HapticSeer: A Multi-channel, Black-box, Platform-agnostic Approach to Detecting Video Game Events for Real-time Haptic Feedback

Yu-Hsin Lin
National Taiwan University
Taipei, Taiwan
yuhsin.lin@outlook.com

Yu-Wei Wang
National Taiwan University
Taipei, Taiwan
willieyuwei4@gmail.com

Pin-Sung Ku
National Taiwan University
Taipei, Taiwan
scott201222@gmail.com

Yun-Ting Cheng
National Taiwan University
Taipei, Taiwan
r09922063@g.ntu.edu.tw

Yuan-Chih Hsu
National Taiwan University
Taipei, Taiwan
wendeehsu@gmail.com

Ching-Yi Tsai
National Taiwan University
Taipei, Taiwan
r09944022@ntu.edu.tw

Mike Y. Chen
National Taiwan University
Taipei, Taiwan
mikechen@csie.ntu.edu.tw

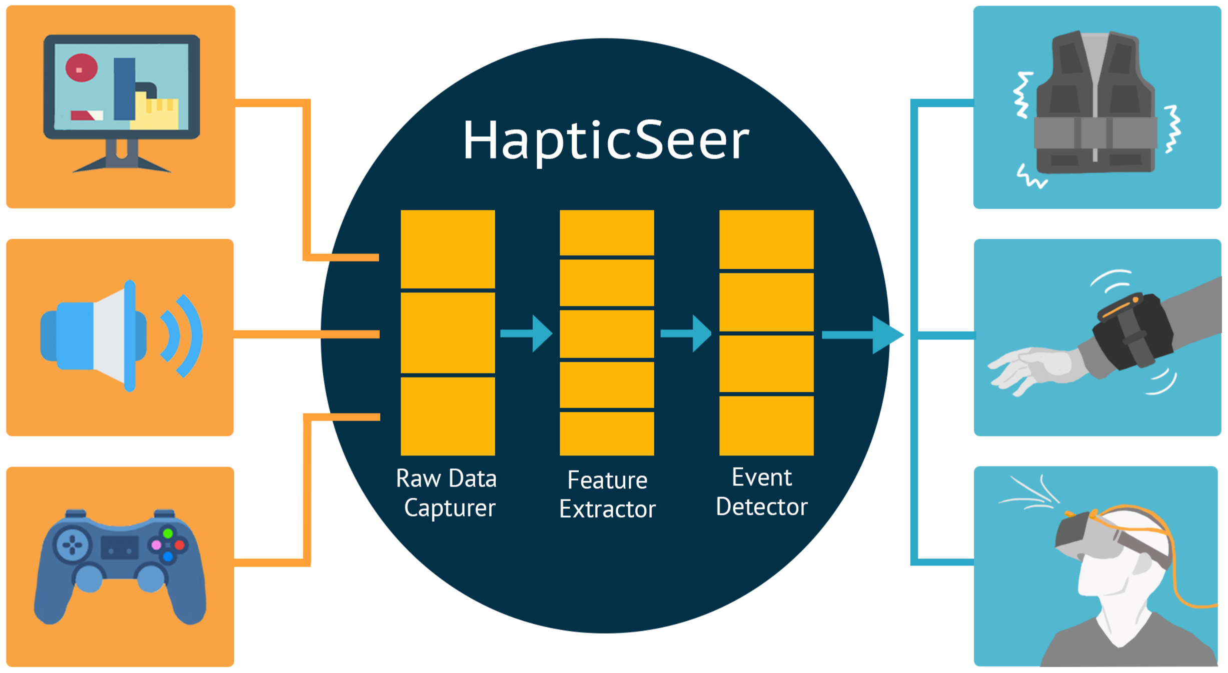


Figure 1: A high-level display of HapticSeer's architecture. Workflow from left to right: a) Use black-box approaches to collect data from video games. b) Process data and generate semantic features with HapticSeer. c) Trigger haptic devices in real-time when game events are detected. (Icon credits: by authors and also from flaticon.com)

CHI '21, May 8–13, 2021, Yokohama, Japan
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445254

ABSTRACT
Haptic feedback significantly enhances virtual experiences. However, supporting haptics currently requires modifying the codebase, making it impractical to add haptics to popular, high-quality experiences such as best selling games, which are typically closed-source. We present HapticSeer, a multi-channel, black-box, platform-agnostic approach to detecting game events for real-time haptic feedback. The approach is based on two key insights: 1) all games have 3 types of data streams: video, audio, and controller I/O, that can be analyzed in real-time to detect game events, and 2) a small number of user interface design patterns are reused across most games, so that event detectors can be reused effectively. We developed an open-source HapticSeer framework and implemented several real-time event detectors for commercial PC and VR games. We validated system correctness and real-time performance, and discuss feedback from several haptics developers that used the HapticSeer framework to integrate research and commercial haptic devices.

CCS CONCEPTS
• Information systems → Multimedia and multimodal retrieval; **• Human-centered computing → Systems and tools for interaction design**.

KEYWORDS
Haptics, Multi-modal, Event detection framework,

ACM Reference Format:
Yu-Hsin Lin, Yu-Wei Wang, Pin-Sung Ku, Yun-Ting Cheng, Yuan-Chih Hsu, Ching-Yi Tsai, and Mike Y. Chen. 2021. HapticSeer: A Multi-channel, Black-box, Platform-agnostic Approach to Detecting Video Game Events for Real-time Haptic Feedback. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3411764.3445254

1 INTRODUCTION
Haptic feedback significantly enhances virtual experiences. However, adding haptics to a virtual experience currently requires having access to modify its codebase. For example, for game developers to vibrate controllers, they need to add code to invoke the haptics API provided by the software development kit (SDK) of the platform, such as `Handheld.Vibrate()` when using Unity3D.

This limitation makes it challenging to add haptics to commercial games, which are typically closed-source. One workaround that haptics developers have used is game audio analysis, by triggering vibration and varying its intensity based on audio features such as volume and impulses. However, this audio-only approach is limited in precision and limited in the types of events it can support.

We present HapticSeer, a multi-channel, black-box, platform-agnostic approach to detecting game events for real-time haptic feedback. Our approach is motivated by two key insights:

- All games produce 3 types of data streams: 1) video, 2) audio, and 3) controller I/O, that can be analyzed to detect game events.
- A small number of user interface design patterns are used by most games, so that code written to detect events for a game can be reused for many other games.

We developed an open-source HapticSeer framework[1] to capture video, audio, and controller I/O in real-time. It provides the captured data, event system, and abstraction for *feature extractors* and *event detectors* to simplify the integration of haptic devices, as shown in Figure 1. *Feature extractors* process raw video, audio, and controller sensor data to produce features that can be used by other feature extractors and event detectors. *Event detectors* generate events that haptics developers can listen for in order to actuate their haptic devices.

To validate the system correctness and performance, we implemented several real-time, multi-channel event detectors for commercial PC and VR driving and first-person shooter games. These include a speed detector, an inertia detector, a player hit detector, and a gun fire detector for Project CARS 2, Half-Life: Alyx and Battlefield 1 (their gameplay are shown in Figure 2).

To understand how UI design patterns are reused across popular games, we surveyed 30 top first-person shooter (FPS) games [12] and 20 top driving games [6, 7] and categorized their designs for 4 common types of information: 1) ammunition count, 2) health status, 3) damage indicator, and 4) driving speed. Figure 3 shows the percentage of games that use a particular UI design, and shows that the top two UI designs are used by more than 80% of the games across all 4 types of information display. For example, the top design for ammunition count displays the information as numbers (87%) and the top two designs for driving speed display the information as numbers and tachometers (80%).

To collected feedback from haptics developers, we invited several developers to integrate their haptic devices with commercial games using HapticSeer. All were successful in using the provided event detectors and expressed interests in using HapticSeer for their devices. Developers reported that HapticSeer could make the integration of haptic devices and commercial games easier while retaining imperceptible end-to-end latency in study cases; moreover, it only took them less than two hours (1 hour and 13 minutes in average) to learn HapticSeer and integrate the devices, which is relatively a short time for developing demo applications.

2 RELATED WORK
Force feedback has been a ubiquitous and famous topic to enhance the immersion of game experiences. However, demonstrating haptic experiences usually require well-designed contents of demo scenes, which is a time-consuming job. To explore the possibility of developing real-time force feedback systems, we review some existing approaches, including white-box approaches and black-box approaches. For specific, white-box approaches require announced application programming interfaces (APIs) for integration with haptic devices, while black-box approaches process data from user-acquirable channels, including visual channels, audio channels, and I/O channels from peripherals.

2.1 Explicit API Approach
2.1.1 Haptic Device API for Games. Some commercial games nowadays support external haptic devices, such as Xbox One Controller [26] and Logitech Gaming Racing Wheel [16]. These devices have released SDKs for everyone to integrate content with their devices;

[1]Available at HapticSeer.org and https://github.com/ntu-hci-lab/HapticSeer

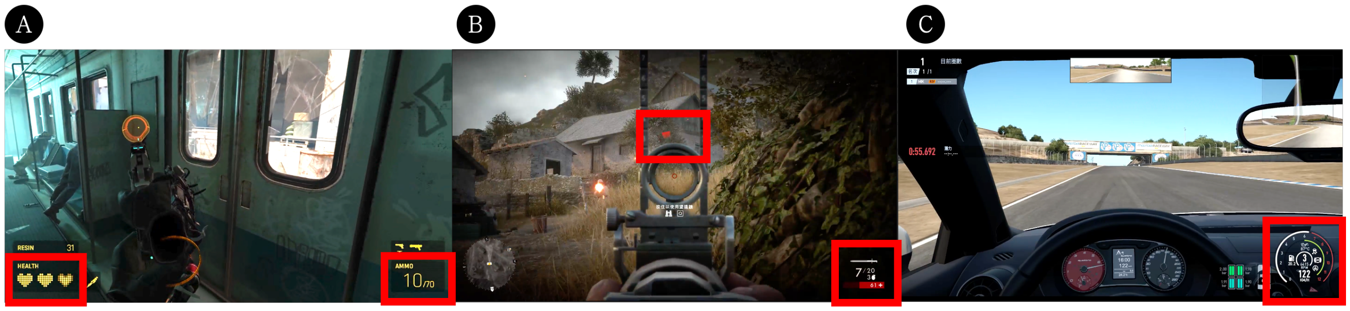


Figure 2: Gameplay footage of three popular commercial games: (A) Half-Life: Alyx, (B) Battlefield 1 and (C) Project CARS 2. The heads-up displays (HUDs) on screen are highlighted.

for example, Xbox provides APIs for developers to fetch the button presses events or transmit a rumble request. Haptic developers do not need to know what game events have occurred. That is, haptic developers only need to provide an interface of haptic devices that can transmit controller status and receive requests from games. However, haptic developers would not be involved in the future integration of new games and their devices, which may lead to compatibility issues.

2.1.2 API Exported from Games. Some commercial games provide telemetry APIs for external software and hardware to fetch in-game information. Some works utilize telemetry APIs to receive in-game information and parse it to control external hardware, such as HeadBlaster [15] and Sim Racing Studio App [21]. However, most of the games supporting such a method to monitor game events are car-racing simulators. Another limitation is that haptic developers can only retrieve predefined events provided natively by games, such as vehicles' velocities. It is challenging for haptic developers to fetch specific information from telemetry APIs, such as the wheels' texture. Thus, this approach is only suitable for limited haptic devices.

Another method to fetch in-game information involves utilizing game modifications (mods). Game mods are external software or materials that would affect game behaviors. Mod-friendly games provide support for modifying game routine, so haptic developers can utilize mods to extract game events not provided natively by these games. For example, HaptiVec [8] modifies an open-source game, Chocolate Doom, to extract the direction of an attack that hits the player. However, not every game supports the modification of games because of the anti-cheating or anti-copying mechanisms. Besides, the mod codes are usually not reusable if haptic developers try to make their devices support other games.

2.2 Black-box Approach

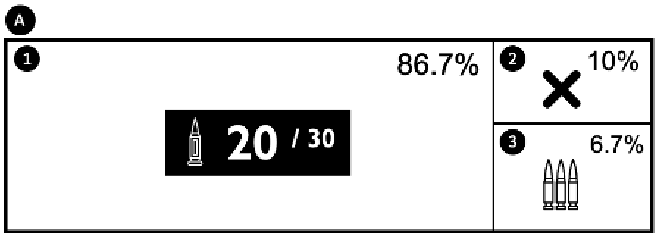
2.2.1 Using Visual Channel. Vision is one crucial sense organ of the five senses because vision contains rich information. Players generally focus on the screen while playing games; therefore, game developers design several information panels providing essential information for players, such as the ammunition count, the speedometer of cars, and the player's health bar. Navigation systems also present multiple information via vision rather than using voice prompts. Several works have presented the feasibility of extracting information from vision, and a famous one is Sikuli [27]. Sikuli is a framework that allows GUI testers to test their graphical user interface (GUI) automatically with writing scripts. Several images are specified in the scripts to define which GUI widgets to be tested. Sikuli utilizes computer vision techniques to search images predefined in scripts from the screen. After a comparison, Sikuli can perform an action on the corresponding widgets on the screen.

2.2.2 Using Audio channel. Sound effects in games can improve immersion significantly. Nowadays, game sound effects are capable of simulating different direction and intensity of the sound source. The most common sound system is stereophonic sound, which uses two independent speakers to create the illusion of sound direction. To further enhance the directional distinguishability, some games support surrounding sound systems. For example, a 5.1 surrounding sound system uses five speakers, including a front-left, a front-right, a left, a right, and a center speaker to reproduce full-range sound effects, and is equipped with a sub-woofer to reproduce low-frequency effects (LFE). A commercial product, bHaptics [3], utilizes the feature of surrounding sounds to integrate commercial games with their haptic devices.

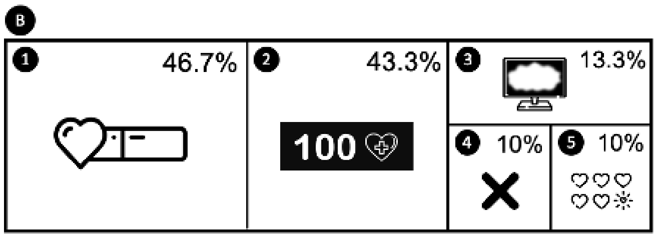
2.2.3 Using Controller I/O channel. Haptic feedback has been explored since the late 1990s and has been added to controllers to enhance immersion experiences. Controller I/O is an integrated part of the interaction between games and players. Some shooting games make the controller vibrate to simulate the recoils of a gun firing. To apply force feedback on self-designed controllers, monitoring the vibration events is an easy way. A commercial force feedback device, ForceTubeVR [11], exploits the controller I/O channels, which is called SteamVR Backward Compatibility. When a rumble request from SteamVR is sent, the ForceTubeVR application will actuate their haptic device. However, other game events may induce vibration, such as walking on gravel or being attacked by enemies. Therefore, it is troublesome to distinguish events from only vibration information.

3 APPROACH

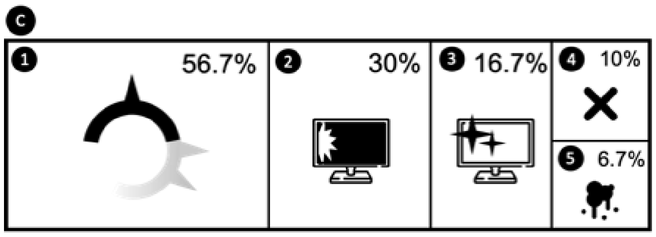
To achieve our purpose in providing a framework for haptic developers to integrate commercial games with their devices, we designed a workflow for the framework. HapticSeer decouples in-game data analysis into interchangeable components with three abstract levels - data capturers, feature extractors, and event detectors. Following



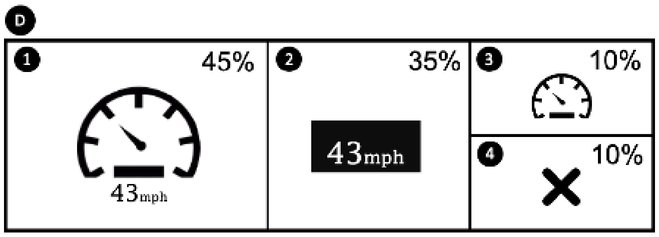
(A) Ammunition count. Display method: (1) digit (2) no display (3) bullet icons



(B) Health. Display method: (1) bar (2) digit (3) reduce point-of-view (4) no display (5) other forms



(C) Damage indicator. Display method: (1) 2D indicator (2) screen edge indicator (3) flashing (4) no display (5) blood effect



(D) Speed. Display method: (1) digit and tachometer (2) digit (3) circular speedometer (4) no display

Figure 3: Display pattern categories of the four HUD types in the top 30 FPS games and the top 20 driving games. A higher percentage means a higher frequency of occurrence in games. Some games may simultaneously possess multiple display methods within one HUD type. (Icon credits: by authors and also from flaticon.com)

this design pattern, HapticSeer become a flexible and collaborative system that allows other developers to design and share their feature extractors or event detectors. As more contributions are made, the system will be more robust and versatile.

3.1 Data Streams

Visual, audio, and controller I/O are three essential input channels to support in-game data collection without any API or game modification. The visual channel provides rich information and extractable cues, though it is restricted in the player's field of view; therefore, events happening off-screen are unobservable. The audio channel, in contrast, can receive omnidirectional information. It is relatively easier to extract features from audio waveform than from images. However, audio-based models are sensitive to noises, and the requirement of time windows causes latencies. Controller I/O channel possesses the advantages of having zero latency and being a noiseless data stream directly derived from user input. However, merely obtaining controller information is inadequate to construct a complete semantic message.

Event detectors in HapticSeer rely on multiple data streams to make up each channel's deficiencies. For example, calculating the centripetal acceleration of vehicles in driving games needs data from both video and controller I/O channels. A video feed provides the current speed by HUD, while controller input indicates the vehicle's turning angle. With the multi-channel approach, the amounts of detectable events could be extended, and the event detectors could operate more precisely.

3.2 Design Patterns

Heads-Up Display(HUD) is a universal interface design in commercial games to display character status, and HUD patterns are identical in games with the same genres. For example, in first-person shooter games, we can find displays such as ammunition count, current health, weapon type, and mini-map. Though each game has its unique interface design, HUDs can still be categorized into few categories. Our event detectors could extract information from the same categories with slight adjustments.
We surveyed 30 top FPS games [12] and 20 top driving games [6] [7] to categorize HUD types and searched for their common grounds. As shown in Figure 3, 86.7% of the ammunition count is displayed by digits, and 6.7% is displayed by icons, while three out of 30 games do not show the ammo information. Health status is displayed 43.3% by digits and 46.7% by health bars. Sometimes, the health display icon might be unique forms such as heart shapes or circular format. 13.3% of the games do not hold a health illustration. Instead, they narrow down the more point-of-view(POV) as the player loses the more health. As for damage indicators, there are six kinds of representations. The most common ones are 2D indicators and screen edge indicators, having 56.7% and 30% occurrence rates. Damages displayed by other types like 3D indicators, screen flashing, screen blur, or bloodstain range from 6.7% to 16.7%. Except for screen flashing and blurring, all of the damage indicators are directional. In the top driving games, 80% of the games use digits as a speed showcase, and 55% of them have an extra circular speedometer or a tachometer.

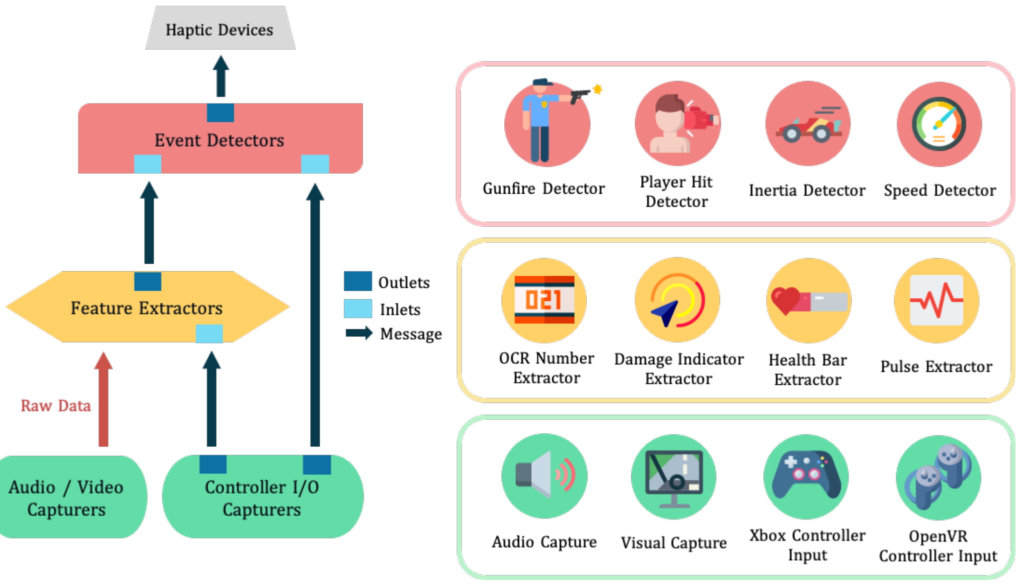


Figure 4: System Overview. (Icon credits: by authors and also from flaticon.com)

We surveyed four HUD types to categorize the display patterns and learn their frequency of occurrence. We chose the patterns with a higher appearance rate to be our feature source in the later detector implementation stage. The selected display patterns are listed as follows: 1) ammunition count with digit display, 2) health with health bar or image display, 3) damage indicator with 2D indicator display, and 4) speed with digit display. Additionally, we added in impulse pattern, which is captured from the audio channel, to be our fifth feature source. These chosen patterns are extracted from the visual and audio channels, which fit the black-box approach category.

3.3 Latency Threshold Requirement

We studied the latency thresholds for different feedback modalities to ensure that our system response times for each channel meet the requirements.
According to Attig [2], if system latency or system response time exceeds a certain threshold, users could notice the delay, therefore affecting the users' experience and satisfaction. The author indicated that a classic latency threshold of 100ms based on expert estimation and empirical data was made popular by Nielsen [18]. On the other hand, an overall latency threshold of 100ms for simple feedback and 200ms for complex feedback from recent latency guidelines was also concluded by the author. As for different feedback modalities, guidelines by Kaaresoja *et al.* [13] mentioned that visual, audio, and tactile feedback have individual latency threshold. The maximum threshold for visual feedback is 85ms, for audio feedback is 70ms, and tactile feedback is 50ms. If the feedback is bimodal (i.e., visual-audio, visual-tactile, tactile-audio), the single modality threshold will have slight changes, but the thresholds are mostly still in the boundary of 50-100ms.

4 SYSTEM DESIGN

We implemented a system whose components are mostly language-independent, loosely coupled, and encapsulated by utilizing a central message broker. As shown in Figure 4, our system architecture comprises three levels of components, raw data capturers, feature extractors, and event detectors. All of the message passing between components, except video and audio streams, is achieved by the message broker. Moreover, we implemented a file-based launcher for users to modify the components of HapticSeer before initiating the system.

4.1 Message Broker

A central message broker is an integrated part of most systems with centralized Publish–subscribe pattern [10]. HapticSeer needs a reliable and high-performance broker to support real-time messaging. After evaluating existing message broker solutions, we decided to utilize Remote Dictionary Server (Redis) as HapticSeer's central message broker, instead of making one by ourselves. Redis is the most popular open-sourced in-memory database that supports working as a message broker (Pub/Sub mode), and it possesses sub-millisecond latency and high throughput, making it the best candidate for our use case.

4.2 Components

A working instance of HapticSeer is composed of a user-defined combination of components, which are, in our case, implemented by C# running under .NET Framework. Every component in HapticSeer will publish their results by their outlets, and a component at higher abstract levels will receive their output by inlets connecting to their outlets. This encapsulation reduces the complexity of HapticSeer because haptic developers will only need to examine the I/O message format instead of figuring out what is precisely done in the components. After ensuring the dependency among components, haptic developers should be able to assemble a working instance of HapticSeer in a short time.

4.2.1 *Raw Data Capturers.*

Raw data capturers are components collecting, pulling, or listening to PC peripherals' signals/events and other data sources such as audio buses and video buffers. However, we found that some of the data streams are inapplicable of being transmitted through Redis Pub/Sub, e.g., uncompressed video streams and Hi-Fi surround-sound tracks. Therefore, raw data capturers belong to this stage are mostly peripheral data listeners. On the other hand, through video and audio streams are inapplicable of being transmitted through Redis, we still implemented two types of raw data capturer (DirectX, WASAPI) transmitting through bypass channels.

4.2.2 *Feature Extractor.*

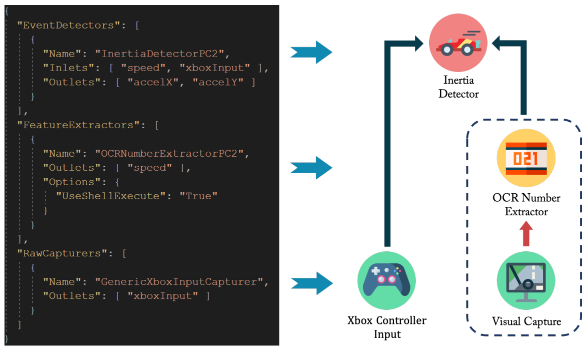
After raw data is retrieved from the data source, some raw data types have to be transformed into features for further processing. Since some extractors share the same data, they need to listen to the same data source; therefore, we categorized them as feature extractors for better system abstraction.

4.2.3 *Event Detectors.*

It is possible to detect certain game events (e.g., gunshots within a single magazine) by a single feature extractor in a controlled environment; however, reference a single feature to detect events can be inapplicable because of high false-positive error stemming from inconstancy through the gameplay. Moreover, some feature extractor could only produce non-semantic features such as audio pulse state, and certain events require additional raw data for inferring. Therefore, we need to parse extracted feature/raw data combination by predetermined rules (symbolic AI) or learning-based models.

4.2.4 *Launcher.*

We implemented a file-based launcher for quick deployment and configuring HapticSeer. Users could generate a configuration file with JavaScript Object Notation (JSON) format to assemble HapticSeer as ease. As shown in Figure 5, users only need to fill in sections of demanded components and set their inlets, outlets, and startup arguments, then the HapticSeer's launcher will automatically instantiate each component for the user.



```
{
  "EventDetectors": [
    {
      "Name": "InertiaDetectorPC2",
      "Inlets": [ "speed", "xboxInput" ],
      "Outlets": [ "accelX", "accelY" ]
    }
  ],
  "FeatureExtractors": [
    {
      "Name": "OCRNumberExtractorPC2",
      "Outlets": [ "speed" ],
      "Options": {
        "UseShellExecute": "True"
      }
    }
  ],
  "RawCapturers": [
    {
      "Name": "GenericXboxInputCapturer",
      "Outlets": [ "xboxInput" ]
    }
  ]
}
```

Figure 5: An example of the configuration file. (Icon credits: by authors and also from flaticon.com)

5 IMPLEMENTATION

HapticSeer proposes a framework that enables users to integrate their haptic devices with commercial games through black-box approaches. In this paper, we demonstrate the feasibility of using user-acquirable information to detect game events. In this section, we introduce the methods of capturing data and analyze the reliability of each channel. According to a survey conducted by Statista [20], PC is the most popular target platform for developers, and more than 66% of game developers are currently developing games for PC. To maximize the coverage of game content, HapticSeer runs on Microsoft Windows 10, which is the most common operating system in Steam Hardware and Software Survey [25]. To further clarify the system properties of HapticSeer, we designed several experiments in this section. All tests were conducted on a 16-Core computer with 32GB DDR4 memory, and the operating system and drivers were upgraded to the latest version. Besides, self-designed programs were all built by Visual Studio 2019, and all of the external programs were the latest stable released version.

5.1 DirectX Capturer

DirectX is a collection of graphics rendering APIs on Microsoft platforms and is commonly used in commercial games. Another famous graphics rendering API for rendering graphics is Vulkan, which provides cross-platform APIs for developers. HapticSeer's DirectX capturer supports every application running under Windows operating systems, not only those implemented with DirectX API.

5.1.1 *Capture Method.*

We use Microsoft DirectX Graphics Infrastructure (DXGI) to capture the screen. DXGI is designed to manage low-level tasks and communicate with the kernel mode driver and system hardware. In DXGI 1.2, it provides the Desktop Duplication API, which allows developers to fetch the whole monitor screen with low latency. Moreover, frames captured by DXGI Desktop Duplication API will not contain the mouse pointer; therefore, we can get pure game frames without any interference from it.

5.1.2 *Capture Latency.*

The capture latency is the elapsed time from when each frame is produced to when it is ready to be processed. DXGI provides a simple way to capture the whole screen with low latency. To measure the latency of capturing, we developed a console application that keeps printing the current time, whose resolution is a millisecond. The time shown in the console would be seen as a rendered timestamp. When the screen capturer received a new frame, we would record the received timestamp, and the new frame also contained a rendered timestamp. The calculation of capture latency is subtracting those two timestamps. The resolution to latency is one millisecond. Our experiment collected 48 samples, and the average latency is 13.6 ms (SD = 6.312), while 87.5% of latencies are below 17 ms.

5.2 WASAPI Capturer

Playing audio streams in Microsoft Windows is quite complicated. Microsoft provides several APIs for developers to perform audio operations, such as recording and sound mixing.

5.2.1 Capture Method. In Windows 10, there is a loopback adapter provided by Windows Audio Session API (WASAPI). It allows developers to record computer playback with low latency and capture the surrounding sound from the audio endpoint buffer. We chose WASAPI to capture audio streams from audio endpoint devices. However, the audio streams WASAPI provides are a chunk of audio, so there is a time window in each capture. In other words, the capture latency includes buffer-filling time and transmission time. Besides, because the capture method WASAPI provides is to fetch data from audio endpoints' buffer, we can only get a mixed audio stream rather than pure audio streams from each application.

5.2.2 Time Window. The time window is used to describe the fixed interval of time when the data stream is processed for query and mining purposes [9]. In HapticSeer, the time window only appears in audio channels. WASAPI provides a method that records the audio loopback stream, and the time window latency depends on the size of the buffer. In HapticSeer, the time window is 10 milliseconds.

5.2.3 Capture Latency. We used Audacity to test the latency of recording audio streams via WASAPI. Audacity is an open-source audio editing and recording application, and it provides an easy way to measure the latency of recording. To simulate the audio capturer's latency of our system, we set all parameters we used in HapticSeer. Expressly, the buffer size was set to 10 milliseconds, the latency compensation was set to zero, and the sample rate was set to 32bit/48000Hz. According to our measurement, the latency, including a time window, is about 16 milliseconds for a 5.1 surrounding sound device.

5.3 Controller I/O Capturers

This paper focuses on Xbox One Controller and off-the-shelf VR Controller, including HTC Vive Controller, HTC Cosmos Controller, and Valve Index Controller.

5.3.1 Capture Method. Microsoft provides an API, XInput, that allows applications to communicate with Xbox controllers. We can fetch controllers' information through XInput, such as a thumbstick's position, the proportion of a trigger pressed, and button presses. HapticSeer fetches the status once per millisecond(1000 Hz), which is relatively faster than the frame update rate. However, due to the lack of function in XInput that can query vibration information, it is impossible to get any vibration information. The only way to get vibration information is to hook XInput APIs in games.

When it comes to VR games, OpenVR is a necessary SDK developed by Valve for supporting SteamVR. OpenVR provides a series of APIs that help developers to communicate with VR hardware devices. Developers can retrieve VR controllers and VR headsets' status, including their position and orientation, vibration events from games, and button status of controllers. The data updating rate of tracking object position and orientation is based on the tracking frequency of inside-out tracking or outside-in tracking. We developed an OpenVR-based application that fetched data once per ten milliseconds (100 Hz). We have tested our application on an inside-out tracking system, HTC Vive Cosmos, and an outside-in tracking system, Valve Index. We also tested the usability of adding new kinds of controllers to our application.

5.3.2 Capture Latency. Since the capture method of controllers I/O calls the native APIs that game developers use, the Controller I/O channel is the only channel that we might fetch data before games. The latency is caused by the time lag between a new controller status occurred and fetching controller status by our application. Thus, the latency depends on the frequency of calling APIs. We call XInput API to fetch Xbox Controller status at 1000Hz, so the latency for Xbox Controller was below one millisecond. For VR Controllers, the position and orientation are calculated by the tracking system, and the frequencies of the tracking system are limited to the mechanical design. To solve the problem, OpenVR uses Inertial Measurement Unit (IMU) sensors to estimate the position and the orientation. Because the estimation was not reliable enough, we called OpenVR API to fetch VR devices' status 100Hz to reduce our system's fault rate.

5.4 Feature Extractors

We implemented a variety of feature extractors for the user study and the evaluation. Though the number of implemented extractors is limited, they can apply to many games because of their fundamentality.

5.4.1 Optical Character Recognition Number Extractor. HUD information such as ammunition count and character status is a useful in-game feature. These data are exact words or numbers to human eyes but need additional procedures to be converted into meaningful text types for computers. To fetch semantic information from the HUD, we applied real-time OCR on video frames to obtain the desired data. The optical character recognition (OCR) engine we used is Tesseract OCR [23], an open-source project started by Hewlett-Packard Company and developed by Google in 2006.

Since most of the game HUD backgrounds are semi-transparent, any object moves behind the HUD can affect word-finding. Therefore, binarization and denoising are essential processes that boost our OCR result's accuracy. We used a fine-tuned pre-trained English/Numerical model to support numeric character detection for the speed extractor and bullet number extractor.

5.4.2 Health Bar Extractor. Health bars are used to visualize the protagonist's health status in-game, and the primary key of the health bar's design is to make players comprehend easily. Most commercial games indicate the protagonist's health status by varing in the health bar's area in HUD. Thus, we can calculate the proportion between the current bar's area and the full bar's area, and the fraction is regarded as the protagonist's health. In HapticSeer, we first remove the background of the health bar and then calculate health status via this approach. Although several shapes have been used in health bars' design, such as a classic horizontal bar in Battlefield 1 and three hearts in Half-Life: Alyx, the approach can still precisely calculate the protagonist's health.

5.4.3 Damage Indicator Extractor. From the very beginning of FPS games' history, damage indicators have been an integral part of these games. We could use directional damage indicators to fetch hit information from games. In our case, we implemented a damage indicator extractor for 2D indicators around the cross-hair. The damage indicator extraction involves these steps:

(1) Crop the incoming frame to leave pixels around the crosshair only.
(2) Filter out pixels that are not similar colors to predefined reference colors for indicators (mostly red).
(3) Adopt the pixel-based difference to find the emerging objects, such as damage indicators.
(4) Apply a binary threshold on the frame to eliminate noises.
(5) Sum all vectors from the middle of the screen to each residual pixel's position as the incoming direction and signal intensity, then send to extractors.

5.4.4 Pulse Extractor. Some shooter games are challenging to retrieve firing information from them, such as games without HUD. Moreover, sometimes developers have to fetch more detailed damage information (e.g., an explosion is occurring). Therefore, we implemented an adaptive audio pulse extractor with the formula proposed by K. Lopatka, J. Kotus & A. Czyzewski [28]. First, the extractor uses a low-pass filter ($f_c = 125Hz$) to filter out mid-to-high frequency signals if a standalone low-frequency effects (LFE) channel is not provided. It then detects whether a bass pulse just happened or not by the proposed formula

$$L = 20 \cdot \left(\sqrt{\frac{1}{N} \sum_{i=1}^{N} (x[n] \cdot L_{norm})^2} \right), where \qquad (1)$$
$$L_{norm} = 1,\ N = 48000Hz \cdot 0.01\ sec = 480$$

and an adaptive threshold,

$$T(i) = \begin{cases} L(0) + m, & \text{if i=0} \\ 1 - \alpha \cdot T(i-1) + \alpha \cdot (L(i) + m), & \text{if i>0} \end{cases} \qquad (2)$$

then sends the result to extractors. Moreover, the extractor could also fetch the event direction if the surround sound is enabled on the system running HapticSeer. Noteworthily, for those games whose LFE capability is well-implemented, such as Battlefield 1, only certain events may be rendered on the LFE channel. Therefore, developers could exploit the pulse extractor for retrieving these types of events precisely.

5.5 Event Detectors

HapticSeer supports a variety of detector models; however, we focused on symbolic AI models as a proof-of-concept. We implemented three types of symbolic AI models for the user study and the evaluation.

5.5.1 Gunfire Detector. We implemented two variants of detectors to detect player-triggered gun firing. The first variant listens to an instance of OCR number extractor to fetch in-game HUD providing bullet counts in players' weapons. However, if the detector references the OCR number extractor solely, it tends to misfire when the player switches weapons and when a random flashing occurs on the screen. Therefore, it also listens to an Xbox/OpenVR controller input capturer for verifying players' intentions. That is, it will only report a weapon firing while the player is holding the trigger. By doing so, it eliminates false-negative outputs most of the time.

On the other hand, not all games possess HUD proving bullet counts. Therefore, we implemented another variant gunfire detector relying on the bass pulse to detect gunfire. This variant of the gunfire detector also listens to an Xbox/OpenVR controller input capturer for verifying players' intention.

5.5.2 Player Hit Detector. Though the health bar extractor could estimate the value of generic health bars, health bars' operating logic varies from game to game. Moreover, for those games possessing damage indicators, the detector could listen to a damage indicator extractor to fetch incoming direction. Therefore, a hit detector should be tailored for games sharing the same logic. We implemented two variants of detectors to detect player hurt events. The first variant of the detector listens to health bar extractors only, while the other variant listens to both a health bar detector and a damage indicator extractor.

The first variant of the hit detector will report the message of being hit and the difference of health points (HP) while finding that the health bar extractor's value is decreasing. However, this approach tends to misfire when a random flashing occurs on the screen, too. Therefore, we suggest that developers enable damage indicators if their target game supports them. If the damage indicator is referenced too, the player hit detector will only report the player hit event while a damage indicator is emerging and an HP loss occurs at almost the same time (50ms).

5.5.3 Inertia Detector. We implemented an inertia detector for racing games. The detector fetches speedometer readings by the OCR number extractor to estimate the car's local longitudinal acceleration. Moreover, by listening to an Xbox controller input capturer, the detector could estimate the car's local lateral acceleration by speedometer reading and states of analog sticks on the game controller. However, because the car physics simulation is complex and requires multiple magic constants [17] [24] to calculate correctly, the detector only estimates cars' lateral motion instead of calculating actual values. We found that the estimated value of lateral acceleration tends to be too large in magnitude if calculated in this way because we neglect the side slip of the car. Therefore, we clamp the lateral acceleration value to [-20, 20] (2G) based on experience. On the other hand, depending on the speedometer types of target games, some extractors could only report the integer part of the displayed speed, which may affect the longitudinal acceleration estimation.

5.6 End-to-end System Correctness

To ensure that the target event was the only event of interest occurring, we located a scene with predictable conditions. Moreover, we selected a series of event detectors having 100% accuracy to provide the ground truth for events. Given these two conditions, the correctness could be verified.

5.6.1 Approach 1: Visual Channel Only. We conducted a test with the HUD-based gun-firing detector (shown in 7-A) and the popular VR shooter game, Half-Life: Alyx (HL:A). Its debug endpoint was used to fetch the ground truth of every shot fired by the tester. We loaded a testing map, activated an in-game cheat code for unlimited ammunition supply, and then emptied the magazine/chamber of three in-game firearms at the earliest for ten times (300/100/60 shots for each weapon). We found that HapticSeer could achieve 100% accuracy in this scenario, showing that the visual approach is reliable.

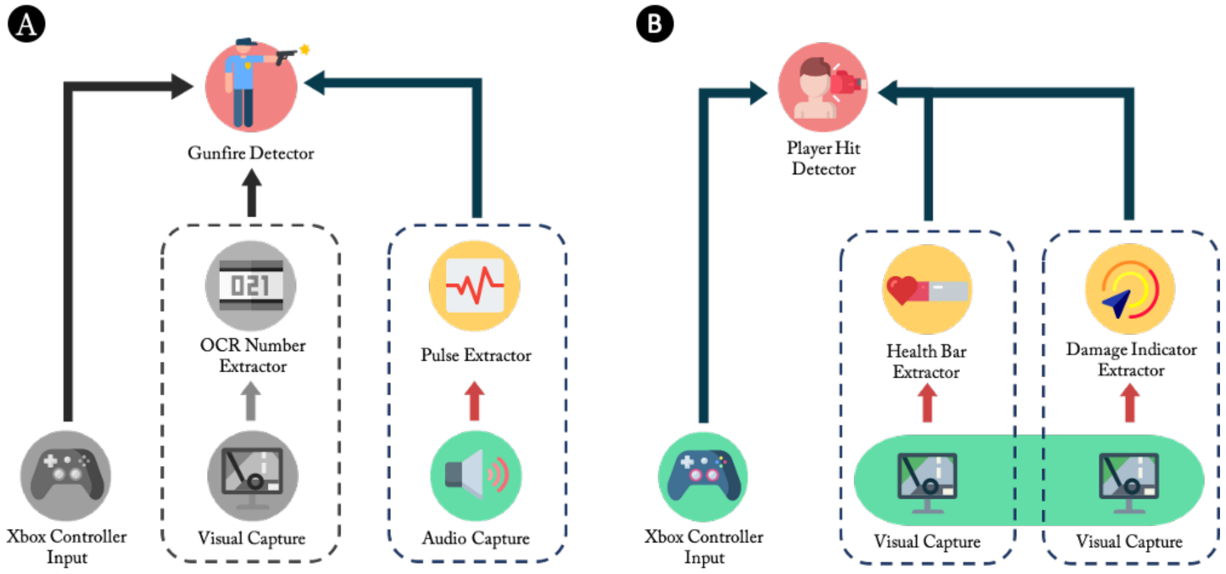


Figure 6: Structures of (A) gunfire detector and (B) player hit detector in Battlefield 1. Some components of A are disabled in the correctness evaluation. (Icon credits: by authors and also from flaticon.com)

5.6.2 Approach 2: Audio Channel Only. We conducted a similar test with the audio-based gun-firing detector and a popular first-person shooter game, Battlefield 1. As shown in Figure 6-A, the visual and controller I/O channels are disabled since we are only interested in the audio channel correctness for now. However, this target game does not provide an endpoint for retrieving game information. Therefore, we recorded the output and the game screen, then performed a manual check of accuracy. We loaded a chapter of the single-player campaign that forced the player to be a tank gunner, then continuously firing for five minutes (63 shots). We found that HapticSeer could also achieve 100% accuracy in this scenario, showing that the audio approach is reliable.

5.7 Detection Results

Unlike the correctness validation for the whole system, we performed an evaluation in natural settings to reach a better understanding of HapticSeer's capability while being used as a developer tool. We adopted the following two metrics to assess HapticSeer

- Sensitivity: the ability to correctly identify positive events.

$$\frac{\text{Number of correctly identified in-game events}}{\text{Number of all in-game events}}$$

- Precision: the ability to not generating false alarms.

$$\frac{\text{Number of correctly identified in-game events}}{\text{Number of reported in-game events}}$$

We tested our implemented detectors in natural settings with two games, Half-Life: Alyx and Project CARS 2 because these two games could provide the ground truth for automatic evaluations. For the first game, Half-Life: Alyx, we fetched the ground truth of player hit event and gun firing by the method mentioned above. Similarly, Project CARS 2 provides the user with a telemetry API to read memory data directly; thus, we exploited it to fetch the ground truth of cars' motion data.

Table 1: The detection result for case 1 (Half-Life: Alyx)

	True Positive	False Positive	False Negative	Sensitivity (Recall)	Precision
Gunfire Detector	442	17	32	0.932	0.963
Player Hit Detector	34	15	24	0.586	0.694

5.7.1 Case 1: Half-Life: Alyx. Unlike the minimal instance of HapticSeer adopted in correctness validation, we activated the OpenVR controller capturer to support the gunfire detector. Moreover, we also activated the health bar extractor and the player hit detector to get player-hit messages (shown in 7-A/B). For this evaluation, we chose a war-raging scene to get shot and open fire intensively. After three trials of gameplay (9.01 minutes in total, results are shown in Table 1), HapticSeer reached a sensitivity score of 0.932, and a precision score of 0.963 in the gunfire detection, being a comparable result to that from our system correctness validation. However, HapticSeer achieved unspectacular results in the player hit detection, which has a sensitivity score of 0.586 and a precision score of 0.694. The main reason may stem from the battlefield's chaotic nature, or a scene with too many objects sharing the same color with the health bar (yellow in this case).

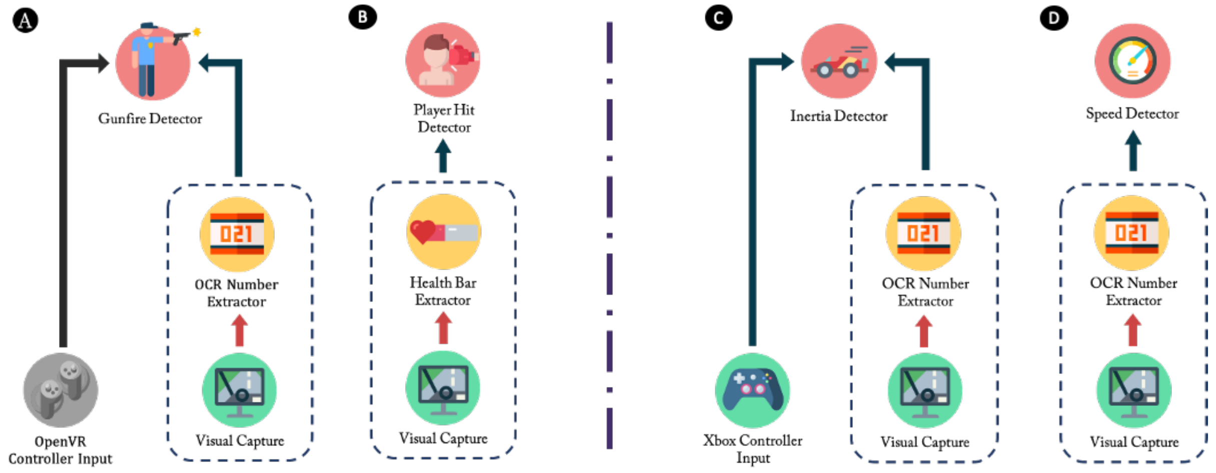


Figure 7: Structures of (A) gunfire detector and (B) player hit detector for Half-Life: Alyx, (C) inertia detector and (D) speed detector for Project CARS 2. Some components of A are disabled in the correctness evaluation. (Icon credits: by authors and also from flaticon.com)

We concluded that

- A more sophisticated feature extractor may be needed for preprocessing raw data better.
- A further investigation of testing scenes should be performed in advance when considering using HapticSeer as a developer tool.

5.7.2 Case 2: Project CARS 2. Project CARS 2 is a racing simulator developed by Slightly Mad Studios. It is famous for its realistic car physics simulation and VR support. We activated an Xbox controller input capturer, an OCR number extractor, and an inertia detector (shown in Figure 7-C/D) to get the player's local acceleration. We chose Greenwood Karting Circuit as the testbed and drove a standard two-stroke go-kart because of its mechanical simplicity. The gameplay result for six laps (5 minutes 42 seconds in total) is listed in Figure 8. We find that lateral acceleration's actual value is mostly within the range of [-20, 20], matching our assumptions. However, we also find that unable to get the actual value of current speed does affect the estimation of longitudinal acceleration.

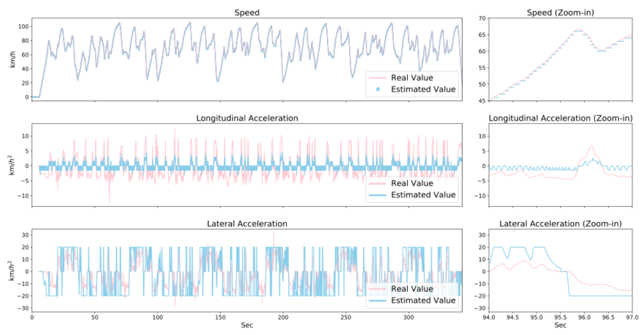


Figure 8: Outputs from Speed Detector and Inertia Detector (Longitudinal, Lateral)

We conclude that

- The inertia detector can estimate cars' motion, but it is incapable of reporting actual values.
- The resolution of extractable features plays an essential role in inferring events depending on them.

5.8 System Latency

For haptic devices, latency is an important issue that might harm the immersion of experiences. To measure the internal latencies in HapticSeer, we used a high-resolution timer provided by Microsoft in C# that could record elapsed time in microseconds.

5.8.1 Feature Extractor Latency. Feature extraction is a stage that computes data from capture channels. Thus the feature extraction latency contains the computation time. To measure the latency, we recorded capture timestamps and the timestamps that the computation is completed, and we could calculate feature extraction latency by subtracting these two timestamps. We found that the latency to be 0.24-5.92 ms on average for all extractors, consisting of a relatively small portion of the end-to-end latency.

5.8.2 Transmission Latency in Redis. After the computations of feature extractors are completed, the results will be transmitted to the event detectors via Redis. We tested the Redis server instance's latency by its integrated monitoring tool while playing Battlefield 1 with maxed-out settings. We found that the average latency to be 0.04ms for 311930 samples, and the maximum latency value in samples is 1ms, showing that the transmission latency in Redis is almost negligible.

5.8.3 Event Detection Latency. In the detection stage, one or more features would be collected for calculation. For example, calculating the centripetal force magnitude requires the velocity and the slip angle of vehicles. To measure the latency, we recorded the timestamp that received messages from Redis and the timestamp that the detector sent results via Redis so that the latency would

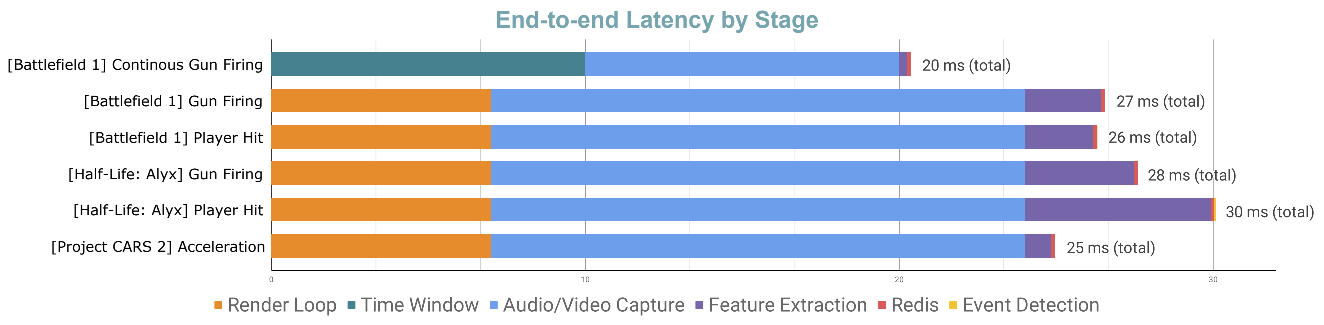


Figure 9: Latencies for each stage of HapticSeer

be the difference between timestamps. We found that the average latency to be 0.004-0.05 ms was too short to measure precisely. It is because example detectors we made were all symbolic AI models, which only consist of predefined rules. If learning-based models were involved in this stage to solve complex tasks, such as object detection with YOLOv4 [5], the latency in this stage will be slightly higher but still fast enough for real-time detection.

5.8.4 End-to-end Latency. In our dataflow, when an event occurs, it needs to wait for the next frame to render the event, and the latency is the length of the render loop, depending on the channel's refresh rate. Our study adopted a 144Hz monitor, so the length of the render loop for the visual approach is about 7ms. After the data is rendered, the capturer might wait for the buffer to be filled, which is the time window for the audio approach, as mentioned previously. Then, acquiring data from channels is accompanied with capture latency. The communication between capturer and extractor exists in Redis's transmission latency, and the extract latency depends on the computation time of features. The more complex feature requires more computation time. Subsequently, features will be transmitted to event detectors via Redis, and the event will be parsed eventually in event detectors. Finally, haptic developers will receive events via Redis, and the haptic devices will start to actuate. The end-to-end latency shown in Figure 9 consists of all latencies mentioned above, and the latency is about 20-30ms depending on which event to parse.

6 DEVELOPER FEEDBACK

We conducted a study to investigate our system's usability and its capability of integrating haptic devices with commercial games. We recruited four participants who are familiar with developing haptic devices to use HapticSeer. The participants have to assemble HapticSeer from scratch with provided components, then were asked to integrate one haptic device familiar to them, which are either off-the-shelf products or self-made devices.

We asked for the age, gender, occupation in project teams, and previous haptic works in the pre-experiment form. At the beginning of the study, we briefly introduced the architecture of HapticSeer and demonstrated all supported events to each participant. The participant would then receive a tutorial for HapticSeer. After that, we started to record the participants' feedback and started the timer.

We recorded the elapsed time taken by participants to complete the integration for quantifying the efficiency of utilizing our system. Moreover, the participants were asked to fill a questionnaire after the study. The questionnaire covers four topics of usability question about HapticSeer, which consists of the quantitative part and qualitative part for each topic. All of the questions could be found in Table 2.

6.1 Apparatus

The study's hardware setup consists of a gaming PC, which is the same as we used in the latency test (A 16-Core computer with 32GB DDR4 memory), and four haptic devices provided by each participant.

The participants were provided with three target games, Half-Life: Alyx, Project CARS2, and Battlefield 1. We provided real-world settings for the participants to test for each game, but we did not force them to test in such environments.

6.2 Participants

Four novice researchers, three males, and one female, ages 24, 23, 23, and 23, were recruited for the study. Three participants have one year of haptic research experience, while one person has three years of experience. Three of the participants have previously published haptic device papers at ACM CHI and SIGGRAPH conferences. All participants have developed three haptic devices, except one participant has developed only one haptic device.

The first participant chose to integrate bHaptics TACTOT [4] with the player hit detector and the gun firing detector for Half-Life: Alyx. TACTOL is a commercial haptic device that contains amounts of vibration motors to create directional haptic feedback. The participant spent 1 hour 40 minutes carrying through this study.

Then the second participant chose to integrate bHaptics TACTAL [4] with the player hit detector for Half-Life: Alyx. TACTAL is a face cushion attached to several VR headsets such as HTC Vive Pro. It contains several vibration motors to reproduces the feelings of being head-shot or punched around their eye sockets for players. The participant spent 45 minutes carrying through this study.

When the first two participants chose to use commercial devices, the third participant brought a VR headset with air propulsion jets attached to it to create directional force feedback. The participant

Table 2: A table for each topic of the questionnaire

Topics	Quantitative Parts (7-point Likert Scale)	Qualitative Parts
Easiness to Learn	"I agree that the framework is easy to learn."	"Why is (not) the framework easy to learn?"
Effectiveness	"I agree that the system makes it easier integrating haptic devices with existing commercial games."	"Why the system makes it (not) easier integrating haptic devices with existing commercial games?"
Latency	"I agree that this system's latency is low enough to support haptic feedback."	"Why I think this system's latency (not) low enough to support haptic feedback?"
Satisfaction	"I would like to use it for integrating haptic devices in the future."	"Why would (not) I like to use it for integrating haptic devices in the future" "What are the pros of the framework?" "What are the cons of the framework?"

chose to integrate the device with the inertia detector for Project CARS 2. The participant spent 1 hour 45 minutes carrying through this study.

Lastly, the fourth participants brought a self-made device which could provide high-frequency force feedback on hand. The participant chose to integrate his device with the gun firing detector for Half-Life: Alyx. The participant spent 45m carrying through this study.

6.3 Feedback Summary

6.3.1 Time to Complete. The average time for development with HapticSeer is 1 hour and 13 minutes, which is a fairly short time to develop a demo application. This result showed that HapticSeer could reduce efforts to develop demo applications if the required components are available.

The most experienced participant in .NET framework deployment spent the least time on development (45 minutes). He said, "This system would enable me to do rapid prototyping with commercial games in haptic research." The participant with the most experience on haptic research also spent a short time (45 minutes and 39 seconds) performing the integration.

However, the participant who had never developed a .NET application spent the most time (1 hour 45 minutes) developing. She stated that the unfamiliar ecosystem burdened her. "But I believe that after getting familiar with HapticSeer, I can connect my device to commercial games in a short time, less than a day. That would save me much time." she further explained.

6.3.2 Easiness to Learn. HapticSeer scored an average score of 4.75 on this topic. A participant claimed that she got familiar with HapticSeer after the first trial, but the document is hard to follow; three participants also complained about the document.

According to their feedback, we believe that by refining tutorials and redesigning a more user-friendly interface, HapticSeer could be easier to use.

6.3.3 Effectiveness. HapticSeer scored an average score of 6.25 on the easiness of integration, indicating that HapticSeer could make the integration of haptic devices and commercial games easier in these cases. One participant indicated, "It is useful for me to integrate because it eliminates the need for modifying the game." A participant also said, "This project could make some research-grade haptic device more useful because they lack applications." However, two participants suggested we design a graphical user interface (GUI) for HapticSeer, and two participants also doubt about the versatility of HapticSeer. A participant also questioned the dependency issue among deprecated components in the future; the participant also indicated that he might not use HapticSeer if there are not available detectors meeting his requirement. Finally, two participants also indicated the problem of false alarm.

Still, more user studies are required for future improvements, but we believe that by refining tutorials and redesigning a more user-friendly interface, HapticSeer could be easier to learn; therefore, it could grow as an ecosystem and expand its versatility.

6.3.4 Latency. As for the latency, HapticSeer scored an average score of 6.75, indicating that HapticSeer is capable of real-time event detection. All participants reported no perceivable latency, but one showed curiosity about whether the latency will become too high when his device needs long preparation time.

6.3.5 Satisfaction. HapticSeer scored an average score of 6.25 on satisfactory, indicating that the participants would like to use HapticSeer in the future, if available. Though supported games and available components are limited now, the participant still showed interesting in HapticSeer.

7 DISCUSSION AND LIMITATIONS

7.1 Scripted Experience

We have shown the feasibility of extracting game events from commercial games to integrate haptic devices. With HapticSeer, haptic developers can easily integrate their works to commercial games. However, there are some works unsuitable for HapticSeer. Haptic devices with longer charge time might be unsuitable because they need to predict events in demo scenes. Some haptic devices require long response time are also unsuitable because the latency of haptic feedback will be too high. Scripted experiences are required for these devices to prepare haptic feedback.

7.2 Fine-grained Haptics

Currently, we support two haptic categories, impact and motion. While some devices are not suitable due to their mechanisms, some devices are not compatible with HapticSeer because of the haptic feedback they try to simulate. For example, tactile events such as the weights of game objects are unobservable through black-box approaches, and the environment temperature and humidity cannot be extracted directly from games. HapticSeer's feature extractors may need more sensory information input to extract fine-grained haptic events, then inference by detectors.

7.3 Console Game

According to the survey conducted by Statista [20], PC is the most interested platform for developers. However, several games are only available for console platforms. To support console games, we need to find different ways to capture each channel's data from consoles such as PlayStation 4. It is possible to capture screen frames via capture cards, such as AVerMedia Live Gamer 4K - GC573. We have tested the latency of capturing PlayStation 4 screen frames with AVerMedia Live Gamer 4K - GC573, and the latency is about 40 milliseconds. Several off-the-shelf audio recording devices could record the audio streams from Sony/Philips Digital Interface Format (S/PDIF), the standard audio interface of PlayStation 4. When it comes to the controller I/O, it is more complicated than other channels. There are test pins on DualShock 4 [1], official controllers for PlayStation 4, so it is possible to parse controller status from Universal Synchronous Asynchronous Receiver Transmitter (UART) pins.

8 FUTURE WORK

8.1 Graphical User Interface

During our user study, several users reported that they prefer a graphical user interface (GUI) for assembling HapticSeer. The procedure of assembling HapticSeer is close to the dataflow programming; a GUI like Pure Data Environment[19] will make HapticSeer system easier to learn. A GUI with deliberately designed constraints can also help prevent users from situations such as mismatching the components or changing the system files by accident.

8.2 Open Platform

HapticSeer's modular design makes its components interchangeable. Therefore, we would like to build a toolkit to help users build and share their components at ease, and let communications between developers be more active and direct. Developers can contribute their high-accuracy detectors and high-speed extractors to HapticSeer. As the components grow larger and diverse, the limitation of supporting tactile events might be resolved.

8.3 Custom Controller Input & Analysis

Currently, HapticSeer only supports standard OpenVR input. However, to support fine-grained haptics, which needs precise skeleton data and additional sensory inputs, we have to design a corresponding API capable of processing sophisticated input messages and redirecting them to applications. For example, controllers with special functions, such as SHAPIO[14] and haptic gloves, need corresponding game systems to operate properly. If custom controller I/O can be processed by HapticSeer, unique controllers will be able to integrate with various games with fewer limits to the game systems.

9 ACKNOWLEDGEMENTS

This work is supported by the Ministry of Science and Technology of Taiwan (MOST 109-2218-E-011-011, MOST 108-2628-E-002-006, MOST 108-2218-E-011-027, MOST 107-2923-E-002-007) and National Taiwan University. We thank participants and reviewers for their feedback, especially during the COVID-19 pandemic.

REFERENCES

[1] Anonymous. 2020. DS4-BT - PS4 Developer Wiki. https://www.psdevwiki.com/ps4/DS4-BT#UART_HCI
[2] Christiane Attig, Nadine Rauh, Thomas Franke, and Josef F. Krems. 2017. System Latency Guidelines Then and Now – Is Zero Latency Really Considered Necessary?. In *Engineering Psychology and Cognitive Ergonomics: Cognition and Design*, Don Harris (Ed.). Springer International Publishing, Cham, 3–14.
[3] bHaptics. 2020. bHaptics Audio-to-Haptic. https://www.bhaptics.com/support/faq/#Audio-to-Haptic
[4] bHaptics. 2020. bHaptics tactsuit. https://www.bhaptics.com/tactsuit/
[5] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:cs.CV/2004.10934
[6] Mike Channell. 2020. The 20 best driving games of the last decade: 10-1. https://www.topgear.com/car-news/electric/20-best-driving-games-last-decade-10-1.
[7] Mike Channell. 2020. The 20 best driving games of the last decade: 20-11. https://www.topgear.com/car-news/electric/20-best-driving-games-last-decade-20-11.
[8] Daniel K.Y. Chen, Jean-Baptiste Chossat, and Peter B. Shull. 2019. HaptiVec: Presenting Haptic Feedback Vectors in Handheld Controllers Using Embedded Tactile Pin Arrays. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3290605.3300401
[9] Alfredo Cuzzocrea. 2009. Synopsis data structures for representing, querying, and mining data streams. In *Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends*. IGI Global, Pennsylvania, USA, 701–715.
[10] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. 2003. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.* 35, 2 (June 2003), 114–131. https://doi.org/10.1145/857076.857078
[11] ForceTubeVR. 2020. ForceTubeVR Documentation. https://www.protubevr.com/img/cms/ForceTubeVR_Documentation.pdf
[12] Samuel Horti GamesRadar Staff. 2020. The 25 best FPS games of all time. https://www.gamesradar.com/best-fps-games/.
[13] Topi Kaaresoja, Stephen Brewster, and Vuokko Lantz. 2014. Towards the Temporally Perfect Virtual Button: Touch-Feedback Simultaneity and Perceived Quality in Mobile Touchscreen Press Interactions. *ACM Trans. Appl. Percept.* 11, 2, Article 9 (June 2014), 25 pages. https://doi.org/10.1145/2611387
[14] Hayato Kajiyama, Akifumi Inoue, and Tohru Hoshi. 2015. SHAPIO: Shape I/O Controller for Video Games. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '15)*. Association for Computing Machinery, New York, NY, USA, 565–570. https://doi.org/10.1145/2793107.2810318
[15] Shi-Hong Liu, Pai-Chien Yen, Yi-Hsuan Mao, Yu-Hsin Lin, Erick Chandra, and Mike Y. Chen. 2020. HeadBlaster: A Wearable Approach to Simulating Motion Perception Using Head-Mounted Air Propulsion Jets. *ACM Trans. Graph.* 39, 4, Article 84 (jul 2020), 12 pages. https://doi.org/10.1145/3386569.3392482
[16] Logitech. 2015. Logitech G920 & G29 Driving Force Steering Wheels & Pedals. https://www.logitechg.com/en-us/products/driving/driving-force-racing-wheel.html
[17] G Mirone. 2010. Multi-body elastic simulation of a go-kart: Correlation between frame stiffness and dynamic performance. *International Journal of Automotive Technology* 11, 4 (2010), 461–469.
[18] Jakob Nielsen. 1993. *Usability engineering*. Morgan Kaufmann an imprint of Academic Press, a Harcourt Science and Technology Company, Burlington, Massachusetts.
[19] Miller Puckette. 1996. Pure Data. In *Proceedings of the second intercollege computer music concerts*. Self-published, Tachikawa, Japan, 37–41.
[20] Felix Richter. 2019. The Most Important Gaming Platforms in 2019. https://www.statista.com/chart/4527/game-developers-platform-preferences/

[21] SimRacingStudio. 2020. Sim Racing Studio. https://www.simracingstudio.com/
[22] Freepik Company S.L. 2020. Flaticon, the largest database of free vector icons. https://www.flaticon.com/
[23] R. Smith. 2007. An Overview of the Tesseract OCR Engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2. IEEE, Parana, Brazil, 629–633. https://doi.org/10.1109/ICDAR.2007.4376991
[24] Monstrous Software. 2003. Car Physics for Games. https://asawicki.info/Mirror/CarPhysicsforGames/CarPhysicsforGames.html
[25] Valve. 2020. Steam Hardware and Software Survey. https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam/
[26] Xbox. 2016. Xbox Wireless Controller. https://www.xbox.com/en-US/accessories/controllers/xbox-wireless-controller
[27] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST '09)*. Association for Computing Machinery, New York, NY, USA, 183–192. https://doi.org/10.1145/1622176.1622213
[28] Kuba Łopatka, Józef Kotus, and Andrzej Czyżewski. 2015. Detection, classification and localization of acoustic events in the presence of background noise for acoustic surveillance of hazardous situations. *Multimedia Tools and Applications* 429 (12 2015). https://doi.org/10.1007/s11042-015-3105-4

A ICON CREDITS

Icons from flaticon.com[22] are used in the following figures:

- Fig. 1. Icons made by smalllikeart, iconixar and Freepik
- Fig. 3. Icons made by Good Ware, itim2101, Smashicons and Freepik
- Fig. 4. Icons made by smalllikeart, Nikita Golubev, Pixel perfect and Freepik
- Fig. 5. Icons made by smalllikeart and Freepik
- Fig. 6. Icons made by smalllikeart, Nikita Golubev, Pixel perfect and Freepik
- Fig. 7. Icons made by smalllikeart, Nikita Golubev, Pixel perfect and Freepik