Step MDR Report

Ryan Daly, EE, Jared Ricci, EE, Joseph Roberts, EE, Steven So, EE

***Abstract*— Step is a virtual reality system that will change the way users interact with virtual worlds through enhanced immersion. Unlike most virtual reality systems, the user's movements will play a role in the virtual environment, as a user's walking, running, and other physical movements will correspond to movements in the virtual world. While making virtual reality more realistic, it will also improve user's health and provide a platform for realistic first responder training.**

I. Introduction

Modern virtual reality (VR) applications fail to engage the user in a truly realistic way. They are static experiences that create environments that move around the user as opposed to dynamic experiences that allow the user to move through the environment. Peripheral devices currently on the market, such as Google's Daydream controller [1], try to improve this static experience by adding motion based input to make the user feel more immersed, yet still do not allow the user to move through the environment or have truly realistic interactions.

Step is designed to free the user and allow them to explore virtual worlds in a natural and intuitive way. The user will be able to interact, move, and feel the environment around them without wearing any additional hardware beyond the VR headset. The user's hand and arm movement is detected using Microsoft's Kinect camera while the user moves on an elliptical. Motion on the elliptical is translated into motion in the virtual environment. The virtual environment is developed using the Unity game development suite and deployed to an Android smartphone placed inside the user's headset.

The societal significance of fixing this problem is that virtual reality could be used as a tool which engages people to exercise more, gives more educational experiences, and provides adequate first responder training. Obesity is a growing problem in America. Obesity can lead to other illnesses from chronic to acute, such as diabetes, high blood pressure, and even cancer [2]. If virtual reality incorporated the actions made by people outside the game then more people would be inclined to play the game and exercise. Virtual reality could also be an educational tool where people can visit other places without physically being there. For instance students can take virtual tours and feel as if they are actually touring the area without having to travel. A significant social impact in making virtual reality more immersive is virtual reality can be used in first responder training. "One of the study's findings was that Soldiers lack access to realistic TC3 simulation that could improve the individual and collective skills Soldiers and squads need to manage the complex environment of simultaneous combat and casualty management," says Mike Casey from the Combined Arms Center US ARMY [3].

The system specifications are shown in Table 1.

Table 1. System Specifications.

Specification	Value
End-to-End Latency [4]	< 200 milliseconds
Frame rate [5]	60 frames per second
Speed Accuracy	± 0.5 MPH
Depth Accuracy	< 1 inch st. deviation
Reset Button	Reset at any time

The above specification for speed accuracy was chosen because ±0.5 MPH is sufficiently accurate to gather useful information such as average speed or distance traveled in a given session rounded to the nearest whole number. Depth accuracy was chosen as a requirement regarding the hand and arm movement because the user is going to be reaching forward while using this system and the difficulty of depth measurements [6] indicates the need for accuracy of movement in the plane parallel to the face of the camera. A one-inch standard deviation was deemed adequate to accurately reflect depth. Based on the findings in reference [4], it was determined that end-to-end system latency needed to be less than 200ms in order to make the virtual environment appear smooth. 60 frames per second was chosen, as reference [5] makes it clear that virtual reality games that run below 60fps can result in users becoming motion sick. A reset button is also listed as a specification to allow the user to reset the system at any desired time.

II. Design

A. Overview

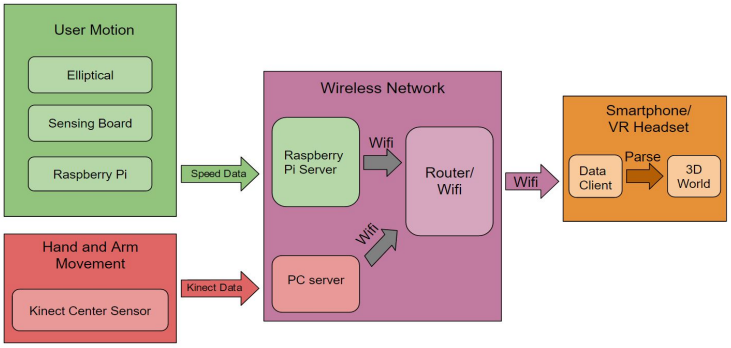

Fig. 1. System Block Diagram

This project has four distinct parts, as the block diagram in Figure 1 illustrates. The first block is the user motion subsystem. This consists of a compact elliptical, a sensing board, and a Raspberry Pi. The purpose of this block is to allow the user to roam freely and determine their speed so it can be used in the virtual environment. The second block tracks the hand and arm movement using a Kinect sensor and sends the coordinates of the hands to the virtual environment. This coordinate data gives the hands a useable virtual interpretation. The third block is the wireless network which takes the data input from block one and block two, and sends it to the virtual environment running on the smartphone. The Raspberry Pi has a server that handles the user motion data, and a PC has a server to interface the Kinect. These are sent on a private router to the smartphone. The smartphone parses through the data it received and uses it as user controls in the virtual environment. The 3-dimensional environment runs on the the smartphone, which is inserted into a headset for the user to wear.

B. User Motion

The User Motion subsystem must calculate the user's speed in MPH as a function of the rotational speed of the elliptical. This is accomplished by placing a series of magnets on the wheel of the elliptical and detecting how often they pass by.

The system uses four magnets mounted at 90° increments on the wheel while a Melexis US5881 Hall Effect sensor [7] is used to detect the presence of a magnet. Every step on the elliptical corresponds to roughly six rotations of the wheel, meaning that the sensor detects a magnet about 24 times per step. This results in the user's speed being updated every 42 ms which is fast enough to accurately reflect the user's real world movement in the VR game, and fulfills the desired specification of having speed accurate to within ± 0.5 MPH.

The hall effect sensor is connected to a Raspberry Pi which runs a Python program that calculates the speed in MPH based on the frequency that magnets pass by. The Raspberry Pi additionally runs a secondary Python program that connects to the smartphone, on which the VR game is running. This program sends the current speed of the user upon request to the game. The phone then uses this data to move the user in the VR game.

The connection to the smartphone is established via TCP over WiFi. The average latency of the connection was found to be 70ms across 100 TCP packets sent between the Raspberry Pi and the Smartphone.

The movement speed is calculated using the following expression:

$$MPH = \frac{RPM * Diameter\ (feet) * \pi * 60\ seconds}{5280\ feet/mile} * \frac{1}{24}$$

A sample of calculated speeds was gathered by modifying the Python program to output the current speed every second. These calculated speeds were then compared against the speed displayed by the elliptical's built-in computer. The speed displayed by the elliptical is being used as the reference speed for this comparison. The calculated speed is accurate ± .5 MPH as shown in the visualization below. Standard deviation was calculated as 0.152 MPH.

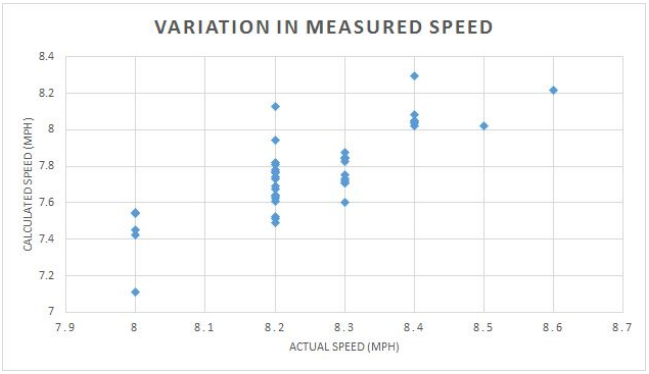

Fig. 2. Variation in speed calculated by the User Motion Subsystem at differing speeds.

C. Hand & Arm Movement

The Hand & Arm Movement subsystem has two primary requirements; it must be accurate and time efficient. The Microsoft Kinect 360 is used for this [8]. It can interface with a PC and is programmable in C#. The Kinect is often

criticized for its poor depth performance [9], so the set specification is a depth accuracy to within a 1 inch standard deviation. If the depth performance is sufficient, then the accuracy of the device is sufficient.

The steps to using and interfacing the Kinect are to first create the server on the PC and then extract the position of the body parts being tracked. The server then waits for a request from the game to send data, and when received, writes the position data as a string onto the server. The game then reads and parses this data and uses it in the virtual world.

The internal processing of the Kinect includes a "skeleton" model of the user. Many of the body parts are recognized and considered a "Joint type" object with certain attributes, one of which is position. Examples of these joints are the hands, elbows, shoulders, hips, and head. Figure 3 shows a typical example of the skeleton generated. The green circles indicate the joints that are considered "Joint type" objects and are connected with a "bone" which is just a segment connecting two joints.

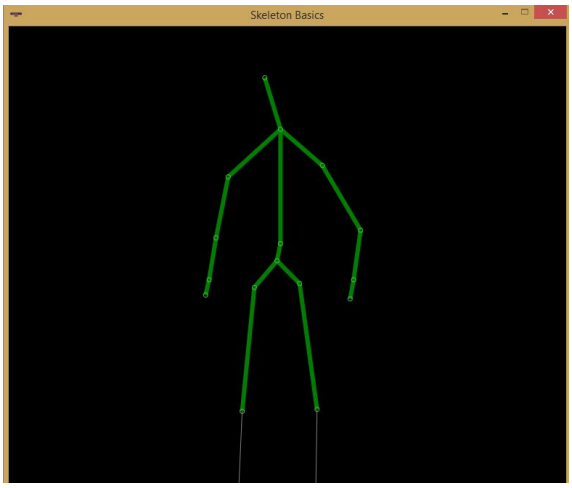

Fig. 3. Skeleton model of a person generated by the Kinect.

Each of these joints include an attribute in (x,y,z) coordinates that describe the position of each joint in reference to the front of the Kinect. The x-coordinate indicates the horizontal displacement of a joint from the perspective of the Kinect. The y-coordinate indicates the vertical component, and the z-coordinate indicates the depth. All three of these are measured in meters. The z-coordinate, the depth, is used to measure the accuracy of the Kinect. To test the accuracy, two different depth measurements were marked off by hanging wire from the ceiling. The two wires were measured and strung at 1 meter and 2.25 meters away from the front of the Kinect. The computer printed out the coordinates of the right hand as the person in front of the Kinect reached forward for the wire at 1 meter and backward for the wire at 2.25 meters. The tester then recorded the z-coordinates into excel and compared the values to the measured distance. One hundred measurements were taken for each wire and the results are summarized in Table 2. Bearing in mind that the requirement is to keep the depth performance within a 1-inch standard deviation, the results in Table 2 indicate that the accuracy of the Kinect is sufficient for the purpose of this project.

Table 2. Results from Kinect depth performance experiment.

Tape Measured	Calculated Kinect mean (n=100)	Standard Dev. (m)	Standard Dev. (in)	Performance <inch (=0.0254 m)	Performance <4cm
1.0 m	1.0021 m	0.0214	0.844	80%	93%
2.25 m	2.2452 m	0.0218	0.859	77%	96%

With sufficiently accurate data, the second requirement is that the processing must occur seemingly in real-time. Standard video is typically shot at 30fps, and this appears smooth to the average eye. The Kinect processes the skeleton data at this same rate [10], but commonly, virtual reality is run at a much higher frame rate. Although 30fps may be satisfactory, it would become a bottleneck if the game needs to wait for new data to redraw the frame.

In this situation, the frame rate would be capped at 30fps. There are two possible ways to avoid this issue. The simpler way is to use stale data if the frame rate is running high enough above the 30fps data rate. Essentially, this method would draw multiple frames using the same position data. Focusing on the hands, if the final frame rate is not high enough, then the movement of the hands would appear to lag. As an example, assume that the frame is drawn twice for every request to the Kinect. If the final frame rate is at 36fps (higher than the 30fps bound caused by the Kinect), the Kinect data is only updating at 18fps, which is slow enough to appear lagged. This is the easiest solution to employ, but it can lead to issues of lagging hand and arm movements.

The second solution is to employ a predictive approach to the joint position. The most basic approach is to linearly extrapolate the position for some future time and then write this data to the server. An example of the linear model would require two Kinect-processed data points, if the current point is the point calculated at time t, then the model would also use the position of the joint at time $t-1$. First, note that because the Kinect runs at 30fps, or 30 Hz, that time t and $t-1$ occur 1/30 seconds apart (~33.33 ms). This model should extrapolate the data between time t and time

t+1, or half of the time between one cycle of the Kinect. The x, y, and z coordinates can be linearized between these two times, so the difference x(t) - x(t-1) (Δx) indicates the change along the x-axis over the time difference. The position one-half of a cycle later could then be estimated to be one-half of the difference [(Δx)/2] in the same direction of the change. This approach would allow for the game to call to the Kinect once and pull two coordinates for the particular joint.

Implementing the Kinect has worked thus far, but a future issue may be "self-occlusion" which is the blocking of a joint of interest by the rest of the body. An example of this would be reaching behind your back or standing at an angle to the Kinect. The Kinect needs to be facing the front of the user to work accurately. If the user remains fixed, then this should be of no concern, but if the user would ever pivot, then the Kinect would have to follow. An avenue for next semester is rotating the person and elliptical when the user indicates that they wish to turn. The world would turn, but they would also physically pivot. This is a large mechanical project, but the control to determine if the user wishes to turn may be possible by using the Kinect to track the hip joints. Canceling out the hip displacement due to walking in one dimension, the amount of the rotation of the hips would indicate how much the user turns, both in the game and in real life. This turning capability of the project requires extensive fine-tuning, but may be possible without additional sensing hardware.

D. Network

The wireless network required for our system allows reliable communication between all the input and output components of the system. The I/O components consist of the PC, Raspberry Pi, and the Android phone; they all communicate via WiFi through a router. TCP/IP protocols were established in order to make sure that the communications between the devices were stable. This was done by creating TCP servers and clients. The PC and the Raspberry Pi were established as the servers and the Android phone was established as the client.

A local network using the router was established in order to allow communication between the devices. Attempting to use a pre-existing network that the school provided was inadequate. When clients try to access other servers through the secured network provided by a third party source, in our case by the university, there was no problem, but when a server was established for the system, the secure network would not allow other clients to access the server. The solution was to establish an independent WLAN, or wireless local area network. In doing so, there was freedom with how the TCP servers and clients were created.

The network must be able to provide quick transmission of data through the wireless interface. WiFi and Bluetooth were the two clear solutions. The reason that transmission via WiFi was chosen for the system over Bluetooth is WiFi has lower latency and higher data throughput. To see this, observe the table below.

Table 3. Comparison of WiFi and Bluetooth

	Bluetooth 4.0 [11]	WiFi (802.11n) [12]
Frequency	2.4GHz	2.4/5GHz
Latency	~100ms	1-10ms
Data Rate	25Mbps	600Mbps

Since the overall system quality will be highly dependant on the efficiency of data transactions, minimizing the latency through the interfaces is key; therefore, choosing WiFi over Bluetooth is the better choice since the typical maximum latency of a 5GHz WiFi network is an order of magnitude less than the average latency of Bluetooth.

Once the system was completed and the phone (TCP client) could successfully retrieve data from the PC and the Raspberry Pi (TCP servers), measurements were taken to observe the end-to-end latency of the system. The measured latency from the data of two inputs, the kinect and the elliptical, which are processed by the PC and Raspberry Pi respectively, to the Android phone game is around 163ms. This latency is satisfactory in maintaining a realistic feel and control of the virtual world.

This latency was calculated by taking a slow motion video, at 240fps, of the phone screen and the user in the same frame and manually counting how many frames it took for the phone screen to mimic the user's body movements. The video was extracted into individual frames by using MatLab and then the frames were manually counted. The average frame count was around 39 frames, therefore end-to-end latency = 39 frames * 1/240 seconds per frame = 163ms. This is significant because reference [3] indicates that 166ms is approximately where gamers would notice controller lag, and a goal of this project is to appear real-time (i.e. no lag).

E. Smartphone Application and Headset

The smartphone application serves as the user's means of vision into the virtual world. This is accomplished by placing the phone inside of a virtual reality headset, inside which the phone screen is oriented towards the user's eyes, effectively turning the phone screen into the user's full range of vision.

The phone application is required to be able to generate a 3D virtual world, receive and process data sent from the PC and Raspberry Pi, and maintain a frame rate of 60fps. This is accomplished through use of the Google VR Standard Development Kit (SDK) [13] for the Unity game engine. The Unity game engine [14] is used to build a 3D environment that can be exported into a phone application. The Google VR SDK, when imported into Unity, allows for the creation of a VR environment by splitting the screen into two sections for each eye in the headset (Fig. 4) and utilizing the phone's accelerometer to orient the image in the direction that the user is facing.

Within the Unity game engine, supported by the Google VR SDK, a virtual world is generated in which objects, textures, and sounds can be imported and manipulated with scripting. For the purpose of making a game to demonstrate the system functionality, an environment was created to simulate the user being in a valley with grass and trees (Fig 4).

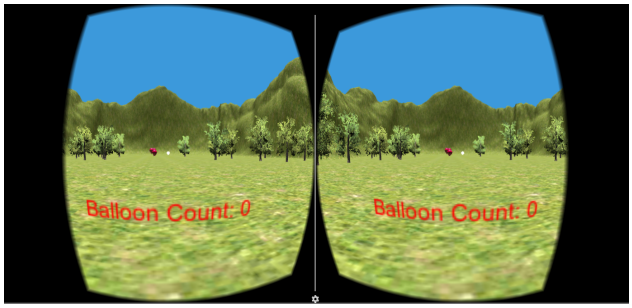

Fig 4. Phone screen with application launched

Within this environment, objects can be imported and manipulated by use of C# scripting. This includes the camera object, which is essentially the portal in which the user sees into the environment (as seen in Figure 4), a multitude of graphical objects, textual objects, light objects, etc.

To establish a connection between the application and the PC and Raspberry Pi, a TCP client script is started when the application is launched. The TCP client sends a request to connect to the Raspberry Pi and PC servers, and once the request is accepted, a data stream is opened for each device. The client script then requests data from each device through each stream. This request is sent on every frame render, to ensure that the data coming from each device is synced with the data available in the application. The client is essentially sending a signal to the servers to indicate when it is ready for new data (each frame render), and the server is sending the data if and only if the client requests it. This avoids any issue with data being sent faster than the speed at which the application can use it.

This byte data is then parsed into usable variables that can be applied in scripts attached to game objects. From the elliptical (Raspberry Pi), a speed variable is used to vary the speed at which the user moves through the environment. This variable is used in a script attached to the camera object, thereby moving the camera at the same speed that the user is moving. This provides the appearance of moving in the game at the same speed that the user is moving on the elliptical. From the Kinect (PC), three coordinate variables are used for each hand to vary the x, y, and z coordinates of the left and right hands in the virtual world. A relative coordinate system is established within the Unity environment to vary the user's virtual hands based on the coordinates received from the Kinect.

The end result is a game that can be launched as a smartphone application, in which the user can stand on the elliptical and move forward in the virtual world at the speed which they are moving on the elliptical, all while using their hands to pop balloons. The game keeps score of how many balloons the user has popped, as a user incentive.

It is important that the frame rate of the application is maintained as close to 60fps as possible [5]. The goal was set at 60fps because 60fps is the maximum possible frame rate on an Android LCD screen, which has a refresh rate of 60 Hz. Limited by vertical synchronization (VSYNC), the smartphone's hardware will not allow an application to render frames faster than the phone screen can refresh. Even if possible to turn off VSYNC, it is not advisable to do so, because screen tearing may result [15]. Meeting this requirement was first approached by ensuring that both servers were able to send data at a minimum rate of 60 Hz. This was an issue only with the Kinect, which was addressed in Section C. To verify that the frame rate can reach 60 fps, the application was simulated within the Unity program. Frame rate was noted to drop when a large number of animated objects were present on the screen. This result must be taken into consideration in all future game development that utilizes this system. Another consideration is the smartphone which the application is

being run on. If the phone has a low clock speed on the central and graphics processing units, then frame rate may drop as well. In trials through the Unity program, 60fps was consistently achieved with the demonstration game.

III. Project Management

Table 4. MDR Goals

MDR Goals	**Status**
Precise Speed Control	Complete
Kinect Motion Sensing	Complete
Data Processing and Transmission to Smartphone	Complete
VR Environment with Inputs	Complete

The team members of this project are working well together. Communication and morale levels are high amongst all members. The team has two meetings per week, one with Professor Goeckel and one without. The interfaces between the blocks and analysis of data involved significant team collaboration. Solutions to difficult parts of the individual blocks were also considered as a team. Efforts are divided evenly and each team member has been successful in reaching their goals.

The speed control has been accurately measured, using the on-board elliptical speedometer as the ground truth. The Kinect motion sensing can accurately track the position of the hands and other joints, and the position is extracted and used in the game. The data processing and transmission to the smartphone is composed of two working servers and a client, that communicate through the wireless interface following TCP/IP. The 360 degree virtual environment programmed onto the phone can accept and use the input data.

Jared Ricci's expertise lies in the elliptical and Raspberry Pi system (Section II, B). Joe Roberts's expertise lies in the Kinect sensor and PC system (Section II, C). Steven So's expertise lies in the system networking between devices (Section II, D). Ryan Daly's expertise lies in the smartphone application game development (Section II, E).

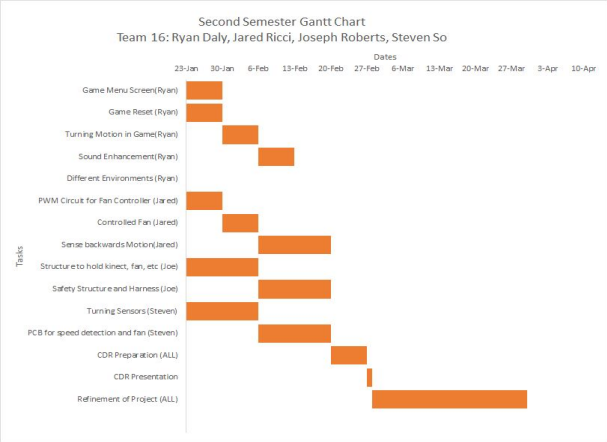

Fig. 5. Step Gantt Chart

IV. Conclusion

At this point in the project, all that has been set out to deliver by MDR was accomplished. The user is able to play a virtual reality game in which they can control their speed via elliptical and have their movements reflected in the virtual world. At this point, the user is able to only move forward and in one direction.

A future goal is to be able to move backwards, and turn in any desired direction, potentially having the elliptical physically turn as the user turns. This is perceived as being the most difficult goal, as the turn detection is expected to be a challenge. Another goal is to have a variable fan in front of the user, so the user may experience the movement of air as they naturally would if moving forward. This will all be accomplished with the already established structures and interfaces that have been created.

Acknowledgments

Professor Dennis Goeckel, Project Adviser
Professor Weibo Gong, Evaluator
Professor Sandip Kundu, Evaluator
Fran Caron, Lab Manager
Professor Christopher Hollot, SDP Professor

References

[1] "Google Daydream", Google, 2016. [Online]. Available: https://vr.google.com/daydream/. [Accessed: 03-Dec-2016]

[2] Begay, Dion. "What Can Obesity Lead To?" Obesity in the Latino Population. N.p., Spring 2005. Web. 13 Oct. 2016.

[3] Casey, Mike. January 25, 2016. "Improving Tactical Combat Casualty Care to save Soldiers' Lives."Www.army.mil. N.p., 25 Jan. 2016. Web. 13 Oct. 2016.

[4] R. Leadbetter, *Eurogamer.net*, 2016. [Online]. Available: "Console Gaming: The Lag Factor", Eurogamer.net, 2016. [Online]. Available: http://www.eurogamer.net/articles/digitalfoundry-lag-factor-article. [Accessed: 01- Dec- 2016].

[5] C. Hall, "Sony to devs: If you drop below 60 fps in VR we will not

certify your game", *Polygon*, 2016. [Online]. Available: http://www.polygon.com/2016/3/17/11256142/sony-framerate-60fps-vr-certification. [Accessed: 13- Oct- 2016].

[6] B. Langmann, K. Hartmann and O. Loffeld, "DEPTH CAMERA TECHNOLOGY COMPARISON AND PERFORMANCE EVALUATION", University of Siegen, Siegen, Germany, 2012.

[7] Melexis, “Unipolar Hall Switch - Low Sensitivity,” US5881 datasheet, Mar. 2012

[8] "Kinect for Windows Sensor Components and Specifications", *Msdn.microsoft.com*, 2017. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed: 01- Feb-2017].

[9] D. Pagliari and L. Pinto, "Calibration of Kinect for Xbox One and Comparison between the Two Generations of Microsoft Sensors", *Sensors*, vol. 15, no. 11, p. 27571, 2015.

[10] "Kinect for Windows Sensor Components and Specifications", *Msdn.microsoft.com*, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx. [Accessed: 30- Nov-2016].

[11] Bluetooth Core Specification, Bluetooth 4.0, 2010.

[12] IEEE Criteria for 802.11n WiFi, IEEE 802.11n, 2009.

[13] "Google VR SDK for Unity," in *developers.google.com*, 2016. [Online]. Available: https://developers.google.com/vr/unity/.

[14] "Unity Scripting API," in *unity3d.com*, 2017. [Online]. Available: https://docs.unity3d.com/ScriptReference/.

[15] "Adaptive VSync | Technology | GeForce", *Geforce.com*, 2016. [Online]. Available: http://www.geforce.com/hardware/technology/adaptive-vsync/technology. [Accessed: 05- Nov- 2016].