HCI Experiments Inside Environmental Narrative Games: Expanding the REVEAL Framework

Johannes Schirm*

Sheffield Hallam University

ABSTRACT

Using existing games to conduct experiments about human-computer interaction makes them not only more fun and natural, but in the case of presence measurement in virtual reality also likely to be more effective. As a contribution to current research in this area, the environmental narrative game framework "REVEAL" was expanded by an experiment system which allows researchers to implement experiment scenarios without wider knowledge of the framework's implementation, while still collecting all necessary data.

Index Terms: Human-centered computing—Human computer interaction (HCI)—HCI design and evaluation methods—User studies; Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Virtual reality

1 INTRODUCTION

"REVEAL" is the name of a project which aims to advance research and development of educational environmental narrative games in virtual reality. Funded by the European Union through the Horizon 2020 programme, the Steel Minions game studio at Sheffield Hallam University creates and commercially releases two full games, while implementing the underlying system in a more general way to make the creation of environmental narrative games easier in the future. Fig. 1 shows two environments from the first commercial game that was developed in the context of this project for Sony's PlayStation VR, using the REVEAL framework.

When selecting an appropriate locomotion technique for the PlayStation VR, which is a stationary virtual reality setup, a novel approach was found to combine the advantages of teleportation and free movement. For this, a graph of nodes has to be placed in the virtual environment. The user can navigate between adjacent nodes by step-wise rotating their base position, looking at the target node's footstep icon and pressing a button on the controller. But besides higher scores in user ratings and improved user performance, the authors found their data indicating that "rapid movement [between the pre-defined nodes] in very short bursts (<300ms) doesn't produce any greater feelings of motion sickness than teleportation," which is "not an intuitive finding and one which deserves a greater level of focus within the literature" [4, p. 6]. Still, they could not demonstrate "any benefits over teleportation in terms of user's feeling of presence," which would be logical due to the much more natural trajectory through the virtual environment.

Would users feel more present in a virtual environment that they explore while rapidly moving to adjacent nodes than in one where they get instantly teleported to adjacent nodes? This is an interesting research question, since current virtual reality games either completely avoid locomotion or implement locomotion systems without rapid movement [5]. Designing an experiment which will answer this question is not an easy task, as no model of presence and therefore no way to measure it is widely accepted yet. But it is clear that

*e-mail:johannes.schirm@student.shu.ac.uk

Figure 1: "The Chantry" is the first of two educational virtual reality games that are being created as part of the REVEAL project.

richer environments give users more to engage with and provide better conditions for experiencing high levels of presence. With the first full game already built on top of it, the REVEAL framework provides a convenient opportunity to reuse this game's assets for the creation of an engaging experiment scenario that feels and plays like a game, but has all necessary mechanisms for data recording integrated. An ideal starting point for experiments that could eventually answer research questions like the one above, for example. Unfortunately, the framework is not sufficiently flexible for this, as it requires knowledge of most of its implementation code.

Instead of directly implementing one experiment that could fail to answer the above research question, the framework was expanded by an experiment system which provides general means for implementing a variety of experiments from the field of human-computer interaction. This will make the process of creating the actual experiment much more agile and allow other researchers to answer similar questions in the future. However, the original research question will serve as a reference for evaluation, since it requires complex measurements to be taken and is likely to represent a large subset of similar questions. Because of this, the research background is shortly summarised at the beginning of this paper to describe the main requirements concerning the experiment system.

2 PRESENCE RESEARCH

Measuring the dependence of presence on the employed locomotion technique is just the first of many relevant research questions that could be asked when thinking about how to improve the user's experience in stationary virtual reality setups, especially with regards to the novel locomotion technique presented by Habgood et al. [4]. What is the most appropriate speed at which the rapid movement should take place? Does this depend on user preference and is there a way to determine it more precisely? How far should the locomotion nodes be apart? Following Mohler et al. [6], how closely would they need to match the exact eye height of the user in which scenario? Does rapid motion, and therefore continuous movement through the virtual environment, support the learning of its spatial layout?

All of these are interesting questions, but the general concept of presence seems to have the most potential to holistically evaluate features like locomotion techniques. Presence in virtual environments has already been subject to research for several decades, but there is still no general agreement about which model most accurately reflects this multifaceted concept. While there is considerable support of a variety of subjective presence measures in the literature [2, 12, 15], continuous efforts have been made to measure presence more objectively by either acquiring physiological sensory data or evaluating the naturalness of user behaviour [3, 8, 11].

The large body of literature on this topic can roughly be split into three different categories, all of which are referenced by two fairly recent examples in the following list:

- Theoretical publications on modelling the concept [1, 14]
- Studies which examine specific aspects of the concept [7, 9]
- Studies which include presence as a dependent variable [3, 13]

All of these can be valuable sources when searching for ways to measure presence as part of an experiment, but it can also be difficult to convincingly decide for a specific approach with so many alternatives. Skarbez et al. [10] recently presented a comprehensive literature survey, in which they analyse existing approaches and give helpful advice on creating an appropriate experimental design when dealing with presence: They recommend to choose one post-questionnaire that fits the scenario and combine it with at least one other measure, ideally with a behavioural one, since physiological measures have proven to only reliably evaluate presence in specific scenarios [10, pp. 96:31–96:32].

3 Experimental Design

For the experiment described here, the Witmer and Singer *Presence Questionnaire* and *Immersive Tendencies Questionnaire* [15] will be used to assess participants' presence after they experienced the virtual environment. This will be complemented by the single-item measure by Bouchard et al. [2], which participants answer in real-time. Finally, embedded in the short "story" played by participants, behavioural measures will be taken during key events.

Behavioural measures will include unexpectedly encountering a close-by ghost, which would lead to the well-known startle reflex for most individuals, and at some point also seeing an object at the ceiling (probably a chandelier) swinging towards the participant with no immediate way out. The last measure is expected to at least temporarily break the participant's sense of presence, since it will become apparent that the object does not pose a threat to them and can easily intersect with their view, but the degree to which they make an effort to doge it before realising this is expected to be another strong indicator of presence.

4 Requirement Analysis

This section aims to define the requirements that should be met by a versatile experiment system in the given context.

- Habgood et al. [4] compared three of the most commonly implemented locomotion techniques in commercial virtual reality games, which were still considered to be relevant for further research at the beginning of the project. Only one of them was developed further after their study—creating the need to reintegrate the others into the REVEAL framework.
- The head-mounted display (HMD) and the tracked controller provide a precise source of tracking data which is vital for evaluating real-time events like the participant's reaction on the behavioural measures.
- An important user interface relevant for data collection (and directly required for Bouchard's [2] real-time measure) is the microphone of the PlayStation VR.
- General user performance data is necessary to set possible anomalies in the main measures into context.
- Sometimes, the game logic would need to change depending on the participant number in order to assign participants to groups with different conditions.
- A dynamic way of recording "high-level conditions" from the game logic is crucial for a smooth analysis.
- With many dependencies from the main REVEAL system, enabling and disabling different data recording functionalities is extremely error-prone, which slows development down and makes data recording less reliable. Some sort of intermediate interface is needed for easy customisation.

5 Implementation

The predominant effort during the implementation phase consisted of understanding, restructuring and expanding parts of the REVEAL framework. Many very specific problems had to be solved which cannot be described in this paper at length. Therefore, this section documents the main contributions in a more synoptic form.

5.1 Source Code

The REVEAL framework is data-driven, so that most of its behaviour can be controlled through configuration files in the JSON format. A number of convenient "system commands" are available to the designer, allowing to execute pre-defined behaviour with custom parameters. There are command blocks for defining sequences of system commands, general configuration options for specific functionalities and a narrative graph which keeps track of the player's progress in the game, forming the heart of the game logic. All command blocks are interpreted at the start and converted to an appropriate system event which can be "fired" without additional overhead during runtime. Any object instance which registered an event handler will receive events once per frame.

General behaviour is defined through a mechanism similar to a finite state machine: In parallel to a global game state, there is always one other active game state which defines tasks that need to be done regularly and any specific application behaviour on an abstract level, for example how to transition from and to the main menu. The virtual camera, tracking matrices and some interaction logic is managed by the VR player class. One locomotion controller at a time possesses the VR player and updates it according to controller input. All integral entities in the system define an update handler which receives the elapsed time since the last update and the current controller state, including pressed buttons and joystick states.

First steps of the implementation phase included modifying the main menu to allow the input of a participant number, creating an experiment game state, designing a dedicated experiment manager and getting accustomed to the build process. This required a significant time of immersion in the existing source code, which had constantly been growing since the start of the project. Code duplication and coupling made it sometimes difficult to fully understand sections of code in the beginning, which reinforced the need of an appropriate interface for researchers. Fortunately, some of the required features had already been implemented in the past, and it often took less effort to reconcile old source code to the current structure of the system than to implement them from scratch. Although many steps are required when introducing a new system command to the framework, this interface proved to be ideal for controlling the experiment system at runtime from a designer's point of view.

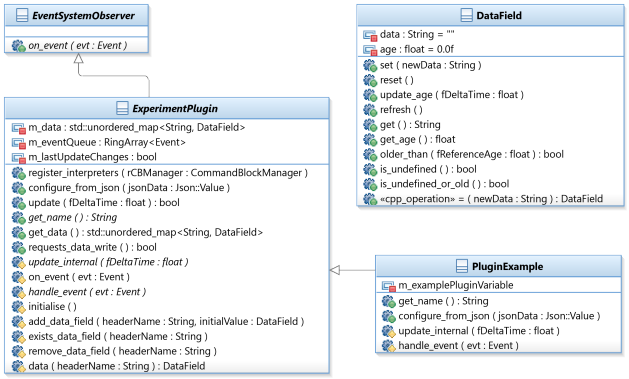

Figure 2: This UML class diagram shows the basic structure of the C++ plug-in system that was added to REVEAL.

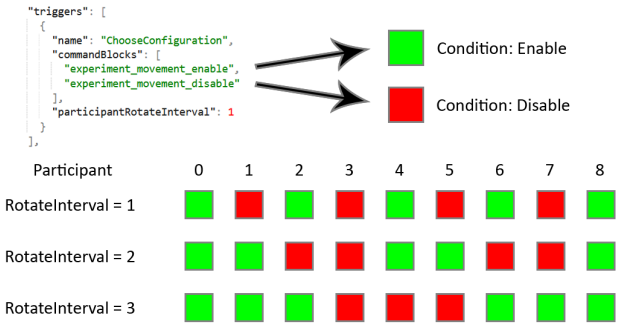

Figure 3: Visualising the experiment trigger system using the example of two groups of participants experiencing different game logic.

5.2 Experiment Manager

This unit is responsible for the general flow of experiments, writing output files and managing its subsystems. All collected data is cumulatively written to one comma-separated values file, whose columns can be defined through the experiment manager's JSON configuration file. Except for the very last line, a new line is only appended to the output file if the value of at least one column was updated. Updates of several column values in the same frame will still result in just one line. This simple system keeps the file compact, readable and easily processable in any statistics software. At the beginning of each line, there is always the participant number and elapsed time in seconds since the experiment started.

The system commands **start_experiment**, **end_experiment** and **abort_experiment** allow to control when the experiment, and therefore data collection should be active in the game.

5.3 Conditions

The first subsystem in the experiment manager are conditions: They offer a way of defining an output variable by name and updating it later using the system commands **set_experiment_condition** or **increment_experiment_condition**. It is possible to define default values and, if this value is a numeric JSON property, increment it using the second system command. This makes it possible to track arbitrary variables, for example the current experiment phase, block number or amount of solved tasks.

One important aspect of conditions is that they always have a valid value, which is why they are written with every new line. Although changing the value of a condition will result in a new line being added to the output file, their concept is more of a passive nature.

5.4 Plug-ins

Also part of the experiment manager is the much more active plug-in system. It allows to add C++ plug-ins which are only concerned with one specific aspect of data collection. Each specialisation of the abstract plug-in class has a well-defined set of functions to override in order to compile as part of the system: In addition to implementing update and event handlers, they need to provide a unique name, reset functionality, (optional) JSON parameter parsing and (optional) registration of any custom system commands. Through their base class, these specialisations can easily register any output variables by name and manage their values through the string-based data field interface. Fig. 2 gives an overview of the involved classes.

Plug-ins have to behave more actively than conditions because for a new line in the output file, their data field values are considered "old" (and as a result *undefined*) if they have not been updated in the same frame. This means that if one plug-in data field value or one condition changes, the subsequently written line contains *undefined* values for all other plug-in data fields with "old" values. An exception to this are data fields which have been constructed with the "always up-to-date flag" set to true, essentially simulating the behaviour of a condition for some special cases.

The following example shows how this can lead to readable and easily processable output: A plug-in recording the locomotion behaviour of participants stores the name of reached locomotion nodes and the world-space distance travelled from the previous node. While the travelled distance must only be recorded once, it would be convenient in the analysis to also know for every subsequent line which node participants were on. This makes sense, since participants always have to be on some node, making the variable eventually act like a condition. By constructing the name data field with the "always up-to-date flag" set to true, one can always write the field and remove the need for post-processing output files.

To activate a plug-in, its class definition has to be added to the rv::Experiment namespace, its header included in the "Experiment-Plugins" header file and its name added to the JSON configuration file of the experiment manager, along with any parameters.

5.5 Trigger

The last subsystem allows the definition of experiment triggers. They are used to diverge game logic according to the participant number under which the experiment is currently running. Each trigger has a unique name, a list of user-defined command blocks and an interval defining the mapping from participant numbers onto the command blocks. Calling the system command **experiment_trigger** with the trigger's name as a parameter will execute the appropriate command block. Fig. 3 shows an example in which there are two possible command blocks to call. With an interval of 1, the chosen command block will alternate for every participant, with an interval of 2, it will alternate for groups of two participants and so forth.

5.6 Locomotion Controller

When planning the project, it was not immediately clear whether an evaluation of different locomotion techniques was to be part of the experiment. For this reason, and because the associated source code was highly coupled with the rest of the system, the controller system was reengineered to the point where locomotion techniques could be switched via the system command **switch_controller** and toggle rapid movement for node-based techniques via **set_controller_movement**. But since there is no need to switch controllers in the current context, most of the actual controller implementations do not work correctly. However, having created a dedicated controller manager, it should be simple to reimplement them and even add new controllers in the future.

Figure 4: The optional preparation phase gives participants time to settle in and test their controller in a neutral environment.

5.7 Preparation Phase

When executing the system command **start_controller_check**, the experiment game state temporarily disables input in the controller manager and plays back a fixed sequence of controller tests. During these, participants will hear an instructor telling them to press a specific combination of buttons. Once the correct input was detected, they advance to the next stage. This will prepare them better for experiments in virtual reality, especially if they are not used to the technology yet. An optional parameter to the system command allows a command block to be executed as "callback" afterwards.

5.8 Audio Recording

Recording the participant's voice was made possible with the system commands **start_audio_recording** and **stop_audio_recording**. The experiment manager does not record audio automatically, but it will pause and resume recording the input from the HMD's microphone according to which of the two commands is executed. When the experiment ended or was aborted by the experimenter, the recorded audio is written next to the output file in WAV format.

6 Evaluation

Whether the experiment system fulfils all requirements stated in the requirement analysis was tested by realising the experiment described at the beginning of this paper with its help. In a sense, the evaluation of the experiment system could be considered an extension of the implementation. This section will demonstrate how easy it now is to integrate custom functionality into the REVEAL framework and then collect any analyse all necessary data.

Five plug-ins were implemented in total, four of which perform general data collection from specific parts of the system. The "locomotion" plug-in continuously records the name of the locomotion node the participant stands on. When a transition is performed, it also records the travelled distance once. The "controller" plug-in records the current type of locomotion controller and whether rapid movement is enabled or disabled, both continuously. The "activity" plug-in measures the participant's physical activity while an experiment is active by accumulating two variables: The total distance travelled by the HMD's position and the total distance travelled by the intersection of the participant's view vector and a unit sphere. For proper analysis, these values can be reset and written to the output file together with a named marker by using the **issue_activity_marker** system command. The "voice" plug-in continuously records whether audio recording is active or not.

To supplement these general variables, the "HMD" plug-in allows to record the HMD's tracking-space matrix in real-time. Using the system commands **start_hmd_recording** and **stop_hmd_recording**, the recording can be controlled by user-defined command blocks.

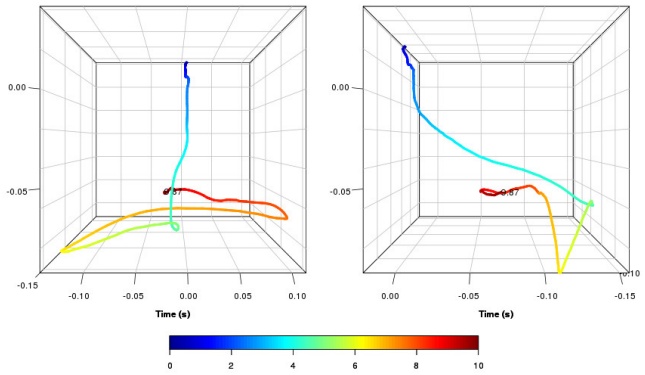

Figure 5: This is a visualisation of a participant's head position over the time of 10 seconds. The left plot views onto the participant's back, the right plot views onto their right side. Time is colour-coded from blue to red. Apparently, the participant examined something more closely after 4 seconds and decided to also view it from the left and right afterwards. Note that the actual range of movement is only between 5 and 15 centimetres, so it could be assumed that they probably did not need to take a step forward for this.

Optionally, a sample delay in seconds can be provided as parameter. This allows for sampling at rates less than or equal to the game's frame rate. Fig. 5 shows how a visualisation of this data in a versatile statistics software like R could look like. With consistent column naming, the matrix can be elegantly recomposed in R. The following code and the R library *plot3D* is everything one needs in order to produce this visualisation from an unmodified output file:

```
# Read the last output file in the working directory:
files <- list.files(pattern = "*.csv")
data <- read.csv(files[length(files)], sep = "\t")

# Extract the matrices of the first 10 recorded seconds
    together with the relative elapsed time:
headerNames <- append("elapsedTime", paste("HMDMatrixC",
    rep(0:3, each = 4), "R", 0:3, sep=""))
matrices <- na.omit(data[headerNames])
startTime <- min(matrices$elapsedTime)
matrices$elapsedTime <- matrices$elapsedTime - startTime
matrices <- subset(matrices, elapsedTime < 10)

# Remap the coordinate axes to plot3D:
x <- matrices$HMDMatrixC3R0
y <- matrices$HMDMatrixC3R2
z <- matrices$HMDMatrixC3R1

# Draw the view onto the back of the participant at the
    last frame, with the whole trajectory visible.
library("plot3D")
i <- nrow(matrices)
scatter3D(
  x[1:i], y[1:i], z[1:i], theta = 0, phi = 0,
  xlim = c(min(x), max(x)), xlab = "",
  ylim = c(min(y), max(y)), ylab = "",
  zlim = c(min(z), max(z)), zlab = "",
  colvar = matrices$elapsedTime[1:i],
  colkey = list(side = 1, length = 0.8),
  type = "l", clim = c(0, 10), clab = "Time (s)",
  ticktype = "detailed", bty = "b2", lwd = 4
)
time <- round(matrices$elapsedTime[i], digits = 2)
text3D(
  x[i], y[i], z[i], c(toString(time)),
  add = TRUE, bty = "b2"
)
```

As shown above, the output file is already optimised for being processed in a statistics software, which is why it does not take more than remapping the translations onto the coordinate system used by the *plot3D* library and (optional) remapping of the time, so that it starts at 0 seconds. Naturally, it would be very simple to generate animations with this code, calculate accumulated variables like the total travelled distance and maximum movement speed or plot acceleration over time. This should provide a good staring point for a meaningful analysis of behavioural measures.

7 Future Work

Because of the time constraints of this project, there are some details that would have certainly improved the experiment system:

- *Plug-ins*
  Recording the HMD's tracking-space matrix can be helpful for analysing user behaviour. But the PlayStation VR also tracks the controller, which would be equally helpful to record. Also, part of the recorded activity could be an integer representing how often the user rotated their base via the controller.

- *Plug-in interface*
  The plug-in base class already offers an intuitive interface, but additional means of managing data fields would be helpful. At least an elegant visitor function or even support for frequently-used types like vectors and matrices. This would save the time of manually iterating through columns and even offer fast access dependent on data type. For example, the matrix of the "HMD" plug-in has to be manually stored in 16 columns at the moment. Also, a general mechanism for reducing the frequency of update calls would be helpful. This has manually been implemented in the "HMD" and "activity" plug-ins.

- *Controller System*
  This is not an immediate part of the experiment system, but it has the most potential for improvement. First, the extreme code duplication should be removed by redesigning the class hierarchy: Just as an idea, introduce another abstract controller class "InteractionController" for cases like the pause controller. The abstract "LocomotionController" also inherits from this class and is used for most controller specialisations. Also, a "LocoNodeController" is put between this abstact class and node-based controllers to encapsulate their similarities. Second, the implementation of the free controllers does not work any more and should be redone. Third, with one of the free controllers selected, the controller manager could try to map the user's position back to an appropriate locomotion node, so they still move on the locomotion graph and trigger according events, without jumping over nodes. And fourth, switching controllers should result in a smooth transition between free and node-based controllers. The base position and rotation would need to gently snap back into place.

8 Conclusion

In summary, the evaluation has been successful, since the experiment system provides all features needed to smoothly implement the reference experiment. The implementation section in this paper was not focussed on technical details, which there were far too many of anyway, but more on presenting the system to newcomers and giving a compact round-up of the project's environment. Furthermore, some approaches to statistical analysis of collected data were presented, which should help to keep the design process of the experiment agile. This work will hopefully prove useful in any future projects about similar topics as an expansion of the REVEAL framework that allows researchers to easily develop experiments in the same environments in which commercial games take place.

Acknowledgments

I want to thank Dr. Jacob Habgood for the supervision of this project and the interesting opportunity to work with the PlayStation VR in the Steel Minions game studio. I also want to thank Dr. David Moore for the technical supervision of this project and frequent support in the challenging environment of PlayStation (VR) development. Last but not least, I want to thank my friendly colleague Andrew Hamilton for helping me with the REVEAL framework and PlayStation (VR) development whenever I was facing problems.

References

[1] Y. Bian, C. Yang, F. Gao, H. Li, S. Zhou, H. Li, X. Sun, and X. Meng. A framework for physiological indicators of flow in VR games: construction and preliminary evaluation. *Personal and Ubiquitous Computing*, 20(5):821–832, oct 2016. doi: 10.1007/s00779-016-0953-5

[2] S. Bouchard, G. Robillard, J. St-Jacques, S. Dumoulin, M. J. Patry, and P. Renaud. Reliability and validity of a single-item measure of presence in vr. In *Proceedings. Second International Conference on Creating, Connecting and Collaborating through Computing*, pp. 59–61, 2004. doi: 10.1109/HAVE.2004.1391882

[3] C. Deniaud, V. Honnet, B. Jeanne, and D. Mestre. An investigation into physiological responses in driving simulators: An objective measurement of presence. In *Science and Information Conference (SAI)*, pp. 739–748, 2015. doi: 10.1109/SAI.2015.7237225

[4] J. Habgood, D. Moore, D. Wilson, and S. Alapont. Rapid, continuous movement between nodes as an accessible virtual reality locomotion technique. In *IEEE Virtual Reality*. IEEE, 2018.

[5] J. Habgood, D. Wilson, D. Moore, and S. Alapont. HCI lessons from PlayStation VR. In *Extended Abstracts Publication of the Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '17 Extended Abstracts, pp. 125–135. ACM, New York, NY, USA, 2017. doi: 10.1145/3130859.3131437

[6] M. Leyrer, S. Linkenauger, H. H. Bülthoff, and B. J. Mohler. The importance of postural cues for determining eye height in immersive virtual reality. 10(5):e0127000, may 2015.

[7] E. Malbos, R. M. Rapee, and M. Kavakli. Behavioral presence test in threatening virtual environments. *Presence*, 21(3):268–280, August 2012. doi: 10.1162/PRES_a_00112

[8] M. Meehan, B. Insko, M. Whitton, and J. B. F. P. Physiological measures of presence in stressful virtual environments. *ACM Transactions on Graphics*, 21(3):645–652, July 2002. doi: 10.1145/566654.566630

[9] R. Skarbez, F. P. Brooks, and M. C. Whitton. Immersion and coherence in a visual cliff environment. In *IEEE Virtual Reality*, pp. 397–398, 2017. doi: 10.1109/VR.2017.7892344

[10] R. Skarbez, F. P. Brooks, and M. C. Whitton. A survey of presence and related concepts. *ACM Computing Surveys*, 50(6):1–39, November 2017. doi: 10.1145/3134301

[11] M. Slater. How colorful was your day? why questionnaires cannot assess presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 13(4):484–493, 2004.

[12] M. Slater, M. Usoh, and A. Steed. Depth of presence in virtual environments. *Presence: Teleoperators and Virtual Environments*, 3(2):130–144, 1994. doi: 10.1162/pres.1994.3.2.130

[13] F. Soyka, E. Kokkinara, M. Leyrer, H. Bülthoff, M. Slater, and B. Mohler. Turbulent motions cannot shake VR. In *IEEE Virtual Reality*, pp. 33–40. IEEE, 2015. doi: 10.1109/VR.2015.7223321

[14] K. Szczurowski and M. Smith. Measuring presence: Hypothetical quantitative framework. In *23rd International Conference on Virtual System and Multimedia*, pp. 1–8, 2017.

[15] B. G. Witmer and M. J. Singer. Measuring presence in virtual environments: A presence questionnaire. *Presence*, 7(3):225–240, June 1998. doi: 10.1162/105474698565686