Enhancing Virtual Reality Systems with Smart Wearable Devices

Salah Eddin Alshaal*, Stylianos Michael*, Andreas Pamboris†,
Herodotos Herodotou‡, George Samaras† and Panayiotis Andreou*,
*Department of Computing, University of Central Lancashire, Cyprus/ InSPIRE Center, Cyprus
Email: {sealshaal,smichael,pgandreou}@uclan.ac.uk
†Department of Computer Science, University of Cyprus, Cyprus, Email: {pamboris.andreas,cssamara}@cs.ucy.ac.cy
‡Department of Electrical Engineering, Computer Engineering and Informatics, Cyprus University of Technology, Cyprus
Email: herodotos.herodotou@cut.ac.cy

***Abstract*—The proliferation of wearable and smartphone devices with embedded sensors has enabled researchers and engineers to study and understand user behavior at an extremely high fidelity, particularly for use in industries such as entertainment, health, and retail. However, identified user patterns are yet to be integrated into modern systems with immersive capabilities, such as VR systems, which still remain constrained by limited application interaction models exposed to developers. In this paper, we present SmartVR, a platform that allows developers to seamlessly incorporate user behavior into VR apps. We present the high-level architecture of SmartVR, and show how it facilitates communication, data acquisition, and context recognition between smart wearable devices and *mediator* systems (e.g., smartphones, tablets, PCs). We demonstrate SmartVR in the context of a VR app for retail stores to show how it can be used to substitute the requirement of cumbersome input devices (e.g., mouse, keyboard) with more natural means of user-app interaction (e.g., user gestures such as swiping and tapping) to improve user experience.**

***Keywords*-virtual reality systems, smart wearable devices, wearable app framework**

I. Introduction

Rapid advances in mobile and smart wearable technologies have recently led researchers and engineers to explore user behavior in the context of complex interdisciplinary areas (e.g., artificial intelligence, behavioral sciences). Despite having access to an abundance of devices and valuable data, as well as the know-how to semantically exploit it, digital immersive environments such as Virtual Reality (VR) systems are falling behind the high standards set by users in terms of usability and truly enticing user experience.

The goal of current VR systems is primarily to immerse users in virtual environments by exposing a 3D visual setting. However, this is achieved at the expense of usability, since users are inherently obstructed from viewing and interacting with physical objects. This poses severe limitations to state-of-the-art VR devices, which typically do not provide a well integrated input method to complement the high-standard visual experience they offer. For example, by losing line of sight of peripheral devices such as the keyboard and mouse, interacting with such devices is impractical to say the least. As an alternative, VR systems allow users to aim with their head (ray-casting approach), which is more intuitive. Nevertheless, using head movements limits precision, adversely affects application responsiveness, and causes discomfort to users after some time.

While the aforementioned limitations have important ramifications for interaction design, typical VR systems also miss out on the opportunity to leverage valuable information regarding runtime user behavior. They merely exploit sensor readings in order to extrapolate user orientation, acceleration, and position for controlling the virtual camera. However, data such as the user's heart rate, with potentially significant semantic value to VR apps (e.g., to infer the emotional and physical status of users at runtime), are completely overlooked. Partly, this is because VR systems are built around single standalone devices, with limitations on the amount of embedded sensors they can include.

However, wearable technology is currently blooming and all the more wearable devices with advanced capabilities (e.g., smart watches, rings, and clothes) are released from production. The main force driving this development lies in the recent advancements in embedded sensor systems and microcontrollers, which allows wearable devices to include an increasing number of sensors that are packaged in small form-factor devices. We see this as an opportunity to address the usability concerns of current VR systems by leveraging wearable technology as a means of interacting with a system in a more natural manner and inferring additional runtime information about users to enrich VR app functionality.

To this end, this paper presents a user interaction system, coined SmartVR, intended to increase the usability and functionality of VR apps using multiple wearable devices. The main contributions of SmartVR are: (1) it supports intuitive and user-friendly interaction models with VR systems by providing a framework for specifying such models with minimal developer effort; and (2) it enables the seamless integration of wearable technology and virtual reality systems by governing the interactions between the different components involved. It exposes an API that can be easily ported to a wider range of smart wearable devices on different platforms, e.g., Android and iOS.

Our demo showcases how wearables can be leveraged within the VR realm by periodically collecting sensor data

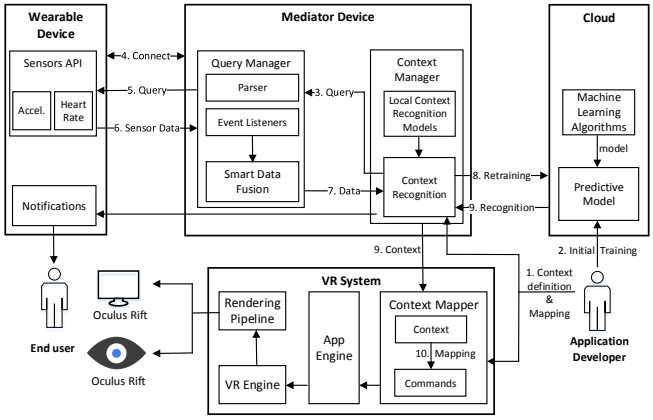

Figure 1. The SmartVR architecture

to infer user hand gestures, which are then mapped to program-defined actions in the VR context. In particular, SmartVR supports VR apps running on desktop machines. We illustrate SmartVR using a VR retail app where users can roam within a shop, browse clothing items, and complete purchases. The demoed setting uses the Oculus Rift headset and Microsoft Band 2, a hand wearable equipped with numerous sensors. The program will accept hand gestures (e.g., swipe left, right, up, and down) as a means of interaction within the program. As part of the demo, a user will navigate through the virtual shop, select an article of clothing, and manipulate its color and size. This will be realized using only the aforementioned devices and no peripherals.

In Section II, we describe the system architecture of the SmartVR platform. Section III discusses related work and Section IV concludes with an overview of the demo.

II. SmartVR platform

In this section, we summarize the main architectural components of the SmartVR platform, shown in Figure 1.

A. System Architecture

SmartVR enables developers to enhance and extend their VR apps by integrating context information such as gesture and behavior recognition. This information is acquired by interpreting sensor data generated by a wearable device, typically mediated by another device (e.g., smartphone or computer), and then mapping this data to specific commands of an app running on a VR system, such as rotating the camera to the left, selecting an item, etc.

The process of context integration starts during (or after) application development. During Step 1, the developer defines a list of context recognition events (e.g., move hand to the left, tap hand) that, when identified by the platform, will trigger the execution of an application command (e.g., rotate camera to the left, select item). These context recognition events are stored in the *Context Manager* of a *Mediator Device* (e.g., smartphone) and are then mapped to application commands in the *Context Mapper*. Developers can use a number of predefined local context recognition models to identify events or alternatively they can define their own by training new predictive models (Step 2).

During application execution, the Context Manager queries the wearable device through the *Query Manager* (Step 3) for sensor data (e.g., accelerometer, heart rate) according to context recognition requirements. The Query Manager first connects to the wearable device (Step 4) and then transmits a number of low level queries to the local sensors through the *Sensors API* (Step 5). The data sent from the wearable device (Step 6) is first received by the *Smart Data Fusion* component, which maps it to a unified data model and then propagates it to the *Context Recognition* component in the desired format (Step 7). The Context Recognition component can use the *Local Context Recognition Models* to infer context. If this is not possible, the associated data may be transmitted to the *Cloud* component (Step 8) for model retraining. Finally, the recognized context events are transmitted to the VR system (Step 9), which are then mapped to specific commands of the app (Step 10).

B. Query Manager

The Query Manager is responsible for establishing communication with the wearable device and for disseminating queries using the device's native Sensors API. The low level query format used by the Query Manager is in the form: $\{(t, f, e)\}$, where t is the sensor type (e.g., accelerometer, gyroscope), f is the frequency of data acquisition and e is an event handler object that will be binded to events raised by the sensor when data is generated. As the wearable device executes the query, it propagates its data response to the Smart Data Fusion component of the Query Manager in a proprietary to the device format. Finally, the Smart Data Fusion component parses the received data to transform it according to a *unified data model* so that it can be easily utilized by other components and applications.

C. Context Manager

The Context Manager handles requests to communicate with wearable devices. In particular, it is responsible for: i) sending queries to the Query Manager; ii) using the query response to (re)train machine learning models; and iii) communicating context information to the Context Mapper.

The Context Manager communicates natively with the Query Manager by querying for unified sensor data. We currently support two different types of queries: i) *Basic queries* that return the unprocessed unified data from one or more sensors; and ii) *Event-based queries* that allow for the correlation of data to programmable events. Event-based queries can also accept a lifetime parameter that defines the duration for which the query will be executed. The queries can be provided directly by the developer or can be communicated through the VR system.

Context recognition based on unified sensor data can be done locally (for simple recognition tasks) or on the cloud. For the latter, the extracted features are sent as JSON objects through an HTTP request to the Cloud component, which encapsulates its response in a JSON object and sends it back to the Context Manager. An important use for the queries' data is for retraining models defined by developers to increase the accuracy of context recognition. This is done by extracting features (e.g., min, max, avg for accelerometer) relevant to training the model and transmitting them to the machine learning model for further training.

The Context Manager supports two modes of interaction: i) *context-based* mode, which recognizes events and returns flags to the requester; and ii) *raw* mode, which enables the continuous transmission of processed data and is typically used for free interactions such as binding camera rotation to processed gyroscope values. The latter includes minimal data processing, thus offering better real-time experience. Nevertheless, it still contains context recognizers for dynamic and intuitive alternating between modes (without affecting user experience by relying on typical peripheral devices like keyboards); for instance, the developer can specify the gesture of knocking to be used for switching between modes.

Finally, the Context Manager can also be used to activate feedback modules such as haptic feedback or sound for better user experience.

D. Cloud Component

The Cloud component assists the mediator device in the execution of computationally-intensive tasks. For example, it can be leveraged for the execution of machine learning algorithms that are used to train models for context recognition. These models would be hosted on the cloud so that they can be reached for retraining and predicting values through service requests. The use of this component is optional and the same functionality can be achieved locally on the mediator device, albeit with degraded performance.

E. Context Mapper

Depending on the app and based on its program state, specific actions can be interpreted differently at runtime. The Context Mapper is responsible for providing meaning to the recognized actions. For example, swiping left in one context definition might translate to selecting a previous item, whereas in another it might translate to canceling the current event. The App Engine is responsible for performing the actions identified in the Context Mapper. Finally, the VR Engine translates the app state into a stereoscopic display that provides an immersing and interactive experience.

F. Communication Layer

SmartVR includes a communication layer for internal communication to overcome the barrier imposed by different programming languages, e.g., Java, C#, or Swift, for apps that execute on the mediator device. This promotes system modularity and overcomes challenges associated with the lack of a direct communication channel between the recognition framework and the graphical project language in game engines like Unity or Unreal. Using SmartVR, developers are able to write programs using their language of preference by utilizing the supported libraries.

We chose SignalR as the basis for our communication layer due to its assurances regarding real-time communication between components. SignalR is open source and can be used across all major wearable platform languages such as C#, Java, Swift, and JavaScript. The main objective of the SignalR server is to receive messages from the VR app, which requests to activate the wearable connection or to execute a query, before redirecting the messages to the Context Manager. The communication layer accepts two formats of communication that correspond to the two aforementioned modes of interaction in the Context Manager.

III. Related Work

Recent work explores approaches for enhancing the human-computer interaction within VR environments, which can be grouped into two main categories. The first one involves the use of motion sensors, cameras, or depth sensors for capturing hand gestures and translating them into actions in VR apps. Leap Motion is a motion-sensor device that can be mounted on a VR headset such as Oculus Rift to track hand and finger movements. It has been successfully used to control avatar movements in VR apps [1] and first-person-shouter games [2]. Cameras [3] and depth sensors [4] have also been proposed for performing complex hand gesture recognition without the use of wearable devices.

Nevertheless, Leap Motion and head-mounted cameras suffer from a limited field of view (typically $135\mathrm{deg}$). As a result, hand gestures may not be detected when the headset is facing away from the hands. In addition, gesture recognition using cameras and sensors is too computationally expensive for smooth real-time usage. Instead, through SmartVR, the use of wearable devices such as smart wristbands and rings avoids such limitations and at the same time supports similar fine-grained hand and finger recognition.

The second category includes specialized (and often expensive) wearable devices, including gloves, smart armbands, and smart vests, that pair with specific VR headsets for improving user control and immersion [5], [6], [7]. In contrast to such approaches, the SmartVR design promotes the interoperability of VR technologies with common wearable devices, which many users may already own (e.g., smart watches and smart wrist bands).

IV. SmartVR Demonstration

A. Demonstration Setup

Equipment: For the demo, we will use the following devices: (i) a Microsoft Band 2 wristband; (ii) a VR worksta-

Figure 2. The SmartVR prototype: (left) Select mode, (right) Navigation mode

tion; (iii) a high definition monitor; (iv) a Bluetooth 4.0 LE transceiver; and (v) an Oculus Rift Development Kit 2 (DK2) headset. The headset and the monitor will be attached to the VR workstation to allow visitors to view the visual output of the application both in 2D and 3D. The Bluetooth transceiver is used to allow communication between the VR workstation and the wearable device. We chose to use the Microsoft Band 2 wearable device as it features multiple embedded sensors such as 3-axis accelerometer, 3-axis gyroscope, UV and heart rate sensors, which we leverage to recognize a variety of gestures. In addition, Microsoft Band 2 operates a proprietary operating system with built-in support for interoperability with different programming languages such as C#, Java, or Swift.

Prototype Implementation: Our platform is implemented using the Band SDK for the Universal Windows Platform in C#. The gesture recognition model is implemented using SVM algorithms with the help of scikit-learn library[1] in Python. On the cloud side, we used AzureML[2] to handle data parsing to train our Python model, as well as to automatically deploy a request service for the predictive model. On the user end, we employed Unity[3] to develop the application and to support Oculus Rift.The game received recognized context through a SignalR-based[4] communication platform.

B. Use Case Scenario

We will demonstrate a prototype implementation of SmartVR by showing how, based on wearable signals, recognizing and using hand gestures can significantly improve user experience in the context of an online virtual clothing store. In particular, during the demo, conference attendees will be able to wear the Oculus Rift headset and enter a virtual store. Inside the store, a user can perform a number of actions: navigate through the store; browse different clothing items; select and view a specific item; and add/remove items to/from the virtual shopping cart.

The aforementioned actions will be supported through two distinct gesture modes, i.e., the *navigation* and the *selection* mode. Using the former, a user will be able to explore her spatial surrounding by swiping her hand accordingly in different directions to move to the left and right, or back and forth. Using the *selection* mode, the camera will remain static, allowing the user to select an item of interest using a cursor that moves according to user swipes. When an item is selected, the user will be provided with additional menus for customizing it according to size and color preferences. Finally, the user can gesture a forward press to add the selected item into the corresponding shopping cart. To switch between the available modes, the user will need to perform a special gesture, named *knock-knock*. Feedback regarding user gestures will be presented in the form of overlay messages at the bottom right corner of the screen.

[1]scikit-learn, http://scikit-learn.org/
[2]Azure Machine Learning Studio, https://studio.azureml.net/
[3]Unity Game Engine, http://unity3d.com/
[4]SignalR, http://signalr.net/

ACKNOWLEDGMENTS

This work was partially supported by the European Commission under project MiraculousLife (FP7-ICT-2013-10).

REFERENCES

[1] C. Khundam, "First Person Movement Control with Palm Normal and Hand Gesture Interaction in Virtual Reality," in *Proc. of the 12th Intl. Conf. on Computer Science and Software Engineering (JCSSE)*. IEEE, 2015, pp. 325–330.

[2] J. Chastine, N. Kosoris, and J. Skelton, "A Study of Gesture-based First Person Control," in *Proc. of the 18th Intl. Conf. on Computer Games (CGAMES)*. IEEE, 2013, pp. 79–86.

[3] M. Van den Bergh and L. Van Gool, "Combining RGB and ToF Cameras for Real-time 3D Hand Gesture Interaction," in *Proc. of the IEEE Workshop on Applications of Computer Vision (WACV)*. IEEE, 2011, pp. 66–72.

[4] Z. Ren, J. Yuan, and Z. Zhang, "Robust Hand Gesture Recognition Based on Finger-earth Mover's Distance with a Commodity Depth Camera," in *Proc. of the 19th ACM Intl. Conf. on Multimedia*. ACM, 2011, pp. 1093–1096.

[5] C.-M. Wu, C.-W. Hsu, and S. Smith, "A Virtual Reality Keyboard with Realistic Key Click Haptic Feedback," in *HCI International 2015*, ser. Comm. in Computer and Information Science, C. Stephanidis, Ed. Springer International Publishing, 2015, vol. 528, pp. 232–237.

[6] S. Hibbert, "Combining the Virtual and Physical Interaction Environment," in *Serious Games*. Springer, 2015, pp. 191–194.

[7] E. Vogiatzaki and A. Krukowski, "Virtual Reality Gaming with Immersive User Interfaces," in *Modern Stroke Rehabilitation through e-Health-based Entertainment*. Springer, 2016, pp. 195–214.