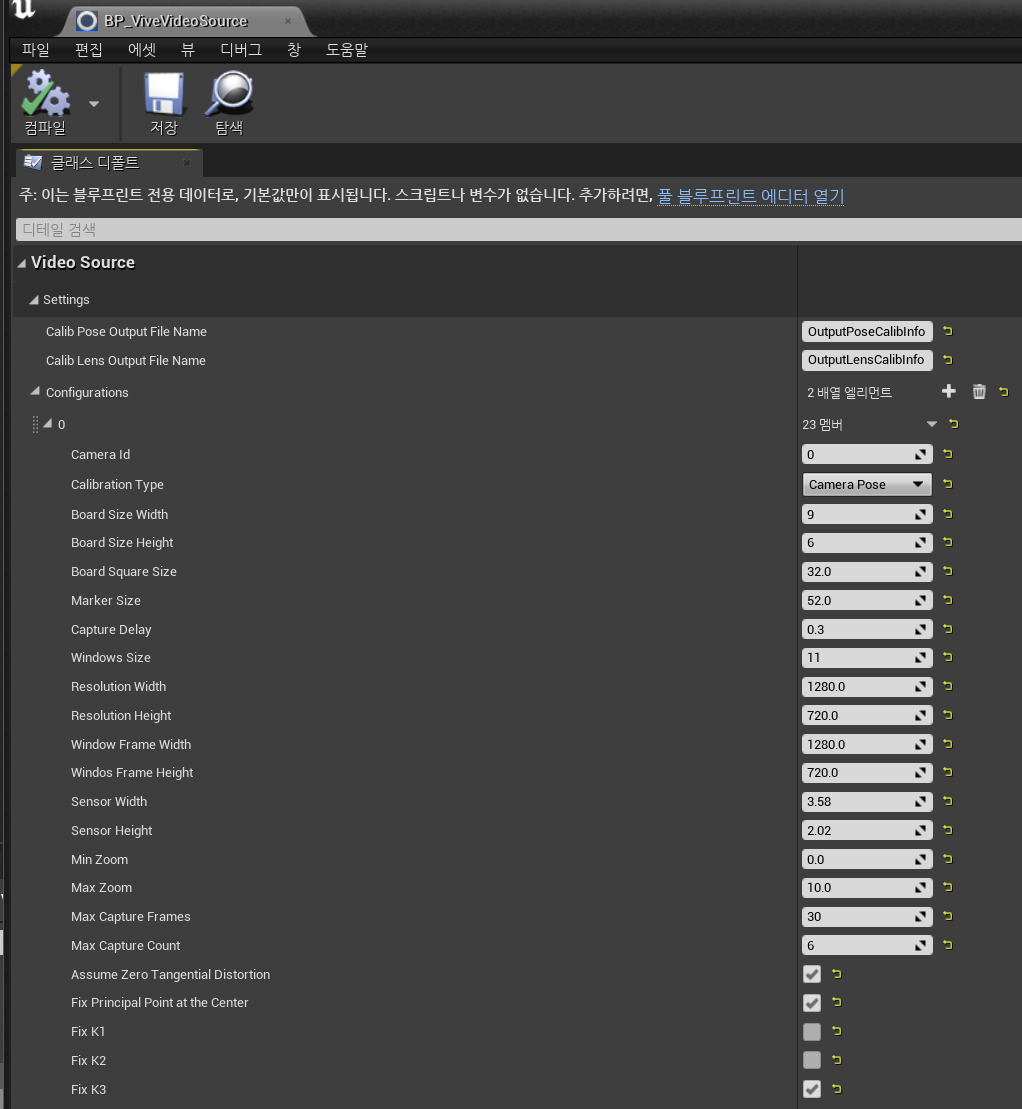
ViveOpenCV 문서

1. **ViveOpenCV Plugin**

: OpenCV 기능을 Unreal Engine에서 사용 할 수 있고, OpenCV를 이용하여 실제 Camera Transform을 구하고, Camera Lens Calibration 기능을 제공 하는 플러그인.

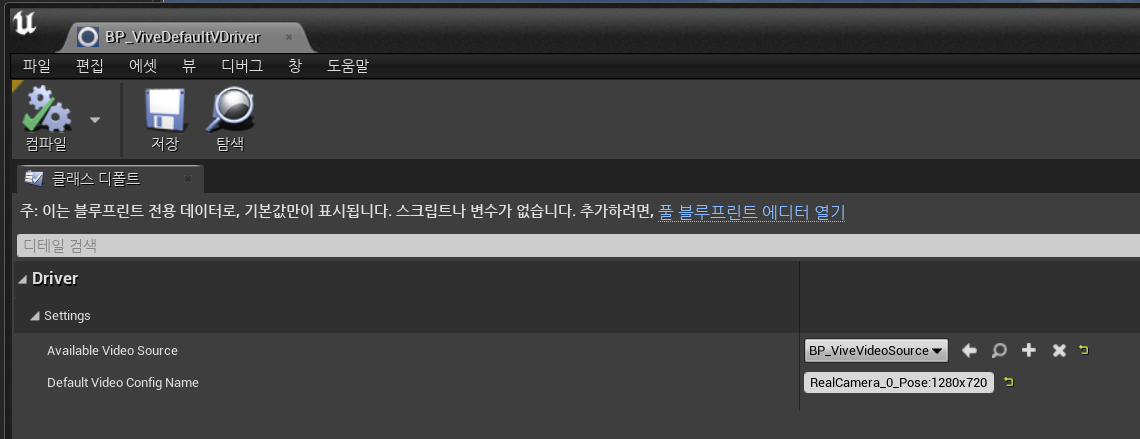
1. **OpenCV로 Camera Position Calibration 하는 방법**

Plugins\ViveOpenCVPlugin\Content\Blueprints 폴더에 BP\_ViveDefaultDriver와 BP\_ViveVideoSource 어셋이 있다. BP\_ViveVideoSouce 어셋은 OpenCV를 사용하여 Calibration 하기 위한 카메라 및 패턴 이미지에 대한 정보를 담고 있는 어셋이다.

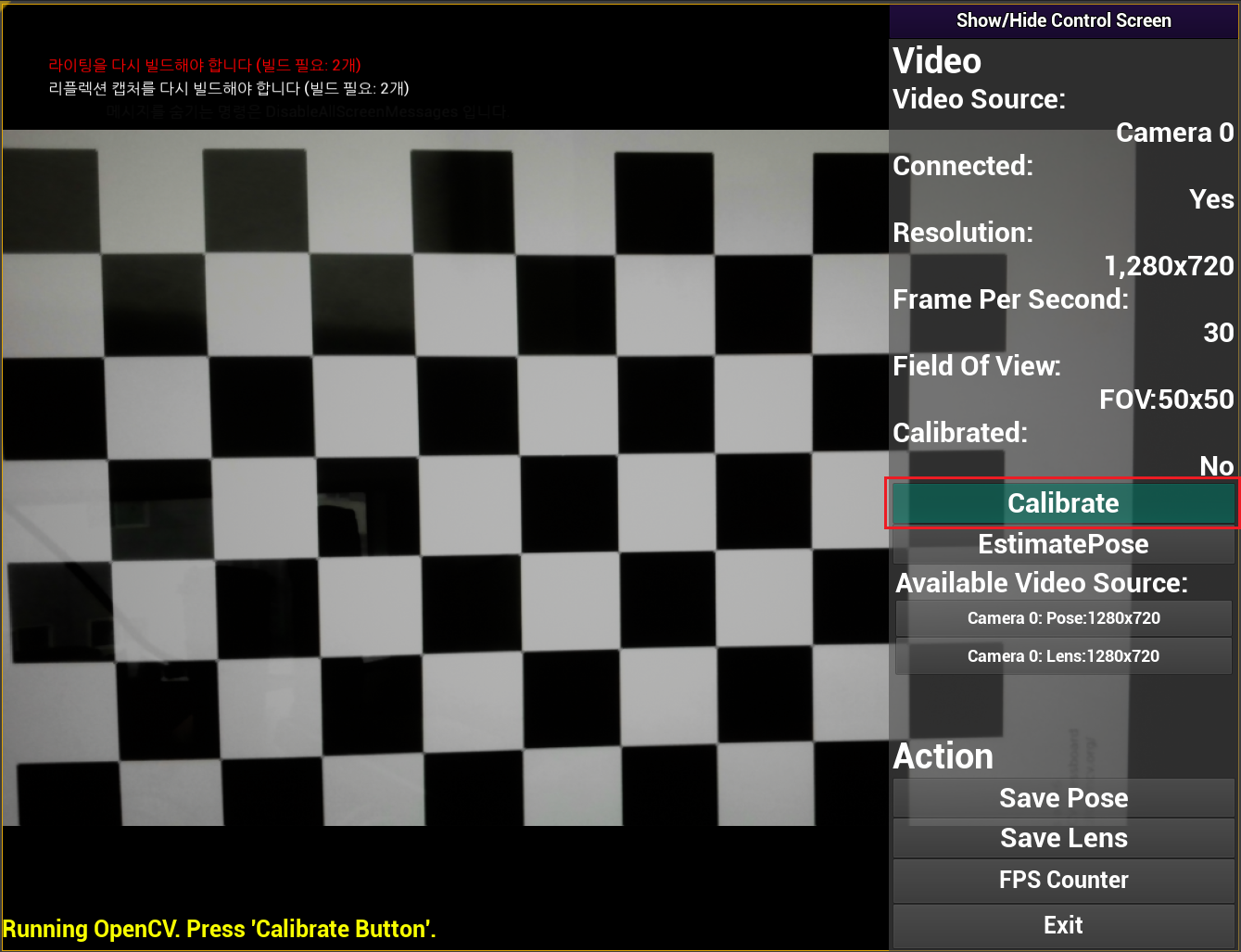


그리고 Calibration 된 정보를 외부에 저장하기 위한 파일 이름을 설정하는 내용도 있다.(현재 BP\_ViveVideoSource에는 Camera Position과 Camera Lens에 대한 Calibration에 대한 예시 설정 정보가 입력되어져 있다. 사용하기전 사용하는 카메라에 대한 정보 및 기타 설정을 해야 한다.)

BP\_ViveDefaultDriver 어셋은 BP\_ViveVideoSource에 설정된 정보 중 어떤 것을 사용할 것인지 선택하고, 또 다른 VideoSource 어셋이 있다면 어떤 VideoSource 어셋을 사용할 것인지 선택 할 수 있다.

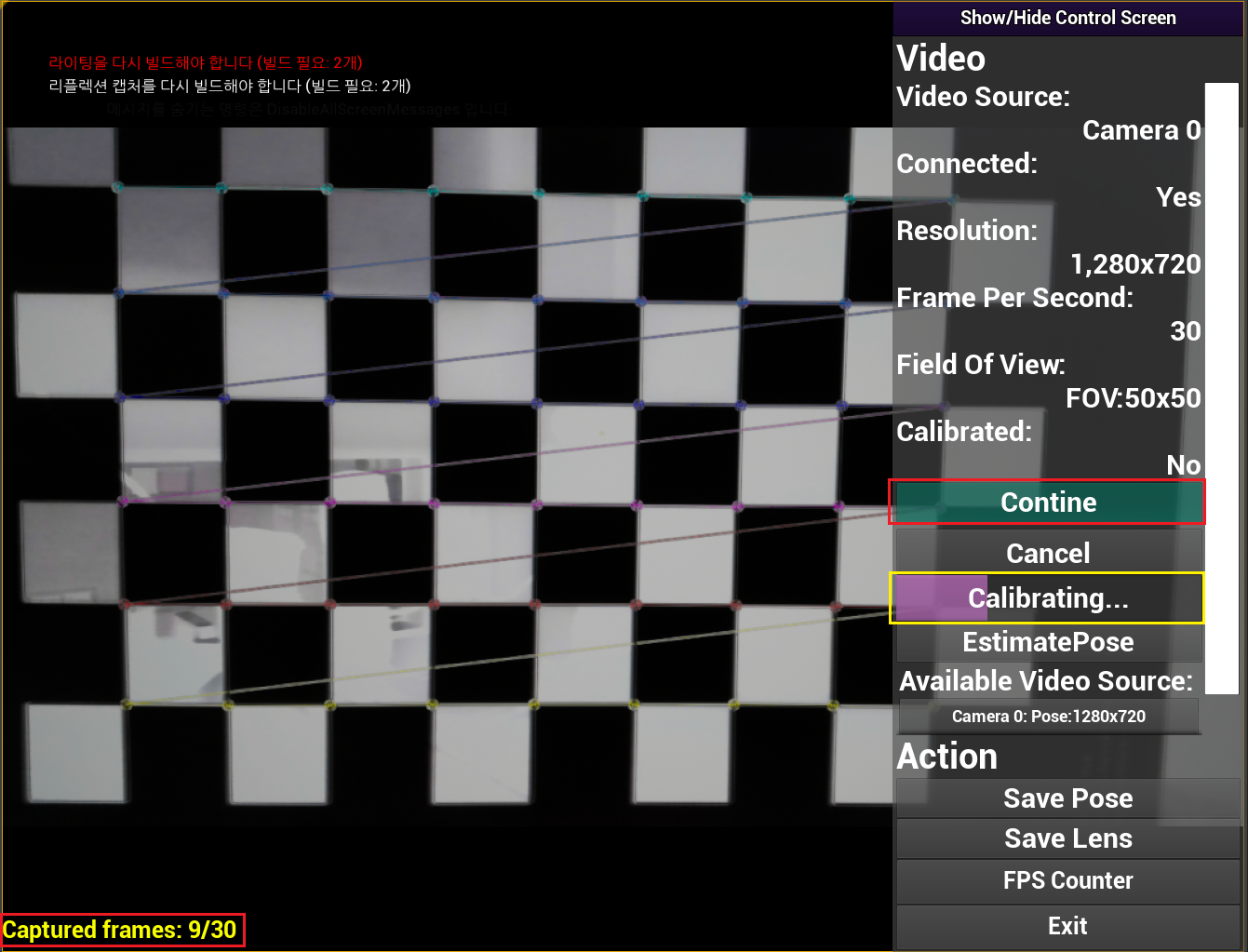


설정 정보 입력이 완료 되었다면 CameraCalib 맵을 실행하면 다음 그림과 같이 카메라가 보는 영상이 화면에 나타난다.

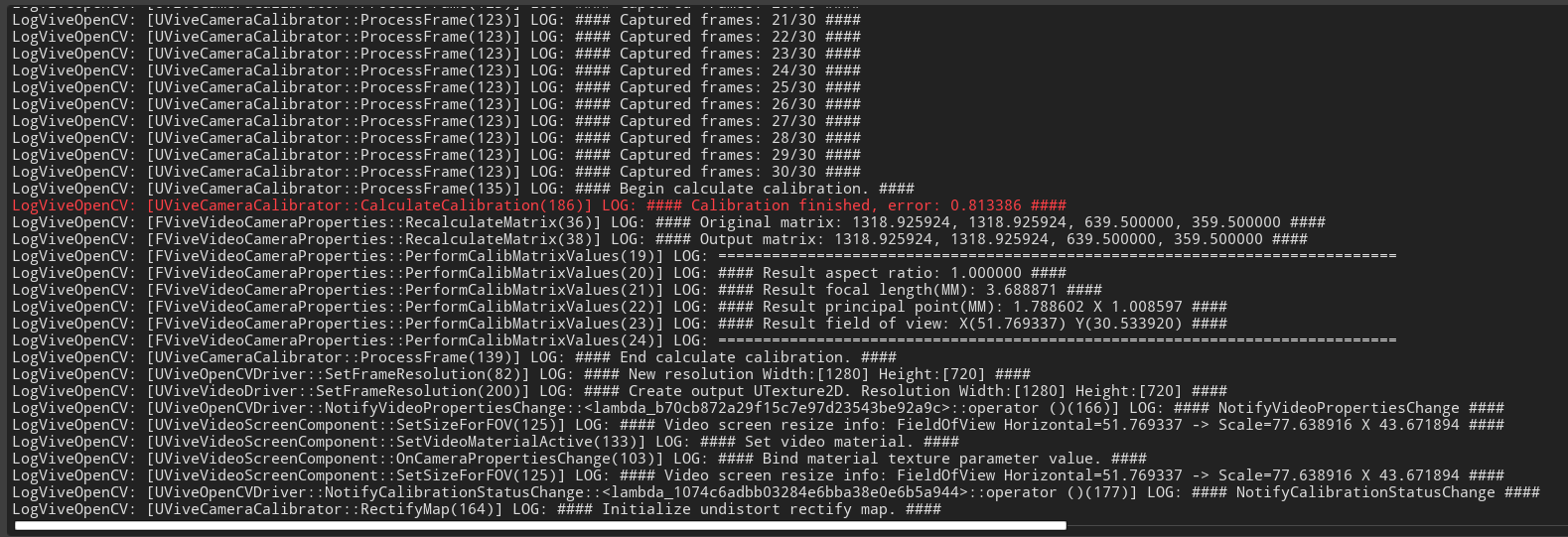


Camera Calibration을 위한 패턴 이미지를 카메라로 촬영하면서 Calibrate 버튼을 누르게 되면 Camera Calibration 설정이 시작된다.

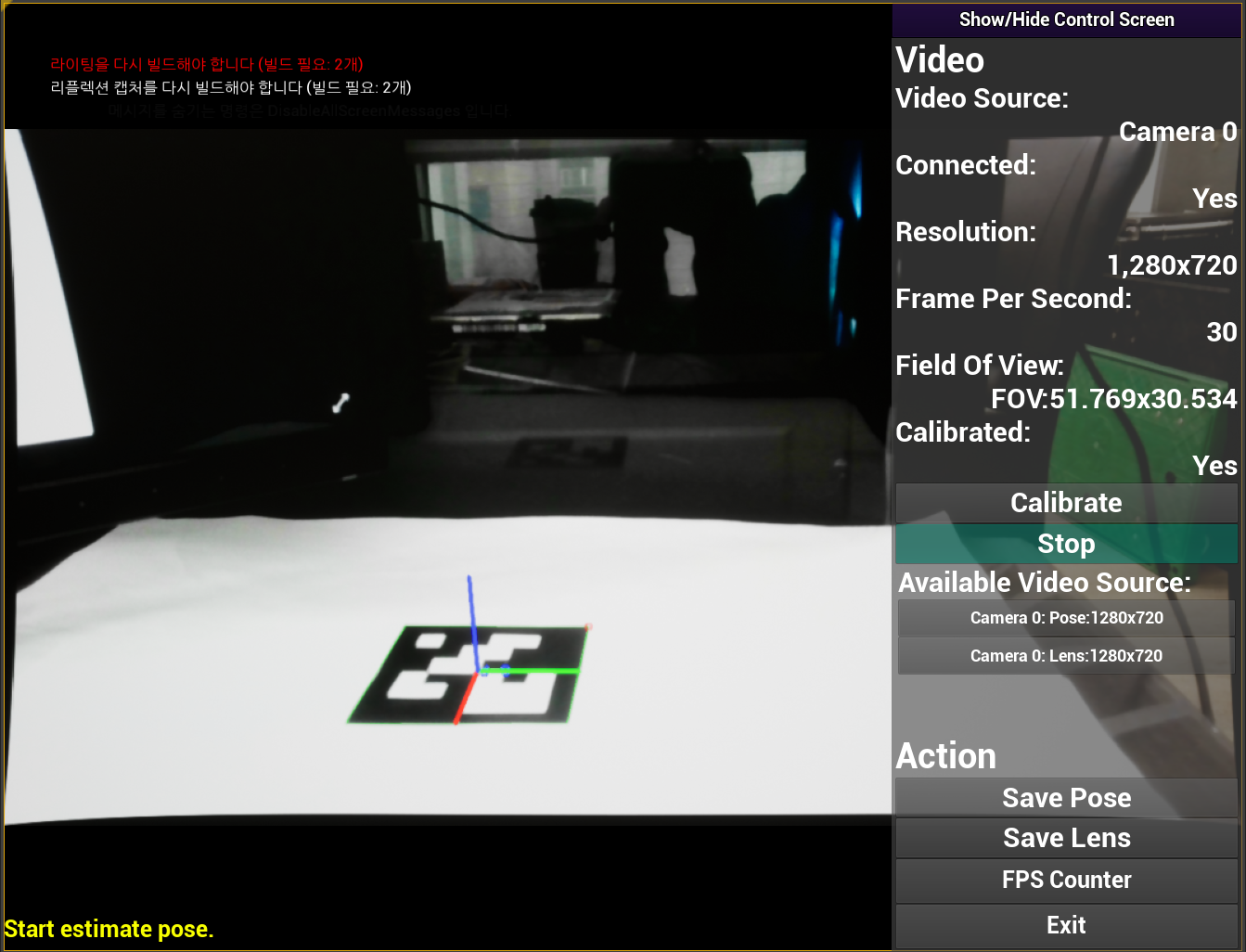
(패턴 이미지는 Plugins\ViveOpenCVPlugin\OpenCVImages 폴더에 있다.)



Continue 버튼을 누르면서 설정된 Captured Frames을 채운다. 한 번 누를 때 마다 일정한 Frame이 캡처된다.(노란색 박스로 현재 Capture 되는 상태를 알 수 있다)

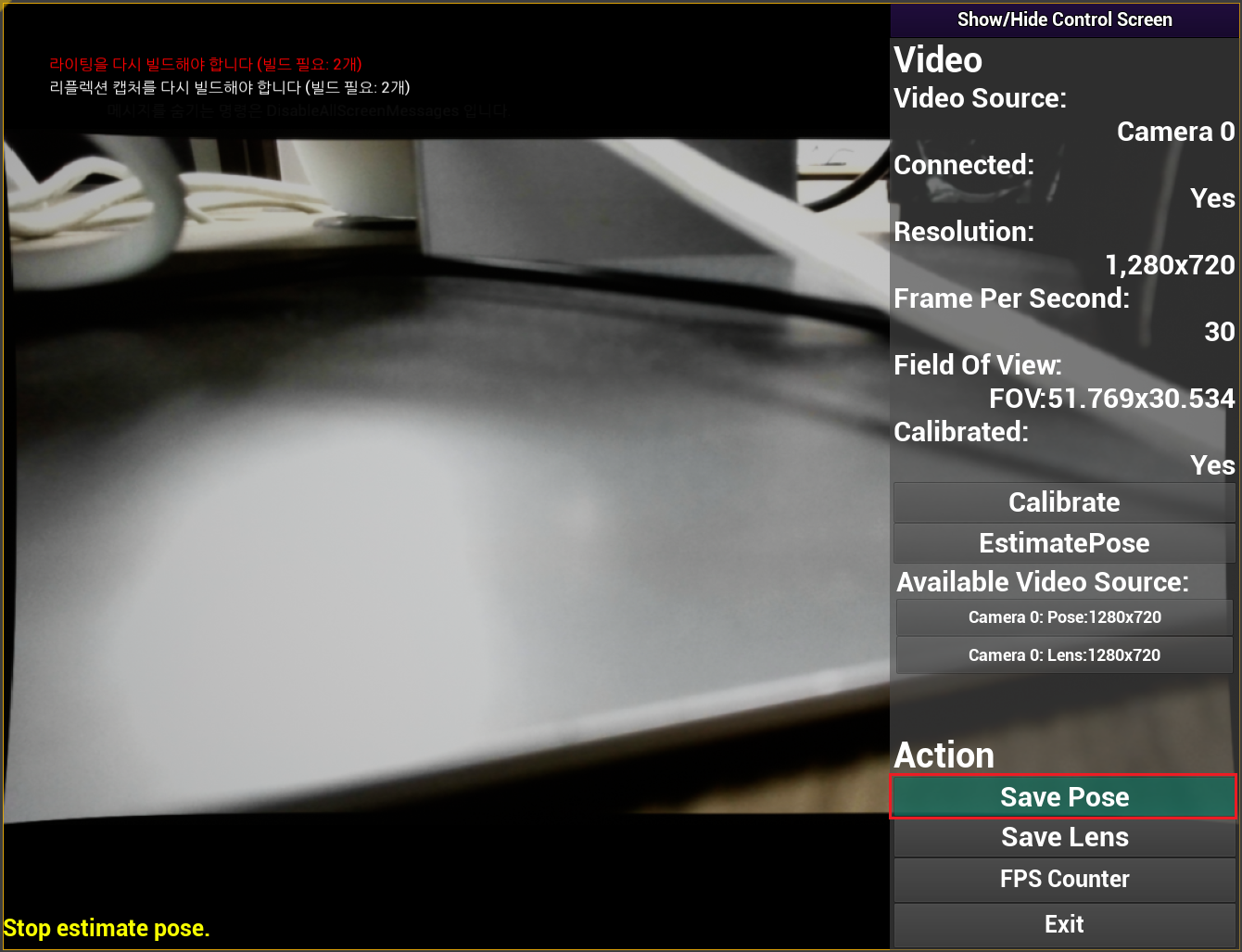


Calibration이 끝나면 로그에 Calibration된 결과 정보를 확인 할 수 있다.

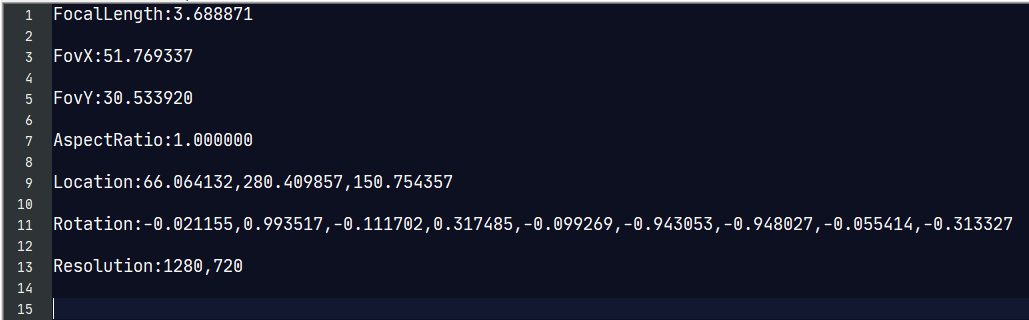


Camera Calibration이 끝나면 결과로 만들어진 정보를 바탕으로 가상 카메라 위치를 계산 할 수 있는데, 가상 카메라 위치를 계산하기 위해 EsimatePose 버튼을 누르면 위 그림처럼 Mark 이미지 위에 좌표 그림이 나타나게 된다. 이로써 가상 카메라 위치에 대한 결과를 만들어진다.

계산된 가상 카메라 위치에 대한 정보를 외부 파일에 저장하기 위해 Save Pose 버튼으로 저장한다.

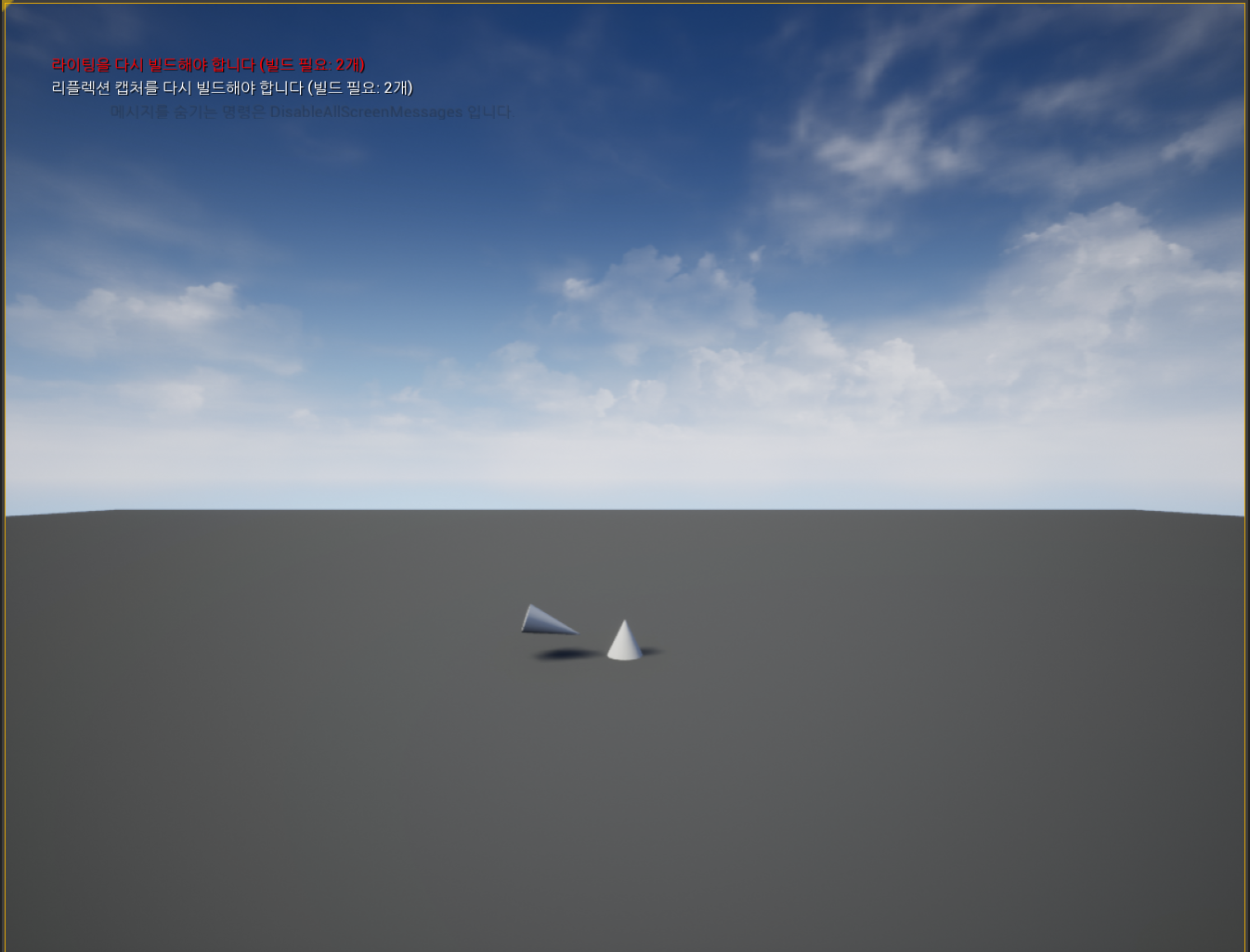


저장되는 내용은 다음 그림과 같다.



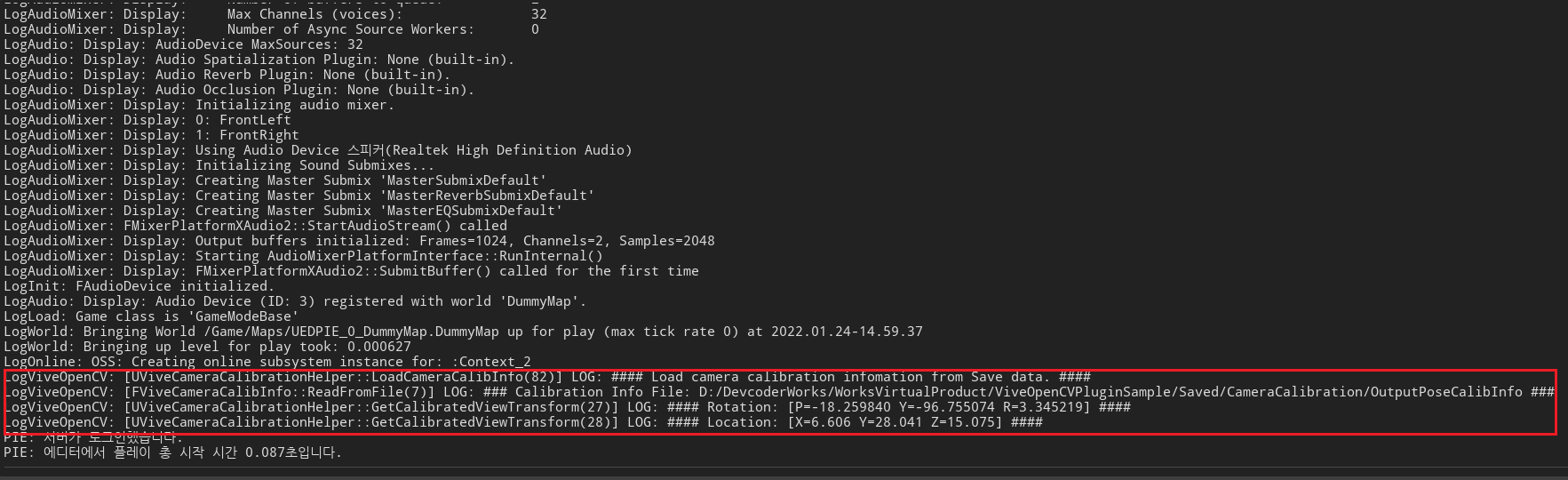
이렇게 저장된 결과를 이용해서 가상 카메라 위치가 제대로 계산되어졌는지 확인을 해보기 위한 절차는 다음과 같다.

DummyMap으로 변경한다. 그리고 실행하게 되면 다음 그림처럼 Calibration 할 때의 카메라 위치가 가상 카메라로 나타나게 된다.



수직으로 세워져 있는 원뿔이 Estimate Pose일 때 좌표 화살축의 지점을 나타낸다. 또 다른 원뿔이 Estimate Pose일 때 카메라 위치를 나타낸다.

위 그림에서 가상 카메라를 나타내는 원뿔의 Transform 정보가 다음 그림처럼 로그에 나와있다.

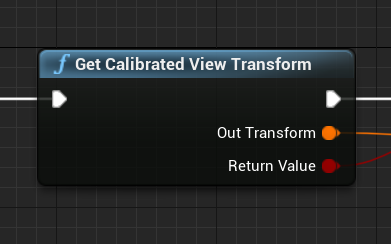


1. **API 정리**
2. **Vive Camera Calibrator**

OpenCV를 통해 구한 Camera Calibration Information 데이터로 Unreal Engine 가상 카메라(예: ACineCameraActor)의 Transform을 계산해 주는 Function Node.

**Out Transform:** 계산된 가상 카메라 엑터의 Transform.

**Return Value:** 정상적으로 계산되어 가상 카메라의 Transform 값을 얻었는지 여부.



OpenCV를 통해 구한 Camera Calibration Information 데이터로 Unreal Engine 가상 카메라(예: ACineCameraActor)의 Focal Length을 계산해 주는 Function Node

**Out Forcal Length:** 계산된 가상 카메라의 Focal Length

**Return Value:** Focal Length 값을 얻었는지 여부.



사용 중인 실제 카메라의 종횡비(Aspect Ratio)을 구하는 Function Node

**Out Aspect Ratio:** 실제 카메라 종횡비 값.

**Return Value:** 실제 카메라 종횡비를 얻었는지 여부.



OpenCV를 통해 구한 Camera Calibration Information 데이터로 Unreal Engine 가상 카메라의 X축 방향 Field Of View 값을 구하는 Function Node

**Out Fov X:** X축 방향 Field Of View 값

**Return Value:** X축 방향 Field Of View 값을 얻었는지 여부.



OpenCV를 통해 구한 Camera Calibration Information 데이터로 Unreal Engine 가상 카메라의 Y축 방향 Field Of View 값을 구하는 Function Node

**Out Fov Y:** Y축 방향 Field Of View 값

**Return Value:** Y축 방향 Field Of View 값을 얻었는지 여부.



1. **Vive Camera Tracker Helper**

참고: ViveVR Tracker는 ViveVR 로고가 보이는 면이 가리키는 방향이 Forward 방향이고, 전원 램프가 있는 방향이 Up가 됨.

밑에 구현한 Node Function들은 ViveVR Tracker가 ViveVR 로고가 보이는 면이 가리키는 방향이 Up, 전원 램프가 있는 방향을 Forward가 되게 설치한 상태로 구현

Tracker

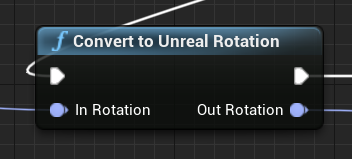
Camera

위에서 본 모습

ViveVR Tracker의 회전 값을 Unreal Engine의 회전 값으로 변환 하는 Function Node

**In Rotation:** ViveVR Tracker의 회전 값.

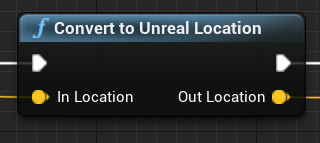
**Out Rotation:** Unreal Engine에서 사용 가능하도록 변환 된 회전 값.



ViveVR Tracker의 이동 값을 Unreal Engine의 이동 값으로 변환 하는 Function Node

**In Location:** ViveVR Tracker의 이동 값.

**Out Location:** Unreal Engine에서 사용 가능핟록 변환 된 이동 값.

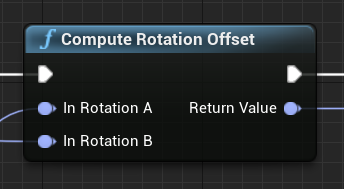


ViveVR Tracker의 회전된 값과 Unreal Engine 가상 카메라의 회전된 값의 차이를 구하는 Function Node

**In Rotation A:** Unreal Engine 가상 카메라 엑터의 현재 회전된 값.

**In Rotation B:** ViveVR Tracker의 현재 회전된 값.

**Return Value:** 두 회전 값의 차이 값.

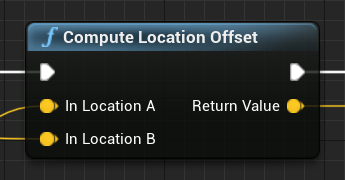


Vive VR Tracker의 이동된 값과 Unreal Engine 가상 카메라의 이동된 값의 차이를 구하는 Function Node

**In Location A:** Unreal Engine 가상 카메라 엑터의 현재 이동된 값.

**In Location B**: ViveVR Tracker의 현재 이동된 값.

**Return Value:** 두 이동 값의 차이 값.

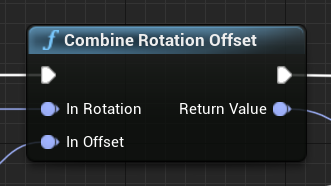


초기 시점에 구해진 회전 Offset 값과 현재 ViveVR Tracker 회전 값이 합쳐진 회전 값을 구하는 Function Node

**In Rotation:** ViveVR Tracker의 현재 회전 값

**In Offset:** Compute Rotation Offset으로 구한 Offset 회전 값.

**Return Value:** 두 회전 값이 합쳐진 회전 값.

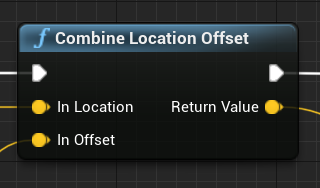


초기 시점에 구해진 이동 Offset 값과 현재 ViveVR Tracker 이동 값이 합쳐진 이동 값을 구하는 Function Node

**In Location:** ViveVR Tracker의 현재 이동 값.

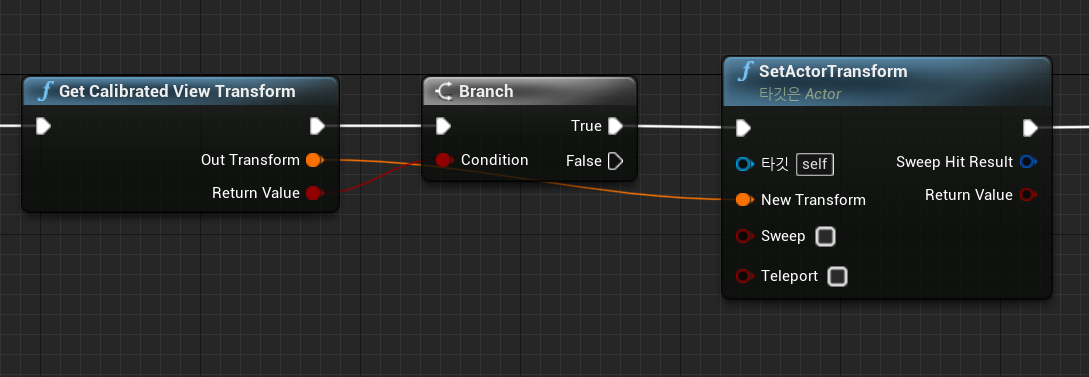
**In Offset:** Compute Location Offset으로 구한 Offset 이동 값.

**Return Value:** 두 이동 값이 합쳐진 이동 값.

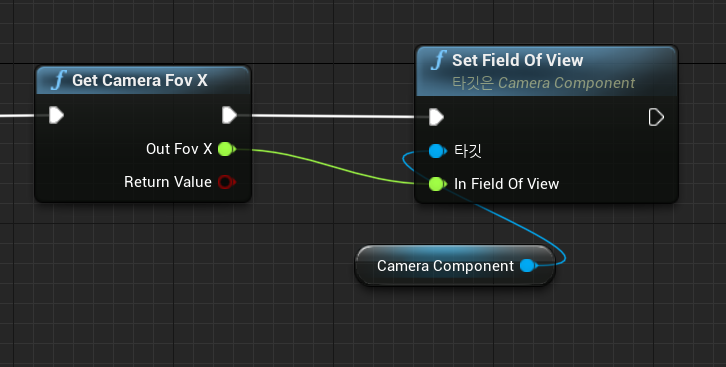


1. 기타 항목1
2. 기타 항목2
3. **Utility Function Node 사용예**
4. **Camera Calibration 설정 예제**

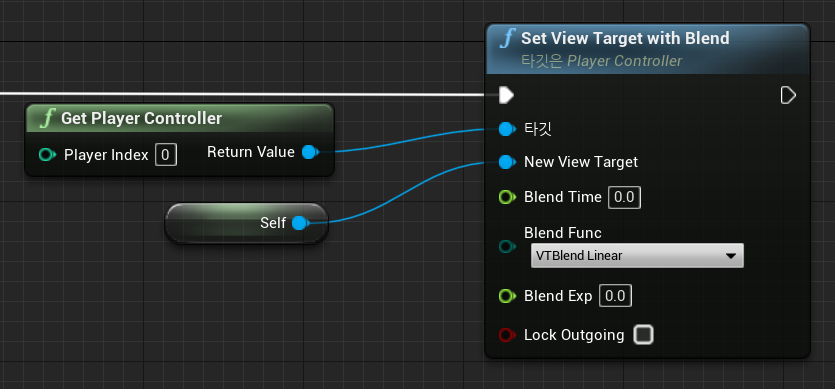
Unreal Engine 가상 카메라 엑터 초기 설정(BeginPlay 이벤트)에서 Camera Calibration View Trasform을 얻어와 가상 카메라 엑터 Trasform으로 설정 함.



Unreal Engine 가상 카메라 엑터 초기 설정(BeginPlay 이벤트)에서 가상 카메라의 Field Of View를 설정 함.

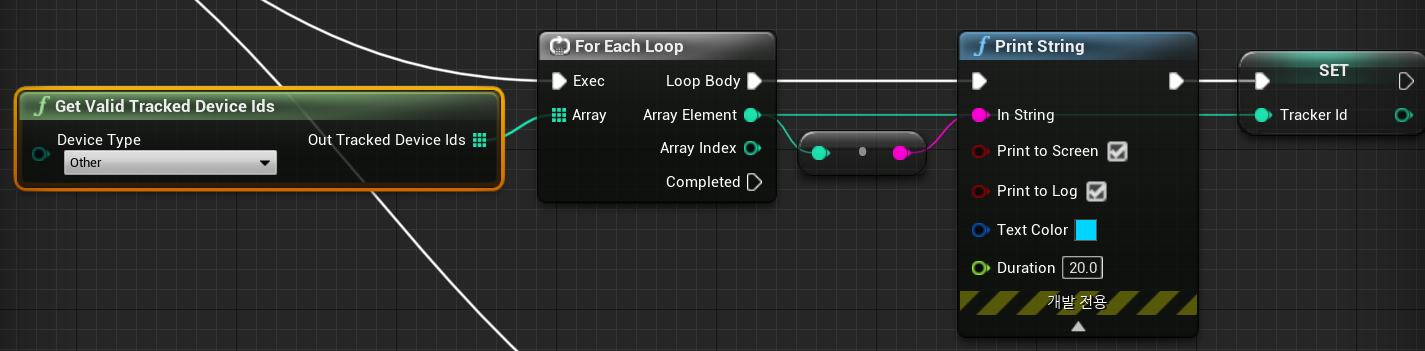


설정된 Unreal Engine 가상 카메라 엑터를 View Target으로 설정 함.



1. **ViveVR Tracker와 Unreal Engine 가상 카메라간 Transform 관계 설정**

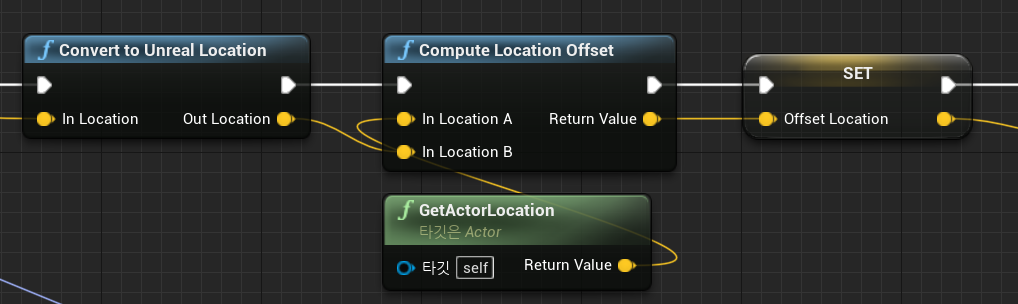
ViveVR Tracker ID를 얻음.



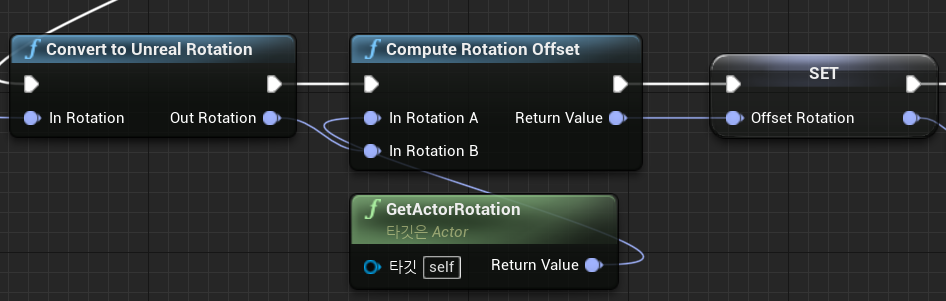
ViveVR Tracker의 Position과 Orientation 값을 얻음.



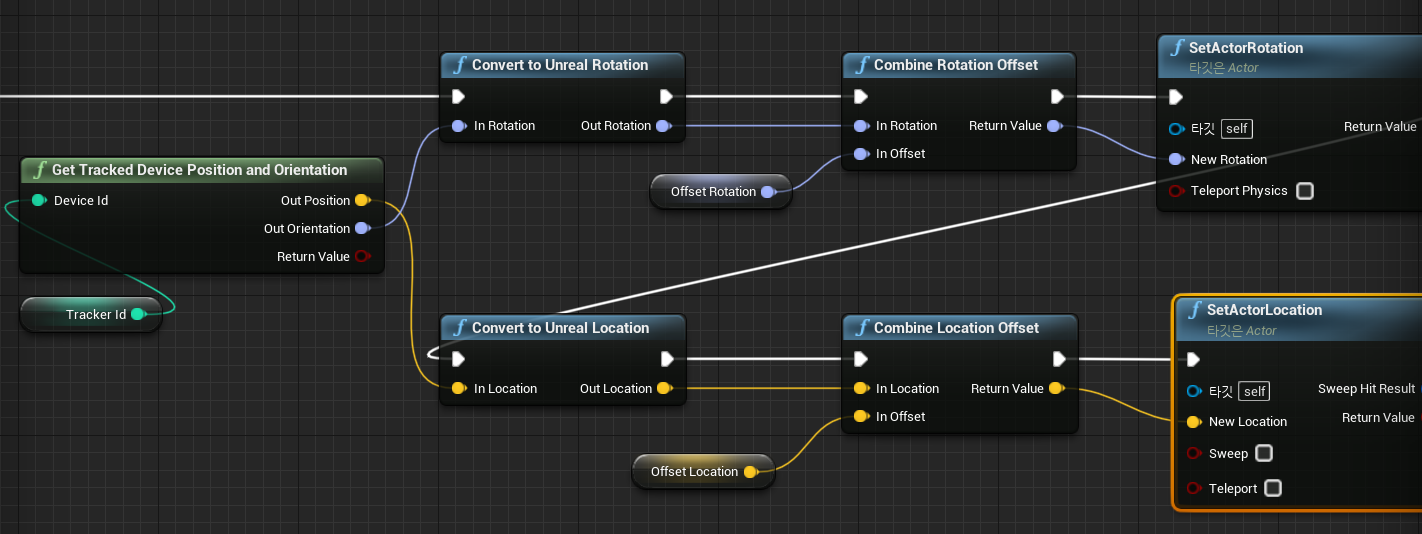
ViveVR Tracker의 Offset 이동 값 얻기



ViveVR Tracker의 Offset 회전 값 얻기



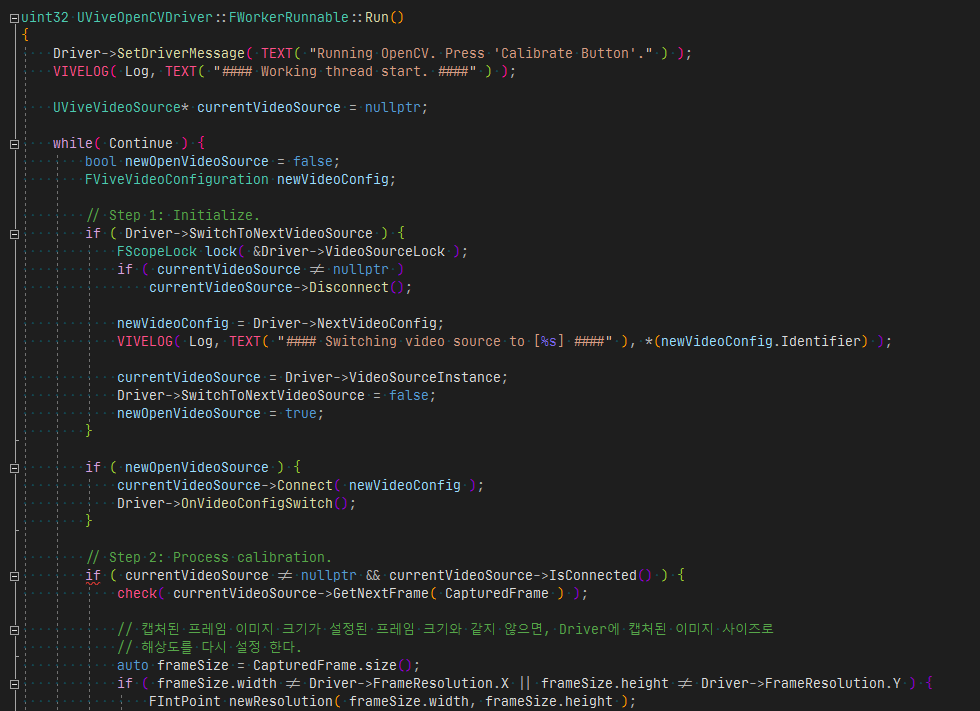
매 프레임 ViveVR Tracker의 이동 값과 회전 값을 Unreal Engine 가상 카메라 엑터에 적용 하기



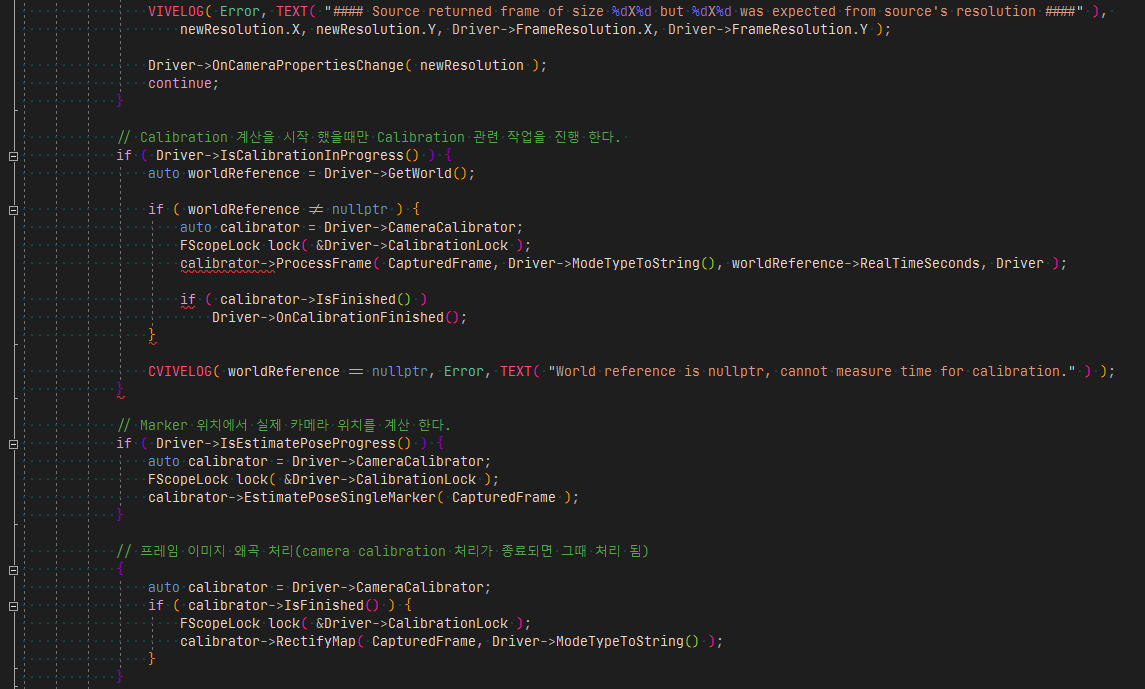
1. **구현 내용**

OpenCV API를 활용하여 Unreal Engine의 Camera Actor의 위치를 설정하는 기능에 대한 구현 내용을 정리한다.

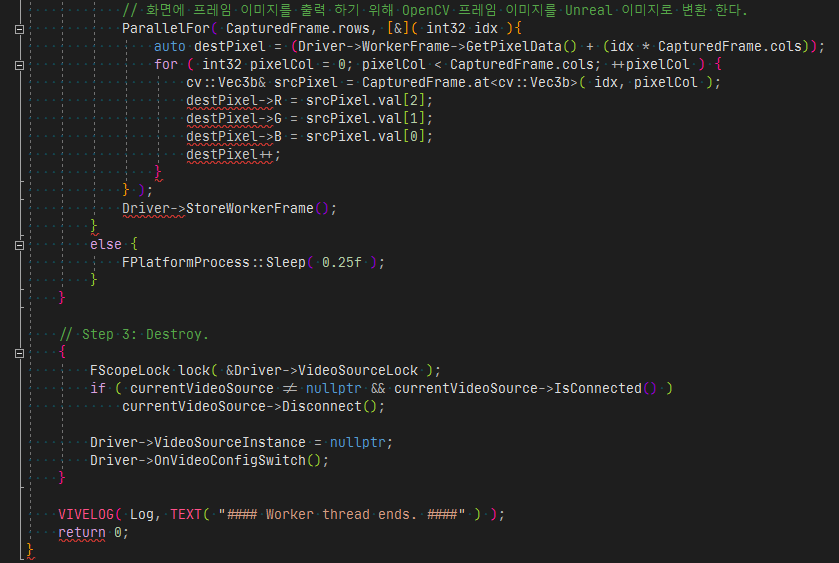
ViveOpenCV 플러그인의 메인이 되는 코드는 UViveOpenCVDriver::FWorkerRunnable::Run() 함수이다. OpenCV API로 카메라가 촬영한 영상 데이터를 처리에는 시간이 다소 소요된다. 대부분 OpecCV API 함수들은 블로킹 함수들로써 함수를 실행하게 되면 함수가 종료되기 전에는 다른 일을 할 수가 없다. 그래서 Unreal Engine의 Thread System을 이용하여 OpenCV API가 영상 데이터를 처리 하는 부분을 병렬 처리 될 수 있도록 구현 하였다.



[그림] UViveOpenCVDriver::FWorkerRunnable::Run() 함수 도입 부분



[그림] UViveOpenCVDriver::FWorkerRunnable::Run() 함수 중간 부분



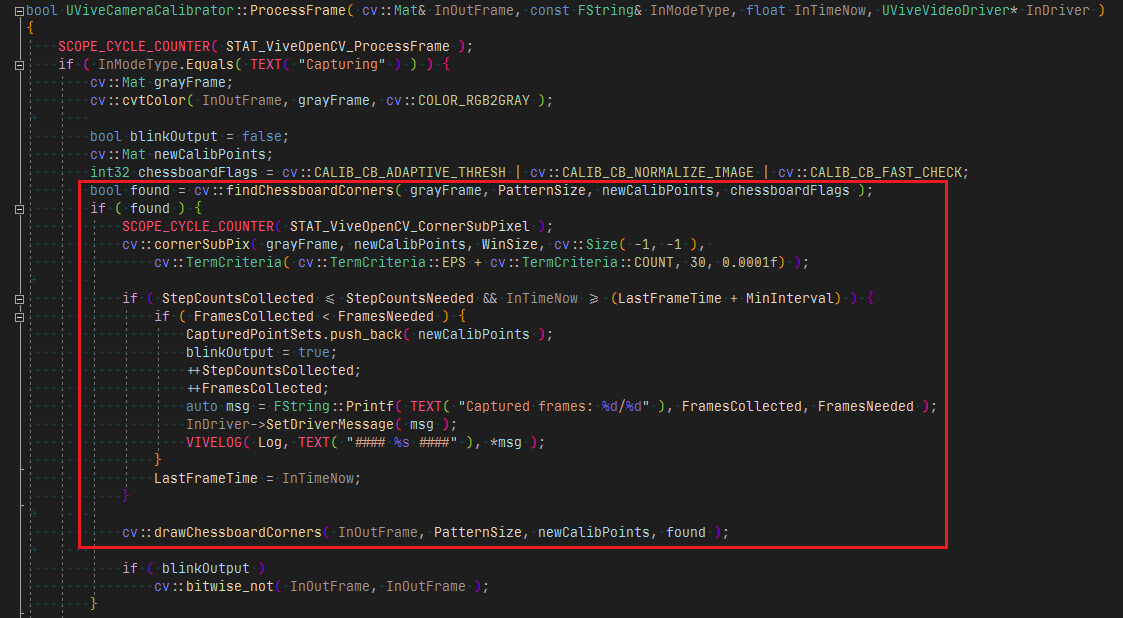
[그림] UViveOpenCVDriver::FWorkerRunnable::Run() 함수 끝 부분

위 코드를 보면 루프문 초기에 OpenCV를 사용하여 카메라 제어를 위한 초기 설정을 하고, 설정이 끝나면 Chessboard 이미지를 활용하여 실제 카메라 Calibration을 진행하게 된다. 카메라 Calibration이 끝나게 되면 Marker 이미지를 활용하여 실제 카메라가 위치한 지점을 계산한다.

이러한 과정에서 촬영되는 영상 이미지를 Unreal Engine으로 전달하는 코드가 3번째 그림에서 ParallelFor 함수를 이용하여 전달하기 위한 이미지 버퍼로 픽셀 데이터를 옮기고 있으며, StoreWorkerFrame() 함수를 통해 Unreal Engine으로 전달된다.

2번째 그림에서 calibarator->ProcessFrame() 함수에서 OpenCV API로 카메라 Calibration을 하기 위해 Chessboard 코너 검출을 하고 여러 위치에서의 Chessboar 코너 검출 데이터를 수집하여 수집된 데이터로 카메라 Calibration을 하게 된다.

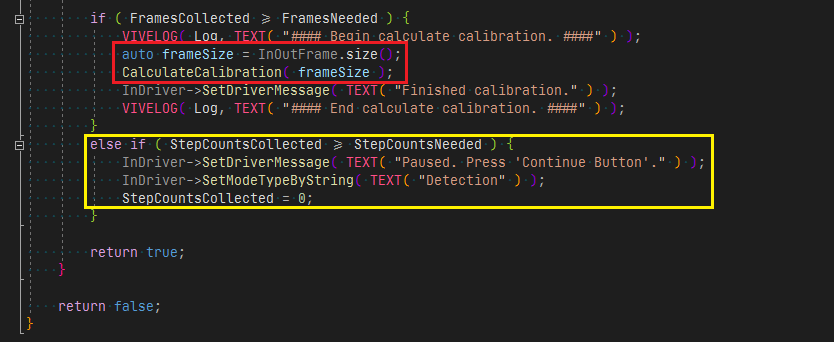
다음 그림에서 빨간 박스 부분이 Chessboard 코너를 검출하여 수집하는 부분이다.



[그림] UViveCameraCalibrator::ProcessFrame() 함수

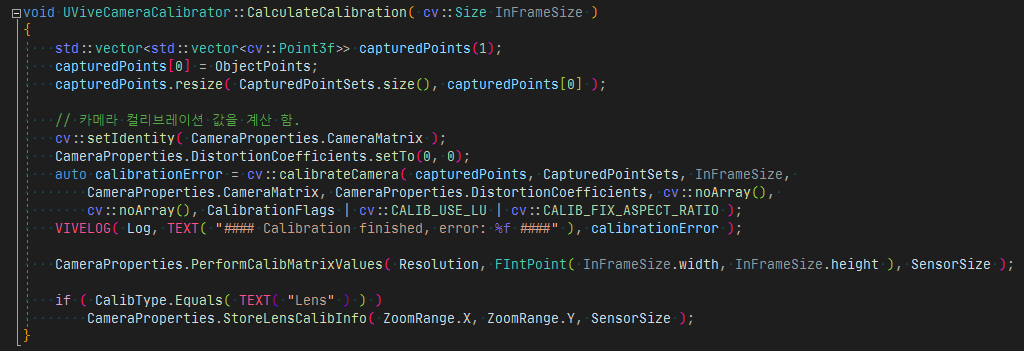
수집된 Chessboard 코너 검출 데이터들을 가지고 실제 카메라 Calibration 처리를 담당하는 함수를 호출하는 부분이 다음 그림에서 빨간색 박스 부분이다.

노란색 박스 부분은 정해진 계수만큼의 Chessboard 코너 검출 데이터가 수집되지 않았었음 화면에 알려주는 코드이다.



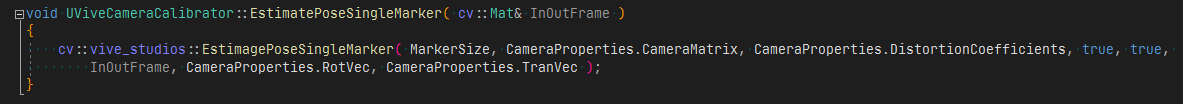
[그림] UViveCameraCalibrator::ProcessFrame() 함수

카메라 Calibration 처리를 하는 부분은 다음 그림의 코드이다.



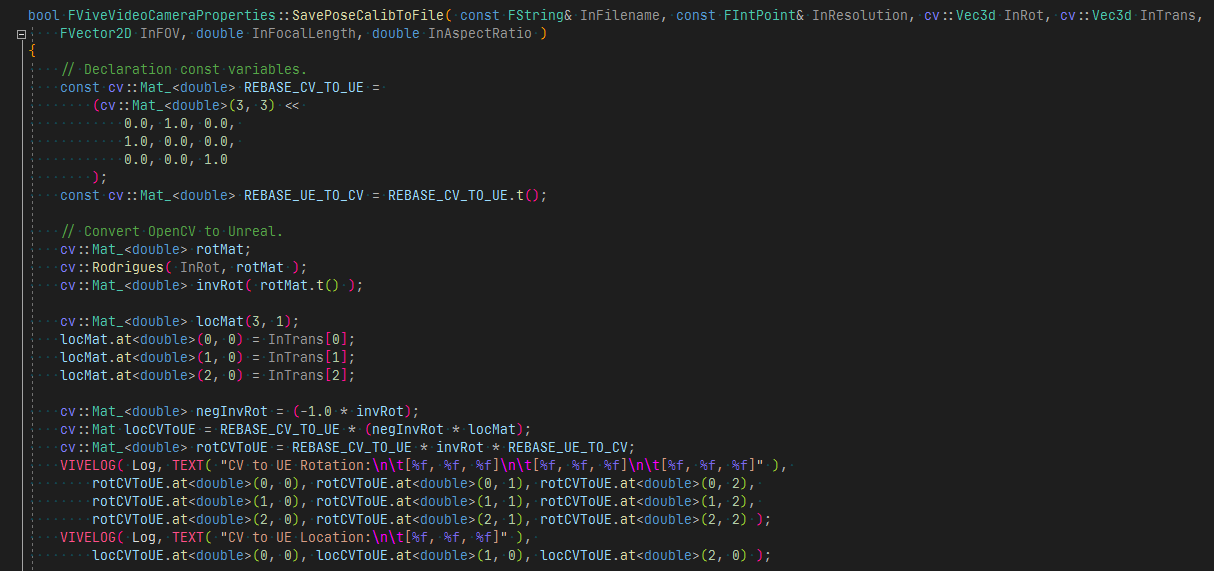
[그림] UViveCameraCalibrator::CalculateCalibration() 함수

카메라 Calibration 정보를 가지고 실제 카메라 위치를 계산하기 위해서는 Marker 이미지를 이용한다. Calibration 정보를 입력 값으로 주고 Marker 이미지 영상을 분석해서 실제 카메라 위치를 계산한다.

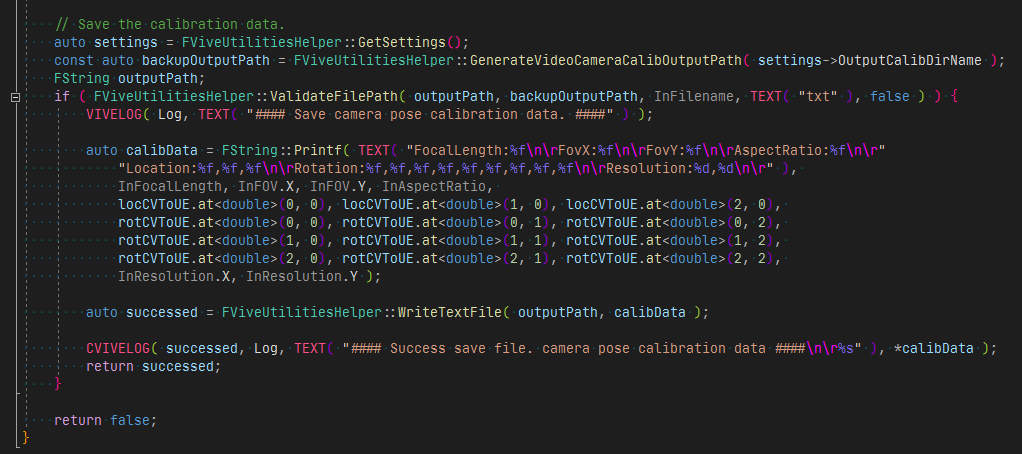


[그림] UViveCameraCalibrator::EstimatePoseSingleMarker() 함수

계산된 실제 카메라 위치는 OpenCV 좌표 시스템 기반이기 때문에 Unreal Engine에서 바로 사용할 수가 없다. 그래서 OpenCV 좌표 시스템에서 Unreal Engine 좌표 시스템으로 변환하여 외부 파일로 저장 한다.



[그림] FViveVideoCameraProperties::SavePoseCalibToFile() 함수 도입 부분



[그림] FViveVideoCameraProperties::SavePoseCalibToFile() 함수 끝 부분

저장 파일은 간단한 텍스트 기반 파일로 속성 ID와 속성 값으로 구성되어 있다.