OpenIA

Version 3.0

User Manual

OpenIA SW

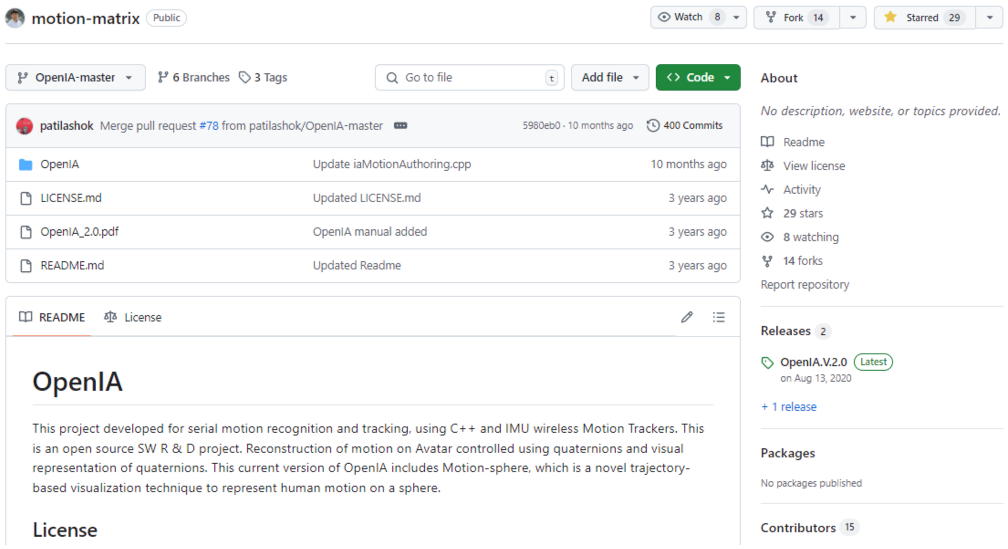
Environment Setup

While the project has started in 2018 and it has only been released in the public domain under BSD license in 2019 and Apache license in 2021. As OpenIA is on open-source project, everyone is free and welcome to extend its capabilities. OpenIA is developed in C++. It is currently compiled on Windows using CMake and for 32bits or 64bits architectures. This manual gives you basic environmental setup and its technical details. Don't hesitate to ask questions and share your experiences and take a look at the Github source repository OpenIA.

Forking the GIT

The OpenIA project is hosted in the GIT under the link https://github.com/younghochai/motionmatrix (Figure1.1).

The latest release is always setup in **the OpenIA 3.0**. One can create a local copy of the entire repositorty by clicking on the fork option available in the right top corner of the github page. It is recommended to fork the repository before bringing the code base into your local machine.



patilashok Merge pull request #78 from patilashok/OpenIA-master 5980eb0 · 10 months ago 400 Commits

OpenIA	Update iaMotionAuthoring.cpp	10 months ago
LICENSE.md	Updated LICENSE.md	3 years ago
OpenIA_2.0.pdf	OpenIA manual added	3 years ago
README.md	Updated Readme	3 years ago

About

No description, website, or topics provided.

Readme
View license
Activity
29 stars
8 watching
14 forks
Report repository

Releases 2

OpenIA.V.2.0 Latest on Aug 13, 2020

+ 1 release

Packages

No packages published

Contributors 15

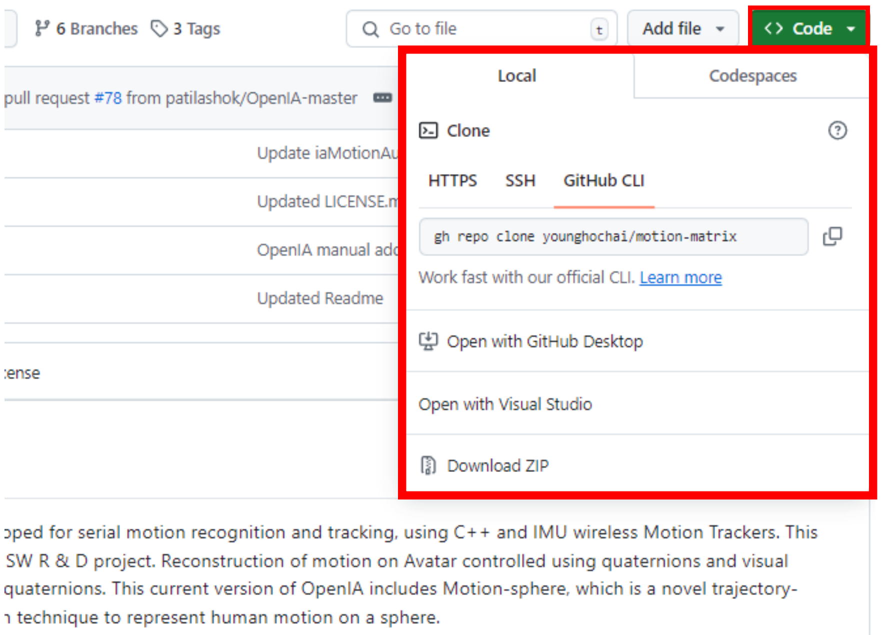
OpenIA

This project developed for serial motion recognition and tracking, using C++ and IMU wireless Motion Trackers. This is an open source SW R & D project. Reconstruction of motion on Avatar controlled using quaternions and visual representation of quaternions. This current version of OpenIA includes Motion-sphere, which is a novel trajectory-based visualization technique to represent human motion on a sphere.

License

FIGURE 1.1: OpenIA project Github home page.

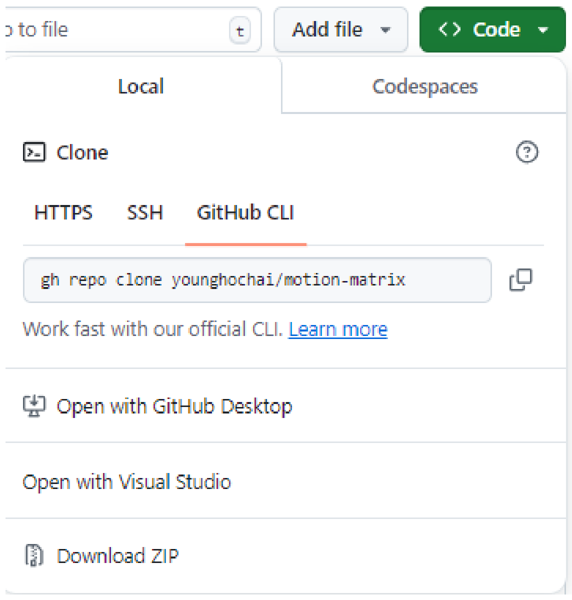
The GIT provides three options (Figure 1.2), direclty clone using https or ssh command, next using a git desktop application or directly download ZIP for cloning the remote repository into the local machine (https://desktop.github.com/ for windows). One of use this desktop tool as an alternate for command line interface to clone the code base to the local machine.



```
gh repo clone younghochai/motion-matrix
```

ped for serial motion recognition and tracking, using C++ and IMU wireless Motion Trackers. This
SW R & D project. Reconstruction of motion on Avatar controlled using quaternions and visual
quaternions. This current version of OpenIA includes Motion-sphere, which is a novel trajectory-
1 technique to represent human motion on a sphere.

FIGURE 1.2: OpenIA Project clone window



```
gh repo clone younghochai/motion-matrix
```

FIGURE 1.3: OpenIA Project clone options

Licenses

This work is dual-licensed under BSD-3 and Apache License 2.0. You can choose between one of them if you use this work. For more detailed description of each license follow the links.

- BSD-3: https://choosealicense.com/licenses/bsd-3-clause/

- Apache License 2.0: https://choosealicense.com/licenses/apache-2.0/

Prerequisites

OpenIA is implimented in C++. Various 3rd party APIs and tools are used within the motion sphere. A list of all those pre-requisite tools and libraries are given in the table 1.1

TABLE 1.1: Software Requirements

Likewise there are some specific hardware requirements for capturing the human motion. Table 1.2 details the hardware that is used to capture human motion. it must be noted that all the software requirements are must have requirements. Whereas the minimal hardware is the computing system. While Xsense and LiDARs are used for motion capture feature. Even without the Xsense IMU sensors and the LiDAR the OpenIA can be used to visualize the precaptured motion data.

TABLE 1.2: Hardware requirements

Building the project

Once the project is cloned on the local system. The project has the following folder tree structure.

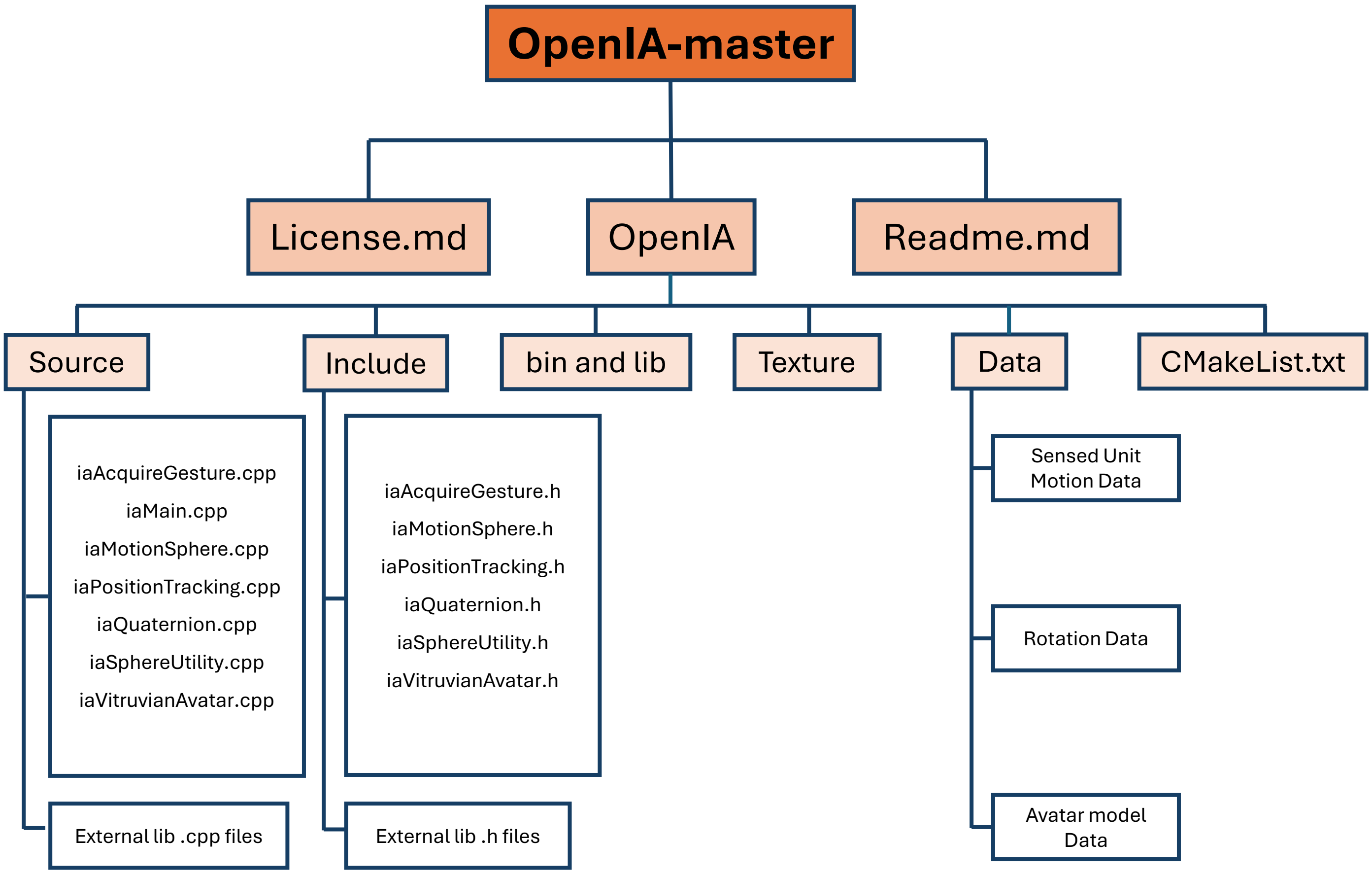


FIGURE 1.4: OpenIA Project folder structure tree in Github repository

***Note**: The folder structure may be slightly different depending on the version of the OpenIA, but the procedure to build the project remains the same.*

The Cmake tool is used to build the solution as shown in Figure 1.5. The solution file thus created in a separate build folder can be used for building the project that can be executed.

FIGURE 1.5: OpenIA Project folder structure tree in Github repository

Execution and Running

Building the project is as simple as selecting INSTALL in the folder structure window of the visual studio and executing build by right clicking (Figure 1.6).

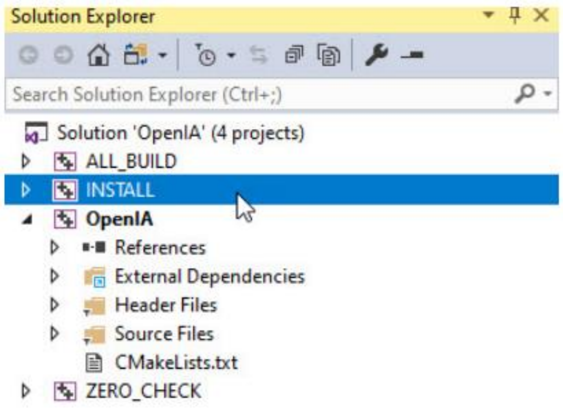


FIGURE 1.6: Building the Project from the Visual Studio.

The project can be run after the build is completed. Three windows appear, one with the 3D avatar that is rendered from the Visualization Tool Kit (Figure 1.7), second a Heterogeneous sensing motion capture window (Figure 1.8) and the other is a Unit Sphere rendered from the OpenGL renderer window(Figure 1.9).

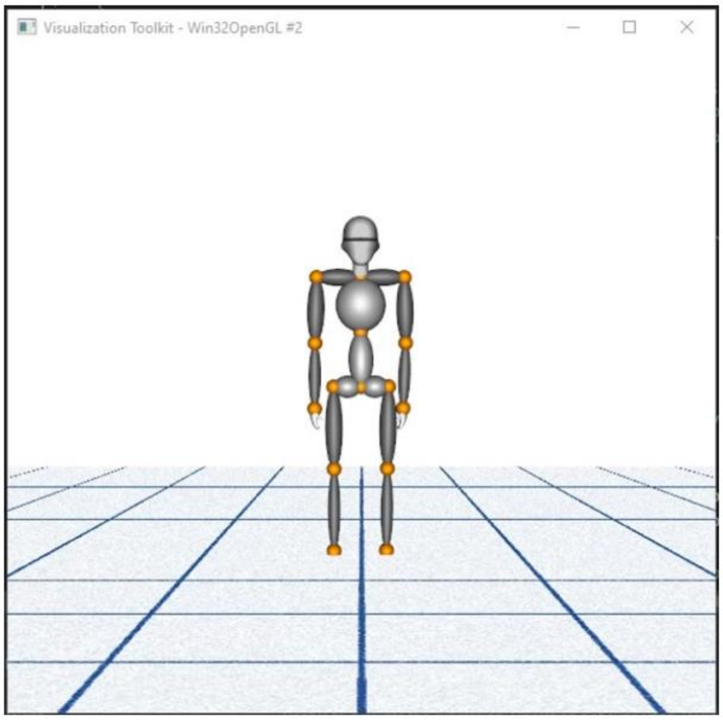


FIGURE 1.7: The first opening window of the OpenIA: 3D Avatar.

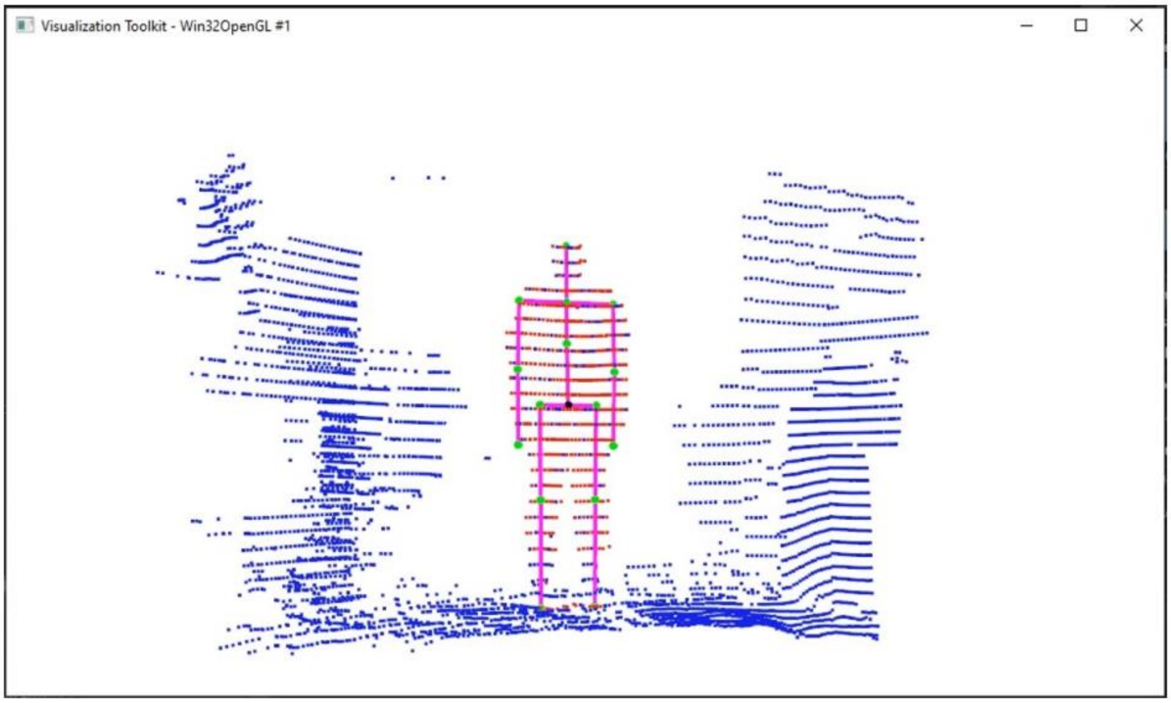


FIGURE 1.8: The second opening window of the OpenIA: Heterogeneous Sensing.

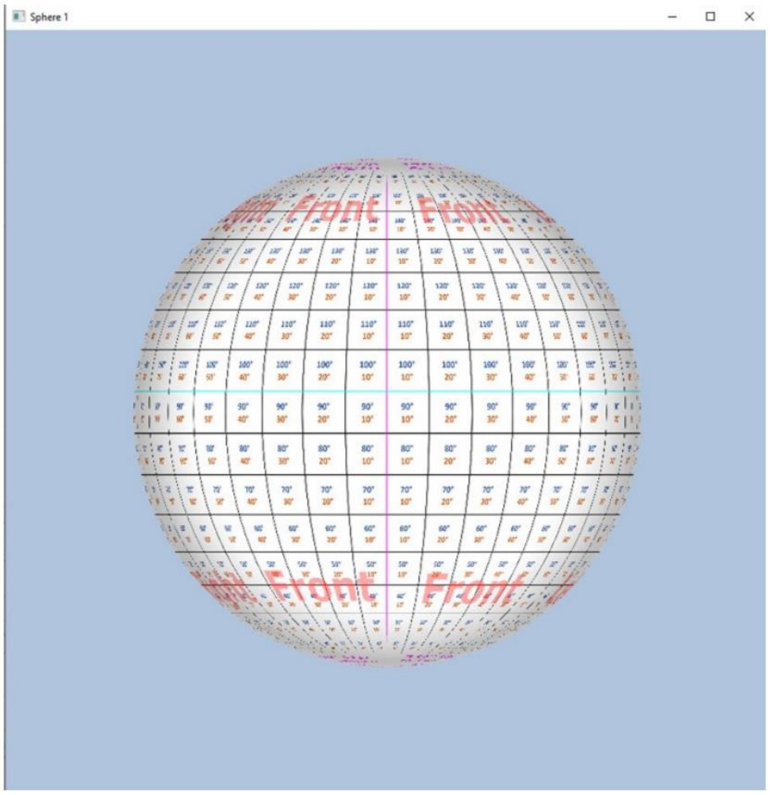


FIGURE 1.9: The third opening window of the OpenIA: Motion-Sphere.

1. 실행
    1-1. vtk 창 - 작은 아바타 출력됨 ***<< thread t2***
    1-2. pcl 창 - 크게 흰색 창 출력됨. ***<< thread t3***
    1-3. cmd 창 - 로그 확인

2. IMU 센서 연결과 Lidar 센서 연결 확인
    2-1. IMU 센서는 usb 포트로 스테이션 연결하거나 동글 꽂아서 연결 확인하기. ***<< thread t1***
    2-2. 라이다 센서 연결 방법은 별첨한 자료 참고
3. 연결 & 센서 준비 완료 확인(IMU 센서 연결되는 갯수 확인 + pcl창에 시각화 진행되는지)

----

4. 키보드 입력: i - 라이다 센서에서 공간에 대한 정보를 캡쳐하여 저장.
    4-1. 이 때, 모션 캡쳐할 사람 없이 배경에 대한 정보만 저장할 수 있도록 함
    4-2. 360도 정보를 전부 저장하는 것이 아닌 시야각을 좁혀서(센서 가려서) 저장하는 것을 권장.

5. 키보드 입력: s - 스캔된 센서들 페어링 및 라이다 센서의 해상도? 정보 저장.
    5-1. 키보드 입력 후 약 3~5초 대기. 이후 페어링 완료되면 메세지 출력됨.

***** 6번과 7번의 순서는 변경되어도 무방함*****

6. 키보드 입력: b & v - IMU센서의 heading reset & pose calibration 진행.
    6-1. b 입력으로 헤딩 리셋 진행 후 pose calibration 진행. 이후 사용자 신체에 부착한 후
        calibration 한번더 진행. 해당 용어 모르면 주변의 선배들에게 도움 요청하기

7. 키보드 입력: d - 캡쳐할 사람이 라이다 센서 앞에서 선 후 키를 입력하여 pcl에서 스켈레톤을 추출.

***** 6번과 7번의 순서는 변경되어도 무방함*****
8. 키보드 입력: a - 아바타 움직임 시작
9. 키보드 입력: n - 레코딩 시작 >> imu 센서 입력에 따라 pcl과 imu의 아바타 모두 움직임

main.cpp

```
thread t1(XSensDataReader); - IMU 센서 연결 후 데이터 읽어주는것 (from 'iaAcquire.cpp')

thread t2(startAvatar); - 아바타(스틱맨) 창 띄워 주는거 (from iaVitruvianAvatar.cpp)

thread t3(getOSLiDARdata); - pcl 라이브러리 띄우고 라이다 관련(from iaPositionTracking.cpp)
thread t5(poseTracking); - 진짜 포즈 트래킹 관련(from iaPositionTracking.cpp)
//thread t6(motionSphere); - 에러로 비활성화.
thread t7(saveIMUData); - imu 데이터 저장관련 (from iaPositionTracking.cpp)


iaPositionTracking<< 사실상 애가 거의 모든 코드 다 들고 있음
        >> 여기에서 출발해서 다른 파일 참조하는것이 대부분
```

iaAcquireGesture.cpp

```
...

...

void iaAquireGesture::getXsensData()
{
        xsens에서 연결 대기 풀고 running 상태로 변환
        imu 센서의 데이터 10개를 담을 변수를 초기화(qInit *10개)
}

void iaAcquireGesture::resetIMUSensor()
{
        connectXS 네임스페이스를 참조해서 imu 센서의 heading reset을 진행.
}
```

iaVitruvianAvatar.cpp

```
...

class vtkTimerCallback : public vtkCommand
{

        ...

        ...

        virtual void Execute(뭔가 복잡해보이는 parameter들)
        {

                if (VitruvianAvatar::isLoaded)
                {
                        roatateAvatar(VitruvianAvatar::vitruvianAvatarUpdate) // 아바타 움직이게 함
                }
        }

}
```

iaPositionTracking.cpp

```
keyboardEventOccurred()
{
        i: PositionTracking::_initLidar = true;

        s: (1) myAcquire.getXsensData();
           (2) PositionTracking::_StartScan = true;

        b: myAcquire.resetIMUSensor();

        v: iaAcquireGesture::calibIMU = true;

        d: PositionTracking::initCalib = !PositionTracking::initCalib;

        a: VitruvianAvatar::isLoaded = true;

        n: PositionTracking::recordData = !PositionTracking::recordData;
                writeJointData(PositionTracking::tFrameIndex);
                PositionTracking::saveSFQuatData(PositionTracking::tFrameIndex);<< 왜 두개를 같이 쓰는지...

                XsensConnection::ukeyPressed = !XsensConnection::ukeyPressed;<< 얘가 메인
}

void writeJointData(int tIndex)
{       //txt 파일로 현재 움직이고 있는 아바타의 데이터들을 저장함 }


void PositionTracking::OSlidar()
{
        //라이다 센서 초기 세팅. 시리얼 넘버 및 포트 선언하는 부분. 라이다 센서 세팅(노션페이지) 참고
        // 센서 연결 및 연결 테스트
        ...

        if (PositionTracking::_initLidar)
        {       //현재 pcl 데이터 값을 저장하여 pcd 파일로 저장하기}
        ...

        if (_StartScan && flag1timeL1)
        {
                cout << "Max truth-> X:" << {Max truth 값 불러와서 출력}<< endl;
                        cout << "Ground truth-> X:" << {Ground truth 값 불러와서 출력}<< endl;
        }
}

void PositionTracking::positionDetection(VitruvianAvatar& vAvatar) // 이거 진짜 길이 개 길다
{
        ...

        while(!viewer -> wasStopped()) // 이게 멈추기 전까지...
        {
                ...
                if(PositionTracking::initCalib)// 이게 트루일 경우
                {
                        //pcl에서 사람을 찾아서 아바타를 그림(아마도)
                        if(myAcquire.caliIMU)
                        {
                                myAcquire.caliberateQSF();
                                AcquireSFQ.caliberateQSF();
                        }
                }
        }


}
void PositionTracking::saveSFQuatData(int noOfFrames)
{       //txt 파일로 현재 움직이고 있는 아바타의 데이터들을 저장함. }
```

Primary Modules

Motion sphere is composed of the following primary modules. 1) iaSphereUtility 2) iaQuaternion 3) iaMotionSphere 4) iaAcquireGesture 5) iaVitruvianAvatar 6) iaPositionTracking

Class Description

Figure 1.10 shows a description of all the C++ structures that is used in the OpenIA. The struct Avatar is the main data structure that is used throughout the code base for accessing the data of a motion frame. A motion frame consists of orientation data of 10 bone segments starting from b0–b9 in form of quaternions.

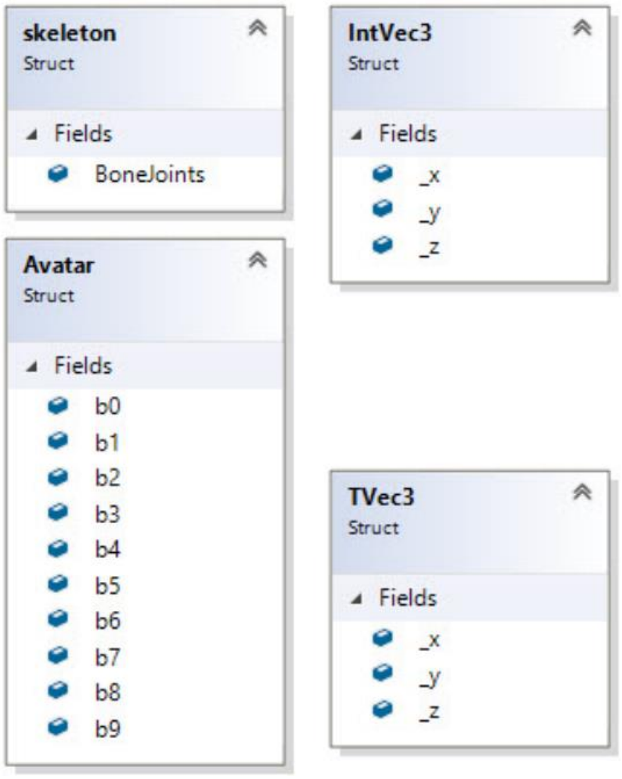


FIGURE 1.10: C++ Structures that are handy in OpenIA

Figure 1.11 shows the description of all the classes in the OpenIA tool's code base.

iaMain

The iaMain.cpp is the function called when the program is run. The execution of all class functions begins in main function with multiple threads and have unique id for each class functions.

iaPositionTracking

The PositionTracking uses the point cloud library and the LiDAR to compute the subject's height. various point cloud trasformations like the segmentation of point cloud, the estimation of pelvis position, saving the quaternions into files, updating the bone segment data in real time and so on are implimented in this class.

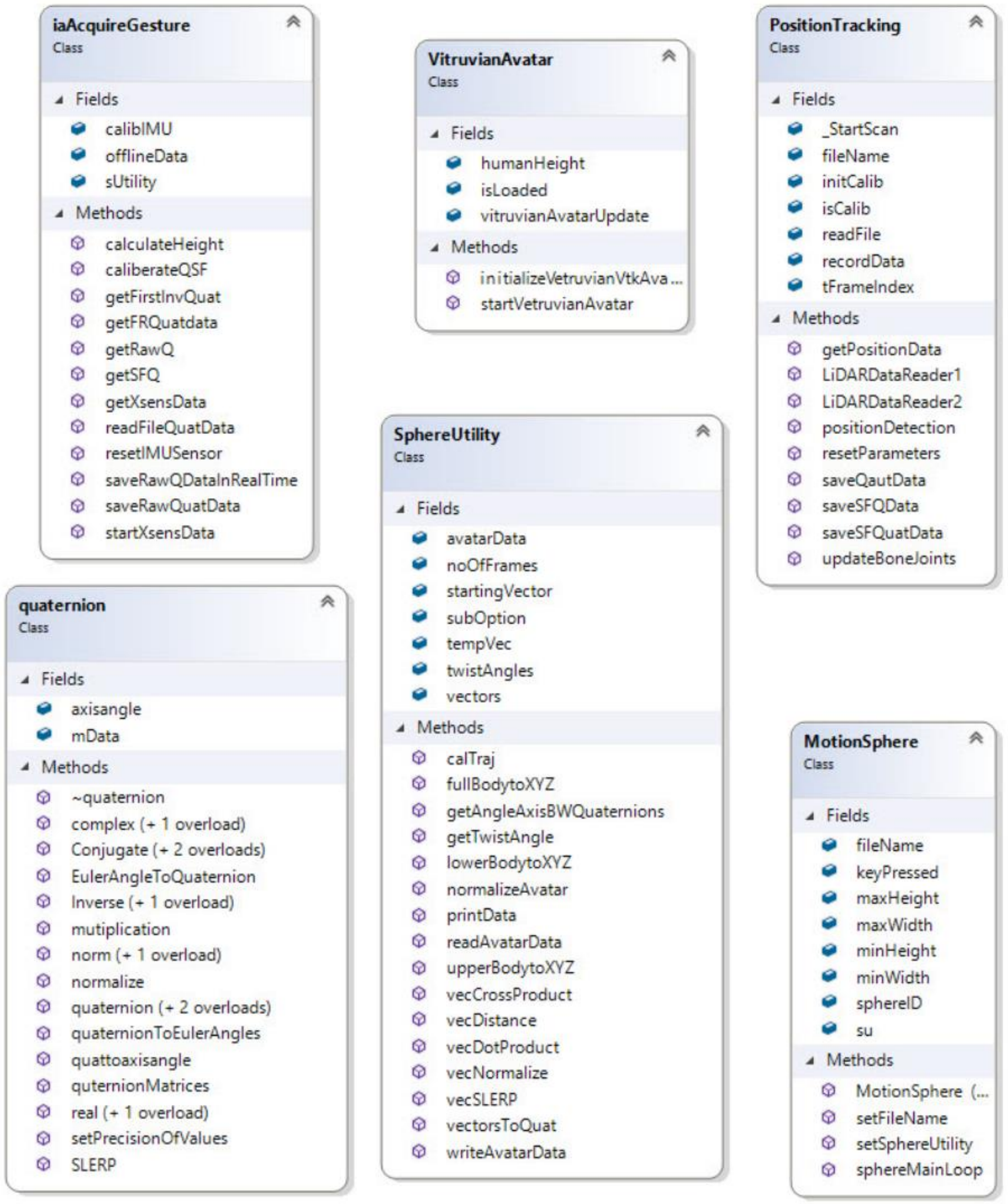


FIGURE 1.11: Class diagram of all the modules in the OpenIA

iaAquireGesture

The iaAquireGesture mainly impliments the motion capture functionality. The class starts the xsense IMU sensors (startXsenseData()), gets raw quaternions from the sensors (getRawQ), calibrates the 3dAvatar (calibrateQSF()), and applies a series of quaternion transformations to save the motion frame in a standard file format.

iaSphereUtility

Contains mathematical operations including vector and quaternion algebra essentials for visualizing the human motion as 3D points. SphereUtility class implements methods not limited to readAvatarData to read from the captured file, fullBodyXYZ to convert all the quaternion frames into 3D points, and calTraj to calculate the trajectory based on kinematic hierarchy.

iaQuaternion

All the quaternion operations are handled in the class quaternion. Operations like the converting of quaternions to euler angles, normalizing, quaternion multiplication, conversion of quaternions into axis and angles, spherical linear interpolation, inverse and so on.

iaVitruvianAvatar

VitruvianAvatar is the VTK (Visualization Tool Kit) implimentation of the 3D avatar. The avatar exposes a attribute called the vitruvianAvatarUpdate, which is of type struct Avatar. This attribute can be updated in realtime to see the animation of the avatar as and when the data is captured or rendered by the iaAquaireGesture or the MotionSphere modules respectively. The Avatar is initialized and triggered using the methods exposes in this class.

iaMotionSphere

MotionSphere is the main module that consists of a GLUT main loop. When the sphereMainLoop() is called the tool enters into a infinite event based loop. The MotionSphere reads the data with the help of the other utility classes and renders the visualization on the unit sphere. Mainly it holds the window parameters, an instance of SphereUtility to operate all the visualization functions that are implimented using OpenGL.

Heterogeneous sensing system

In this project work, a human motion tracking system that uses lidar and IMU sensors to estimate 3D human pose in real-time. Human motion tracking includes human detection, height and skeleton parameters estimation, position, and orientation estimation by fusing lidar-IMU sensor data. Finally, reconstructing the human motion on a virtual 3D avatar. The flow of the proposed system is shown in Figure 1.13.

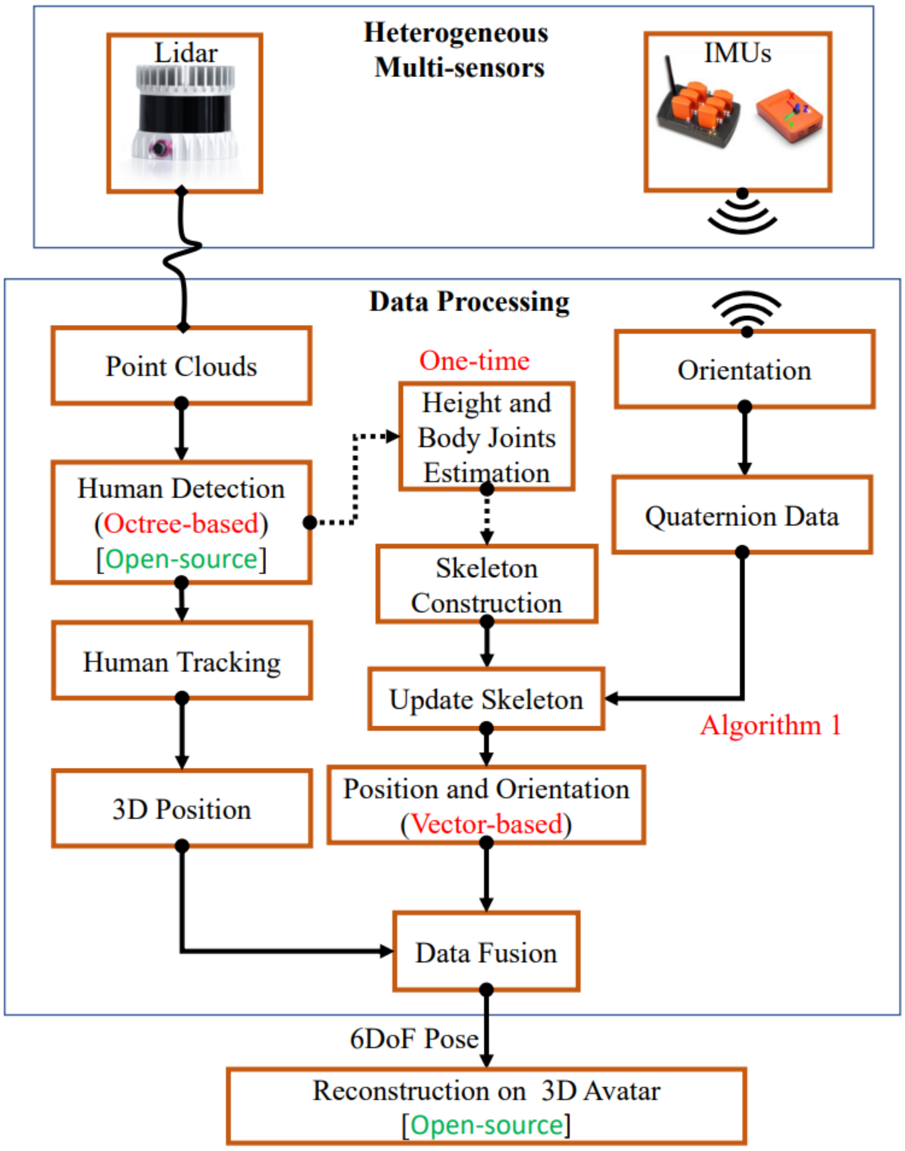

Lidar
Heterogeneous Multi-sensors
IMUs
Data Processing
Point Clouds
One-time
Height and Body Joints Estimation
Orientation
Human Detection (Octree-based) [Open-source]
Quaternion Data
Skeleton Construction
Human Tracking
Update Skeleton
Algorithm 1
3D Position
Position and Orientation (Vector-based)
Data Fusion
6DoF Pose
Reconstruction on 3D Avatar [Open-source]

FIGURE 1.12: Heterogeneous sensing flowchart

Motion Sphere

Motion-sphere is a trajectory-based visualization technique to represent human motion on a unit sphere. Motion-sphere adopts a two-fold approach for human motion visualization, namely a 3D avatar to reconstruct the target motion and an interactive 3D unit sphere, that enables users to perceive subtle human motion as swing trajectories and color-coded miniature 3D models for twist.

Any human motion is a sequence of quaternion rotations, represented visually as a trajectory on a 3D unit sphere. Figure 1.13 shows an overview of the proposed Motion-sphere. A 3D LiDAR sensor is used to track the position of a moving human body. The 3D position of the pelvis is computed using the segmentation of raw point cloud data based on the user's height and IMU sensors are used to estimate the orientation of each bone joints during human body motion in a real environment. When a user performs a motion, orientation data from the IMU sensors are transformed into a 3D avatar's coordinate frame for reconstruction.

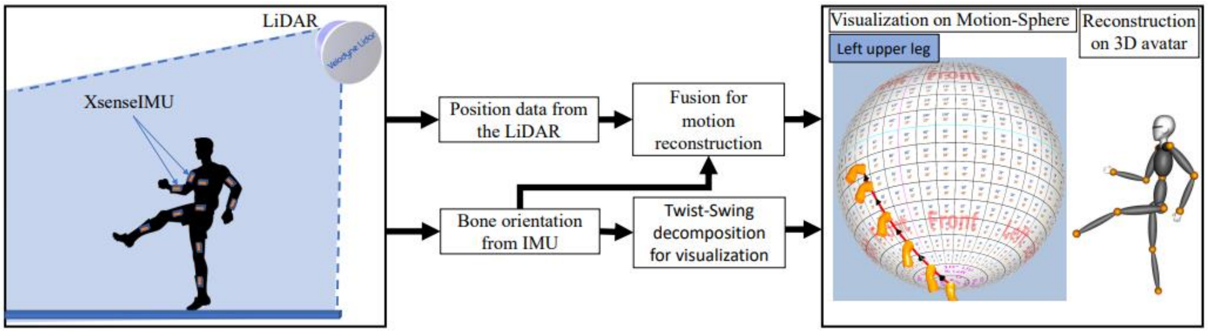

LiDAR
XsenseIMU
Position data from the LiDAR
Fusion for motion reconstruction
Bone orientation from IMU
Twist-Swing decomposition for visualization
Visualization on Motion-Sphere
Left upper leg
Reconstruction on 3D avatar

FIGURE 1.13: Overview of the Motion-sphere

OpenIA 3.0 release

iaMotionSphere

Real Time Pose Tracking

User Interface
There is a frame editing function which is utilized by keyboard and mouse interaction. (Up and down, right and left, and scroll) We want to observe ~~~

We adopt the library for user interface named ImGUI.

- Edit Mode (Spherical / Quaternion)
    - Spherical : Edit function on spherical coordinate. You can modify the longitude and latitude of each points.
    - Quaternion : Edit function about the quaternion of each points. You can modify x, y, z value.
- Magnitude : You can adjust
- Previous Index / Next Index : You can choose
- Save
- Undo
- Reset

iaQuaternion

Additional operators about addition and subtraction of quaternion.

- addition
- subtraction