OpenVR

Project Proposal

Profs: Duane Marcy and Jennifer Graham

CSE/ ELE 492

February 1, 2019

Khaled Aloofey

Anqi Chen

Li Song

Table of Contents

Page #

Abstract

In this proposal, we present a technical plan for building a complete, high end virtual reality system with a tethered headset and a user-facing depth sensor as its controller. The system uses the Unity game engine for the virtual environment simulation, the Xbox One Kinect as its depth sensor, and the Microsoft SDK Wrapper to program the captured data from the user's body movement into interactions inside the virtual environment. Further, we will provide assembly instructions, components information and specifications, schematics, design files, in addition to software documentation. Hence, we call this system OpenVR.

1.0 Introduction

1.1 Problem & Solution

Virtual reality (VR) is defined as the computer-generated simulation of a three-dimensional image or environment that can be interacted with in a seemingly real or physical way by a person using special electronic equipment, such as a helmet with a screen inside or gloves fitted with sensors [1].

VR systems on the market are classified as either having mobile or tethered headset- with the tethered being the superior in terms of driving high quality graphics and the overall immersive experience they provide via their controllers. However, tethered headsets are generally pricy, unupgradable, and their controls are un-customizable.

Consequently, we propose to make a new VR system that uses a tethered headset that has a head mounted display (HMD) connected to a PC for superior graphics, and a modular design scheme for easy upgrading. Further, the system will have a user-facing sensor to capture the user's motion and implements controls accordingly. We call it OpenVR.

1.2 Objectives

1.2.1 Project Main Objective

Our main intent for making OpenVR is to provide an opensource, free recipe for making a complete, high-end VR system that uses motion-capture as its controller and a tethered headset. We will be providing assembly instructions, components information and specifications, schematics, design files, in addition to software documentation. OpenVR is not a commercially available product. But, we will build a prototype for demonstration and documentation purposes in this project.

2.0 Design

2.1 Design Description

OpenVR has a hardware bundle and a software bundle (see Appendix A). First, the hardware bundle consists of an HMD housing that includes the lenses and embeds the HMD itself, and an IMU linked to a microcontroller- both mounted on the front of the HMD housing. Collectively, they make the helmet or the headset. Moreover, the Xbox One Kinect will be used as a user-facing motion sensor. The software bundle includes Microsoft (MS) Kinect SDK, Unity, Arduino IDE, MS-Kinect SDK Wrapper, and GoogleVR SDK. Lastly, the HMD will be connected to a PC via HDMI.

2.2 Design Requirements

- The helmet shall be able to track the user's head orientation
- The helmet shall be stylish.
- The helmet shall be durable and sturdy.
- The helmet shall be lightweight and comfortable to wear.
- The HMD inside the helmet shall be at least Full HD in resolution (1920 x 1080 pixels).
- The Kinect sensor shall have eight main programmable gestures, which are: Swipe left, swipe right, swipe up, swipe down, jump, squat, cursor movement, and clicking.
- The Kinect sensor shall have the ability to track the user's position in real-time in the three-dimensional spatial directions with respect to the main camera inside the virtual environment.
- The Kinect sensor shall have the ability to recognize at least two players simultaneously and recognize their gestures independently.

2.3 Design Features

- It has a modular design that allows it to be highly customizable and fully upgradable down the road.
- It implements controls from motion-captured data with no need for physical, handheld controllers.
- It implements player positional tracking with six degrees of freedom (rotational + translational) for a fully immersive experience.
- It achieves all of its sensing capabilities without the use of bulky stationary sensors.

2.4 Block Diagram

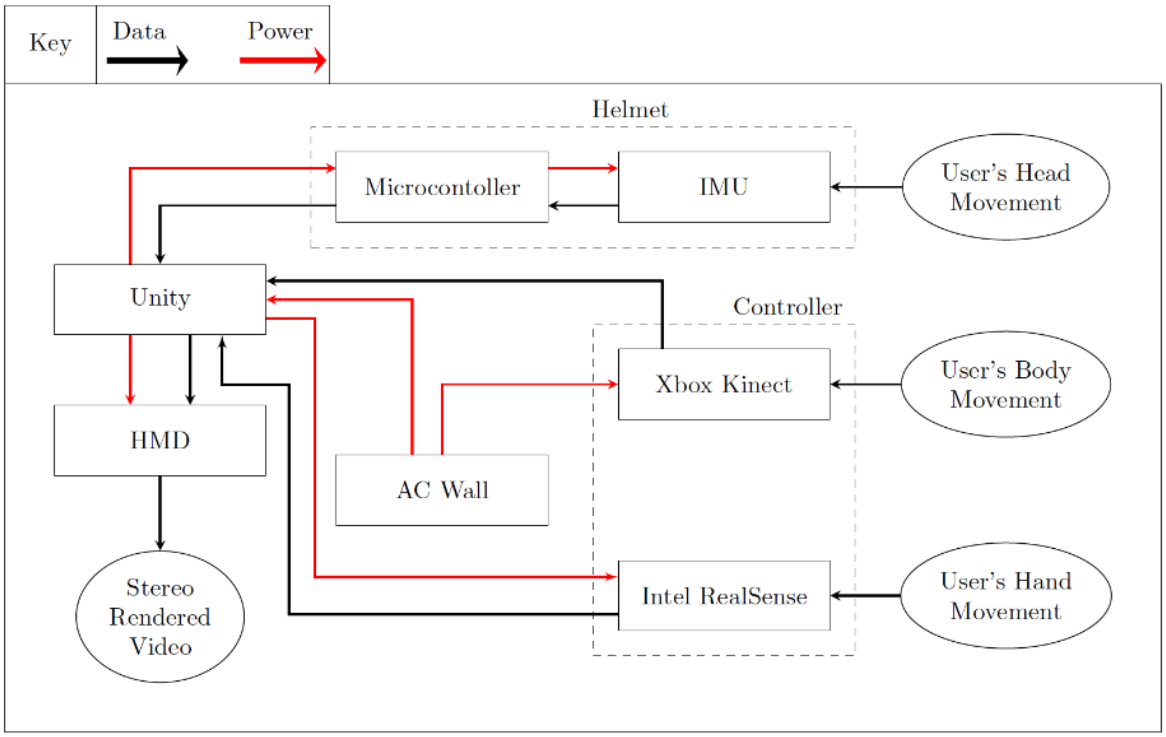


Figure 2.1: OpenVR Hardware Block Diagram

2.5 Blocks Descriptions

2.5.1 Head Mounted Display (HMD)

The HMD receives the image data from Unity via HDMI and outputs the stereo rendered video for the user to view through the lenses. We plan on using the Topfoison 6" 2K LCD display for its high resolution, thin size, and its low power consumption. Also, this Topfoison display has the same screen size as the Google Nexus 6, which is necessary for the stereo rendering and lens distortion to work correctly.

Figure 2.2: Topfoison 6" Full HD HMD [2]

Furthermore, the HMD will be placed inside a designated housing that contains the lenses. For our OpenVR prototype, we will go with the ViewMaster Deluxe VR Viewer because its stylish look, price, and housing capacity fit our design requirements.

Figure 2.3: ViewMaster Deluxe VR Viewer [3]

In addition, the ViewMaster has a display holder which uses a clamping mechanism to tightly hold the HMD in place (Figure 2.4). Finally, there will be an HDMI and a micro-USB cables coming out of the headset to the PC running Unity- one for data transmission and the other for power.

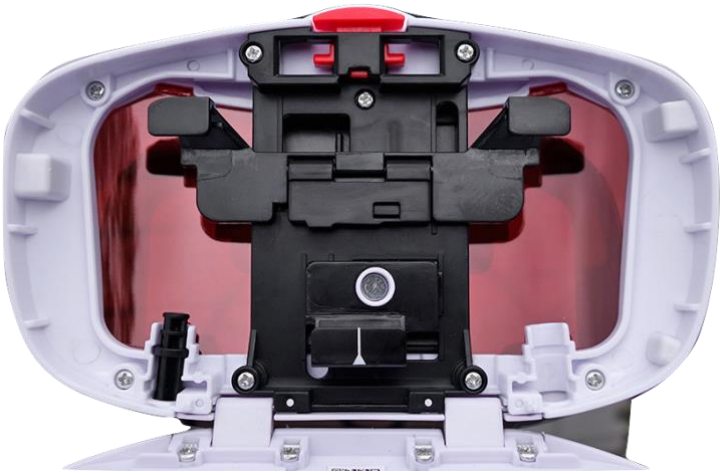


Figure 2.4: The Interior of the ViewMaster [4]

2.5.2 Unity

Unity receives data from the microcontroller and the Kinect serially via USB. Then it processes these data and outputs the game video after the lens distortion and the stereo rendering have been performed. Additionally, Unity needs to run on a PC running Windows 8, so that the needed drivers for the Kinect sensor to interface with the computer are installed using MS Kinect SDK.

OpenVR uses Unity as its game engine for its ease of use and VR compatibility. Unity simplifies (to an unprecedented level) the process of implementing game logic, game physics, collision detection, animation, 3D graphics rendering, I/O interfacing, networking, and much more. Further, the main method of implementing stereo rendering and lens distortion in Unity is through the GoogleVR SDK. This SDK makes implementing such task extremely simple.

2.5.3 IMU

The IMU, which is linked to the microcontroller via the I2C bus in a master-slave configuration and mounted on the front of the HMD housing reads and digitizes the data coming from the user's head movement and then stores it in a set of sensor registers for them to be accessed by the microcontroller. This means that the IMU is configured as slave. Furthermore, the IMU takes 3.3V from the microcontroller for power. Last, we will be using the InvenSense MPU-9250 as our IMU due to its popularity, price, and positive reviews.

Figure 2.5: The InvenSense MPU-9250 IMU [5]

2.5.4 Microcontroller

The microcontroller in OpenVR addresses the 3-axis accelerometer and gyroscope sensor registers on the IMU through the I2C serial bus and computes a quaternion that tracks the orientation of the player's head. This quaternion is then sent to the PC running Unity as a parsed string of data to via USB, which is also the source of power for the microcontroller. Finally, we will select the Teensy v3.2 as the microcontroller for our OpenVR prototype since it is compatible with the Arduino libraries and has superior processing speed to expedite the Euler's angles calculations of the user's head orientation.

Figure 2.7: The Teensy (V3.2) Microcontroller [6]

2.5.5 Xbox Kinect

The Xbox Kinect captures the user's motion using a depth sensor and sends that data to Unity through USB. It serves as the main controller for OpenVR, the user's motion is translated into executable actions inside the simulated virtual environment (see Software section).

Figure 2.8: The Xbox One Kinect (V2) [7]

2.6 Software

2.6.1 MS Kinect SDK

This SDK installs a few drivers in Windows, so that the PC recognizes the Kinect as a device connected to it via USB. We will be using version 2.0 of the SDK since our prototype uses Kinect v2. However, OpenVR is compatible with the Kinect v1 as well. Also, this SDK provides the developer access to raw sensor streams and skeletal tracking data.

2.6.2 MS Kinect SDK Unity Wrapper

This software tool provides a family of methods which contain calls for the Kinect SDK libraries inside Unity and can be obtained freely from the Unity Asset Store. It allows developers to program their games in Unity to read Kinect-detected player gestures as true or false (bool data type) with relative ease. Further, this wrapper gives developers access to skeletal tracking data in Unity for character "avateering." Lastly, Kinect Unity Wrapper will enable us to track the position of the player with respect to the camera inside the virtual environment.

2.7 Circuits Schematics

2.7.1 Headset Schematic

The IMU is modeled as an ADC in this schematic because all of its sensor registers are tied to the I2C bus, which has two lanes: The serial clock (SCL) (D1) and the serial data (SDA) (D0). For the Teensy and the IMU to communicate properly, their corresponding SCL and SDA pins need to be wired together. The SCL and SDA pins on the Teensy are pins A4 and A5 respectively (see Appendix B).

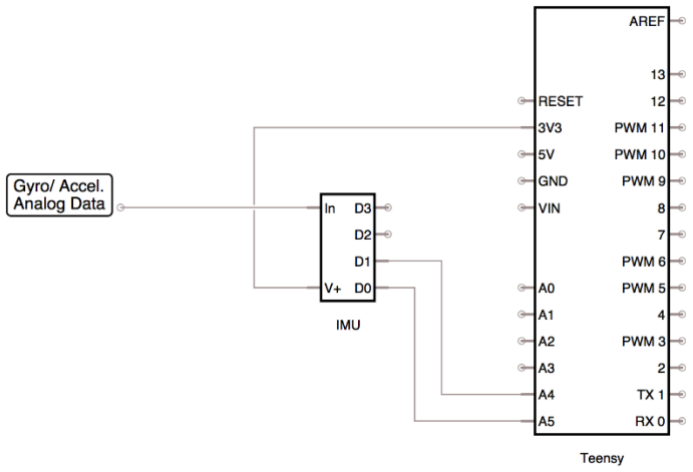


Figure 2.9: Schematic for the OpenVR Prototype Headset

3.0 Calculations

3.1 Representing 3D Rotation as Quaternions

Unity understands rotation in 3D space as quaternions to avoid the gimbal-lock problem that Euler's angles pose. Therefore, we need to provide some basic mathematical definitions of quaternions in order to properly convert the IMU measurements into a quaternion that estimates the player's head orientation and send it to Unity via the Teensy.

Quaternions are an extension of complex numbers (which can be thought of as vectors) with the following definitions:

$$q(w,x,y,z) \stackrel{\text{def}}{=} q_w + iq_x + jq_y + kq_z$$

$$i \neq j \neq k$$

$$i^2 = j^2 = k^2 = ijk = -1$$

$$||q|| = \sqrt{(q_w)^2 + (q_x)^2 + (q_y)^2 + (q_z)^2} = 1$$

Our plan now is to represent each axis of rotation (roll, pitch, and yaw) with its own quaternion and then combine these three quaternions into a single quaternion that describes the orientation of the player's head. Our calculations rely on the following result, which describes conversion from angle axis to quaternion (derivation omitted):

$$q(\theta, v) = \cos\left(\frac{\theta}{2}\right) + iv_x \sin\left(\frac{\theta}{2}\right) + jv_y \sin\left(\frac{\theta}{2}\right) + kv_z \sin\left(\frac{\theta}{2}\right)$$

Where:

$$\cos\left(\frac{\theta}{2}\right) = q_w;\ iv_x \sin\left(\frac{\theta}{2}\right) = q_x;\ jv_y \sin\left(\frac{\theta}{2}\right) = q_y;\ kv_z \sin\left(\frac{\theta}{2}\right) = q_z$$

and

v_n is the respective normalized axis.

The pitch quaternion can be calculated by:

$$q_p = \cos\left(\frac{\theta_p}{2}\right)[1\ 0\ 0\ 0] + \sin\left(\frac{\theta_p}{2}\right)[0\ 1\ 0\ 0]$$

Similarly, the roll quaternion can be calculated as follows:

$$q_r = \cos\left(\frac{\theta_r}{2}\right)[1\ 0\ 0\ 0] + \sin\left(\frac{\theta_r}{2}\right)[0\ 0\ 1\ 0]$$

Likewise, with the yaw quaternion:

$$q_y = \cos\left(\frac{\theta_y}{2}\right)[1\ 0\ 0\ 0] + \sin\left(\frac{\theta_y}{2}\right)[0\ 0\ 0\ 1]$$

Hence, the total 3D rotation can be obtained from:

$$q_{3D} = q_p q_r q_y\ (simple\ quaternion\ algebra)$$

Although the quaternion q_{3D} fully describes the orientation of the player's head, it does not have a sense of direction with respect to our world. Since gravitational pull is what defines "down" in our world, we will make a quaternion q_m that matches the direction in our world to that of q_{3D}.

Let $\dot{q}_m = [0\ m_x\ m_y\ m_y]$, where m_x, m_y, m_z are the measurements from the IMU's magnetometer, then $q_m = q_p q_r \dot{q}_m {q_r}^{-1} {q_p}^{-1}$.

The final task is to combine q_m and q_{3D} into a single quaternion, and now we have a full description of the player's head orientation inside the virtual environment in Unity. We will use Mr. Joseph Malloch's from Dalhousie University Arduino program to pass the newly calculated quaternion onto Unity as a parsed string [8].

3.2 Converting from Quaternions to Euler's Angles.

Quaternions have several advantages over Euler's angles other than their immunity to be gimbal-locked. For example, both linear and spherical interpolation between angles can be performed much easier with quaternions. Angles interpolation allow for exceptionally smooth rotation, and hence they are considered crucial to an excellent VR experience. Conversely, turning off an axis of rotation is far easier with Euler's angles than with quaternions.

Since the roll axis needs to be turned off inside the virtual environment, we will need to convert the quaternion that describes the player's head orientation into an Euler's angles form. This means that we will have to sacrifice a little bit of smoothness because we don't have the mathematical sophistication to disable an axis of rotation of a quaternion. Luckily, Unity has a member function of the class Quaternion called "eulerangles", which gives you Euler angles as a float point from a quaternion.

4.0 Requirements and Verification

4.1 Requirements and Verification Table

Requirement	Verification	Area	Points
HMD 1. Helmet weighs < 0.25Kg. **2.** Helmet is comfortable.	1. Verification Process for Item 1: a. Weigh the helmet + HMD + IMU + microcontroller. 2. Verification Process for Item 2: a. Attach a headband to the helmet. b. Ask two people to wear them for 3 minutes while interacting with the simulated environment.	Hardware	10
Unity 1. Only expected collisions are detected. 2. Measure Euler's angles within 10º inside the virtual environment	1. Verification Process for Item 1: a. Write a code segment in Unity that keeps track of the number of detected collisions. b. Check that the 3D models inside Unity are rigged properly with all of their textures and materials. 2. Verification Process for Item 2: a. Place a protractor on a paper grid and decide on a 0º reference. b. Write a code segment in Unity that streams Euler's angles to the screen. c. Rotate the helmet CCW and compare the actual angle with the readings on the screen. d. Subtract the two to get the difference.	Software	25
IMU 1. The scaling error of the magnetometer measurements is within 4%.	1. Verification Process for Item 1: a. Collect data on the strength of the magnetic field in Syracuse, NY. b. Write a code segment in Arduino that outputs the magnetometer values of the uncalibrated IMU in the serial monitor of the IDE. c. Calibrate the IMU.	Software	15

	d. Run the program again. e. Compare the calibrated magnetometer data with the actual data. f. Offset any errors.		
Microcontroller 1. Microcontroller needs to run reliably for two hours at an overclocked speed.	1. Verification Process for Item 1: a. Send the head orientation data via USB to the virtual environment from the Teensy and run the simulation for two hours.	Software	10
Xbox Kinect 1. The sensor must have a detection range of at least 2m. 2. The sensor must accurately detect the gestures 85% of the time 3. The average latency of the sensor must be < 100 ms for a program running at 30fps	1. Verification Process for Item 1: a. Hold the Kinect at a fixed position. b. Mark a 2m, 3m, and 4m points away from the Kinect using a measuring tape. c. Check that the Kinect can detect the user while standing at each of these points. 2. Verification Process for Item 2: a. Stand 2 meters away from the Kinect. b. Perform the same gesture 20 times. c. Record the success rate. 3. Verification Process for Item 3: a. Poll a USB mouse using a designated application. b. Poll the Kinect click gesture inside Unity. c. Subtract the difference. d. Add 20ms to the result to account for worst case scenario.	Software	40

5 Cost and Schedule

5.1 Cost Analysis: Parts List

Item	Description	Brand	Unit Cost	Quantity	Cost
HMD	6" 2K LCD Panel	Topfoison	$100.00	1	$100.00
HMD Housing	HMD Housing with Lenses	ViewMaster	$12.00	1	$12.00
Microcontroller	Teensy Microcontroller	PJRC	$19.80	1	$19.80
IMU	9-Axis 9 DOF 16 Bit Gyroscope Acceleration Magnetic Sensor 9-Axis Attitude	HiLetgo	$8.49	1	$8.49
Detachable Headband	Elastic Detachable Headband for Google Cardboard	IHUAQI	$6.99	1	$6.99
Xbox Kinect	Kinect Sensor for Xbox One	Microsoft	$49.98	1	$49.98
Cables	Lengthy HDMI and USB Cables	Several	$20.00	3	$20.00
Shipping	Shipping of all Parts	-	-	-	$75.00 (estimated)
				Total Cost	$292.26

5.2 Schedule

Week	Task	Delegation
01/27	Making the proposal presentation	All
	Writing the final proposal	
02/03	Designing and ordering the acrylic spacer for the headset	Khaled
	Programming the Kinect gestures	
02/10	Create a dark fantasy theme for the virtual environment	Li
	Implement the game logic	
	Calibrate the IMU magnetometer	Khaled
	Creating the specification sheet	All
02/17	Writing the user manual document	All
	Send head orientation data from the IMU to the virtual environment	Khaled
	Programming the positional tracking of the Kinect	
	Headset build	Anqi
02/24	Headset test	Anqi
	Implement game menus and obstacles inside the virtual environment	Li
	Program the score keeping and enemy A.I.	
	Programming the skeletal tracking	Anqi
	Character Animation	Khaled
03/03	Test the accuracy of the Kinect gestures	Anqi
	Test the range of the Kinect	
	Test the latency of the Kinect	
	Mid semester demo	All

03/17	Designing the project logo	All
	Finish testing the rest of the hardware components	All
	Writing the formal testing procedures	All
03/24	Program dynamic health for the game	Li
	Start writing the project report	All
03/31	Making the Open House poster	All
04/07	Open House demo	All
04/14	Final report draft	All

6.0 Miscellaneous

6.1 Demonstrating the OpenVR Prototype

Once we are done building the OpenVR prototype, we will program an endless runner game like Temple Run to demonstrate the capabilities of the system. Our choice for selecting such genre is because of the interactive nature of the gameplay and the beautiful scenery. The game will have several gameplays, each demonstrating a feature of OpenVR such as gesture control of the characters, avateering, changing the camera angle with the player's head movement, and a few more. The demonstration will take place in the Syracuse University Science and Technology Center in April of 2019. Lastly, this game is not part of OpenVR and hence its source code won't be shared.

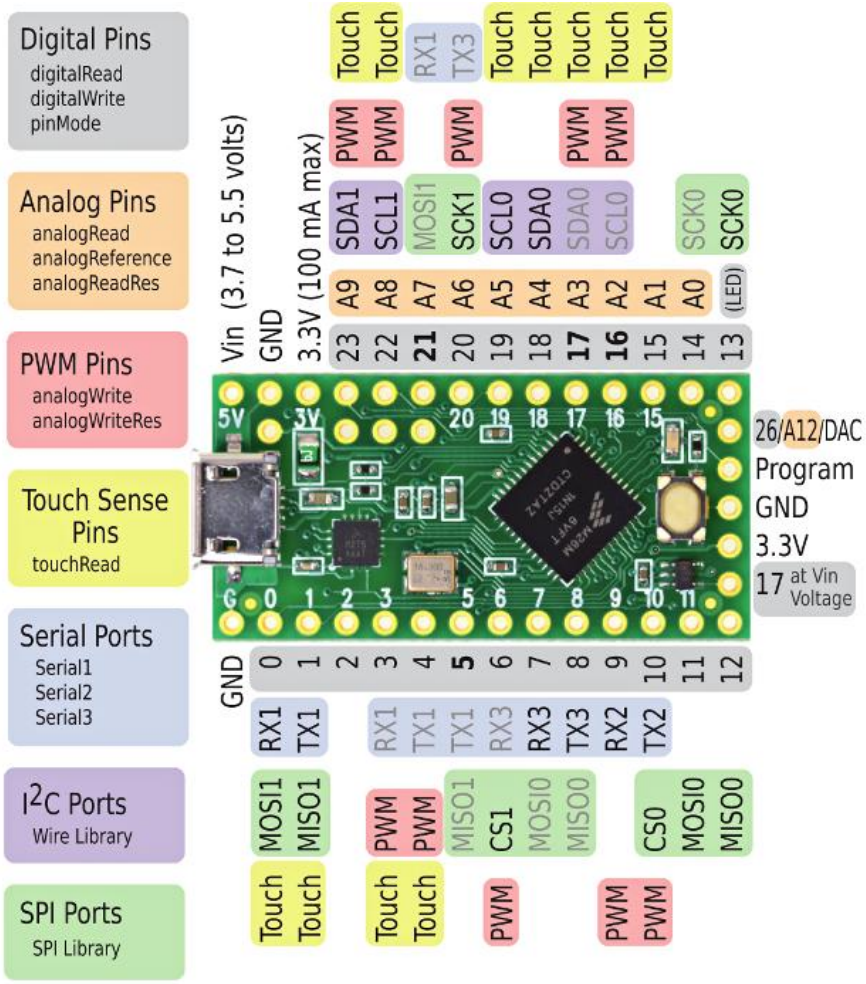
Appendix A: OpenVR Hardware Teardown







Appendix B: Teensy Pinouts



References

[1] K. McMillan. (2017). Virtual reality, augmented reality, mixed reality, and the marine conservation movement [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aqc.2820

[2] TopFoison, "5.9inch 1080p display with 1920x1080 MIPI display," TF60001A datasheet, March. 2016.

[3] Amazon. (2016). View-Master Deluxe VR Viewer [Online]. Available: https://www.amazon.com/Mattel-DTH61-View-Master-Deluxe-Viewer/dp/B01CNSO79Q

[4] Gizmodo. (2016). The New View-Master Deluxe VR Is The Best Cheap VR Headset [Online]. Available: https://www.gizmodo.com.au/2016/08/the-new-view-master-deluxe-vr-is-the-best-cheap-vr-headset-available/

[5] InvenSense, "MPU-9250 Product Specification Revision 1.1," MPU-9250 datasheet, June. 2016.

[6] PJRC. (2014). Teensy 3.2 & 3.1 - New Features [Online]. Available: https://www.pjrc.com/teensy/teensy31.html

[7] Techradar. (2013). Kinect 2 for Windows moving to developers this November [Online]. Available: https://www.techradar.com/news/gaming/consoles/kinect-2-for-windows-moving-to-developers-this-november-1162066

[8] J. Malloch. (2013). Arduino IMU [Online]. Available: https://github.com/malloch/Arduino_IMU