Oregon State University
College of Engineering

CS Capstone Executive Summary

April 20, 2020

Design and implementation of a framework for bio-informed 3D user interaction

Prepared for

Oregon State University College of Engineering

Raffaele de Amicis _____ Signature _____ Date

Prepared by

Group 51
BioMR

Ley Aldinger _____ Signature _____ Date

Ayush Choudhury _____ Signature _____ Date

Kyle Hiebel _____ Signature _____ Date

Zunyue Qiu _____ Signature _____ Date

Abstract

VR is gaining new applications, especially in manufacturing. With people spending a long time in VR, our customer wants to study how a VR environment can affect a user's time perception. To do this, we were tasked with designing a system architecture which can modify a VR scene based on biometric sensor data. The biometric data includes eye tracking, skin conductance, and brain activity. It is thought that data from these sensors quantitatively correlates with time perception. Creating this system architecture will allow our client to design a study where they can manipulate parameters in the VR scene, and measure the resulting time perception.

CONTENTS

1 Overview

Over the last few years, virtual reality (VR) and other immersive technologies have rapidly grown in popularity. A new usage for this technology is emerging in manufacturing shops throughout America. Smart machines, with enhanced visualization capabilities, allow for remote operation through a virtual interface. Workers using these new interfaces can expect to be in a virtual environment for up to 8 hours a day. Due to this large amount of time spent in VR, our client would like to study the effects of certain environmental conditions on VR users, and use this information to improve VR working environments.

To gain insight on how to improve VR work environments, our client wants to study time perception in VR. Particularly, he wants to study how lighting, weather, movement, and sound can impact an individual's time perception. The client intends to measure time perception using biometric sensors such as eye tracking, galvanic skin response (GSR), and electroencephalogram (EEG). The insight gained will be used to optimize VR work environments to improve development costs, time-to-market, employee safety, and general efficiency for an industrialized company.

To allow a researcher to gain insight on how environmental factors affect a VR user's perception of time, we developed a system architecture in which a researcher can perform such a study. A researcher can interact with our system, named BioMR, through a custom API which serves as middleware between biometric sensors and a VR scene. A VR subject interacts with a VR environment using an HTC Vive headset while biometric sensors quantitatively measure their perception of time.

The API provides a researcher with a user interface to control and monitor the study. Using the interface, the researcher can manually modify effects in the game engine using input widgets. Additionally, the researcher can view data from the biometric sensors using a 3rd party component, iMotions. iMotions allows for the aggregation and storage of biometric sensor data from multiple sources, and forwards this data to the BioMR API for processing.

We have created two test scenes, one in Unreal Engine 4 (UE4) and another in Unity to validate that the system architecture is collecting meaningful data. Each of these scenes contains controllable effects which can be modified by the BioMR API in real time. Testing with two different game engines ensures that the API is written in a portable way.

2 Requirements

This section contains a full list of condensed client requirements. For each requirement, we describe how the BioMR system architecture acknowledges or meets the requirement.

2.1 The system architecture must have some quantitative measurement of time perception

To meet this requirement, BioMR includes an electroencephalogram (EEG). EEG measures electrical activity in the brain, which can be correlated with time perception.

2.2 The system architecture must store timestamped raw data from sensors

To meet this requirement, BioMR implements a third party application, iMotions, which is responsible for reading and persistently storing biometric sensor data. All data in iMotions is timestamped with both an absolute timestamp, and a media timestamp, which corresponds to the start of video or screen recordings.

2.3 The system architecture must be able to read information from any biometric sensor

To meet this requirement, BioMR uses iMotions to aggregate sensor data. iMotions natively supports more than 50 biometric sensors sensors, and includes an API. The iMotions API allows support for custom sensor types specified by an XML file, thus allowing support for any sensor.

2.4 The system architecture must process raw data, and analyze the data for results to trigger an action

To meet this requirement, iMotions forwards every sample of raw biometric data to the BioMR API for processing. The BioMR API compares all automatic triggers set by a researcher against every received sample. If the sensor threshold is exceeded, an action in the game engine is triggered.

2.5 The BioMR API must be able to communicate with any game engine

To meet this requirement, the BioMR API and game engine implement UDP sockets for communication. Sockets are widely supported in Mac OS, Linux, and Windows. There are libraries for sockets in almost every programming language: Python, C, C++, C#, etc. It is likely that all future game engines will have support for sockets.

2.6 The game engine must send signals to the API indicating user interactions

BioMR implements a test scene in Unreal Engine 4. In this scene, when the player picks up an object, a datagram is sent to the API to indicate this interaction.

2.7 A test scene must be able to modify light conditions, weather conditions, and environmental dynamics based on API commands

BioMR implements a test scene in Unreal Engine 4. This test scene allows the API to change rain intensity, the length of a day, and create new tasks for the VR user. All these changes are controlled through UDP sockets.

2.8 The API GUI must be easy enough to operate without a manual

To meet this requirement, the BioMR API interface emphasizes the design principles of visibility, feedback, and constraints. The API widgets give feedback by changing color during an action, such as an automatic trigger occurring. The interface keeps all options visible so users do not need to recall the possible actions. Additionally, the API restricts user interaction in certain states to prevent errors, such as only allowing valid sensor types to be selected.

2.9 VR Subjects must not be shown 2D interfaces and instructions

Both the Unity and Unreal Engine 4 scenes are purely 3D. There are no 2D menus in either of these scenes.

2.10 The API must take less than 100ms to respond to changes in biometric sensor levels

To meet this requirement, datagrams are processed synchronously in both the API and the Game Engine. Additionally, UDP is used rather than TCP to increase throughput and performance. Dropping a datagram is rare on Localhost, but if it does occur, the next data sample will likely still cause the trigger, so we lose no reliability.

3 Overview of Design

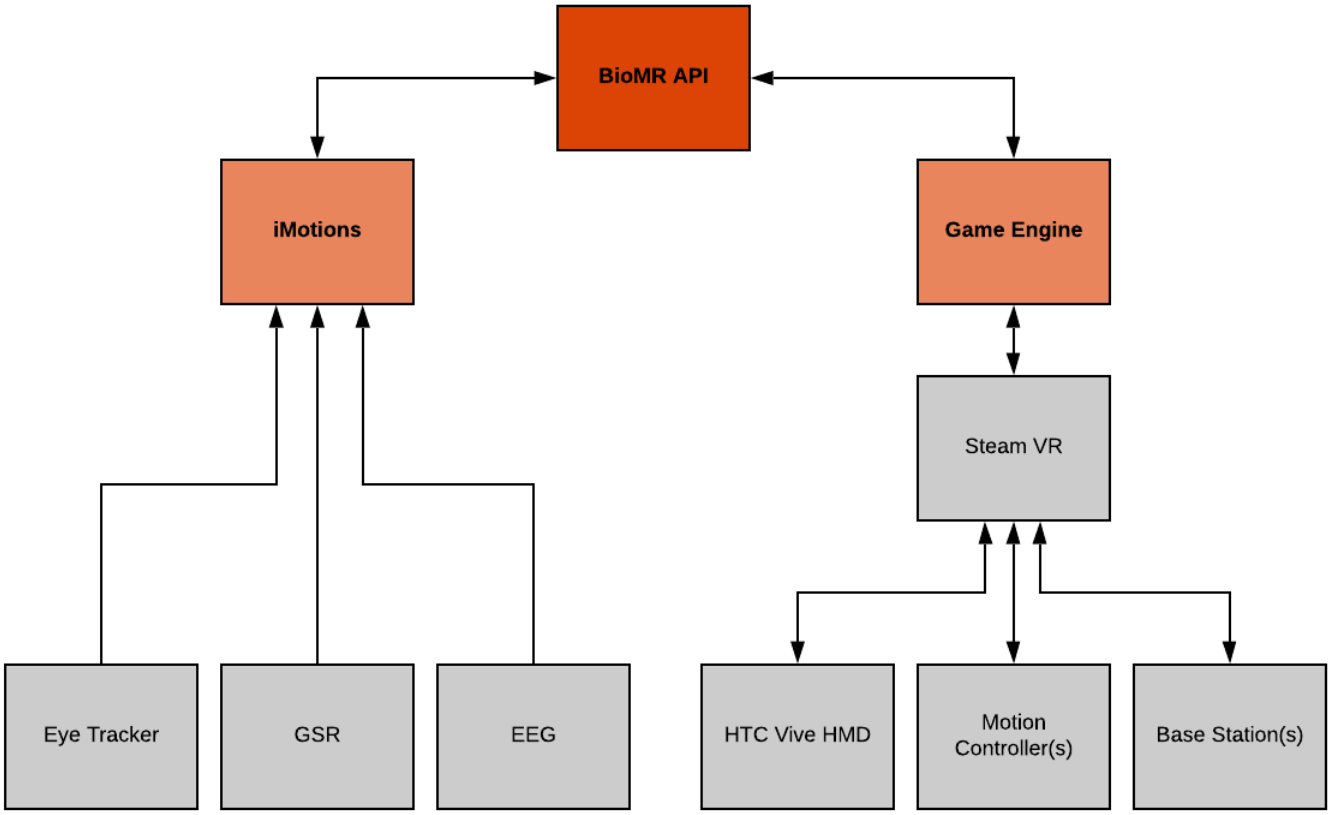

Fig. 1: Dataflow of our system architecture. The sensors send biometric data from the subject to a third party application, iMotions. iMotions stores the data, then sends a copy to the BioMR API. Our API reads the data and determines if any parameters need to be changed in the game engine. If so, the BioMR API sends a signal to the game engine (Unity or Unreal Engine 4) which tells it to make the change. The game engine communicates with the HTC Vive through steamVR and responds to user input. The game engine can also send data to the BioMR API when the user performs certain actions in VR. These commands are forwarded to iMotions and aggregated with the data, so user actions can be correlated with time perception.

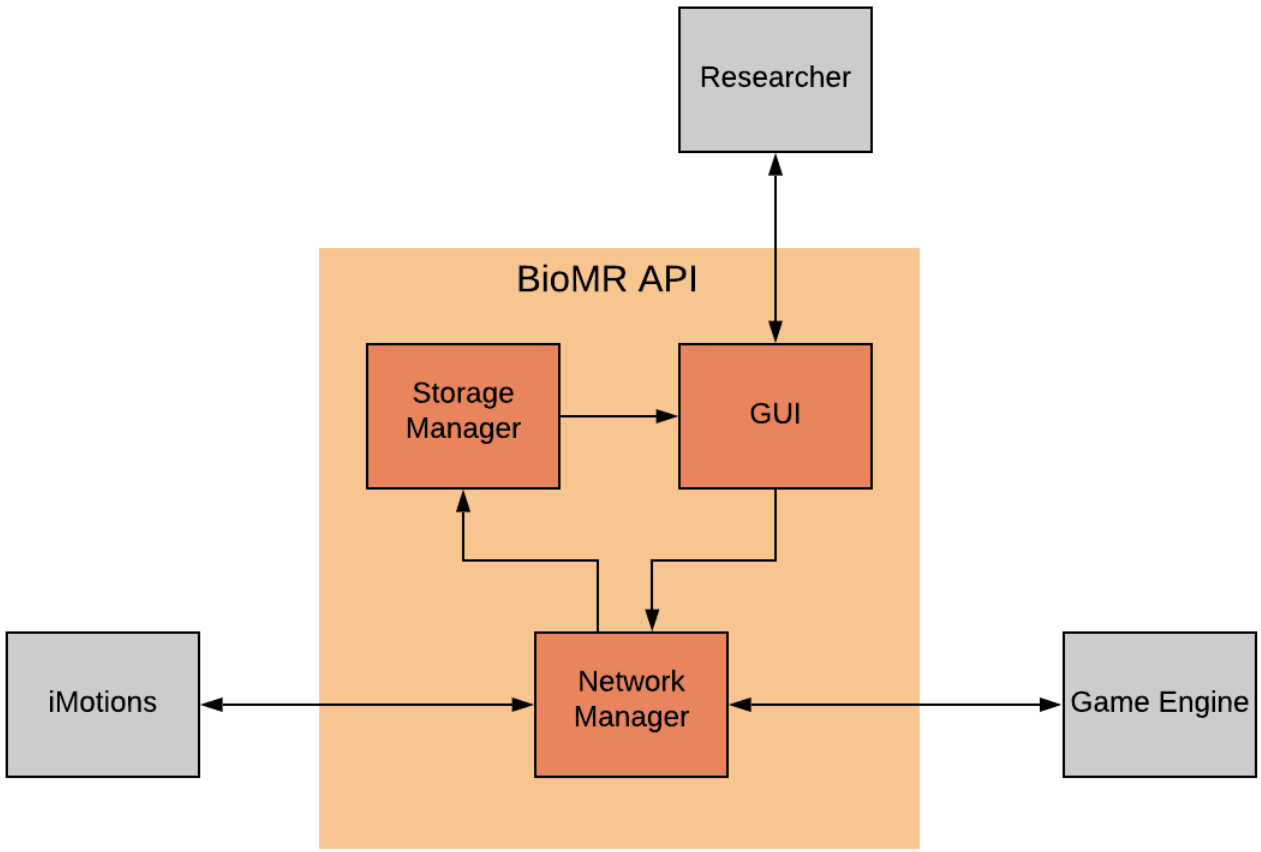

Fig. 2: Design of the BioMR API. iMotions and the game engine communicate with a special module called the network manager. This module consists of 2 UDP senders and 2 UDP receivers. The network manager stores all communications in short term memory in the storage manager. The storage manager updates the GUI based on incoming data. The GUI can automatically trigger a change in the game engine, or respond to researcher manual input. Any generated commands are given to the network manager to be sent over ports.

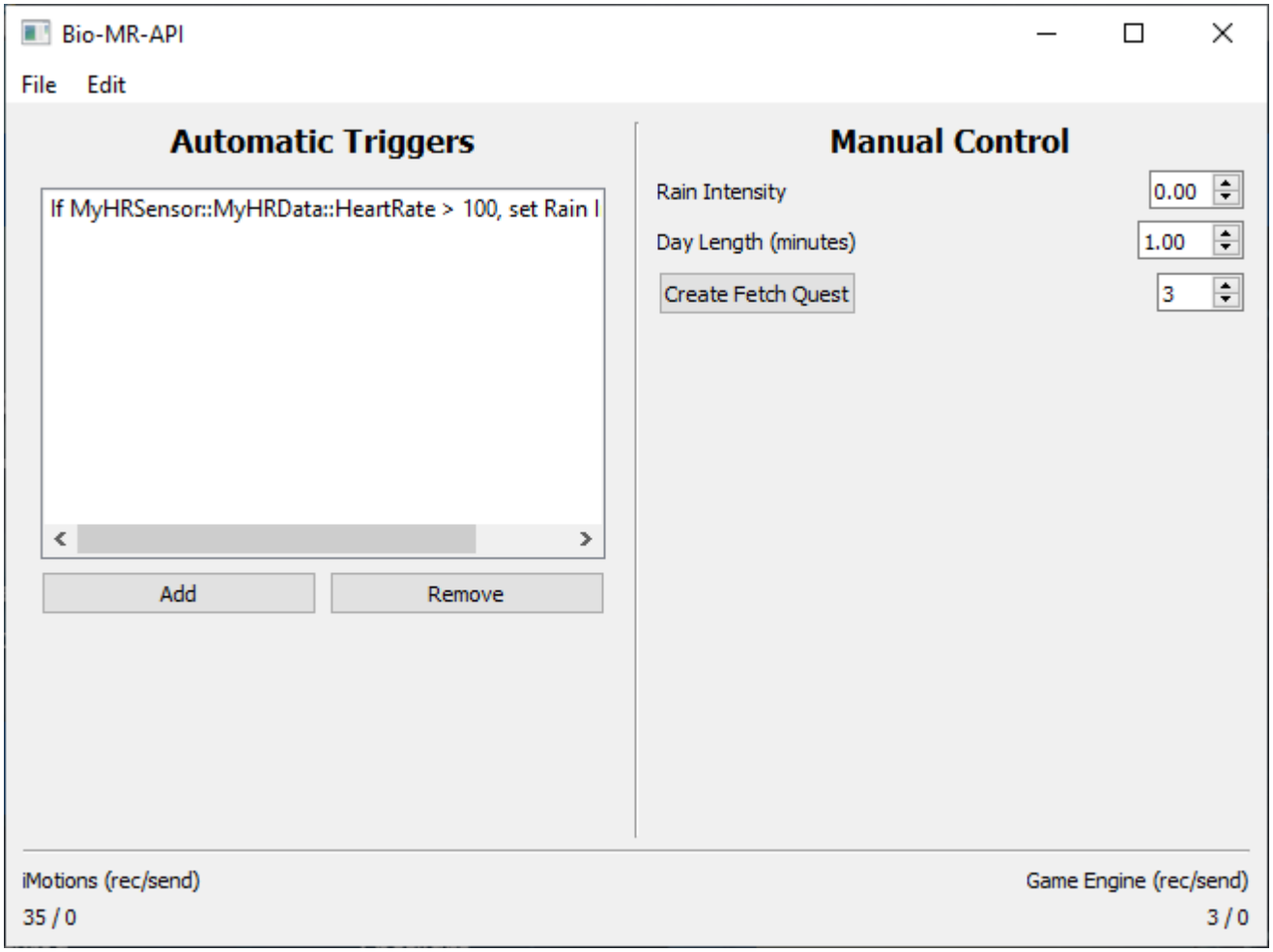

Fig. 3: BioMR API layout. The left side allows the researcher to setup triggers based on sensor data. See Figure 4. The right side allows the researcher to manually change parameters in the game engine. For example, the rain intensity can be modified in real time by changing the value in the spinbox. The bottom counts datagrams sent and received between iMotions and the game engine. This can be used to monitor connectivity.

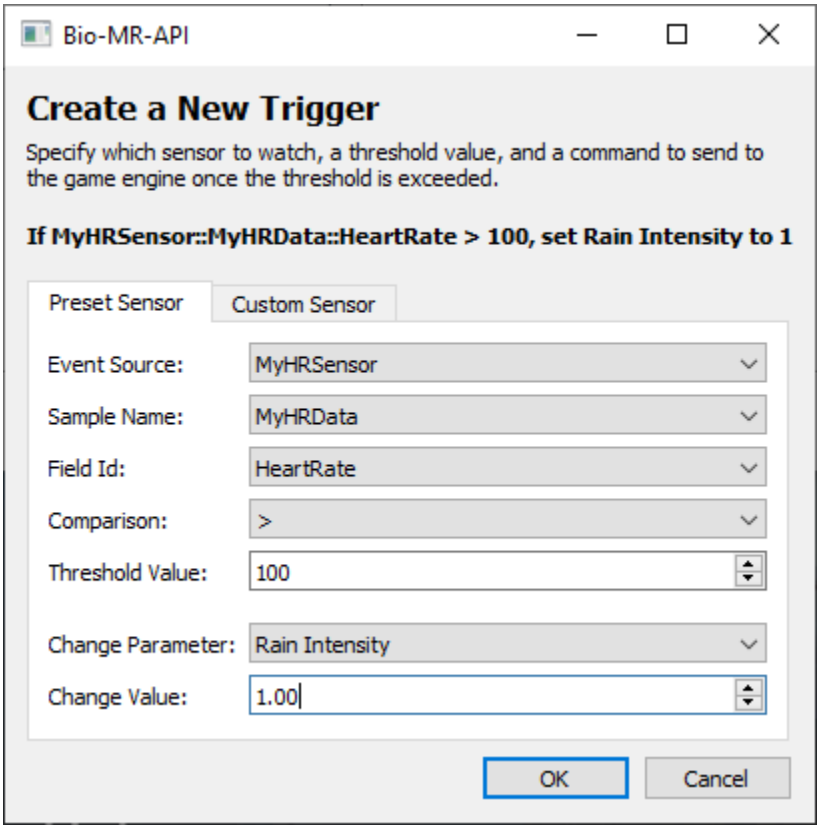

Fig. 4: This widget allows a researcher to set a new trigger. The event source indicates the name of the sensor (i.e. MyHRSensor), the sample name is the name of the particular data (i.e. MyHRData), and the field ID indicates which value in the raw data we're interested in (i.e. HeartRate). The researcher can select a comparison function, and a value to compare to. Once the biometric data passes this threshold, the "Change Parameter" will be set to "Change Value".