg	Mag (Earth mag field) B = 10..90uT = 100..900 mG
0, 0, 0	0, 0, **+G**	**+B Cos(i),** 0, +B Sin(i)
+90, 0, 0	0, **+G**, 0	**+B Cos(i),** +B Sin(i), 0
0, **+90**, 0	**-G**, 0, 0	-B Sin(i), 0, **+B Cos(i)**
0, 0, **+90**	0, 0, **+G**	0, **-B Cos(i),** +B Sin(i)
+90, **+90**, 0	**-G**, 0, 0	-B Sin(i), **+B Cos(i),** 0

3)

그림 3 중력가속도와 Roll, Pitch, Yaw

또한 각속도를 적분하여 사용한다면 현재 센서의 오일러 각을 계산할 수 있다. 먼저, 결과를 빠르게 보기 위해 선형가속도의 중력가속도를 이용하여 센서를 회

3) Andrea Vitali, Residual linear acceleration by gravity subtractionto enable dead-reckoning, p.2

전시키는 방법을 적용하기로 하였다.

첫 번째로, TF Tree에서 고정된 좌표계를 하나 생성하여야한다. Ouster_example을 처음 작동 할때는 [그림 4]와 같이 OS_Sensor 아래에 라이다와 imu가 이어져있는 TF Tree가 만들어진다.

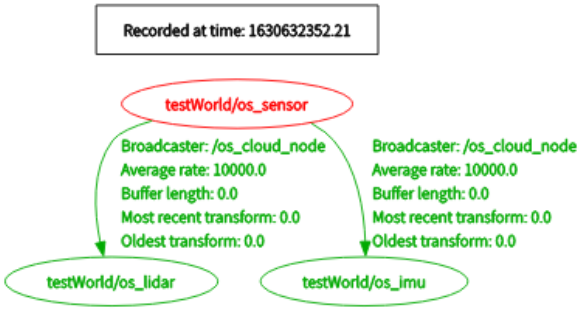


그림 4 TF Tree

고정된 좌표계를 생성하기 위해 ouster_ros/src 내에 있는 소스코드인 os_cloud_node.cpp파일을 수정하였다. 먼저 main함수에 좌표계의 이름을 추가하였다.

```
auto main_frame = tf_prefix +"odom";
```

이후, 라이다에서 패킷을 받을때마다 실행되는 콜백함수에서 CloudPoint를 퍼블리시 할 때, sensor_frame이 아닌 main_frame에 그려지도록 수정해야한다. 따라서, auto lidar_handler = [&](const PacketMsg& pm) mutable 함수에서 값을 아래와 같이 수정하였다.

```
lidar_pub.publish(ouster_ros::cloud_to_cloud_msg(
                        cloud, h->timestamp, main_frame));
```

또한, IMU가 보내주는 데이터를 받아 그에 맞게 sensor_frame의 좌표계를 변경해야 한다. 따라서, auto imu_handler = [&](const PacketMsg& p)함수에서 아래와 같이 수정하였다.

```
double x = ouster_ros::packet_to_imu_msg(p, imu_frame, pf).linear_acceleration.x;
double y = ouster_ros::packet_to_imu_msg(p, imu_frame, pf).linear_acceleration.y;
broadcaster.sendTransform(
        tf::StampedTransform(
                tf::Transform(tf::Quaternion(-y/20, x/20, 0, 1), tf::Vector3(0, 0, 0)),
        ouster_ros::packet_to_imu_msg(p, imu_frame, pf).header.stamp,
                main_frame, sensor_frame));
```

x, y는 각각 선형가속도의 x축, y축 값이며, 이를 이용하여 main_frame에서 sensor_frame으로 가는 좌표를 변환하였다.

결과적으로 [그림 5]와 같이 새로운 프레임인 testWorld/odom프레임이 생성되었으며 [그림 6]과 같이 sensor프레임의 좌표계가 변화하는 것을 확인할 수 있었다.

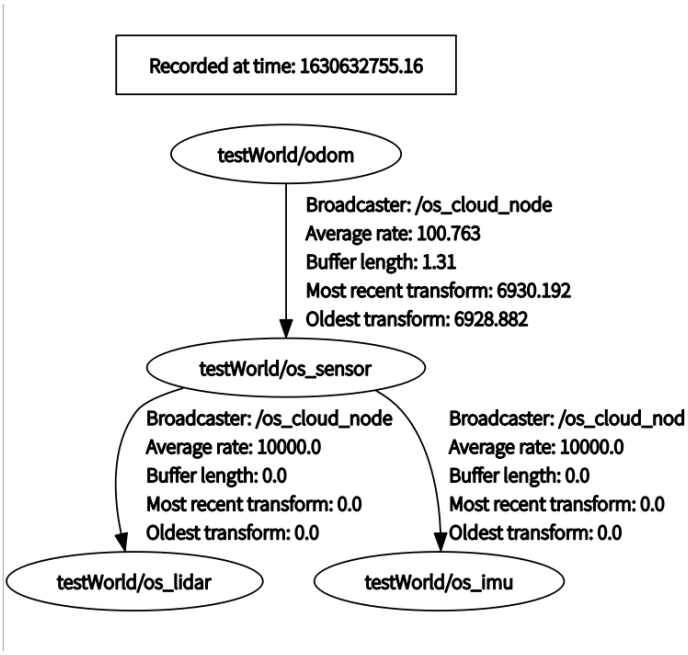


그림 5 변화된 TF Tree

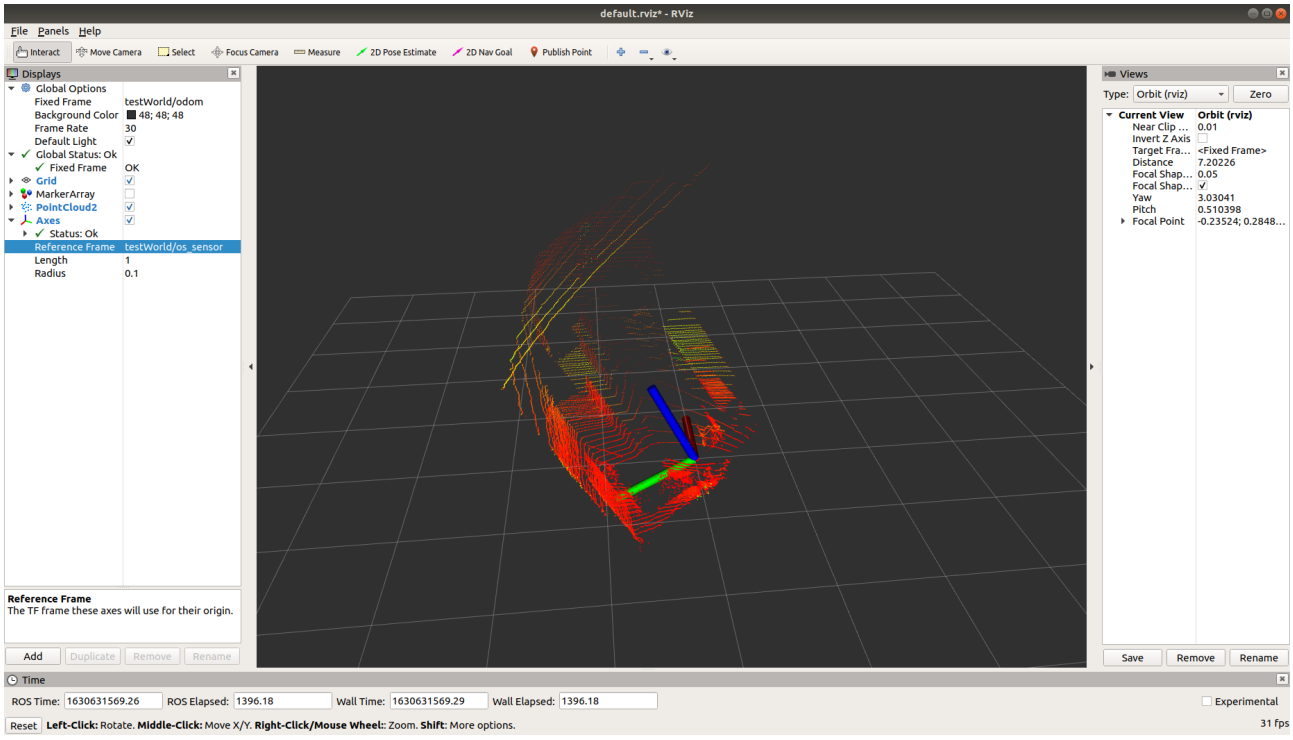


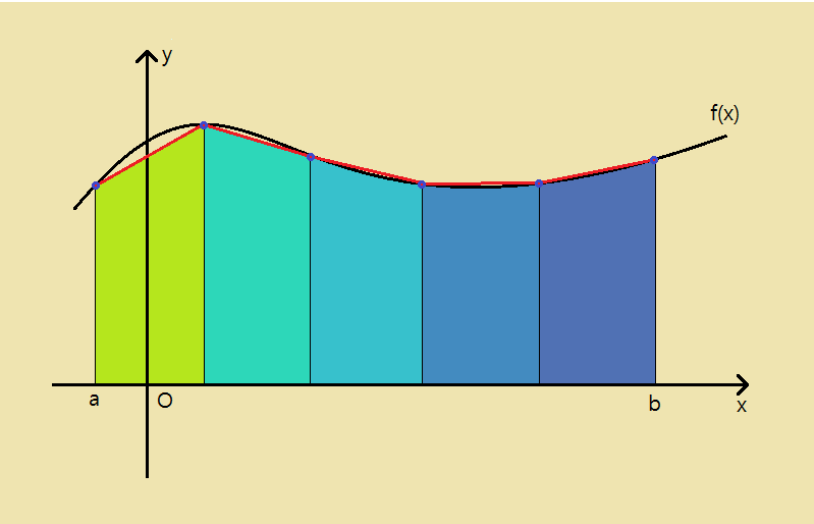
그림 6 OS_Sensor의 좌표축

문제점

정지상태일 경우 일정하게 중력가속도를 받아 현재 센서의 기울기 값을 추종할 수 있다. 하지만 이동 중인 상태일 경우 특정 방향으로 가속도 값이 들어오게 되므로 정확한 값을 측정할 수 없게 된다. 또한 z축 회전에 대해서는 중력가속도가 일정하게 들어오므로 측정할 수 없다.

라. 문제 해결 2 - 각속도를 이용한 계산

두 번째 방법으로는 IMU메세지([그림 2])의 Angular_velocity의 적분값을 이용하여 각 변화량을 추정하는 방법이다. 이는 단위시간 변화량에 각속도를 곱해서 더하는 방법으로 각 변화량을 얻을 수 있다. 아날로그 값을 컴퓨터로 적분하는 만큼 생기는 오차를 최소화하기 위해 [그림 7]과 같이 사다리꼴의 면적을 계산하는 방법으로 적분을 진행하였다.



4)

그림 7 사다리꼴 적분

적분을 하기위해 사용한 코드는 다음과 같다.

```
int pre_time, cur_time;
double X=0.0,Y=0.0,Z=0.0;
double pre_ax =0.0, pre_ay =0.0, pre_az =0.0;
double ax =0.0, ay =0.0, az =0.0;
```

먼저 시간변화량을 측정하기위한 pre_time, cur_time, 적분한 값을 저장하기 위한 X, Y, Z, 이전 각속도 및 현재 각속도를 저장하기 위한 pre_ax, pre_ay, pre_az, ax, ay, az를 만들어준다.

4) https://hs36.tistory.com/38, 자이로 센서값 적분 방법들

```
pre_time = cur_time;
cur_time = ouster_ros::packet_to_imu_msg(p, imu_frame, pf).header.stamp.nsec;
double dt = (cur_time - pre_time)/1000000000.0;
pre_ax = ax;     pre_ay = ay;     pre_az = az;
ax = ouster_ros::packet_to_imu_msg(p, imu_frame, pf).angular_velocity.x;
ay = ouster_ros::packet_to_imu_msg(p, imu_frame, pf).angular_velocity.y;
az = ouster_ros::packet_to_imu_msg(p, imu_frame, pf).angular_velocity.z;
```

이후 imu callback 함수 내에서 다음과 같이 입력하여 시간변화량과, 현재 각속도 및 이전 각속도를 저장한다. 여기서 pre_time과 cur_time은 나노초 단위이므로 dt를 계산할 때 10의 9승으로 나누어 초 단위로 변경하였다.

```
X += dt*(ax+pre_ax)/2;
Y += dt*(ay+pre_ay)/2;
Z += dt*(az+pre_az)/2;
```

마지막으로 시간변화량에 각속도를 곱해 X에 더함으로써 갑을 적분할 수 있었다.

문제점

각속도를 적분하는 방법으로 각 변화량을 구했으나, imu의 떨림에 의한 오차, 빠른 움직임에서 생기는 오차가 누적되어 센서를 움직이지 않고 두어도 값이 점점 발산하는 모습을 보이고 있다. 아래의 [그림 8]은 30분동안 가만히 둔 IMU의 각 축별 그래프이다. 오차가 점점 발산하는 것을 확인할 수 있다. 이는 시간에 따른 imu의 누적오차로 드리프트(drift)라고 한다. 드리프트를 보정하기 위해서는 칼만필터와 같이 다양한 보정 필터를 적용해야한다.

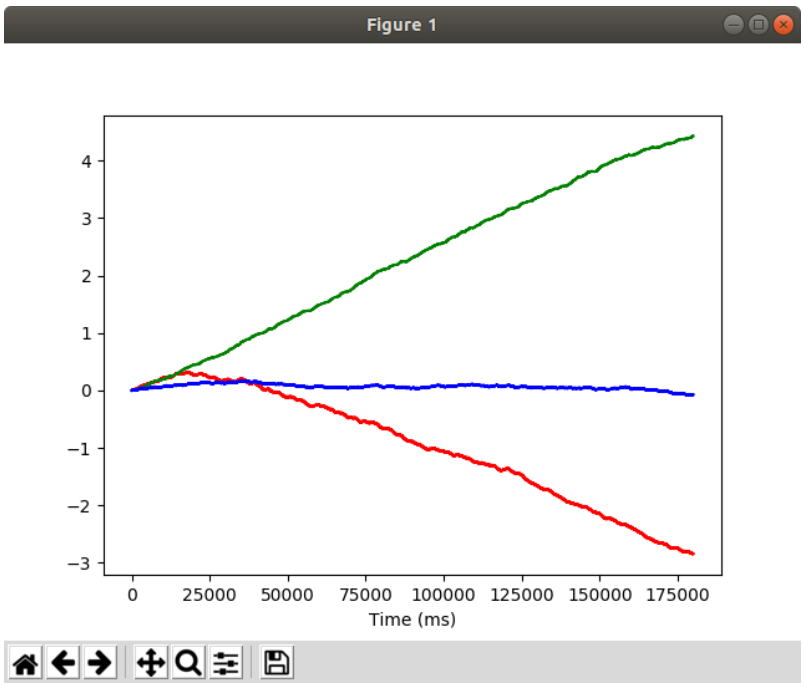


그림 8 시간에 따른 각 변화량 (빨강 : x, 초록 : y, 파랑 : z)

마. 문제 해결 3 - Madgwick Algorithm

IMU신호의 오차를 보정하기 위한 필터로 Kalman 필터와 Madgwick 알고리즘이 있다. Kalman 필터와 Madgwick 알고리즘 모두 같은 역할을 수행하며 각각의 성능은 다음과 같다.

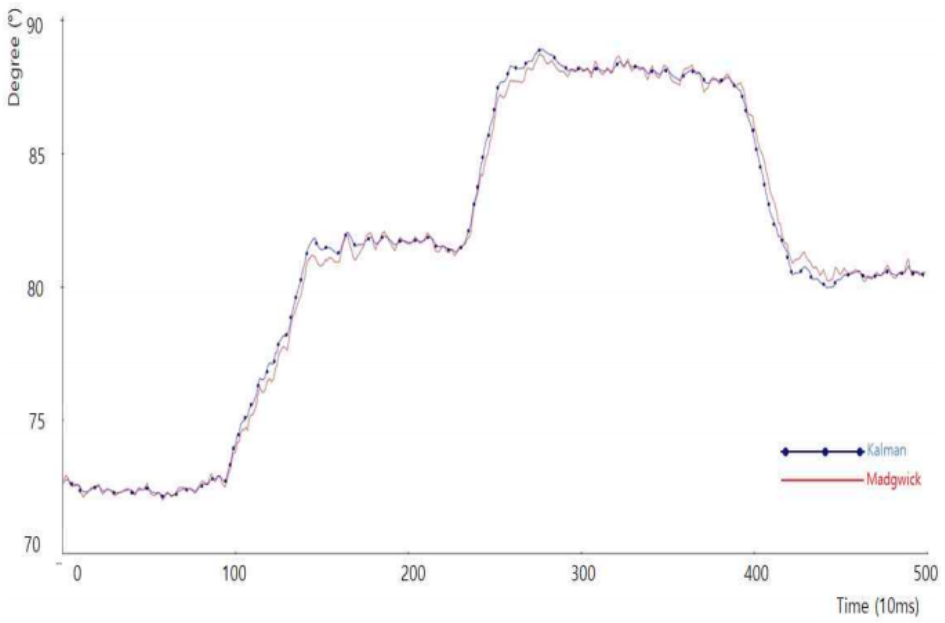


그림 9 Yaw각의 Kalman과 Madgwick, 한지수 외 6인. (2018). 화력발전소 보일러노 내 드론 자세보정을 위한 Madgwick 알고리즘과 kalman filter 알고리즘에 관한 연구. 항공우주시스템공학회 학술대회 발표집, 2018(2): 1-2

[그림 9]에서 Kalman과 Madgwick은 평균 0.125도 차이로 비슷한 성능을 보이고 있다. 또한 연산 속도는 Madgwick 알고리즘이 약 12배 빠른 것으로 도출되었다.

이번 실험에서는 Madgwick알고리즘을 사용하여 IMU의 값을 비교해 보았다. Madgwick 알고리즘은 깃허브의 패키지[5]를 가져와 사용하였으며 imu sensor메시지를 Subscribe하여 quaternion메시지로 변환 후 Publich한다. 위와 마찬가지로 30분동안 IMU를 가만히 두어 측정했다.

5) https://github.com/marooncn/imu_filter

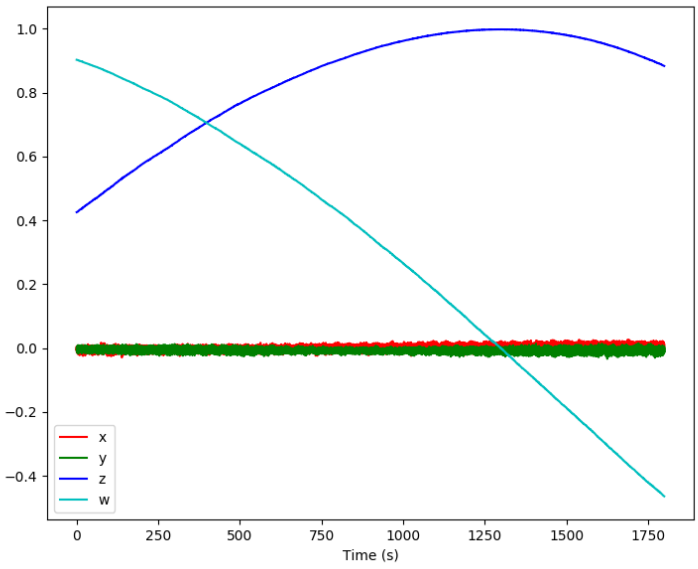


그림 10 Madgwick알고리즘

Madgwick 알고리즘 적용결과 x, y축에 대해 오차가 거의 생기지 않는 것을 확인할 수 있다. 하지만 z축과 w에 대해서는 오차가 발생함을 확인하였다.

문제점

1. 느린 속도
   - 기본적으로 필터링 속도가 느려 회전을 천천히 하게 된다. 이럴 경우 Octomap을 사용할 때 매핑이 제대로 되지 않는다.
2. z축 회전 감지 못함
   - 속도가 느리더라도 x축과 y축에 대한 회전은 거의 정확하게 이루어 진다. 하지만 z축에 대한 회전을 감지하지 못해 매핑이 제대로 되지 않는다.

3. OctoMap

가. Octree

Octree는 3D에서 다양한 활용성을 가지고 있는 트리의 한 종류로 기본 구조는 [그림 11]과 같이 하나의 부모노드에 8개의 자식 노드가 있는 형태를 한다.

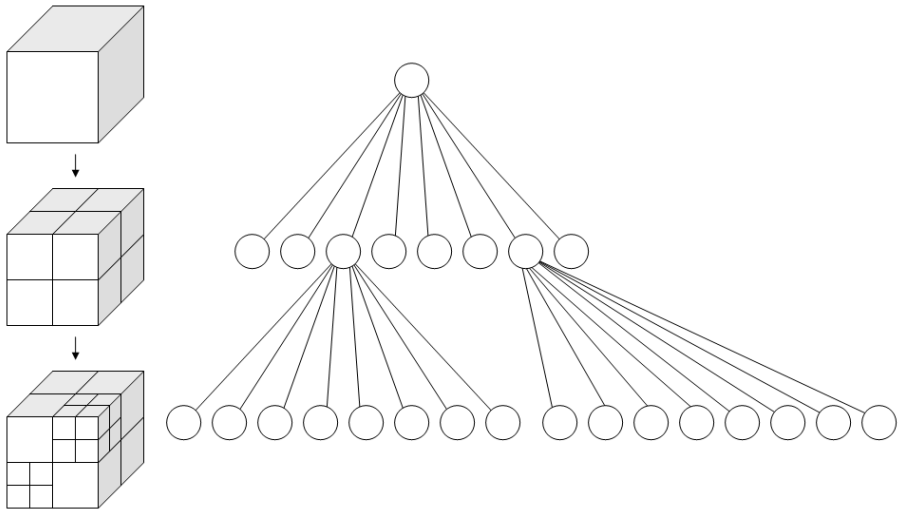


그림 11 Octree

기본적으로 Octree는 3차원 공간을 분할하는데 쓰이며, 2차원에서 사용하는 Quadtree와 다르게 높이에 대한 분할까지 시도한다.

나. OctoMap

OctoMap은 Octree를 기반으로 SLAM을 구현하는 라이브러리이다. ROS의 경우 깃허브에서 패키지를 다운 받은 후 런치파일을 이용하여 실행할수 있으며 런치파일의 내용을 수정하여 원하는 결과를 얻을 수 있다.

```
<launch>
        <node pkg="octomap_server" type="octomap_server_node" name="octomap_server">
                <param name="resolution" value="0.1"/>

                <!-- fixed map frame (set to 'map'if SLAM or localization running!) -->
                <param name="frame_id" type="string" value="odom“/>

                <!-- maximum range to integrate (speedup!) -->
                <param name="sensor_model/max_range" value="500.0"/>

                <!-- data source to integrate (PointCloud2) -->
                <remap from="cloud_in" to="/os_cloud_node/points"/>

        </node>
</launch>
```

런치파일 내에서 각 파라미터의 역할은 다음과 같다.

1. resolution : OctoMap에서 각 박스의 크기를 나타내며 값이 커질수록 커진다.
2. frame_id : 고정 frame의 이름으로 움직이지 않는 프레임을 넣는다.
3. sensor_model/max_range : 라이다의 최대 감지거리로 Ouster라이다의 경우 500m이다.
4. cloud_in : 라이다에서 퍼블리시 되는 Pointcloud의 토픽명을 넣는다.