Simultaneous Run-Time Measurement of Motion-to-Photon Latency and Latency Jitter

Jan-Philipp Stauffert* University of Würzburg
Florian Niebling† University of Würzburg
Marc Erich Latoschik‡ University of Würzburg

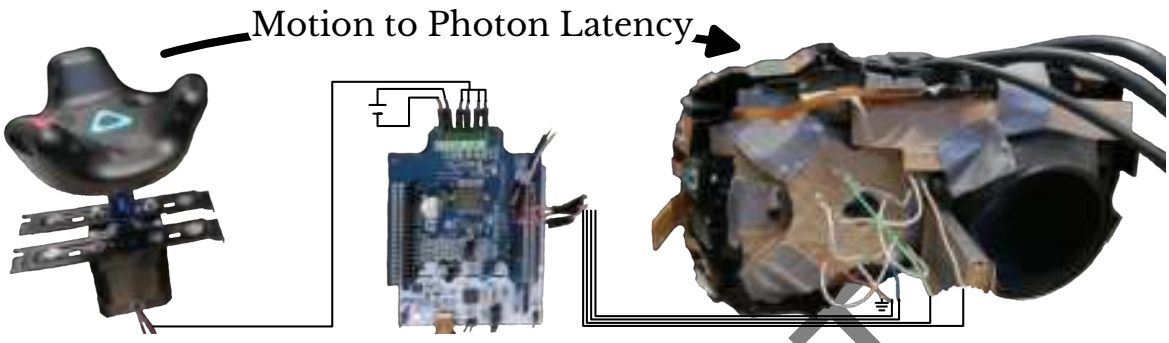

Figure 1: Illustration of the apparatus developed to measure the Motion-to-Photon latency. A microcontroller (middle) reads the difference between the rotation of a tracked controller as moved by a motor (left) and the reported rotation on the Vive display (right). The illustration shows the experimental prototype which disassembled the Head-Mounted Display for easier access to the internals, i.e., the lenses and displays.

ABSTRACT

Latency in Virtual Reality (VR) applications can have numerous detrimental effects, e.g., a hampered user experience, a reduced user performance, or the occurrence of cybersickness. In VR environments, latency usually is measured as Motion-to-Photon (MTP) latency and reported as a mean value. This mean is taken during some specific intervals of sample runs with the target system, often detached in significant aspects from the final target scenario, to provide the necessary boundary conditions for the measurements. Additionally, the reported mean value is agnostic to dynamic and spiking latency behavior. This paper introduces an apparatus that is capable of determining per-frame MTP latency to capture dynamic MTP latency and latency jitter in addition to the commonly reported mean values of latency. The approach is evaluated by measuring MTP latency of a VR simulation based on the Unreal engine and the HTC Vive as a typical consumer-grade Head-Mounted Display (HMD). In contrast to previous approaches, the system does not rely on the HMD to be fixed to an external apparatus, can be used to assess any simulation setup, and can be extended to continuously measure latency during run-time. We evaluate the accuracy of our apparatus by injecting a controlled artificial latency in a VR simulation. We show that latency jitter artifacts already occur without system load, potentially caused by the tracking of the specific HMD, and how mean latency and jitter increase under system load, leading to dropped frames and an overall degraded system performance. The presented system can be used to monitor latency and latency jitter as critical simulation characteristics necessary to report and control to avoid unwanted effects and detrimental system performance.

Index Terms: D.4.8 [Operating Systems]: Performance—Measurements; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities

*e-mail:jan-philipp.stauffert@uni-wuerzburg.de
†e-mail:florian.niebling@uni-wuerzburg.de
‡e-mail:marc.latoschik@uni-wuerzburg.de

1 INTRODUCTION

Each instruction of a computer system has an associated execution time. As a result, any output that is calculated after changing user input will always be delayed, introducing latency into the human-computer interaction loop. The impact this delay causes on the system qualities of usability and user experience depends on the interactivity and potential real-time requirements of the human-computer interface and the employed interaction metaphor [6]. Even users of a spreadsheet software in a 2D graphical user interface (GUI) expect a timely response to a mouse click. The response should feel to be instantaneous. The conditions to experience feedback as instantaneous depend on several factors of the interaction metaphor.

Direct interaction metaphors are more sensitive to delays than indirect ones. Here, VR systems are specifically sensitive to delays between input and output processing since they directly and continuously couple input, including head movements, to the visual output. The overall latency in VR between an action, e.g, moving an input controller measuring hand or head movements, and its corresponding effect shown on a screen is denoted as Motion-to-Photon (MTP) latency. Unwanted delays during the input-to-output processing not only risk to annoy users but they potentially might induce more severe consequences of visually-induced motion sickness (VIMS) and cybersickness [25]. Hence, a central requirement of VR systems is to measure and finally control a VR system's latency behavior to judge its performance before negative consequences arise.

Deducing the latency from code inspection and counting the execution times of all instructions of a VR application is largely out of reach today. The interplay of all soft- and hardware sub-systems of modern general-purpose computer systems introduce a dynamic complexity that can only be observed as a black box [28]. Low-level system interrupts of the system's motherboard, modern CPU speed-up approaches including parallelization, caching, and branch prediction, as well as high-level operating system and application-layer aspects of concurrency, multi-threading and scheduling, and

Table 1: Comparison of previous approaches to latency measurement. Camera based approaches are less intrusive but can't capture an object and an HMDs screen with the exception of augmented reality headsets [23]. Camera based approaches are not viable if the user wears an HMD as the view is obstructed. Photodiodes are more intrusive but provide high temporal resolution. Approaches that use photodiodes attached to the screen can allow to use a majority of the screen to display a VR experience. Only a small screen area needs to be reserved for the information that gets picked up by the sensor. Other approaches that use photodiodes have either only tested with a monitor screen instead of an HMD or require the HMD to follow specific movement patterns. Our approach is the only one that uses photodiodes with an HMD without the need of a predetermined HMD movement pattern. This potentially allows to measure latency while a user consumes a VR application.

Author	Method	Capture	Latency	Use HMD	HMD wearable	Measurement frequency	Temporal Resolution
Becher et al. [2]	Continuous	Photodiode	Mean, SD, Min, Max	yes	no	11 ms	Frame
Di Luca et al. [7]	Sine Fitting	Photodiode	Mean, SD	yes	no	20 Hz	Once
Friston et al. [11]	Event	Camera	Mean, SD, Min, Max	yes	no	Acceleration peak	Movement dependent
He et al. [14]	Event	Camera	Mean	no	no	Grid line crossed	Movement dependent
Kämäräinen et al. [17]	Event	Photodiode	Mean, SD	no	possible	16 ms - 100 ms	Once
Liang et al. [19]	Continuous	Camera	Mean	no	no	20 Hz	Frame
Mine [22]	Event	Photodiode	Mean	no	possible	Pendulum zero position	Movement dependent
Papadakis et al. [23]	Continuous	Photodiode	Mean, SD	no	possible	0.1 ms	0.1 ms
Sielhorst et al. [26]	Continuous	Camera	Distribution	AR	no	< 1 ms	< 1 ms
Steed [32]	Sine Fitting	Camera	Mean	no	no	25 Hz	Once
Wu et al. [35]	Continuous	Camera	Distribution	no	no	2 ms	2 ms
Zhao et al. [36]	Sine Fitting	Photodiode	Mean, SD	yes	no	11 ms	Once
Our approach	Continuous	Photodiode	Mean, SD, Distribution	yes	yes	11 ms	Frame

I/O communication (including networking) and the interplay of all these aspects induce a potential fluctuation in execution time. After all, real-time capabilities currently are not a central requirement for the typical consumer-grade general purpose computer systems. As a result, the overall MTP latency of VR systems is often reported as a mean value – if it is reported at all. However, each of the multiple hardware and software parts either contribute their own variable latency during run-time or the interplay of all parts creates a dynamic pattern of latencies not well represented by a mean value derived from sample runs.

Ideally, we would either be able to control and assure run-time behavior as provided by real-time systems, or to continuously monitor latency and latency jitter during execution of a VR system. For example, spikes in latency may influence and invalidate results during VR experiments since they induce a potential confound. The ability to detect latency spikes during experiments allows to sort out trials when the experimental condition is overshadowed by technical inconsistencies. Overall, monitoring latency in VR systems is a critical quality assurance measurement to optimize run-time behavior and to assess and guarantee good usability and user experience of VR systems.

Contribution

This paper introduces an apparatus that is capable of determining per-frame MTP latency to capture dynamic MTP latency and latency jitter in addition to the commonly reported mean values of latency. The approach is evaluated by measuring MTP latency of a VR simulation based on the Unreal engine and the HTC Vive as a typical consumer-grade Head-Mounted Display (HMD). We inject artificial latency in a VR simulation and show that latency jitter artifacts already occur without system load, potentially caused by the tracking of the specific HMD, and how mean latency and jitter increase under system load, leading to an overall degraded system performance. In contrast to previous approaches, the system does not rely on the HMD to be fixed to an external apparatus, can be used to assess any simulation setup, and can be extended to continuously measure latency during run-time.

2 Related Work

Visual delay was found as a major contributing factor already in early simulators [10]. Time invariant latency causes cybersickness [5], decreases performance [15] and reduces presence [21]. Reoccuring latency spikes also have negative effects on performance [24, 33] and cybersickness [31]. Interruptions in VR can cause a break in presence [27].

Users are able to distinguish changes in latency for hand [9] as well as for head [8] movements that are faster than 33 ms. Building on this work, Mania et. al. test sensitivity to head tracking latency in virtual environments [20] where they show that differences of 15 ms are still distinguishable. The currently common VR displays running on 90 Hz would exceed this detectable threshold if tracking information is even one frame delayed.

The performance of VR applications is usually assessed by measuring MTP latency which tracks the time between an input on a certain input channel and the time it takes to show its effect on a display. He et. al. [14] employ manual frame counting. They record a tracked controller's movement and its virtual counterpart at the same time with a high-speed camera. They count the time delay between movement discontinuities to infer the latency. Steed [32] replaces the error prone determination of discontinuities in the video with sine fitting. Steed attaches the tracked controller to a pendulum and fits the movement with a sine curve. The use of a continuous signal instead of detecting distinct events reduces the impact of inaccuracies due to limited temporal video resolution. Fitting a sine to the real controller's movement and its virtual image yields the MTP latency in the phase difference. The still manual video analysis is replaced by a direct deduction of the sine's gradient with photodiodes by Di Luca [7] or image processing in Friston and Steed [11]. Papadakis et al. [23] correlate a continuous movement of a tracked object with a photodiode reading which is attached to a monitor screen. An oscilloscope shows measurements of the system's latency. Becher et al. [2] use multiple photodiodes on the HMD screen to pick up the brightness encoded HMD orientation. A motor rotates the HMD to provide the base truth. The latency is derived from the difference

of the orientation in reality and the orientation reported on screen. Kämäräinen et al. [17] use a photodiode and a simulated touch event to measure latency in a remote rendering application, inducing further latency through additional non-local network communication. All approaches report one mean latency value and if the approach allows it a standard deviation.

Latency, however, changes with time and shows repeated spikes [7, 28]. This is a result of the complexity of VR systems that often consist of multiple software components to handle various input and output modalities that run in parallel or on distributed machines [1, 18, 29]. Regarding latency as time invariant allows to only measure it once and claim that it will be the same at a later time. Sielhorst et al. [26] describe the latency behavior of an augmented reality system with the distribution of measured latencies. Their measurements exhibit infrequent outliers. Wu et al. [35] propose a camera-based approach to measure latency at a 1 ms resolution again showing the time variability of latency. If latency is to be assumed to change during runtime of a system, it is neccessary to measure latency spikes during user studies that might be influenced by this jitter.

Latency measurement either focuses on the latency between certain events, or use sine fitting to correlate a known movement to the measurement data. If the measurement is precise enough and can be repeated often, a continuous approach is taken. The capturing of latency measurements is either done with a camera that records both the tracked object and the result on a screen, or with photodiodes attached to a screen that are correlated with a known movement. Camera based measuring is less invasive but is hard to use with HMDs, as their screen is difficult to capture. Photodiodes are more invasive and need additional hardware to support their usage, but can potentially report measurements during runtime of a system, and not only in post hoc analysis. A comparison of the reported approaches is presented in Table 1.

Research shows that latency changes over time and users of VR applications can detect small changes in latency, but measuring approaches are restricted to report mean values. We introduce a setup that allows to measure MTP latency for every frame, that can be extended to work during VR experiments. We use this to describe a VR system's latency under different conditions.

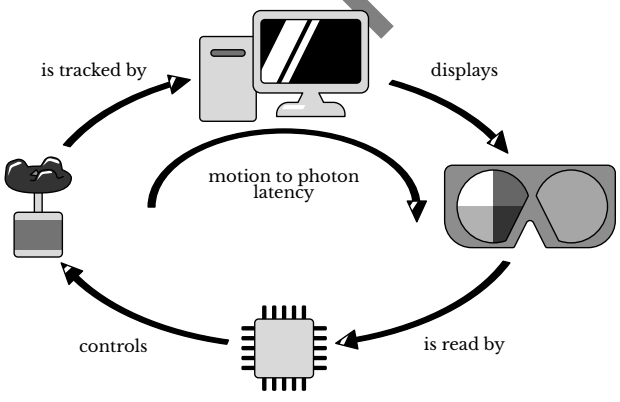

Figure 2: End-to-end latency in the setup: The top part shows MTP latency from a tracked controller to its representation on the HMD screen. The bottom shows the microcontroller comparing controller and display to calculate the MTP latency.

3 Setup

We measure the time between a known real-world rotation of a tracked controller, and the effect of said rotation on a HMD screen. A motor rotates a tracked controller with a known speed. The tracker sends the position and orientation of the controller to a computer. The computer then calculates the motor angle from the orientation. This detected angle is rendered to a VR HMD screen, encoded into rectangles with certain brightness levels. Each processing and communication adds to the final latency both by our own software and by the hard and software we are using. A microcontroller is employed to drive the motor and to read the HMD screen using photodiodes. It calculates the difference between the known motor angle and the angle reported on the screen. Knowing the speed of the motor, introduced latency between movement of the controller to display of the correct orientation on the HMD screen can be deduced. An overview is shown in Figure 2.

We use an HTC Vive tracker, first and second generation, which is mounted on a NEMA17-01 2 phase hybrid stepper motor. The tracker orientates itself with an IMU and the Lighthouse tracking system at 120 Hz [16]. The tracking data is sent to the connected computer. A VR application based on the Unreal Engine 4.22.3 receives the data through the SteamVR/OpenVR connection. The microcontroller is a NUCLEO L152RE developer board with ARM Cortex M3 processor, attached to a motor driver shield X-NUCLEO IHM01A1. Four OSRAM BPW 21 photodiodes are attached to the microcontroller and HMD to read back the orientation of the tracked controller encoded on the HMD screen.

The Unreal Engine collects the most recent tracker orientation every frame and provides it for the subsequent user logic. We receive the orientation in euler angle form and convert it to a quaternion to extract the motor angle. This extraction needs a calibration before the experiment. The tracker rotates multiple times around its axis. Due to the intermittent Euler angle representation, the orientations form a line in quaternion space. The quaternion at the beginning of the line q_0 is used to normalize all other rotations. It is characterized by having a negative distance to its predecessor, with the distance being calculated using the dot product between itself and its predecessor. The calculation is done multiple times to account for sampling errors. All potential quaternions for q_0 are collected in a set S.

$$S = \{q_t | \langle q_t, q_{t-1} \rangle < 0\}$$

where q_t is the tracker's orientation at time t. The quaternion q_0 is the element of S with minimal w component.

$$q_0 = \arg\min_{p \in S} p(w)$$

A multiplication of an orientation q_t of the tracked controller with the inverse of q_0 removes the base orientation, i.e. the slope the tracker stands on. This assumes that the motor and attachement provide a rotation without nutation. The normalized quaternion is denoted q_n.

$$q_n, t = q_t \cdot q_0^{-1}$$

The motor/tracker angle a_t is then computed by taking the inverse cosine of the w component.

$$\text{angle}_t = cos^{-1}(q_{n,t}(w))$$

The stepper motor moving the tracked controller rotates at 0.9° per step. Microstepping increases the motor vibrations that influence the IMU part of the tracking significantly, and was therefore not used. The motor is attached to a small slope to prevent gimbal lock. The calculated controller angle is converted to its stepper motor step equivalent and encoded on the HMD, by rendering a brightness pattern to a specific area of the HMD screen. Four photodiodes attach to the display area to read back the brightness values rendered

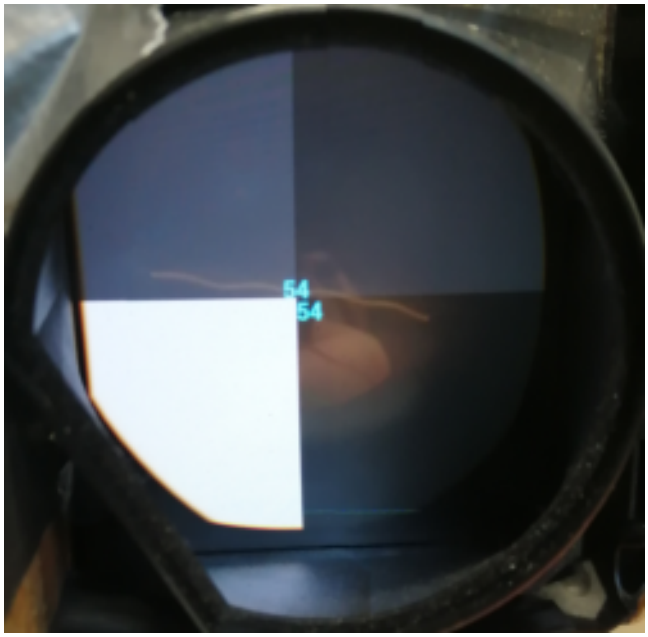

Figure 3: The Vive display with a number encoded as a brightness pattern. The lens is taken out, and the rectangles encoding the number are enlarged for a better view. The number displayed in the middle is for debugging only, as the photodiodes are attached at the border of the screen.

on the screen. The photodiodes are able to distinguish four different values consistently, leading to a total number of 256 (4^4) different values that can be encoded. The 400 possible steps per single rotation of the motor exceed the 256 distinguishable values and are therefore encoded modulo 200. Figure 3 shows an example number encoded on the display.

The Vive display is dark most of the time, with a light wave occuring every frame. The microprocessor reads the attached photodiodes at a frequency of 4 kHz, delivering approximately 44 brightness samples per frame for each photodiode, of which 16-17 are in the bright region of each frame. The bright region consists of a plateau of 6-7 samples. The remaining 10-11 samples describe the rising and diminishing brightness. Figure 4 shows an exemplar reading of the brightness levels of the various photodiodes during several frames of the HMD. The photodiodes have different dark levels due to variation in their attachment. The brightness reading rises once an image is shown. The absolute brightness units carry no meaning here, as the measurements are used for relative comparison between the different brightness levels, and are normalized in the figure. The computer performs a calibration with the microcontroller over its serial interface prior to the measurements, to ensure accurate and repeatable detection. The calibration phase starts to display a black HMD screen for the microcontroller to pick up which sensor values equal black. To be robust against small variations, the black threshold is set to be at 1.15 times the maximum measured black value. All readings above this threshold are then counted as belonging to a brightness level encoding a specific grey level of the display. The computer then presents each possible brightness combination to the microcontroller, while communicating which actual grey level is set over the serial interface. The microcontroller deduces for each sensor which measured brightness interval represents the respective number. The figure shows the four different brightness values used in our system, presented to one of the photodiodes each. Relatively large gaps between amplitudes of the different curves indicate that more intensity levels could be distinguished by the photodiodes. This

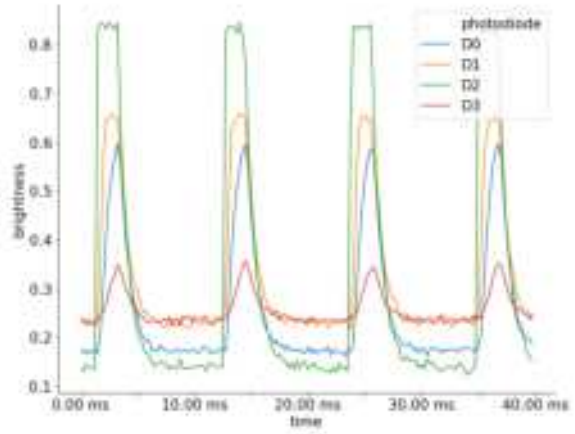

Figure 4: Brightness readings of the Vive display with four different brightness values shown as read by four photodiodes. The Vive display is black most of the time with a short burst of brightness to show the image. The maximum brightness represents the color shown on screen.

optimization was not followed up upon, to keep the measurements as reliable as possible. During measurement, the microcontroller saves the two highest brightness values for each phase, while the brightness reading is above the black threshold for each respective diode. The encoded number is deduced at the fourth reading below the black threshold by averaging the two highest values and comparing the result to the brightness intervals calibrated during system startup. We discard a frame's reading if at least one photodiode doesn't detect the light flank end within 2 ms of the other diodes to guard against erroneous readings.

The microcontroller drives the stepper motor with two revolutions per second. Using a motor with 400 steps per revolution, one step of the motor is occuring every $\frac{1}{800}\text{s} = 1.25\text{ms}$. This value that depends on the employed stepper motor is a limiting factor determining the maximum possible accuracy of the measurements in the following experiments.

4 Experimental Measurements

We use our setup to observe a computer systems's baseline MTP latency behavior and MTP latency behavior under load. We validate the system setup with an experiment: Known artificial latency jitter is introduced at the application stage of our VR system, to determine if the additional latency is visible in the final readings compared to an established baseline containing no additional latency.

4.1 Experiment 1: Baseline

The baseline measurement describes the MTP latency behavior of our system. Common VR applications are expected to show a similar latency behavior.

4.2 Experiment 2: Artificial Latency Jitter

We introduce artificial latency jitter to see if manipulation at the application stage is visible in the final measurement. The introduced jitter discards position and orientation updates of the tracker every 16th frame for the duration of two frames. The implementation follows Stauffert et al. [31] but exchanges the probability distributions with fixed numbers to achieve the desired effect.

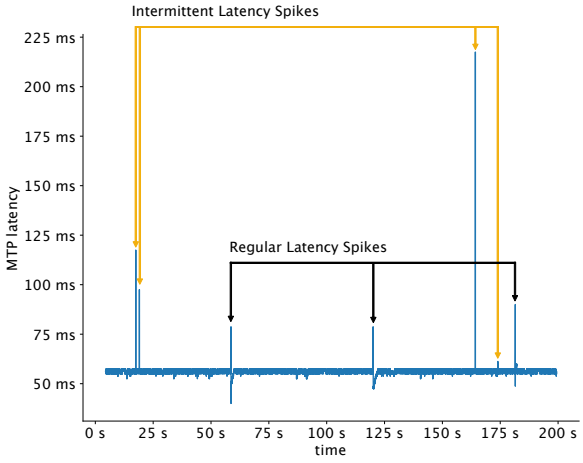

Figure 5: Difference in between the real tracker rotation and the reported tracker rotation on the HMD screen reported as delay in ms as deduced by the rotation speed of the tracker. The difference was measured for every frame of the HMD. Most latencies are close to the mean value with few latency spikes. There are regular reoccuring spikes and irregular spikes.

4.3 Experiment 3: Load Analysis

We stress the computer running the application "HeavyLoad" [13] that creates load on both the main processor and the graphics card. This leads to a CPU load of 100% on all cores, and a GPU load close to 100% causing other applications running concurrently to suffer.

We assume to encounter a fixed angular offset between the motor rotation and the motor rotation reported on screen, as long as the computer has sufficient free resources. Stressing the computer here with a background benchmark or in general with a demanding simulation should evoke similar spikes in angular difference as in the previous experiment, where latency was introduced in a predetermined period. The expected difference is a varying latency spike duration with varying occurrence.

5 Results

5.1 Baseline Measurements

Figure 5 shows parts of a test run. We discard the first four seconds of the motor accelerating. Most of the latency measurements gather around one value differing only by one stepper motor step difference equaling 1.25 ms. Repeated outliers to both sides represent disturbances in the tracking or processing resulting in latency jitter. We do not have any insight where the latency jitter originates. Examples of sources can be the tracking, different components like scheduling and background tasks, application stage like specifics in the employed engine, the display, or any stage in our setup.

The mean latency in the baseline test run was 56.14 ms with a standard deviation of 1.6 ms. There is no difference between using a Vive tracker of the first generation compared to the second generation version. The plots here show a test run with a tracker of the second generation.

We measure repeated latency spikes followed by a period of lower latency that degrades to the mean after few samples. Figure 6 shows a segment with two such patterns. These patterns occur approximately every 61.4 seconds. The measurement in Figure 5 features three occurrences with in between times of 61.443 ms and 61.432 ms. There are irregular outliers besides this periodic pattern.

5.2 Artificial Latency Jitter

Introducing artificial latency jitter every 16th frame for two frames shows the angular difference to increase in the first not updated frame and to increase further in the second not updated frame as was expected (cf. Figure 7). The smaller time differences visible in the graph are quantised to the measurement accuracy of 1.25 ms. The mean latency in the latency jitter test run was 58.20 ms with a standard deviation of 6.22 ms. The comparison plot in Figure 9 shows two bands representing the two introduced delays.

5.3 System Load

Measuring latency with the computer brought to its capacity shows many and irregular latency spikes. Compare Figure 8 for a visualisation. The load test had a mean of 54.51 ms with a standard deviation of 8.62 ms. In addition to the many outliers with increased latency, there are some latencies below the MTP latency mean.

5.4 Comparison

The histogram of latency measurements in Figure 11 shows the majority of the samples gathering around 55 ms latency as does the scatterplot of Figure 9. Frequent outliers are delayed only by a small amount of time, while a decreasing number of outliers exhibit higher latency. Note that the y scale of the figure is logarithmic to better show the distribution. The figure shows the result of a stacked-z test following Stauffert et al. [30]. It assumes that latencies are normally distributed and applies a z-test to detect outliers. The test is then recursively applied to the outliers again. A first outlier category to start at larger latencies indicate a bigger variation of the underlying distribution.

The baseline measurement describes a narrow normal distribution with outliers close to the mean. The outliers themselves have a small variation and therefore create multiple outlier groups that are each close to a mean. The samples around the mean contribute 88.1% of all the samples with the first outlier group containing 11.6%. The remaining 0.3%. The artificial latency condition shows the two introduced peaks in latency. Its distribution has a larger variation. The z-test separates the main values (76.1%) from the introduced latency jitter (23.9%). The load condition has a large variation which comprises 95.4% of the samples in the main part with 4.5% in the first outlier category. All conditions show rare extreme outliers.

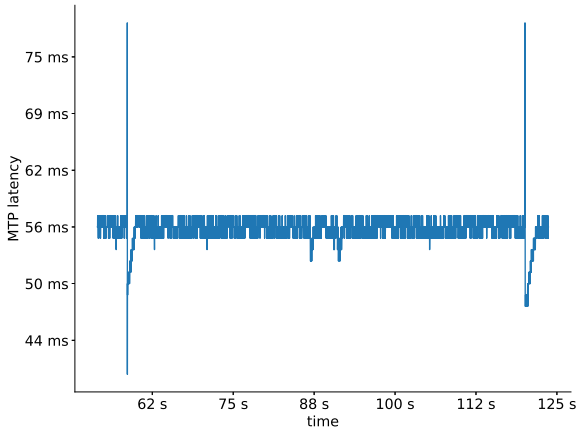

Figure 6: Detail of Figure 5 to show the repeated occuring pattern of latency spikes. We observe a regular pattern with a big spike in latency followed by lower latency that converges back to the mean latency.

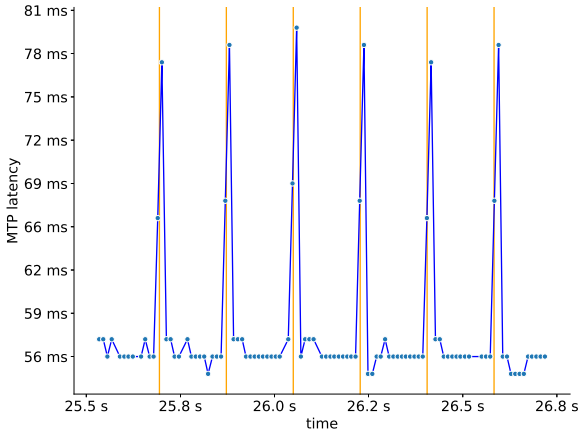

Figure 7: Measurements with artificial latency spikes: A spike, delaying tracking information for two frames, was introduced every 16th frame. The latency measurement returns the introduced pattern of one frame with an increased latency by 11 ms and the subsequent frame with an increased latency by 22 ms. The orange vertical lines mark the first delayed sample.

The quantile-quantile plot in Figure 10 shows how the distributions compare to one another. If a distribution is the same, its quantiles lie on a line as seen in the diagonal. All distributions have one or two big outliers at the 100% quantile. The lower quantiles are similar. The comparison between the baseline and the artificial latency jitter condition shows a similar distribution until the latency timing of the first spike. The second spike shifts the q-q plot further from the diagonal. The comparison between the artificial latency spike and the load condition show more percentage of the samples for the latency spike condition in the latencies of the provoked spikes and then more samples above those latencies in the load condition.

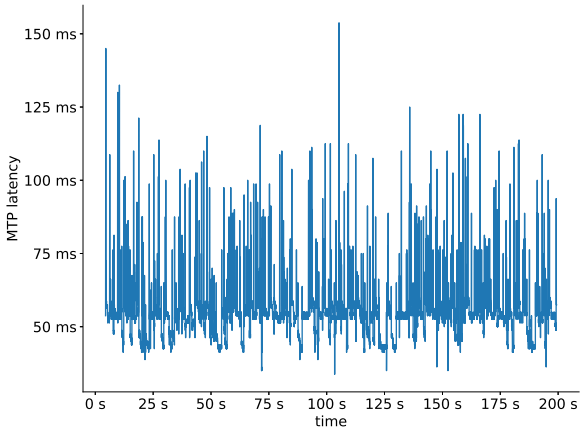

Figure 8: Measurements with artificial load: The processor and graphics card were stressed by a background software. Multiple irregular latency spikes result. The majority of samples still gather near the mean. Interesting is the increase in samples with latencies lower than the mean.

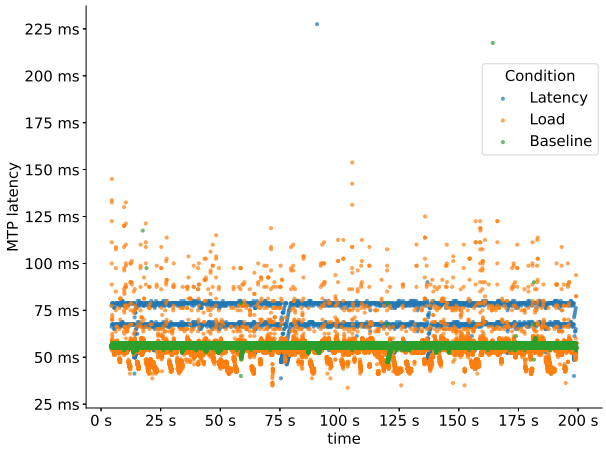

Figure 9: Scatterplot comparing the measurement runs with different conditions. The baseline run samples form a narrow band. The artificial latency jitter run samples show two additional bands that equal the two injected delays. The load condition shows multiple outliers. All conditions show some outliers.

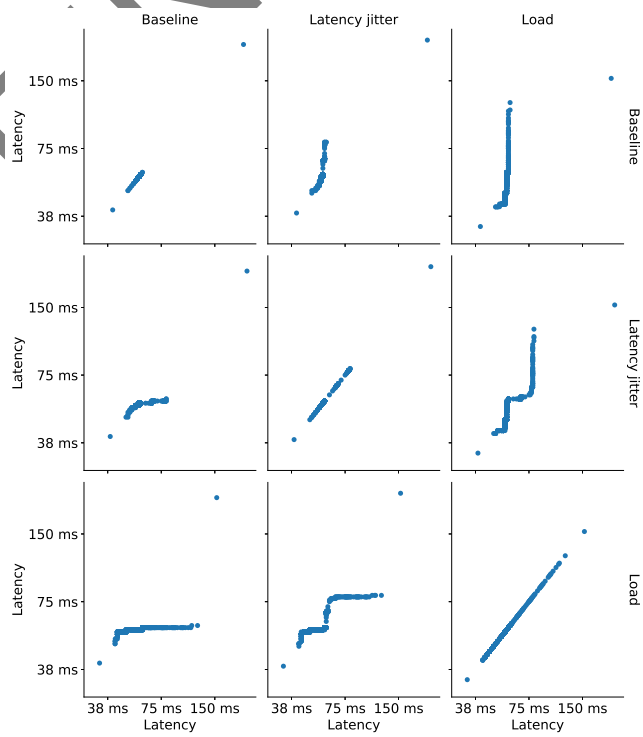

Figure 10: Quantile-quatile plot to compare the distribution of the different measurement runs. The distributions are similar, i.e., close to the diagonal, in their lower quantiles. The higher quantiles show the two introduced latency spikes in the latency jitter condition. The load condition has more samples in the higher quantiles. The few extreme outliers are clearly visible and independent of the condition.

6 Discussion

The evaluation of our approach validates that the injected artificial but controlled latency is properly reflected in the measured data. We

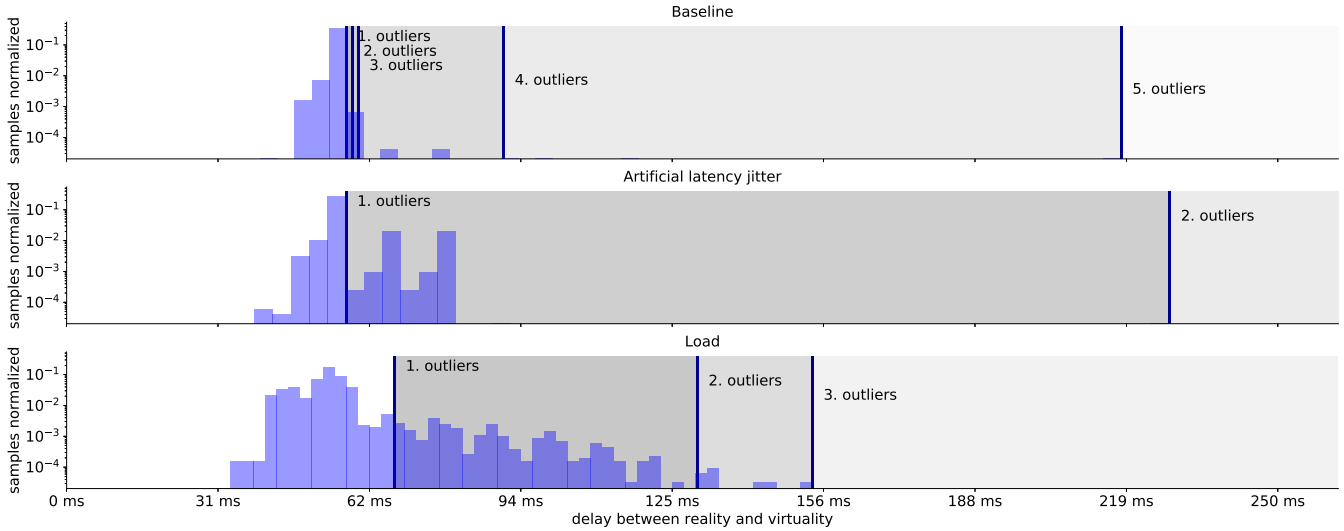

Figure 11: Illustration of the stacked z-test of the latency measurements for the three load conditions. Top: the baseline condition without any additional load. Middle: system behavior with injected latency jitter. Bottom: system behavior under load stress. Without any load, the majority of the latency measurements gather around a mean as a central tendency. The different latency distribution patterns depending on the load condition slowly move this mean to the right since the variance and hence the jitter increases. This mean does not accurately reflect the latency-related system behavior. The stacked-z test shows outliers of the assumed normal distribution and therefore encodes the variance as well. Notably, our approach detects that the no-load baseline condition is already affected by two groups of outliers which would result in repeated reoccurring lost frames not well represented by one mean value. Also, the injected latency (middle) is well detected and reported by our apparatus. The heavy-load condition at the bottom reports a system behavior which can be assumed to generate a severe negative impact on usability and user experience.

find that a normal test run carries both distinct patterns as shown in Figure 6 as well as less regular outliers. The stacked-z test works well to separate outliers. It separates the regular outlier pattern in the baseline run, separates the introduced latency spikes in the artificial latency jitter run and splits the majority of outliers from the rarer more extreme outliers in the load condition. A q-q plot proves to be a suitable tool to compare different behaviors. It allows to test the system with a baseline measurement and then compare what changes in more taxing situations.

A computation between motion and photon or an interaction of multiple parts takes longer and therefore misses one screen refresh to create the distinct pattern in Figure 6. This computation occurs in regular intervals. As the rotational difference between the real tracker and the reported rotation is below the mean difference after such a spike, we assume that the tracker recalculates its rotation based on the external reference system, and the samples afterwards that approach the mean value again are the increasing drift of the sensors. The nature of MTP latency entails that this can only be speculation. Other parts in the pipeline like drivers receiving the signal, the VR application, other applications in the background or the display could be sources of this emerging pattern which can only be analyzed further by separately analyzing the different subsystems.

Our prototype setup uses photodiodes directly attached to one display of the Vive HMD. Mounting the photodiodes on the lens would allow easier baseline measurements of VR hardware. However, tests revealed that it produced less distinct patterns. It would also have an impact on the usability and would potentially induce measurement artifacts since then the photodiodes and cabling would be visible by the users, hence we did not follow this approach but placed the photodiodes directly on the display. Today's HMDs usually don't utilize all pixels of the displays. The circular lenses only cover parts of the rectangular displays, leaving unused pixels around the display edges, prominently in the corners of the rectangular screens as can be seen in Figure 3. The photo illustrates this partial coverage of the screen clearly in the bottom left corner. It shows how the circularly shaped rendered image leaves the corner of the rectangular display dark. A similar partial coverage is detected for the other three corners not seen so prominently in Figure 3. However, physical access to these unused areas of the screens requires a more invasive approach to the hardware of the HMD. For example, with our currently used HMD the photodiodes would require to get attached via drilled holes on the side of the HMD chassis enclosing the view cones. In addition, this placement requires rendering to the uncovered areas. We currently refine our prototype and investigate other HMD types to support these features.

We evaluated the feasability of our approach with the setup shown in Figure 1. The Vive HMD is disassembled to get access to the display. Photodiodes are then attached to measure brightness in distinct regions of the screen. This setup can be extended as shown in Figure 12 to be used during normal VR usage: The photodiodes are attached at the corners of the display as described before. Most of the screen can then be used by the actual VR application, with only a small area reserved for the brightness readings. The better the photodiodes are attached, the smaller the reserved region needs to be, as adjacent brightness values don't influence the readings as much. The encoded data can be rendered over the scene in the same way a user interface is rendered on top of a 3D computer game. The wires can be led along the cable of the HMD. Some tracked object needs to be in the tracked space to follow a known rotation. The tracked object can be placed in a corner to not obstruct the users pathway.

The tracked movement was chosen to be a rotation, which is not common for human movement. Input devices, however, include algorithms to improve tracking for human motion [12]. Start and stop movements show an initial and settling delay [4], which we ignore, by disregarding the first measurements when the tracked controller accelerates and stopping the measurement before the tracked controller is brought to rest again. The rotation has the benefit of providing a continuous signal that is easy to control with a microcontroller. Non continuous movement patterns potentially introduce

more tracking artifacts due to inertia influencing the sensors inconsistently.

We chose to use a Vive tracker as the tracked device fixed in the setup. This frees the normal motion controllers and the HMD to get used in an experiment. The HMD only needs few additional wires to attach the sensors which can be guided alongside the existing cables. The measured latency can only be taken as a guideline and does not need to express the latency between a movement of the HMD and its respective effect on screen. Some optimizations such as asynchronous timewarp [34] only apply to the HMD to reduce the perceived latency. Late-latching [3] renders controllers with updated positions, which are not reflected in our approach.

The latency mean of the measurements with load is lower than the one without load. The plot shows many outliers with increased latencies, but there are more readings below the mean as well. A possibility is that a system under load might be pressed to sometimes read the tracker information later, and therefore closer to the next display scanout. The effect could be similar to an unvoluntary late-latching.

The described setup measures the total MTP latency but does not provide insight into where the latency originates. Finer details can be obtained by feeding the tracker data directly into the microcontroller, by means of its exposed pins to estimate the tracker influence or cut out the tracker of the setup by providing the tracking signal directly from the microcontroller via e.g., the audio input as in the approach by Di Luca [7].

Our setup assumes the motor to actuate the change in position perfectly, and without time variant latency. We did not test the motor and motor circuit latency which needs to get deduced from the measured end to end latency.

The setup is not restricted to HMDs, but can be used for large screen systems as well. The aquisition of screen brightness values is currently fitted to the Vive display and would need to get adapted for different displays.

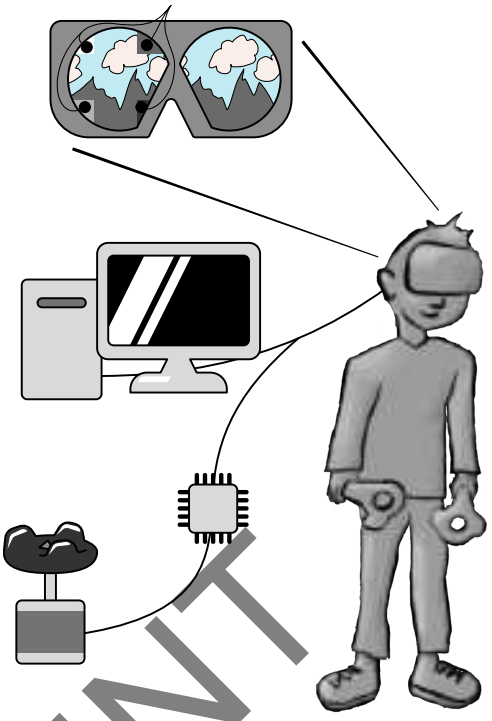

Figure 12: Proposed extension: The photodiodes are positioned at the rim of the screen to be minimally disturbing the experience. If the HMD allows to render outside the visible area, attach them there. The wires that connect to the photodiodes can be led alongside the HMD cable.

7 Conclusion

Latency in Virtual Reality (VR) applications can have numerous detrimental effects, e.g., a hampered user experience, a reduced user performance, or the occurrence of cybersickness. Today, measuring latency in VR systems usually is restricted to report mean values sampled over some dedicated and often isolated application runs which do not accurately reflect overall system behavior under varying load conditions. This paper introduced an apparatus to continuously measure per-frame latency and that therefor is able to capture latency spikes and hence latency jitter. A microcontroller drives a motor with an attached Vive tracker. The tracker's orientation is encoded by a VR application onto parts of the HMD screen as regions of different brightness. The microcontroller reads this data with photodiodes and calculates the difference between the real tracker orientation and the reported orientation as MTP latency. The apparatus can pick up the addition of artificial latency into a VR application for validation.

We evaluated our approach under three different load conditions. We inject artificial latency in a VR simulation and showed that latency jitter artifacts already occur without system load, potentially caused by the tracking of the specific HMD, and how mean latency and jitter increase under system load, leading to an overall degraded system performance. In contrast to previous approaches, the system does not rely on the HMD to be fixed to an external apparatus, can be used to assess any simulation setup, and can be extended to continuously measure latency during run-time. We stress the fact that continuous measurements of latency behavior of VR application should be a central means to monitor and control an important characteristic of VR systems to assure a high usability and user experience and hope we could contribute an adequate approach to implement such measurements. The source code for the setup is available at `https://github.com/Nighink/latency-rotation`.

References

[1] J. Allard, V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin, and S. Robert. FlowVR: a middleware for large scale virtual reality applications. In *Euro-par 2004 Parallel Processing*, pp. 497–505. Springer Berlin Heidelberg, Berlin, Heidelberg.

[2] A. Becher, J. Angerer, and T. Grauschopf. Novel Approach to Measure Motion-To-Photon and Mouth-To-Ear Latency in Distributed Virtual Reality Systems. *arXiv:1809.06320 [cs]*, Sept. 2018. arXiv: 1809.06320.

[3] A. Binstock. Optimizing vr graphics with late latching. `https://developer.oculus.com/blog/optimizing-vr-graphics-with-late-latching/`, 2015.

[4] C.-M. Chang, C.-H. Hsu, C.-F. Hsu, and K.-T. Chen. Performance measurements of virtual reality systems: Quantifying the timing and positioning accuracy. In *Proceedings of the 24th ACM international conference on Multimedia*, pp. 655–659. ACM, 2016.

[5] S. Davis, K. Nesbitt, and E. Nalivaiko. A systematic review of cybersickness. pp. 1–9. ACM Press. doi: 10.1145/2677758.2677780

[6] J. Deber, R. Jota, C. Forlines, and D. Wigdor. How much faster is fast enough?: User perception of latency & latency improvements in direct and indirect touch. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pp. 1827–1836. ACM, New York, NY, USA, 2015. doi: 10.1145/2702123.2702300

[7] M. Di Luca. New method to measure end-to-end delay of virtual reality. *Presence: Teleoperators and Virtual Environments*, 19(6):569–584, 2010. doi: 10.1162/pres_a_00023

[8] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes in latency during head movement. In *Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Com-*

munication, Cooperation, and Application Design-Volume 2 - Volume 2, pp. 1129–1133. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1999.

[9] S. R. Ellis, M. J. Young, B. D. Adelstein, and S. M. Ehrlich. Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 43, pp. 1182–1186. SAGE Publications Sage CA: Los Angeles, CA, 1999. doi: 10.1177/154193129904302203

[10] L. H. Frank, J. G. Casali, and W. W. Wierwille. Effects of visual display and motion system delays on operator performance and uneasiness in a driving simulator. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 30(2):201–217, 1988.

[11] S. Friston and A. Steed. Measuring latency in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 20(4):616–625, 2014. doi: 10.1109/TVCG.2014.30

[12] E. Gach. Lighthouse tracking examined. `https://kotaku.com/valve-updates-steam-vr-because-beat-saber-players-are-t-1832536574`, 2019.

[13] J. S. GmbH. Heavyload v3.5. `https://www.jam-software.com/heavyload/index.shtml`, 2019.

[14] D. He, F. Liu, D. Pape, G. Dawe, and D. Sandin. Video-based measurement of system latency. In *International Immersive Projection Technology Workshop*, p. 111, 2000.

[15] Z. Ivkovic, I. Stavness, C. Gutwin, and S. Sutcliffe. Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3d Shooter Games. pp. 135–144. ACM Press, 2015. doi: 10.1145/2702123.2702432

[16] O. Kreylos. Lighthouse tracking examined. `http://doc-ok.org/?p=1478`, May 2016.

[17] T. Kmrinen, M. Siekkinen, A. Yl-Jski, W. Zhang, and P. Hui. Dissecting the end-to-end latency of interactive mobile video applications. In *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications - HotMobile '17*, pp. 61–66. ACM Press. doi: 10.1145/3032970.3032985

[18] M. E. Latoschik and H. Tramberend. A scala-based actor-entity architecture for intelligent interactive simulations. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2012 5th Workshop on*, pp. 9–17. IEEE. doi: 10.1109/SEARIS.2012.6231175

[19] J. Liang, C. Shaw, and M. Green. On temporal-spatial realism in the virtual reality environment. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*, pp. 19–25.

[20] K. Mania, B. D. Adelstein, S. R. Ellis, and M. I. Hill. Perceptual sensitivity to head tracking latency in virtual environments with varying degrees of scene complexity. In *Proceedings of the 1st Symposium on Applied Perception in Graphics and Visualization*, APGV '04, pp. 39–47. ACM, New York, NY, USA, 2004. doi: 10.1145/1012551.1012559

[21] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks. Effect of latency on presence in stressful virtual environments. In *IEEE Virtual Reality, 2003. Proceedings.*, pp. 141–148, March 2003. doi: 10.1109/VR.2003.1191132

[22] M. Mine. Characterization of end-to-end delays in head-mounted display systems.

[23] G. Papadakis, K. Mania, and E. Koutroulis. A system to measure, control and minimize end-to-end head tracking latency in immersive simulations. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pp. 581–584. ACM, 2011.

[24] K. S. Park and R. V. Kenyon. Effects of network characteristics on human performance in a collaborative virtual environment. In *Virtual Reality, 1999. Proceedings., IEEE*, pp. 104–111. IEEE, 1999. doi: 10.1109/VR.1999.756940

[25] L. Rebenitsch and C. Owen. Review on cybersickness in applications and visual displays. *Virtual Reality*, 20(2):101–125, 2016.

[26] T. Sielhorst, W. Sa, A. Khamene, F. Sauer, and N. Navab. Measurement of absolute latency for video see through augmented reality. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 215–220. ISSN: null. doi: 10.1109/ISMAR.2007.4538850

[27] M. Slater and A. Steed. A virtual presence counter. *Presence: Teleoperators & Virtual Environments*, 9(5):413–434, 2000.

[28] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Reducing application-stage latencies for real-time interactive systems. In *9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE Computer Society. doi: 10.1109/SEARIS.2016.7551584

[29] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Reducing application-stage latencies of interprocess communication techniques for real-time interactive systems. In *Virtual Reality (VR), 2016 IEEE*, pp. 287–288. IEEE, 2016. doi: 10.1109/VR.2016.7504766

[30] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Towards comparable evaluation methods and measures for timing behavior of virtual reality systems. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pp. 47–50. ACM, 2016.

[31] J.-P. Stauffert, F. Niebling, and M. E. Latoschik. Effects of latency jitter on simulator sickness in a search task. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 121–127. IEEE, 2018. doi: 10.1109/VR.2018.8446195

[32] A. Steed. A Simple Method for Estimating the Latency of Interactive, Real-time Graphics Simulations. In *Proceedings of the 2008 ACM Symposium on Virtual Reality Software and Technology*, VRST '08, pp. 123–129. ACM, New York, NY, USA, 2008. doi: 10.1145/1450579.1450606

[33] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and S. I. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*, pp. 43–50. IEEE, 2009. doi: 10.1109/3DUI.2009.4811204

[34] J. Van Waveren. The asynchronous time warp for virtual reality on consumer hardware. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, pp. 37–46. ACM, 2016.

[35] W. Wu, Y. Dong, and A. Hoover. Measuring Digital System Latency from Sensing to Actuation at Continuous 1-ms Resolution. *Presence: Teleoperators and Virtual Environments*, 22(1):20–35, Feb. 2013. doi: 10.1162/PRES_a_00131

[36] J. Zhao, R. S. Allison, M. Vinnikov, and S. Jennings. Estimating the motion-to-photon latency in head mounted displays. In *2017 IEEE Virtual Reality (VR)*, pp. 313–314. ISSN: 2375-5334. doi: 10.1109/VR.2017.7892302