Remote Multi-Player Synchronization using the Labstreaming Layer System

Fanourakis, Marios Aristogenis; Lopes, Phil; Chanel, Guillaume

Remote Multi-Player Synchronization using the Labstreaming Layer System

Marios Fanourakis
marios.fanourakis@unige.ch
University of Geneva
Geneva, Switzerland

Phil Lopes
phil.lopes@epfl.ch
EPFL
Lausanne, Switzerland

Guillaume Chanel
guillaume.chanel@unige.ch
University of Geneva
Geneva, Switzerland

ABSTRACT

We present an open-source collection of modules and plugins that can be used to perform synchronous data collection from multi-player games. In its current state, the system can acquire players behaviors (facial expressions, eye-movements, posture, and mouse/keyboard events), physiological reactions (electrodermal activity, heart rate, etc.) together with events indicating the game state. Two games are supported: "Xonotic", and "Counterstrike: Global Offensive" but the system can be easily extended to other games. The different data streams are recorded and synchronized through the use of the Labstreaming Layer (LSL) system. We demonstrate the successful synchronization of physiological responses, eye-movements, and a camera to within 42*ms*.

KEYWORDS

affective computing, video games, multi-player, data synchronization, physiological data, Labstreaming Layer

ACM Reference Format:
Marios Fanourakis, Phil Lopes, and Guillaume Chanel. 2020. Remote Multi-Player Synchronization using the Labstreaming Layer System. In *Proceedings of FDG 2020*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Interactive media experiences such as video games are a versatile and multi-faceted stimulus. Video games are more and more realistic with detailed graphics, accurate physics, convincing emotional characters, and immersive virtual reality. As such, video games elicit complex player experiences which makes them quite attractive in the research community. A common finding is that it is necessary to perform multimodal recordings in order to capture as much as possible of the player experience during video game play; a task that requires the synchronization of several heterogeneous sensors and events. Research labs either develop their own experimental platforms or purchase an often costly platform that fits their needs like Biopac[1] and iMotions[2]. Although both options have advantages, we aim at developing a modular open source solution which might fit the needs of several research cases without being costly.

In this paper, we present a modular solution for synchronous recording of video game play (a video of the game and game events); player behaviors such as facial expressions, eye-movements, posture, and mouse/keyboard events; and physiological reactions such as electrodermal activity (EDA), electrocardiogram (ECG), respiration, and electromyography (EMG). The proposed solution consists of a collection of modules which can record data from multiple players connected remotely. With this collection we seek to further facilitate experiments for studying human emotions, behaviours, and responses during video game fruition. The software modules are open source[3] and licensed under the *MIT* license. We developed these modules for the Labstreaming Layer (LSL) [4], a popular and open source data synchronization system. Several applications have been developed for various sensors and software to stream their data to LSL and are available online at the official LSL repository or from other sources.

The LSL system is confined to the local area network (LAN) which is adequate for the majority of experiments. However, in some situations it is necessary to collect data remotely. For example, to allow players to be at the comfort of their own home. To facilitate remote data collection and synchronization we propose the use of a virtual private network (VPN).The performance of the system in term of modality synchronisation is thus evaluated for both local and remote connections.

2 RELATED WORK

Over the past few years the collection of player data has become a serious consideration and necessity for both researchers and developers [1, 2]. Player telemetry allows designers to observe and obtain an accurate representation of several player behaviours, which in turn can be used to make important game design decisions and modifications. Telemetry is also often used for matchmaking and ranking players in competitive multiplayer games [3], as a way of providing a more enjoyable experience for players who fall in different skill ranges.

Due to the ability of video games to immerse players and to elicit complex emotions, it has been widely studied in the domain of psychology where, in addition to in-game measures, various physiological data (ex. ECG, EDA), facial expressions, and body posture are collected [4–9]. It can often be difficult to manage the multitude of different data streams due to synchronicity issues. Although the usage of physiology to study player behaviour is quite common in the literature, it is rarely done within a multiplayer setting where

[1] https://www.biopac.com/
[2] https://imotions.com/

[3] https://gitlab.unige.ch/sims/esports-data-platform
[4] https://github.com/sccn/labstreaminglayer

FDG 2020, September 15-18, 2020, Malta

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

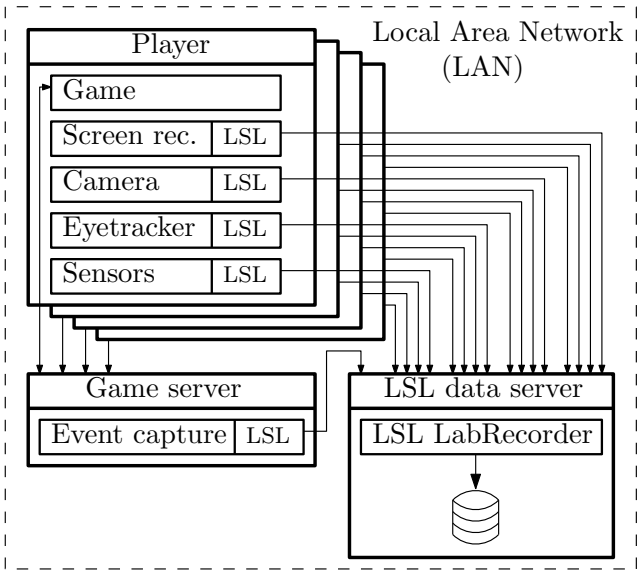

Figure 1: The architecture of the data collection and synchronization platform.

each player's data streams are collected concurrently. Furthermore, although multimodal data collection has been achieved, we are not aware of a study which collected all the modalities presented in this paper in a multi-player scenario. Topics of interest include, behavioural effects of video games, differences of affective responses to games between different types of players, dynamic difficulty adjustment based on player's affective state, effect of game elements on player experience, and many more. Developers can also benefit from the this research by creating emotionally intelligent games and peripherals to better engage players.

3 ARCHITECTURE

The resulting platform has a client-server architecture that relies on the LSL system for data synchronization and modularity. Adding or removing components is trivial as long as these components have an LSL stream outlet and have a command-line interface (CLI) with the required parameters.

An overview of the architecture is shown in Figure 1. There are several components and sub-components:

- The **game server** (section 3.1).
- The **player** (section 3.2) includes several sensors and components: the client **game**, the **screen recorder**, the input to **peripherals**, the **camera**, the **eyetracker**, the **sensors**.
- The **LSL data server** (section 3.3).

3.1 Game server

Game events and state are captured on the game server and pushed to an LSL outlet. The game server may be modified to provide a richer set of events and/or a more native LSL integration. Currently, two video games are compatible with the recording platform: Xonotic, and Counter strike: Global Offensive.

Figure 2: Xonotic gameplay screenshot.

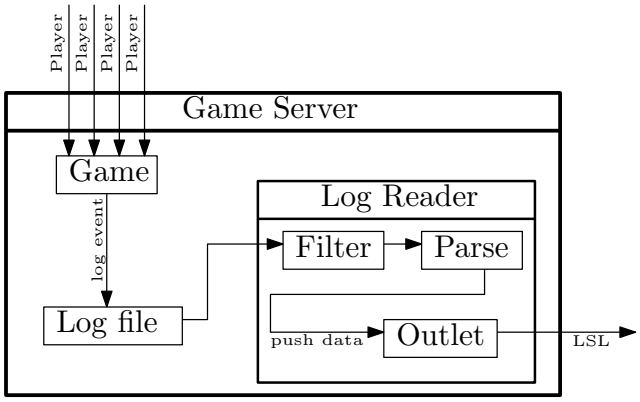

Figure 3: Xonotic data capture architecture.

3.1.1 Xonotic. Xonotic is an open source multi-platform online multiplayer first person shooter (FPS) game that resembles Quake III (Figure 2. Our strategy to record Xonotic events is to develop a client which parses the game log and send relevant events to an LSL outlet (see Figure 3). It is preferable to configure the output log as a POSIX pipe in order to minimize latency. Given that Xonotic is open-source we modified the logging system to add events of interest such as power ups and health stats. However, given that several events are logged by default our recording strategy does not require any modifications per se and can easily be applied for games that are difficult to modify. By using this strategy, we are able to capture several events like: item pickups, kill events, damage events, and more.

3.1.2 Counter strike: Global Offensive. Counter strike: Global Offensive (CS:GO) is a popular free online multiplayer FPS game (Figure 4. It is developed by Valve and is using the Source engine. Valve provides various tools to modify the game engine and subsequently there is a large community of modders and plugins for the game.

Figure 4: Counter strike: Global Offensive gameplay screenshot.

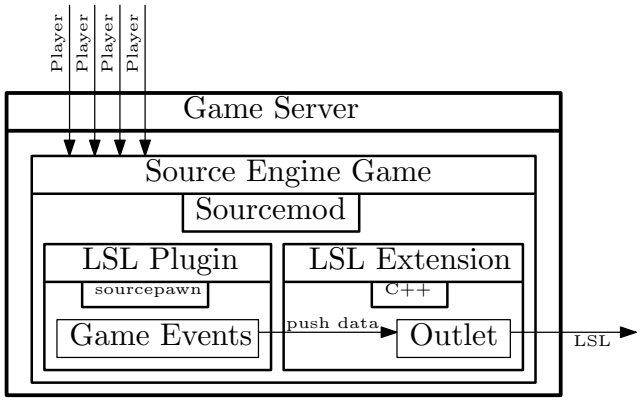

Figure 5: CS:GO data capture architecture.

We have used Sourcemod and Metamod:source to further accelerate the development and portability of our modification plugins. First, we developed a native extension in C++ to provide an interface to LSL that any CS:GO plugin can exploit. Then we developed a plugin written in the sourcepawn programming language to capture game events and use the LSL interface to push data to an LSL outlet. Using this plugin we are able to capture many events like: position of player, player health, player armor, kill events, and many more. The data capture architecture is shown in Figure 5.

3.2 Player components

For each player we capture data from several devices. The LSL applications developed to acquire this data are based on different architectures depending on the size of each sample. The sample-to-outlet architecture in Figure 6 is used for small per-sample size (e.g. physiological signals, eye-movements). The ID-to-outlet architecture in Figure 7 is used for large per-sample size (mainly video recordings). Doing this distinction is important for assuring the recorded data is manageable, to not overload the network.

We have either adopted existing LSL applications, developed new ones, or integrated LSL to existing software to capture data from several hardware sensors and peripherals. Each component is described in the sub-sections that follow.

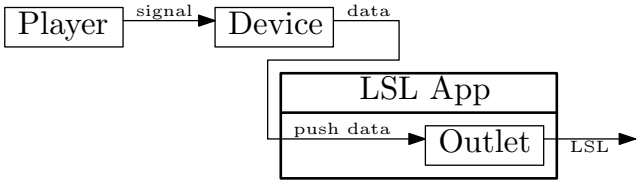

Figure 6: Sample-to-outlet architecture for components with small per-sample size.

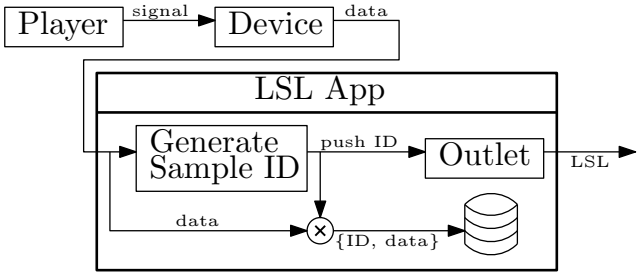

Figure 7: ID-to-outlet architecture for components with large per-sample size.

3.2.1 Game client. The game client is an unmodified version of the Xonotic or CS:GO game.

3.2.2 Screen recorder. It is an LSL-enabled software application which is able to record the screen of the player and send synchronization parameters for each frame through LSL. The purpose of this recording is two-fold: (*i*) having a screen capture can be useful to annotate the game, (*ii*) game streaming is becoming a common activity which might also gain research interest. The screen recorder must be on the same physical computer as the game client. We have opted to use the Open Broadcaster Software[5] (OBS) to record the screen of the user as it is a common platform among game streamers. This software allows us to record any in-game audio, microphone, and webcam video in the same video file. We have developed an LSL plugin for this software which uses the ID-to-outlet architecture since video frame samples are relatively large. This means that the LSL plugin only pushes the frame number to the LSL outlet while the video is stored locally.

3.2.3 Peripherals. The native LSL repository includes applications for streaming data from input devices such as the mouse and keyboard. Since the sample size is relatively small, the sample-to-outlet architecture is used. This component must be on the same physical computer as the game.

3.2.4 Camera. Is used to record the player to potentially extract facial expressions or other visual features. We use the Intel Realsense D435 RGBD camera[6] which is a stereo camera able to capture video and depth. Intel provides a Software Developer Kit (SDK) to develop software that can interface with the camera. We have developed a software application in C# that records the data in RAW format following the ID-to-outlet architecture.

[5]https://obsproject.com/
[6]https://www.intelrealsense.com/depth-camera-d435/

3.2.5 Eyetracker. Is able to detect where the player is looking at on the screen, their pupil diameter, the position of their eyes relative to the tracker device, and other gaze related data. We use the Tobii pro nano[7] to capture eye tracking information. Tobii provides an SDK to develop software that can interface with the device. The LSL repository includes an application written in C. Since the per-sample size is relatively small for the eye tracking camera, the sample-to-outlet architecture is used.

3.2.6 Physiological sensors. To measure palyer's physiological activity we are using the BITalino platform[8]. This is a low-cost and open-source hardware which can measure several bio-signals. In our setup we measured EDA, ECG, EMG, and respiration. The data can be sent via bluetooth to a computer. The BITalino SDK was used to develop a component in C# that pushes the data to LSL. Since the per-sample size is relatively small for the physiological data, the sample-to-outlet architecture is used.

3.2.7 Pressure Mat. We are using a Sensing Tex seat pressure mat[9] to record the seating posture and movements of the players while they are engaged in the video game. We have developed a component in C# to push this data to an LSL outlet. Since the per-sample size is relatively small for the pressure mat, the ample-to-outlet architecture is used.

3.3 LSL data server

This component can be on the same physical computer as the game server or a separate computer. We use the LSL LabRecorder[10] to subscribe and synchronize all the data samples that are being pushed to LSL and store the data in a single file in the extensible data format (XDF).

3.4 Remote play

LSL is limited to the local area network, however, there are situations when it is necessary to perform an experiment where players and devices are on separate networks. To overcome this, we have successfully tested two solutions: tierZero, and openVPN which are both open-source solutions.

4 SYNCHRONIZATION PERFORMANCE

We have evaluated the synchronization performance on a LAN between the following sensors: Bitalino with ECG sensor, Intel Realsense D435 camera, and Tobii pro nano eyetracker. To evaluate how well these devices are synchronized in the recorded LSL data we require an independent signal that is captured by each of the devices simultaneously. To that effect we have used a square wave signal generator with frequency $0.5Hz$ and duty cycle 5% to drive a LED and as an input to the ECG sensor. This provided a common signal for the camera (via the LED) and the ECG sensor. This signal could not be used for the eyetracker. To address this issue, we needed an additional signal compatible with the eyetracker and at least one of the other sensors. A subject was instructed to sit still in front of the eyetracker, to look at the camera, and to blink frequently. It was suggested to use the LED light of the other signal as a trigger to blink consistently, but it was not a strict requirement. The subject's blinking provided a common signal for the camera and the eyetracker. The Bitalino was setup to sample at $100Hz$ and was connected to the computer via Bluetooth. The Intel Realsense camera was setup to capture both the LED and the subject's face at 30 frames per second at a resolution of $640x480$ pixels. The Tobii pro nano eyetracker was setup to sample at $60Hz$.

Once the data was recorded, the Intel Realsense video had to be annotated manually to label the blinking events and the LED events. The blinks in the eyetracking data were labeled automatically as they are detected by the recording system. Then, the delays were calculated for the following pairs: (1) between the ECG sensor and the camera, (2) between the eyetracker and the camera.

For the first pair, we calculated a mean delay of $543ms$ (camera data arrived later than the ECG data) with a standard deviation of $11ms$. This relatively small standard deviation means that we can easily set an offset parameter in the LSL outlet to mitigate this delay. Once this offset is accounted for, we can expect that most of the data will be synchronized to within twice the standard deviation ($22ms$).

For the second pair we calculated a mean delay of $63ms$ (camera data arrived later than the eyetracker data) with a standard deviation of $13ms$. Again, the relatively small standard deviation means that we can adjust the LSL outlet offset parameter to mitigate some of the delay. For this pair, we can expect that most of the data will be synchronized to within $26ms$.

The ECG sensor and the eyetracker did not have a common signal for direct evaluation of the synchronization. To estimate how well they are synchronized we can add the two standard deviations of the previous pairs (ECG sensor+camera, eyetracker+camera) to get a synchronization measure of within $48ms$ for most of the data. It is important to note that, for the camera and eyetracker, we used the system timestamp and not the device timestamp. It would therefore be worth to investigate using the device timestamp for pushing the LSL sample to the outlet as a way to reduce the relatively large initial offset without doing manual measurements.

5 LIMITATIONS AND FUTURE WORK

We plan to evaluate the synchronization of additional sensors in the system which will require more complicated modifications. For example to test the synchronicity between the keyboard input and the Bitalino sensors it would require a hardware modification on the keyboard.

Although the LSL system has adequate performance for the evaluated sensors in the LAN setup which it was developed for, there is the need for additional timing adjustments in the remote recording setup (using VPN). These adjustments should be transparent to the user and dynamic with respect to the network conditions.

Several of the modules described here only support Microsoft Windows operating system. In an effort to expand the usability of our software we plan to port these modules to C++ for native compatibility in linux.

The number of sensors with LSL integration is relatively low, to address this, we may develop additional modules as needed for our experiments which will also be available publicly.

[7] https://www.tobiipro.com/product-listing/nano/
[8] https://bitalino.com/en/
[9] http://sensingtex.com/sensing-mats/seating-mat/
[10] https://github.com/sccn/labstreaminglayer

REFERENCES

[1] Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. *Game analytics*. Springer, 2016.

[2] Anders Drachen, Alessandro Canossa, and Georgios N Yannakakis. Player modeling using self-organization in tomb raider: Underworld. In *2009 IEEE symposium on computational intelligence and games*, pages 1–8. IEEE, 2009.

[3] Olivier Delalleau, Emile Contal, Eric Thibodeau-Laufer, Raul Chandias Ferrari, Yoshua Bengio, and Frank Zhang. Beyond skill rating: Advanced matchmaking in ghost recon online. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):167–177, 2012.

[4] J. Matias Kivikangas, Guillaume Chanel, Ben Cowley, Inger Ekman, Mikko Salminen, Simo Järvelä, and Niklas Ravaja. A review of the use of psychophysiological methods in game research. *Journal of Gaming & Virtual Worlds*, 3(3):181–199, sep 2011.

[5] Thomas Christy and Ludmila I. Kuncheva. Technological Advancements in Affective Gaming: A Historical Survey. *GSTF Journal on Computing (JoC)*, 3(4):38, apr 2014.

[6] Georgios N Yannakakis, Hector P Martinez, and Maurizio Garbarino. Psychophysiology in games. In *Emotion in games*, pages 119–137. Springer, 2016.

[7] Andrea Clerico, Cindy Chamberland, Mark Parent, Pierre-Emmanuel Michon, Sebastien Tremblay, Tiago H Falk, Jean-Christophe Gagnon, and Philip Jackson. Biometrics and classifier fusion to predict the fun-factor in video gaming. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[8] Adapting software with Affective Computing: a systematic review. *IEEE Transactions on Affective Computing*, 3045(c):1–1, 2019.

[9] Guillaume Chanel and Phil Lopes. User evaluation of affective dynamic difficulty adjustment based on physiological deep learning. In *2020 Proceedings of the Conference on Human-Computer Interaction International*, 2020.