# Effects of induced latency on performance and perception in video games

Timo Hoth

*February 26, 2022*
Version: 1.0.1

Changelog after submission:

- added copyright info
- removed student number
- removed declaration of academic integrity as that was only necessary for submission
- repositioned and resized most graphs for aesthetic reasons
- description of figure 5.3 now specifies that the graphic shows a histogram

Westfälische Wilhelms-Universität Münster


Fachbereich Mathematik und Informatik
Institut für Informatik

Master of Science Thesis

# Effects of induced latency on performance and perception in video games

Auswirkungen von Latenz auf Leistung und Wahrnehmung in Videospielen

Timo Hoth
-

*1. Reviewer* Prof. Dr.-Ing Lars Linsen
Institut für Informatik
Westfälische Wilhelms-Universität Münster

*2. Reviewer* Dr. rer. nat. Dimitar Valkov
Institut für Informatik
Universität des Saarlandes

*Supervisor* Dr. rer. nat. Dimitar Valkov

November 8, 2021

**Timo Hoth** (-)
*Effects of induced latency on performance and perception in video games*
Master of Science Thesis, November 8, 2021
Reviewers: Prof. Dr.-Ing Lars Linsen and Dr. rer. nat. Dimitar Valkov
Supervisor: Dr. rer. nat. Dimitar Valkov

**Westfälische Wilhelms-Universität Münster**
Institut für Informatik
Fachbereich Mathematik und Informatik
Einsteinstraße 62
48149 Münster

# Abstract

Local end-to-end latency has been shown to reduce performance in human–computer interactions. I further investigated this relationship in an online study with over 600 participants, using the car soccer video game Rocket League. The game poses fundamentally different challenges compared to the aiming in first-person shooter games that have received the majority of research focus in the past. The goal of the study was to determine the performance loss due to added latency, as well as how players perceive latency in the game. I also examined factors that influence the effects of latency such as the level of skill of players, their peripheral vision, and the graphical effect density. On a task where players have to shoot with precision and power, latencies of 33 ms and above result in a significant loss in performance. The effect size is greater for the higher skilled players. The graphics and peripheral vision variables showed no interaction with latency. Additional latency of 8 ms caused a significant increase of the perceived latency for the participants as a group. The individual JND of participants that are at the 99.5th percentile of skill is 28 ms. The JND of players of average skill is above 50 ms, which is the highest latency condition in the experiment. Unfamiliar graphics settings result in a small constant increase of perceived latency with all levels of latency. The peripheral vision does not change perceived latency.

# Zusammenfassung

Wissenschaftliche Studien zeigen, lokale Latenzen führen zu einem Leistungsverlust in Videospielen. Ich habe dieser Verbindung weiter nachgeforscht in einer Online-Studie mit über 600 Teilnehmern, durchgeführt in dem Autofußball-Spiel Rocket League. Das Spiel stellt eine grundlegend andere Herausforderung dar als das Zielen auf Gegner in Shooter-Spielen, welche bisher die größte Forschungsaufmerksamkeit erhalten haben. Das Ziel der Studie war es zum einen den Leistungsverlust der Spieler durch hinzugefügte Latenz zu bestimmen und zum anderen zu testen, wie genau die Spieler Latenz wahrnehmen können. Außerdem habe ich zusätzliche

Faktoren untersucht, bei denen ich eine Interaktion mit Latenz vermutet hatte. Die Faktoren sind: das Können der Spieler, die unterschiedlichen Grafikeinstellungen und das periphere Sehen. Die Teilnehmer hatten die Aufgabe den Ball mit möglichst hoher Präzision und viel Kraft zu schießen. Bei einer Latenz von 33 ms oder mehr ist die Leistung im Durchschnitt signifikant schlechter. Bei den besseren Teilnehmern ist die Effektstärke von Latenz größer. Es gibt keine signifikante Interaktion zwischen Latenz und Grafik oder dem peripheren Sehen. Die Teilnehmer demonstrieren als Gruppe die Fähigkeit, die Minimallatenz von 8 ms signifikant von der Basiskondition zu unterscheiden. Die differentielle Wahrnehmbarkeitsschwelle der 0.5 % besten Spieler beträgt 28 ms. Die Schwelle eines durchschnittlichen Spielers liegt dagegen über 50 ms. Dies war der maximal getestete Wert in der Studie. In den Abschnitten, in denen die Teilnehmer mit Grafikeinstellungen gespielt haben, an die sie nicht gewöhnt sind, haben sie geringfügig mehr Latenz wahrgenommen. Das periphere Sehen hat als Faktor keinen signifikanten Unterschied hervorgerufen.

# Acknowledgement

First and foremost my thanks are towards Prof. Dr.-Ing Lars Linsen for giving me the ability to work on a topic that I am personally interested in. I owe my greatest thanks to my supervisor Dr. rer. nat. Dimitar Valkov for answering my questions and teaching me about the aspects of scientific work that I was unfamiliar with. I want to express my deepest appreciation for my girlfriend Deep for being there for me every single day to lift my spirits, motivate me, help my confidence, and proofread my thesis. A shoutout to all the 763 anonymous participants of the experiment, who gave their time to science and me, without a personal reward. An additional thanks to everyone that tested beta versions of the experiment to help make the experiment run smoothly. I extend my gratitude to Psyonix for creating Rocket League and allowing players to run local mods, which was a necessity for the experiment. Furthermore, I am very grateful of Chris Mulder for creating BakkesMod and answering my mod related questions, without which the experiment would not have been feasible. I'd also like acknowledge Mark Rejhon of Blur Busters for improving my understanding of monitors, which helped me significantly with the identification process of the monitors in the experiment. Lastly, I want to thank my parents and grandfather for always supporting me on my university endeavors.

# Contents

# Introduction

Gaming, with esports in specific, has been a growing market over the past years [www1]. The amount of related research has also been increasing at a rapid pace [1, 2]. Entire workshops and journals have recently been created to further the scientific understanding of esports [3, 4]. One of the topics of analysis has been latency[1]. Research has shown that it impacts player performance in games (see section 1.4). This means low latency is especially crucial in esports, as any disadvantage could cost players the win and attached prize money. The effect has been most documented in aiming tasks such as those found in first-person shooters. Although a large amount of research exists, and performance degradation with latency has been modeled in aiming tasks, there is less work available for other types of games (see chapter 2). One popular esports game is Rocket League (RL)[2] [G1]. It gives the player control over a car, making the controls fundamentally different from first-person shooters. Existing models cannot be applied to such a different and complex task. I ran a remote experiment that participants did on their own computers because an in-person experiment with sufficient experienced players was unfeasible during the COVID-19 pandemic. The goal of this thesis is twofold: First, I seek to reproduce the results of Martens et al. [5] and see the impact of visual latency on performance and perception in RL for players of different levels of expertise. Second, I try to look for interacting effects with latency. For that purpose, I modulate the graphics settings to see if players perceive or act differently with low or high graphics settings in combination with latency. Furthermore, I analyze the *field of view* (FOV) sizes used by the players, and test whether having more of the image in the player's FOV has an interaction with latency. Since the experiment is done on the participants' own computers, I attempt to estimate the baseline latency in order to control for it as a factor.

[1] Local system end-to-end delay/motion-to-photon latency (see section 1.2)
[2] Car soccer video game, published 2015 by Psyonix, used in the study (see section 1.1)

## Thesis Structure

**Chapter 1**: Foundations

The chapter *Foundations* contains the basic knowledge needed to fully understand the further chapters. It covers an explanation of the chosen game Rocket League (1.1), definition of the term latency (1.2), what game mechanics are (1.3), why latency is important in gaming (1.4), and what the different orders of control are (1.5).

**Chapter 2**: Related work

As the name of the chapter indicates, it summarizes the existing research on latency. It is split into three aspects: measuring latency (2.1), perception of latency (2.2), and the performance and behavior changes due to latency (2.3).

**Chapter 3**: Experiment methodology

The most important part of the chapter is the explanation of the research questions that motivate the experiment (3.1). Afterwards, an illustration of why Rocket League was the game of choice (3.2), followed by a short excerpt about the participants that the experiment targets (3.3), and then how the experiment works start to finish (3.4).

**Chapter 4**: Implementation details

The *Implementation details* chapter breaks down the steps that were necessary to make the experiment possible. First, there is a long section clarifying how the latency was estimated based on prior measurements and knowledge of the sources (4.1). A thorough understanding of all sources of latency is required to ensure that no significant factor is left out. The second section groups the documentation of the other challenges that had to be overcome in order to ensure a smooth automatically running experiment (4.2).

**Chapter 5**: Results

The chapter first breaks down the participants demographics, skill level, and other relevant information (5.1). Then it states the resulting effects of the experiment conditions on the performance of the participants (5.2) and their perception (5.3).

**Chapter 6**: Discussion

The following chapter elaborates on the results (6.1–6.3), contains interpretations, and covers limitations (6.4). Lastly, there is a recommendation for software developers on how to set a latency target based on scientific evidence, along with guidelines

that Rocket League players can use to make educated decisions regarding latency (6.5).

**Chapter 7**: Conclusion and future work

The final chapter gives a summary of which goals of the study were achieved, which weren't. It also highlights the most significant findings and the holes in the knowledge, where further research is necessary in order to fully understand the implications.

# 1 Foundations

## 1.1 Rocket League

Rocket League (RL) [G1], published in 2015 by Psyonix, is a sports video game that can be summarized as soccer played with cars. It is a popular esport that has been growing ever since its inception, both in terms of viewership [www2] as well as prize money. The most recent season of the *Rocket League Championship Series* (RLCS X, the biggest RL esports event) awarded more than $4.5 million prize money in total [www3]. In the past, the game has had eight world championships that required players and allowed fans to attend in person [www4]. Since the beginning of the COVID-19 pandemic, all such events have been canceled, including the planned RLCS X world championship [www5]. Regional online events were held instead, splitting the prize money between the regions. In the U.S., the game has many big college esports events [www6, www7]. According to *The Esports Observer* (TEO), RL is the 7th most impactful PC game as of the first quarter of 2021 [www8]. The press has repeatedly touted the game as having the potential to become the most popular esport in the world. The most common reasons cited are that it is easy to understand even by non-players, and that it is not based around in-game violence, making it accessible for people of all ages [www9–www11].

Beyond using cars to play the game, RL differs from soccer in other ways. The primary game mode is played with 3 players per team, although 1v1, 2v2, and 4v4 modes are also available for play. Players are not assigned to fixed positions and can drive around freely to take any role when needed. Matches last 5 minutes plus an infinite time golden goal overtime if the game was tied at the end of regular time. The field is bounded by walls and a ceiling that keeps the players and the ball inside, ensuring non-stop play until a goal is scored. This is also made possible through the fact that there are no additional rules like offside or fouls since the players cannot actually get injured. The cars are also unlike regular street cars. They can jump, double jump, do directional flips, and use rocket boosters. Due to the ability to freely tilt the car when it is in the air, the rocket boosters allow the cars to fly via propulsion, creating fully 3-dimensional play. This is assuming that the players are experienced enough to control their cars in these situations. Unlike soccer simulations (e. g. *FIFA 21* (2020) [G2], *eFootball PES 2021* (2020) [G3]),


Fig. 1.1.: Rocket League screenshot showing a car shooting the ball.

when the player is near the ball, there is no automated portion where the avatar controls the ball and the player can decide whether to run with the ball or press a button to shoot/pass. In Rocket League, the player can only control how their car moves. The ball reacts on impact with the car, following the game's laws of physics. With enough practice, players are able to dribble, pass, and shoot precisely and powerfully. However, this requires a mastery of the car's movements, similar to how a real soccer player would need to learn to angle and move their foot for a powerful shot.

## 1.2 Latency definition

In the context of this thesis, *latency* (or lag/system latency/local latency/transmission latency) refers to the delay between the user performing an action and the relating result appearing on the screen. The term often used in research is *end-to-end latency* [6, 7], or more recently, *motion-to-photon latency*, originating from Oculus VR [8]. The term latency will also be used when referencing just a segment of the full end-to-end latency. An example of a segment of the total latency is the display latency, which refers to the delay from the point where the computer outputs an image signal until it appears on the display. In those cases, the contextual information will clarify what is meant.

Audio latency and tactile latency are factors that aren't specifically investigated in the thesis; however, the method used to create artificial additional visual latency in the experiment increases the other latencies by the same amount.

In the gaming sphere, latency is often referred to as input lag. The computer and display lag are considered a part of the total input lag, despite the term specifically referencing "inputs". *Latency* is the exclusive choice in this thesis, but other terms appear in cited works.

Network latency is introduced when games are played with other players over an internet connection. Originally, games like *Quake* (1996) [G4] waited until the server verified the inputs to display them on screen [9]. This means any network latency will add to the local end-to-end latency. Savery et al. [10] give an overview of other networking strategies used in video games. *Lag compensation* [11] is most common in modern shooter games, as the plethora of game netcode analyses by the YouTube channel Battle(non)sense show [www12]. This strategy avoids adding local latency. RL's netcode works by simulating the future state of the player and game world (dead reckoning), only requiring adjustments for the parts of the gameplay that couldn't be predicted [12]. Since the client has perfect information of the user's actions, the unpredictable part is only related to other online players, which is why the player experiences zero additional local latency on their character. This thesis only covers the effects of local latency.

## 1.3 Game mechanics

In order to win at video games, the player has to perform actions such as running, jumping, aiming, or shooting. These actions are all classified as game mechanics. Different games have different core mechanics that are essential to the gameplay, although there is a large overlap between games of the same genre. Often, gamers split the skills required for gaming into two categories: decision-making and mechanical ability[1]. Decision-making covers anything from long-term strategy to split second decisions. Good mechanical ability is about how accurately and reliably players execute the game actions required to actualize the decision they made. This is very similar to common classifications in psychological literature, where the distinction is between mental and sensory-motor skills. The two aspects are never completely separate, but they can be assigned a priority with the following

[1] Mechanical ability is commonly shortened to *mechanics*, e. g. if a player has good mechanics, they are good at executing the game's mechanics.

rule: “In sensory-motor skills the overt actions clearly form an essential part of the performance, and without them the purpose of the activity as a whole would disappear. In mental skills overt actions play a more incidental part, serving rather to give expression to the skill than forming an essential part of it.” [13, p. 21].

## 1.4 Latency in gaming

I consider latency in gaming to be relevant in three ways:

1. player reaction time
2. player performance degradation
3. player perception of latency

The first relationship is simple. In most real-time games, players will have to react to some sort of unpredictable behavior at some point. The reaction can only occur after the related event is displayed on their screen, and their inputs also need to be processed after the reaction. Thus, any extra millisecond of latency subtracts from the available time the player has to react appropriately. In short, there is less time for decision-making. The game is more difficult the higher the latency is.

Player performance is tied to latency beyond just reaction time. Assume a player gets shown a target that they’re supposed to point at. Then they have to close their eyes while they move the mouse to the target. The expected accuracy is low. With the eyes open, the expected accuracy is much higher because the player can see the mouse cursor and target. They can see the result of their movements on screen and adjust based on that [14, pp. 81–82]. This is a constant loop of predict, do, observe, adjust that will happen in any real-time game.[2] The higher the latency, the further the prediction needs to be in the future, and the later the adjust step will be, which is especially problematic if the target is moving. In short, this hurts mechanical ability. MacKenzie and Ware [15] first demonstrated in *human–computer interaction* (HCI) that performance can degrade beyond reaction time. In their experiment, movement time increased by more than the added latency.[3]

Player perception of latency is a common topic online in gaming spheres. In communication with the community, I found many comments like “just tried it and i instantly noticed a difference oh my god this is amazing thankyou man

[2] This effect does not have an impact on rapid disconnected movements below 0.3 s. There is no time to react and adapt on movements that short.
[3] Further examples will be discussed in chapter 2..

:handshake:" in response to an optimization which reduces latency by 7 ms [www13]. Self-proclaimed latency enthusiasts do not accept any noticeable delay, regardless of whether it puts them at a disadvantage. Some of them claim to notice as little as 1 ms of extra latency. Although such feats have never been demonstrated in an indirect input task under scientific conditions, it is clear that the enthusiasts will react negatively to any latency that they can notice. This may lead to complaints about "sluggish" controls, which can negatively affect the perception of the entire game. Scientific demonstrations of such effects on non-enthusiasts are available. Kaaresoja et al. [16] found that the quality of a button was rated poorly when the response to the press was delayed more than 100 ms. For games in specific, Jörg et al. [17] found increased player frustration when the controls were delayed, without players being aware that the reason was latency. Some did call the controls bad, however.[3]

Although not specifically demonstrated in games, interactions between the perception of latency and the performance degradation may exist. For example, a player notices a degradation in performance and concludes that there must be latency. Or a player notices latency and performs worse because they have been primed[4] to think that they will do worse.

## 1.5 Control order

The mouse is the most commonly used aiming device in esports games. The mouse is an example of *zero order position control*, meaning the location of the mouse has a direct correlation to the position of the cursor/reticle on the computer (as defined by Wickens et al. [19, pp. 147–148]). With a controller, aiming is a *first order control task* as the analog stick determines the velocity of the cursor.[5] If the analog stick determines the acceleration, then it is considered *second order control*. The naming is based on the equations of motion where zero order: $\vec{x}(t)$, first order: $\vec{x}'(t) = \vec{v}(t)$, and second order: $\vec{x}''(t) = \vec{a}(t)$.

Higher order control tasks add an extra level of indirection to the control. This could either increase or decrease the interaction with latency. The decrease hypothesis rests mainly on the fact that in higher order control, delays already exist. The user has to use a control strategy that performs well despite the movement delay caused by acceleration. The strategy will presumably also work well with additional latency.

[4] As explained by Kahneman [18].
[5] Additional smoothing may exist.

The increase hypothesis can best be described with an example. Let there be a car along a one-dimensional track that can be moved forwards using the right arrow key and backwards using the left arrow key. The goal is to get as close to one of the ends of the track, which one switches at random intervals. When one of the swaps happens, the player switches direction as fast as they can. Once there is added latency of e. g. 100 ms, this will increase the time until the player reacts on average by 100 ms. In this time, the car continues to move in the wrong direction. This requires an extra 100 ms to get back to the position where the swap happened without the delay. So in fact the player is at a 200 ms disadvantage. This would be the case with first order control, but the issue only gets more severe with higher order control.

Rocket League always uses higher order controls. The throttle, brake, and rocket boosters are second order controls over the location, as the player controls the acceleration of the car in the forward direction. The steering is zero order orientation control of the wheels, and therefore angular velocity of the car. That would make it a first order control over the car's orientation, and second order control over location. However, the wheels are limited by the game physics which means that the actual orientation cannot change instantaneously like the input could, causing some potential extra delay. In the air, the player has second order orientation control as the inputs control only the angular acceleration of *pitch*, *yaw*, and *roll*. Second order orientation control means third order location control. The only type of zero order control in RL is the camera movement which the user can do additionally to the default automated camera systems. These are important at the highest level of play to allow players to keep a better overview of the field of play; however, there is no precision required. The experiment in this thesis focuses solely on car control.

# 2 Related work

The related work regarding latency can be split into different categories. First there is the topic of measuring latency present in human-computer interactions, for which there are numerous methods that were developed across the last 35 years. Then there are the effects of latency which are relevant to gaming, as defined in section 1.4. I cover what the available research shows about the perception of latency in human–computer interaction and specifically games. Although there are multiple effects on player performance, they are not separable in most tasks. Thus, I group the influences on player performance under one section.

## 2.1 Measuring latency

To calibrate my experiment, I need to measure the baseline system latency. To determine which method to use, I looked for related research, and found 37 different sources discussing measuring methods. Since Rocket League is played with a gamepad controller by the majority of players, the measuring method needs to be compatible with it.

The majority of methods are related to tracking devices for virtual reality. These often attempt to create a location or angle delta between the tracker and displayed object by using controlled movements [20–26]. A picture is most often used to measure the spatial delta, and since the movement is controlled, it can be used to calculate the time delta, meaning the latency. Other methods measure the time directly by starting the measurement when the tracker is past a set point and stopping it when the virtual response happens [6, 27–29]. The measurement can be done with a high-speed camera or photosensors with an oscilloscope or special measuring device. While such methods can in theory be adapted to a controller, the movement of the analog sticks or buttons is at most a few millimeters, requiring very high precision for acceptable accuracy measurements.

Rather than relying on movement to visualize latency, Graves and Bradley [30] used the electrical signals of a microphone picking up the sound of a button press as a trigger point. Casiez et al. [31] provide another non-intrusive method using a

vibration sensor attached to the user's finger. Both methods have limited accuracy as the activation point of a button press is not necessarily at the exact location where sound and vibration is maximal. For this purpose, other methods open input devices and attach directly to the circuitry in order to measure the exact moment where a button is pressed, or trigger it through external means [32–35, www14–www16]. Rejhon's [www14] method wires an LED to a button so that it lights up when the button is pressed, and uses a high-speed camera to capture measurements. The number of frames between the LED lighting up and the screen response is used to calculate the latency. The measurement is not free of bias since a human has to analyze the video. High accuracy for the average latency can be achieved when using many samples. The technique has been used in peer-reviewed research [36–39]. Schmid and Wimmer's [35] method works by triggering the button of an input device with a microcontroller and measuring the screen response using photodiodes. The microcontroller has a time resolution of 4 $\mu$s, the expected error is small, and it is easy to automate for a high sample count. My own measuring device was developed separately but is essentially identical in function to Schmid and Wimmer's. More information is available in section 4.1.2.

## 2.2 Perception of latency

The oldest sources on the perception of latency are educated guesses on thresholds not based on experimental research. Miller wrote in 1968, "This response should be immediate and perceived as a part of the mechanical action induced by the operator. Time delay: No more than 0.1 second" [40, p. 271]. Nielsen [41] is commonly cited, and references this recommendation as well as Card et al. [42]. It also states the 100 ms limit, based on experiments about the perceptual fusion of visual and auditory signals less than 100 ms apart [42, 43, pp. 31–34]. However, as the original source states, the limit may be 50 ms or possibly even lower under special conditions [43, p. 34]. These recommendations thus stand on limited evidence, and they do not consider indirect effects of latency that may be perceivable.

The available research shows that the perception of latency is highly dependent on the task and input or output type. Users may not perceive latency directly in those situations, but nevertheless, they may get a subpar experience.

First, there is the distinction between direct and indirect input, which describes the difference between a touchscreen (direct) and mouse (indirect). Both offer zero

order control but the finger or stylus on a touchscreen is always on top of the cursor, while the mouse is on a different surface and the cursor moves relative to its position. Deber et al. [44] created an experiment which was identical in all aspects but where the cursor was projected directly underneath the finger or a wall. They found that in a dragging task, the participants' *just-noticeable difference* (JND) was on average 11 ms for the direct and 55 ms for the indirect version. When dragging an object with latency, the cursor trails a distance behind the finger, which is clearly visible in direct input where the cursor is then right next to the finger. In indirect input, there is no direct reference. This effect has been demonstrated even when the direct input is offset by 65 mm [45]. The size of the cursor plays a role in how visible the spatial difference due to latency becomes [46].

The difference between a static and a moving task is also severe. Deber et al. [44] also tested a flashing cursor upon a virtual button press with direct and indirect projection. They found JNDs of 69 and 96 ms respectively. This result reveals that latency on a button press is more difficult to detect than on movement, and the direct input case also allows the detection of lower latencies. This result shows that direct and indirect inputs are still different, despite no observable spatial difference. The results from Deber et al. [44] offer a one-to-one comparison of conditions, and approximately match other research [16, 46–48]. The 96 ms JND for a static indirect input task matches the theoretical recommendations from the first paragraph of this section.

Virtual reality is another environment in which direct input exists in the form of tracking the head. When the user moves their head, the virtual view has to move the exact same amount or the static virtual world tracked by the eye will appear to move. Latency will cause a delay in that movement and reduce the spatial stability of the scene. The data of multiple studies shows that the JND of latency in this condition is similar to direct touch input results [7, 49, 50]. Ellis et al. [51] and Adelstein et al. [52] previously reported 10 ms higher numbers. Jerald [50] also investigated how the user is more likely to perceive latency the faster they accelerate their head. He developed the following model which he verified in experiments.

$$\Delta t = \tau + \psi\left(\frac{1}{\phi''}\right) \quad \Big| \quad \tau = 10\,\text{ms}, \psi = 1\,°/\text{s} \quad [1] \tag{2.1}$$

$\phi''$ is the peak head acceleration in $°/s^2$. Mania et al. [49] and Ellis et al. [53] investigated what happens if the simple single object environment of the previous studies

[1] The values of $\tau$ and $\psi$ were estimated from the experiments.

is replaced by one with a background or a complex "photorealistic" scene[2]. The results confirm the data from previous experiments, with no statistically significant difference for the environment conditions.

The output type plays a role in how likely the user is to perceive latency. Using a custom-built device with haptic feedback, a speaker, and an LED, Kaaresoja et al. [16] determined JNDs of 52 ms for tactile responses, 80 ms for auditory responses, and 85 ms for visual responses. Mäki-Patola and Hämäläinen [54] measured similar audio latency perception using a Theremin[3].

Professional gamers may be more sensitive to latency. A draft by Banatt et al. [55] compared the ability of expert gamers to detect latency after pressing a button compared to non-gamers. The experiment was done with a custom-made device with near zero baseline latency. The result for the JND of non-gamers was 114 ms, which is within the margin of error of what previous studies found. The gamers detected latency on average at 48 ms, significantly lower. Unfortunately, the experiment results required the exclusion of some datapoints due to issues with the apparatus and chosen conditions. Although the quality of evidence is not enough to draw any conclusions with certainty, it is still a hint that experienced gamers can perhaps become more attuned to notice smaller latencies. If that is the case, then it draws into question the validity of all known latency thresholds for this group that were established with casual or non-gamers.

The research on the perception of latency in a higher order control task is limited. Martens et al.'s [5] experiment finds that users rate perceived latency significantly higher at 97 ms of additional latency but does not report a JND. Furthermore, they found that additional latency of 28 ms significantly increased the participants' rating of perceived difficulty. This demonstrates that users can perceive the downsides of latency without being able to attribute them to the root cause. The experiment is very different from the simple cursor used by Deber et al. [44]. A direct comparison is therefore not possible.

Latency can cause negative changes in perception. Kaaresoja et al. [16] and Kaaresoja et al. [56] studied how user perception of a virtual buttons declines with a 100 ms latency increase. Participants consistently perceived buttons with increased latency as less pleasant to use and being of lower quality. They were also less willing to buy devices with high latency buttons. Jörg et al. [17] used a custom game and added 150 ms of latency for one group of players. Participants in the group rated

[2]The complex scene shows a completely static room with a table, two chairs, a mirror, a shelf, and a handful of decorative objects. The lighting was pre-rendered using radiosity algorithms. The listed polygon count of 35 000 is low by today's standards.

[3]Electronic instrument played with two hands.

the controls as more difficult, were less satisfied with their own performance, and were more often frustrated than those of the control group.

## 2.3 Performance and behavior changes due to latency

Performance declines when latency increases. Conklin [57] and Poulton [58, pp. 201–202] reported on a tracking task latency experiment by Warrick performed in 1949, which is likely to be one of the oldest experiments studying sub-100 ms latencies. The original source [59] was not obtained but the reports are quite detailed. The experiment involved a CRT which displays a target and a cursor which is controlled by a subject [57]. The target gets oscillated by a function generator while the subject attempts to stay on it as precisely as possible. For the results of the most difficult tracking task in the experiment, Poulton [58, p. 202] reports, "The percent time on target is reliably reduced with a time lag as short as .04 sec." This task involved the target moving along overlapping 0.25 and 0.05 Hz sine waves. With a simple 0.1 Hz sine wave, the performance reduction was "reliable" at 0.08 s, while a 0.05 Hz wave was not affected. At the highest latency of 0.32 s all waves were affected. This experiment demonstrates that latency below 100 ms can affect performance and hints that more difficult tasks may be more affected by latency.

To further investigate these effects, MacKenzie and Ware [15] set out to measure and model the performance impact of latency on a pointing task with the mouse. The experiment involved targets at different distances and sizes that the cursor had to be moved to. This allowed them to calculate the index of difficulty ($ID$) in bits for every condition via Fitts' law [14, pp. 249–255, 60]. They measured the movement time to target ($MT$), the error rate ($e$). In their comparison, they found a significant impact on both variables. The movement time at 225 ms increased by 63.9 % compared to the 8.3 ms result, and the error rate increased by 214 %. They calculated the throughput via Fitt's law [60], and found a reduction of 46.5 % at the highest latency. There was also a significant impact of $ID$ on the performance, as expected. Lastly, there was a significant interaction between latency and task difficulty. The performance degradation of latency increases with task difficulty. They created a model to predict the movement time based on task difficulty ($ID_e$[4]) and latency

[4] $ID_e$ is the index of difficulty accounting for error.

($LAG$). A multiplicative relationship provided the best fit for the experimental data:

$$MT = 230 + (169 + 1.03LAG)ID_e \tag{2.2}$$

The experiment is one of many showing that the performance degradation of latency increases with task difficulty [36, 61–64]. Ivkovic et al. [29] provide an exception to this rule as the lower target speed causes a larger decrease in performance with additional latency. This is, however, likely related to the method of scoring, which is *time on target*. By just swiping left and right over the target randomly, the time on target will already be above zero. The more latency gets added, the more difficult it becomes to do any purposeful tracking, and it therefore approaches the score of the random strategy. The player gets a significantly better score on the lower speed, and therefore has more to lose. This hypothesis can perfectly explain Ivkovic et al.'s [29] data. Time on target is not a recommended method of scoring tracking tasks [58, pp. 46–49].

Latency can impede performance at levels below the 0.04 s mark from Warrick's [59] experiment [5, 48, 65–67]. Spjut et al. [66] investigate very low latencies. They compared 12 and 20 ms as two scenarios seen in real-world optimized esports games. They found a significant difference in median task completion time. The median task completion time increases from 1.348 to 1.530 s ($d = 0.319$) with the increased latency. This is a very noticeable disadvantage at the highest level of competition.

Performance loss due to latency can be reduced by using compensation techniques such as post-render warping [68–70]. This can result in virtual artifacts. An alternative strategy is to predict inputs ahead of time [71–73]. This has the issue of potentially causing jittery movement. Ivkovic et al. [29] investigate the option to add performance enhancing input modifications that counterbalance the loss due to latency.

Jittering latency was investigated by Pavlovych and Stuerzlinger [62] in a tracking task. They found that it significantly increases the error by more than the same value of constant latency. However, in a follow-up study, the jitter did not result in a statistically significant increase of the error at all [63].

Increasing latency can lead to the user changing how they control virtual movements [58, pp. 203–205, 5, 64, 74–76]. The common denominator between the studies is that users adjust their input less often but use larger magnitudes to compensate. Poulton [58, pp. 203–204] describes that in a tracking task with 0.18 s of latency, the user makes corrections to the path every 0.38 s, which matches the latency plus

reaction time (0.2 s). Friston et al. [64] demonstrate that in a pointing task (like MacKenzie and Ware [15]) their participants move the mouse faster when latency is increased, and then need more time to correct mistakes from overshooting the target.

A direct comparison of zero to first order control was done by Claypool [37] in a custom game called *Puck Hunt*. The game involves a puck with constant speed bouncing inside a square area. The goal is to get the cursor on the puck and click, which was done with both a mouse and a gamepad. The experiment involved latencies of 50–450 ms. They found that the selection time was well approximated by an exponential model of latency $d$ and target speed $s$.

$$T = 2 + 0.3e^d - 0.03e^s + 0.2e^d e^s \tag{2.3}$$

It shows a steeper increase with latency than the mouse data [77]. Since the task is overall more difficult with the gamepad, it is unclear whether the difference would still persist in a task of equal difficulty. The effect size was not reported. Pantel and Wolf's [78] results of lap times in a racing game can also fit an exponential model, although no modelling was attempted by the authors.

Martens et al.'s [5] study is the only one using second order control with a focus on smaller latencies. This makes it the closest predecessor to the experiment in this thesis. Based on prior work in control theory, Martens et al. argue extensively for the hypothesis that higher order control requires lower latencies for maximum performance, especially with large control gains. They did, however, not expect that to necessarily mean that players perceive lower latencies. The experiment setup involves a ball balanced on a beam. The participants' task is to keep the ball as close to the center as they can. This differs from other second order control tasks, in that the beam displays the user's zero order inputs. The baseline latency of the system was 11 ms.

In the first round of tests each subject did trials on 0, 49, 97, and 194 ms of added latency with two different levels of input gain. The order of conditions was randomized. They found a significant decrease in the ability of the participants being able to keep the ball centered on the beam. This difference was significant for the lowest added latency of 49 ms. Performance was also affected by control gain, but there was no interaction between the two factors. Further analysis showed that latency cause participants to adapt their mouse movements, and additionally, the authors note, "[ . . . ] the participants, who adapted more strongly to , tended to show a smaller performance loss" [5, p. 9]. The control adaptation data showed an interaction between latency and control gain. Both factors and their interaction

also caused an increase in perceived control difficulty. Participants were also polled about their perceived latency, which they were able to distinguish at better than random chance for the 97 and 194 ms conditions. The discrimination was more pronounced at the higher of the two gain settings.

In their second round of testing the conditions were simplified, using only the higher gain and not asking the participants about perceived latency. The added latencies were 0, 14, 28, 42, 49, and 97 ms. Performance was significantly affected at all levels, with an effect size of $d = 0.61$ at 14 ms. Perceived difficulty did not pass the significance level at 14 ms once Bonferroni correction was applied, but it did at every level above. Similarly, control behavior changed significantly at 28 ms and higher.

In summary, the study by Martens et al. [5] shows that 14 ms latency can significantly affect human performance in a second order control task. Latencies below this have been insufficiently researched. Users are able to notice differences of 28 ms and above, but were not able to attribute these changes to latency until 97 ms. There is insufficient evidence to conclude whether latency is more or less of an issue in higher order control tasks.

# 3 Experiment methodology

## 3.1 Research questions

The purpose of the study is to determine the effects of induced latency on player performance and perception in an esports video game with experienced players, and to further the understanding of latency by investigating potential interactions. The research is to be done with a video game that doesn't involve aiming with a mouse, as those are already well researched.

As a minimum, I have the goal of replicating the following findings of previous studies. I test degradation of player performance with 8.33–50.00 ms of added latency, as well as their perception of the added latency. The evaluation of performance is based on shot power, precision, whether the ball is scored, how many tries the player needs to make contact with the ball, and a custom score metric. This is a compound metric of the stats that is supposed to approximate how effective the shot would be in an actual game (explained in section 4.2.2).

H1 Added latency degrades player performance.

H2 Players report higher values when polled about their perceived latency if latency is added.

The difference compared to most previous esports studies is that RL doesn't involve a zero order control aiming task. Compared to a similar second order control study [5], RL is an established game. I expect that the experienced players are able to distinguish smaller latencies due to their familiarity with the game's controls. I expect that the best players' performance will degrade the most with added latency, as their mechanical ability requires great timing and accuracy.

H3 Experienced players are able to distinguish added latencies better.

H4 Experienced players' performance degrades more with added latency.

Beyond the difference in the game, the purpose of the study is to test the following hypotheses:

H5 The graphical effect density has an interaction with the effects of added latency.

H6 The players' field of view (screen size in relation to distance) has an interaction with the effects of latency.

These are untested hypotheses that will further the understanding of latency. They would allow players and developers to make decisions when choosing graphics settings to partially mitigate negative effects of latency. Similarly, a different monitor or sitting distance may be chosen if there is an interaction with latency.

The significance level for the hypotheses is $\alpha = .05$.

## 3.2 Why was Rocket League chosen?

There were multiple reasons for why RL was chosen. The most important points are in the list, with a detailed explanation afterwards.

Rocket League ...

1. is popular, with a thriving esports scene and a lot of experienced players.
2. is easily moddable through BakkesMod (BM)[1].
3. is not based around zero order control aiming mechanics.
4. has a high degree of mechanical difficulty.
5. is a game I am very knowledgeable about. I understand how it works on a gameplay and technical level.
6. has a community in which I have contacts and followers, making it possible to find many experienced participants.

1. The game's popularity and esports scene makes it more likely that a large group of people will participate. It is also easier to find participants who are highly experienced in the game. Comparing players of different experience levels was one of the goals of the study (see section 3.1).

2. Modding allows for better control in experiments, automation, and detailed data collection that leads to better studies. I knew that this is possible in Rocket League through a BakkesMod plugin. There is even a way for easy plugin distribution[2] which allows download and installation in a few clicks for anyone using the mod. Since the

[1] User created Rocket League mod with an open API to allow developers to create their own mods for the game.
[2] bakkesplugins.com

mod already has thousands of unique users daily, it makes it easy to convince players to participate. Similar tools are not available for most other esports games. Originally, the experiment was planned to be done at an in-person professional tournament. Due to the COVID-19 pandemic, it was unclear when the next tournament of that kind would happen. With time constraints in mind, it was decided to move the experiment online. This meant it would have to be fully automated to ensure that every player follows the same steps, making it impossible without a mod.

3. As mentioned in section 1.5, RL controls are of higher order than all the other most popular esports. Even though controlling the car can be classified as a tracking task, it is fundamentally different from the other games because there are more degrees of freedom. There have been a significantly fewer studies on the effects of latency on higher than zero order control [5]; none with esports games. This experiment is supposed to be a step in that direction. It is relevant for racing and flying games, and it may also be relevant for shooter games, as the player movement in those games is either first order or second order control. Player movement is only of secondary importance in those games. It is still often expected that players make movements with high precision.

4. Rocket League has a lot of mechanical depth. There are many techniques that take hundreds to thousands of hours of practice until a player can do them semi-reliably. A player picking the game up for the first time will have trouble steering the car to a stationary ball and shooting it into the goal. This is part of the appeal of the game, as it expected that the high difficulty will result in it being very latency sensitive. This is also different from any artificially created task for the purpose of latency testing. Such a test would inherently be new to the participants, and thus, could not be made so difficult that players need hundreds of hours of practice to learn it.

5. I am personally highly knowledgeable about Rocket League, having spent thousands of hours testing, researching, and producing videos on how the game works. The Lead Gameplay Engineer of Rocket League once said he learned a few new things about the game from my videos [12]. The knowledge gives me a head start in setting up experiments and knowing what to look out for. This turned out to be especially useful when having to fully automate the experiment due to it being run online.

6. In addition to the benefits in knowledge, my videos have allowed me to make contacts in the community and get followers on YouTube and Twitter. This makes it considerably easier to get a large amount of experienced players and even professionals to participate. This was one of the goals for the experiment.

## 3.3 Participants

Since the experiment would have to be done online due to the COVID-19 pandemic, I decided to let anyone participate to generate a large dataset. In order to attract participants, I put out a video on YouTube and announcements on my social media explaining the experiment and how to partake [www17]. I personally contacted all professional players that were following me on Twitter in order to ensure participation of players up to the highest skill level. No personal compensation was offered to anyone. The analytics of the platforms show that the audience is 98.7 % male, and $\frac{3}{4}$ are age 18–34 y. The vast majority of viewers are from North America and West Europe. Due to past content being mainly about Rocket League and some about latency, it is expected that the average viewer of the announcement is more invested in RL and more informed about latency than the average player. The information collected about the participants of the experiment is relayed in section 5.1.

## 3.4 Experiment setup

In order to answer the research questions, I set up the experiment with the independent variables: graphical effect density, added latency, and shot. The between-subjects variables that I consider are player skill group, the baseline latency of their system, and the monitor being in the peripheral vision of the participant.

There are three graphics conditions. One with all the effect settings to the minimum, one with every effect set to the maximum, and one condition with the settings the participant usually uses. The *RENDER QUALITY* setting does not get changed because it is just a render resolution setting. Anti-aliasing and motion blur are also left unchanged, as I consider them to be about sharpness and not a graphical effect, similar to resolution. The biggest effect differences between the minimum and maximum graphics conditions are bloom lighting, lens flares, grass straws extending from the ground, dynamic shadows, moving background audiences, flashing cameras, waving flags, and weather effects in the form of moving clouds and a shader simulating rain droplets. The map *MANNFIELD (STORMY)* was chosen to ensure this density of effects. A previous study investigated the impact of graphical complexity on perceived latency, but both conditions only tested a fully static scene [49]. The highly dynamic effects in this experiment may make the start and ending of a movement less obvious which I believe could negatively influence the perception of latency. It may also be outright distracting.

The added latency conditions are 0.00, 8.33, 16.67, 33.33, and 50.00 ms. The minimum step size is due to technical reasons explained in section 4.2.1. The step size is increased beyond the two minimum steps in order to cover a wider range of latencies without creating too many conditions. The maximum of 50 ms is expected to be enough to see effects based on prior latency studies and blind self-tests.

I designed 5 shots to cover different game situations, requiring different approaches, and covering different difficulty levels. The shots are designed to balance two factors: completable by less experienced players, and player skill being an important factor in how precise and powerful the shot is scored. Due to the highly experienced target audience, the second factor takes precedence. I consider it important because I assume that when skill makes a large difference, latency does too. All shots require significant steering adjustments to touch the ball, in order to ensure that the participants cannot simply drive a straight line at the correct speed, which I presume would be less affected by latency. In a turn, a player has to decide when to stop the turning input based on the visual information, which is more difficult when the player has to predict further into the future due to latency. Detailed descriptions of each shot can be found in appendix A.1.[3] There is a limit of two touches per attempt to avoid dribbling. Allowing dribbles would change the outcome dramatically, equalize different shots, and introduce a lot of noise if a significant portion of the participants attempted this strategy.

The experiment follows a randomized block design with the conditions, graphics by latency by shot. Each scenario gets repeated exactly twice $3 \times 10 \times 5$. The 10 can be thought of as each of the 5 different latency conditions existing twice. The order in which the three graphics conditions are shown is random for each participant, but all repetitions on the same graphics settings happen in a row. The order of the (10) latency conditions is random but stays constant for a set of five shots, which are also in a random order. Before the 150 shots, there is a warm-up period of 10 shots with unchanged latency and graphics. This is done to limit noise, as the first trial of a shot will be very different from performing a shot that one knows. Of course, further learning is to be expected across the experiment, but it isn't easily preventable and there are no order effects due to the complete randomization of the condition order.

In order to partake in the online experiment, a potential participant has to download and run the experiment on their own hardware. The experiment is an easy installation for the user. The majority of the target audience is expected to already use

[3] The shots can also be seen on video [WWW18] or loaded in the game with the custom training code CAF1-E26A-0AC0-F792.

BakkesMod. If not, it requires the download of an installer and clicking a common installation process. When BM is installed, plugins — such as the one for the experiment — can be installed with a single click on bakkesplugins.com. To activate the plugin, a single button press was needed. This brings up the privacy policy, which informs the user about all the data that is being collected in the experiment and how it is being used. If the user agrees to it, they can continue and will be loaded into the map for the experiment.

At the beginning of the experiment the participants are instructed on what they should be aiming for: "Your goal is to shoot into the center of the net as precisely as possible. Power is good, but the priority should be precision." They are also provided information on the experiment structure.

Detailed data gets collected on every shot. This involves basic requirements to assert which shot and scenario the player is on and checks of whether it was correctly loaded and the status code for completion. Furthermore, there is data that allows the assessment of how well the player performed, i. e. the location and velocity of the ball and car, how often the player missed the ball, how many times the ball was touched, and whether a goal was scored. Lastly, there is performance information about the game i. e. frame time, game thread time, render thread time, and gpu time. These are necessary to estimate the participant's baseline end-to-end latency.

During the experiment, the participants are polled after every set of five shots. The question displayed is, "How delayed did your inputs feel during this part of the experiment?" (see figure 3.1) The answer is given on a scale of whole numbers between 0 and 6 which matches the 0–6 physics ticks of delay applied during the experiment.

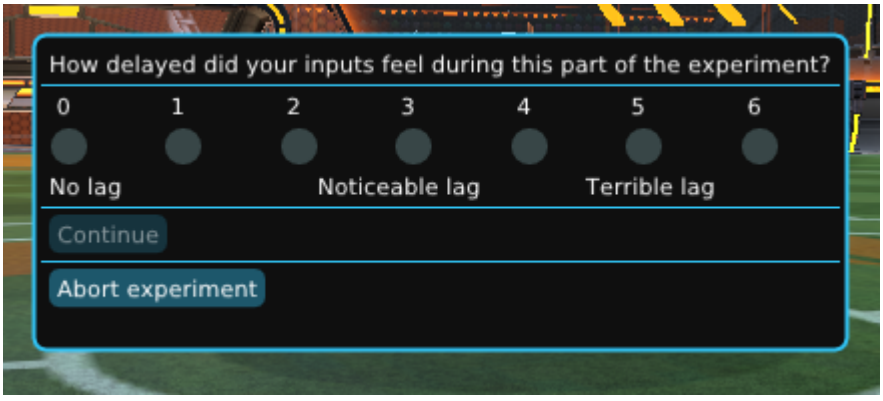


Fig. 3.1.: Dialog displayed to the participants after every set of shots.

After completing all the shots, which is expected to take around 20 minutes[4], the participant is shown a survey to provide additional information. This includes how far the player sits from the monitor[5], estimated hours of RL experience[5], whether they are a professional player, age[5], and gender[5]. Additionally, there are questions about the player's hardware in order to estimate the latency on the participants computer. When possible, the information was directly read through code. The details are covered in sections 4.1.4 and 4.1.5. The participant can then press "Finish". This brings up a confirmation dialog, informing the user that their data will be sent.

[4]It may take longer for a player of low skill because shots get repeated when no contact is made.
[5]Answering was not required

# 4 Implementation details

## 4.1 Measuring and estimating latency in Rocket League

Since the experiment is done on the participants' own computers, they are experiencing different baseline latencies. During the experiment, additional latency gets added to compare player performance and perception on different amounts of latency (see section 3.4). It is expected that a participant's performance and perception is adapted to their baseline latency. A player used to a higher baseline latency could have learned to be better at dealing with latency in general. The opposite effect might also be possible if there is a soft border at which latency becomes so high that performance drops off significantly. Since the players can judge their own performance, it could affect the perception equally. In order to control for these potential effects, I try to figure out the participants' baseline latencies.

Asking each of the participants to accurately measure their own latency as in section 2.1 is not a feasible solution as it requires specialized gear or a high frame rate camera. In order to get an estimate of players baseline latency, I take an alternative approach. I estimate the input lag based on the participant's monitor, input device, frame times, and other related measures.

This section explains how the estimation of latency is done. In order to make an accurate estimate, one has to fully understand all the sources of latency that exist for an end-user playing the game. This is the topic of the first subsection. The knowledge fuels the theory behind the estimation, while measurements calibrate and validate the estimation. The subsequent subsections cover how latency was measured, how the measurement data was used, and the math of the theory.

### 4.1.1 Understanding the sources of latency

There are various sources of latency when playing a video game. These can be split into three main parts of the input-output-chain. The input device (capturing the player's action), the computer (handling and calculating the game state and graphics

based on the captured input), and the monitor (displaying the graphics calculated by the computer) [www19]. Some latency is caused by the connection of the three parts. Further subdivision of the three parts will be discussed in the subsections.

#### 4.1.1.1 Input device

Whenever the user wants to interact with the computer, they need some kind of input device. These devices can range from the keyboard and mouse over touchpads and trackballs to joysticks, steering wheels, and controllers specifically created for gaming. Rocket League displays a recommendation to use a controller when the user starts the game for the first time on PC. A keyboard with a mouse is the alternative default input method. Other platforms (PlayStation, Xbox, Nintendo) are only compatible with the officially supported controllers. 89 % of the participants use a controller. The remainder use a keyboard either with or without a mouse. I decided to only use controllers in the estimation for the following reasons:

- The participants used numerous different keyboards and mice, which would all have to be identified, and their individual latency measured or sourced. 77 % of the controller players used a PlayStation or Xbox controller.
- The keyboard and mouse are separate devices, which are likely to have different amounts of latency. It is unclear how differing latencies in combination will affect the player's performance or perception.
- The dataset is still large without the non-controller players.

There are three different input types on a regular controller: buttons, joysticks and triggers. Buttons can be any kind of digital switch that is either on or off, the joysticks are 2D analog inputs that can be pushed up-down left-right, and triggers are 1D analog inputs that measure how far the trigger is pushed.

**Buttons** Buttons are an input method with only 2 possible states, *on* (1) and *off* (0). The point where a button is pushed a certain distance to switch from the *off* to the *on* state is called the *activation point* [79]. There is a delay between the time the user's finger starts to move and the time when the activation point is reached. This will depend on the travel distance of the button, required force, and how the user pushes the button. The users' behavior is impossible to account for without additional equipment to measure the true first reaction. Because of this, I only account for the latency that happens after the activation point is reached.

Regular buttons make use of a digital switch. An electrical connection is made or broken once the switch is pushed beyond the activation point, which causes a voltage rise or drop. The signal propagates through the connection to the microcontroller at close to the speed of light. This delay is orders of magnitude lower than the milliseconds of total delay relevant for the experiment; therefore, it can be ignored. The microcontroller can either detect this change through interrupt circuitry or by manually checking the signal state at an interval (polling). With an interrupt based approach, the microcontroller can immediately handle the change. With the polling based method, it is possible for the electrical connection to be made right after it was polled. When polling with a frequency of $f_{\text{but}}$, the maximum delay will be $\frac{1}{f_{\text{but}}}$ (e. g. $f_{\text{but}} = 1000\,\text{Hz}$, $\frac{1}{f_{\text{but}}} = 1\,\text{ms}$). Since the user is unaware of this internal polling, it is expected that on average a button press will occur uniformly random in the interval since the last poll. The average delay therefore is $\frac{1}{2f_{\text{but}}}$.

This does not account for switch bouncing. In regular buttons, the switch from *off* to *on* (or the other way around) is not a clean transition. The electrical contacts in the button bounce off each other when contact is first made. After a few milliseconds, a solid connection is made and the switch stays on continuously. If this issue is left ignored in the design of the circuit and software, then the microcontroller will detect multiple button presses when there was only one. This would always happen with an interrupt based method. With polling, it may or may not happen, and the lower the polling frequency $f_{\text{but}}$ the lower the likelihood of a false positive double press. This is insufficient, as bounces can still happen. One possible solution in the circuit is a capacitor. A capacitor can smooth and slow down the rise and fall time of the signal. This will cause a delay between the time the activation point is hit, and the time the signal measurement transitions from 0 to 1. This setup can also not guarantee that the signal only transitions from 0 to 1 once per press, as switch bouncing can occur right as the delayed signal transitions from 0 to 1. An additional noise gate with hysteresis is required to guarantee that such spikes do not make a difference [80]. A double throw switch can also offer a solution. It offers two terminals; one to the on state and one to the off state. The switch will never bounce all the way from one terminal to the other. This allows for a switch [81]. Alternatively, it is possible to handle bouncing issues in software. This requires some waiting until the bouncing has subsided, and the signal is constantly on/off or a certain threshold of on/off switches is passed. This will cause a delay on press and release. The more certain about true change an algorithm is, the more delay there is. It is also possible to look only for the leading edge of a press or release and immediately count that as a press with no delay, ignoring any changes for a period in which the switch is expected to bounce. This is more likely to work well on a button press than release,

as shifting the finger on a pressed button may cause very short disconnections of the electrical connection that wasn't intended as a release by the user. Depending on the exact type of switch, an unintended press could also happen with this method if there are vibrations from the user shaking the controller. This is more likely if the user already applies a low amount of force to the switch, mainly used in gaming to lower the physical time until the button is pressed. Therefore, such a strategy would not be robust.

Some gaming hardware manufacturers (e. g. SteelSeries, RAZER) have started using optical switches to get rid of bouncing issues and consequently reduce latency. As of the time of this writing, these switches are only used in their keyboard and mice, not their controllers.

**Analog sticks** The analog sticks are a type of 2-axis isotonic[1] thumb joystick, installed on all official controllers of the major console platforms (PlayStation, Xbox, Switch). They can be moved in a circular area, and as their name suggests, the controller measures an analog signal for each axis and turns those into digital values to send to the computer ($[x, y] \mid x, y \in \{8 \text{ or } 16\text{-bit int}\}$). Functionally this works by using a lever that is connected to two rotary potentiometers sitting perpendicular to each other [WWW20]. Both act as a variable voltage dividers, the voltage depending on the angle of the lever on each axis. The voltage is measured by the microcontroller with an *analog-to-digital converter* (ADC) that is either integrated or external. The measurement only takes a few processor cycles, and is therefore fast enough to be considered instantaneous on the millisecond-scale of total delay relevant in this context. Joysticks based on hall effect and optical technology exist, but are not used in commercial gaming controllers.

Similar to button polling, the analog signal gets sampled at distinct timepoints. When sampling with a frequency of $f_{\text{ana}}$, the analog stick position known by the microcontroller at any random moment may be 0 to $\frac{1}{f_{\text{ana}}}$ old. The average delay is $\frac{1}{2f_{\text{ana}}}$ as it is for buttons. It is also possible to define the latency in terms of distance. If the analog stick is moved with a constant velocity, the known location at time $t$ will on average be $[\frac{x(t)+x(t-\frac{1}{f_{\text{ana}}})}{2}, \frac{y(t)+y(t-\frac{1}{f_{\text{ana}}})}{2}]$. That is exactly halfway in between the true locations at time $t$ and $t - \frac{1}{f_{\text{ana}}}$. For non-constant velocities, it can be said that the stronger the acceleration, the larger the average spatial delay. The measured position will on average be closer to the true position at time $t - \frac{1}{f_{\text{ana}}}$. The opposite is true for deceleration, where the measured position will on average be closer to the real position. In the latency estimation, these acceleration based differences

[1] sensing the angle of deflection, as opposed to isometric (sensing force) [82, pp. 106–107]

are ignored as it is impossible to account for movement that the controller does not capture. This can theoretically cause up to 4 ms of inaccuracy, but assuming randomly distributed movements the average is exactly correct.

Noise in the signal is an important consideration for the design of the controller. If the manufacturer of the device determines analog noise to be too high after trying to minimize general electrical noise in the circuit, they may take additional steps to reduce it specifically for the analog sticks. As mentioned in the button section, a capacitor directly connected to the potentiometer output can smooth the signal. The introduced delay will depend on the rise/fall time of the capacitor. There are also specific noise reduction modes that can be used in some microcontrollers that will purposefully shut down elements not needed for the measurement [83]. This could potentially add an overhead and thus reduce the sample rate. One way to reduce noise through software would be to oversample the measurements and take the average of $n$ samples as the signal. The delay calculations from the previous paragraph can be simply adjusted to this change by replacing $\frac{1}{f_{\text{ana}}}$ with $\frac{n}{f_{\text{ana}}}$. Other software methods of filtering, such as a Kalman filter, may add no latency or different amounts of latency. They may also consider small movements noise, and filter those out entirely. A player using a controller with a strongly filtered analog stick may feel as if the game is not responding to their adjustments. This potential issue is difficult to quantify and compare to regular delay.

**Triggers** Triggers are a single-axis analog input method. They are present in most controllers in the form of a lever, but the rotation axis is often positioned in a way that pulling the trigger feels similar to pressing a button with a long travel distance. There are different ways manufacturers measure the trigger position. The Thrustmaster eSwap Pro controller uses a rotary potentiometer, the Microsoft Xbox controllers use Hall effect sensors, and Sony uses pressure sensitive buttons for their controllers. Each of these methods does not have a significant impact on latency and is most likely chosen for cost or reliability reasons. They all modulate an analog voltage signal that the microcontroller can measure. The expected noise concerns and delays of triggers are exactly the same as those of the analog sticks.

**USB, microcontroller hardware and software** The task of the microcontroller in the input device is to measure signals, translate them to the desired format, and send them to the computer. Each of these tasks requires instructions to be run by the microcontroller. One of the limiting factors of the speed of that operation is the clock speed of the processor. There are cheap microcontrollers available capable of running

at over 100 MHz. This results in more than 100 000 clocks in a single millisecond. For power saving reasons (battery powered), the clock speed of some processors may be lowered to as low as 32 kHz in which case properly handling all inputs within 1 ms may not be possible. Since the commercial controllers are not openly documented, information on the chosen clock speeds isn't publicly available.

It is more likely that purposeful waiting for issues like bouncing introduces significantly more delay than actual execution time of measurements and calculations. Incorrect prioritization of the computing resources could also cause increases in latency. This is a challenge for complex controllers like the DualShock 4, which has audio capabilities that it needs to control along with the handling of inputs.

In order to send the information to the computer, the controllers used in the experiment all use USB connections in wired operation. The Device Class Definition for HID 1.11 specifies how the connection is used for input devices [84]. The biggest consideration is the polling rate. The device itself defines the `bInterval` value, which tells the computer at which frequency it should poll. It does however need the context of the USB device speed. Low speed devices can be polled at up to 125 Hz, full speed devices at up to 1 kHz, and high speed devices at up to 8 kHz. As mentioned in the button polling example, the average delay introduced is given by $\frac{1}{2f_{\text{but}}}$ and the worst case delay $\frac{1}{f_{\text{but}}}$. Thus 125 Hz polling can introduce in the worst case 8 ms of input lag, while 8 kHz polling will cause at most 0.125 ms. Most gaming mice have the ability to request USB polling at 1 kHz. For controllers, special drivers are needed to run them out of their intended specification. All the popular controllers used in the experiment request 250 Hz polling, increasing average latency by 1.5 ms compared to 1 kHz. According to the spec, the controller is supposed to send input reports through the `Interrupt In` pipe at the rate of polling [84]. The controller doesn't have to do that though, which the Xbox controllers demonstrate. They only send up to 124 reports per second, and none if the inputs haven't changed. It also has to be noted that the interrupt pipe doesn't work like a true interrupt, as it will only be handled when the computer does the next USB poll request, and does therefore not circumvent the delay.

Wireless operation is popular through Bluetooth, or custom wireless protocols, both in combination with a USB dongle or an integrated chip. The signals travel through the air insignificantly faster than through cables. Wireless connections can suffer from lack of signal strength and interference. This can result in inputs never arriving at the computer, which means there is extra latency until the new information finally does arrive. Depending on the protocol, inputs completely dropping out may be possible, and a button press might never be recorded. The impact of wireless

technology is hard to quantify on the average input lag. As long as the protocol used allows very low latencies, the average delay is going to be the same as on USB with the same polling rate. There is also the possibility that it is slightly faster through an integrated Bluetooth chip that isn't bounded by the limits of USB. However, the worst case delay is likely to be much higher than on cable, unless one would try to measure in a perfectly isolated lab.

**Conclusion** The average latency of a controller cannot be estimated without knowing anything about the code running on it. However, it is possible to use external tools to measure the latency of a controller. It is expected that equal controllers run the same code and use the same components and therefore have the same latency. Based on available external evidence [www15, 85] and my own testing, this assumption holds true.

#### 4.1.1.2 Display

The display is the device which presents the visual result calculated by the computer. It is the final part of the input-output chain. The section is only the second of three though. The software has to account for some aspects of the monitor which is why this order is easier to understand than a linear one. There are several ways in which a monitor can introduce latency. Some are dependent on the specific display technology, but not all types will be covered.

**Scanout** The scanout process is best described by explaining how a *cathode-ray tube* (CRT) display functions. The tube creates an electron beam which gets bent with an electromagnet and hits a phosphorus screen. The screen lights up where the beam hits it. To turn this into a display capable of showing any image, there is a pattern of *red, green, and blue* (RGB) portions, the combination of which can create any color. The beam gets bent across the entire display from one side to the other, and this gets repeated for every line of resolution that the display has [86]. Meanwhile, the intensity of the beam gets modulated to allow any specific raster point (pixel) to have any brightness. This entire process happens 60 times a second ($f = 60\,\mathrm{Hz}$) on a typical display. So fast that it creates the illusion of a continuous image without flicker, despite each pixel only being lit up a fraction of the time after the electron beam hit it.

The CRT display gets its signal over an analog connection which is exactly what modulates the intensity of the electron beam [87]. Information cannot be transmitted

the entire time. Whenever the beam has moved from one side to another, it will readjust back to the first side while blanking [88, p. 122]. This is called horizontal blanking. When the entire display area sweep is completed the same principle happens vertically too. At the vertical blank, the monitor will send a signal back to the device providing the image signal, which is used for the purposes of synchronization. The use of the *vertical synchronization* (VSync) signal prevents the image from being updated in the middle of scanning out, which creates a visual artifact known as tearing [89, p. 538].

CRT displays are regarded as zero latency because they display the signal immediately. The time delay due to the signal traveling through the cables and the electrons through the CRT is only a few nanoseconds, and therefore, negligible. When considering the scanout, CRT displays do still have inherent latency. Even though the pixel being currently illuminated is as up-to-date as it can be, each individual pixel has to wait $\frac{1}{f_{\text{rr}}}$ time to get the next update. On as screen with a 60 Hz refresh rate that is $\frac{1}{60}\,\text{s} = 16.7\,\text{ms}$. So a random pixel on the display will at any moment have a latency in the range of $[0, \frac{1}{f_{\text{rr}}})$ and on average $\frac{1}{2f_{\text{rr}}}$. That is 8.3 ms on a 60 Hz screen. This does not account for the blanking times which are monitor dependent and will affect the true number by less than 5 %.

*Liquid crystal displays* (LCDs) are the most common type of display in use today [90]. Liquid crystals act as polarizers that can change their function depending on the applied voltage. With the help of a second static polarizer and a backlight this makes it possible for any liquid crystal to create any brightness in theory. In practice, perfectly blocking all light is not possible. The crystals are organized in a matrix with RGB filters [88, p. 122], making it possible to mix any color. In this matrix, it's not possible to address all pixels at the same time which is why a multiplexing is used to apply voltages to pixels one at a time [91]. This ultimately means they are driven in order like in a CRT and suffer from scanout latency the exact same way. Although blanking would not be needed from a technical standpoint, the displays still have it for intercompatibility with CRT signals [89, p. 538].

CRT displays can be driven at higher refresh rates by reducing the resolution, as that reduces the amount of lines the beam has to trace per refresh. The total lines drawn per second stays the same. Modern LCDs for gaming often have refresh rates of 144 Hz (3.5 ms avg.scanout latency) at resolutions of up to 3840 $\times$ 2160, and the monitor with the fastest scanout currently available in a consumer LCD runs at 390 Hz (1.3 ms avg.). This display is likely to be faster than a CRT overall, despite suffering from the other issues that can cause extra delays in LCDs.

**Pixel response time** Liquid crystals do not change state instantaneously. A great deal of research has gone into making the changes fast enough to be used in motion displays at all [92]. Modern displays are able to transition between most brightness levels fast enough to fit into a 60 Hz refresh cycle. Gaming displays are often advertised as having 1 ms response times, which are based on 10 to 90 % transition completion [93]. Reviews show, however, that this is only true for a portion of the transitions and the average lies multiple milliseconds higher (e. g. this review by O'Keeffe et al. [www21]). A brightness transition from 0 to 50 takes a different duration than one from 205 to 255 (8-bit values). That is the reason reviews test a variety of brightness values.

The exact impact of response times on latency is not possible to put into a single number without some loss of information. If the display is showing a ball moving across the screen, the latency with a bright background could change compared to a dark one. A change in the brightness of the ball also could. Even if it was possible to quantify the average response time of a display in relation to the average content displayed on it[2], it would still not provide a solution. The issue is fundamental to pixel responses being non-instantaneous. If the transition was stopped at 50 % would the user still see the change? In most cases, yes. Does that mean that the 0 to 50 % response times should be used instead of 10 to 90 %? The user may still be able to detect even a 10 % completed transition, however, human reaction times are affected by stimulus intensity [94]. Thus, it would be expected for a display with a 1 ms fully completed response to allow for a faster reaction than one which has only completed 10 % in 1 ms and takes an extra 9 ms for the rest. The effects of this continuous change on human reaction time has not been sufficiently researched yet.

In the thesis experiment, monitor response times are disregarded because quantifying the effects is not possible. I assume the players react quickly even to small stimuli. The results and estimations used are for the first measurable response. More info in section 4.1.5.

**Backlight strobing** Some modern LCDs have the option to strobe their backlight as a means of reducing perceived motion blur. The details of this are not relevant for this thesis. However, it is important to know how strobing affects response times. The strobe is timed to happen after most pixels have completed the response transition. Therefore, when strobing is active, small deviations in response times are

[2] Ideally one would also weight each object for its importance to the user, i. e. in Rocket League the ball and cars are important but a pixel displaying a part of the background scenery is not.

not going to affect the user's reaction time. When the strobe occurs, the stimulus is already at full intensity. This does make it possible to clearly define the total latency of a monitor. Unfortunately (for the experiment), strobe modes are not used by most gamers.

**Digital processing** Modern displays also introduce latency through their digital processing chain. The digital part starts at the connection where the display uses a DisplayPort or HDMI interface. Despite working digitally, both are able to transfer pixel data in real-time and HDMI even has a mode to transfer the data faster than the refresh cycle demands. This could theoretically allow a 60 Hz display to scanout an image in $\frac{1}{120}$ s, reducing latency, but it isn't active in monitors right now. The latency in digital monitors happens due to the monitor processing. A digital signal has to be turned into an analog one to drive the pixels. In order to calculate the correct voltages for the pixels, a gamma curve has to be used, although manufacturers may employ additional techniques. As long as the techniques are simple transformations, they should cause only negligible delays. Sometimes more complex transformations (e. g. sharpening) are offered. In order to sharpen the image, the processor needs to know local information around the area it is currently processing. For that purpose, manufacturers may use a frame buffer which stores an entire frame, then applies the processing, and does the scanout with delay. On a 60 Hz display, this would account for at least 16.7 ms of latency alone. The latency is often even higher in some TVs, which try to smooth motion by storing at least two frames and interpolating in between those. For gaming displays, these methods are usually passed by, and if buffering is needed, it can be limited to few pixel rows instead of the entire frame.

**Adaptive vertical synchronization** An LCD with *Adaptive Sync*[3] can refresh at varying intervals. Each display has a minimum and maximum frequency. When the display has finished a refresh cycle and there is no signal for the next refresh, the display will wait until a maximum waiting duration has expired. If at any point during the waiting period new information gets sent to the display, it will immediately start the scanout again. This allows the system providing the signal to determine at which rate to refresh, rather than having to adjust to the display. If the system attempts to refresh faster than the maximum or slower than the minimum refresh rate, it will behave exactly like a monitor without adaptive synchronization. The benefit for latency lie mainly in the system itself, as will be discussed in section 4.1.1.3. Additionally though, the scanout always happens as fast as the maximum frequency

[3] A term used by VESA. NVIDIA: *G-Sync*, AMD: *FreeSync*

allows, saving a bit of latency, i. e. a 48–144 Hz display being driven with 60 fps will scan out each image in 6.9 ms.

**Conclusion** Similar to a controller, the latency cannot be estimated due to many unknown factors that depend on the manufacturers choices. However, it is possible to use external tools to measure the latency of a display. It is expected that the equal monitors run the same code and use the same components and therefore have the same latency. Online reviews with accurate latency measurements confirm this assumption.

#### 4.1.1.3 Computer

The latency in the computer can be split up into the parts of USB/wireless drivers and OS handling the input device, the game calculating the game state changes based on the inputs, the game rendering the visuals of what is happening in the game, and buffers in between the steps and post rendering [www19].

**USB** The USB hardware on the PC has to handle the polling of the input device as explained in section 4.1.1.1. When the input device sends data, there is a transmission time based on the size of the report. Typically, with the speeds of USB this should not be a relevant portion of time on the scale of total latency.[4] When data is received it will be handled by drivers and the OS component, both of which could introduce delays by running a lot of code. On Windows, these are black boxes. There is no reason to assume that they do add a relevant amount of latency. When measuring the total latency present in a system and subtracting all the known factors, there was less than 1 ms left which could possibly be assumed to be caused by the OS and drivers (see section 4.1.2).

**Windows window messages** In Windows, the inputs get passed on to individual windows through messages which have to be handled by a `WindowProc` callback function. Usually a game engine will handle these inputs once per frame [89, pp. 529–531], as does Rocket League. This means that during this time, fresh inputs may be waiting for the start of the next frame. The average wait time in this step is therefore $\frac{1}{2f_{\text{frame}}}$, and the worst case $\frac{1}{f_{\text{frame}}}$. There have been reports online of overfull message queues when running games in exclusive full screen mode [www22]. The

[4] Plant et al. [95] measured a transmission time of 0.1 ms.

game only gets a limited amount of messages per frame, so they start accumulating. This can cause drops and additional delays. Attempts to replicate this issue in RL failed.

**Game engine fundamentals** While some games are built from the ground up, many modern games use a common engine that is licensed or developed in-house for multiple games [89, pp. 31–37]. So does RL, using the Unreal Engine 3 [12]. The engine handles the core game loop, where every frame can be subdivided into three steps. First, the engine captures the inputs and processes different controllers to give the developers a device independent action ($t_{\text{input}}$[5]). Then the main gameplay logic runs, which is defined by the game developer ($t_{\text{game}}$). Based on the new state of the game world the engine renders the next frame. This involves both the CPU ($t_{\text{render}}$) and the GPU ($t_{\text{GPU}}$). Once it is fully done, the frame can be presented to the output device, and the next frame can be started. Each of the steps requires processing time, which consequently means that the frame can only be displayed with latency. The input handling is very simple and takes only a negligible amount of time. The other parts will depend on the game details. In a multithreaded game architecture the steps may be designed to be partially independent of each other and thus the added latency $\leq t_{\text{input}} + t_{\text{game}} + t_{\text{render}} + t_{\text{GPU}}$ [96].

**Game logic code** For Rocket League this means handling the car input and stepping the physics forward in time. In case the game is played online, there may be additional steps required to send network packets and check and reconcile the state. The graphics rendering step in Rocket League contains no extraordinary parts. The overall processing time and therefore attributable latency of the frames is very much dependent on the used hardware and graphics settings. It can be lower than 1 ms for each frame on the fastest processors.

Rocket League has a factor specific to the game which can cause additional latency. The physics of the game run at a fixed tick rate of 120 ticks per second. Since the car's movements themselves are tied to the physics, an update can only happen up to 120 times a second. So when the game runs at 240 fps, the physics will only get updated every second frame, and thus, there is a 50 % chance that there is an additional frame of wait time until an action gets handled. The code to calculate this additional delay is given in listing 4.1.

[5] $t_x$ is the time the PC needs to process $x$.

```python
def lag_through_ticks(frametime, vardiv=10, max_it=2001409):
  ticktime = 1000/120 # 8.33 ms
  lag_sum = 0
  for i in range(0, max_it):
    # random noise +- 1/(2*vardiv) to prevent alignment issues
    noise = -1/(2*vardiv) + random.random()/vardiv
    # calculate how current frame lines up with the physics ticks
    mod = (noise + i*frametime) % ticktime
    if mod < frametime:
      # enough time has passed for another physics tick, so there is 0 additional
          ↪ delay
      continue
    # calculate how long it takes until the next physics tick is generated
    lag_sum += math.ceil((ticktime - mod) / frametime) * frametime

  return lag_sum/max_it
```

**Listing 4.1:** Calculation of the average added latency caused by waiting for the next physics tick (Python).

**Pipelining and queues** In order to get the maximum performance out of the CPU and GPU, they should be able to be fully utilized at the same time. Since the GPU needs to wait for the CPU steps before it can start its work, a pipeline is necessary to fully utilize the GPU. That essentially means, once the CPU has calculated a frame of gameplay logic, it won't just do the rendering portion of the game loop, but it will also start working on the next frame of inputs and game logic. Since the GPU has to do the majority of the rendering work, the CPU will usually have enough resources left to work on the next frame. As soon as the work on the new frame begins, the inputs for that frame have to be captured and set in stone. This creates a potential latency issue. If the render step of the pipeline takes longer than the game logic step, there is a wait time for the frame where it cannot be rendered yet. The frame is waiting in the render queue. The scenario can get even worse, as the game starts working on yet another frame of gameplay logic before the previous one has even started rendering. Therefore, a limit on the amount of pre-rendered frames is necessary. In RL's case this limit is two frames.[6] This time, where the frames are waiting in the render queue, is additional latency on top of the time needed to produce the frame. So even though pipelining can increase the frame rate, and that is beneficial to latency[7], it may still increase overall latency. Purposeful limiting of the frame rate in the right moment can therefore reduce the latency. If the queue fills completely, the additional latency is $\frac{2}{f_{\text{frame}}} - t_{\text{game}}$. A setting in the Rocket League configuration files (`OneFrameThreadLag=false`) can make sure there are never any additional frames in queue, reducing the worst case delay by one frame at the potential cost of frame rate.

[6] Determined experimentally by forcing conditions that fill the render queue.
[7] See the paragraph on Windows window messages

**Frame buffers** are the parts of the graphics memory which store the frames and allow the monitor to read out the pixel information [89, pp. 663–664]. In the past, some games used a technique called racing the beam in order to render the pixels just before they get transferred to the monitor [97]. This has the advantage of not requiring a frame buffer, which wasn't feasible due to required amount of memory. The entire idea of pipelines and the game loop doesn't apply then, which allows for very low latencies. It also has a significant disadvantage. Doing more complex calculations is only possible once per frame as opposed to once per pixel or pixel row. While the engine is working on rendering a frame, the buffer it is rendering to – called back buffer – is in an unfinished state that should not be displayed [98, p. 300]. The monitor, however, continually requires pixel data to read through the connection. For that, there exists the front buffer which contains the newest finished image. When an image is done rendering in the back buffer, a page flip is done which turns the front buffer into the back buffer and vice versa. As long as the page flip is done immediately, the latency of the buffer is already accounted for as $t_{\text{render}}$.

**VSync** When doing the page flip immediately, the monitor may be reading out any portion of the frame buffer at that moment. Thus, the monitor may display half of the previous and half of the newest frame, an artifact known as tearing [89, pp. 663–664]. Delaying the page flip until the vertical blanking of the monitor prevents this issue. This is known as VSync. There are two types of VSync that affect the latency in fundamentally different ways. One uses three buffers in total, two of which are back buffers that get rendered to continuously, while on every VSync signal the newest finished frame gets flipped to the front buffer [89, p. 664].[8] This means that any frame, after it's done rendering, has to wait a random time from 0 ms to $\frac{1}{f}$. If the frame rate is synchronized with the refresh rate of the monitor, the latency stays constant for the random value it started out at. Any deviation will cause varying latency in the aforementioned interval. With double buffered VSync, which is also known as regular VSync, rendering cannot continue until the single back buffer has been emptied. Thus, if the computer is capable of generating more frames than the refresh rate of the monitor, the game will end up filling the render queue to the maximum. That will result in the latency increase that comes with a full render queue. Additionally, the wait for the next VSync guarantees an extra refresh cycle of latency regardless of how quickly the render finishes. If the computer is producing frames at a lower rate than the monitor is capable of, the

[8] The method is known under the terms *Fast Sync* (NVIDIA) and *Enhanced Sync* (AMD). Sometimes it is referred to as triple buffering, but that can also refer to a method that is functionally identical to double buffering with an additional buffer.

render queue does not fill. Limiting the frame rate can therefore also reduce the latency when using VSync. The downside is uneven frame pacing, which Adaptive Sync (see section 4.1.1.2) can prevent, along with preventing extra latency by not having to wait for the next refresh cycle. It should be noted that the same average latency can be achieved with a non-Adaptive Sync monitor by deactivating VSync. This is the option that the majority of competitive gamers choose.

### 4.1.2 Measuring Rocket League latency

In order to estimate the latency present on a participant's PC, calibration and validation of the model is necessary. For this purpose, I created a measuring method similar to Schmid and Wimmer's [35] (cf. section 2.1). An Arduino Due set up as a USB game controller sends inputs to the PC and also measures the final response on the screen using a set of photodiodes in series. They are attached to two breadboards placed in front of the screen in a cardboard box to shield off extraneous light. The photodiodes span the entire vertical range of the screen in order to allow the detection of a change at any point during scanout process of the display. This minimizes the variance introduced in the measurement by the scanout of the display, and in turn a more accurate estimate of the game's factors. The Arduino was also set to use 1000 Hz USB polling for the same purpose, causing at most 1 ms of variance. The response of photodiodes is fast enough not to matter on the timescale of milliseconds [99]. Using photodiodes in this context means that the device can only detect simple changes in brightness. In order to measure the latency in Rocket League, I devised a special map with a black and white background. By swiveling the camera, the game will then switch from white to black where the photodiodes were placed. One additional caveat regarding the camera has to be addressed. By default, the game smoothly moves the camera across a few frames, even at the highest sensitivity setting. By editing the memory, the value can be set one hundred times higher than allowed in the menu, removing any delay caused by smoothing.

The exact measuring process is as follows. The Arduino stores the the start time found by calling `micros()` [9], and then immediately uses the *Arduino Joystick Library* [www23] to send the button state. Following that, it does an `analogRead(diode)` to determine the current brightness, which has to be above a minimum threshold in order for the measurement to be considered valid. If it passes, the Arduino runs analog measurements at a rate of about 375 kHz. The analog measurement of the

[9] Resolution: 4 $\mu s$

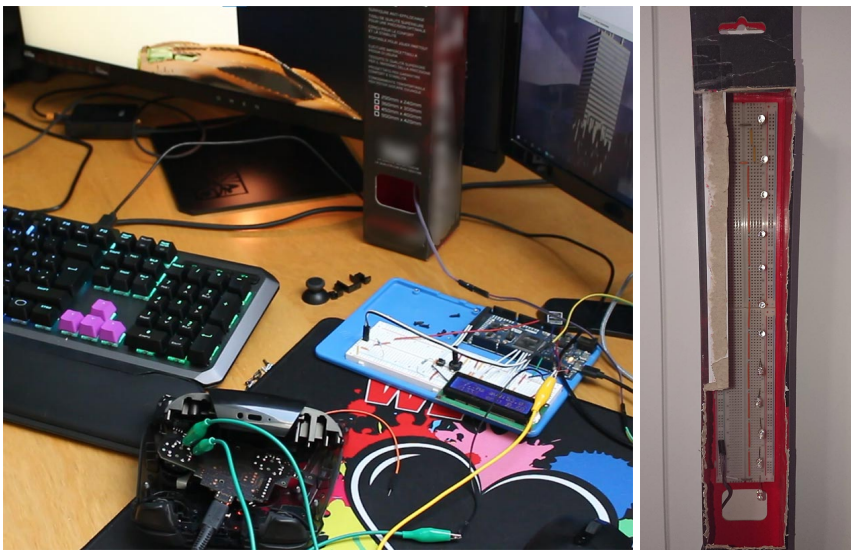

Fig. 4.1.: Measurement setup showing the Arduino with a breadboard and display. The controller is connected to trigger analog input. The monitor shows the custom black and white map. The cardboard box in front of the monitor can be seen on the right. It houses the photodiodes.

photodiodes has a significant amount of noise. When the measured value drops below the treshold which is 250 below the starting point (12-bit ADC) – about 20 % of the full transition – the end time gets measured, and the difference to the start is considered the end-to-end latency.

I developed the measurement setup in 2018 [www24] and compared it to Rejhon's [www14] method using a *Casio Exilim EX-FH20* 1000 fps camera and a *RAZER Naga Original* mouse.[10] The testing device measured 1.25 ms more latency, with a 95 % confidence interval of 1.03–1.48 ms. Of this difference 0.2 ms can be attributed to the threshold, based on measurements of the pixel response times of the monitor by O'Keeffe et al. [www21]. Even though the apparatus uses 11 photodiodes vertically across the screen, $\frac{1}{11}$ of the scanout is still 0.63 ms at 144 Hz, and assuming a randomized first response, the additional delay in the measurement will be half of that. Since the photodiodes don't take a perfect one pixel sample, the attributable portion of this effect is more likely to be in the range of 0.1–0.3 ms. The remaining difference doesn't have a certain cause. I assume that it is caused by the human analysis in the camera based method, the camera's rolling shutter, and random variance.

In the most stable scenario – which will be considered the baseline from now on – the game runs at 1000 fps. The measurement shows an extremely low end-to-end latency (excluding scanout) of 3.78 ms, with a standard deviation of 0.35 ms. The

[10] How an external input device can be used with the testing device is explained in section 4.1.4.

same measurements were done at a multitude of graphics settings and resolutions to create different loads. Additionally, some samples were collected using the camera based method on the real Rocket League map used in the experiment, in order to compare. This was all done for the next step, which uses the data to calibrate the estimation.

### 4.1.3 Estimating game latency

This section shows how to put the theory together to estimate the average latency. Collected for that purpose, were the following averages:

- frame duration ($t_{\text{frame}}$)
- CPU game logic processing time ($t_{\text{game}}$)
- CPU render processing time ($t_{\text{render}}$)
- GPU render processing time ($t_{\text{GPU}}$)
- frame time lower bound ($t_{\text{cap}}$)

I found a constant not easily attributable 0.7 ms of latency in my measurements. I assume this to be driver or Windows related latency. The first part of the latency that I have an estimate for is where the input has to wait to be captured by the next frame $\frac{1}{2}t_{\text{frame}}$. Then the game logic gets calculated $t_{\text{game}}$[11]. The rendering is both dependent on the CPU and GPU. There is a large overlap though, and the best approximation was simply using $t_{\text{GPU}}$ for their combined time. The most difficult part is detecting if frames are getting queued. The situation where the buffer fills is when the GPU is the bottleneck. This was reasonably predictable based on the collected data. When the computer isn't reaching the set frame limit $t_{\text{cap}}$ and $t_{\text{game}} < 0.95 \cdot t_{\text{GPU}}$, the GPU is assumed to be the bottleneck. On the local computer, this rule has always correctly identified the situation in a variety of tests at low and high resolutions and graphics settings with different frame limits. However, when capping the frame rate at about the limit of what the GPU can produce, then inconsistent frame times lead to the buffer being neither full nor empty. Then, the prediction can be off by a full frame time. Lastly, anyone running the game at above 120 fps is affected by the latency caused by the limited physics tick rate $t_{\text{tick}}$ (see section 4.1.1.3).

[11] In the data gathered from RL, $t_{\text{game}}$ includes $t_{\text{input}}$.

Everything combined gives the equation in 2 parts:

$$t_{\text{queue}} = \begin{cases} (1 + \texttt{OneFrameThreadLag}) \cdot t_{\text{frame}} - t_{\text{game}} & \text{if } \frac{t_{\text{cap}}}{t_{\text{frame}}} < 0.98 \wedge \frac{t_{\text{game}}}{t_{\text{GPU}}} < 0.95 \\ 0 & \text{else} \end{cases} \tag{4.1}$$

$$t_{\text{PC}} = 0.7\,\text{ms} + 0.5t_{\text{frame}} + t_{\text{game}} + t_{\text{tick}} + t_{\text{GPU}} + t_{\text{queue}} \tag{4.2}$$

When VSync is enabled, the formula has to be adapted. If the user doesn't manually cap the frame rate below the refresh rate, the buffer will fill up. Additionally, after finishing the render, the GPU still has to wait until the next vertical blanking interval to display the frame (as explained in section 4.1.1.3. Therefore, $t_{\text{queue}}$ has to be adapted the following way:

$$t_{\text{v\_queue}} = (1 + \texttt{OneFrameThreadLag}) \cdot t_{\text{frame}} - t_{\text{game}} + t_{\text{refresh}} - t_{\text{GPU}} \tag{4.3}$$

If the user does use the frame limiter, the queue will only fill under the same conditions as with VSync off. Without an adaptive sync monitor, there is still an average waiting period of half a refresh to align the page flip with the next vertical blank.

$$t_{\text{vcap\_queue}} = t_{\text{queue}} + \begin{cases} 0 & \text{if Adaptive Sync} \\ 0.5t_{\text{refresh}} & \text{else} \end{cases} \tag{4.4}$$

$t_{\text{v\_queue}}$ and $t_{\text{vcap\_queue}}$ take the place of $t_{\text{queue}}$ in equation (4.2) when the participant's settings show they are on. They were listed separately to keep the equation readable.

### 4.1.4 Estimating input device latency

The latency of input devices has been investigated many times in the past. The sources which have high quality data for some of the game controllers used by the participants are Petit [www15] and Wimmer et al. [85]. However, neither testing method involves Windows and Rocket League, which are factors that could introduce input device dependent software latencies that the methods would not account for. By using my own testing method, I can make sure there are no such effects, and also test more of the controllers used by the participants.

In order to adapt the Arduino based testing method for controllers, one change has to be made. Instead of the Arduino acting as a game controller and sending inputs directly to the PC, it uses the digital I/O to flip the state of a button or analog axis on the controller to be tested. This requires opening up the controller and sometimes soldering in order to attach wires to the internal circuits. The process is the same as described by Petit [www15] and Wimmer et al. [85]. The difference to those methods is that the response will get measured on screen. The measurement includes all the PC and display latency. In order to know which part of the latency is caused by the controller, I compare the most stable baseline test to the same scenario with the controller in the chain. The difference is caused by the controller.

Using this method, 79 % of the participants' controllers were measured. For all controllers also tested by Petit [www15] and Wimmer et al. [85] the measurement results are all within 0.5 ms. This validates the method and shows no signs of the controllers behaving differently on Windows and Rocket League.

In order to determine the controller used by the participant, input device information was read out via the `hidsdi.h` header [100] and listed to them in the post-experiment survey. They were then told to pick which device they used, or note down the correct name if it did not show up on the list. When analyzing the data post experiment, the device and connection type was manually identified based on the product ID, manufacturer ID, and provided name.

### 4.1.5 Estimating display input latency

Unlike the controllers, there were a lot more different monitors in use by the participants. Testing them all personally is not feasible and thankfully, there are several monitor reviewers with high quality latency data.

Similar to the process with the controllers, monitor information was read out from the WMI Core Provider [101] and listed to the participant. If they did not see their model, they could provide their own name. The manual identification of the monitors was a lot more involved. Unfortunately, the information stored in the *Extended Display Identification Data* (EDID) of the displays is not always unique for some manufacturers. When identifying a singular model was impossible, the participant's data was not used in the dataset with latency estimation. I cannot guarantee with certainty that no monitor was identified incorrectly. The monitor and controller data along with their identified names are public for the purposes of verification [www25].

Measuring latency can be done in many ways as discussed in section 2.1. Those methods may result in different measurements, which is why those differences need to be accounted for if any comparison is to be attempted. rtings.com, a reviewer with a large database of monitors uses a measuring method very similar to mine [www26]. There are only two differences. The measurement is done in a custom program instead of Rocket League, and the measurement area covers only a single spot in the center of the screen. The former should only improve consistency and the latter can be easily accounted for. It means that the average scanout latency is included in the measurement, which is part of the total latency anyway. When comparing three monitors available to my testing, they match to under 0.3 ms with the expected results found by rtings.com. Due to having the largest high quality sample size of monitor latencies measurements, the site was used as the primary resource whenever a specific monitor was in their dataset.

The Leo Bodnar LagTester [www27] is another testing device with very similar capabilities. It does not need a computer, but it only works at 60 Hz refresh rates. rtings.com used the device prior to creating their own, and stated that the results were within 1 ms of their new method. I found 24 monitors to compare between rtings.com and the Leo Bodnar method. The difference between the two is at most 1.83 ms, on average 0.51 ms, and the standard deviation is 0.72 ms. This is relatively minor, especially considering that the majority of Leo Bodnar based results were rounded to the nearest millisecond. For the estimation purposes, I do not assume that single millisecond accuracy is possible, and consider this variance acceptable. The sources for Leo Bodnar based data were displaylag.com, pcmag.com, and prad.de.

Hardware Unboxed updated their testing in 2019 to a method very similar to rtings.com [www28]. In a 22 monitor comparison with rtings.com, the difference is at most 1.98 ms, on average 0.50 ms, and the standard deviation is 0.71 ms. I use this source on the same criteria stated previous paragraph.

tftcentral.co.uk and pcmonitors.info use a measuring method called SMTT 2.0 [www29]. The method works by displaying timer information on two monitors at once. The timers are placed across the entire vertical range of the screen. The difference between the two newest timers on each screen is the difference in latency between the two monitors. This can be determined with a photo camera with a global or fast rolling shutter. By comparing to a known display latency such as that of a CRT, the absolute latency can be determined. Unlike the previous methods, scanout latency has a negligible effect on the measurement. The effect of pixel response times is dependent on the human looking at the picture of the timers. In order to account for these differences, I used linear regression on 21 and 19 monitors

found on both the respective sites and rtings.com. The linear regression uses the duration of a refresh ($t_{\text{refresh}}$) as the one independent variable. After correcting with the linear regression, the deviations are at most 3.84 and 2.06 ms, on average 1.00 and 0.60 ms, and the standard deviations are 1.41 and 0.89 ms.

More sources were considered. They were disregarded either because of an intransparent or inaccurate testing method.

### 4.1.6 Total latency estimation

The idea of the total latency estimation is very simple. It is simply given by adding the three parts of latency together.

$$t_{\text{total}} = t_{\text{controller}} + t_{\text{PC}} + t_{\text{monitor}} \tag{4.5}$$

The accuracy of the estimation is expected to be limited, as shown by e. g. the monitor alone being off by over 3 ms. There is also a recently discovered bug that can cause 4 ms of additional latency on select controller mainboard combinations. A firmware change in a controller or monitor happens rather infrequently but can potentially change the latency too. The biggest inaccuracy possible would be due to frames queuing up and the estimation not predicting this. However, in the experiment, all participants with a frame rate below a limit had to be removed in order for the method of added artificial latency to function correctly. This happens to also limit the extent to which predictions may be wrong. I cannot guarantee that there is not a single result with 15 ms of inaccuracy in the dataset. However, I do not expect this to be common. In the large sample size, this noise is likely not important.

In order to validate the estimation method, I reached out to participants who signed up for a newsletter regarding the experiment. I asked them whether they had a high frame rate camera (common in iPhones and high-end Android phones) and were willing to help me gather extra data. The idea was to compare the estimated latency to real measurements. The method involved them recording their screen and controller while pressing a button. This is an inaccurate version of Rejhon's [www14] method, as it is difficult to see when the button activates without an LED. I analyzed the videos myself to minimize between participant bias. There were 10 volunteers, but unfortunately only three had an estimated latency in the dataset. At first, one of the videos showed 40 ms higher than expected latency. I was able to identify that the participant had turned on VSync since the experiment. Once

they had disabled the setting, the estimate was off by at most $5.5 \pm 0.9$ ms. When trying the same measuring method on my own computer, I found that I also got 4 ms less latency than my previous measurements. This means it was most likely an inaccuracy of the measurement method. Furthermore, I asked the helpers to test at both low and high graphics settings like in the experiment. That way the difference between the two estimations should be the same as the difference between the two measurements regardless of the constant error of the method. The biggest error between the expected and measured difference was $2.8 \pm 1.2$ ms. This is not a large enough sample size for it to accurately predict the error of the estimation, but it demonstrated that a change like the VSync setting can be caught.

## 4.2 Experiment implementation

The experiment was implemented as a BakkesMod[12] plugin using C++. The user interface was created with *Dear ImGui* [www30] as an overlay on top of the game. Some aspects of the modding process required reverse engineered proprietary game code and data structures. Based on prior unrelated communication, it is against the will of the developers to have this information public. Therefore, the implementation can't be publicized and there will only be portions of the implementation explained.

### 4.2.1 Adding artificial latency

Adding artificial latency can be done at any step of the input-output chain. Since I don't have access to the participants monitors and controllers, it has to be done in software. There are multiple options to introduce the delay here. Options at the driver level of the inputs or frame buffers may be possible but unnecessarily complicated when modding the game directly is possible.

One way to delay inputs is frame by frame; though, if the frame times are for example 10 ms, and 15 ms of latency is supposed to be added, then neither one nor two frames of delay would result in the desired latency. While it is possible to match an average of 15 ms by randomly switching between one and two frames with a 50 % chance of each, this would have an impact on the distribution of the latency. The thesis only focuses on average latency, as estimation of the exact distribution

[12] User created Rocket League mod with an open API to allow developers to create their own mods for the game

would be impossible. However, for the purposes of the added latency, which is the most important factor analyzed in the experiment, it is preferable that it does shift the entire distribution of possible latencies without changing it. That ensures that any measured effect isn't actually the effect of changing variance, but that of changed average latency. The way to achieve this feat is through the way Rocket League handles physics with a fixed time step. By delaying inputs by an integer amount of physics steps, the additional delay is always constant, and the distribution is conserved at any frame rate $\geq 120$ fps. The reason for this lower bound is that the physics simulation runs at rate of 120 steps per second. When the frame rate is below that value, multiple physics steps will be run in a single frame and use the same input from that frame. This might then result in no additional delay at all or less delay than intended. Therefore, submissions had to be removed due to not reaching the average required frame rate (see chapter 5).

The code in listing 4.2 demonstrates how the inputs get stored in a FIFO queue and replayed when the amount of integer steps of desired latency, given by `current_lat`, is reached.

```
1  std::queue<ControllerInput> store;
2
3  void OnSetVehicleInput(CarWrapper caller, void* params, std::string eventName)
4  {
5    ControllerInput* input = static_cast<ControllerInput*>(params);
6    store.push(ControllerInput(*input));
7    while (store.size() > current_lat) // clears itself if it's too full for the
         ↪ setting
8    {
9      *input = store.front();
10     store.pop();
11   }
12 }
```

**Listing 4.2:** How artificial latency was added in a function that manipulates the inputs before each physics step (C++).

### 4.2.2 Score evaluation

An important part of evaluating player performance is measuring how good a shot. A single stat like distance to the goal only values a single aspect of the shot. A weak shot would be easier to save for a defender, but this will only matter if the ball is even shot on target. I created a compound metric called `score` in order to have a better numerical estimate of how good a shot is. The first part of the equation is simply about not touching the ball at all. During the experiment, whenever the player does not make contact with the ball, the shot is reset in order to not have

missing data for shot speed and accuracy. Every miss of the ball subtracts 2 points from the score (`misses`). In order to give points based on the precision of the shot, I use a variation of the actual difference of the ball ($\vec{b}$) to the center of the goal ($\vec{g}$).

$$\texttt{dist}_{\text{mod}} = \left\| \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{pmatrix} \left( \vec{b} - \vec{g} \right) \right\| \tag{4.6}$$

The vertical distance is halved before calculating the magnitude of the vector. This is done because even though the goal may be missed by shooting too high, there is a backboard which bounces it back to the team on offense. Helped by gravity, it often leads to beneficial situations where a goal can be scored regardless. In order to give higher points for smaller distances, the inverse of the distance is used and clamped to a minimum of 50 to prevent the value from going to infinity. The minimum is an educated guess for an upper limit of what is humanly possible in terms of precision. Anything more precise is likely just random variation. In order to get less extreme values, the logarithm is applied. The maximum distance of 2500, at which the addition to the score becomes 0, is 2.8 times the distance from the center of the goal to the goal post [`www`31]. This will be so far off that the ball is no longer in a scorable position after bouncing off the backboard.

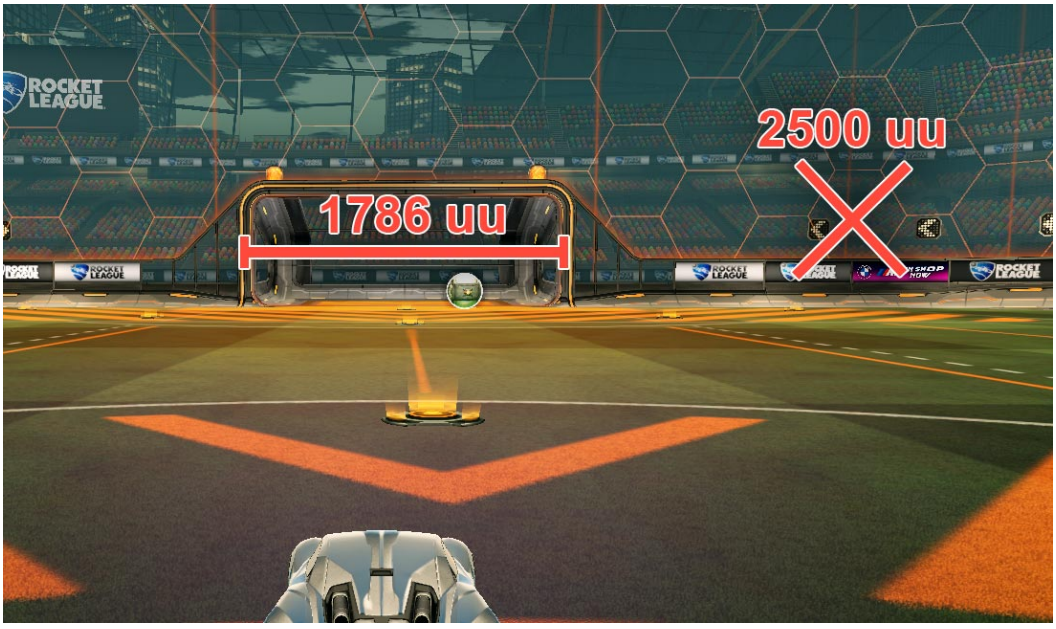


Fig. 4.2.: Graphic showing the size of the goal and the distance at which the score is 0.

$$\texttt{score} = \left( \sum_{\texttt{misses}} -2 \right) + \log_2 \left( \frac{2500}{\max\left(2500, \min\left(\texttt{dist}_{\text{mod}}, 50\right)\right)} \right) + \texttt{goal} \cdot \left( 1 + \frac{\max\left(0, v_y - 2000\right)}{600} \right) \tag{4.7}$$

The remaining factor is only active if the shot attempt results in a `goal`. This gives one point by default, and additional points based on the ball velocity in the direction of the goal ($v_y$). There is a minimum velocity to gain points because the save for the defender will not be any more difficult depending on whether the ball is rolling slow or very slow.

### 4.2.3 Miscellaneous

Due to changing the participant's graphics and BakkesMod settings during the experiment, they have to be backed up safely prior, and restored once the experiment is finished, aborted, or the game crashes. When starting, the settings are stored as plain text files as to allow manual restoration of settings in case the automatic part fails. After the files are created, there is a three-second timer before the experiment starts. This is an extra precaution against antivirus deletions or unexpected corruption of the files. At the end of the timer, the files get reread from disk, and using a hash function, it is assured that they are still correct. Only then, can the experiment actually begin safely. After the files are used to restore the settings, they get deleted. In case of a crash during the experiment, the files will still exist when the game gets restarted, and the settings will get restored.

In order to collect the data from the experiment, I use a virtual server in a datacenter to maximize uptime. The server listens on one port and stores the content of every post request in a file. At the end of the experiment, the client gathers all the relevant data into a string buffer in the JSON format. This buffer gets encrypted using the *OpenSSL* library envelope ( `evp.h` ) asymmetric encryption, using AES-256 for the symmetric part of the encryption. The private key for the data is not stored on the server or anywhere online. It is only used offline to decrypt the data after downloading it from the server. Using the *libcurl* library, the client establishes an SSL connection to the server and sends the already encrypted information in a post request. The server stores no additional information.

# 5 Results

In the period of 28th August–15th September 2020 where the experiment was open, there were 763 successful submissions. 16 were second submissions from the same people, and therefore removed. For the remaining 747, the monitor and controller data was exported, manually identified, and annotated. 666 used a controller as the input device, of which 587 were successfully identified and had available latency data. Independently, for 352 of the participants, monitors were identified and latency data obtained. The cross-section of valid controller and monitor data leaves 285 participants. Further 33 participants were removed for one of the following reasons:

- The VSync method was not able to be determined. Their data showed that VSync was not limiting the frame rate despite being activated in the settings, which can either mean it was forced off or set to Fast/Enhanced Sync in the driver.
- They had a monitor with an aspect ratio other than 16 : 9 or 16 : 10. It was later determined that aspect ratio might have a significant effect on the planned FOV tests.
- The frame rate was too low,[1] which makes the method used to add artificial latency behave differently.
- They changed the graphics settings during the experiment, despite being told not to do so.
- They reported in the survey that they were sitting 5 and 8 m away from a small monitor, which was assumed to be a mistake.
- They had no valid skill rating.

The valid dataset with baseline latency estimation is therefore of size 252. When disregarding the baseline latency and monitor related FOV, the valid dataset is 635 submissions large.

[1] The reason is explained in section 4.2.1

The chosen significance level is $\alpha = .05$. All $\pm$ intervals and error bars reported are 95 % confidence intervals. The effect size for the ANOVA is given by $\eta_p^2$, and for pairwise comparisons Cohen's $d$ is reported.

## 5.1 Participants

The participants geometric mean age is 23 y, and the standard deviation is 5.74 y. 2 participants chose female, 3 chose other, 4 did not answer, and the remaining 246 chose male. This is not dissimilar to the analytics of the YouTube channel on which the experiment was advertised.

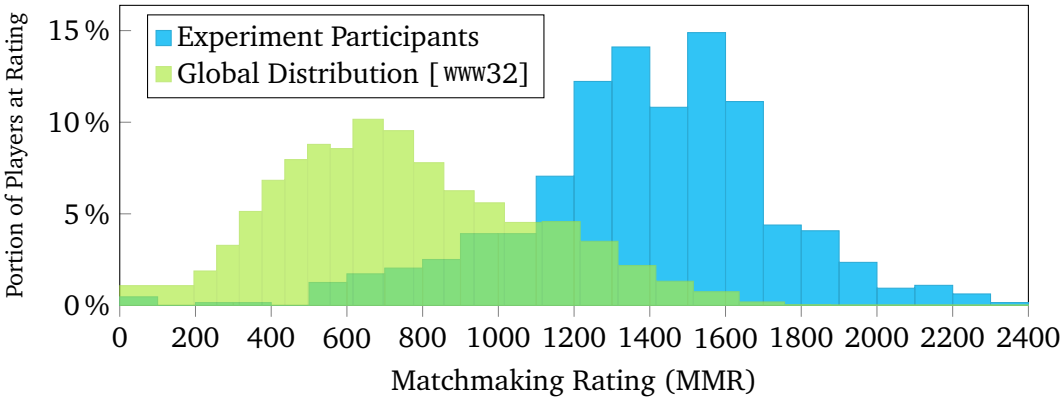


**Fig. 5.1.:** Histogram of the *matchmaking rating* (MMR) of the participants compared to the general playerbase of the game. The MMR serves as a good approximation of the skill of the participants.

The participants are highly experienced in Rocket League according to the self-reported number of hours played. The geometric mean is 2000 h with a standard deviation of 1604 h. The values lie between 100 and 9000 h. For this experience, the participants are significantly above average in skill, achieving a mean rating of 1455 in RL's ranked match making (311 standard deviation). This puts the average participant in the top 2 % of the playerbase [www32]. The average player of Rocket League is about three standard deviations below the average participant. 9 participants self-reported as professional players. In order to utilize the player skill as a fixed factor in the repeated measures model, the variable was split into 5 groups with an MMR range of 360 each (`skill_group`). The lowest group is open to $(-\infty, 900]$ rating and contains players up to the top 75th *percentile* (pctl.) of the general playerbase, matching the average RL player's skill. The other groups are estimated to therefore contain only players of the following percentiles and above: 75th, 93rd, 99.5th and 99.99th. The geometric mean hours of RL experience for each of the categories are 589, 1156, 1981, 3365, and 5939 h.

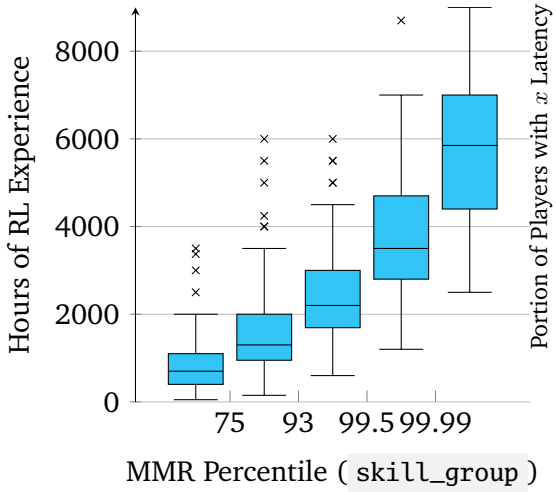


Fig. 5.2.: Hours of experience by skill group.

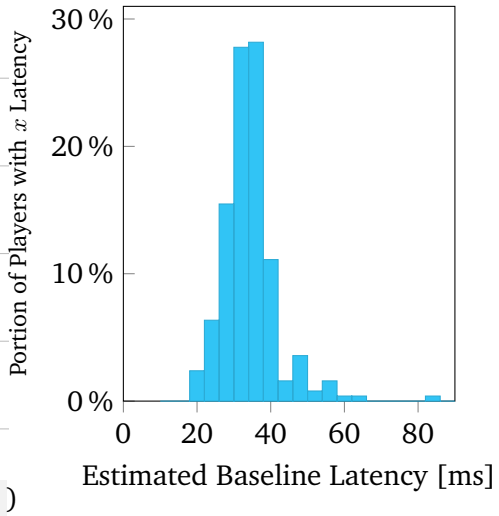


Fig. 5.3.: Histogram of the estimated average baseline end-to-end latency of the participants.

The estimated baseline end-to-end latency for the participants has a geometric mean of 33.43 ms and a standard deviation of 7.62 ms. The values range from 18.31 to 83.68 ms. The continuous variable is again converted into a fixed factor ( `base_lat_group` ), each grouping spanning 8.33 ms in order to match the smallest step size of the added latency category. Latency of 43 ms and up are combined into one group, as there are very few datapoints above. This group has a geometric mean estimated latency of 52.17 ms.

The horizontal field of view is calculated based on monitor size and self-reported seating distance. The participants' geometric mean FOV is 51.31° with a standard deviation of 15.17°. They are split into two groups ( `peripheral_vision` ). One which only has the monitor within the near peripheral vision (<60°), and the other where the edges of the monitor reach into the mid peripheral vision. This means that the hypothesis H6 has to be narrowed as it can only be tested for the mid peripheral vision and not the far peripheral vision.

The larger dataset without latency estimation is very similar. Regarding gender, 13 gave no answer, 6 chose other, 5 chose female, and 611 chose male. The geometric mean age is 22 y and a standard deviation of 5.86 y. The geometric mean hours of experience are lower at 1787 h, which is mostly due to the lowest skill group quadrupling in size. Within the skill groups, the experience is only different for the highest group. The geometric mean hours for the skill groups are 553, 1161, 1956, 3232, and 5524 h.

## 5.2 Effects on performance

I ran a *General Linear Model* (GLM) repeated measures doubly multivariate ANOVA with *IBM SPSS Statistics 27* (SPSS). The response variables were `score`, `dist_to_goal`, `shot_vel`, `goal_scored`, `ball_misses`, and `ball_touches`. `dist_to_goal` measures the distance between the ball and the center of the goal at the point where it is closest to the goal line. The lower the distance is, the more precise the player was. `shot_vel` measures the velocity of the ball at the same time as `dist_to_goal`. `goal_scored` is 1 when the shot resulted in a goal, and 0 otherwise. `ball_misses` gives the number of times the player has not made contact with the ball, in which case the shot is restarted to avoid missing data. `ball_touches` gives the number of touches the player had on the ball in the attempt. `score` is a custom compound metric created to approximate how effective the shot would be in an actual game (see section 4.2.2). The within-subjects model is `graphics`, `added_lat`, `shot`, `graphics*added_lat`, `shot*added_lat`. The `graphics` variable has three possible values. One which uses the user's own graphics settings, one which turns every effect to the minimum, and one which turns every effect to the maximum. `added_lat` refers to the five added latency conditions 0.00, 8.33, 16.67, 33.33, and 50.00 ms. The `shot` variable gives one of the five shots described in appendix A.1. The between-subjects model is `skill_group`, `peripheral_vision`, `base_lat_group`. The variables and groupings are explained in the previous section (5.1). All interactions of the within- and between-subjects model are automatically included in the analysis by SPSS.

In the multivariate results, the factors `base_lat_group` and `peripheral_vision` show no significant effects on their own, nor with any of the interactions with the within-subjects model. Therefore, I reject hypothesis H6 for the performance response variables. There is no evidence that the mid peripheral vision has an interaction effect with latency on player performance. Since all effects that require monitor data and estimated latency are not significant, the larger dataset can be used to analyze the other factors. Significant differences to the smaller dataset will be noted. The sample size for each of the skill groups subsequently becomes 50, 143, 303, 118, and 21.

For the multivariate results of the larger dataset, the random subject intercept has a large effect ($F(6, 625) = 136830.73, p < .001, \eta_p^2 = .999$). The `skill_group` also has a significant effect on the player's performance ($F(24, 2512) = 19.00, p < .001, \eta_p^2 = .154$). The within-subjects main effects are all significant ($p < 0.001$).[2]

[2] `graphics` as a main effect and interaction is not significant in the smaller dataset ($p \geq .097$).

Their interactions with the `skill_group` was too ($p \leq .012$). The interaction between `graphics` and `added_lat` is not significant ($F(48, 583) = .866, p = .727$). I therefore reject hypothesis H5 for the performance response of players. There are significant interaction effects between the three main within-subjects factors and the player skill group. The variables `added_lat` and `shot` have a significant interaction effect, suggesting that the type of shot makes a difference in how the player is affected by latency. All three-way interaction effects are not significant.

Post-hoc univariate analysis shows that there is a significant effect of `skill_group` on every response variable tested with varying effect sizes. Pairwise comparison using Games–Howell test shows all groups to be significantly different from one another ($p \leq .001$), except the highest two pairs on all metrics but the touches on the ball. On this metric, the two lowest skill groups are significantly different from the two highest groups.

The sphericity assumption is violated by `shot` and `added_lat` effects for some response variables. It was corrected for with the *Greenhouse–Geisser* (G–G) correction and *Huynh–Feldt* (H–F) for when $\varepsilon < 0.75$ [102, p. 84].

In the post-hoc univariate ANOVAs the main effect of `added_lat` is significant for every response variable except `ball_touches`. The effect sizes $\eta_p^2$ are .122 for `score` and .120 for `shot_vel`. The other variables show a smaller effect at .072 for `dist_to_goal` and .041 for `goal_scored` and `ball_misses`. Using Bonferroni corrected pairwise comparisons shows that the results for the 0, 8.33, and 16.67 ms conditions do not significantly differ for any of the response variables. At 33.33 and 50.00 ms most pairwise comparisons are statistically significant.[3] In every case the increase of latency results in a degradation of performance (cf. table 5.1), and I consequently accept hypothesis H1.

The univariate results for the effect of `shot` are statistically significant for all response variables ($p < .001$). There are only three pairwise Bonferroni corrected comparisons that are not significant.[3] Therefore, the different shots chosen are all significantly different on most response variables.

The final main effect `graphics` is statistically significant on the response variables `ball_misses` ($F(2, 1260) = 13.72, p < .001, \eta_p^2 = .021$) and `score` ($F(2, 1260) = 4.54, p = .011, \eta_p^2 = .007$). Pairwise Bonferroni corrected comparisons show that for both response variables there is only a significant difference for the low graphics

[3] As expected, the smaller dataset has fewer comparisons with a significant difference. The trend is the same, but the sample size is smaller.

| Response Variable | Cohen's $d$ | |
|---|---|---|
| | 33 ms | 50 ms |
| score | -.077 | -.200 |
| dist_to_goal | .042 | .136 |
| shot_vel | -.058 | -.142 |
| goal_scored | -.029 | -.105 |
| ball_misses | .058 | .120 |

Tab. 5.1.: Effect size of added_lat for each response variable.

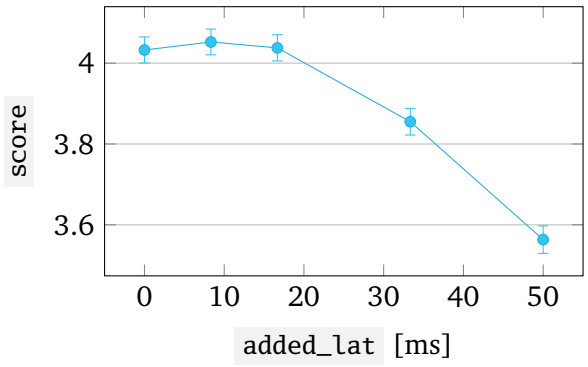


Fig. 5.4.: Average score lowers with additional latency. Error bars denote 95 % confidence interval.

condition compared to the user's default and high graphics. Players miss more at the low graphics settings and consequently score less points ($d = .035, -.018$).

The interaction effects between skill_group and added_lat is significant only for the response variable shot_vel ($F(16, 2520) = 3.10, p < .001, \eta_p^2 = .019$). The differences between the skill groups getting affected by 33.33 and 50.00 ms of added latency are shown in table 5.2 (cf. figure 5.5). The effect size of added_lat grows with increased skill. Therefore, I accept hypothesis H4 for the response variable shot_vel.

| | shot_vel | | | | | |
|---|---|---|---|---|---|---|
| skill_group | 33 ms | | | 50 ms | | |
| | $p$ | $d$ | % | $p$ | $d$ | % |
| **< 75th** | 1.000 | -.019 | -.718 | .034 | -.071 | -2.749 |
| **75th–93rd** | .367 | -.030 | -1.051 | <.001 | -.110 | -3.930 |
| **93rd–99.5th** | <.001 | -.068 | -2.291 | <.001 | -.158 | -5.306 |
| **99.5th–99.99th** | <.001 | -.076 | -2.440 | <.001 | -.168 | -5.444 |
| **> 99.99th** | .025 | -.108 | -3.390 | <.001 | -.211 | -6.526 |

Tab. 5.2.: Reduction of the average shot velocity when with added latency, split by the skill groups. Results given are the Bonferroni corrected $p$-values, the effect size as Cohen's $d$, and the percentage reduction compared to no added latency.

The interaction between shot and skill_group is also significant for all response variables ($p < .001$). The effect sizes $\eta_p^2$ are largest for score, dist_to_goal, and shot_vel: .166, .111, and .120. The biggest difference is on shot number 4, which high ranked players are by large margins more likely to touch, score, and shoot with high velocity. This is most evident in the compound metric score (cf. figure 5.6).

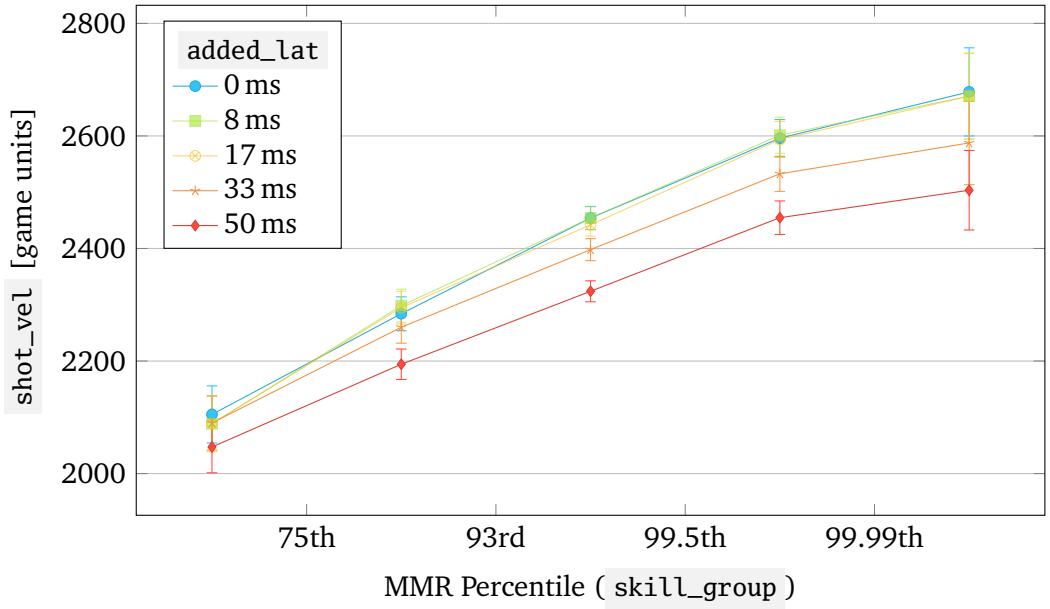


Fig. 5.5.: Average shot velocity is more affected by added latency as the player skill increases. Error bars denote 95 % confidence interval.

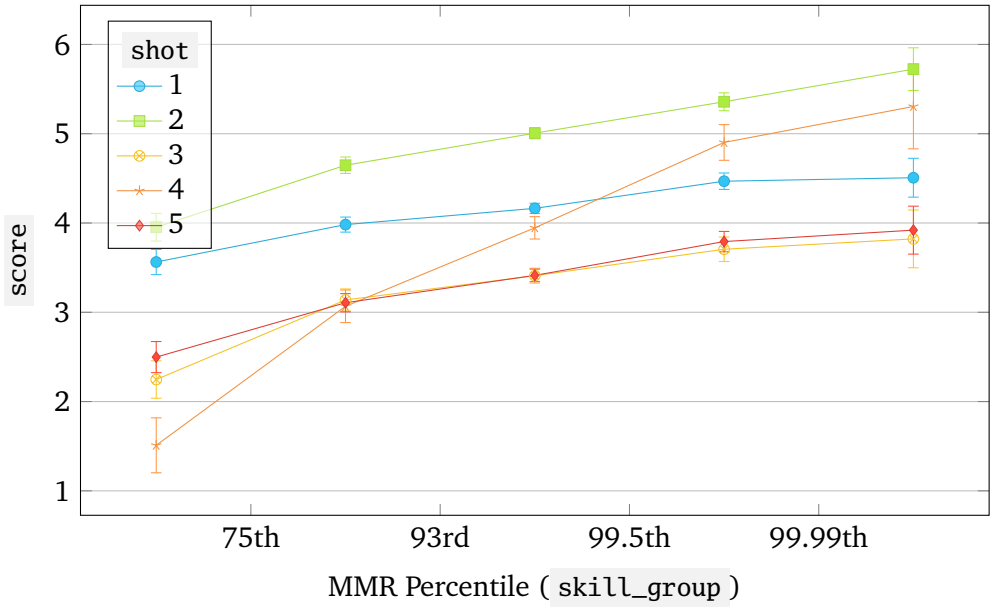


Fig. 5.6.: Player skill has a different effect on the average score of each shot. Error bars denote 95 % confidence interval.

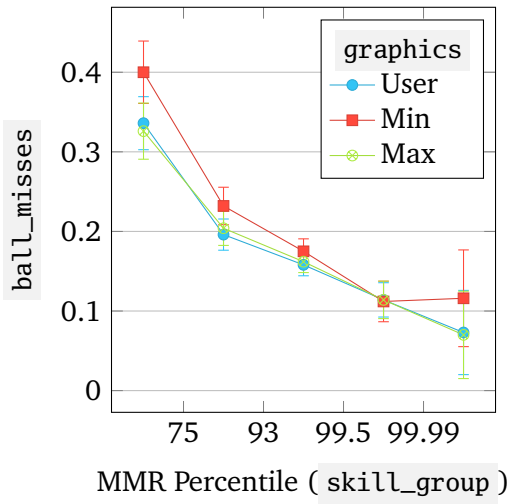


**Fig. 5.7.:** The two lowest skill groups miss the ball significantly more often with low graphics settings. There is no significant difference for the groups of higher skill. Error bars denote 95 % confidence interval.

There is a small but significant effect for the interaction between `graphics` and `skill_group` on the response variables `ball_misses` ($F(8, 1260) = 2.68, p = .006, \eta_p^2 = .017$) and `ball_touches` ($F(8, 1260) = 2.91, p = .003, \eta_p^2 = .018$). The trend shows low graphics increasing misses for lower ranked players, with the difference decreasing up until the second-highest skill group (cf. figure 5.7). The highest skill group does not follow this trend, but when using Bonferroni corrected pairwise comparisons the difference is not statistically significant ($p = .331, .235$) unlike the lowest two skill groups ($p \leq .015$). No trend is visible on the metric of `ball_touches`.

The interaction of `added_lat` and `shot` has a significant effect on all response variables ($p \leq .005, \eta_p^2 \leq .011$). Shot 5 is least affected by additional latency while the other shots each show the biggest difference depending on the response variable (cf. figure 5.8).

| Response Variable | Cohen's $d$ with 50 ms for Shot # | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| `score` | -.238 | -.376 | -.220 | -.194 | -.070 |
| `dist_to_goal` | .104 | .276 | .220 | .073 | .069 |
| `shot_vel` | -.345 | -.292 | -.181 | -.091 | -.200 |
| `goal_scored` | -.153 | -.207 | -.234 | -.031 | .021 |
| `ball_misses` | .118 | .067 | .076 | .246 | .061 |

**Tab. 5.3.:** Effect size (Cohen's $d$) varies greatly between shots, comparing 50 to 0 ms of added latency for each response variable.

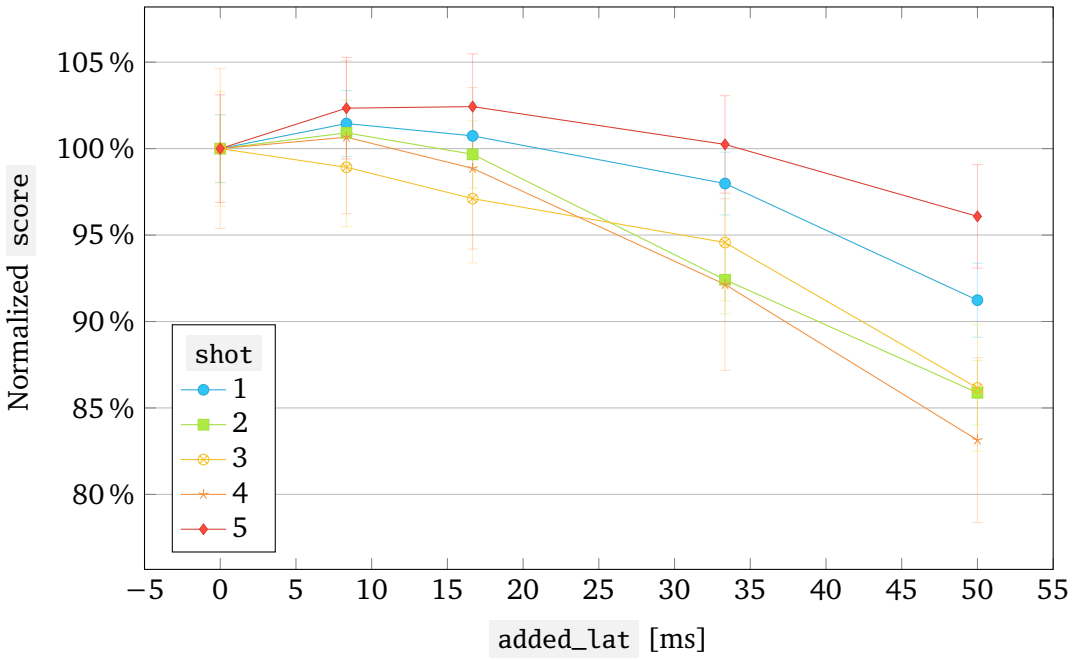


Fig. 5.8.: The normalized score demonstrates how the shots are affected differently by additional latency. Error bars denote 95 % confidence interval; faded for visibility.

## 5.3 Effects on perception

I ran a GLM repeated measures ANOVA with SPSS. The only response variable is `perceived_lat` which is the participant's response between 0 and 6 on the survey displayed after every set of five shots (see figure 3.1). The rest of the model is the same, with the exception that the dependent variable `shot` does not exist. Therefore, the within-subjects model is then `graphics`, `added_lat`, `graphics*added_lat`. The between-subjects model is `skill_group`, `peripheral_vision`, `base_lat_group`. All interactions of the within and between-subjects model are automatically included in the analysis by SPSS.

The factors `base_lat_group` and `peripheral_vision` show no significant effect on the perception of latency on their own, nor with any of the interactions with the within-subjects model. I consequently reject hypothesis H6 for the perception of latency in the mid peripheral vision. Since all effects that require monitor data and estimated latency are not significant, the larger dataset can be used to analyze the other factors. Significant differences to the smaller dataset will be noted. The sample size for each of the skill groups subsequently becomes 50, 143, 303, 118, and 21.

For the between-subjects effects, the random subject intercept has a large effect ($F(1, 630) = 1860.76, p < .001, \eta_p^2 = .747$). The `skill_group` also has a significant effect on how players perceive latency ($F(4, 630) = 16.38, p < .001, \eta_p^2 = .094$). Pairwise comparison using Games–Howell test shows all groups to be significantly different from one another ($p \leq .049$), except the lowest and highest two pairs.

The sphericity assumption was violated by all within-subjects effects. It was corrected for with the G–G correction.

For the within-subjects effects, `added_lat` has a statistically significant and strong effect on the perceived latency ($F(4, 2520) = 661.79, p < .001, \eta_p^2 = .512$). Thus, I accept hypothesis H2. Players notice when latency is added. Pairwise comparisons with Bonferroni correction show that all pairs are significantly different from one another ($p \leq .003$).[4]

The interaction between `added_lat` and `skill_group` is significant ($F(16, 2520) = 24.77, p < .001, \eta_p^2 = .136$). Pairwise comparisons with Bonferroni correction show that while the skill groups do not differ significantly at the baseline latency, once at least 16.67 ms of additional latency is present the two lowest skill groups give

[4] In the smaller dataset the 0.00, 8.33, and 16.67 ms conditions are not statistically significantly from each other. The trend is the same, but the sample size is smaller.

| skill_group | 0.00 ms | 8.33 ms | 16.67 ms | 33.33 ms | 50.00 ms |
|---|---|---|---|---|---|
| **< 75th** | 1.027 | 1.080 | 1.050 | 1.517 | 2.023 |
| **75th–93rd** | 1.097 | 1.219 | 1.346 | 1.802 | 2.599 |
| **93rd–99.5th** | 1.191 | 1.263 | 1.465 | 2.075 | 3.154 |
| **99.5th–99.99th** | 1.202 | 1.331 | 1.638 | 2.599 | 3.956 |
| **> 99.99th** | 1.167 | 1.524 | 1.579 | 2.897 | 4.206 |
| **Total** | 1.137 | 1.283 | 1.416 | 2.178 | 3.188 |

**Tab. 5.4.:** Average perceived latency answers (scale 0–6) with added latency, split by skill group.

significantly lower scores compared to the highest two cf. figure 5.9. The middle group is also different from the lowest group. At 33.33 ms and higher, the separation is bigger. All comparisons except the two lowest and highest pairs are significant. At 50 ms, the lowest two groups are also separatable. Therefore, I accept hypothesis H3. Experienced players are able to distinguish added latency better.

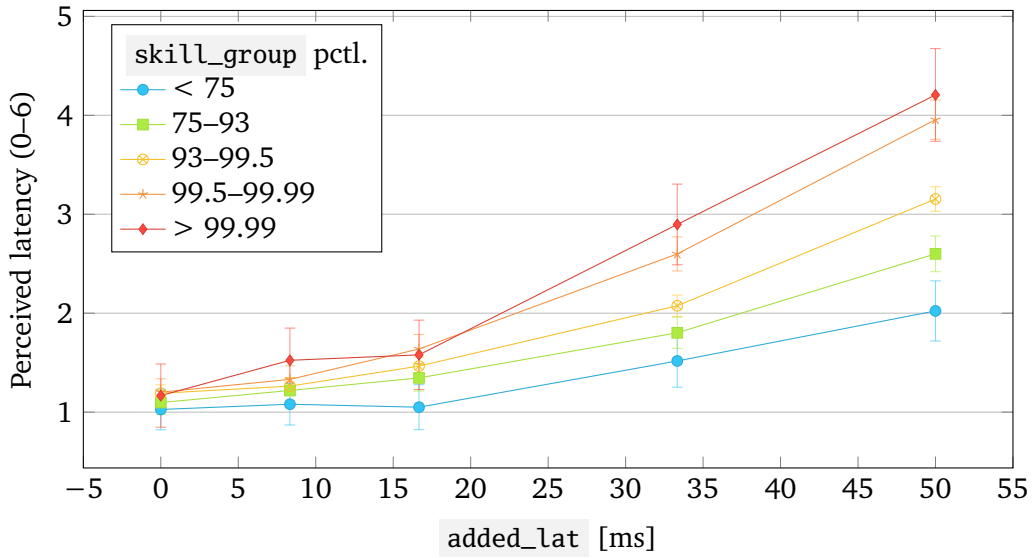


**Fig. 5.9.:** Average perceived latency increases the most with added latency for highly skilled players. Error bars denote 95 % confidence interval.

The factor of `graphics` has a small but statistically significant impact on the perceived latency ($F(2, 1260) = 12.74, p < .001, \eta_p^2 = .020$). Bonferroni corrected pairwise comparisons show that the graphics settings users usually play with are significantly lower than both the other options. The difference between minimum and maximum graphics settings is not significant ($p = .058$).

The interaction effect `graphics*added_lat` is not significant ($F(8, 5040) = 1.87, p = .063, \eta_p^2 = .003$). I reject hypothesis H5 for the perception of latency. There is no

evidence that players perceive added latency differently at different graphical effect densities.

All other interaction effects are also not significant (`graphics*skill_group`, `graphics*added_lat*skill_group`).

### 5.3.1 Just-noticeable difference

I calculate the JND to allow for comparisons to related works. In order to get the JND from the collected data on a scale, I compare the choice of the participant at a specific added latency to their choice at no added latency. If it is higher, the value is 1; if it is lower, the value is 0; if it is the same, the value is 0.5. Since every player was subjected to every scenario twice, the possible scores are 0.00, 0.25, 0.50, 0.75, and 1.00.

Since the experiment was only run at specific latency values, the exact 75 % JND threshold is unknown. I use linear interpolation between the two closest measurements to calculate an estimate. The average JND of the participants is approximately 38 ms.

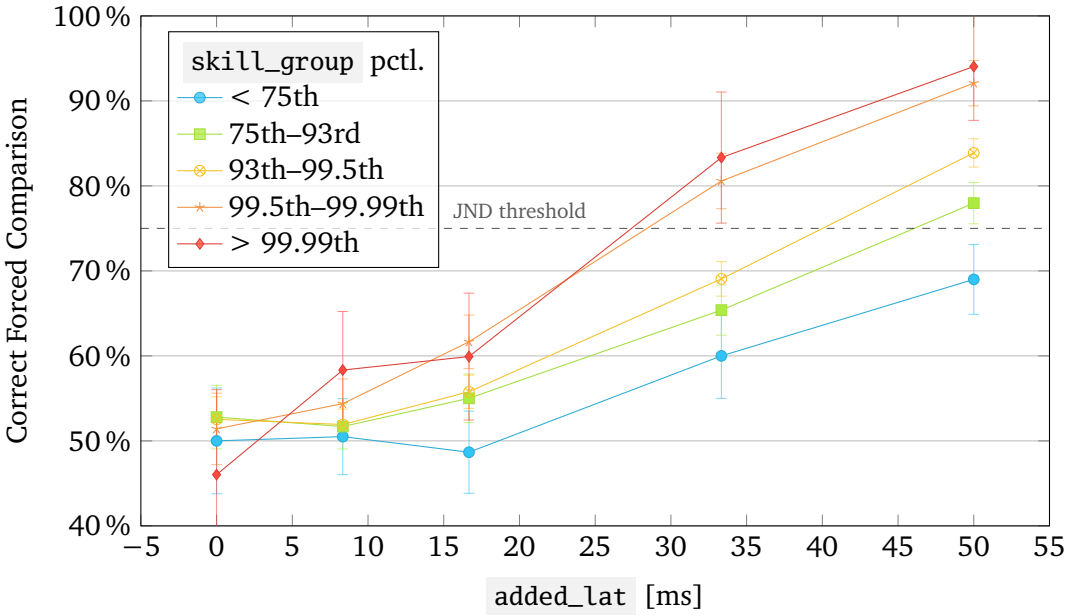


Fig. 5.10.: Comparing the rate of answers that was higher than the answers given at 0 ms. 50 % expected if answers are random. Highly skilled players are able to recognize added latency much more accurately. Error bars denote 95 % confidence interval.

Player skill has a significant effect on the perception of latency. I therefore report the best approximation for each skill group. The lowest skill group of the participants differentiated 50 ms of additional latency 69 % of the time. As 50 ms is the highest condition measured, this group has an unknown JND above 50 ms. For the other groups, the best estimates in order from low to high skill are 46, 40, 28, and 27 ms. Based on Bonferroni corrected pairwise comparisons at each level of added latency, the two highest rank groups show no significant difference, while all others do.

## 5.4 Observational results

Participants were not specifically asked to report on their experience, and since it was not possible to directly observe them, there was no further feedback from the majority of participants beyond the experiment data. Regardless, there were some unprompted responses on social media and email, the main ideas of which are listed here.

The most common topic was the perception of the latency. Participants said they were not able to judge it as well as they may have in other situations due to the constant switching of latency. One participant stated, “I did realize that the amount of lag from the prior 5 shots had a significant effect on how my brain felt about the current 5. For example, after 5 shots of very bad lag, the next 5 would seem so much better that I’d sometimes choose "No Lag", even if I was torn between choosing that or a "1".” The lack of a constant 0 ms reference to compare to was mentioned. Some also related that they didn’t rate the latency as high on the scale because they “didn’t know if it would get any worse.”

There were similar comments about the player performance, e. g., “I could get 5 out of 5 goals completely in the center with good power with horrible input lag. But then when I got no lag right after the car would be too fast, and I missed all of the shots.”

Another factor that players considered a limiting factor to their ability to judge latency was the tasks chosen. Dribbling the ball was suggested many times as a better alternative, a task during which the car makes constant contact with the ball. “Small and constant adjustments give you a good idea of changes in controls [. . . ].”

Three people mentioned that their ability to score the five shots of the experiment drastically improved over the 160 iterations. One suggested at least 100 shots of warm up.

A few participants reported mild discomfort and frustration from having to play with the additional latency. One said, “honestly i felt sick after playing with the really high input lag.”

There was concern about the self-reported hours of play time. Steam shows hours played for each game, but this includes any time the game is opened in the background or being used for non-playing purposes. This may vary significantly between players. In-game statistics for RL matches played do exist, but they do not include training time which is also experience.

One player with 30 h of experience said the test is too hard for new players. They said they needed up to 20 tries to make contact with the ball on some shots.

# 6 Discussion

## 6.1 Participants

Almost all participants were male and most were young adults. Prior academic research did not show major differences between genders and age groups. More research may be needed.

The participants were vastly above average in skill. This has to be kept in mind when examining the average results of the experiment. They do not represent the average RL player. The average RL player is represented in the experiment as the lowest skill grouping, while below average players are not represented at all.

## 6.2 Player performance

Before discussing the results, there is a most important limitation that has to be mentioned. The experiment was done on five shots that are supposed to be representative of shooting skills on offense. They can never represent a player's capability at dribbles, saves, other mechanics, or smart decision-making. Furthermore, it is not clear how much added latency overestimates the impact of latency in the real world, where players get used to their latency for very long periods.

The first goal of the experiment was to replicate previous findings showing that additional latency reduces performance in video games (H1). The results show a clear drop in player performance at 33.33 and 50.00 ms of added latency. The effect size is small (cf. table 5.1) compared to Martens et al. [5], however, this may at least partially be due to the chosen task. The effect size of each of the bordering skill groups is similar to 50 ms of added latency. Therefore, a player of the middle skill group with 50 ms added latency gets slightly worse scores on average than a player of the second-lowest group with no added latency. In the case of the highest skill group, 33 ms are enough to equalize them with the second-highest group. Considering the fact that players require about 800 and 2300 h more RL experience in order to make up for the difference caused by latency, the effect is anything but negligible. I assume that the small effect size is due to large amount

of variance humans show from attempt to attempt (cf. figure 6.1). There is also a large difference between the five shots that may play a role in this. The effect size of `added_lat` on `score` is $d = -.376$ on shot 2, but it is $d = -.070$ on shot 5. The most obvious difference , is that it is intended as an aerial shot for the higher skilled players. This means they won't be dodging, which is a highly timing dependent mechanic. The experiment does not show a statistically significant degradation of performance with 8.33 and 16.67 ms for any of the response variables. This may also be a result of the variance of the task. The alternative hypothesis is based on reports from section 5.4 (Observational results). The constant switching of latency may reduce the participants' performance whenever the latency changes by a large amount, which would disproportionately reduce the score for the 0 and 50 ms conditions. As there is no statistically significant difference, this is only a hypothesis that is covered separately in section 6.4.

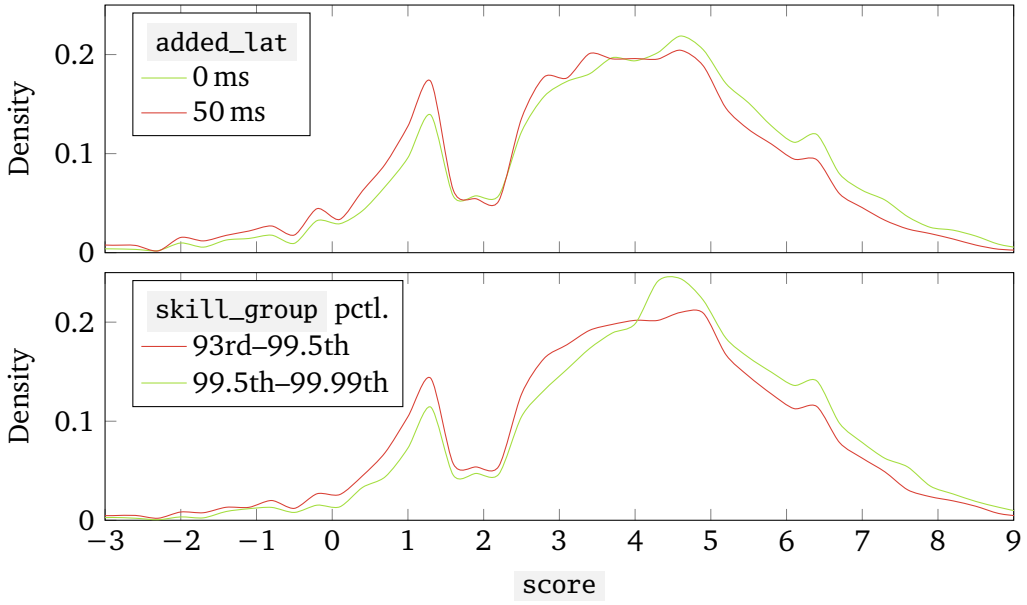


**Fig. 6.1.:** Density plots of the score of individual shots show that there is large variance. Neighboring skill groups reveal a similar offset to additional latency of 50 ms.

The next aim of the experiment was to test whether players of higher skill are more affected by added latency (H4). The results confirm the hypothesis for the response variable `shot_vel`. For the highest skill group, 50 ms of added latency drops the shot velocity by 6.5 %, compared to 2.7 % in the lowest skill group. That, along with the fact that higher level players are more consistent, means that the effect size of latency is three times higher. At the highest skill level, 33 ms of added latency hurt shot velocity by more than 2300 h of practice difference. This is not the case for the skill group that represents the average RL player. All other response variables showed no significant interaction between `skill_group` and `added_lat`.

Although not significant, there is a visible reverse trend on the response variables `goal_scored` and `ball_misses`. This is not necessarily opposed to the hypothesis, as the same shots will be easier for the highest rank players. On shot 4, they missed the ball on average .178 times as opposed to the .767 for the lowest skill group. Even with additional latency, it will still be an easy task for a high ranked player to not miss the ball, while the lower ranked players struggle and the latency is more important. I expect that if there was a shot that the best players miss .767 times, then they would be more affected by latency as such a shot likely requires more precise timing than the one in the experiment. It is unclear why there is no trend in the response variable `dist_to_goal`. Since `score` is a compound metric, it is unsurprising that the extra misses, which are weighted heavily, cancel out the extra shot power. Prior research either found no interaction between latency and player skill [66, 103], or the reverse [104]. There are multiple possible reasons for this. It can be due to the same effect that I described regarding misses. If an aim and click task is so easy that a great player will almost always succeed on first try regardless of latency, then a worse player's task completion time will be more affected. It can also be because most studies use special applications created for the experiment. The hypothesis that players of higher skill are more affected by latency is based on the assumption that they are deeply familiar with the environment, which allows them to do very high precision tasks. That precision necessitates low latency. Any new game will be unfamiliar and will feel different, especially when done in laboratory conditions with a different input device than the player usually uses. Therefore, the players can't rely on this kind of precision anyway, and just the general ability to adapt to new games and conditions will give the experienced players the upper hand with and without added latency. Whenever no interaction was found, it can also be due to imprecise assessment of skill (usually self-rated), small sample size, and lack of very high skilled players. My experiment has a significantly larger sample size than any previous study on the topic, and was able to draw comparisons between players of the 99.99th percentile and the average player.

The final goal of the performance evaluation was to check for interactions between graphical effect density/peripheral vision and latency. The main expectation was that they interact in terms of perception, but this could indirectly affect the performance response variables. No significant interaction is present in the experiment data. These points are further elaborated upon in section 6.3.

The baseline system latency caused no statistically significant performance difference. This is not unexpected though, as the results are already corrected for by player skill rating. Any player that does worse at the experiment because of their baseline latency is expected to do worse in the game's ranked mode, and would on average

be sorted into a lower skill group. However, the game revolves around more than just the shooting in the experiment, and it is therefore not a given that the overall player skill has to be affected by the same amount. The indifference may also be in part due to the statistical power, or because with enough practice differences in baseline latency of about 30 ms (the difference between the highest and lowest group) are nearly nonexistent. There was no interaction between the baseline and added latency, which allowed the analysis of the larger dataset.

The independent variable `graphics` shows that the participants perform worst at the minimum graphics settings in terms of `ball_misses` and `score`, which is dependent on the misses. This is despite the fact that minimum graphics will result in the lowest latency. The minimum graphics in Rocket League lack dynamic shadows, which can help with depth perception, allowing players to more accurately assess the location of the ball. Furthermore, there is a significant interaction between `graphics` and `skill_group` that showed only lower ranked players are negatively affected by the minimum graphics settings. This fits with the shadow hypothesis, as more experienced players can use, for example, the exact size of the ball on the screen to determine the distance to the ball. A less experienced player may be required to rely on cues that can carry over from the real world.

The results demonstrate that the type of shot greatly influences how latency as well as player skill affects the performance. Therefore, I also expect that a different task within the game that isn't about shooting may be affected differently. It has to be noted that the shots that are the most influenced by the player's skill are not all the most influenced by added latency. That means it cannot simply be that the response variables are more sensitive for those shots. The disadvantages of latency that lead to decreased performance are not the same as being less skilled at the game. These differences also make it difficult to make an accurate estimation of how the game as a whole is affected by latency.

Based on the within-subjects contrasts, the linear and quadratic terms of added latency are significant for all response variables. When using regression to fit a quadratic mode, the results fit the means for each response variable very well (all $R^2_{\text{adj}} > .98$). When calculating $R^2_{\text{adj}}$ with the individual datapoints, the values are all $< .007$ due to the large amount of variance present between trials. The resulting equation is: $y = -2.676 \times 10^{-4}x^2 + 3.820 \times 10^{-3}x + 4.037$, where $x$ is added latency and $y$ is the `score`.

## 6.3 Latency perception

The first in terms of latency perception was also replication (H2). The results show that even an increase of 8.33 ms can cause a significant increase in perceived latency. The difference is very small, but in a group of many players, the difference can become visible. As a game developer, that means that even a small trade-off in latency has to be well considered, as the game's community may notice it. Compared to Martens et al. [5], players notice significantly less latency. Their results stated that at an additional latency of 97 ms, participants chose a higher score 59 % of the time. That is the equivalent percentage to my experiment with an additional 20 ms. Even the skill group that represents the average player reaches that percentage with 33 ms. One of the likely reasons for this difference was already hypothesized by Ellis et al. [53] to explain significant differences in latency perception. Knowledge of what latency looks and feels like is very important when trying to identify it. Most participants of my experiment are expected to have pre-existing knowledge about latency as explained in section 3.3. Martens et al. [5] also tracked perceived difficulty of the task, which can serve as a comparison. The perceived difficulty rose significantly with as little as 28 ms but the effect size is still slightly smaller than the perceived latency in my experiment.

The second goal of the experiment was to test if higher skill played players are better at perceiving latency (H3). The data shows a very clear difference between the average RL player and the 99.99th percentile. The average player was not able to notice 8 and 17 ms of additional latency at all. At 33 and 50 ms the effect size of `added_lat` on the perceived latency is almost 4 times higher for the highest skill group. The highest skill group rated an additional 50 ms of latency on average 3 points higher than no additional latency, and therefore correctly identified it 94 % of the time. The relative effect size difference exceeds the relative score difference, which shows that higher rated players were not just more annoyed[1] by the same amount of latency, but they also gave less random answers. The idea of experience playing a role in the ability to judge latency is neither novel [53, 55], nor will it be surprising to most; however, I don't believe it has been documented to this level of different groups and sample size. Furthermore, the players of the lowest skill group had a median RL play time of 600 h. They cannot be classified as beginners at all as they are average players of the game that are capable of precisely navigating their car at high speeds. This is quantified by the average players scoring a shot 50 % of the time, that a new player missed 20 times in a row (reported in section 5.4).

[1] The questionnaire displayed the annotation "Terrible lag" underneath the highest point options (see figure 3.1).

When considering that the average participant is likely more informed about latency than the average gamer, the difference to professional players may be even larger in practice. Comparing once more to the perceived difficulty of Martens et al. [5], the effect sizes match with the skill group that makes up the 75th–93rd percentile of RL players. There is however a big difference. The effect size values of perception in the RL experiment greatly exceed those of performance by a factor of 2–4, while Martens et al. found slightly lower effect sizes than the performance. And in my case, players were able to perceive increased latency even when they performed (insignificantly) better with it. This shows that the participants do not all just use the perceived difficulty to differentiate between the scenarios. The two measures can't be considered equal.

| **Skill group percentile** | Cohen's $d$ vs. 0 ms | | | |
|---|---|---|---|---|
| | 8.33 ms | 16.67 ms | 33.33 ms | 50.00 ms |
| **< 75th** | 0.041 | 0.017 | 0.338 | 0.635 |
| **75th–93rd** | 0.099 | 0.197 | 0.527 | 1.018 |
| **93rd–99.5th** | 0.058 | 0.213 | 0.633 | 1.303 |
| **99.5th–99.99th** | 0.102 | 0.329 | 1.007 | 1.970 |
| **> 99.99th** | 0.273 | 0.316 | 1.212 | 2.281 |
| **Total** | 0.081 | 0.216 | 0.663 | 1.280 |

**Tab. 6.1.:** Average perceived latency answers with added latency, split by skill group.

I calculated the JND in order to compare the perception to other related work. Even though the measurements end at 50 ms, the data shows that for the lowest skill group the JND approximately matches the 55 ms Deber et al. [44] found for an indirect dragging task. Rocket League constitutes an indirect second order control task. In order to confirm that the input type does not change the outcome, a direct replication of Deber et al.'s experiment with a controller would be needed. The JND highlights again the huge difference between the skill groups. The best players are able to notice half the latency of the average player which already has 600 h of experience. Even the group with a geometric mean experience of 1981 h only has a JND of 40 ms in the experiment, 12 ms more than the group with barely over 1300 h more experience. The top 1 % players of RL have a latency JND of 28 ms, which is substantially lower than any other indirect input task I have found in related work.

These results bring into question any sort of latency perception threshold established in previous literature. While the results for the average participant in those experiments are valid, imposing a design choice based on a loose threshold may negatively affect not just gamers but also power users. In this experiment, the player skill is not perfectly controlled for and causation cannot be asserted. While I do control for

baseline latency, graphics settings, and monitor FOV, there may be other factors in which the groups differ. Age and hours of experience were correlated with the skill of players. Therefore, I ran extra ANOVAs to see whether the factors interact with the latency more than skill group does. As this was not part of the planned analysis, it may be seen as torturing the data, which is why it is not part of the results chapter. Neither age nor hours showed an interaction with `added_lat`. However, players with more hours generally gave higher answers, regardless of the current latency.

One more goal of the experiment was the investigation of the graphical effect density in combination with latency (H5). The participants do not perceive additional latency significantly different at different graphics settings. This is the same result as Mania et al. [49]. However, the graphics settings are not without any effect. They act like a fixed factor. The difference between minimum and maximum graphics settings is not statistically significant and is also expected as the higher graphics do result in a slight increase in latency on average. The difference between the user's own graphics settings and the others is most likely due to the user being familiar with the exact look and latency. The participants rate the perceived latency of their own settings lower on average, despite the minimum graphics resulting in either the same or lower latency.

The final aim of the study was to compare the role of peripheral vision in latency perception (H6). Due to the typical monitor size to distance ratio, the far peripheral vision was not investigated. This may be relevant for virtual reality. I found no statistically significant effect of the player FOV on the perception of latency. The hypothesis may still be true in other games. In Rocket League, the camera is focused onto the ball. While there is an alternative camera mode, it is the less commonly used, and the majority of the datapoints will have stayed in the ball focus mode. When a player starts a turn or stops it, the movement in the player's peripheral vision will not drastically change as it would when the camera is directly attached to the turning. Shooter games, where the player directly controls the camera, will show more abrupt movement changes in the peripheral vision. Thus, they should be considered as a test before concluding that the peripheral vision cannot aid in the detection of latency.

When looking at the within-subjects contrasts, added latency has a significant linear and quadratic term. Using regression to fit a quadratic equation results in an $R^2_{\text{adj}} > .999$ for the means, and $R^2_{\text{adj}} = .200$ for the individual datapoints. When splitting the dataset by skill groups, all of them fit a quadratic equation. The values are listed in table 6.2. The non-linear response could be related to the fact that player performance follows the same shape.

| skill_group | Model | | | |
|---|---|---|---|---|
| | Intercept | $x$ | $x^2$ | $R^2_{\text{adj}}$ |
| **< 75th** | 1.023 | -1.036E-03 | 4.271E-04 | .0614 |
| **75th–93rd** | 1.116 | 5.075E-03 | 4.881E-04 | .1265 |
| **93rd–99.5th** | 1.195 | 2.785E-03 | 7.255E-04 | .2056 |
| **99.5th–99.99th** | 1.183 | 1.392E-02 | 8.347E-04 | .3587 |
| **> 99.99th** | 1.175 | 2.217E-02 | 7.824E-04 | .3875 |

**Tab. 6.2.:** Quadratic model factors for perceived latency.

## 6.4 Observational results

Multiple times players stated that they were not able to judge the latency as well as they would have been able to if the experiment had a different design. Many past latency studies with the focus of finding perception thresholds use forced choice comparison, where the player gets shown in random order the baseline scenario and the added latency one to compare it to. This direct comparison always allows the user to have a reference, and they can always choose one clear winner. I do expect that this will lead to lower JND measured in an experiment, but it's impossible to predict how much. One of the primary goals of the experiment was to measure player performance in a structured, unlikely to be exploited manner. The perception survey was built afterwards. A similar criticism was uttered about the chosen task. The shots are the best way for a predictable performance measurement. Dribbling has a crossover point where a player can become so good at it that they will be able to keep the ball up for hours. While it would be possible to deduct score for when the ball isn't perfectly centered above the car, this sort of task will have nothing in common with real games of Rocket League. A dribbling task where the player has to navigate a certain route has the opposite problem. The average player would not be able to complete it. I was aware of the limitations of the chosen tasks. It is unknown how different tasks in the same game with unchanging controls may affect the perception. If the focus was solely on perception of latency, the experiment would have certainly been better designed another way. Splitting the experiment into two different parts was not done because this would increase the length of the experiment and likely decrease participation.

Some participants said their performance was affected by the constant switching between latencies. Specifically, that going from high to low latency supposedly made the task a lot more difficult than just staying at the high latency. The data does show that participants did better with 8.33 ms of added latency on all metrics but `shot_vel`. On `dist_to_goal` and `goal_scored`, even the 16.67 ms condition

had a better outcome than the baseline. However, all of these differences are not statistically significant when performing Bonferroni corrected pairwise comparisons. In order to further investigate the claim, I ran a separate analysis in *R* that wasn't possible within the GLM repeated measures doubly multivariate ANOVA. I used a linear mixed model and added the new factors, `scenario_it` — denoting which of the five shots per scenario (set of five shots) the participant got in which order — and `lat_difference` — the difference in latency between the current and previous scenario. When using the full interaction model of the two factors, the ANOVA determined them not significant. When using only `scenario_it`, it does reach significance ($p = .017$). The result shows that on the first shot of any scenario the participants perform worse (cf. figure 6.2), which could be due to the change in latency but also because their flow has been disrupted by the latency perception dialog window. I removed all the first shots of every scenario and observed no change at all in the trend of performance with added latency. Even when using only the last of the five shots of each scenario, the trend stays the same. While this does not prove that no such adverse effects exist, it means that if they do, five shots are not enough to adjust to latency.

Looking at the issue from a theoretical point of view, I assume for now that the changing latency does cause a shift in results. The average added latency throughout the experiment is 21.67 ms. If the players adapt their overall play to that, then any deviation from that figure will decrease performance. Then I did regression with the terms `added_lat` + $\left|$ `added_lat` $- 21.67\right|$, which gives $R^2_{\text{adj}}$ values almost identical to those with the polynomial model. With this model, it is possible to correct the data or simply ignore the second term and use the model to predict the assumed true impact of latency. The corrected score is then highest at the lowest latency condition as one would expect (cf. figure 6.3). This may be a better assumption of the true impact of additional latency. However, since the original results at 0.00–16.67 ms did not differ significantly from each other, there is no scientific reason to use this more complex model over the standard one. It is built on multiple assumptions. One more issue that the correction does

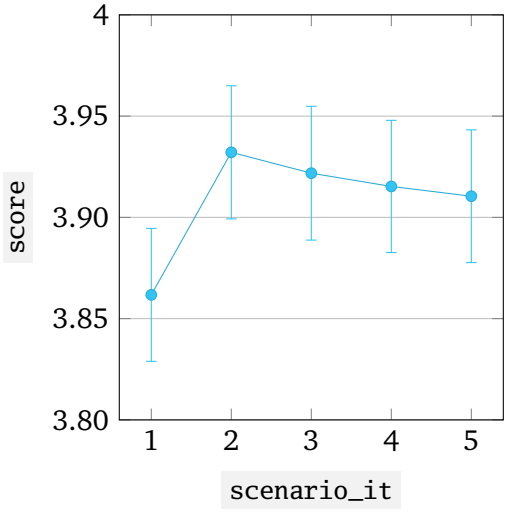


Fig. 6.2.: The first shot in every set of five shots has lower performance.

not account for is that the player would also be expected to start out with a strategy that works at the baseline latency they usually play at, and then over the course of the experiment adapt until they have an optimal average strategy for the average latency. The adaptation may not even be completed by the end of the experiment. It remains a hypothesis for now that might be worth considering in future work. Experiments don't usually have the luxury of giving participants a very long time to adjust to every condition if more than a handful of scenarios are to be tested. A direct comparison between short and long adjustment times on the same task may also provide more insights into the relevance of baseline latency. Pavlovych and Stuerzlinger [62] and Friston et al. [64] had similar trends in their data, while most other studies found a near linear reduction of performance at these latency levels. One of Friston et al.'s hypotheses for the trend was that participants are more used to higher latency than what they had in the baseline laboratory setup. This cannot be true in my experiment, as the participants were playing on their own setup with the baseline latency that they are used to.

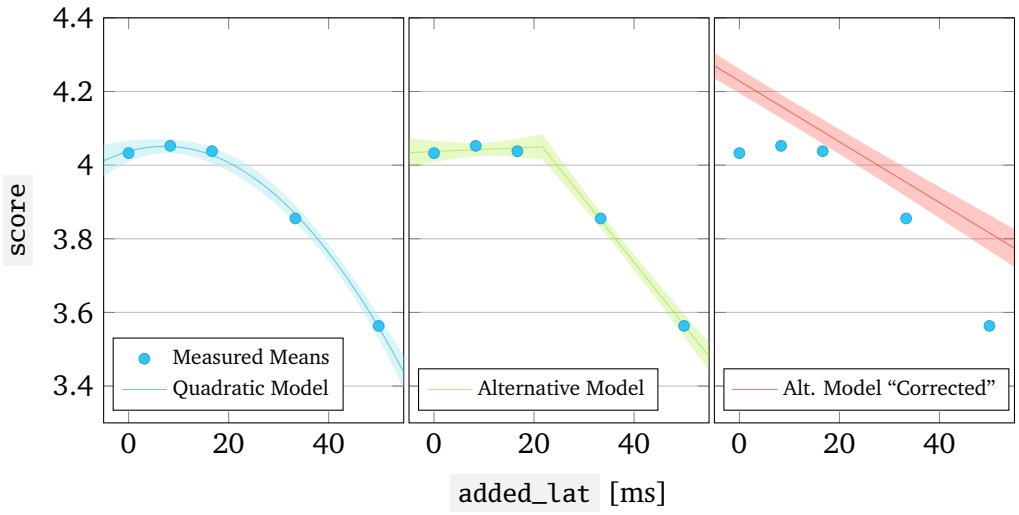


Fig. 6.3.: The quadratic model fits the data equally well as the one using the distance from the mean `added_lat`. The third graph displays what the alternative model predicts if the participants had ample time to adapt to the latency. Shading denotes 95 % confidence interval.

## 6.5 Recommendations

My results show that prescribing a one-size-fits-all latency recommendation may be harmful. If a developer takes a commonly cited recommendation, such as the 100 ms figure by Nielsen [41], and concludes that optimization beyond this mark provides

no benefit, they end up making multiple fallacies. First, the academic research as a whole shows that latency affects both the perception and performance very differently depending on the task and input type (see chapter 2). This thesis shows that even tasks that seem similar are affected by a significantly different margin. Furthermore, the experiment reveals that experts of the 99th percentile are vastly more affected by latency, especially in terms of perception. The average Rocket League player tolerates more than twice the latency that an expert does. These groups have a geometric mean lifetime experience of 550 and 3200 h in the game. While it is unclear whether this difference is as severe for all kinds of tasks, I do not have any reasons to believe that Rocket League would be entirely different. Thus, if a developer wants to have a latency target to aim for, I recommend the following process:

1. Determine the level of user impact in terms of performance (e. g. 5 % accuracy) and perception (e. g. 50 % detection or 1 extra point on a 5 point scale) that you deem acceptable.

2a. Find latency research which has tested a task very similar to those the user needs to perform in the application or game, or

2b. set up a dummy application and run an experiment with the desired tasks.

3. Cut the determined latency time from the previous step in half to account for expert users that are more attuned to the latency.

4. Subtract at least 6 ms to account for a very fast input device and monitor.[2]

The developer still needs to take into account factors like the operating system's composition manager and VSync to stay within the determined end-to-end latency target.

For Rocket League players, I provide guidelines of what to keep in mind when optimizing for latency. While lower latency is better in general, one has to balance the costs with the benefits. The benefits of small latency improvements on performance are minor, even if one assumes the adjusted model (see section 6.4). Having lower latency is also not equivalent to being a player of higher skill. I do not have information on how other aspects of the game like dribbling and defending are affected. These are the costs to keep in mind:

[2]The more a user cares about latency, the more likely they are to get fast hardware. Therefore, there is generally no need to hit the latency target for the users with slow hardware. Accounting for higher latency devices makes sense when the hardware is known, such as the official controller on a console.

- cost of hardware, such as a more expensive monitor, input device, CPU, and GPU
  - If one is playing for prize money of $10 000, then spending $100 on hardware has to provide a 1 % increased chance of winning to break even.
  - If the prize money is $1 million, then anything above a 0.01 % chance to win is worth $100.
- lowering some specific graphics details may reduce player ability to see and accurately judge situations
  - lack of shadows may impair depth perception and causes misjudgments of the cars' and ball's location[3]
  - low resolution may make it difficult to see which direction a car is facing on the other end of the field
  - transparent goalposts make sure that the ball and opponents stay visible at all times
  - the ball trail shows the spin of the ball when the world detail is set to high
- config options like `OneFrameThreadLag=False` only provide latency improvements in specific situations and may harm frame rate stability

Some external programs for customizing inputs do add significant latency (Steam Controller Configuration) while others do not (x360ce, DS4Windows, Durazno). Users should make informed decisions if this is important to them.

[3] This is visible in the experiment data.

# 7 Conclusion and future work

The experiment has replicated previous findings that show that the player's performance is affected negatively by additional latency, and that players are able to notice when this latency is present. The hypothesis that graphics effects significantly interact with the effects of latency has yielded no evidence. With the similar idea tested by Mania et al. [49] also not demonstrating any effects, I don't predict further research in this direction will be fruitful. Unlike the graphics condition, there were limitations of the FOV range present in the experiment. Thus further research on peripheral vision and latency in the domain of virtual reality may lead to different results. Due to the virtual camera used in Rocket League, movements at the edge of the screen are also significantly different from any first-person game. This is another option to explore.

The most significant finding of the experiment is the impact of player skill on the effects of latency. In terms of performance, shot velocity, which highly depends on timing, is more sensitive to latency for highly skilled players ($d = -.211$ vs. $d = .-071$). The perception of latency is drastically different between players of different skill levels. Players of the 99th percentile notice 23 ms of latency at the same rate that the average player notices 50 ms. The groups in between show that the ability to perceive latency increases with every bit of experience. The fact that the lowest skill group in the experiment still has 550 h of game experience demonstrates that this difference is not compared to beginners. The result poses a lot of potential questions for future research. Are people working 8 hours a day in a text editor also twice as latency sensitive as those doing it for 1 hour. What are the limits of latency perception in a task with indirect input? The experiment was not using a direct comparison between two latency options which likely underestimates the limit.

The non-linear effects of latency with an apparent floor were surprising to me. Due to the large sample size, I expected there to be a small but statistically significant reduction of the performance at 8.33 ms. A future study could investigate the hypothesis, that frequently changing latency causes these numbers. This can be tested by giving participants more and less time to adjust to each latency condition. Alternatively, a study only comparing two latency conditions such as done by Spjut et al. [66] would not be affected by such a problem. A long-term study in which

players have weeks or longer to retrain their muscle-memory would be useful. That would also make it possible to test the effects of latency on learning and the ability to get better at games.

One of the results I did not expect was how much of a difference the individual shots had. Shot number 5 was 3–5 times less affected by latency based on effect size. The shot was likely done without a dodge by the majority of players. A dodge is a highly timing dependent mechanic. This is currently the only reason I can see for the drastic difference. A study that tests this theory by categorizing different mechanics and analyzing what specifically is affected by latency would be very useful. The tasks in this experiment were simply designed to cover a number of different scenarios rather than test specific aspects.

The ability to run the experiment online allowed for a very large sample size and a large variety of skill. The downside is the lack of a controlled lab environment, which can also be an upside, as players are taking the experiment in their most familiar environment. The estimation of latency is difficult to prove accurate. If the majority of players had their latency measured, then prediction is no longer required. Very accurate estimation of the PC part of latency is likely possible if the developers measure it themselves using timers. The identification of monitors and input devices was over 100 hours of work, and still required more than half of the participants to be removed from the dataset with estimated latency.

# Bibliography


[1] Paul Martin: "The Intellectual Structure of Game Research." In: *Game Studies. The International Journal of Computer Game Research* 18.1 (Apr. 2018). ISSN: 1604-7982. URL: http://gamestudies.org/1801/articles/paul_martin (visited on July 5, 2021).

[2] Jason G. Reitman, Maria J. Anderson-Coto, Minerva Wu, Je Seok Lee, and Constance Steinkuehler: "Esports Research: A Literature Review." In: *Games and Culture* 15.1 (Apr. 15, 2019), pp. 32–50. DOI: 10.1177/1555412019840892.

[3] Benjamin Watson, Josef Spjut, Joohwan Kim, et al.: "Esports and High Performance HCI." In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450380959. DOI: 10.1145/3411763.3441313.

[4] Sharon Andrews and Caroline M. Crawford, eds.: *International Journal of eSports Research (IJER)* (2021). ISSN: 2691-9273. DOI: 10.4018/IJER.

[5] Judith Martens, Thomas Franke, Nadine Rauh, and Josef F. Krems: "Effects of low-range latency on performance and perception in a virtual, unstable second-order control task." In: *Quality and User Experience* 3.1 (2018), pp. 1–17. ISSN: 2366-0147. DOI: 10.1007/s41233-018-0023-z.

[6] Mark R. Mine: *Characterization of end-to-end delays in head-mounted display systems*. Tech. rep. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 1993. URL: https://www.cs.unc.edu/techreports/93-001.pdf.

[7] Matthew J. P. Regan, Gavin S. P. Miller, Steven M. Rubin, and Chris Kogelnik: "A real-time low-latency hardware light-field renderer." In: *SIGGRAPH 1999 Conference Proceedings*. ACM Press Ser. Boston: Addison Wesley Professional, 1999. ISBN: 0201485605. DOI: 10.1145/311535.311569.

[8] Richard Yao, Tom Heath, Aaron Davies, et al.: *Oculus VR Best Practices Guide*. Oculus VR, July 23, 2014. URL: http://energylab.hpa.edu/public/brain/oculus/oculussdk/doc/oculus_best_practices_guide.pdf (visited on June 23, 2021).

[9] Epic Games: *Unreal Networking Architecture*. 2009. URL: https://docs.unrealengine.com/udk/Three/NetworkingOverview.html (visited on July 14, 2021).

[10] Cheryl Savery, Nicholas Graham, Carl Gutwin, and Michelle Brown: "The effects of consistency maintenance methods on player experience and performance in networked games." In: *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, Feb. 2014. DOI: 10.1145/2531602.2531616.

[11] Yahn W. Bernier (Valve): *Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization*. In: Game Developers Conference (GDC), San Francisco, CA, USA, Mar. 2001. URL: https://web.archive.org/web/20041107064247/https://www.gdconf.com/archives/2001/bernier.doc (visited on July 16, 2021). Also available at: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization.

[12] Jared Cone (Psyonix): *It IS Rocket Science! The Physics of 'Rocket League' Detailed*. In: Game Developers Conference (GDC), San Francisco, CA, USA, Mar. 21, 2018. URL: https://www.gdcvault.com/play/1024972/It-IS-Rocket-Science-The.

[13] Alan Traviss Welford: *Fundamentals of Skill*. London, England: Methuen & Co, 1968. 428 pp. ISBN: 9780416700206. URL: http://hdl.handle.net/123456789/13954.

[14] I. Scott MacKenzie: *Human-Computer Interaction. An Empirical Research Perspective*. Elsevier Science & Technology, Feb. 21, 2013. 370 pp. ISBN: 9780124058651.

[15] I. Scott MacKenzie and Colin Ware: "Lag as a determinant of human performance in interactive systems." In: *Human factors in computing systems*. Ed. by S. Ed Ashlund. CHI '93. Amsterdam, Netherlands: Association for Computing Machinery, 1993, pp. 488–493. ISBN: 0897915755. DOI: 10.1145/169059.169431.

[16] Topi Kaaresoja, Stephen Brewster, and Vuokko Lantz: "Towards the Temporally Perfect Virtual Button: Touch-Feedback Simultaneity and Perceived Quality in Mobile Touchscreen Press Interactions." In: *ACM Trans. Appl. Percept.* 11.2 (2014). ISSN: 1544-3558. DOI: 10.1145/2611387.

[17] Sophie Jörg, Aline Normoyle, and Alla Safonova: "How responsiveness affects players' perception in digital games." In: *Proceedings of the ACM Symposium on Applied Perception*. Ed. by Peter Khooshabeh, Matthias Harders, Rachel McDonnell, and Veronica Sundstedt. New York: ACM, 2012. ISBN: 9781450314312. DOI: 10.1145/2338676.2338683.

[18] Daniel Kahneman: *Thinking, fast and slow*. New York, NY, USA: Farrar, Straus and Giroux, 2011. ISBN: 978-0374275631.

[19] Christopher D. Wickens, Justin G. Hollands, Simon Banbury, and Raja Parasuraman: *Engineering psychology and human performance*. Fourth edition. London, England and New York, NY, USA: Routledge, 2016. ISBN: 9781317351320.

[20] Steve T. Bryson and Scott S. Fisher: "Defining, modeling, and measuring system lag in virtual environments." In: *Stereoscopic Displays and Applications*. Ed. by John O. Merritt and Scott S. Fisher. Santa Clara, CA, USA: SPIE, Sept. 1990. DOI: 10.1117/12.19894.

[21] Ding He, Fuhu Liu, Dave Pape, Greg Dawe, and Dan Sandin: "Video-Based Measurement of System Latency." In: *Fourth International Immersive Projection Technology Workshop*. IPT 2000. Ames, IA, USA: Iowa State University, June 2000. URL: https://www.evl.uic.edu/documents/latency_ipt2000.pdf.

[22] Colin Swindells, John C. Dill, and Kellogg S. Booth: "System lag tests for augmented and virtual environments." In: *Proceedings of the 13th annual ACM symposium on User interface software and technology*. UIST '00. New York, NY, USA: Association for Computing Machiner, 2000, pp. 161–170. DOI: 10.1145/354401.354444.

[23] Dorian Miller and Gary Bishop: "Latency Meter: A device to measure end-to-end latency of VE systems." In: *Stereoscopic Displays and Virtual Reality Systems IX*. Ed. by Andrew J. Woods, John O. Merritt, Stephen A. Benton, and Mark T. Bolas. SPIE, May 23, 2002. DOI: 10.1117/12.468062.

[24] Anthony 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. New York, NY, USA: Association for Computing Machinery, Oct. 27, 2008, pp. 123–129. DOI: 10.1145/1450579.1450606.

[25] Robert J. Teather, Andriy Pavlovych, Wolfgang Stuerzlinger, and I. Scott MacKenzie: "Effects of tracking technology, latency, and spatial jitter on object movement." In: *2009 IEEE Symposium on 3D User Interfaces*. IEEE, 2009. DOI: 10.1109/3dui.2009.4811204.

[26] Massimiliano Di Luca: "New Method to Measure End-to-End Delay of Virtual Reality." In: *Presence: Teleoperators and Virtual Environments* 19.6 (Dec. 1, 2010), pp. 569–584. DOI: 10.1162/pres_a_00023.

[27] Scott J. Horowitz: *Measurement and Effects of Transport Delays in a State-of-the-Art F-16C Flight Simulator.* Tech. rep. San Antonio, TX, USA: Air Force Human Resources Laboratory, Sept. 1, 1987. URL: https://apps.dtic.mil/sti/citations/ADA187367 (visited on July 22, 2021).

[28] Jiandong Liang, Chris Shaw, and Mark Green: "On temporal-spatial realism in the virtual reality environment." In: *Proceedings of the 4th annual ACM symposium on User interface software and technology*. UIST '91. New York, NY, USA: ACM Press, Nov. 11, 1991. DOI: 10.1145/120782.120784.

[29] Zenja Ivkovic, Ian Stavness, Carl Gutwin, and Steven Sutcliffe: "Quantifying and Mitigating the Negative Effects of Local Latencies on Aiming in 3D Shooter Games." In: *CHI 2015*. New York, NY, USA: The Association for Computing Machinery, 2015. ISBN: 9781450331456. DOI: 10.1145/2702123.2702432.

[30] Roger Graves and Ron Bradley: "Millisecond interval timer and auditory reaction time programs for the IBM PC." In: *Behavior Research Methods, Instruments, & Computers* 19.1 (Jan. 1987), pp. 30–35. DOI: 10.3758/bf03207667.

[31] Géry Casiez, Thomas Pietrzak, Damien Marchal, et al.: "Characterizing Latency in Touch and Button-Equipped Interactive Systems." In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, Oct. 2017. DOI: 10.1145/3126594.3126606.

[32] Richard R. Plant, Nick Hammond, and Tom Whitehouse: "Toward an Experimental Timing Standards Lab: benchmarking precision in the real world." In: *Behavior research methods, instruments, & computers : a journal of the Psychonomic Society, Inc* 34.2 (2002), pp. 218–226. ISSN: 0743-3808. DOI: 10.3758/BF03195446.

[33] Richard R. Plant, Nick Hammond, and Garry Turner: "Self-validating presentation and response timing in cognitive paradigms: how and why?" In: *Behavior research methods, instruments, & computers : a journal of the Psychonomic Society, Inc* 36.2 (2004), pp. 291–303. ISSN: 0743-3808. DOI: 10.3758/BF03195575.

[34] Florian Bockes, Raphael Wimmer, and Andreas Schmid: "LagBox – Measuring the Latency of USB-Connected Input Devices." In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI EA '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–6. ISBN: 9781450356213. DOI: 10.1145/3170427.3188632.

[35] Andreas Schmid and Raphael Wimmer: "Yet Another Latency Measuring Device." In: *EHPHCI: Esports and High Performance HCI*, Apr. 10, 2021. CHI '21. OSF Preprints, Apr. 10, 2021. DOI: 10.31219/osf.io/tkghj.

[36] Mark Claypool, Ragnhild Eg, and Kjetil Raaen: "The Effects of Delay on Game Actions." In: *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*. CHI PLAY Companion '16. New York, NY, USA: Association for Computing Machinery, Oct. 15, 2016, pp. 117–123. DOI: 10.1145/2968120.2987743.

[37] Mark Claypool: "Game Input with Delay—Moving Target Selection with a Game Controller Thumbstick." In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 14.3s (Aug. 2018), pp. 1–22. DOI: 10.1145/3187288.

[38] Mark Claypool, Andy Cockburn, and Carl Gutwin: "Game input with delay: moving target selection parameters." In: *Proceedings of the 10th ACM Multimedia Systems Conference*. New York, NY, USA: Association for Computing Machinery, June 18, 2019. ISBN: 9781450362979. DOI: 10.1145/3304109.3306232.

[39] Mark Claypool, Andy Cockburn, and Carl Gutwin: "The Impact of Motion and Delay on Selecting Game Targets with a Mouse." In: *ACM Transactions on Multimedia Computing, Communications, and Applications* 16.2s (July 2020), pp. 1–24. DOI: 10.1145/3390464.

[40] Robert B. Miller: "Response time in man-computer conversational transactions." In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*. New York, NY, USA: ACM Press, 1968. DOI: 10.1145/1476589.1476628.

[41] Jakob Nielsen: “Chapter 5 - Usability Heuristics.” In: *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann, 1993, pp. 115–163. ISBN: 9780125184069. DOI: 10.1016/B978-0-08-052029-2.50008-5.

[42] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay: “The information visualizer, an information workspace.” In: *Proceedings of the SIGCHI conference on Human factors in computing systems Reaching through technology - CHI '91*. ACM Press, 1991. DOI: 10.1145/108844.108874.

[43] Stuart K. Card, Thomas P. Moran, and Allen Newell: *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1983. 488 pp. ISBN: 9780898592436. DOI: 10.1201/9780203736166.

[44] Jonathan Deber, Ricardo Jota, Clifton Forlines, and Daniel 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. Association for Computing Machinery, 2015, pp. 1827–1836. ISBN: 9781450331456. DOI: 10.1145/2702123.2702300.

[45] Michelle Annett, Albert Ng, Paul Dietz, Walter F. Bischof, and Anoop Gupta: “How Low Should We Go? Understanding the Perception of Latency While Inking.” In: *Proceedings of Graphics Interface 2014*. GI '14. Montreal, Quebec, Canada: Canadian Information Processing Society, 2014, pp. 167–174. ISBN: 9781482260038. URL: https://dl.acm.org/doi/10.5555/2619648.2619677.

[46] Albert Ng, Michelle Annett, Paul Dietz, Anoop Gupta, and Walter F. Bischof: “In the blink of an eye.” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2014. DOI: 10.1145/2556288.2557037.

[47] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz: “Designing for low-latency direct-touch input.” In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. Ed. by Rob Miller. ACM Digital Library. New York, NY, USA: ACM, 2012. ISBN: 9781450315807. DOI: 10.1145/2380116.2380174.

[48] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor: “How fast is fast enough?” In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2013. DOI: 10.1145/2470654.2481317.

[49] Katerina Mania, Bernard D. Adelstein, Stephen R. Ellis, and Michael 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*. ACM Press, 2004. DOI: 10.1145/1012551.1012559.

[50] Jason J. Jerald: “Scene-motion- and latency-perception thresholds for head-mounted displays.” PhD thesis. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 2009. DOI: 10.17615/CD1T-9Y62.

[51] Stephen R. Ellis, Mark J. Young, Bernard D. Adelstein, and Sheryl 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: Communication, Cooperation, and Application Design – Volume 2*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., Aug. 1999, pp. 1129–1133. ISBN: 9780805833928. URL: https://humansystems.arc.nasa.gov/publications/Ellis_1999_Head_Movement_Latency.pdf.

[52] Bernard D. Adelstein, Thomas G. Lee, and Stephen R. Ellis: "Head Tracking Latency in Virtual Environments: Psychophysics and a Model." In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 47.20 (Oct. 2003), pp. 2083–2087. DOI: 10.1177/154193120304702001.

[53] Stephen R. Ellis, Katerina Mania, Bernard D. Adelstein, and Michael I. Hill: "Generalizeability of Latency Detection in a Variety of Virtual Environments." In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 48.23 (Sept. 2004), pp. 2632–2636. DOI: 10.1177/154193120404802306.

[54] Teemu Mäki-Patola and Perttu Hämäläinen: "Latency Tolerance for Gesture Controlled Continuous Sound Instrument Without Tactile Feedback." English. In: *Proceedings of ICMC 2004, the 30th Annual International Computer Music Conference*. Miami, FL, USA, Nov. 2004, pp. 409–416. ISBN: 9780971319226. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.8653&rep=rep1&type=pdf.

[55] Eryk Banatt, Stefan Uddenberg, and Brian Scholl: "Input Latency Detection in Expert-Level Gamers. An experiment in visuomotor perception." Apr. 21, 2017. URL: https://cogsci.yale.edu/sites/default/files/files/Thesis2017Banatt.pdf (visited on July 7, 2021). Draft.

[56] T. Kaaresoja, E. Anttila, and E. Hoggan: "The effect of tactile feedback latency in touchscreen interaction." In: *2011 IEEE World Haptics Conference*. Institute of Electrical and Electronics Engineers (IEEE), June 2011. DOI: 10.1109/whc.2011.5945463.

[57] Jack E. Conklin: "Effect of control lag on performance in a tracking task." In: *Journal of Experimental Psychology* 53.4 (1957), pp. 261–268. DOI: 10.1037/h0040728.

[58] E. C. Poulton: *Tracking Skill and Manual Control*. New York, NY, USA: Academic Press, 1974. ISBN: 9780125635509.

[59] M. J. Warrick: *Effect of transmission-type control lags on tracking accuracy*. Tech. rep. 5916. Dayton, OH, USA: USAF Air Materiel Command, 1949.

[60] Paul M. Fitts: "The information capacity of the human motor system in controlling the amplitude of movement." In: *Journal of Experimental Psychology* 47.6 (1954), pp. 381–391. DOI: 10.1037/h0055392.

[61] Andriy Pavlovych and Wolfgang Stuerzlinger: "The tradeoff between spatial jitter and latency in pointing tasks." In: *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. EICS '09. New York, NY, USA: Association for Computing Machinery, July 15, 2009, pp. 187–196. DOI: 10.1145/1570433.1570469.

[62] Andriy Pavlovych and Wolfgang Stuerzlinger: "Target Following Performance in the Presence of Latency, Jitter, and Signal Dropouts." In: *Proceedings of Graphics Interface 2011*. GI '11. St. John's, Newfoundland, Canada: Canadian Human-Computer Communications Society, 2011, pp. 33–40. ISBN: 9781450306935. URL: https://dl.acm.org/doi/10.5555/1992917.1992924.

[63] Andriy Pavlovych and Carl Gutwin: "Assessing Target Acquisition and Tracking Performance for Complex Moving Targets in the Presence of Latency and Jitter." In: *Proceedings of Graphics Interface 2012*. GI '12. Toronto, Ontario, Canada: Canadian Information Processing Society, 2012, pp. 109–116. ISBN: 9781450314206. URL: https://dl.acm.org/doi/10.5555/2305276.2305295.

[64] Sebastian Friston, Per Karlström, and Anthony Steed: "The Effects of Low Latency on Pointing and Steering Tasks." In: *IEEE Transactions on Visualization and Computer Graphics* 22.5 (2016), pp. 1605–1615. ISSN: 1941-0506. DOI: 10.1109/TVCG.2015.2446467.

[65] Josef Spjut, Ben Boudaoud, Kamran Binaee, et al.: "Latency of 30 ms Benefits First Person Targeting Tasks More Than Refresh Rate Above 60 Hz." In: *SIGGRAPH Asia 2019 Technical Briefs*. ACM Digital Library. New York, NY, USA: Association for Computing Machinery, 2019. ISBN: 9781450369459. DOI: 10.1145/3355088.3365170.

[66] Josef Spjut, Ben Boudaoud, and Joohwan Kim: "A Case Study of First Person Aiming at Low Latency for Esports." In: *EHPHCI: Esports and High Performance HCI*, Apr. 15, 2021. CHI '21. OSF Preprints, Apr. 15, 2021. DOI: 10.31219/osf.io/nu9p3.

[67] Shengmei Liu, Mark Claypool, Atsuo Kuwahara, Jamie Sherman, and James J. Scovell: "Lower is Better? The Effects of Local Latencies on Competitive First-Person Shooter Game Players." In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, May 2021. ISBN: 9781450380966. DOI: 10.1145/3411764.3445245.

[68] Jiawei Sun and Mark Claypool: "Evaluating Streaming and Latency Compensation in a Cloud-based Game." In: *Proceedings of the 4th IARIA International Conference on Advances in Computation, Communications and Services*. ACCSE 2019. Nice, France, Aug. 2, 2019. ISBN: 9781510894204. URL: https://web.cs.wpi.edu/~claypool/papers/drizzle/paper.pdf.

[69] Ben Boudaoud, Pyarelal Knowles, Joohwan Kim, and Josef Spjut: "Gaming at Warp Speed: Improving Aiming with Late Warp." In: ACM, Aug. 2021. DOI: 10.1145/3450550.3465347.

[70] Joohwan Kim, Pyarelal Knowles, Josef Spjut, Ben Boudaoud, and Morgan Mcguire: "Post-Render Warp with Late Input Sampling Improves Aiming Under High Latency Conditions." In: 3.2 (Aug. 2020), pp. 1–18. DOI: 10.1145/3406187.

[71] Niels Henze, Markus Funk, and Alireza Sahami Shirazi: "Software-reduced touchscreen latency." In: *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*. MobileHCI '16. New York, NY, USA: Association for Computing Machinery, Sept. 2016, pp. 434–441. DOI: 10.1145/2935334.2935381.

[72] Rosane Ushirobira, Denis Efimov, Gery Casiez, Nicolas Roussel, and Wilfrid Perruquetti: "A forecasting algorithm for latency compensation in indirect human-computer interactions." In: *2016 European Control Conference (ECC)*. IEEE, June 2016. DOI: 10.1109/ecc.2016.7810433.

[73] Axel Antoine, Sylvain Malacria, and Géry Casiez: "Using High Frequency Accelerometer and Mouse to Compensate for End-to-end Latency in Indirect Interaction." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI' 18. New York, NY, USA: Association for Computing Machinery, Apr. 2018. DOI: 10.1145/3173574.3174183.

[74] Reiza Rayman, Serguei Primak, Rajni Patel, et al.: "Effects of Latency on Telesurgery: An Experimental Study." In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2005*. Ed. by James Duncan and Guido. Gerig. Image Processing, Computer Vision, Pattern Recognition, and Graphics. Berlin, Heidelberg: Springer Berlin Heidelberg and Imprint: Springer, 2005, pp. 57–64. ISBN: 978-3-540-32095-1. DOI: 10.1007/11566489_8.

[75] Jon S. Kennedy, Marc J. Buehner, and Simon K. Rushton: "Adaptation to Sensory-Motor Temporal Misalignment: Instrumental or Perceptual Learning?" In: *Quarterly Journal of Experimental Psychology* 62.3 (Mar. 1, 2009), pp. 453–469. DOI: 10.1080/17470210801985235.

[76] Mircea Lupu, Mingui Sun, David Askey, Ruiping Xia, and Zhi-Hong Mao: "Human strategies in balancing an inverted pendulum with time delay." In: *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*. Buenos Aires, Argentina: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2010. ISBN: 9781424441235. DOI: 10.1109/iembs.2010.5626298.

[77] Mark Claypool: "On Models for Game Input with Delay — Moving Target Selection with a Mouse." In: *2016 IEEE International Symposium on Multimedia (ISM)*. IEEE, Dec. 2016. DOI: 10.1109/ism.2016.0125.

[78] Lothar Pantel and Lars C. Wolf: "On the impact of delay on real-time multiplayer games." In: *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*. NOSSDAV '02. ACM Press, 2002. DOI: 10.1145/507670.507674.

[79] Sunjun Kim, Byungjoo Lee, and Antti Oulasvirta: "Impact Activation Improves Rapid Button Pressing." In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2018. DOI: 10.1145/3173574.3174145.

[80] Burke Davison: *Techniques for Robust Touch Sensing Design*. Microchip Technology Inc., 2013. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/00001334B.pdf (visited on Nov. 1, 2021).

[81] Jack Ganssle: *A Guide to Debouncing*. Baltimore, MD, USA: The Ganssle Group, Apr. 2007. URL: http://www.ganssle.com/debouncing.htm (visited on Nov. 1, 2021).

[82] Julie A. Jacko, ed.: *The Human–Computer Interaction Handbook*. Third Edition. CRC Press, May 4, 2012. ISBN: 9781439829431. DOI: 10.1201/b11963.

[83] Microchip Technology: *AVR® ADC Noise Reduction Mode*. 2021. URL: https://microchipdeveloper.com/8avr:adcnoisereduce (visited on Sept. 5, 2021).

[84] USB Implementers Forum, Inc.: *Device Class Definition for Human Interface Devices (HID): Universal Serial Bus (USB)*. Version 1.11. USB Implementers Forum, Inc., May 27, 2001. URL: https://www.usb.org/sites/default/files/hid1_11.pdf (visited on June 8, 2021).

[85] Raphael Wimmer, Andreas Schmid, and Florian Bockes: "On the Latency of USB-Connected Input Devices." In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–12. ISBN: 9781450359702. DOI: 10.1145/3290605.3300650.

[86] Gerhard Gassler: "Cathode Ray Tubes (CRTs)." In: *Handbook of Visual Display Technology*. Ed. by Janglin Chen, Wayne Cranton, and Mark Fihn. Cham, Switzerland: Springer International Publishing, 2016, pp. 1595–1607. ISBN: 978-3-319-14346-0. DOI: 10.1007/978-3-319-14346-0_70.

[87] Karlheinz Blankenbach: "Panel Interfaces: Fundamentals." In: *Handbook of Visual Display Technology*. Ed. by Janglin Chen, Wayne Cranton, and Mark Fihn. Cham, Switzerland: Springer International Publishing, 2016, pp. 675–684. ISBN: 978-3-319-14346-0. DOI: 10.1007/978-3-319-14346-0_35.

[88] Steven LaValle: *Virtual reality*. 2020. URL: http://lavalle.pl/vr/ (visited on June 23, 2021).

[89] Jason Gregory: *Game Engine Architecture*. Third Edition. Boca Raton, FL, USA: CRC Press, May 29, 2018. ISBN: 9781138035454.

[90] Statista: *Display Technology*. Dossier. June 2020. URL: https://www.statista.com/study/35591/display-technology-statista-dossier/ (visited on Nov. 1, 2021).

[91] Karlheinz Blankenbach, Andreas Hudak, and Michael Jentsch: "Direct Drive, Multiplex, and Passive Matrix." In: *Handbook of Visual Display Technology*. Ed. by Janglin Chen, Wayne Cranton, and Mark Fihn. Cham: Springer International Publishing, 2016, pp. 621–644. ISBN: 978-3-319-14346-0. DOI: 10.1007/978-3-319-14346-0_33.

[92] H. Kawamoto: "The history of liquid-crystal displays." In: *Proceedings of the IEEE* 90.4 (2002), pp. 460–500. ISSN: 0018-9219. DOI: 10.1109/jproc.2002.1002521.

[93] Karlheinz Blankenbach: "Temporal Effects." In: *Handbook of Visual Display Technology*. Ed. by Janglin Chen, Wayne Cranton, and Mark Fihn. Cham: Springer International Publishing, 2016, pp. 3153–3176. ISBN: 978-3-319-14346-0. DOI: 10.1007/978-3-319-14346-0_146.

[94] David L. Woods, John M. Wyma, E. William Yund, Timothy J. Herron, and Bruce Reed: "Factors influencing the latency of simple reaction time." In: *Frontiers in Human Neuroscience* 9 (2015), p. 131. ISSN: 1662-5161. DOI: 10.3389/fnhum.2015.00131.

[95] Richard R. Plant, Nick Hammond, and Tom Whitehouse: "How choice of mouse may affect response timing in psychological studies." In: *Behavior research methods, instruments, & computers : a journal of the Psychonomic Society, Inc* 35.2 (2003), pp. 276–284. ISSN: 0743-3808. DOI: 10.3758/BF03202553.

[96] Akimitsu Hogge (Activision): *Controller to Display Latency in 'Call of Duty'*. In: Game Developers Conference (GDC), San Francisco, CA, USA, Mar. 2016. URL: https://www.gdcvault.com/play/1026327/GDC (visited on Sept. 5, 2021).

[97] Mark Mine and Gary Bishop: *Just-in-time pixels*. Tech. rep. Chapel Hill, NC, USA: University of North Carolina at Chapel Hill, 1995. URL: https://www.cs.unc.edu/techreports/93-005.pdf.

[98] Mark Segal and Kurt Akeley: *OpenGL 4.6 (Core Profile)*. Ed. by Chris Frazier, Jon Leech, and Pat Brown. The Khronos Group, Oct. 22, 2019. URL: https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf (visited on Nov. 1, 2021).

[99] Alexander O. Goushcha and Bernd Tabbert: "On response time of semiconductor photodiodes." In: *Optical Engineering* 56.09 (Sept. 2017), p. 1. DOI: 10.1117/1.oe.56.9.097101.

[100] Microsoft: *hidsdi.h header*. May 9, 2018. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/hidsdi/ (visited on Sept. 26, 2021).

[101] Microsoft: *WMI Core Provider*. May 31, 2018. URL: https://docs.microsoft.com/en-us/windows/win32/wmicoreprov/wmi-core-provider- (visited on Sept. 27, 2021).

[102] J. P. Verma: *Repeated Measures Design for Empirical Researchers*. Hoboken, NJ, USA: John Wiley & Sons, Aug. 21, 2015. ISBN: 9781119052692.

[103] Ragnhild Eg, Kjetil Raaen, and Mark Claypool: “Playing with Delay: With Poor Timing Comes Poor Performance, and Experience Follows Suit.” In: IEEE, May 2018. DOI: 10.1109/qomex.2018.8463382.

[104] Shengmei Liu and Mark Claypool: “Game Input with Delay – A Model of the Time Distribution for Selecting a Moving Target with a Mouse.” In: *MultiMedia Modeling*. Cham, Switzerland: Springer International Publishing, 2021, pp. 506–518. DOI: 10.1007/978-3-030-67832-6_41.

## Web pages

[WWW1] Jurre Pannekeet (Newzoo): *Newzoo’s Esports Consumer Predictions for 2021: A Quarter of the World’s Population Will Be Aware of Esports*. (Aug. 7, 2018). URL: https://newzoo.com/insights/articles/newzoos-esports-consumer-predictions-for-2021-a-quarter-of-the-worlds-population-will-be-aware-of-esports/ (visited on July 3, 2021).

[WWW2] TwitchTracker: *Rocket League - Statistics*. (June 11, 2021). URL: https://twitchtracker.com/rocketleague/statistics (visited on June 11, 2021).

[WWW3] Ian Nowakowski and Murty Shah (Psyonix): *Announcing RLCS X*. (July 1, 2020). URL: https://esports.rocketleague.com/news/announcing-rlcs-x/ (visited on June 11, 2021).

[WWW4] Ian Nowakowski (Psyonix): *Season 8 World Championship Regional Preview*. (Dec. 11, 2019). URL: https://esports.rocketleague.com/news/season-8-world-championship-regional-preview/ (visited on June 12, 2021).

[WWW5] Ian Nowakowski (Psyonix): *Introducing the RLCS X Championships*. (Apr. 15, 2021). URL: https://esports.rocketleague.com/news/introducing-the-rlcs-x-championships/ (visited on June 12, 2021).

[WWW6] Ian Nowakowski (Psyonix): *Introducing the 2021 CRL Spring Season*. (Feb. 25, 2021). URL: https://esports.rocketleague.com/news/introducing-the-2021-crl-spring-spring-season/ (visited on June 13, 2021).

[WWW7] Level Next – The College Esports League: *Announcing the Level Next Rocket League Spring Showcase*. (May 3, 2021). URL: https://levelnextesports.com/news/2021/5/3/announcing-the-level-next-rocket-league-spring-showcase (visited on June 13, 2021).

[WWW8] Tobias Seck (The Esports Observer): *Q1 2021’s Most Impactful PC Games: LOL, CS:GO, and Fortnite Stay on Top While COVID-19 Policies Continue to Upset the Ranking*. (Apr. 28, 2021). URL: https://esportsobserver.com/q1-2021-impact-index/ (visited on June 12, 2021).

[WWW9] Joab Gilroy (IGN): *Why Rocket League Might Be the Perfect Mainstream Esport: It's simple to follow, easy to learn and difficult to master.* (Aug. 29, 2017). URL: https://www.ign.com/articles/2017/08/23/why-rocket-league-might-be-the-perfect-mainstream-esport (visited on June 12, 2021).

[WWW10] Max Thielmeyer (Forbes): *'Rocket League' Is Poised To Become The Next Major Esport. Here's Why:* (Jan. 19, 2019). URL: https://www.forbes.com/sites/maxthielmeyer/2019/01/19/rocket-league-is-poised-to-become-the-next-major-esport-heres-why/ (visited on June 12, 2021).

[WWW11] Trent Murray (The Esports Observer): *Opinion: Rocket League Will Never Be a Tier One Esport, but it Can Dominate High School*. (Apr. 5, 2021). URL: https://esportsobserver.com/rocket-league-scholastic-esports/ (visited on June 12, 2021).

[WWW12] Battle(non)sense: *Netcode & Input Lag Analyses*. (2021). URL: https://www.youtube.com/playlist?list=PLfOoCUS0PSkXVGjhB63KMDTOT5sJ0vWy8 (visited on July 13, 2021).

[WWW13] Rocket Science: *(DS4 PC) Lower input lag for free! - Rocket Science*. (Jan. 28, 2020). URL: https://youtu.be/x0wcJM4FtXQ (visited on July 30, 2021).

[WWW14] Mark Rejhon (Blur Busters): *Preview of NVIDIA G-SYNC, Part #2 (Input Lag)*. (Jan. 13, 2014). URL: https://blurbusters.com/gsync/preview2/ (visited on July 30, 2021).

[WWW15] Loïc Petit: *Controller Lag. Methodology*. (Sept. 7, 2019). URL: https://inputlag.science/controller/methodology (visited on July 31, 2021).

[WWW16] Seth Schneider (NVIDIA): *NVIDIA Reviewer Toolkit for Graphics Performance*. (Sept. 4, 2020). URL: https://www.nvidia.com/en-us/geforce/news/nvidia-reviewer-toolkit/ (visited on July 27, 2021).

[WWW17] Rocket Science: *Does input lag matter? A RL experiment*. (Aug. 28, 2020). URL: https://youtu.be/yLcukmRfVt8 (visited on Oct. 31, 2021).

[WWW18] Rocket Science: *How Much Faster Does A SSL Shoot? (feat. SunlessKhan, FreaKii, Justuszzz)*. (June 5, 2021). URL: https://youtu.be/qKFoKD_Xk6I?t=99 (visited on Nov. 8, 2021).

[WWW19] Seth Schneider (NVIDIA): *Introducing NVIDIA Reflex: Optimize and Measure Latency in Competitive Games*. (Sept. 1, 2020). URL: https://www.nvidia.com/en-us/geforce/news/reflex-low-latency-platform/#nvidia-reflex-latency-analyzer (visited on Apr. 27, 2021).

[WWW20] Alps Alpine: *Alps Alpine's Stick Controllers (ThumbPointer™)*. (May 2020). URL: https://tech.alpsalpine.com/e/products/faq/multi/thumbpointer.html (visited on Nov. 1, 2021).

[WWW21] Daniel O'Keeffe, Dimitris Katsaounis, and Yannick Khong (rtings.com): *Acer GN246HL Bbid Monitor Review*. (May 28, 2018). URL: https://www.rtings.com/monitor/reviews/acer/gn246hl-bbid#page-test-results (visited on Sept. 18, 2021).

[WWW22] Kaldaien: *A Brief History of Swapchain Time (Why High Mouse Polling Rates Will Kill Framerate)*. (Dec. 6, 2018). URL: https://steamcommunity.com/groups/SpecialK_Mods/discussions/0/1745605598705601598/ (visited on Nov. 1, 2021).

[WWW23] Matthew Heironimus: *Arduino Joystick Library*. (Aug. 20, 2020). URL: https://github.com/MHeironimus/ArduinoJoystickLibrary (visited on Sept. 26, 2021).

[WWW24] Rocket Science: *RL input lag retested 1.44 – Rocket Science #15*. (Apr. 30, 2018). URL: https://youtu.be/-lFXLKv3DhM (visited on Sept. 18, 2021).

[WWW25] OfficialHalfwayDead: *ControllerMonitorInfo*. (Apr. 21, 2021). URL: https://github.com/OfficialHalfwayDead/ControllerMonitorInfo (visited on Sept. 27, 2021).

[WWW26] rtings.com: *#8 - New Input Lag and Response Time Tool*. (Sept. 21, 2017). URL: https://www.rtings.com/company/input-lag-tool (visited on Sept. 28, 2021).

[WWW27] Leo Bodnar Electronics: *Video Signal Input Lag Tester*. (2014). URL: https://www.leobodnar.com/shop/?main_page=product_info&products_id=212 (visited on Sept. 28, 2021).

[WWW28] Hardware Unboxed: *Massive Monitor Testing Overhaul, New Benchmarks, More Charts, Better Data*. (Aug. 6, 2019). URL: https://www.youtube.com/watch?v=081ccrxYwDo (visited on Sept. 28, 2021).

[WWW29] TFTCentral: *Input Lag Testing*. (Dec. 14, 2011). URL: https://tftcentral.co.uk/articles/input_lag (visited on Sept. 28, 2021).

[WWW30] ocornut: *Dear ImGui*. (July 20, 2014). URL: https://github.com/ocornut/imgui (visited on Sept. 30, 2021).

[WWW31] tarehart, digli, Noodleguitar, et al.: *Useful Game Values*. (Sept. 26, 2021). URL: https://github.com/RLBot/RLBot/wiki/Useful-Game-Values (visited on Oct. 4, 2021).

[WWW32] u/Psyonix_Devin (Psyonix): *Season 14 Rank Distribution*. (Oct. 13, 2020). URL: https://www.reddit.com/r/RocketLeague/comments/jaio07/season_14_rank_distribution/ (visited on Nov. 7, 2021).

## Games

[G1] Psyonix: *Rocket League*. Game [PC, PS4, PS5, Xbox One, Xbox Series X/S, Nintendo Switch]. San Diego, CA, USA: Psyonix (Epic Games since 2019), July 7, 2015. Used version: 1.78, from Aug. 28, 2020.

[G2] EA Vancouver and EA Romania: *FIFA 21*. Game [PC, PS4, PS5, Xbox One, Xbox Series X/S, Nintendo Switch, Stadia]. Redwood City, CA, USA: EA Sports, Oct. 9, 2020.

[G3] Konami Digital Entertainment: *eFootball PES 2021*. Game [PC, PS4, Xbox One]. Tokyo, Japan: Konami Digital Entertainment, Sept. 15, 2020.

[G4] id Software: *Quake*. Game [MS-DOS, Windows, Mac OS, Linux, AmigaOS, RISC OS, Sega Saturn, Nintendo 64]. New York, NY, USA: GT Interactive, June 22, 1996.

# Abbreviations

**ADC** analog-to-digital converter 30, 42

**ANOVA** analysis of variance 54, 56, 57, 62, 73, 75

**BM** BakkesMod 20, 24

**CPU** central processing unit 38, 39

**CRT** cathode-ray tube 15, 33, 34, 46

**EDID** Extended Display Identification Data 45

**FIFO** first in, first out 49

**FOV** field of view 1, 73, 79

**fps** frames per second 37, 42, 49

**G–G** Greenhouse–Geisser 57, 62

**GLM** General Linear Model 56, 62, 75

**GPU** graphics processing unit 38, 39

**HCI** human–computer interaction 8

**H–F** Huynh–Feldt 57

**JND** just-noticeable difference v, 13, 14, 64, 65, 72, 74

**JSON** JavaScript Object Notation 51

**LCD** liquid crystal display 34–36

**LED** light-emitting diode 12, 14, 47

**MMR** matchmaking rating 54, 99

**pctl.** percentile 54, 63, 64

**RGB** red, green, and blue 33, 34

**RL** Rocket League 1, 5, 7, 10, 19–22, 25, 38, 39, 43, 54, 66–68, 71, 72

**SPSS** IBM SPSS Statistics 27 56, 62

**SSL** Secure Sockets Layer 51

**TEO** The Esports Observer 5

**USB** Universal Serial Bus 31–33, 37

**VSync** vertical synchronization 34, 36, 40, 41, 44, 47, 48, 53, 77

# Glossary

**BakkesMod** User created Rocket League mod with an open API to allow developers to create their own mods for the game. vii, 20, 24, 48, 51

**first order control** The input (mouse/analog stick) directly determines the velocity (linear/angular) of the controlled object. 9, 10, 14, 17, 18, 21

**isotonic** sensing the angle of deflection, as opposed to isometric (sensing force) [82, pp. 106–107]. 30

**latency** Local system end-to-end delay/motion-to-photon latency (see section 1.2). v, 1–3, 6–25, 27, 28, 30–49, 53, 55–80, 99–101, 103

**mechanic** Game specific action that players can perform, e. g. aiming, shooting, or jumping (more info: section 1.3). 2, 7, 20, 67, 68, 80

**netcode** Refers to the networking code in games. How the networking code is designed to work with multiple players connecting, and how it deals with network latency, jitter and packet loss. 7

**Rocket League** Car soccer video game, published 2015 by Psyonix, used in the study (see section 1.1). v, vii, 1–3, 5–7, 10, 11, 19–22, 25, 27, 28, 35, 37–39, 41, 43–46, 49, 54, 66–68, 70–72, 74, 77, 79, 99

**second order control** The input (mouse/analog stick) directly determines the acceleration (linear/angular) of the controlled object. 9, 10, 14, 17, 18, 21, 72

**zero order control** The input (mouse/analog stick/touch) directly determines the state (location/orientation) of the controlled object. 9, 10, 12, 17, 19–21

# List of Figures

# List of Tables

# List of Listings

# Experiment setup Appendix

# A

## A.1 Detailed shot description

The five shots in the experiment are supposed to represent different setup, difficulty, and execution.

The first shot is the easiest. The ball rolls with moderately slow speed and a light bounce in front of the goal. The player is facing away from the goal and has to turn 60–120° in order to put the ball on target. Participants of all ranks are able to score this goal. It requires minimal estimation of the car's turning behavior, and there is lots of time for corrections. Precise placement of the shot is easily possible. Shot power requires a well-timed flip.

The second shot is more difficult in one way: estimation. The ball starts out high in the corner and falls with sideways speed in front of the net. Hitting the ball at the moment it bounces in front of the net, is easy as long as the player is capable of estimating when the bounce will approximately occur. This requirement is tighter for a precisely placed shot. Higher shot power is easier to achieve on the second shot, as the player is set up to shoot after the bounce. This is known as a power shot because it will reliably produce a powerful outcome. It is expected that higher rated players will get to the ball before it even bounces. This requires a high jump off the ground.

The third shot requires a lot more control from the player. The player starts close to the goal, facing away from it. The ball is in front of them, rolling away from the goal with moderately slow speed. The player needs to accelerate with boost in order to go all the way around the ball to shoot on target. There are three possible strategies. The first one requires great timing and involves executing the turn to perfection to score as early as possible. The second option involves driving deliberately far beyond the ball, and taking the time to turn. That allows the car to be faster for a powerful shot. The last option involves two touches. The first touch is deliberately soft in order to set up for the follow-up shot. I consider the two touch strategy to be the easiest and most consistent.

The fourth shot has the ball crossing the field diagonally flying past the goal. This is called a redirect shot, as the ball will miss the goal without a touch, but all that is needed is a bounce off the car to redirect the trajectory into the goal. This shot requires good judgement of the ball's speed. The high velocity at which the ball is flying from the start, means small errors result in misses. This makes it the most difficult shot for lower skilled players. They do not have the option to take the shot slowly. Due to the starting velocity of the ball, the touches will generally be faster than on the other shots. However, there is still significant room for different outcomes. While a poorly touched ball can go 80 $\frac{\text{km}}{\text{h}}$, a perfectly struck shot will go 160 $\frac{\text{km}}{\text{h}}$.

The fifth shot is intended to be an aerial shot. The ball is deliberately off to the left side, while the player is central before the goal. The ball gets launched high in the air. The player is expected to try and fly around the ball in a slightly curving path to strike it towards the center of the goal. In order to ensure that lower ranked players can complete the shot, it is possible wait for the ball to drop to the ground. This still requires driving around the ball, but it is significantly easier than the option in the air.