# HMD Implementation in Unreal Engine with 3-DOF Tracking Using Real-Time Serial Data from Arduino

Andrew Bechdolt
Department of Electrical Engineering
mohuna@stanford.edu

Dina Hashash
Department of Computer Science
hashash@stanford.edu

## Abstract

*Despite advancements in virtual reality (VR) hardware and software, the software development landscape remains dominated by Unity Engine and reliance on commercial head-mounted displays (HMDs). To address this, we aim to expand VR prototyping beyond Unity and existing HMD ecosystems by implementing a pipeline in Unreal Engine to support* ***any display and real-time three degrees of freedom (DoF) serial input in custom Unreal environments.*** *To demonstrate, we present a VR experience developed in Unreal Engine on our E267 Viewmaster HMD. The headset streams real-time quaternion data from an Arduino Teensy using inertial measurement unit (IMU) sensors, and renders a stereo scene with distortion correction for immersive viewing. The experience draws aesthetic inspiration from the Apple TV+ series Severance, incorporating interactive audio-visual events and custom scenebuilding. This work highlights the potential of hardware-agnostic VR development and creates a path forward for future implementations.*

## 1. Introduction

With advances in display, processing, and sensor technology, Virtual Reality is rapidly becoming a more accessible and attractive industry. In 2022, the VR market was sized at 59.96 billion USD with a predicted compound annual growth rate of 27.5% until 2030 [1]. Despite this, most virtual reality content development today is constrained to a few software environments (e.g., Unity) and existing commercial HMDs (e.g., Apple Vision Pro, Meta Quest).

Our goal is to expand accessible VR prototyping by developing a 3-DoF HMD experience for the most elementary of VR headsets in Unreal Engine to harness powerful game and scene building tools. This opens up new creative possibilities for developers and designers working with minimalist, open source, and low-cost VR systems.

Arduino is an open source hardware and software platform for microcontrollers. It maintains a wide user base and support network. Arduino devices range from low-power to ultra-high performance devices, all interoperable and programmably swappable. While many products supported by the Arduino environment support serial communication, such as the Arduino nano and ESP32, for our project we leverage an Arduino supported Teensy, reading from an MPU-9250 IMU. The Arduino Teensy is lightweight, easy to program, and, as demonstrated through multiple course assignments, capable of real-time reading and writing orientation for HMD tracking.

Unreal Engine (Unreal) is a competitor to Unity in development of gaming and visual effects. Unreal's focus on professional level quality has paved the way for high quality lighting, shading, and textures for real-time photoreal and immersive experiences. Unreal blueprint and C++ integration also make it easy to learn for introductory projects. Given support for implementing VR, Unreal becomes a promising alternative to Unity. To demonstrate the power of Unreal, we integrate our HMD into an immersive Unreal Engine Scene. We leveraged Unreal's lighting, interactive environment, and spatial sounds to create an unsettling experience.

Inspired by the eerie and stylized atmosphere of *Severance [2]*, our interactive demo blends story, spatial design, and real-time hardware integration to immerse the player.

## 2. Related Work

### 2.1. Tracking and Orientation

Most available consumer VR applications rely on proprietary tracking systems and environments from the headset designer. These headsets are usually capable of 6-DoF orientation and position tracking using either outside in tracking, as with the Oculus Rift, HTC Vive, and Playstation VR, or with inside out tracking, such as with the Meta Quest and Apple Vision Pro. These devices rely on a combination of

IMU data including gyroscopes, accelerometers, and magnetometers as well as visual data including infrared and image capture to accurately track the headset, controllers, and user. This method is more accurate, but more complex and expensive, than our 3-DoF method using just an accelerometer and gyroscope for orientation tracking.

## 2.2. Development Environments

Academic projects using IMU tracking often utilize Unity due to more seamless serial communication support and plugin availability. Course materials from EE267 provide a serial streaming solution for Unity-based tracking, but Unreal remains less explored. Unreal is more popular for hobby and professional development of non-VR and commercial VR plugins given its OpenXR implementation and additional libraries for SteamVR and HTC.

## 2.3. HMD Implementation

From past efforts, both Unity and Unreal Engine have HMD support for different headsets and environments. In addition to proprietary standards, the common standard for developers on both platforms is OpenXR [3]. OpenXR is an open source and royalty-free standard used for adapting experiences to AR and VR. It allows for the design of new VR platforms and HMDs and is interoperable with most VR HMDs and software (Fig 1). This does not work with our current display, which connects directly over HDMI with no on-board communication available for OpenXR.

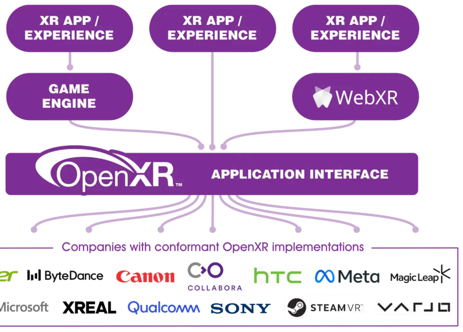

Figure 1: A flowchart of OpenXR Application Interface between experiences, engines, and VR devices, sourced from [3].

Unreal Engine also provides support for a plugin called SimpleHMD. Simple HMD is capable of converting a stereo-camera output to the left and right eye in the VR environment. However, when we attempted to use it in our third-person or first-person environment we were unable to output proper images, with the right eye displaying garbage data (Fig. 2). The SimpleHMD plugin also did not allow for distortion correction, which would be essential for maintaining straight lines along the display edges.

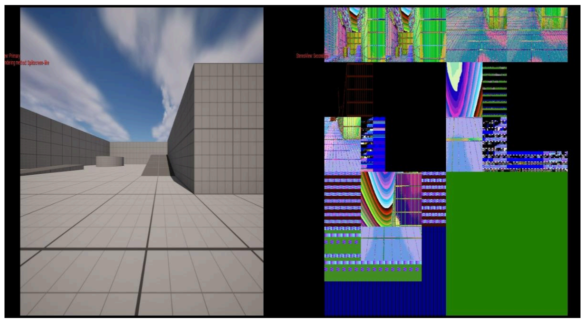

Figure 2: VR Display in 3rd Person Environment using SimpleHMD Plugin.

## 3. Method

### 3.1. Hardware and Serial Streaming

We used the class-built HMD, powered by an Arduino Teensy and InvenSense MPU-9250 IMU. Using Velcro, the board containing the Arduino and IMU are mounted to the front of the ViewMaster Headset as shown in Fig. 3. Using the code from EE267 homework 5, the tracking software fused accelerometer and gyroscope data read from IMU to form quaternions that represent the orientation of the headset. The quaternions are sent over serial at a baud rate of 115200 in the form:

```
QC W X Y Z
```

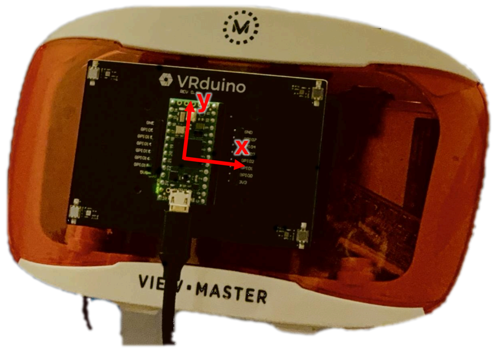

Figure 3: VRduino, including Arduino Teensy and MPU-9250 IMU mounted to the View Master HMD with X and Y axis reference for relative IMU orientation.

We then implemented the SerialCOM plugin [4], in Unreal Engine, connecting the Arduino Teensy and allowing for serial data to be used in the environment.

## 3.2. Unreal Engine Serial Integration

The Unreal scene is built from the third-person game template, which we modified to support stereo rendering and headset control. Once per game tick, a blueprint system reads serial data and orients the in-game camera (rigged to the head bone of the player pawn as described further in 3.3) based on calculated quaternions. To do so, we map the orientation of the VRduino, (right hand orientation with Y up) to Unreal (right hand orientation with Z down[1]), as seen in Table 1.

| Camera Space | Sensor Space |
|---|---|
| W | W |
| X | -Z |
| Y | X |
| Z | -Y |

Table 1: Sensor space axes from the serial input of the quaternion to the Unreal Engine rotation quaternion, before rotating 180 degrees to adjust to world space.

1. Unreal's axes of orientation are right handed with z-up, however, for reference orientation before the 180 degree rotation, the z-axis remains down. The final orientation after rotating about the y-axis is with z-up.

The quaternion is then rotated 180 degrees about Y. Combined, these transformations result in the correct orientation mapping about all 3 axes (Fig. 4).

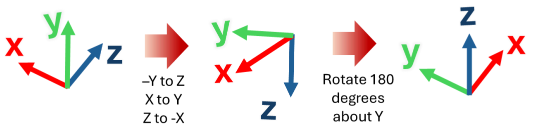

Figure 4: Axis mapping from serial input quaternion to camera space.

This transformation is then applied to the camera in the Third Person Blueprint, as depicted in Fig. 5. It remains a computationally cheap method of remapping axes versus rotating the relative orientation in sensor space to match our Unreal environment .

The camera orientation is then mapped to the player location and controller orientation. This allows movement to be relative to the viewing direction. (Fig. 6).

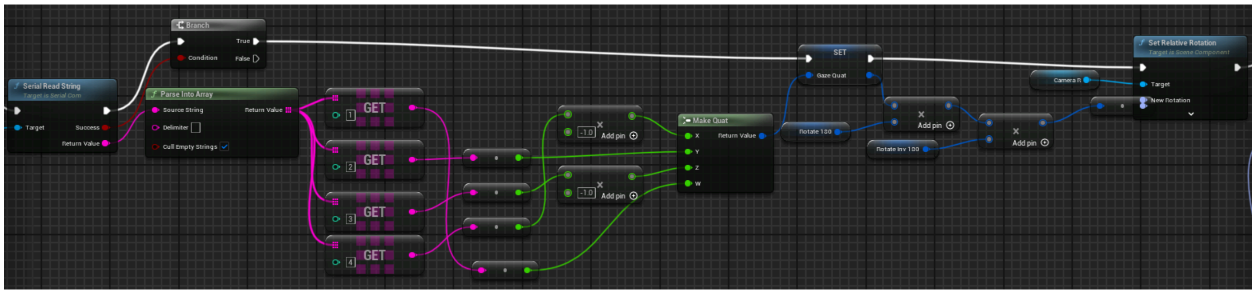

Figure 5: Camera orientation from serial input reading in Unreal Engine third-person blueprint.

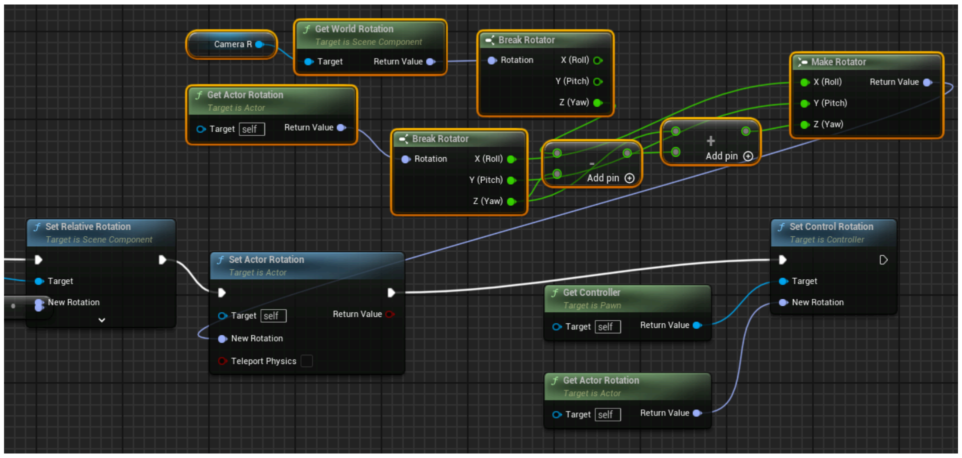

Figure 6: Character and motion orientation update from camera direction.

### 3.3. Unreal Engine HMD Output

To properly recreate a stereo view, we implemented a modified stereo-camera scene capture. The scene is captured from the perspective of the third person actor. In game development, this is the pawn controlled by the player. Here it serves as reference for the camera height and movement. Two SceneCapture2D objects are attached to the pawn's head spaced 63 cm apart to match the IPD of the View Master headset (Fig 7 ).

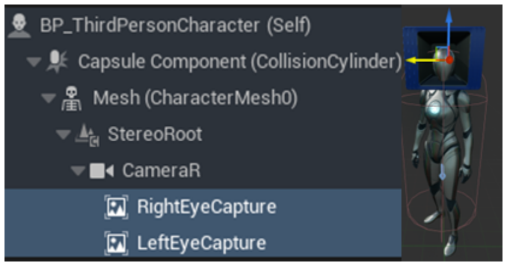
BP_ThirdPersonCharacter (Self)
Capsule Component (CollisionCylinder)
Mesh (CharacterMesh0)
StereoRoot
CameraR
RightEyeCapture
LeftEyeCapture

Figure 7: Implementation of Stereo Scene Capture attaching scene capture as children of the third-person character (Imaged right).

The scene captures are then applied to a render target, stored as a texture. The render target is then applied to custom user interface (UI) materials, as shown in Fig. 8.

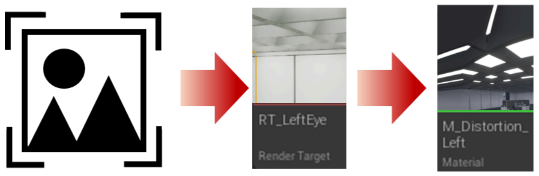

Figure 8: Mapping of the SceneCapture object (left, to the render target texture, middle, which is used in the material for display, right).

In the material, the UV map is adjusted to apply barrel distortion, increasing the distortion based on distance from the center as specified in [5],

$$x_d \approx x_u(1 + K_1 r^2 + K_2 r^4)$$
$$y_d \approx y_u(1 + K_1 r^2 + K_2 r^4)$$

Where xu and yu are the undistorted coordinates of the UV texture map, r is the distance from the eye's center, and xd, yd are the distorted coordinates with $K_1 = K_2 = 2$.

Each image is then applied to a user interface widget, with the left half of the display, bisecting the complete canvas between the two viewpoints. We then overlay the UI on top of the viewport, and ensure it runs for the duration of the experience, as visualized in Fig 9.

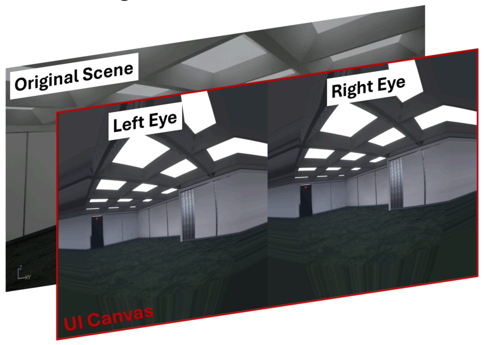

Figure 9: User interface overlay of the stereo view using a User-Interface (UI) canvas and captured scene saved and distorted to the UI materials.

### 3.4. Scene Design and Interaction

We designed a fully immersive office scene inspired by *Severance,* shown in Fig. 10, using a mix of custom-modeled and sourced assets. The desks were imported from an existing model, but the entirety of the rest of the scene and props [6] were built and textured manually using a hybrid of Blender and Unreal. Worldbuilding included:

- Footstep SFX tied to the pawn's animation, with randomized pitch per step.
- A proximity trigger that plays the *Severance* elevator SFX and theme song *[7]*.
- Environmental light flickering tied to the music cue.
- A moody, minimalist lighting setup of emissive material optimized for atmosphere.

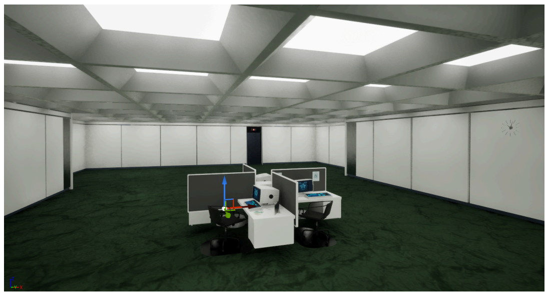

Figure 10: Minimalist office environment modeled on the office from *Severance* with emissive lighting and shadows.

## 4. Analysis & Evaluation

We achieved a complete 3-DoF interactive VR experience in Unreal depicted in Fig. 11. We confirmed stable orientation tracking with perceptually low drift over 1 hour of operation.

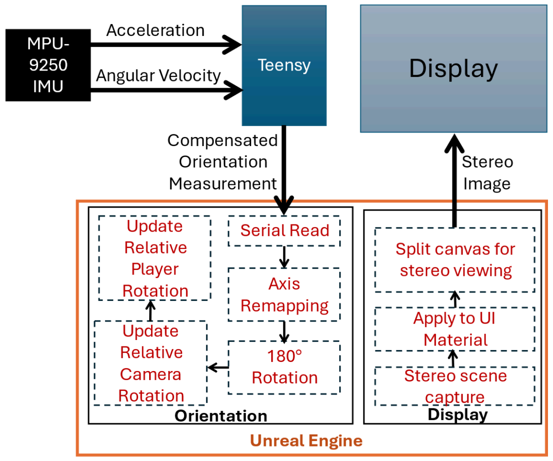

Figure 11: Unreal Engine implementation for barebones VR HMD using the Arduino Teensy and MPU-9250 IMU for orientation and an HDMI display for viewing.

Latency between head motion and scene response is ~15ms tied to the in-game tick speed. Large discrepancies are likely tied to computer hardware and the demand of Unreal rather than the connection between the HMD and our designed blueprint. Lighting and physics are generated in real-time, placing a heavy load on the computer. Pre-baking some lighting, and further optimizing geometry would improve the latency.

Implementation through blueprints allowed clean modularity, enabling easy edits for future versions of the project.

## 5. Results

### 5.1 Qualitative

Users reported a sense of immersion and eerie feeling in the triggered elevator sequence. Head movement mapped reliably to in-scene camera control, allowing users to look around and navigate naturally with WASD keys. Some motion sickness at rapid turns were reported.

Some visual issues still occur with the stereo view. Unreal Engine overrides our current custom StereoSceneCapture class with their general SceneCapture2D object which leads to an imperfect implementation of the asymmetric frustum, which would be more noticeable in headsets with larger fields of view.

### 5.2 Quantitative

*Serial Communication*

At the time of submission, with an RTX 3080 and AMD Ryzen 9 5900, the project runs at an average of 72 FPS using 3 GB of system memory. The serial input updates at a baud rate of 115200 and a read frequency of ~60Hz based on a 0.017s game tick. This frequency can be increased by increasing the number of updates per game tick. However, varying the number of updates per in-game tick from 1 to 20 had no impact on perceived smoothness.

In an 8 minute stress test, after extensive usage of the VRduino, 12 out of 28800 packets were dropped in serial communication. Most packet drops occurred when interacting with multiple physics objects at once while moving the VRduino erratically.

| Metric | Value |
|---|---|
| Average FPS | 72 |
| Update Rate | ~ 60 Hz |
| Packet Drop Rate | 0.04% |

Table 2: Summary of Unreal Engine Quantitative performance during HMD demonstration.

*Tracking and Orientation*

To test the stability of the compensated IMU measurements from the Teensy, we ran a 10 minute drift measurement, calculating orientation every 5 seconds. Across two 10 minute measurements, the maximum drift across all axes was 0.0001 units per minute with a maximum variation of 0.001 units per minute, as plotted in Fig. 12.

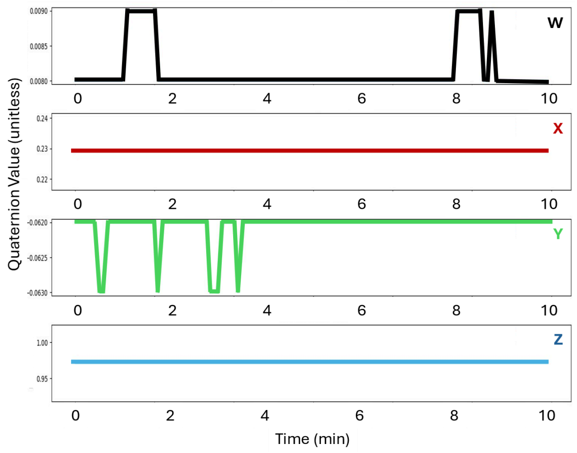

Figure 12: Map of quaternion drift and variation across 10 minutes.

## 6. Discussion

For the time and resources given, we would consider our demonstration successful. In two weeks with limited prior experience we are satisfied with the progress we have achieved in adapting Unreal Engine for custom VR HMDs. While we experienced challenges in HMD implementation, with slight inaccuracies inducing motion sickness, and Unreal Engine overwriting the asymmetric capture with it's own symmetric scene capture, we have achieved reliable 3-axis orientation tracking and in-game alignment for both the camera and player character using an Arduino and digital IMU, and updating a stereoscopic display with no VR drivers. Additionally, we have implemented an immersive experience which participants described as eerie during informal testing, despite challenges with texture compatibility between Blender and Unreal Engine. However, several limitations remain, as discussed in 6.1, and we have several goals for future work to innovate on this idea, as discussed in 6.2.

### 6.1 Limitations

- Over time, the Arduino must be reset due to drift and struggles in long-term accuracy.
- Unreal engine requires fully packaged textures, which cannot be imported directly from blender. Objects still need to be textured within Unreal Engine.
- Due to the stereo rendering implemented as image renders as well as a low framerate, some participants reported motion sickness.
- While our final code can maintain proper orientation tracking for over an hour, we cannot reset orientation to a new reference without adjusting the rotation axes.

### 6.2 Future Work

Given our time and hardware limitations, our proof of concept remains limited to hardware that matches the behavior of our current implementation. With the current code, Unreal Engine requires a Windows computer and knowledge of which serial port the Arduino Teensy is connected to. HMD orientation is also hardcoded to match our headset implementation. In future work, we aim to make the Unreal Engine implementation more generally applicable, more efficient, and more easily implementable. This includes:

- Reworking the serial communication to find and communicate with arbitrary MCU in Windows, Mac and Linux.
- Allowing remapping of orientation to match alternate IMU configurations.
- Expanding to 6-DoF with positional tracking and allowing full developer control over functionality.
- Using a higher-precision IMU or sensor fusion algorithm.
- Adding direct player interaction with the scene via handheld controllers.
- Releasing this HMD integration as an Unreal plugin for open-source use.

## 7. Conclusion

In this work, we implemented a functional VR headset pipeline in Unreal Engine using low-cost, open source hardware for a head-mounted display and 3-DoF orientation tracking. Our implementation leverages quaternion data from an Arduino-based IMU, serial streaming, and stereo rendering with distortion correction to port immersive environments to VR.. We demonstrated this through an interactive experience in a custom made office environment inspired by *Severance*. By overcoming Unreal Engine's limited support for custom hardware, we create a potential path forward for open-source, hardware-agnostic HMD development. Future extensions include generalizing for 6-DoF tracking and controller integration.

## References

[1] "Virtual Reality Market Share & Trends Report, 2021-2028," Grand View Research, https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-market#

[2] *Severance*, Apple TV+, 2022. https://tv.apple.com/us/show/severance/umc.cmc.1srk2goyh2q2zdxcx605w8vtx

[3] "OpenXR" *The Khronos Group*, Dec. 06, 2016. https://www.khronos.org/openxr/

[4] R. M. D. Oca, "Serial Communication Plugin for Unreal Engine," *GitHub*, Nov. 29, 2022. https://github.com/videofeedback/Unreal_Engine_SerialCOM_Plugin

[5] G. Wetzstein, Virtual Reality, *Lecture 7: Head Mounted Display Optics I.* EE267 , Stanford University, Stanford CA, April 22, 2025. https://stanford.edu/class/ee267/lectures/lecture7.pdf

[6] R. Workshop, *Lumon Office - Severance TV Show*. Unreal Engine Model, Fab, October 4, 2024. https:/www.fab.com/listings/4f83bdd4-d2b9-4b71-8b17-648c27204cc8

[7] T. Shapiro, "Main Titles," *Severance: Season 1 (Apple TV+ Original Series Soundtrack)*, Endeavor Content, Feb. 18, 2022. [Online]. Available: https://theodoreshapiro.bandcamp.com/album/severance-season-1-apple-tv-original-series-soundtrack

## Acknowledgements

Thank you to Manu Gopakumar and Prof. Gordon Wetzstein for support on this project and in this class. This project includes code from EE267 Homework 5 VRduino solution and sound effects from *Severance.*