Novel Method for Modular Integration of Tactile Input Devices Into Portable AR/VR Training Systems

Suzanne Borders, Jad Meouchy, Zachary Gaines, Damian Ugalde
BadVR, Inc.
Los Angeles, CA
suzanne@badvr.com, jad@badvr.com, zachg@badvr.com, damianu@badvr.com

ABSTRACT

The training and simulation industry has made tremendous progress with software recreation of physical training environments, but the integration of tactile input/output devices into portable solutions remains underdeveloped. While wired peripherals can easily connect to desktop computers, significantly fewer solutions exist for mobile applications. Connecting physical devices through Universal Serial Bus (USB) or Bluetooth Low Energy (BLE) to battery-powered AR/VR headsets is technically challenging, as these headsets run mobile operating systems such as Android, IOS, or Windows Holographic that do not have drivers or even ports for hardware peripherals. Furthermore, the communication protocols that these peripherals use are rarely, if ever, documented or accessible to app developers.

This paper presents a recently patented engineering technique for rapidly integrating USB & BLE devices into mobile platforms, a 'no-code' solution that automates peripheral interfacing for both software developers and non-technical users. First, open-source tools discover and analyze the target device's emitted messages. Next, a human-readable configuration file is generated with semantic protocol descriptions and measured values. Finally, this configuration is fed into a cross-platform virtual device driver that proxies raw wired/wireless packet interchange, including the oft-overlooked Human Interface Device (HID) protocol. Bidirectional support enables both user sensor input and tactile feedback output.

With this technology, mobile training applications will be able to integrate wearables such as smartwatches and fitness bands, "mountables" like commercial flight simulator controllers and smart home panels, and other Internet of Things (IoT) devices without the need for device driver code. Ultimately, compatibility with physical devices will empower the next generation of immersive training solutions to be more realistic, user-friendly, and–most importantly–versatile for a variety of mission needs.

ABOUT THE AUTHORS

Suzanne Borders is the CEO and co-founder of BadVR. She previously led product UX at Osurv, Remine, and CREXi, where she set a new standard for geospatial data visualization and successfully brought to market multiple analytics products. Suzanne is the recipient of the first Independent Creator's Program grant from Magic Leap and was named a rising star by IEEE in 2019. She is passionate about democratizing insight and thrives at the intersection of product design, immersive technology, and data. Suzanne studied psychology and design at the University of Missouri, Kansas City.

Jad Meouchy is the CTO and co-founder of BadVR. Originally from Virginia, Jad holds dual B.S. degrees in Computer Engineering and Psychology from Virginia Tech and attended the Thomas Jefferson High School for Science and Technology. Over his 15-year career, Jad founded and exited multiple startups, and engaged his healthy passion for user-centric innovation, software architecture, and AR/VR. As the Principal Investigator for BadVR, he led the development of proprietary algorithms and techniques for spatialization and visualization of data that resulted in awarded patents.

Zachary Gaines is a project manager at BadVR and is also an aerospace engineering student at Cal Poly Pomona. He has led a variety of new space related projects focusing on the implementation of Artificial Intelligence and Machine Learning systems into small spacecraft technology. He has also partnered with NASA and other prominent industry leaders for research and development and runs student educational programs at his university.

Damian Ugalde is a computer science and aerospace engineering major at Cal Poly Pomona and a software engineer at BadVR. He previously interned at the Air Force Research Lab and NASA, working on web development, machine learning, cloud infrastructure, systems engineering, and satellite engineering. He is passionate about aerospace and aviation and is working towards his private pilot certificate.

Novel Method for Modular Integration of Tactile Input Devices Into Portable AR/VR Training Systems

Suzanne Borders, Jad Meouchy, Zachary Gaines, Damian Ugalde
BadVR, Inc.
Los Angeles, CA
suzanne@badvr.com, jad@badvr.com, zachg@badvr.com, damianu@badvr.com

INTRODUCTION

VR/AR/MR (immersive) training is a rapidly expanding field with ever-increasing use cases and newfound applications. In recent years, the introduction of immersive training environments to disciplines, such as medical and military training, has shown great promise. The retention rate of VR training has been shown to be 75% higher compared to traditional methods like lectures or reading, which have rates of 5% and 10% respectively (Witte, 2020).

A key advantage of immersive training environments is the ability to simulate dangerous situations, such as natural disasters or battlefields, without ever having to introduce trainees to any real danger. These immersive solutions make the training highly effective, with a much lower risk profile and resource cost than real-world alternatives. The virtual and mixed-reality training sector has recently experienced a big push to adopt mobile, portable hardware devices, including mobile augmented reality (AR) solutions that enable trainees to move training into real-world environments, overlaid with interactive 3D displays. However, until now, the adoption of mobile hardware has been held back due to the difficulties of running heavy, complex training simulation programs on devices that lack the processing power offered by traditional headsets, which are tethered to powerful desktop computing systems.

The mobility of training equipment is highly important and provides a major strategic advantage, enabling onsite training to be rapidly deployed, and allows for the rapid transport and execution of training courses in all sorts of ‘real world’ scenarios. Containerizing training by making on-site training devices that are easily transported and shipped to the place of need quickly and without external dependencies offers huge on-the-job real time performance enhancements. Full immersion, inclusive of natural input and output devices allows for users to not only increase situational retention but also aids in the development of muscle memory. In a study from LMU Munich, researchers directly connected higher levels of immersion with higher levels of user satisfaction, increasing information retention (Mütterlein, 2018). The most effective immersive MR/VR training platforms utilize tactile IO input. Engaging multiple senses into the learning process enables students to get more value out of the training simulation and leads to increased performance and retention. The learning results of sample groups show that after training in multiple sensory environments, trainees report a 90% correctness in their task completion (Shams & Seitz, 2008). Building muscle memory is important for fast and accurate response times and aids in learning to perform delicate operations in high intensity and stressful environments.

A lack of tactile I/O devices in immersive environments hinders the efficacy of such solutions, since virtual headset-based environments cannot fully replicate touch sensations, lowering training effectiveness. Users remain self-aware and are prevented from hitting peak attention and activity levels, as there are limits to the training potential offered by visual-only activity. These limits have historically held back the development-and effectiveness-of more advanced training platforms. Additionally, many headsets struggle with delivering depth perception, which leads to an accommodation-convergence mismatch, where the brain receives divergent cues between the distance of a virtual object and the focal distance of the eye. This issue leads to eye strain and fatigue and is one of the many reasons that designing fully effective, immersive applications with purely virtual I/O is incredibly difficult, if not impossible.

Current technology offers additional limitations, including the inability to integrate real-world physical objects into immersive environments. The user is only able to use commercial controllers with a laser-pointer input to interact with their virtual space, which results in an unrealistic interface that instantly reduces the level of immersion, and lowers the value add of the immersive training. There is almost no haptic feedback, further causing users to suffer fatigue due to awkward arm positioning, which limits the amount of interactivity time. The best immersive training experiences enable trainees to use their hands or real tools to interact. This helps users develop the appropriate muscle memory and fine motor skills needed to perform the task properly, with simulated tools closing approximating the real-world tools they will use in the field. This familiarizes trainees with the correct workflow and process. It is currently challenging to accurately simulate pressing a button, or turning a knob, to the action of picking up a tool, within a fully virtual space. Most current methods leave users disconnected from the simulated environment or provide training that is not accurate to the real-world situations, tooling, and workflows encountered in the field.

As of now, immersive training is largely confined to custom developed, high resource solutions only available in educational centers. This obviously has many drawbacks, especially when it comes to delivering on-site and on-the job training. To ensure that trainees can learn in the best possible environments, immersive solutions need to be fully mobile, such as flyaway "training in a box" solutions. However, deploying immersive training solutions on mobile hardware comes with many challenges, especially in integrating I/O devices. Tactile I/O devices are mostly high-impact, high-performance, bulky, and not designed to be portable or to function on processing-constrained devices, or in hostile environments. When the proposed technology solution is utilized however, existing bulky tactile I/O devices can be replaced with low-cost, lightweight solutions by using commercially available off-the-shelf products or parts, such as smartwatches. This will lead to a jump in the readiness and reconfigurability of training kits, and a new level of environmental ruggedness.

Simultaneously, on-site training and training in on-the-job environments will become much easier and more accessible. By using the recently awarded patent (#11,355,094) on wireless virtual display controllers, physical devices can be represented in the virtual space by also using the technique outlined in this paper. The technology driving this solution makes team-based training possible with or without networking; combined with current training curriculums and programs, this innovative new solution aids in the creation of a robust network of decentralized education and training programs that offer an exciting new path forward for mobile training solutions.

Communication Between Peripherals and AR/VR Headsets

The core issue this paper addresses is the current difficulty with cross-platform device drivers and the difficulties of integrating physical IO devices to AR/VR/MR headsets and applications.

Drivers are a piece of software that allows the operating system to communicate with a peripheral. They are essential to the correct functioning of a peripheral device. However, operating systems are not standardized, and each one requires a different driver version. Therefore, companies creating these devices do not equally support all operating systems and must create specific variations for each. This presents a challenge for a rapidly expanding technological world, as there are many more platforms now than ever before, each requiring different drivers. For example, Microsoft HoloLens uses Windows ARM UWP, HTC Vive typically uses Windows x86, and Meta/Oculus and Magic Leap use variations of the Android operating systems. Each one is different and thus supports different peripherals, even when the underlying software application is easily compatible with all the operating systems.

Additionally, there is also a lack of standardization for drivers even within the same operating systems. Inconsistent quality across operating systems and versions of the same operating system exist as developers do not have the resources to create and test driver versions for each operating system and computer. Drivers needed for each situation are often proprietary and hard to obtain, as they are usually delivered via a CD (even in 2022) packaged with the device or downloadable through the original equipment manufacturer (OEM) websites that are poorly maintained. This presents a challenge for inventories of legacy equipment, often spanning decades, that may still be usable. The proprietary nature and scarcity of some drivers can force end-user developers to reverse-engineer drivers to allow a device to work on originally unsupported operating systems. This is an arduous process that requires specialized knowledge and is hindered by a lack of quality documentation. Even generic interfaces like WinUSB, a Windows generic driver for USB devices, are not often detected, supported, or documented, and require Windows utilities like Zadig to function. Drivers are designed with the intention of working with as many programs as possible, leading to unnecessary code abstraction, which further complicates hardware development.

A further challenge experienced when developing for AR/VR/MR headsets is the connection between the peripherals and the headset. The most widely used protocol is USB, which requires a physical connection. Most major headsets only have one communication port, limiting the number of devices that can be connected. This physical connection also hinders one of the technologies' main advantages, mobility. Adding a wired connection limits the movement area to the size of the cable and creates tripping hazards. A wired connection to the headset must also contend with how to supply power to the peripherals, as most peripherals are powered through their USB connection. However, USB ports are limited in capacity, so connecting an array of devices will hinder all the connected devices. The headset also suffers from thermal throttling as heat increases with computational demand. All mobile devices, including these headsets, will reduce CPU and GPU throughput to allow temperatures to lower. This decrease in performance will lower the ability to use the headset for training for prolonged amounts of time and can cause the training experience to suffer. In extreme cases, the extra power consumption can cause soft or hard fuses to activate, which can render the headset temporarily inoperable.

This paper presents a novel technology that can identify compatible devices and create a "no-code" configuration file to replace the need for OEM drivers on immersive (AR/VR/MR) devices. By automating the protocol analysis (usually the most intensive part of creating a driver) the developer would only have to interact with the configuration file. This file is a simple human-readable text file that outlines the capabilities of the peripheral device. A developer would later map an action performed on the peripheral device to a reaction by the training software, using virtual keystrokes or custom message channels. This configuration file can also be used across multiple devices, creating a pseudo-universal driver. By bringing driver design and configuration from a lower-level programming language to the higher-level human readable version, it allows for a wider application, accessible to far more people than the currently used solutions.

IMPLEMENTATION

The technique suggested is to create an intermediate software to handle the communication between a peripheral and headset, replacing ordinary OS-dependent drivers. This intermediate software talks with the device using the peripheral's communication protocol and passes a byte stream to a developer who can then create application-level bindings. The software generates a configuration file when the device is first connected that contains the information necessary for this byte stream to be interpreted, including suggestions for message blocks and state changes. This interpreting software can then be compiled for different operating systems, creating a pseudo-driver that eliminates the need for a developer to compile drivers for every peripheral the device will connect to. The configuration file thus functions as a "write once, run anywhere" extension.

The abstraction proposed also allows this technique to expand to different protocols. The two-layer approach allows for the system that reads and controls peripherals to be independent form the transmission of data to the applications. The system currently only supports USB, HID, and BLE protocols. With further development and refinement, other communication protocols that could be supported include Zigbee, Z-Wave, LoRa, UART, and I2C.

USB Protocol

This technique currently supports communication with the peripheral through the Universal Serial Bus (USB) protocol and the Human Interface Device (HID) class, which specifies devices that a person would use to operate a computer such as keyboards, mouse, and joysticks. The USB and HID protocols have defined communication between the peripheral and the device that it plugs into. This can be advantageous to driver development, as the standardized shared information between the devices can be used to create assumptions of the device's operation. These assumptions are further refined by use of the "no-code" configuration file. When a USB device is plugged in, the peripheral device sends a USB descriptor table to the host machine that contains information about the device's features and the organization of the device's data. This information can be parsed to create generalized assumptions of the device's functionality.

The USB protocol defines different descriptors that can contain multiple types of sub-descriptors within them. These levels are shown in Figure 1. The main level is the device descriptor, which contain critical information needed for the device's correct functioning. This includes USB version and device manufacturer. Underneath the device descriptor, it also contains one or many configurator descriptors, which specify how the device should be powered for different operating modes. Next is the interface descriptor. These bundle different device functions into one device feature for ease of communication and organization with the host machine. Each represents one device function. Finally, the lowest level of descriptors is the endpoint descriptor. These specify how data is transferred between the peripheral and the host machine. Endpoints define the packet size, transfer direction, and interval.

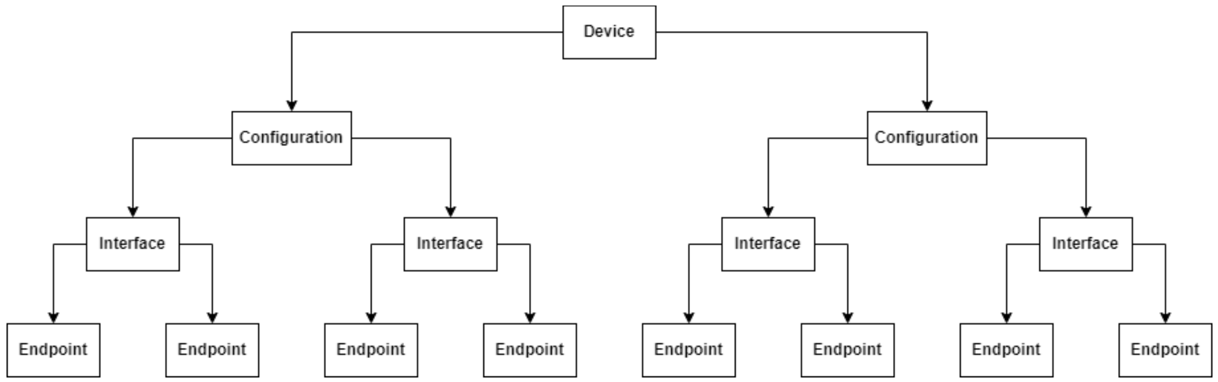
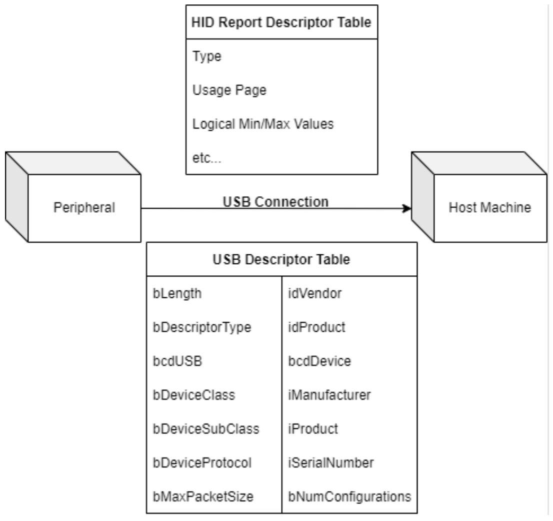


Figure 1. Hierarchy of USB Descriptors

As most devices that a trainee would interact with are HID, it is advantageous to look at the information shared by the HID protocols. Besides the USB descriptor tables, HID also shares a report descriptor table. Of usefulness to this application, the type, usage page, and logical minimum and maximum values are shared with the host machine. The type can be an input, output, or feature. An input type is data that the peripheral can send to the host machine (for example, mouse button movement or keyboard typing). An output type is data that the host machine can send to the peripheral (for example, turning on an LED on the device). A feature is data that can be shared in both directions (for example, force-feedback on a simulation steering wheel). For now, this technique is mostly focused on the data from the peripheral to the headset.

The usage page uses predefined tables from the USB protocol to categorize what the peripheral is designed to perform. Such categories include simulation controls like flight, automobile, spaceship, and tank simulation devices. Finally, the logical minimum and maximum values help define what the boundaries of the data are and help with generalizing the controls. These tables are illustrated in Figure 2. Note that this technique reduces the amount of work required by the driver developer by creating assumptions based on the USB protocol, but it will still require some input from a technical user before it can reach the end-user. The goal of this technique is not to eliminate the need to observe and deduce a device's functioning with trial-and-error, but rather to create a simpler way for the device drivers to be used by different host machines and operating systems.



HID Report Descriptor Table
Type
Usage Page
Logical Min/Max Values
etc...

USB Descriptor Table	
bLength	idVendor
bDescriptorType	idProduct
bcdUSB	bcdDevice
bDeviceClass	iManufacturer
bDeviceSubClass	iProduct
bDeviceProtocol	iSerialNumber
bMaxPacketSize	bNumConfigurations

Figure 2. USB and HID Tables Shared from Peripheral to Host Machine

Bluetooth Low Energy Protocol

Devices using Bluetooth and Bluetooth Low Energy (BLE) also have a similar initial information sharing between the peripheral and host machine, as shown in Figure 3. BLE uses an attribute table to define how other devices communicate with it. There are three different types of attributes: services, characteristics, and descriptors. Each has their own Universally Unique Identifier (UUID). Like USB, each service contains characteristics which in turn contain descriptors. A service represents a feature that the device can perform. Characteristics are individual items of data, states for that device, or configuration data. Characteristics are how the peripheral communicates with the host machine. It can perform three actions: READ, WRITE, and NOTIFY. From these combinations of operations, all Bluetooth devices can function in the applications we use every day. A host machine can subscribe to the characteristics provided by the peripheral to be notified of changes (for example, when a button is pressed), read data (for example, take a photo from a camera), or write to the peripheral (for example, draw a picture on the built-in screen). A device must communicate in at least one of these actions, but it will not necessarily have all three.

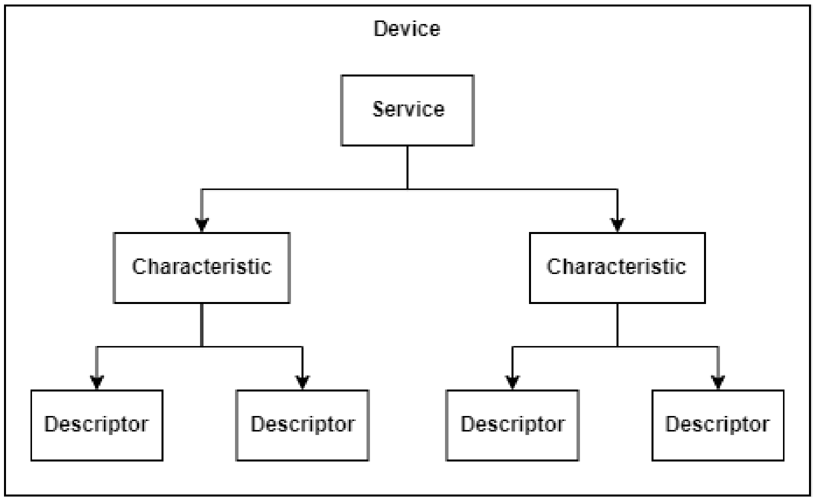


Figure 3. Bluetooth Services, Characteristics, and Descriptors

The data shared by the USB protocol standard when a device is first mounted to the host machine or by the BLE device when it is first connected gives context on what the device is and what it can do, but the largest amount of data on the device's operation is gathered by looking at the data shared by the peripheral as the user interacts with it. By sniffing the USB packets shared with the host machine, it can be determined what changes in the peripheral cause what changes in the data. USB packet capture can be done using commercially available applications such as Wireshark. This same package sniffing technique can also be applied for BLE devices, by inspecting their services and characteristics to see what data is shared with the host machine. Presented below is a simple example to illustrate the how packet sniffing allows for reverse-engineering device drivers. Figure 4 is an example of a commercially available flight throttle for flight simulators. From observation, there are three sliders and three buttons that can be 'neutral,' 'up,' or 'down.'

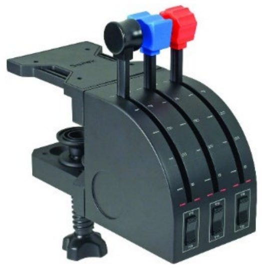


Figure 4. Saitek Flight Throttle Quadrant

When the device is first connected with random slider positions and the USB packets are examined, it reveals that the data stream is packed in 4 bytes, each containing binary numbers. At first, the data might look like Figure 5.

Byte 1	Byte 2	Byte 3	Byte 4
0 1 1 1 0 1 0 0	1 1 0 0 1 0 1 1	0 1 1 0 0 0 0 0	0 0 0 0 0 0 0 0

Figure 5. Data stream when first connected

At first glance, this information is not insightful. However, the position of the sliders and the buttons can be changed to gather more information. The sliders are moved to the top of the device as shown in Figure 1 and this results in a reading of Figure 6.

Byte 1	Byte 2	Byte 3	Byte 4
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Figure 6. Data stream when all sliders are at the top

The first, second, and third bytes changed to values of zero. Therefore, it can be inferred that the slider positions are related to the first three bytes. Then each slider can be manipulated individually. Leaving sliders two and three in the same position and moving slider one to the lowest position available, Figure 7 is the next data reading.

Byte 1	Byte 2	Byte 3	Byte 4
1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Figure 7. Data stream when all sliders are at the bottom

As the second and third bytes now stay constant while the first changed, it shows that byte one dictates the value of the first slider. Using similar experimentation with sliders two and three, bytes two and three control those sliders respectively. Also, it is also inferred that the bytes are zero when the slider is at the highest position and are all one when the slider is at the lowest position. Any values in between mean the slider is somewhere in the middle.

Moving on to the buttons, the experimentation and data gathering is summarized in Table 1. Only the fourth byte is shown as the first three are constant. The first and second bits of the last byte are always 0, no matter the button or slider configuration, so it can be assumed that they are not used.

Table 1. Experimentation and Data Gathering of Button Controls

	Change in peripheral	Byte 4	Conclusions
A)	Button 1 set to ‘up’	0 0 0 0 0 0 0 1	The last bit shows a change in the button
B)	Button 1 set to ‘down’	0 0 0 0 0 0 1 0	Bit 7 also changed with the button. Bit 7 shows that the button is ‘down’ while bit 8 shows that the button is ‘up’.
C)	Button 2 set to ‘up’ Button 1 back to ‘neutral’	0 0 0 0 0 1 0 0	Bit 6 changed with button 2.
D)	Button 2 set to ‘down’	0 0 0 0 1 0 0 0	Bit 5 also changed with button 2, following the pattern of button 1. Therefore, bit 5 shows button 2 as ‘down’ and bit 6 as ‘up’.
E)	Button 1 set to ‘down’ Button 2 set to ‘up’	0 0 0 0 0 1 1 0	Bits 6 and 7 are on. This validates the previous conclusions that button 1 is ‘down’ and button 2 is ‘up’.
Button 3 follows a similar pattern to those above and is omitted here for brevity.			

Through experimentation of pressing buttons and moving sliders, then observing how the data stream changes, the software was able to determine patterns of activity that indicate what each byte and the bits inside represent. A developer, using this information, can then indicate what each detected action should perform in their program, such as simulating a key press, using a shortcut, or triggering an event listener.

A “no-code” configuration file still requires some input from a developer to map the actions from the peripheral to the required effect on the training program. This technique aims to reduce the amount of time required to develop a pseudo-driver by making the device discovery process simpler and by creating a universal configuration that can be used for multiple devices and platforms.

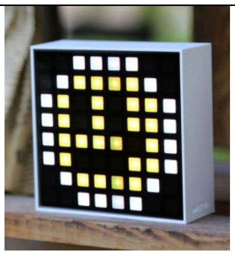
Because the configuration file is a human-readable text format, source control can be used to track edits over time and distribute revisions through a collaborative process. Configurations can be easily embedded into application source code for open redistribution, where only an interpreter is needed rather than a compiler. This also allows the pseudo-driver to be deployed at runtime for portability and ease of use, such as downloading configurations on-demand when new devices or peripherals are detected. This also allows for development to be quicker as less device optimization is required for each host machine.

RESULTS

Analyzing the performance of this novel approach and comparing it to more traditional techniques is crucial for justifying the integration of this technology into common functions. This communication protocol was evaluated with an assortment of commercially available off-the-shelf products and showed demonstrable compatibility with the list of devices shown in Table 2. Tests were conducted on both training and non-training related peripherals to demonstrate the breadth of the solution, and to open training curriculum creators to a larger realm of potential devices to incorporate into their courses.

Training needs can occasionally be unknown or require creative improvisation in a short window of time. By supporting many types of consumer- and commercial-class devices, composite peripherals can potentially be assembled by trainers in the field to adapt to a variety of mission needs.

Table 2. Commercially Available Devices That Are Compatible with the Technique

Saitek Flight Controller	Korg Nano Series MIDI Controllers	Powermate Smart Knob	Tangent Video Control Surface
Davinici Speed Editor	Samsung Galaxy Watch 4 Smartwatch	Samsung Fitness Band	Weather Anemometers
			
Vion Smart Multimeter	Nuimo Smart Button	Dotti Smart Display	Satechi S2 Smart Remote

The list in Table 2 shows a diverse set of devices that each have unique use cases and in combination can create diverse training environments. The key element for designing training environments is understanding how these devices function.

Table 3. Device Capabilities

Device	Protocol	Visual Feedback	Fine Precision	Biometrics	Original Intended Use
Saitek Flight Controller	USB	X	X	-	Video Games
Korg Nano MIDI controllers	USB, BLE	X	X	-	Music
Powermate Smart Knob	BLE	-	X	-	Productivity
Tangent Video Control Surface	USB, HID	-	X	-	Multimedia
Davinci Speed Editor	USB	X	X	-	Multimedia
Samsung Galaxy 4 Smartwatch	BLE	X	X	X	Productivity
Samsung Fitness Band	BLE	-	-	X	Health
Weather Anemometers	BLE	-	-	-	Weather
Vion Smart Multimeter	BLE	X	-	-	DIY
Nuimo Smart Button	BLE	X	-	-	Home Automation
Dotti Smart Display	BLE	X	-	-	Home Automation
Satechi S2 Smart Remote	HID	-	X	-	Office
MixFader	BLE	-	X	-	Music

In Table 3, each device is laid out with what communication protocol it uses as well as what type of feedback and interaction can be expected.

Through implementation with mobile training systems, users can get physical feedback from their actions, enabling large leaps in response from the system and greater levels of immersion. Feedback allows training designers to make systems that are more in tune with real situations and actual movements that trainees will encounter in the real world.

Consumer devices not originally meant for virtual reality applications can already be used in the virtual realm. For instance, hobby-level flight simulation hardware can be used to for flight training for a fraction of the cost of expensive simulators by adapting them to immersive hardware. Another example is firefighting captains that need to reliably measure the health of their firefighters during a wildfire. Using biometric sensors and trackers, it is possible to track and understand an individual firefighter's health in real-time. By fitting these sensors in the uniform of the fire crew, captains can make better informed decisions about the safety of their operations. Using simple commercial tools makes it cheaper to operate while the technique proposed allows for those commercial products to be easily integrated into immersive headsets.

FUTURE WORK

The state of the current work makes it suitable for adoption and implementation now but there are still subfields of this technique require further improvement. One of the first things to be addressed is solving the issue of latency between the physical movement to action inside the mobile application. Current headset technology is set up in a way where when moving hands or controllers the cameras on the device must be processed and then the system can adjust the virtual position, but this makes for an inherent disconnect between the timing of a user's movement and the response of the system to this movement. This challenge needs to be addressed by creating high performance drivers that minimize latency in applications where millisecond-level reaction time is critical. There are also latencies from wireless communication techniques in general as well as minor delays from the hardware such as the event listeners in transmission and receiving. All these small latencies compound into measurable lag as components and transformations are added to the pipeline, which could directly affect user engagement and comfort. The proposed technique is swift in isolation but does not leave much margin of error for application developers, which is important for adoption.

Force feedback is used in many training and simulation peripherals, such as steering wheels and joysticks for relaying physical vibration from the application to the user, while also limiting its strength to prevent injury to the trainee. Where existing devices were primarily used for vehicular training systems, future force feedback style peripherals include wearable vests and suits that simulate environmental and medical conditions or impairments. A further area of research is expanding the virtual driver solution to be compatible with the next generation of force feedback protocols. Also, force feedback devices and haptics can potentially rely on closed loop updates at high refresh rates (100Hz+) that will require special drivers or techniques to accurately sample and control. Future work needs to support this special classification of high-frequency hardware control, which may also be present and prone to interference from industrial equipment.

Additionally, as this technology expands, and adoption becomes more prevalent, there are large benefits from supporting a more diverse group of communication protocols. If the technique can encompass an even wider scope of communication techniques, this will allow for a large network of new devices to become integrated into training environments. As the device list expands there may be a desire to establish a public repository of compatible devices and examples of their respective configuration files. This will cause a reduction in the technical knowledge needed to implement the innovation and makes the addition of tactile I/O peripherals more trivial for the average developer, resulting in greater compatibility and ease-of-use for the end user.

CONCLUSIONS

As the field of mobile AR/VR/MR training continues to emerge and develop, one of the key technologies will be the inclusion of tactile I/O into simulated environments. Without this, users lack the ability to incorporate physical sensations, limiting the depth of their immersive experience. This also has profound impacts on the efficacy of immersive training (and other) solutions. Device drivers currently suffer from issues in operating system combability, with each device needing its own custom driver, presenting a large bottleneck in the integration of tactile devices into virtual environments.

With the novel "no-code" automation technique proposed in this document, developers and creators will no longer need to write OS specific drivers to support new I/O peripherals. Everyone will be able to easily incorporate a large variety of commonly used devices into virtual environments and users will be able to consistently add or subtract devices with little hassle, opening a wide new world of training tools that instructors and course designers can use to optimize trainee performance. This simple solution will allow the fast-moving minds of educators to quickly adapt their coursework and training as situational needs change and will reduce the cost of both implementing and updating these solutions. Lastly, this solution will encourage the use of low-cost, tether-free headsets and hardware, and will encourage everyone to repurpose existing peripheral devices as new interactive elements of immersive training environments, benefitting instructors and trainees globally.

REFERENCES

Atta, M. T. (2021). *The impact of AR/VR on spatial memory performance of learners: A Review.* Retrieved from https://ieeexplore.ieee.org/document/9537083

Bates, B. M., Dezmelyk, R., Ingman, R., & Lieb, R. (2004, 10 28). *Universal Serial Bus (USB), HID Usage Tables.* Retrieved from USB: https://usb.org/sites/default/files/documents/hut1_12v2.pdf

Bergman, M., Peurach, T., & Schmidt, T. (2001, May 27). *USB.* Retrieved from Universal Serial Bus (USB) Device Class Definition for Human Interface Devices (HID): https://www.usb.org/sites/default/files/hid1_11.pdf

Maidenbaum, S., Patel, A., Garlin, I., & Jacobs, J. (2019, January 1). Studying spatial memory in augmented and virtual reality. *bioRxiv*, 777946. Retrieved from https://www.biorxiv.org/content/10.1101/777946v1.full

Microchip. (2021). *USB Descriptors*. Retrieved from Microchip Developer Help: https://www.microchipdeveloper.com/usb:descriptor

Murphy, R. (2018). *USB hid mouse with psoc 3/PSOC 5LP.* Retrieved from Infineon: https://www.infineon.com/dgdl/Infineon-CE95394_USB_HID_Mouse_with_PSoC_3_PSoC_5LP-Code+Example-v03_00-EN.pdf?fileId=8ac78c8c7cdc391c017d0d6cef3874fd

Mütterlein, J. (2018). The Three Pillars of Virtual Reality? Investigating the Roles of Immersion, Presence, and Interactivity. *Hawaii International Conference on System Sciences* (pp. 1407-1415). LMU Munich: HICSS. Retrieved from ScholarSpace: http://hdl.handle.net/10125/50061

Peacock, C. (2018, April 12). *USB Descriptors*. Retrieved from Beyond Logic: https://www.beyondlogic.org/usbnutshell/usb5.shtml

Shams, L., & Seitz, A. R. (2008). Benefits of multisensory learning. *Trends in Cognitive Sciences, Volume 12, Issue 11*, 411-417.

Witte, C. (2020, June 8). *FrontCore*. Retrieved from Is VR Training More Efficient than Other Learning Methods?: https://frontcore.com/blog/is-vr-training-more-efficient-than-other-learning-methods/

Zaman, F., Roeca, G., Lupascu, I., Gardony, D. A., Natick, M. A., Rife, D. J., & Intriligator, D. J. (2021). Analyzing Spatial Memory in a Virtual Reality-Based Subterranean Scenario: Implication for Military Augmented Reality Systems. *Human Factors and Ergonomics Society Annual Meeting* (p. 65(1)). HFES.