Taking the VRduino Wireless
EE267 Virtual Reality: Final Course Project
Course Instructors: Gordon Wetzstein, Keenan Molner, Hayato Ikoma, Robert Konrad

Meera Radhakrishnan
Stanford University
mradhak@stanford.edu

Abstract

The VRduino is a hardware and software system developed by the EE267 course staff to serve as an instructional and development tool for virtual reality. This project evaluated different methods to make the VRduino system completely wireless and replace the LCD screen of the system with a mobile phone running the Chrome Web browser. In particular, it evaluated the following three technologies: "classic" Bluetooth, Bluetooth Low Energy, and Wi-Fi. Ultimately, a completely wireless VRduino system was built by connecting the VRduino board and computer via a virtual serial connection over Bluetooth, and connecting the computer and mobile phone via Wi-Fi.

1. Introduction and Motivation

The VRduino is a tool developed by the EE267 course staff for instruction and development in virtual reality. The current setup for the VRduino consists of the VRduino board itself connected to a computer via a USB connection, as well as a mobile phone-sized LCD screen connected to the computer as an external monitor via two additional cables. Removing the need for the three cables and the LCD screen could introduce improvements in mobility for users of the course hardware and accessibility to the hardware itself.

Therefore, this project sought to achieve the following objectives. First, I sought to evaluate different methods to (1) make the VRduino system wireless, and (2) replace the LCD screen with a mobile phone. Next, I sought to prototype an effective wireless VRduino system, and display a virtual scene on a Chrome mobile browser that updates in real time with wirelessly transmitted inertial measurement unit (IMU) data from the VRduino.

2. Previous Work

The VRduino hardware and software possess a number of functionalities. In my project, I sought to evaluate how wireless functionality would most effectively and efficiently interface with the existing VRduino setup.

The VRduino system consists of the following elements:

First, the VRduino board itself is comprised of a Teensy microcontroller connected to an inertial measurement unit (IMU) via an I2C connection, as well as four photodiodes which can be used for beacon-based positional tracking. After some processing, data from the IMU and photodiodes are sent from the Teensy via a USB (serial) connection to a computer. On the computer, the data can be accessed by monitoring the serial stream coming through the virtual COM port that the Teensy is connected to. The board is meant to be mounted on the front of a Google-Cardboard style stereoscopic viewer, so that it can be used to track the orientation and position of the viewer's head.

Next, the VRduino software accesses the data, and uses the information to manipulate a virtual scene rendered using the WebGL JavaScript library in a Chrome browser window on the computer. In particular, the *Homework 5: IMU and Sensor Fusion* program uses Node.js to access the serial data arriving from the Teensy through the USB port and send that data to the browser via a WebSocket server. [1] In the browser, locally stored HTML and JavaScript files display a virtual scene (a set of coordinate axes), which rotates in real time in response to the incoming orientation data from the WebSocket. One of multiple orientation modes can

be selected, including Euler angle-based rotation and quaternion-based rotation.

Finally, a small LCD screen (approximately the size of a mobile phone) is connected to the computer as an external monitor via a USB cable (for power) and an HDMI cable. After starting the Node server, the user can simply open the relevant HTML file containing the virtual scene in the browser, drag the browser into the LCD screen from the PC, and place the LCD screen in a stereoscopic viewer (a ViewMaster was used for this course) that has the VRduino mounted on it. The scene then responds to movement of the user's head. [1]

While this setup offers great flexibility in terms of the user's ability to interact with different aspects of VR at a low level, it could be improved by eliminating the need for the three cables to the computer, and replacing the LCD screen with a mobile phone running the virtual scene in its browser. This would allow for less cumbersome operation of the setup, as well as greater accessibility to the requisite hardware for those who might not be able to easily acquire an LCD screen. As a result, my project seeks to build on the current setup and evaluate options for achieving these two goals.

3. Methods

The ultimate objective of this project was to determine an effective way to make the entire VRduino system wireless, and display virtual scene rendered using WebGL on a mobile browser rather than a computer browser displayed on an external wired display. First, different options for the overall data flow (from VRduino to phone browser) were evaluated, and one was selected. Then, different technologies - namely Bluetooth, Bluetooth Low Energy, and Wi-Fi via the ESP8266 chip - were evaluated in the context of the selected implementation. Finally, a fully wireless system was built using the selected technology by connecting the hardware and applying the appropriate software changes to existing VRduino code.

3.1. Determining the Flow of Data

Two main data flow schemes were considered for the wireless VRduino system: (1) connecting the VRduino directly to the mobile phone browser, or (2) having the data flow from the VRduino wirelessly to the computer, and then have the computer serve the virtual scene and orientation data to the mobile browser via an HTTP server.

Ultimately, the second scheme presented greater advantages within the scope of the project. In evaluating the first scheme (direct communication from the VRduino to the phone browser), Google Chrome's Web Bluetooth API presented a promising option for having the phone browser interact directly with a Bluetooth module connected to the Teensy on the VRduino [2]. However, the Web Bluetooth API is meant to connect to Bluetooth Low Energy devices (as opposed to classic Bluetooth), and does not directly support a serial stream of data in the style that would most efficiently interface with the rest of the VRduino code. Furthermore, the documentation suggested that while Android Marshmallow was well-supported, iOS support was not as fully developed.

As a result, the decision was made to evaluate wireless methods of communication in the context of seeking to have the Teensy communicate data wirelessly to the computer, from where the computer could use a modified version of existing VRduino code to serve both the virtual scene and the VRduino data to a phone connected on the same Wi-Fi network.

3.2. Establishing Computer-to-Mobile Browser Communication

The first step to evaluate this scheme was to test whether the phone could be wirelessly connected to the computer, and whether the computer could serve both the virtual scene and orientation information to the phone over Wi-Fi. For this test, the physical serial connection (over a USB cable) between the VRduino and the computer was maintained.

For this test, existing orientation tracking code (*Homework 5: IMU and Sensor Fusion*) for the VRduino was modified to allow the computer to wirelessly serve the virtual scene to the phone browser. Currently, a JavaScript program in the existing *Homework 5* code (server.js) establishes a

WebSocket connection with the local machine's port 8081. Using the SerialPort library, it acquires orientation data incoming from the Teensy through the serial port it is connected to and sends it to localhost:8081. Then, the rendering program (axisRender.js) pulls the information from the WebSocket and uses it to update the rotation of a displayed set of coordinate axes. The user can view the display by simply navigating to the corresponding HTML file (axis.html) on their computer. [1]

However, in order to serve the rendering files to a mobile phone, the mobile phone needs to be connected to the same Wi-Fi network as the computer, and the computer needs to have an HTTP server that serves the relevant JavaScript and HTML files. (The phone cannot simply access local files stored on the computer arbitrarily through the shared Wi-Fi connection.) In order to achieve this, the HTTP server functionality from *Homework 7: Spatial Audio* was inserted into the *Homework 5* server.js file [1, 3]. Furthermore, the file structure of the *Homework 5* code was modified so that all files to be served to the phone were located in a "public" directory. The public directory was then placed in the same folder as server.js. The HTTP server can then serve the JavaScript and HTML files to a client that accesses it through the shared Wi-Fi connection.

The previous (wired) version of the axisRender.js code for *Homework 5*, which opens the WebSocket and collects the orientation information to render the coordinate axes in the browser, opens the WebSocket using the address localhost:8081 [1]. Furthermore, stateController.js, which initiates the WebSocket and sends the orientation data to it, similarly assumes a local address when initiating the WebSocket. However, using this address assumes that the program is being called from the same machine as the WebSocket server. As a result, in order to make it compatible with the phone, the WebSocket address in axisRender.js was changed from localhost:8081 to IPAddress:8081 (where IPAddress is the IP address of the computer serving the files, as assigned by the shared Wi-Fi network).

Finally, to actually test the setup, the orientation tracking Teensy code from *Homework 5* was uploaded to the VRduino (which remained connected to the computer via USB), the computer was connected to a Wi-Fi hotspot on the mobile phone, server.js was run using the Node command prompt, and then the rendering page was navigated to on the Chrome browser on the phone by typing IPAddress:8080/axis.html (where IPAddress is the address of the computer, and axis.html is the file containing the virtual scene).

This setup successfully allowed the phone to wirelessly display a set of coordinate axes served by the computer, with real-time updates from the (wired) VRduino to rotate the axes.

3.3. Evaluating Wireless Technologies for Teensy-Computer Communication

After successfully establishing a wireless connection from the computer to the phone, the remaining step was to determine the most effective way to communicate between the Teensy on the VRduino and the computer, in a way that would interface correctly with the connection between the computer and the phone downstream. Ideally, a device that could wirelessly interface with a virtual COM port on the computer would allow the code downstream to remain unchanged and allow for a smooth transition to a completely wireless system.

There exist a number of wireless modules designed to allow microcontrollers to connect wirelessly with other devices. For this project, I evaluated three potential methods of communication between the Teensy and the computer: Bluetooth Low Energy (via the nRF51822 Bluetooth radio), classic Bluetooth (via the HC-06 module), and Wi-Fi (via the ESP8266).

3.4. Comparison of Classic Bluetooth and Bluetooth Low Energy

The Bluetooth specification contains two main Bluetooth standards: "classic" Bluetooth (which includes Bluetooth specifications 1.0 to 3.0) and Bluetooth Low Energy (which was introduced with Bluetooth specification 4.0 and is also known as Bluetooth Smart) [4].

Bluetooth Low Energy (BLE) devices are markedly different from classic Bluetooth devices and are intended for a different subset of applications [9]. For my project, I sought to evaluate these two

technologies against each other in the context of the VRduino wireless setup.

While classic Bluetooth devices can pair with a computer and can immediately transmit raw data through a virtual COM port, BLE devices do not have this capability. This is because classic Bluetooth devices possess a Serial Port Profile (SPP) to define behavior for virtual serial port connections, but BLE devices do not [5].

Furthermore, BLE devices are typically intended for low-power applications, in which data transfer is not continuous and do not require high speeds. As a result, data speeds are slower than those for classic Bluetooth [6].

On the other hand, classic Bluetooth boasts higher data rates but also higher power consumption. Classic Bluetooth is intended for applications in which large amounts of data are continuously streamed and availability of power is not a significant issue [4,8].

From this comparison, it is evident that classic Bluetooth presents several advantages over BLE in the context of the VRduino. First, the ability of classic Bluetooth to plug into a virtual serial port on a computer is ideal for interfacing with the rest of the VRduino code, since the downstream code is already designed to acquire data from a serial connection to the computer. Furthermore, classic Bluetooth is designed for fast, high-data-throughput applications, whereas BLE is meant to infrequently send small packets of data [7]. As a result, a VRduino wireless system seems to be an appropriate application of classic Bluetooth capabilities.

In order to evaluate each of these technologies in practice, I sought to build a wireless prototype for each of Bluetooth Low Energy and classic Bluetooth. Ultimately, I identified potential methods to implement a BLE solution, but found that classic Bluetooth was indeed a quicker and more ideal way to implement this wireless system, and was successful in building a classic Bluetooth-based prototype.

3.5. Bluetooth Low-Energy with the nRF51822

To investigate a potential method of leveraging BLE for this project, I used an Adafruit Bluefruit LE UART Friend, a BLE module based on the nRF51822 Bluetooth radio [10, 11]. A Teensy microcontroller can be configured to communicate to the device via serial communication through one of its Hardware Serial ports, or via the SoftwareSerial library [12].

Since BLE devices do not possess a Serial Port Profile (SPP), they cannot immediately connect to a virtual serial port on a computer. As a result, a different method of connecting the BLE device wirelessly to the computer was required.

In order to do this, I investigated whether Node.js had support for Bluetooth Low Energy, and if the data output from the BLE device could be acquired by the Node server. The "noble" Node module is a module written to provide BLE support for Node.js [13]. Among its various functionalities, it can be used to read and write characteristics of BLE devices. Furthermore, I identified a software library which demonstrates the use of noble to emulate a serial connection with a BLE device [14].

However, it was ultimately discovered that using the noble Node module seems to require a dedicated Bluetooth 4.0 USB adapter, which was not acquirable during the limited timeframe of the project. As a result, I was not able to further investigate options for implementing BLE-based communication.

3.6. Classic Bluetooth with the HC-06 Module

The HC-06 Bluetooth module is a widely used classic Bluetooth module for microcontroller-based Bluetooth projects. It presents a number of advantages in the context of the wireless VRduino. The device is relatively simple, and like the BLE Bluefruit, can be connected to a Teensy via a Hardware Serial or SoftwareSerial connection. Little to no configuration of the device is required to stream data from it; any information that is sent from the Teensy through the Teensy-Bluetooth module Serial connection is broadcast from the module when it is in DATA mode. The Teensy and HC-06 are connected by attaching the TX (transmit) of the Teensy's Serial

connection to the RX (receive) of the Bluetooth module, and vice versa. [15]

Since classic Bluetooth has an SPP, the HC-06 can connect wirelessly to a computer through a virtual serial port. Data being transmitted to the computer from the Bluetooth module will appear as serial information coming in through a virtual COM port, exactly how it appears when it is entering directly from the Teensy via a USB connection.

First, I tested the HC-06's capability to connect to a computer via a virtual serial port, and print serial information to the port. I connected the HC-06 to the Teensy (after configuring which pins would be used on the Teensy for the SoftwareSerial connection), and then programmed the Teensy to print dummy values to both the Bluetooth module and the Arduino Serial Monitor. Then, I paired the HC-06 to the computer, and opened the COM port that the Bluetooth module was connected to in a terminal program (I used TeraTerm to monitor the serial output through the COM port). When I compared the Bluetooth output to the Serial Monitor output directly from the Teensy, they were identical.

From there, I was able to incorporate the Bluetooth connection into the existing VRduino setup. I modified the existing vrduino.ino program to print orientation information to the Bluetooth module (to the SoftwareSerial connection to the HC-06) instead of the Serial Monitor. Then, in server.js, I modified the code to read information from the COM port to the Bluetooth module instead of the COM port to the Teensy. Finally, I connected a battery to the Teensy and removed the USB cable altogether. From here, I was able to pair my computer with the Bluetooth module, run the Node server as previously described, and wirelessly view the coordinate axes on the browser on my phone, updated in real time.

Ultimately, I was able to successfully build a wireless VRduino system based on a classic Bluetooth connection from the Teensy to the computer.

3.7. Wi-Fi with the ESP8266

A third potential method of communication that I evaluated was a Wi-Fi connection from the Teensy via an ESP8266 Wi-Fi chip. The ESP8266 supports a variety of functionalities and can run as a standalone microcontroller [16]. The setup for the part is far more complex than that of the HC-06 and so it was not further investigated within the timeframe of this project, since the HC-06 setup proved to be functional and efficient.

In order to interface the Wi-Fi chip with the VRduino setup, one could establish a TCP connection between the chip and the Node program running on the computer, and then have the Node program send the information through the WebSocket. Node.js has various modules which can be used for TCP connections. The ESP8266 could be worth further investigation in the future if better data transmission rates are needed than Bluetooth can offer. However, it will require a greater amount of modification of the existing code library for the course, whereas classic Bluetooth-based transmission is already quick to implement and easy to debug.

4. Evaluation

A particular concern for a wireless VRduino is the latency, or the lag between the motion of the VRduino and the motion of the virtual scene on the phone.

From qualitative observation of the virtual scene on a mobile phone, with the phone and computer connected to a Wi-Fi hotspot set up on the phone, there was almost no visually perceptible latency (with a baud rate of 9600 set for the Bluetooth communication). However, this may change depending on whether the phone has good reception of cellular service and may be highly variable between phones. As a result, an external Wi-Fi router may be a better option than a mobile hotspot.

Furthermore, maximum possible Bluetooth transmission speeds are lower than those of Wi-Fi. Maximum classic Bluetooth transmission speeds are on the order of 1-3 Mbps [6]. On the other hand, Wi-Fi connections can theoretically extend to tens of Mbps [17]. (These can be compared to BLE, which supports only short packets at a data rate of 1 Mbps [6].) As a result, if latency becomes a problem due to the Bluetooth connection being the limiting factor, a Wi-Fi-based alternative may be worth investigating.

5. Discussion and Future Work

Ultimately, classic Bluetooth was found to be the most effective method to make the VRduino-to-computer connection wireless, in terms of maintaining functionality and minimizing the necessity of modifying large portions of the current code library. Furthermore, establishing a Wi-Fi connection between the computer and phone to serve the IMU data and rendering files to the phone browser was a logical extension of the current software setup. I was able to build a successful wireless setup on both Android and iOS mobile devices using these methods.

However, this setup does present certain limitations, and potential for future investigation. First, the project investigated the effectiveness and latency of the system with only one VRduino system connected to the Wi-Fi network. Further experimentation should be done to investigate the presence of latency when multiple VRduino systems are connected to the same Wi-Fi connection (as would be the case during class sessions). In addition, while the project confirmed the effectiveness of an incoming connection to the computer, it did not investigate using keyboard input from the computer across Bluetooth to control the Teensy's data streaming mode.

Furthermore, the current wireless setup requires the computer to act as an intermediary between the VRduino and phone browser. Further work could be done to identify ways to eliminate this intermediate step. In particular, the Web Bluetooth API shows promise as a potential way to connect the Chrome mobile browser directly to a Bluetooth device. However, it has limitations of its own; namely, it is intended to work with BLE, and also may not have full support for iOS devices [2]. In further investigating BLE as a wireless option, the noble Node.js module may deserve more investigation. The ESP8266 Wi-Fi module may also present possibilities in terms of eliminating the computer intermediary, and also increasing transmission speeds from the VRduino if necessary.

Next, this project did not evaluate battery consumption on the mobile phone during wireless transmission and rendering of the virtual scene. This may be important depending on the battery lives of different mobile devices, and on the length of time for which the virtual scenes need to be run.

Another area for future work is to test and correct stereo rendering using the wireless system. While the system effectively transmits orientation data wirelessly, the screen display parameters and other rendering parameters used in the code will have to be modified in order to correctly render stereo scenes on the display. This may involve resolving discrepancies between CSS pixels and the physical device pixels, and modifying other display specifications.

Finally, future investigation could also evaluate other wireless methods not tested here, such as NFC (near-field communication).

6. Acknowledgements

I would like to thank Professor Gordon Wetzstein, Keenan Molner, Hayato Ikoma, and Robert Konrad for their instruction and guidance throughout this course.

In particular, I would like to thank Hayato Ikoma and Keenan Molner for their suggestions and assistance throughout this project.

References

[1] EE267 Course Instructors, *Homework 5: IMU and Sensor Fusion.* Stanford, CA, 2017.

[2] Beaufort, Francois. (2015, Jul.). Interact with Bluetooth devices on the Web. Google. [Online]. Available: https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web

[3] EE267 Course Instructors, *Homework 7: Spatial Audio.* Stanford, CA, 2017.

[4] The Bluetooth Special Interest Group. Bluetooth Core Specification. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification

[5] Texas Instruments. (2015, Aug). UART to Bluetooth Low Energy (BLE) Bridge Design Guide. [Online]. Available: http://www.ti.com/lit/ug/tidu997a/tidu997a.pdf

[6] The Bluetooth Special Interest Group. Ten Important Differences between Bluetooth BR/EDR and BLE. [Online]. Available: https://blog.bluetooth.com/ten-important-differences-between-bluetooth-bredr-and-bluetooth-smart

[7] The Bluetooth Special Interest Group. (2009, Dec. 17) Press Release: SIG introduces Bluetooth Low Energy wireless technology, the next generation of Bluetooth wireless technology. [Online]. Available: https://www.bluetooth.com/news/pressreleases/2009/12/17/sig-introduces-bluetooth-low-energy-wireless-technologythe-next-generation-of-bluetooth-wireless-technology

[8] The Bluetooth Special Interest Group. What is Bluetooth? How It Works. [Online]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works

[9] Torvmark, Karl. (2014, Mar.). Three flavors of Bluetooth: Which one to choose? Texas Instruments. Dallas, TX. [Online]. Available: http://www.ti.com/lit/wp/swry007/swry007.pdf

[10] *Creating Bluetooth Low Energy Applications Using nRF51822,* nAN-36, Nordic Semiconductor, 2014.

[11] Adafruit. Introducing the Adafruit Bluefruit LE UART Friend. [Online]. Available: https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-uart-friend?view=all

[12] PJRC. SoftwareSerial Library. [Online]. Available: https://www.pjrc.com/teensy/td_libs_SoftwareSerial.html

[13] Mistry, Sandeep. (2015) *noble: A Node.js BLE (Bluetooth Low Energy) central module.* [Online]. Available: https://github.com/sandeepmistry/noble

[14] Igoe, Tom, et al. *Bluetooth LE Examples.* [Online]. Available: https://github.com/tigoe/BluetoothLE-Examples

[15] Guangzhou HC Information Technology Co., Ltd. HC-06 Datasheet. [Online]. Available: http://silabs.org.ua/bc4/hc06.pdf

[16] Sparkfun. Wi-Fi Module - ESP8266. [Online] Available: https://www.sparkfun.com/products/13678

[17] Intel. (2017, Apr. 17) Different Wi-Fi Protocols and Data Rates. [Online]. Available: https://www.intel.com/content/www/us/en/support/network-and-i-o/wireless-networking/000005725.html