Using Facial Feature Recognition and Head Tracking to Control Games

Patrick Wouterse

Daily supervisor: dr. ir. Ronald Poppe
Project supervisor: dr. ir. Arjan Egges

Master thesis
Utrecht University
ICA-4000188
November 16, 2015

Abstract

Many interfaces exist to control computer games, both for desktop computers and for consoles. For desktop gaming, the most commonly used interface is the mouse and keyboard. Most alternative interfaces often require hardware that users do not already possess, such as gamepads, joysticks, a Kinect, or a Virtual Reality-device. If games want to use alternative interfaces they often have to be specifically designed to be able to do so. In this work is presented a novel interface 'FaceDriver' to control games. FaceDriver makes use of head tracking, and only requires a webcam. This interface is usable on a large number of existing games, as it allows games to interpret the interface as if it were an interface for which the game was designed. A user study is conducted in which 25 users play a racing game using this interface. The in-game performance of this interface is compared to traditional interfaces, and the user engagement is evaluated. The results show, that while the interface isn't competitive in terms of in-game performance compared to a mouse and keyboard, users do prefer FaceDriver over traditional interfaces.

Contents

1 Introduction

1.1 Goal

Many ways exist to control computers and, more specifically, computer games. Most of these require additional hardware not commonly possessed by most people, or are not usable while engaged in other activities. Allowing users to control games using only a regular webcam, playing games can become much more accessible, as a keyboard or mouse is not required to play. For example, it's often difficult to use a mouse without access to a desk, such as in a train. This can be accomplished by allowing the webcam to read a user's head movement, and then transforming this data into a format usable by existing games. Many people have cameras attached to their desktop computers, or have them built-in in case of laptops. This means that a large number of people would be able to make use of this technology today, and would allow them to experience a novel method to control games without requiring the purchase of additional hardware.

For this purpose 'FaceDriver' was developed. FaceDriver allows a user to control games using only their head movement, and can be used on any computer that has a camera attached. It achieves this by tracking a user's head and transforming this data into a method of input that existing games can understand. The software should work for the majority of people without the need for any prior calibration, as shown by the test approach and results. It should allow a wide variety of games to be controlled using only a persons head, assuming the game does not require more input factors than possible using head controls.

1.2 Approach

In order to allow users to use this method of input, an application was developed that allows an attached camera to behave as a gamepad would. Note that, normally, not every input on a gamepad is required in a game. This means only the used gamepad inputs have to be mapped to what a user can do using their head, simplifying the controls. Doing so allows users to use the camera to play games which are playable using a gamepad by default. The software tracks faces and detects various points on the face and the movement of the head itself, and transforms these so that games can be played with them.

To determine how users experience and handle controlling games using only their head movement, a user study was performed where users would play a racing game using the method of input. Users would race several laps in the specified game, while varying their method of input (camera or keyboard) and pose (sitting or standing) every few laps. For each lap, the laptime is recorded and compared to other laptimes. Data is also captured using a motion capture suit, which allows for an analysis on the movement of the user while playing. The user is also given various questionnaires, which allows the user's engagement to be measured, along with their prior experiences and also allows their

demographic to be determined.

1.3 Structure

This section describes the layout of the rest of this document. In section 2, related works are discussed in the current state of Human-Computer Interaction (HCI), Head tracking and facial feature recognition, and player engagement in games. Section 3 describes the application written for the experiment, the methods it employs, and its requirements. Section 4 describes the experiment performed in order to obtain the results, and the setup used for the experiment. It also presents the results of the experiment and the various conclusions which can be obtained from them. Section 5 states final conclusions, and Section 6 discusses the possibilities for any future work which could be performed.

2 Related work

In this section, related works are discussed. Section 2.1 Human-Computer Interaction (HCI) describes the current state of interfacing with computers and how they relate to the novel method of interfacing presented in this work. Section 2.2 describes the current state of head tracking technologies and why facial feature recognition is required for it, and how this work makes use of them. Section 2.3 describes in what ways user engagement can be measured and influenced by different methods of input.

2.1 Human-Computer Interaction

In order to interface with a computer, a human needs to be able to send input to it in some manner. For this purpose, many types of interfaces have been created in the past several decades. Nowadays, the most commonly used interfaces are the mouse and keyboard. A recent overview on Human-Computer Interaction (HCI) [1] shows that more and more interfaces are emerging, and that many of these interfaces have some form of continuous input. Interfaces which allow for continuous input, such as a joystick, allow for very fine-grained movements. Fully discrete interfaces, such as a keyboard, only have two states per key, either on or off. Other devices, such as gamepads, have both. On gamepads, the continuous input sticks are also known as the 'analog' sticks. Having more fine-grained control can greatly influence how well users can control games, and creates more possibilities for game developers due to not being limited to only two states.

Many new interfaces with some form of continuous input have recently appeared for gaming consumers. Interfaces that work by using full body motion tracking such as the Microsoft Kinect [2] (for Xbox), and interfaces that rely on the user to handle devices using their body movement, such as the Nintendo Wii-controller and the Sony Playstation Go. All of these result in the game using

continuous input. More recently, many Virtual Reality (VR) devices have surfaced, and these also track the user in various ways. The Oculus Rift (for PC) is a head-mounted VR device with 6DOF head-tracking (6 Degrees Of Freedom (6DOF), namely 3-axis positional and 3-axis rotational), allowing games that make use of the Oculus Rift to mimic the movements in game. It handles positional tracking using an external camera. Similarly, Sony's upcoming 'Project Morpheus' for the Playstation 4 is another head-mounted VR device, also has 6DOF head-tracking, and handles position tracking using the Playstation 4's included camera. the upcoming HTC Vive (for PC) is another such VR device, but achieves improved positional tracking due having dual base stations placed in the same room to prevent occlusion related problems. Several other VR devices have recently appeared, but share similar or identical methods of interfacing to those previously mentioned.

Many of the mentioned devices have a significant cost of purchase, or requires the user to wear them at all times while playing. This work aims to provide an interface requiring only a standard camera, such as a webcam. This means anyone who currently owns a webcam will be able to make use of this interface on a large number of existing games. Compared to similar interfaces, this work researches the effects on the user experience and in-game performance of motion-based gaming, rather than the robustness of the interface or making a comparison between different motion-based interfaces. In-game performance will be researched to determine to what extent less casual gamers might be able to make use of motion-based interfaces. Certain games that currently require more traditional interfaces such as a mouse and keyboard to achieve the required in-game performance might be playable with motion-based interfaces as well. User-engagement is measured to ensure games do not become less engaging when a user is playing with the motion-based interface used in this work, which might in turn affect in-game performance.

In order to make it possible for users to play games using a webcam, head tracking will be used. The user's head will be tracked, and user motion will be transformed in such a way that games can be played with it.

2.2 Tracking head position and rotation

In order to make use of head movement and head rotation in games, a face needs to be detected by the software. After finding a face, the software will be able to use the face to calculate the position and rotation of the entire head. In order to calculate the rotation of the head, facial features need to be recognized, as they need to be used for orientation.

Various methods exist for face detection. The challenges in this field relate to solving both the aspect of computational performance, and improving the recognition. Often, a trade-off is chosen between computational speed and detection

rate. According to a survey by Zhang et al. [3], the method introduced by Viola and Jones[4] had a large impact on how face recognition progressed, as it introduced several of the main modern ideas that make it possible to perform face detection in real-time. The main ideas this paper proposed are the integral image, classifier learning with AdaBoost, and the attentional cascade structure. Using variations of these ideas, a large number of new methods were proposed since then, extending to recent methods such as [5] which improves upon the original by making use of RealBoost instead of AdaBoost, and [6] introduced learning from image samples without requiring subcategory labels.

Facial features can be extracted using different methods. In 1978, Ekman and Friesen proposed the Facial Action Coding System (FACS) [7] to identify various facial Action Units (AUs) in the face, each corresponding to individual facial motions such as raising an eyebrow. These allowed for a standardized method of treating facial feature recognition. Since then, FACS has been widely used in various research relating to facial expressions [8][9]. More recently, systems have been developed to automate recognizing FACS AUs on human faces, both offline [10] and in real-time [11][12]. As is it nowadays possible to use this technology in real-time, games and other software can make use of facial feature recognition for input, and as such let games react to the user's facial expressions. In the experiment that follows, facial features will only be used indirectly, in order to compute the rotation of the head.

Several important differences exist between approaches to facial feature recognition. When detecting facial features, image patches are formed from local spatially-coherent image observations. These image patches are centred around landmarks within the facial region. Landmarks are predefined points that correspond to facial areas. Two major approaches exist to handle these image patches after obtaining them; holistic approaches, and independent approaches. Holistic approaches make use of previously obtained information (for instance, the Haar detection) to make an initial estimate for the image patches, making these require the Haar detection to give correct and complete results. Independent non-holistic approaches assume these image patches are conditionally independent of each other, and do not require the initial Haar detection to create the image patches. Independent approaches seem to show superior performance compared to holistic approaches, one of the reasons being that a holistic approach requires a full facial match on the Haar detection in order to make an estimate, preventing it from working well with partial occlusion [13][14][15][16].

The image patches are typically found using descriptors learned from manually labelled training data. Typical training data contains a wide variety of faces, both in terms of appearance and in visual quality, in order to allow the descriptors to match faces in a large number of situations, making it much easier for image patches to be placed on faces. However, due to lack of local support on the image patches due to their relatively small size and lack of context, image patches are often falsely assumed to be placed correctly. This problem is

increased by the strong variations in the training data, leading to significant ambiguity problems, often causing entire faces to be falsely detected.

While faster face tracking algorithms do not give perfect results, they are usable in real-time. This opens up a wide array of possibilities, including the control of games. In practice, obtaining an invalid result from a tracker is a temporary problem. If an invalid face is found, it tends to quickly go into an invalid state and will cause the tracker to lose track of it. Once the tracker finds the correct face it limits the 'scan region' to the area around the face, thereby preventing further issues caused by the Haar detection. This should make it safe to use for real-time application such as gaming.

2.2.1 FaceTracker

For the application developed in this work, the FaceTracker library, created by Saragih et al. in [12] was used. This library was chosen over others as it has a readily available implementation and is easily integrated. FaceTracker performs both real-time face detection and facial feature recognition, and provides information regarding the position and rotation of the head. FaceTracker provides a robust library that makes use of OpenCV to perform the face detection, and to perform facial feature tracking based on FACS.

In order to track faces and facial features, FaceTracker first attempts to detect a face using Haar detection, from the Viola-Jones work [4]. This works by decomposing the image into light and dark patches, and determining if the segments match a certain trained pattern. This allows Haar detection to quickly analyse an image and detect faces. Each segment has histogram normalization applied to prevent skin tone or lighting conditions from significantly affecting the outcome. Typically, this process detects several rectangular areas on a single face, with varying confidence per rectangle. In order to obtain the correct one from these, Non-maximum suppression (NMS) can be applied the rectangles, making sure non-optimal rectangles have significantly lower confidence. Afterwards, all rectangles below a certain confidence threshold can be removed, only preserving those rectangles that are sure to contain faces.

After a face has been detected, it can be analysed for facial features. First, it attempts to fit the Haar-patches onto certain facial landmarks based on the trained data. This results in very rough estimates of facial landmark locations, which is the response map. These locations are refined using the mean-shift algorithm, making use of a pre-constructed 3D mesh of the face, and neighbouring landmarks, leading to greatly improved accuracy. Several constraints apply to speed up the process, based on the trained set. This leads to several limitations but improves general performance. For example, eyebrows are generally assumed to be the same height, and the AUs associated with the mouth behave identically as well.

The vertices on the facial 3D mesh correspond to facial landmarks, and are adjusted when the landmarks are detected. This causes the 3D mesh to appear in the same manner as the detected face, essentially wrapping the face mesh around the detected face. This mesh can then be used to determine both the position and rotation of the facial mesh.

FaceTracker was trained on the MultiPie database[1], which is a large collection of annotated faces, posing a large variety of expressions. The faces in the MultiPie database are constrained to real-world situations, and therefore makes FaceTracker perform optimally during normal conditions and facial expressions. This includes many common gaming environments, such as rooms with overhead lighting. This makes it especially well-suited for use in the experiment.

2.3 Using bodily motion for game interaction

A person playing games with a mouse and keyboard requires minimal bodily motion in order to interface with the computer. Playing a game with a steering wheel or similar peripheral requires more bodily motion, and using full-body motion capturing systems such as the Kinect requires a large amount of movement. Bodily motion of players playing games has an effect on the player engagement, as shown in a study performed by Bianchi et al. [17]. This study makes use of the Game Engagement Questionnaire (GEQ) [18] to assess how engaged players are when playing a game on two different peripherals; a guitar, and a gamepad. The guitar requires significantly more bodily motion in order to control the game, and appears to result in increased engagement in the player.

Many ways of controlling games require only the hands; mouse, keyboard, gamepad, joysticks, etc. Others require more than only the hands; a racing wheel requires both hands and feet. Sony's Sixaxis is a gamepad with motion sensing and as such requires movements of the arms and upper body as well. Other methods require full body movement; notably those that directly track the player by means of a camera such as the Kinect, but also those that indirectly track players such as a dance pads which have several pressure points.

Natural user interfaces (NUI) are interfaces designed in such a way that they mirror exactly what a user is capable of, what the user expects, make full use of a user's capacities, and meet task and context demands exactly[19]. An ideal NUI would have a minimal learning curve, and should make users feel as if they've mastered the controls soon after. Many modern methods of interfacing are often placed in the category of NUI; touch, speech, handwriting, gestures, and vision.

The Kinect has been used to create natural user interfaces (NUI) by means of user gestures[20], where users perform certain motions to achieve specific ac-

[1] `http://www.multipie.org/`

tions. For example, the right hand can be made into a fist to perform an action such as 'scroll', while the left hand is moved upwards to specify the direction to scroll in. Technically, such an action requires only tracking of the hands rather than the entire body, which is what the Kinect does. Other gesture-based methods[21] only track the hands, and are able to achieve robust recognition of gestures using only a regular camera. This allows for simplified hardware, which in turn allows for a cheaper solution.

Using a NUI does have downsides. An important part of a NUI is being relatively transparent to the user, but still exhibiting complex behaviour behind the scenes. There are many ways in which a NUI can fail to comprehend the user. For example, speech recognition can fail to recognize what the user is trying to say, or a gesture-based NUI can fail to recognize the user motion as an existing gesture. Hardware such as a keyboard leaves very little room for mistakes, as pressing the wrong key is usually noticed quickly. Apart from pressing the wrong key, not much can go wrong. With gesture-based NUI, there is often no direct way of communicating failure to the user[22], leading to possible confusion for the user if something goes wrong.

Controlling games using head movement can be considered a gesture-based NUI, as it aims to provide users with a seamless and simple method to control games. Controlling games which are not natively designed for use by NUI prevents the game from being able to show feedback to the user in case the NUI fails in some manner, which means feedback should be communicated in some other manner. Playing games with head movement makes users consciously use parts of their body to directly control a game, rather than 'subconsciously' using their hands when controlling a game through a mouse and keyboard. This should lead to increased engagement when playing games. To prove this, a GEQ will be given to subjects during the user study. It is likely that users will not exclusively move their head when playing, and that movement of other parts of the body affects the GEQ score as well. Whether or not there is a relation between these is explored later in this work.

3 Using head tracking to control games

Before head tracking can be used to control games, several requirements need to be met, both for the head tracking software and any game that is to be used by this software. This section will describe both the requirements for the head tracking software, and for the game. It then continues by describing the method employed to achieve game control through head tracking.

3.1 Requirements and assumptions

In order to control games using facial features and head tracking, this data has to be collected and interpreted so that games know how to react to this input.

The section specifies which requirements the various elements have.

3.1.1 Head tracking

For the head tracking itself, several requirements exist:

- The tracker must run in real-time while the user is playing. For the tracker, this means 15 frames per second or more.
- The tracker must be able to detect the user on any standard webcam with a resolution of 480p or greater.
- The tracker must be able to correctly track a new user without requiring manual configuration beforehand (to increase transparency of the NUI.)
- The tracker must be able to detect the user regardless of facial appearance (glasses or facial hair.)
- The tracker must be able to track a user as long as the head fits on the camera view, and is not out of range (actual range greatly depends on lighting conditions.)
- The tracker must be able to detect the user as long as the lighting conditions are adequate:
  - The face is equally lit.
  - The luminosity on the face falls within the limits of the camera (not too dark or bright for the camera.)

If all these requirements are satisfied, the tracker should be able to reliably track the user's head, and determine the position and rotation of the head.

3.1.2 Application

An application is required that is able to track the user, interpret the data, and translate it into a form of input that the game can understand, and then forward the data to the game in some manner.

The head-tracking application itself needs to satisfy several requirements in order to be flexible enough:

- The application should show feedback to the user indicating if a head is currently being tracked (see section 2.3). Preferably, this feedback includes a visual indication of what the tracker is tracking, and how many frames per second the tracker is handling.
- The application needs to have configurable controls, so that any game can be controlled without requiring users to change code.

- The application needs to be able to map head-tracked input to both the gamepad's analog axis and buttons, by some means of threshold.
- The application needs to work without any per-user configuration or per-user calibration.

3.1.3 Game

In order to achieve an optimal experience using the head tracking software on games, the game should satisfy the following requirements:

- The game's controls should be translatable in some form to input that can be obtained from tracked data.
- The game needs to run in real-time (30 frames per second or more) on the PC on which the head tracking software is used.
- The game still needs to be able to run smoothly while the head-tracking software is running.

Every game that meets these criteria should work well with the head tracking software. Additional requirements for the game might depend on the design chosen to communicate the input to the game.

3.2 Implementation

Based on the requirements, software can be written. The design of the working system has been detailed in this section. First, an overview is given of the system and the separate layers. Afterwards, each individual layer is discussed.

3.2.1 Overview

As mentioned in the requirements, the application needs to be able to send input to the game, to allow the game to by controlled by the user's head. As this is one of the primary goals of the application, it is vital the application is able to send input in a reliable manner.

Two options exist to achieve this; either the game is modified in some way to read the input from the application, or the game is sent the input from the application without the game being aware that this data originates from the face tracker. For the latter option, it should be possible to emulate a virtual keyboard or virtual gamepad and let the game use that instead.

Modifying software requires either open-source software, which allows anyone to modify the software as they see fit, making it suitable for head-tracked input, or a closed-source game, which can be modified to accommodate for head-tracked input. Based on the availability and quality of open-source or otherwise modifiable games available amongst potential candidates for the experiment in section

4, the choice was made to use a virtual gamepad instead. A virtual gamepad was chosen over a virtual keyboard, as both the gamepad axis and tracked input are continuous and therefore give users more similar and accurate control over the game. Using a virtual device will allow the head-tracking software to send input to games that accept input in the form of gamepads or joysticks, without requiring any modifications to the game itself.

This leads to the following application stack:

- The face tracking library, which allows the application to track the face.
- FaceDriver, which uses the face tracking library to track a user's face, and transforms the data into a format that imitates a gamepad, and sends it to the driver.
- FaceDriver's device driver, which reads the data from FaceDriver and clones it onto the virtual gamepad so that games can use it.

The flow of the application stack works as follows:
As can be seen in figure 1, the user initiates the application start-up, after which it will start pushing updates to the operating system through the driver. After the operating system has received the updated virtual gamepad state, games are able to see this new state and behave accordingly.

3.2.2 Facial Feature Recognition and Head Tracking

To track faces, functionality is required to allow the application to perform face detection. Rather than writing this code manually, the choice was made to use an existing library, due reliability concerns and time constraints.

Based on the available choices in face tracking libraries, the FaceTracker library was chosen. This library was chosen over others because:

- FaceTracker runs in real-time, and can run at a high framerate (30+) if the lighting conditions are good enough.
- FaceTracker has existed for several years and has been widely used, both commercially and non-commercially, ensuring it is very stable software.
- FaceTracker is easily integrated into projects, due to being standard C++ code, and requiring only OpenCV.

The FaceTracker library is written in C/C++, so in order to make use of it, the FaceDriver application would need to either be written in C++ as well, or a small wrapper would need to be written in order to encapsulate the functionality. As the FaceDriver application mostly consist of GUI-related functionality, C#/WPF was chosen as framework for FaceDriver. This means FaceTracker would need to be encapsulated, so a small C++/CLR wrapper has to be written to encapsulate the FaceTracker functionality and convert various resources

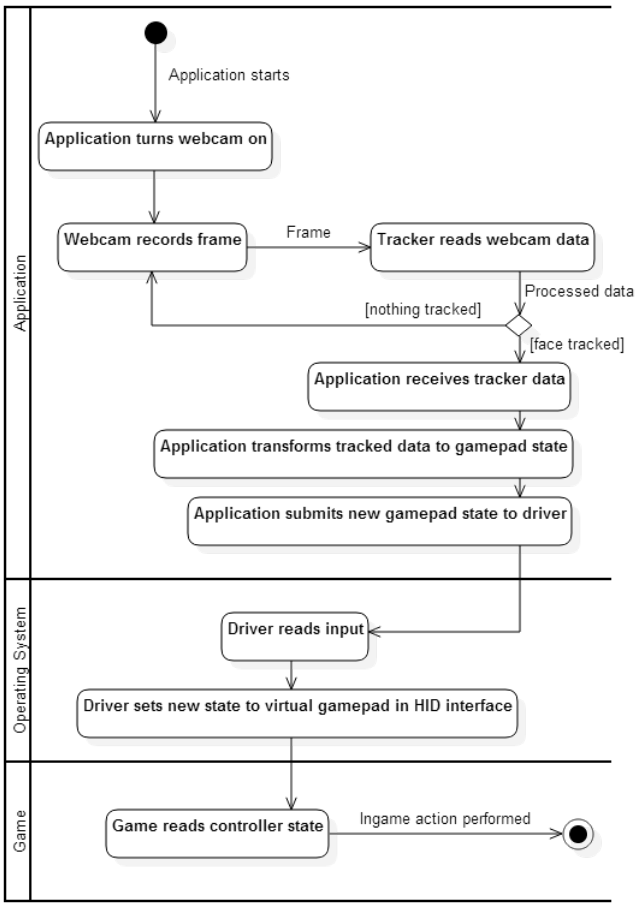

Figure 1: FaceDriver application flow

such as camera images and tracking data to a format compatible with C#/WPF.

When started by the user, the application begins obtaining camera images continuously, and processes them by using FaceTracker. Data relating to the head and face are then generated, namely head position, head rotation, and FACS data. Internally, when a face has been detected, FaceTracker transform a 3D mesh to match the current facial pose, which is superimposed in 2D upon the user's current camera image and always displayed in the application as feedback.

When controlling games, users tend to face the screen. As the camera is not placed in the center of the screen, there tends to be a slight offset compared to the 'natural' pose of facing the screen. FaceDriver makes no attempt to prevent this offset, as users often correct their pose within seconds of getting control of

the game. The time it takes for users to realize what FaceTracker's neutral pose is depends on the game; a racing game will quickly give the user feedback, as the neutral pose lets the user steer straight ahead.

Calibration would solve the problems caused by the offset of the head's rotation. However, the ability to start playing without any prior calibration is important, and as such the application has been designed without requiring users to calibrate. The ability to play without calibration makes it far easier for users to begin playing, and allows users to switch. If the application would require calibration, each and every user would need to re-calibrate the application before playing. This, in turn, would require the user to quit or if possible switch away from the game, which can be a lengthy process and greatly interrupt the gameplay.

3.2.3 Virtual gamepad

In order to create a virtual gamepad, a Human Interface Device (HID) device driver for Windows had to be written. HID is a standard[2], allowing devices to interface with operating systems in a standardized way, creating a common interface for many types of keyboards, mice, gamepads, joysticks, etc.

In order to get a new device to interface with Windows, it needs to be able to 'talk' to the operating system. The only way to achieve this is by means of a driver, as drivers give access to functionality not normally available within programs in Windows. There are two layers that need to be passed through to get the data from devices to applications; the hardware/kernel layer, and the user-mode layer.

The hardware/kernel layer requires a kernel driver to interface with the hardware directly, and control hardware components in order to get information from the device, and into the driver. Once the kernel driver has obtained the data from the hardware, it can be forwarded through the operating system to send the data to the user-mode driver. If the device is a USB-compatible device this process can often be simplified by using existing Windows drivers that allow interfacing with USB-compatible devices.

After obtaining the data in the kernel layer, a user-mode driver can receive the data. The user-mode driver will be able to decide where the data goes and format it into an HID-compatible layout, so the operating system understands which type of device is sending the data, how many buttons the device has, how many continuous inputs the device has, and several other parameters. The driver will also be able to notify the operating system of changes to the device's input and update the values.

[2] `http://www.usb.org/developers/hidpage/`

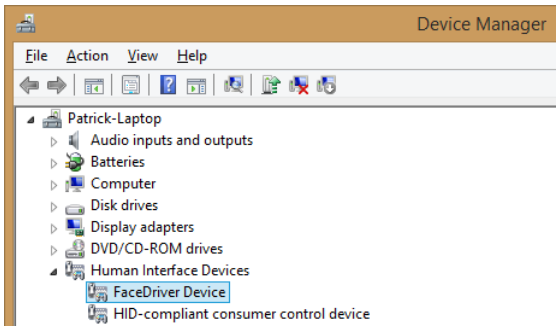

Figure 2: FaceDriver Device listed in Windows' Device Manager

Finally, in case of gamepads or joysticks, the operating system will provide the input to games that request it. The operating system will provide the HID data to the API used to receive input in the game, and the game will be able to act on it.

The driver used to supply input to games from the head tracker is a user-mode driver, and is not directly accompanied by any device. Having the driver as a user-mode driver has several advantages over a kernel driver; a kernel driver is loaded as part of the Windows kernel itself, meaning, if the driver is being debugged, it will suspend the entire operating system, making it inoperable. Similarly, any crashes in a kernel driver will cause the entire operating system to crash. User-mode drivers do not have these issues as they run as part of a regular user-mode process, and will therefore only suspend or crash the process, rather than the entire kernel.

Every user-mode driver needs to be accompanied by a kernel driver, so Windows knows what type of driver is being supplied and what tasks it can be expected to perform. Windows 8.1 brings UMDF 2.0[3], which is a new, more flexible, user-mode driver framework. Windows 8.1 also brings a standard HID kernel minidriver, allowing the creation of a user-mode driver for the head tracker that makes use of the built-in HID minidriver for the kernel part. Making use of an existing built-in kernel driver greatly simplifies the process, as the existing kernel driver can be expected to be stable, and prevents the need for any kernel-related development. Prior to Windows 8.1, neither UMDF 2.0 nor the HID minidriver existed, which is why the decision was made to only support the head tracking application on computers running Windows 8.1 or later.

Since the driver used for the tracker is not directly accompanied by a device, no actual data will be received from the kernel driver and as such all the data sent to the operating system from the user-mode driver will need to be obtained

[3] `https://msdn.microsoft.com/en-us/library/windows/hardware/dn384105(v=vs.85).aspx`

from other sources. To obtain the data, a separate tracking application runs and sends this data to the driver, which will then forward it to the operating system. See figure 1 for an overview.

3.2.4 Application

An application was written to provide the following:

- Initiate the head tracking process.
- Give head tracking feedback to the user.
- Let the user configure how the head tracked data should be interpreted by games.
- Send the data to the driver.
- Record various data for experiments.

After initiating the head tracking, the application will communicate with the C++/CLR wrapper to tell FaceTracker to begin tracking. The current camera image is displayed to the user, and if a head is found, feedback of where it has been found will be displayed to the user by means of superimposing a triangulated version of the face on the camera image. The amount of frames processed per second will be displayed below the image.

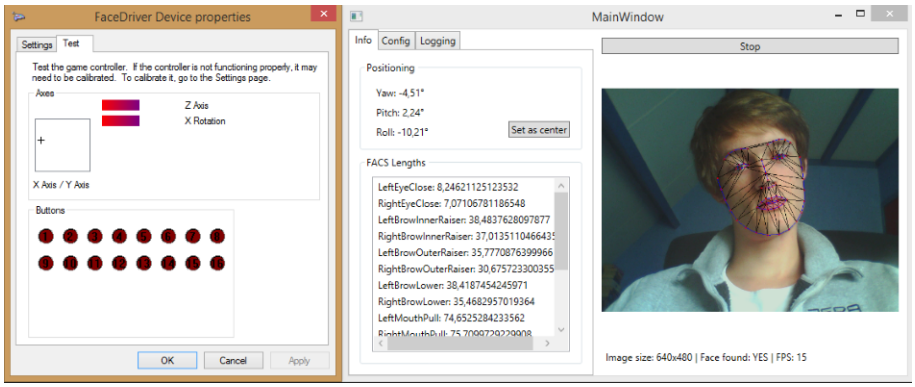

Figure 3: Left: Windows USB Gamepad setup. Right: FaceDriver application GUI

In figure 3 the GUI can be seen, tracking a user with his head turned towards the left. Next to the GUI is a window showing the Windows USB gamepad setup window, where the controller indicates the primary stick is now also turned towards to left.

Every frame in which FaceTracker has successfully tracked a face will cause the data about head position and rotation to be updated. After sending the frame to the application, the application requests this data and displays it to the user, while transforming it internally into gamepad data that can be used by the driver and game.

Data is transformed from head-motion to gamepad data by using a schema the user can configure in one of the setup pages. It allows for several ways of configuration; users can transform head rotation directly into an analog axis assisted by a multiplier (to allow for changes in sensitivity or inverting axis), or users can transform head rotation into button presses by setting a certain threshold in angle, which, when exceeded, will cause the button to be held.

Users can also use FACS data to either control an axis or button presses. This allows the user to use the lengths given by the FACS data in a similar manner as head rotation, meaning, either directly translated into an axis assisted by a multiplayer, or able to hold a button down if it exceeds a certain threshold in distance. The distance used by FACS has no consistent unit of measurement and can only be used if compared with calibrated data. Using FACS data in the application was not used in the experiment due to inconsistencies in tracking caused by lack of calibration, and therefore not observed in the rest of this document.

When the application has had its transformation scheme configured and is tracking a head, it will proceed to submit an update to the driver every frame. It does this by having a common section of memory with the driver, and storing the transformed data there. In Windows, it is possible to create a shared memory view, allowing applications and drivers to create a piece of memory accessible by name. Upon start-up, the driver will create the shared memory view using a name known both to the driver and the application, and along with the memory create an event that will allow the application to signal the driver to let it know the shared memory has been updated. The application will perform similar actions on start-up, and open both the shared memory and event.

For every frame that has been processed by the application, the application will write updated information to the shared memory and signal the event. Upon signalling the event, the driver will wake up and proceed to send this data to the operating system, allowing it to be used by games, and then waits for the next event to occur.

If, at any point during the tracking, the tracker either loses the user that was being tracked or begins tracking a new user, the application is notified and is able to take action. In case of the tracker losing track of a user, it might make sense in certain cases to set the gamepad inputs to the default status, rather than preserving the last known input. In case of a racing game, this means the car will stop moving and steer straight ahead, and by doing so give clear

feedback to the user that the tracker lost track. As this behaviour might not be desirable in every possible game, this has been made a configurable option in the application.

3.2.5 Use in games

In order for a game to work with the head tracking software it needs to be able to take input from gamepads. The head tracking software transforms head tracking data to simulate a gamepad, and therefore requires the game to accept input in this form. Furthermore, in order to provide input to the game within reasonable limits, the game should be able to run on at least 25 frames per second while the tracker is running. Less than this will cause an unreasonable amount of input delay and make it very difficult to control the game.

By itself, head tracking provides a very limited set of inputs. In theory, it provides full 6DOF tracking; rotation on 3 axis, and movement on 3 axis. Realistically, rotation is limited to only two axis, as FaceTracker does not have reliable tracking when varying yaw, and users might be limited in movement by their environment, giving only pitch and roll as usable inputs. While FaceTracker does not always have issues tracking yaw, the loss of both visible facial surface area and symmetry makes it difficult for FaceTracker to keep tracking in all cases. In a more optimized scenario, such as when using this technology in a professionally controlled environment with ideal conditions, it should be possible to make use of all 6 degrees of freedom.

Nearly every game requires input from more than just continuous controls, especially in order to navigate through menus. Often, button presses are required to start the game. While it is possible to use one of the DOF to trigger a button press, these are likely more useful as an analog axis during gameplay, and therefore not available to control buttons. In these cases, keyboards or actual gamepads can be used to provide additional input, and only once the gameplay has started the user can switch to using head tracking. If games would be designed with head tracking in mind, it would be possible to navigate the menus and start playing without the use of actual button presses, as can be seen in games designed for the Kinect. These games often require the user to move a virtual cursor around on the screen by perpetually giving motion input, and allow for button presses by letting the cursor hover over the button for several seconds. Similar methods could be employed by games designed for head tracking.

Games best suited to head tracking solely require continuous input, and do not require any button-presses between parts of gameplay, thus minimizing interruptions for the user. Good examples are racing games and flight simulators, as these can be controlled with minimal button presses (assuming gears and such are automated), and can often have long gameplay sessions with minimal interruptions for the user.

If the application would be extended to allow multiple users to be tracked, this technology might be viable for use by groups of players. Gamepad-controlled games tend to have game-modes where multiple players are able to simultaneously play on the same screen, making these games good candidates for head tracked controls, as a single camera can be used to accommodate as many players as can fit in the recording. Most other types of input require a physical device per player, which tends to be problematic in larger groups.

4 Experiment: Controlling Trackmania Forever using head tracking

In this section, the experiment will be described. First, the goal of the experiment will be described, followed by a description of each individual part of the experiment. After that, the results are evaluated and conclusions are formed based on the experimental results.

The experiment consists of a user study in which users will play a racing game using the keyboard and FaceDriver. From this, both objective results such as their in-game performance, and subjective results such as their user engagement will be measured.

4.1 Goal

Several hypotheses and sub-hypotheses can be formed on the use of FaceDriver:

4.1.1 Hypothesis 1

1. Playing games using FaceDriver will allow a user to get similar in-game performance compared to playing with a keyboard.
   (a) When using FaceDriver, users who move less will have an improved in-game performance.
   (b) Users with prior experience in body-controlled gaming perform better in-game.

Hypothesis 1 investigates if the FaceDriver interface can replace the keyboard while remaining a competitive interface for gamers to play on. It also investigates that the game difficulty will not significantly increase due to the controls and as such ensure that games remain playable. This is important, as FaceDriver is meant to replace existing interfaces for games rather than be used for games designed specifically for use with FaceDriver. It is likely that similar or better performance can be obtained compared to using a keyboard, as the head tracker should offer more accurate and responsive movement.

With FaceDriver, a user playing will be able to achieve greater in-game accuracy than with a keyboard due to having access to a continuous form of input, and the ability to directly specify the value of an axis. This is unlike the keyboard, which is limited to only two states. If the user is playing a racing game, the user will have to steer the vehicle through bends. On a keyboard, it is possible to either steer straight forward, or fully left or right. If a user with a keyboard would want to take a gentle bend, it can only be done rapidly tapping left or right. With a continuous interface such as the tracker, this is far simpler, due to being able to directly indicate how much steering is desired. Because of this, users should have a greater potential to perform accurate tasks in games.

The time it takes for a user to be able to react to events on the monitor is known as the response time. This is usually the sum of the user's reaction time, the processing delay of the input interface, and the time it takes for a game to process the new input. In case of 'choice reaction time' (CRT), human reaction time is 55ms[23]. The human reaction time can be divided into three groups. Out of these, CRT is the only group that requires the user the make a choice before reacting. Most games require the user to make choices before acting, so CRT should be the appropriate measure. The game's framerate is in most cases at 60Hz, meaning the input is processed 60 times per second (every 16ms). As the input can arrive anywhere in the frame, it means the time it will take for the game to pick it up is 0ms to 16ms. There may be an additional frame of delay between obtaining the input and processing it in games, which means it can be processed anywhere between 0ms and 32ms, giving an average of 16ms. The processing time for the tracker is limited by FaceTracker, which captures the camera's image and processes it. In the testing environment, this seems to be performed at a rate of 20Hz-30Hz (41ms on average), however, multiple images might be in flight at the same time. Measuring the delay between head movement and the corresponding movement of the tracked head inside the FaceDriver application shows an actual delay of 120ms when the tracker is running at 15 frames per second. For a keyboard, the average response time is 30ms[24], but greatly depends on the model of the keyboard. In total, this means using FaceTracker gives the user a reaction time of on average $55 + 16 + 120 = 191ms$. For racing with a keyboard, this is on average $55 + 16 + 30 = 101ms$. This means using the tracker gives users double the response time, which should be noticeable to the user in most cases. For a racing game, a slightly increased response time should not have a large impact on the performance, as any required input can often be anticipated some time in advance.

It is possible for the tracker to lose track of the user, making it temporarily impossible for the user to control the vehicle. Every time this happens, the user will significantly decrease their in-game performance. However, this can be considered to be part of learning how to use FaceDriver. A sufficiently experienced user will not behave in a way that causes the tracker to lose track, as the user will be more aware of the invisible 'bounds' where the tracker will lose track and try to stay in them. Staying within these bounds means not moving too far

too fast, or rotating too far. Experience alone does not prevent the tracker from losing track, as another important factor is how visible the face is. A partially obscured face (such as with facial hair, glasses or a scarf) can lead to worse tracking.

Hypothesis 1a states that the more a user moves, the less accurate their movement will be. Users who make large or rapid movements are likely to move too far, and in turn will try to compensate. Compensation usually requires a quick response to correct the initial mistake, leading to a rapid movement which again leads to the user moving too far. This causes the user to repeatedly overcompensate and unable to find a balance. Users who make minimal movements are likely very precisely controlling the tracker, and therefore obtain better results in-game.

Hypothesis 1b states that users who have previously played games using body-controlled interfaces will perform better with FaceDriver. This should be due to similarities, such as knowing how to stand, how responsive the interface is, and how accurately it performs. Being able to apply this prior experience should give better in-game performance with this interface as well.

4.1.2 Hypothesis 2

2. Playing games using FaceDriver will increase the user engagement compared to playing with a keyboard.

    (a) When using FaceDriver, users that move more have an increased user engagement.

Studies [17][25] show that increased movement leads to an increased user engagement. Tapping on a keyboard causes considerably less movement than using a face tracker, which means that using a tracker increases the user engagement due to increased movement. Hypothesis 2b continues on the same idea, stating that the more a user moves, the higher the user engagement will be, even if using head tracking the entire time. Both standing and sitting users will be analysed, and the movement of their entire body will be taken into account. The movement values obtained from the motion-capture suit will be checked against the movement values recorded by the head tracker to ensure there is a correlation.

4.1.3 Hypothesis 3

3. Users prefer FaceDriver over traditional keyboard controls.

    (a) Users who play games less are more inclined to prefer FaceDriver over users who play games more often.

Hypothesis 3 states that users prefer the FaceDriver controls. Both the novelty factor and increased user engagement should cause users to prefer motion-based controls over the traditional keyboard.

Lastly, gamers who play on a keyboard more often are more skilled using a keyboard. While it might be possible for gamers to achieve similar in-game performance after practising, this will not be the case initially due to their skill on the keyboard. Gamers who play more often will also be more likely to play for in-game performance, rather than playing for the novelty factor. Because of this, it can be expected that gamers will also try to use the tracker to play for in-game performance, and feel that this is much easier to achieve using a keyboard. This will lead to gamers who game a lot preferring the tracker less than users who play games less often. Hypothesis 3a will validate this assumption.

4.2 Experimental setup

In order to obtain data, a user study was be performed. It was decided to perform the experiment in a central location, under supervision, to reduce potential configuration errors by users, and allow for more accurate experimental control. In this section, the setup of the experiment is described.

4.2.1 Game

In order to let users play a game using the tracker, a game needs to be selected. In addition to the requirements stated in section 3.1.3, several other requirements can be introduced in order to allow actual results to be obtained from a game:

- The game has to be deterministic. Every user should face the exact same challenge to make comparing results possible.
- The game needs to have easily measurable data so that results can be obtained about the user's in-game performance. This means the game needs to have a clear and measurable goal, which excludes many sandbox, simulation, and strategy games.
- The game should be easily understandable, even by users who have never before played any such game, to reduce learning effects and decrease the amount of time a user needs to spend testing.

Any game that meets these requirements is in theory suitable as a candidate for the user study. A large number of racing games meet these criteria, as they are often easily controllable and have a very short learning curve compared to more complex types of games. The controls of a racing game also translate intuitively to the rotation of the head.

Specifically, the game that was chosen for the user study was 'TrackMania^{TM}Forever' (TMF), a freely available racing game[4]. This game was preferred over other racing games for the following reasons:

- TMF has very low hardware requirements, allowing a large range of hardware to be used for testing.
- TMF does not obstruct the racetrack with other vehicles, allowing for an identical experience every lap.
- TMF allows for the creation of new racetracks, which makes it possible to create a very suitable track for the user study.
- TMF is freely available. This makes it easier to allow users to perform testing at home, if required.

As the experiment is going to be conducted only in a supervised environment, the game and software can be pre-configured. In order to optimally measure the user performance, a custom track was created. The track focuses on preventing the user from getting 'stuck' and thereby preventing the laptime from increasing by an unpredictable amount. This could easily lead to most of the lap being spent on a single point where the user got stuck, rather than a proper indication of how well the user performed in general that lap. The final track used for the experiment contains mostly straight pieces of track and soft bends, and an open section where the user can drive unguided for several seconds. The track contains no square corners and will therefore let users easily correct their course when they have come to a halt. The track contains a minimal amount of ways to get 'lost' by falling out of the racetrack and being forced to restart to the last checkpoint, again to prevent laptimes from increasing unpredictably. The track is relatively simple compared to the default tracks, and is likely to give users a better in-game performance relative to the keyboard compared to the performance that would be obtained on default tracks. However, the track is easier for both keyboard and FaceDriver, so while the difference in in-game performance between the interfaces might be decreased on this track, the track should not favour a specific interface.

Visible in figure 4 is the entire custom racetrack as used in the experiment. The green gate is where the lap starts, and the red gate is where the lap ends.

4.2.2 Measurements

Users will be performing the same experiment on both the tracker and the keyboard in order to be able to measure the differences between them. This allows for an accurate comparison. Similarly, users will be performing the experiment with the tracker, both standing and sitting, to allow for comparison between a situation in which the user is free to move, and in which a user is restricted from

[4] `http://trackmaniaforever.com/`

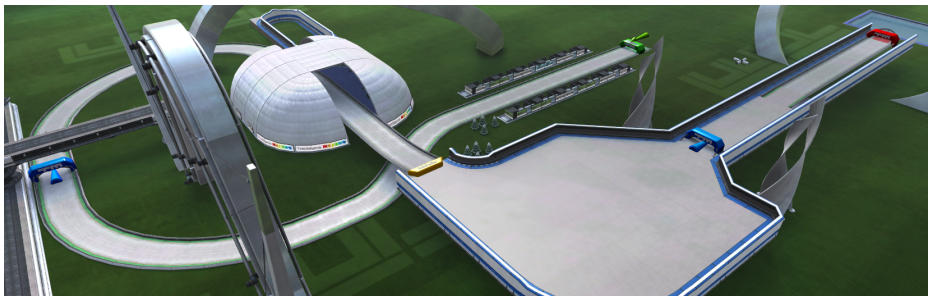

Figure 4: The entire custom racetrack.

Figure 5: The view when racing, in front of the finish.

moving. Measuring these differences is required for the validation of hypotheses 1 and 2.

For hypothesis 1, the in-game performance will be measured. In the case of a racing game, this can be measured by the lap time. Several laps need to be played per user in order to get sufficient data. Each user will perform the same amount of laps using the keyboard, using the tracker while sitting, and using the tracker while standing. The average lap time of each condition will be used for the analysis.

Users will fill out questionnaires in order to measure their user engagement, opinion regarding FaceDriver, and any past experiences with similar interfaces. At the very start of the experiment, users will answer the 'intro' questionnaire (see appendix A) to determine their past experiences with similar interfaces and similar games. After completing the experiment, users will answer the 'exit' questionnaire (see appendix B) regarding their opinion on FaceDriver and whether or not they prefer FaceDriver over traditional methods of input. These questions have been added to make it possible to measure how the user experienced the controls in that specific round.
These questions will allow hypothesis 2 and 3 to be tested. To test hypothesis 2,

the user engagement will additionally be measured after each round of testing using another questionnaire.

The questionnaire that is given to the user after each round (see Appendix C) gives data about the user engagement and controls. The first part is based on the Game Engagement Questionnaire (GEQ) [18], with several of the questions modified to make it suit better to the current experiment. The questions posed in this questionnaire are based on the Likert scale [26], and have a scale from 1-7, where 1 is strongly disagree, and 7 is strongly agree.

The questions that have been removed compared to the original GEQ are related to non-relevant emotions (such as 'I feel scared'), questions that require the experimenter to actively engage the user during testing (such as 'If someone talks to me I don't hear'), and questions that cannot be answered properly due to how the experiment is scheduled (such as 'I play longer than I meant to').

A motion-capture suit will be used for hypotheses 1a and 2a to determine how much movement a user makes while playing. Without a motion capture suit, only the motion of the head can be measured, which is only a subset of the data a motion capture suit would provide. As the hypotheses require analysis of the movement of the user as a whole, and not only of the head, the suit is required.

Making users wear a motion capture suit during testing will allow the capture of their body movement. While wearing the motion capture suit itself is not required for the tracker, it is used to collect more data about the user's movements during the testing. In section 4.2.2 is described for which parts of the experiment this data will be used. Note that, for this experiment, movement of the lower body is not recorded, as it was not deemed relevant.

The motion capture suit is a lightweight wireless suit by Xsens, minimally interfering with the testing process. The suit software needs to be started and stopped after each testing round, and needs to be calibrated prior to any testing.

The motion-capture suit continuously measures the position of certain parts of the body, and records this data at 30FPS. In order to calculate the user's final amount of movement from this, the recorded samples from the motion capture suit will be taken, and the total delta between subsequent samples will be used as movement, as follows:

$$m_t = \sum_{t=1}^{n} p_{i,t} - p_{i,t-1}$$

For the motion capture suit, p denotes a captured point, such as the left or right hand. The delta is calculated as the euclidean distance between the current position, and the position at the previous sample. This results in movement m_t

Figure 6: Left: The entire setup, with a racing user. Right: (edited for clarity) The left screen shows the game, at which the user is looking, the right screen shows the tracker and how it is currently tracking the user.

for the current sample. After obtaining m_t, the average movement of the entire round can be calculated as follows:

$$m_{total} = \frac{\sum_{t=1}^{s_n} m_t}{s_n - 1}$$

Where s_n denotes the total sample count, and m_{total} denotes the final movement. By having transformed the recorded data into a single variable, the data can now be used in the analysis. Note that the first and last 10 seconds of the recorded data have been stripped to account for the delays between the racing and the starting and stopping of the recording. Normally, the recording would be started first, and then the racing would begin several seconds later. Similarly, after the user has finished racing, it takes several seconds to stop the recordings. Stripping both ends of the recording takes care of these inaccuracies. Note that the recording did not pause between the 1-5 seconds of idle time between laps, during which users had the ability to look around freely. Due to the short idle time, this was limited, but might have introduced slight bias in the recorded data.

4.2.3 Procedure

Upon starting the experiment, users will first need to fill out the 'intro' questionnaire A. If a user has been selected to wear a motion-capture suit, the user will put it on at this point. Selection is based on whether or not the suit is available for usage at that point (unpredictable technical issues prevented this on occasion). If possible, the user will wear the suit. After that, the user will perform three rounds of testing. Each round consists of racing 5 laps in the game, and filling out a questionnaire containing questions on how they experienced that round. After each lap, the experimenter records the laptime and

initiates the next lap. Each round, the user will play the game differently. Each of the following methods needs to be played by each user; sitting using the keyboard (henceforth referred to as 'keyboard'), sitting using FaceDriver ('sitting'), and standing using FaceDriver ('standing'). The order of the rounds is chosen randomly to eliminate any bias in the test results that might be caused by users gaining experience in the earlier rounds, and as such performing better in-game. After the testing is complete, the user is given the final 'exit' questionnaire to assess the entire experience.

Users were not given any laps to race prior to the experiment in order to be able to better measure the learning effect on the motion-controlled interfaces. Giving them time to race prior to their first experience with the motion-controlled interfaces would diminish the visible effect. This does mean users did not have the ability to get used to the game itself either, but using any specific interface to allow them to do this would bias the results and as such was not possible.

The experiment is aimed to take roughly 40 minutes per user, up to an hour if a user is using a motion-capture suit. Up to 5 minutes should be reserved for introductions, explaining what the user has to do, and letting the user fill out the first and last questionnaires. Users were verbally explained and shown which motions to make prior to starting. When using a motion-capture suit, putting on the suit, calibrating, and getting the suit off again after testing can take up to 15 minutes extra. This leaves roughly 25 minutes for testing, meaning 8 minutes per round of testing. This includes having the user fill in the survey after each round, leaving roughly 5 minutes of time in which the user can play.

As each round of testing is aimed to be 5 minutes on average, the track cannot take too long to complete. Additionally, the user should race several laps in order to get usable averages. Racing 5 laps should allow the user to establish a reasonable average time, so the track should not take longer than a minute to complete on average. The track has been tested to take roughly half a minute on an optimal lap, so the total in-game time should be below the time reserved for testing.
In order for the game to be controlled by the head tracked interface, the head tracking application needs to be configured and running. To get the software to a usable state, minimal setup is required. A computer with an attached camera needs to be used, and the 'start' button needs to be pressed in the program. After this, the program will automatically recognize a user that appears in front of the camera. Optimal performance is achieved when the requirements stated in section 3.1.1 are met. Generally, this means having adequate lighting conditions, making sure the camera is level, and having the user directly in front of the camera. As the experiment is performed in a controlled environment, all these conditions can be met.

When a user is recognized, the software will begin sending input to the virtual controller, and therefore to the game. For TMF, the user only needs to steer left

and right, and control the speed of the vehicle. Other actions such as resetting to a previous checkpoint were handled by the experimenter when deemed necessary. These directions can be naturally mapped to the head, by mapping the steering to rolling the head left and right, and mapping the speed of the vehicle to changing the pitch of the head. Note that movement using the yaw of the head is not used, as testing showed that this felt less natural and that FaceTracker has issues identifying heads turned beyond a limit of roughly 20° in yaw from the center.

4.3 Evaluation and results

The experiment consisted of 25 users playing a game using FaceDriver. The majority of subjects consisted of bachelor and master students from IT-related studies, and were verbally asked to participate in an experiment by 'playing a game'. No compensation was offered for participating. The subjects were aged 19 to 28, apart from 3 subjects which were aged 35 to 50. Of the participating subjects, 20 were male and 5 were female. The motion capture suit was successfully used to record data for 6 users while sitting, and 9 users while standing.

The amount of time subjects spent gaming greatly varied, 4 subjects did not play any games, and 7 of the subjects spent at least 11 hours per week gaming. On average, subjects spent 8 hours per week gaming. 15 subjects had prior experience with body-controlled gaming devices such as the Kinect, 7 subjects had prior experience with the exact game being used for this experiment, and 1 subject had never played a racing game before.

The experimental results will be analysed in this section in order to compare results with the expectations and draw conclusions. First, the data will be analysed. After results have been determined, the results are examined for correlation and afterwards compared to hypotheses.

4.3.1 Hypothesis 1

Hypothesis 1 states that users using FaceDriver will achieve similar in-game performance compared to users playing with a keyboard. A two-tailed Paired Sample T-Test was conducted on the laptimes, with the first set of samples being the laptimes from when users played using a keyboard, and the second set of samples being the laptimes from when users played using the tracker. The test shows a significant difference ($p < 0.0005$) indicating that players playing with the trackers have significantly different laptimes ($M = 68.42, SD = 22.44, N = 25$) to players playing with a keyboard ($M = 34.18, SD = 5.88, N = 25$). This means the hypothesis can be rejected as users playing with a keyboard have a significantly improved in-game performance compared to when playing with FaceDriver.

Two main reasons can be determined as cause. First, users were given little time with FaceDriver, whereas users have extended experience using a keyboard. Given more time, users would likely improve and learn how to use FaceDriver better, and by doing so improve their in-game performance. Second, users who did not perform well in-game often caused FaceDriver to perform worse as well. Users who perform worse tend to move more (see hypothesis 1a), and therefore require FaceDriver to track faster movement. The more a user moves, the more likely it is for FaceDriver to lose track of the user, which in turn makes the in-game performance worse. Both of these points have been analysed for correlation in section 4.3.4.

Hypothesis 1a states that users who move less will have an improved in-game performance. There are several measures usable for movement, and all of them can be compared. Two separate movement totals can be calculated (as specified in section 4.2.2), one for the movement of the head, and one for the movement of the entire body.

First, the movement for sitting rounds will be analysed. A two-tailed Pearson's Correlation between the sitting laptime, sitting head movement, and sitting body movement can be performed based on data from the motion capture suit, as it provides data from the entire body. In order to verify the head and body movement behave in a similar manner, the correlation between these can be looked at first ($r = 0.87, p = 0.024, N = 6$). This indicates a strong correlation between head and body movement, as is to be expected. The correlation between sitting head movement and laptime ($r = 0.275, p = 0.6, N = 6$) shows both weak statistical significance, and a weak correlation, which means it can be regarded as an insignificant correlation. The correlation between sitting body movement and laptime ($r = -0.098, p = 0.854, N = 6$) also indicates only a weak correlation, and considering the low statistical significance this can also be regarded as no significant correlation.

Similarly, the correlation between the movement of standing users and laptimes can be analysed. Again, the correlation between head and body ($r = 0.977, p < 0.0005, N = 9$) is strong. The correlation between standing head movement and laptime ($r = 0.652, p = 0.057, N = 9$) is far greater than for sitting rounds, and shows a strong correlation. Between the standing body movement and laptimes ($r = 0.617, p = 0.077, N = 9$) a strong correlation is also visible.

This shows that standing users perform better if they move less, while sitting users do not necessarily perform better. For sitting users, this can be explained by inaccuracies in the measurements. The relatively little movements a sitting user makes (compared to standing users), can become insignificant compared to any 'noise' movement a user makes between laps, such as looking around. Relatively, this makes noise a far greater factor for sitting users, therefore reducing the accuracy of the analysis. Thus, it can be concluded that the hypothesis is valid and that users who make less movements do perform better in-game.

Hypothesis 1b states that users who have prior experience with body-controlled gaming will have an improved in-game performance. An Independent Samples T-Test was conducted on the laptimes obtained using the tracker while grouping users based on prior experience. This analysis has weak evidence ($p = 0.065$), and indicates that players who have prior experience with body-controlled gaming ($M = 64.05, SD = 16.26, N = 25$) perform better than those who do not ($M = 74.97, SD = 29.18, N = 25$). Due to weak evidence, no conclusive result can be obtained, but there seems to be a trend, meaning the hypothesis can be assumed to be valid.

4.3.2 Hypothesis 2

Hypothesis 2 states that users playing with FaceDriver will have an increased user engagement compared to users playing with the keyboard. In order to determine the GEQ score for each round, the mean value is taken from the answered questions (which are on a scale from 1 to 7). To determine which GEQ score is higher, a Paired Sample T-Test was conducted on the GEQ score from the keyboard, and the GEQ score from the tracker. See figure 7 for the distribution of GEQ scores when using the tracker. This analysis has insufficient statistical significance ($p = 0.859$) to be able to conclude anything. From the analysis can be observed that there is virtually no difference between GEQ scores from users playing with the keyboard ($M = 4.13, SD = 0.82, N = 25$) and users playing with the tracker ($M = 4.11, SD = 0.80, N = 25$). This might be due to the user's engagement being dependent on their in-game performance. Certain users performed exceptionally bad or well, which likely reflects on their user engagement. The analysis in section 4.3.4 shows that only a minor correlation exists between the in-game performance and the user engagement, if it exists at all.

Hypothesis 2a states that users who move more while using FaceDriver have an increased user engagement. Performing Pearson's correlation on the sitting user engagement and sitting body movement ($r = -0.359, p = 0.485, N = 6$) shows a correlation where increased movement leads to a decreased user engagement, but due to weak statistical significance it can't be taken as a conclusive result. For standing users ($r = 0.485, p = 0.186, N = 9$) a stronger correlation is visible, with a increased statistical significance, where increased movement does lead to an increased user engagement. Neither results have sufficient statistical significance to provide conclusive results.

Considering sitting users do not move their body much, due to being seated, no increase in user engagement is expected when they do. Standing users, however, have the ability to move their entire body. As shown in these results, this gives a strong correlation between body movement and user engagement, thereby validating the hypothesis.

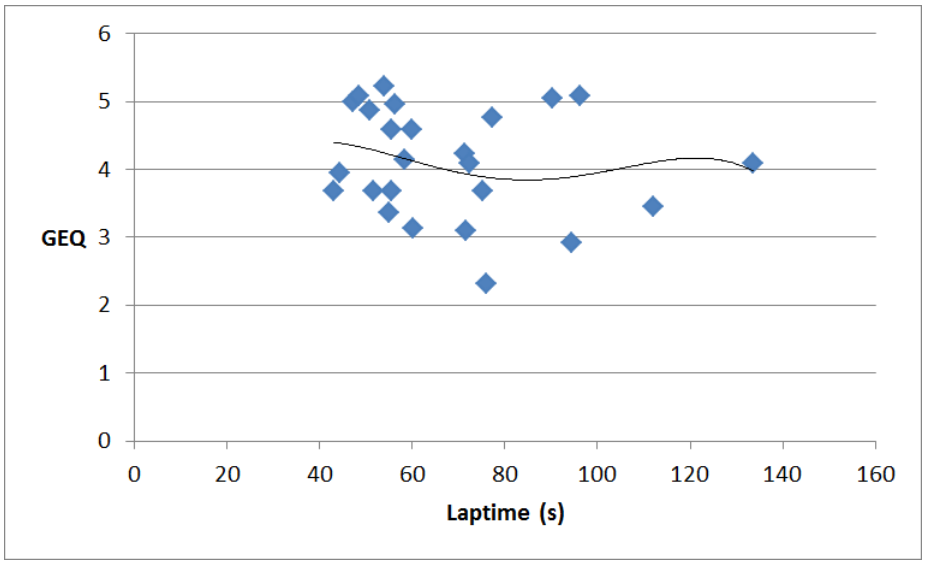

Figure 7: GEQ score for mean laptimes using the tracker (mean of both sitting and standing)

4.3.3 Hypothesis 3

Hypothesis 3 states that users prefer FaceDriver over traditional keyboard controls. The exit questionnaire contains a question asking the user which controls they enjoyed the most. From the sample set, 7 users preferred the keyboard, 3 users preferred FaceDriver while standing, and 15 users preferred FaceDriver while sitting. However, a total of 23 out of 25 users indicated that they would like to use motion-based controls more often. This shows that users have a strong preference for using FaceDriver over a keyboard, as it has a total of 18 picks, against 7 picks for the keyboard. Furthermore, nearly every user would like to use motion-based controls more often, which clearly indicates that these users are open to using controls other than a keyboard. With improvements, even more users might prefer FaceDriver over a keyboard.

Hypothesis 3a states that players who play games more often are more inclined to prefer keyboard controls. For the analysis, the amount of hours users play per week (on a 1-5 scale) have been grouped by whether or not users preferred a tracker-based controlling method. This has been analysed using a One-Way ANOVA test, and gives an insufficient statistical difference ($p = 0.582$). This means no conclusive result can be reached, and the time spent playing games per week might not affect whether or not the user prefers a tracker ($M = 2.89, SD = 1.41, N = 25$) or keyboard ($M = 2.57, SD = 0.79, N = 25$).

4.3.4 Correlation

Several possible correlations can be observed in sections 4.3.1, 4.3.2 and 4.3.3. In this section, those will be analysed.

In the analysis of hypothesis 1, it can be seen that users using the keyboard perform significantly better in-game than users using FaceDriver, and a learning effect is assumed to be a likely cause. In order to verify that a learning effect exists, this will be analysed. For every condition (keyboard, sitting, standing), the user gradually becomes more aware of the mechanics of the game, the track, and how the car behaves. While using a keyboard, the learning effect is likely a lot less than when using FaceDriver, as most users will already have a large amount of experience using a keyboard. In order to observe the learning effect, the first 7 laptimes will be compared with the last 7 laptimes. This will show the overall improvement users make during the course of the experiment. Then, the first 5 laps a user has driven using FaceDriver will be compared to the last 5 tracks a user has driven with FaceDriver. This will show the improvement a user makes using FaceDriver only.

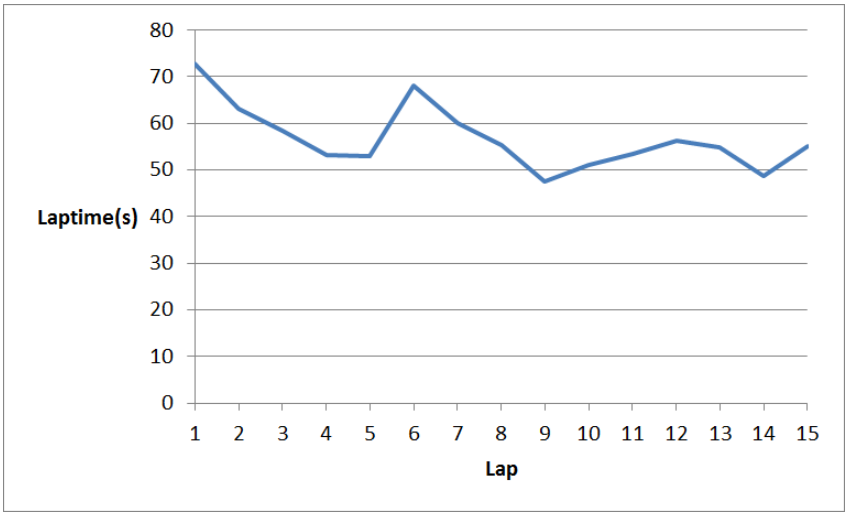

Figure 8: Mean laptimes per lap

As can be observed when looking at figure 8, the laptimes decrease on average. After each 5 rounds, they stop decreasing and instead increase. This happens when users switch to another condition, such as going from using the keyboard to using the tracker, for example. In figure 9 the laptimes for each individual condition are shown. This means there is a learning effect going on for each individual condition, and also for the experiment as a whole, considering the slowest lap of all is at the start. Noticeable is that the last lap performs sig-

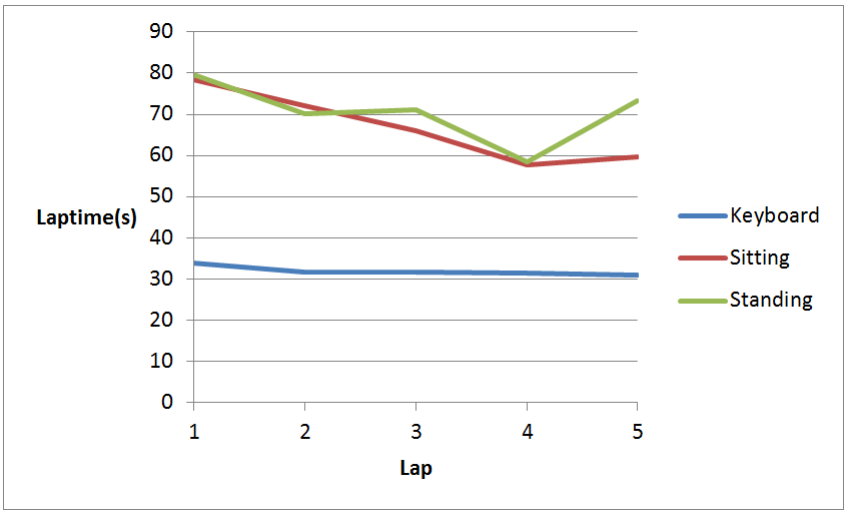

Figure 9: Mean laptimes per condition

nificantly worse than the prior one, this might be due to a lack of effort at the end. Further, a linear decrease can be observed in the laptimes for the tracker (excluding the last lap) which suggests that users continued to learn how to use the tracker at a steady pace. In order to verify these effects, an analysis needs to be made.

First the overall learning effect will be analysed. Performing a Paired Samples T-Test on the first and last 7 laptimes, the first half ($M = 61.25, SD = 20.82, N = 25$) has a marginally higher laptime than the second half ($M = 52.46, SD = 17.71, N = 25$), with weak statistical significance ($p = 0.061$). Secondly, we will be comparing the first and last 5 laptimes within FaceTracker, and expect to see a stronger learning effect. The first half ($M = 75.94, SD = 29.49, N = 25$) compared to the second half ($M = 60.90, SD = 19.90, N = 25$) shows a significant ($p = 0.003$) learning effect, and is much stronger than the overall learning effect. From this, we can conclude that users still have a lot of room left to improve, and the average FaceDriver laptimes used for the analysis of hypothesis 1 would improve if users were given more time with the tracker, and might eventually allow users to get laptimes equal to those of the keyboard.

In the analysis, the movement of the user has been shown to have a correlation to the user's laptimes. The more a user moves, the worse the laptimes become. In order to find the reason behind this, the recorded data from FaceDriver itself can be analysed. FaceDriver recorded events such as losing track of the user, making it possible to see how well the tracker behaved during a user's round. This data will be converted to 'losses per minute' as an indicator of how often the tracker lost track of the user. Analysing the correlation between the sitting

laptimes and the losses per minute while sitting ($r = 0.25, N = 25$) shows that no strong correlation can be found, partly due to low statistical significance ($p = 0.219$). The same analysis for standing rounds ($r = 0.493, N = 25$) does show a significant ($p = 0.012$) correlation between the amount of times the tracked face has been lost, and the laptime. This correlation gives an explanation as to why users who move more (and therefore have a higher chance of making the tracker lose track) achieve worse in-game performance.

In the analysis for hypothesis 2, the user engagement appears to not have any strong correlation to how much the user was moving. In order to find factors that do influence the user engagement, the correlation between the user engagement and laptimes can be analysed. Analysis shows that standing rounds ($r = -0.303, p = 0.141, N = 25$) show some correlation, but it is not statistically significant enough to draw conclusions from. Sitting rounds ($r = -0.092, p = 0.662, N = 25$) show no correlation. This means the user engagement might be marginally affected by the laptimes, but the correlation is not significant enough to explain why hypothesis 2 is not valid.

4.3.5 User feedback

Part of the questionnaire allows users to give their suggestions for FaceDriver. In this section, the suggestions will be briefly discussed. The general feedback was positive, and most of the suggestions were related to technical criticism.

Several users did not realize that there are multiple ways to steer left and right, and that the tracker behaved best if the roll of the head was adjusted, rather than the yaw. Prior instructions in this aspect did not seem to affect how users turned their heads, as users simply seemed to prefer adjusting either the yaw or the roll.

A technical issue in the game would on occasion cause the car to take a sharp turn to the right, immediately after starting. Restarting the game entirely temporarily fixed this issue. As soon as the user begins to steer, the problem goes away for the remainder of the lap. Some users noticed this and made a remark on it as feedback.

Calibration is something several users missed, as they felt their neutral pose did not reflect the neutral pose in the game. Automatic and non-invasive calibration is something that could be added as a future improvement, as it would improve the user experience.

The sensitivity was set very high in FaceDriver, meaning that users would only have to make very minor movements with their head in order to move the car around. This was done to improve the reliability of the tracker, as it would encourage users to make smaller movements, and therefore make it less likely for the tracker to lose track of the user. Several users found the sensitivity too

high, and would have liked to have it decreased. While FaceDriver is capable of adjusting the sensitivity, the sensitivity remained constant throughout the entire experiment, to ensure the tracker would behave in the same way for every user.

5 Conclusion

This work presented an interface called 'FaceDriver' to control games using the motion of a user's head. Using FaceDriver, a user study was conducted on 25 users. In this user study, users had to play the racing game 'TrackMania Forever' using the keyboard, and using FaceDriver.

FaceDriver is software which can run on any computer that has access to a camera. When started, it begins tracking heads in front of the camera, and translates the movement of their head into input games can use. This means that a large amount of games can be played using FaceDriver without requiring any changes. The FaceDriver software is configurable, and can be adjusted to cope with various control-schemes required by games. Other methods exist that can track user motion and make it usable for games, such as the Kinect. FaceDriver can be used without any additional hardware, requiring only a camera, and does not require specially-engineered games.

A user study was conducted on 25 users in a controlled environment. These users had to race 5 laps for each condition. These conditions were: playing using a keyboard, playing using FaceDriver while seated, and playing using FaceDriver while standing. While racing, their laptimes were recorded. A small amount of users also wore a motion capture suit, allowing their full body movement to be recorded while playing. The players were given a questionnaire before starting the experiment, after the experiment, and after completing the laps for each condition.

Evaluation showed that users did not perform as well in-game with FaceDriver as they did with a keyboard. Likely causes include lack of experience using FaceDriver, and technical shortcomings. Analysis showed that users using FaceDriver performed significantly better towards the end than they did at the beginning. Users showed more improvement using FaceDriver than they did with the keyboard, which indicates users were still in the process of learning how to use FaceDriver. In the long term, users might achieve equal or better in-game performance with FaceDriver compared to the keyboard. Technical issues caused, on occasion, decreased in-game performance. FaceDriver did not perform equally well on every user, and was sensitive to problems for users who had parts of their face covered (facial hair, glasses, scarf).

Between each round, when changing between conditions, the user engagement

was evaluated. For this, the players were given a questionnaire (see appendix C). Measuring the user engagement showed a correlation between the movement of the players, and FaceDriver. Specifically, users who were standing while playing using FaceDriver had a higher user engagement than playing while using the keyboard or playing while seated using FaceDriver.

Based on questionnaire results, users preferred FaceDriver over the keyboard, and 23 out of 25 users chose motion-based controls in general over non-motion-based controls. This means motion-based gaming has a lot of potential, as many users are interested in using motion-based controls.

In conclusion, FaceDriver provides a usable alternative interface to play games with. While having some technical issues, users still prefer FaceDriver over the traditional keyboard, and enjoy playing with it.

6 Future work

Several improvements could be made to the existing software. Firstly, the technical issues users were experiencing limited the in-game performance for a few users. Making FaceDriver more robust in terms of head tracking would allow for a more enjoyable experience for those users. For example, improving recognition on users with their faces partially covered, and improving recognition and tracker speed in non-optimal lighting conditions. Secondly, several improvements could be made to the software to make it more user-friendly, such as adding calibration in some form, allowing users to set a sensitivity they prefer. Another feature that was ultimately not used due to technical issues is the ability to make use of Facial Action Coding System (FACS) data in order to trigger button-presses in the game. The user study consisted only of using the rotation of the head. Further, if FACS is used to get facial info, it should be possible to determine if the user is expressing any specific emotion. For example, a frustrated face could be interpreted by the game to change the difficulty.

A number of users indicated interest in seeing this system employed while playing games with a group of people. In order to make this work, the tracker would need to be able to recognize multiple individual faces, identify them correctly, and persistently keep track of them. This would allow for research on groups of people playing with this software.

Currently, the software is limited to very simple games, partly due to the lack of distinct inputs when using only the rotation of the head. Another reason is that more complex games often require the user to look around the screen more, which might conflict with the head tracking. Research could be conducted to let users play different types of games and see how much users move their heads naturally while playing. Some games have inventories which require the player to inspect each slot, or games that have several pieces of HUD on different

areas of the screen which require constant attention. This data could be used to determine the optimal use case for interfaces such as FaceDriver.

References

[1] Fakhreddine Karray et al. *Human-Computer Interaction: Overview on State of the Art*. 2008.

[2] Jamie Shotton et al. "Real-time human pose recognition in parts from single depth images". In: *Communications of the ACM* 56.1 (2013), pp. 116–124.

[3] Cha Zhang and Zhengyou Zhang. *A survey of recent advances in face detection*. Tech. rep. Tech. rep., Microsoft Research, 2010.

[4] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–511.

[5] Stan Z Li et al. "Statistical learning of multi-view face detection". In: *Computer VisionECCV 2002*. Springer, 2002, pp. 67–81.

[6] Edgar Seemann, Bastian Leibe, and Bernt Schiele. "Multi-aspect detection of articulated objects". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 2. IEEE. 2006, pp. 1582–1588.

[7] Ekman P. and Friesen W. V. *Manual for the facial action coding system. Consulting*. Psychologists Press, 1978.

[8] Paul Ekman and Erika L Rosenberg. *What the face reveals: Basic and applied studies of spontaneous expression using the Facial Action Coding System (FACS)*. Oxford University Press, 1997.

[9] Robert W Levenson, Paul Ekman, and Wallace V Friesen. "Voluntary facial action generates emotion-specific autonomic nervous system activity". In: *Psychophysiology* 27.4 (1990), pp. 363–384.

[10] James J Lien et al. "Automated facial expression recognition based on FACS action units". In: *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*. IEEE. 1998, pp. 390–395.

[11] Akshay Asthana et al. "Incremental face alignment in the wild". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE. 2014, pp. 1859–1866.

[12] Jason M Saragih, Simon Lucey, and Jeffrey F Cohn. "Face alignment through subspace constrained mean-shifts". In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 1034–1041.

[13] Xiaoming Liu. "Generic face alignment using boosted appearance model". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[14] Iain Matthews and Simon Baker. "Active appearance models revisited". In: *International Journal of Computer Vision* 60.2 (2004), pp. 135–164.

[15] Minh Hoai Nguyen and Fernando De La Torre. "Local minima free parameterized appearance models". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[16] Shaohua Kevin Zhou and Dorin Comaniciu. "Shape regression machine". In: *Information Processing in Medical Imaging*. Springer. 2007, pp. 13–25.

[17] Nadia Bianchi-Berthouze, Whan Woong Kim, and Darshak Patel. "Does body movement engage you more in digital game play? And Why?" In: *Affective Computing and Intelligent Interaction*. Springer, 2007, pp. 102–113.

[18] Jeanne H Brockmyer et al. "The development of the Game Engagement Questionnaire: A measure of engagement in video game-playing". In: *Journal of Experimental Social Psychology* 45.4 (2009), pp. 624–634.

[19] Daniel Wigdor and Dennis Wixon. *Brave NUI world: designing natural user interfaces for touch and gesture*. Elsevier, 2011.

[20] Maged N Kamel Boulos et al. "Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation". In: *International journal of health geographics* 10.1 (2011), p. 45.

[21] Marco Roccetti, Gustavo Marfia, and Angelo Semeraro. "Playing into the wild: A gesture-based interface for gaming in public spaces". In: *Journal of Visual Communication and Image Representation* 23.3 (2012), pp. 426–440.

[22] Donald A Norman. "Natural user interfaces are not natural". In: *interactions* 17.3 (2010), pp. 6–10.

[23] G Robert Grice, Robert Nullmeyer, and V Alan Spiker. "Human reaction time: Toward a general theory." In: *Journal of Experimental Psychology: General* 111.1 (1982), p. 135.

[24] Hidemi Shimizu. "Measuring keyboard response delays by comparing keyboard and joystick inputs". In: *Behavior Research Methods, Instruments, & Computers* 34.2 (2002), pp. 250–256.

[25] Marco Pasch et al. "Movement-based sports video games: Investigating motivation and gaming experience". In: *Entertainment Computing* 1.2 (2009), pp. 49–61.

[26] Rensis Likert. "A technique for the measurement of attitudes." In: *Archives of psychology* (1932).

Appendices

A Intro Questionnaire

1. What is your age?
2. What is your gender?
3. How many hours do you play games per week?
4. Do you have any experience with body-controlled games? (such as the Kinect, not the Wii)
5. Do you have any experience with racing games?
6. Do you have any experience with Trackmania Forever specifically?

B Exit Questionnaire

1. The controls I enjoyed the most are:
   (a) Keyboard
   (b) Motion controlled, standing
   (c) Motion controlled, sitting
2. Would you like to use motion controls more often?
3. Suggestions:

C Post-round questionnaire

C.1 Modified Game Engagement Questionnaire

1. I lose track of time.
2. The game feels real.
3. I get wound up.
4. Playing seems automatic.
5. I play without thinking how to play.
6. I really get into the game.
7. I feel like I just can't stop playing.
8. I could play for a long time.

9. I felt tired after playing.
10. Controlling the game feels natural.
11. I enjoyed playing with these controls.

C.2 Other

1. The game is challenging.
2. I feel like I am in control of the vehicle.
3. I got used to the controls quickly.