MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

REACTIVE PROGRAMMING BASED GESTURE DETECTION IN VIRTUAL REALITY USING LEAPMOTION

LICENSE THESIS

Graduate: Radu PETRIŞEL
Supervisor: Assist. Prof. Dr. Eng. Adrian SABOU

2019

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Radu PETRIŞEL**

REACTIVE PROGRAMMING BASED GESTURE DETECTION IN VIRTUAL REALITY USING LEAPMOTION

1. **Project proposal:** *A Reactive Programming oriented Unity asset for gesture detection using the LeapMotion controller*
2. **Project contents:** *(enumerate the main component parts) Presentation page, advisor's evaluation, title of chapter 1, title of chapter 2, ..., title of chapter n, bibliography, appendices.*
3. **Place of documentation:** *Technical University of Cluj-Napoca, Computer Science Department*
4. **Consultants:** Assist. Prof. Dr. Eng. Adrian SABOU
5. **Date of issue of the proposal:** November 1, 2018
6. **Date of delivery:** June 14, 2019

Graduate: ______________________________

Supervisor: ______________________________

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) **RADU PETRIŞEL** legitimat(ă) cu **C.I.** seria **CJ** nr. **315937** CNP **1960920125844**, autorul lucrării **REACTIVE PROGRAMMING BASED GESTURE DETECTION IN VIRTUAL REALITY USING LEAPMOTION** elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea **CALCULATOARE, ENGLEZA** din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea **IULIE** a anului universitar **2018-2019**, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă.*

Data   Nume, Prenume

______________   **RADU PETRIŞEL**

Semnătura

Contents

Chapter 1

Introduction - Project Context

Virtual Reality is an experience that has gained huge popularity in the recent years. Because of this new means of interaction with this virtual world are needed and they should feel as natural as possible. Ergo, hand tracking and gesture detection is a "must have" for modern VR applications.

1.1 Virtual reality

The term "virtual" began its life in the late 1400s, meaning "being something in essence or effect, though not actually or in fact" [1], but, in the IT context, the word has the meaning "not physically existing but made to appear by software" [1]. The original use of the phrase "virtual reality" is found in French playwright' Antonin Artaud collection of essays *Le Théâtre et son double*, first published in 1938 [2].

1.1.1 History

The precise roots of virtual reality are challenged, partially because of how hard it was to formulate a definition of an alternate reality notion. In 1968, Ivan Sutherland created what was widely regarded as the first head-mounted display system for use in immersive simulation applications, with the help of his students. In the next two decades, VR devices were mainly used for medical, automobile industry design, servility training and flight simulation purposes.

The 1990s saw the first commercially extensive release of consumer headsets, notably *Sega VR* (1991) and *Sega VR-1* (1994) launched by Sega, and *Nintendo's Virtual Boy* (1995). The 2000s were a period of comparative indifference from the public and investment towards VR techniques available on the market. Google launched *Street View* in 2007, a service that offers panoramic views of a growing amount of global locations such as highways, indoor houses and rural regions, which also integrates a stereoscopic 3D mode as of 2010.

The modern, consumer version of headsets started developing in the early 2010s. In 2013, Valve Corporation found and freely shared the breakthrough of low-persistence screens that make it possible today to show VR content lag-free and smear-free.

This discovery was quickly adopted by the other companies on the market, with Sony announcing *Project Morpheus* in 2014 and Google announcing *Cardboard* in 2015. In 2016, HTC released and shipped the first units of *Vive SteamVR*, the first major commercial headset for average users.

Figure 1.1: Project Morpheus (PlayStation VR) at gamescom in 2015

1.1.2 Modern technology

Present virtual reality headset displays rely on smartphone technologies including: gyroscopes and motion sensors for head, hand and body position monitoring, tiny high definition stereoscopic displays and small, lightweight and powerful computer processors.

Special input devices are required for interaction with the virtual world, such as hand controllers, haptic gloves, 3D mouse and optical tracking sensors. Both haptic gloves and hand controllers provide force feedback (in the form of vibration), with haptic gloves providing also feedback in the form of response force (like when picking a rubber duck).

1.2 Gesture recognition

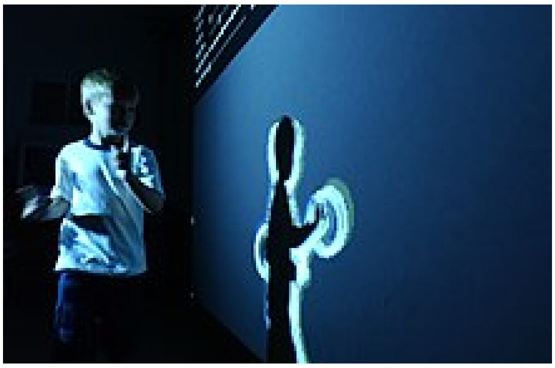


Figure 1.2: A child being recognized by a simple gesture detection algorithm

Gesture recognition is an active research field with the objective of comprehending human gestures through mathematical models. Gestures can come from any posture or position of the body, but they typically come from the hand. Without actually touching them, users can use simple motions to command or communicate with machines.

Gesture recognition may be seen as a means for machines to commence to comprehend human body language, establishing a stronger link between computers and individuals than conventional text user interfaces or even GUIs (graphical user interfaces), which still restrict most inputs to the keyboard and/or mouse and communicate naturally with no mechanical instruments.

Chapter 2

Project Objectives and Specifications

2.1 Introduction

The purpose of this chapter is to collect, analyze and define high-level needs and features of the Unity asset named Fluent Motion. It focuses on the capabilities needed by the stakeholders and the target users, and why these needs exist.

2.2 Positioning

2.2.1 Problem statement

As Virtual Reality (VR) is becoming more accessible to the average person, more problems arise with the means of interacting with the VR world. A solution to this issue is the Leap Motion hand tracking device, which offers a natural means of human-VR interaction. The problem with Leap Motion is its non-friendly Application Programmer Interface (API).

The problem of	Leap Motion's unfriendly API
affects	developers in the VR field who use Leap Motion
the impact of which is	a limited number of applications using Leap Motion
a successful solution would be	easy to use fluent (in terms of code readability) adhere to the reactive programming paradigm available on Unity's asset store

2.2.2 Product Position Statement

Fluent Motion comes as a union between three technologies – Virtual Reality, Leap Motion and ReactiveX.

So far, Virtual Reality and Leap Motion already are integrated (by means of LeapMotion's API), but ReactiveX can offer a more fluent way of expressing what an application using the first two mentioned technologies together.

For	Virtual Reality developers
who	use Leap Motion
Fluent Motion	is an extension of Leap Motion using ReactiveX
that	offers a fluent API for Leap Motion
unlike	the default API
Fluent Motion will	be easy to use be fluent (in terms of code readability) adhere to the reactive programming paradigm

2.3 Stakeholder and User Descriptions

2.3.1 Stakeholder summary

Name	Description	Responsibilities
Developer (VR)	Person who wants to create Virtual Reality applications	Use Fluent Motion
Developer (Fluent Motion)	Person who creates and maintains Fluent Motion	Create, improve and offer technical support for Fluent Motion

2.3.2 User summary

Name	Description	Responsibilities	Stakeholder
Developer (VR)	Person who wants to create VR applications	Use Fluent Motion	*Developer (VR)*

2.3.3 User environment

2.3.3.1 Users

The API will be used by developer teams of any size.

2.3.3.2 Infrastructure

The infrastructure needed by Fluent Leap is an aggregation of the hardware requirements of the combined systems and technologies, i.e.:

Operating system	**Windows 7 SP1**, **Windows 8.1** or later, **Windows 10**
Middleware	**SteamVR** platform
Additional hardware	**Leap Motion** hand tracking device a VR headset (at the time of writing, **Occulust Rift**, **HTC Vive** or **Valve Index**)
Miscellaneous	**.NET Framework 4.6** or newer **Unity** 5.6 or later

2.3.4 Summary of key stakeholder or user needs

Need	Priority	Concerns	Current solution	Proposed solution
VR API	0	Developer	Leap Motion default API, using **ANSI C** language imperative style	Fluent Motion, using **C#** language and ReactiveX
Desktop API	1	Developer	Leap Motion default API, using **ANSI C** language impertive style	Fluent Motion, using **C#** language and ReactiveX
Usability	0	Developer	Leap Motion defualt API, using **ANSI C** language imperative style	Fluent Motion, using **C#** language and ReactiveX

2.3.5 Alternatives and competetion

External competition is represented by the current API offered by LeapMotion.

2.4 Product overview

The API should provide all the functionality already provided by Leap Motion's default API, but in a higher-level language.

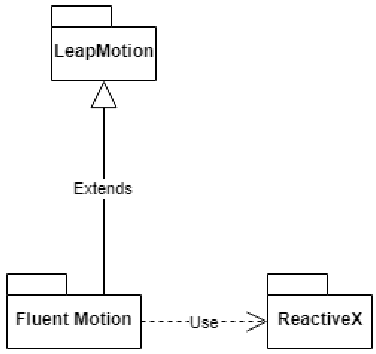


Figure 2.1: Fluent Motion architectural diagram

2.4.1 Product perspective

This product will extend existing features from Leap Motion, making them more readable and developer friendly.

2.4.2 Assumption and dependencies

For developers:

Operating system	**Windows 7 SP1**, **Windows 8.1** or later, **Windows 10**
Middleware	**SteamVR** platform
Additional hardware	**Leap Motion** hand tracking device a VR headset (at the time of writing, **Occulust Rift**, **HTC Vive** or **Valve Index**)
Miscellaneous	**.NET Framework 4.6** or newer **Unity** 5.6 or later

For end products that reference Fluent Motion:

	Minimum	Recommended
CPU	Intel Core i3-8100	Intel i5-4590 or AMD FX 8350 equivalent
GPU	Nvidia GeForce GTX 1060 3GB or AMD Radeon RX 570	Nvidia GeForce GTX 970 or AMD Radeon R9 290 equivalent
Memory	8GB	16GB
Output	HDMI 1.4, DisplayPort 1.2	DisplayPort 1.2
Input	2x USB 3.1 gen 1 (Type-A)	2x USB 3.1 gen 2 (Type-A)

2.5 Product features

1. **Hands module**

   The module that will allow the developer to use the two hands objects from the scene in order to detect gestures or motions, and assign callbacks when certain criteria regarding the hands are met.
   This module is available in both Virtual Reality and Desktop modes.

2. **Fingers module**

   The module that will allow the developer to use the fingers (individually or in groups) in the scene for detecting gestures or motions, and assign callbacks when certain criteria regarding the fingers are met.
   This module is available in both Virtual Reality and Desktop modes.

3. **Virtual Reality module**

   The module that will allow the user to develop Virtual Reality applications. This module will act as a dependency to other modules.
   It isn't mutually exclusive with the Desktop Module, BUT at least one of the two must be present.

4. **Desktop module**

   The module that will allow the user to develop applications using Leap Motion in desktop mode.
   This module will act as dependency to other modules. It isn't mutually exclusive with the Virtual Reality Module, BUT at least one of the two must be present.

5. **Gesture definition module**

   The module that allows the user to define new gestures besides already existing ones. This module depends on either Virtual Reality module or Desktop module, and on either Hands module, Fingers module or both.

6. **Gestures module**

   This module will provide defitions to some basic gestures (like finger pointing to some object, hand swipe, etc.). This module depends on either Virtual Reality module or Desktop module, and on either Hands module and Fingers module.

7. **Demo**

   This module will provide some demonstrative code for new users to acquaint themselves with the basic flows and code syntax of Fluent Motion.

2.6 Other product requirements

1. **High readability**

   The main purpose of the API is to be fluently read, i.e. the code should sound almost like natural language when read by other developers.

2. **Open source**

   The project will be open source and anyone will be able to contribute to it.

3. **Performance**

   Virtual Reality applications shouldn't fall bellow 80 frames per second (FPS), and, as such, the API shouldn't introduce a high processing time per frame to to fall below that threshold.

4. **Scalability**

   The API should support detecting up to 10 distinct gestures per application and interacting with at least 20 objects (excluding hand to hand interactions).

5. **Maintainability**

   The VR world is still young and technologies evolve fast, so the API should be highly maintainable to keep its edge.

6. **Extendibility**

   The API should be easily extendable by any backer on Git.

Chapter 3

Bibliographic research

3.1 Virtual reality

Virtual Reality (VR) is the usage of computer technology to produce a world that is simulated. Unlike conventional user interfaces, VR positions the user in an environment in which they are immersed and able to interact with 3D worlds rather than watching a screen in front of them. The computer is converted into a arbiter for this alternative reality by simulating as many senses as applicable, such as sight, hearing, touch or even scent.

The head-mounted display (HMD) is by far the most instantly recognizable element of Virtual Reality. Human creatures have always been visual animals, and display technology is often the greatest gap between immersive systems of Virtual Reality and conventional GUIs.

The future of wearable devices is unraveling but still uncertain with a multitude of evolving wired (tethered) and wireless (untethered) options. Concepts like the HTC Vive Pro Eye, Oculus Quest and Playstation VR are leading the way, but competitors such as Google, Apple, Samsung and others mighty also shock the sector with fresh amounts of immersion and usability.

Virtual Reality has a variety of uses in a multitude of domains, such as:

1. **Education and training**
   VR is used to provide a virtual world for trainees to improve their abilities without the risk of failure in the real world. Applications such as flight simulators, surgery training and spacewalk training have been used for decades in strongly technologised countries.

2. **Engineering and robotics**
   The use of 3D computer-aided design (CAD) data was limited by 2D monitors and paper printouts until the mid-to-late 1990s, when video projectors, 3D tracking, and computer technology enabled a renaissance in the use of 3D CAD data in virtual reality environments. Innovative VR engineering systems allow engineers to visualize virtual prototypes before any physical models are available.

3. **Entertainment**
During the early to mid 1990s, several vanilla commercial virtual reality headsets have been released for gaming, such as the *Virtual Boy* developed by Nintendo or *VFX1 Headgear* developed by Forte Technologies. Films produced for VR permit the audience to view a 360-degree environment. This can involve the use of VR cameras to produce films and series that are interactive in VR. VR may help people attend concerts without effectively being there.

4. **Healthcare and clinical therapies**
A 2017 report by Goldman Sachs investigated healthcare applications of VR and AR. [3]. Some companies are adapting VR for fitness by using gamification concepts to encourage exercise. Since the 2000s, virtual reality has been used for rehabilitation. Despite countless research, there is a lack of excellent quality proof of its effectiveness compared to other techniques of rehabilitation without advanced and costly facilities for Parkinson's disease therapy. [4]

5. **Heritage and archaeology**
Virtual reality allows highly accurate recreation of historic locations so that artistic renderings can be published in different media. The initial sites are very often unavailable to the public or difficult to portray due mainly to the bad condition of their conservation. Using this technology, virtual replicas of grottoes, environment, ancient cities and monuments, sculptures and archaeological components can be developed.

6. **Occupational safety**
VR simulates real workplaces for occupational safety and health purposes. Perspective, viewing angle, and acoustic and tactile characteristics alter depending on where the individual is standing and how they move relative to the setting. VR enables all phases of a product life cycle, from design, through use, up to disposal, to be simulated, analyzed and optimized.

All of the above mentioned domains are part of the day-to-day life of some people already. For them, interacting with the virtual reality they are put in should feel free and natural. In a real life situation, they would use their hands to perform the actions they are performing in virtual reality during those ”training” periods. So the question arises: how do they perform their everyday tasks in virtual reality?

If somebody were to ask that question to the two big players in the VR game (namely, HTC and Occulus), the answer will be the same from them all - handheld controllers (depicted in figure 3.1). They are very versatile and offer various means of interaction, but they lack one simple feature - feeling natural. Take, for instance, a surgeon training in VR to perform a new kind of surgery that he hadn't performed before. By using the controllers, he has a handicap that would not be present in a real life situation. As such, his training does not quite copy the real thing.

So, how would one improve the issue of VR interaction? As of now, there are two possibilities - *haptic gloves* or *hand-tracking devices*. Haptic gloves are special devices that users put on their hands, which track their hands' motions and send them to the computer to process. Haptic gloves offer the advantage of sending back haptic feedback (pressure and force). Hand tracking devices use cameras and complex algorithms to track the users' hands. The advantage of hand-tracking devices is that the hands can move unrestricted and without needing additional hardware. Both hand tracking devices and haptic gloves have a common denominator - **gestures**.

(a) HTC Vive controllers

(b) Oculus Rift controllers

Figure 3.1: The two types of VR controllers available on the market

3.2 Gestures

A gesture is a type of non-verbal communication in which real physical movements transmit specific messages, both in place or in combination with speech. Gestures include movement of the hands, face, or other parts of the body. Gestures differ from physical non-verbal communication that does not communicate specific messages, such as purely expressive displays, proxemics, or displays of joint attention. [5]

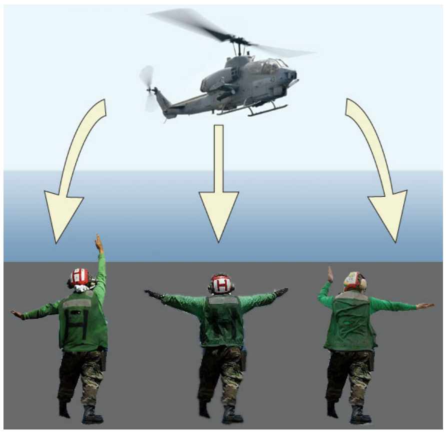


Figure 3.2: Military air marshallers use hand and body gestures to direct flight operations aboard aircraft carriers

Gestures can be of two types: *informative (passive)* or *communicative (active)*. *Informative gestures* are passive gestures that provide details about the speaker as a person and not about what the speaker is attempting to communicate, while *communicative gestures* are deliberately and meaningfully made by a individual as a means of height-

ening or altering speech generated in the larynx (or
with hands in the situation of sign languages) although he or she may not be actively conscious of the fact that they produce communicative gestures.

Within the realm of communicative gestures, the first distinction to be made is between gestures made with the hands and arms, and gestures made with other parts of the body, such as head or shoulders. From now on, we shall focus only on manual gestures.

3.2.1 Manual gestures

Manual gestures are split into four categories:

1. **Symbolic (emblematic)**
   These are standard, culture-specific gestures which can be used as a substitute for words (like waving your hand to say "hello" or "googbye"). In distinct cultural contexts, a single emblematic gesture can have a very distinct meaning, varying from complimentary to extremely offensive. Symbolic gestures are iconic gestures that are widely recognized, fixed, and have conventionalized meanings. [6]

2. **Deictic (indexical)**
   Deictic gestures may happen concurrently or in place of vocal expression. Deictic gestures are gestures consisting of indicative motions or pointing movements. They often replace words and pronouns like "this", "there" or "that".

3. **Motor (beat)**
   In verbal speech, motor or beat gestures typically consist of brief, rhythmic, repetitive motions strongly linked to sentence construction. Beat gestures do not happen separately of verbal expression, unlike symbolic and deictic gestures. Some individuals wave their hands, for instance, as they talk to highlight a word or sentence. These gestures are closely coordinated with speech.

4. **Lexical (iconic)**
   Other spontaneous gestures used in speech known as iconic gestures are more content-filled, and the significance of the co-occurring voice may echo, or be elaborated. They portray elements of pictures, behavior, individuals, or items in space. For instance, a gesture depicting the throwing act may be synchronous with the saying, "He threw the ball right into the window.".

3.3 Gesture recognition

Gesture recognition is an active research field with the objective of comprehending human gestures through mathematical models. Gestures can come from any posture or position of the body, but they typically come from the hands. Without actually touching them, users can use simple motions to command or communicate with machines.

Gesture recognition may be seen as a means for machines to commence to comprehend human body language, establishing a stronger link between computers and individuals than conventional text user interfaces or even GUIs (graphical user interfaces), which still restrict most inputs to the keyboard and/or mouse and communicate naturally with no mechanical instruments.

There are two gesture types in the human-computer interaction context:

1. **Offline gestures**
   These gestures are processed after the object is interacted with by the user (e.g. activate a menu gesture).

2. **Online gesture**
   Direct manipulation gestures, like scaling or moving an object.

3.3.1 Algorithms

The strategy to translating a gesture could be performed in distinct ways based on the category of input data. Most of the methods, however, are based on important pointers in a 3D reference system. The gesture can be identified with considerable precision depending on the relative movement of all these, depending on the quality of both the input and the strategy of the algorithm.

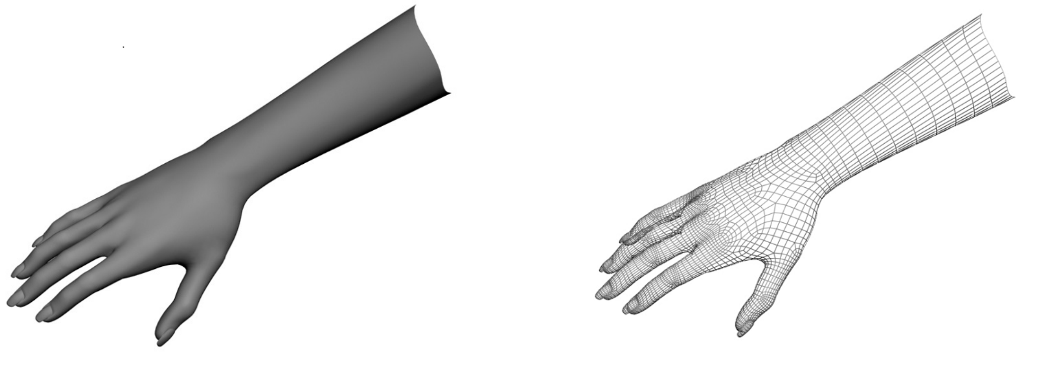
Gesture detectiona algorithms can be split in three major categories, based on the hand models they are using:

1. **3D model-based algorithms**
   The 3D model approach can use volumetric or skeletal models, or even a combination of the two. Volumetric methods were used extensively in the field of computer animation and computer vision. The advantage of this strategy is the simplicity of the parameters for all these objects. This technique's disadvantage is that it is very computational-intensive, and technologies still need to be developed for real-time interpretation.

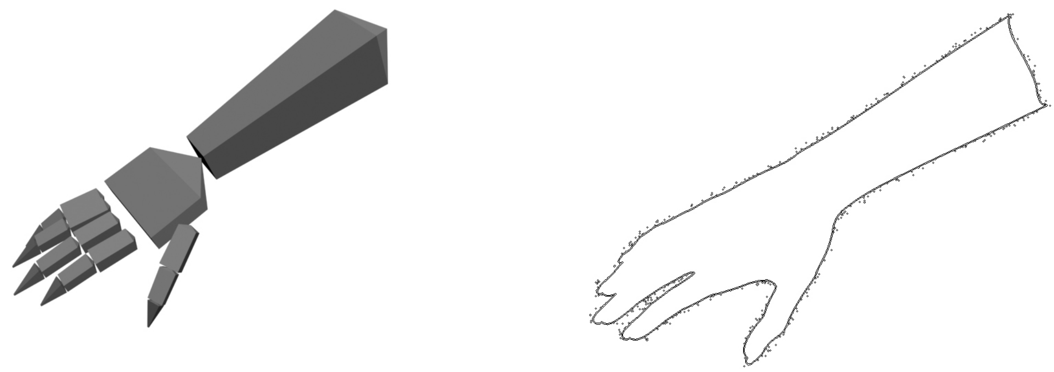
2. **Skeletal-based algorithms**
   Instead of using intensive 3D model processing and working with several parameters, one could just make use of a simplified version of the parameters, like the joint angle along with the dimensions of the section. The advantage of this method is that the algorithms are faster (thus requiring less resources) and the templated patterns can be persisted in a database and used for later matching. The drawback is that, because of the reduced parameters, the algorithms do not perform as acurately as 3D model-based algorithms.

3. **Appearance-based models**
   These algorithms no longer use a body's 3D representation since they obtain the

parameters from the videos and images directly using a template database. One of the approaches of gesture detection using appearance-based models uses image sequences as gesture templates (either using the images directly or features extracted from the images).



(a) Original hand (b) 3D representation



(c) Skeletal representation (d) 2D representation (appearance)

Figure 3.3: The different digital representations of a hand

3.3.2 Input devices

While there is a big quantity of studies conducted in gesture recognition focusing on image/video, there is a certain variation among applications within the instruments and implementations used.

1. **Wired gloves**
   Using magnetic or inertial monitoring devices, wired gloves can provide information to the computer about the hands' position and rotation. In addition, some gloves can sense the curling of fingers with a high degree of precision (5-10 degrees) or even provide the user with haptic feedback, which is a model of touch sensation.

2. **Depth-aware cameras**
   Using dedicated cameras like structured light or time-of-flight cameras, one may create a depth map of what one sees through the camera in a limited range and use this information to approximate a 3D model of what one sees. Due to their limited range capacities, these can be efficient in detecting hand gestures.

3. **Stereo cameras**
   Using two cameras whose positions are known to each other, the cameras' output can calculate an approximate 3D representation. A positioning reference such as a lexian-stripe or an infrared emitter can be used to get the distance between the cameras.

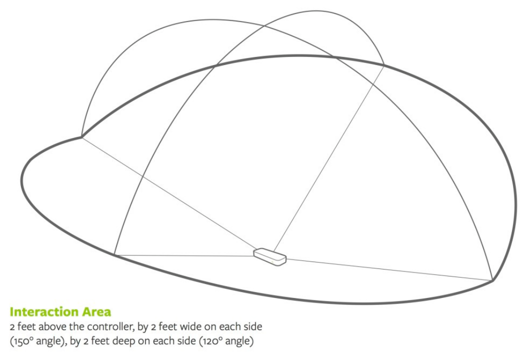
4. **Gesture based controllers**
   These controllers function as a an extension of the body so that most of their movement can be easily recorded by software while gestures are made. An instance of developing gesture-based movement recording is skeletal hand tracking, created for apps of virtual reality and augmented reality.

One of the more popular and accessible gesture detection devices is the Leap Motion controller, which is presented in the following section.

3.4 Leap Motion

The *Leap Motion Controller* is a tiny peripheral USB device intended to be facing upwards on a physical desktop, but can also be mounted on a VR headset.

(a) The LeapMotion controller



(b) The interaction area

Figure 3.4: The LeapMotion system

3.4.1 Hardware

The *Leap Motion Controller* is really quite straightforward from a hardware view. Two cameras and three infrared LEDs are the core of the device. These track infrared light with a wavelength of 850 nanometers, which is outside the visible light spectrum. [7]

The unit has a big interaction room of 0.22 m^3 thanks to its wide-angle glasses, which takes the form of an inverted pyramid – the intersection of the areas of perspective of the binocular cameras (see figure 3.4b). The viewing range of the device is 60cm to 80cm, depending on the version of the firmware used.

This raw data is then stored in the device's local memory and then sent via USB to the *Leap Motion tracking software*. As the cameras work with near-infrared light, the data is in the form of grayscale stereo images, as shown in figure 3.5.

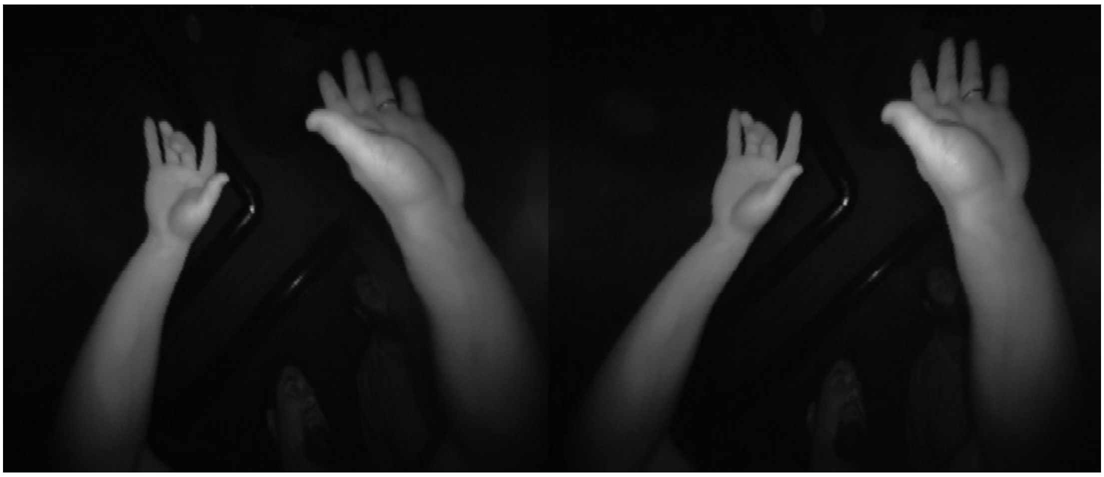


Figure 3.5: Leap Motion raw data

3.4.2 Sofware

It's time for some heavy mathematical lifting once the picture information is streamed to the computer. The *Leap Motion Controller* does not produce depth maps despite common misconceptions - instead it applies sophisticated algorithms to the raw sensor information.

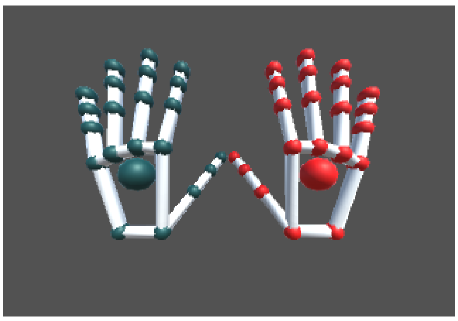


Figure 3.6: Capsule hands

The Leap Motion Service first compensates background objects (e.g. head) and lightning, and then extracts from the data the relevant information - arms, hands and fingers.

Though a transport layer, the results(frames) are fed to the *Leap Motion Control Panel* or to native and web clients. These orgaize the data into an object-orietented structure.

Although the device itself has been the same since it's launch in 2013, the software has undergone several do-overs and upgrades. Their SDK started with desktop-only tracking capabilities for its first two major versions. In 2014, they added a VR tracking mode to it and in 2016 they released Orion, Leap Motion's VR-dedicated SDK.

3.4.3 Gestures and detectors

The Leap Motion API [8] defines mappings for four human body parts:[8]

1. **Arm**
   The name of this data structure is a bit misleading because it actually represents the forearm - the lower part of the arm, from elbow downto wrist. There are a maximum of two arms in the scene, each with only one **Hand** attached to it.

2. **Hand**
   There can be a maximum of two hands in the scene at any given time. Hands have a special attribute called *handedness*, which determines if the hand is *left* or *right*. Each hand has exactly five fingers attached to it. If the Leap Motion controller cannot see one of the fingers (because it is obscured by the hand or by another object), it will try to determine its pose based on past data.

3. **Finger**
   Each finger has three joints that can be used to attach new visual models to the hands (e.g. different colour capsule hands, natural looking hands). Fingers only have data about their tip position and direction, and the four bones inside them. Fingers can be of one of five types - *thumb, index, middle, ring* or *pinky.*

4. **Bone**
   Bones can be of one of four type - *metacarpal, proximal phalange, intermediate phalange* or *distal phalange.* Even though the thumb does not have a *metacarpal* bone in it in the real world, Leap Motion decided to add a *zero length metacarpal bone* to the thumb, so that the fingers are kept consistent regardless of their type (see figure 3.7).

Leap Motion offers a variety of gesture detectors already implemented, which can also be combined by the use of a Logic Gate. The logic gate is a higher level detector, combining two or more basic detectors.

As an example, a "thumbs up" gesture would be detected as combination of the following detectors:

- **Finger Extended Detector** - configured to detect a thumb extended and other fingers not extended (figure 3.8a)

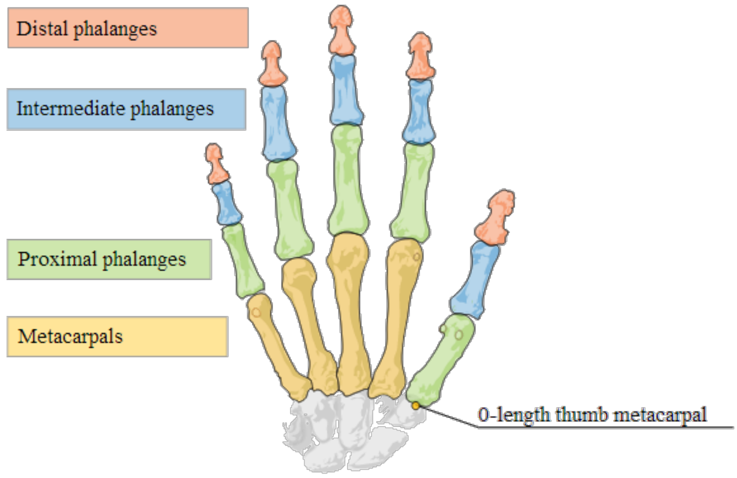
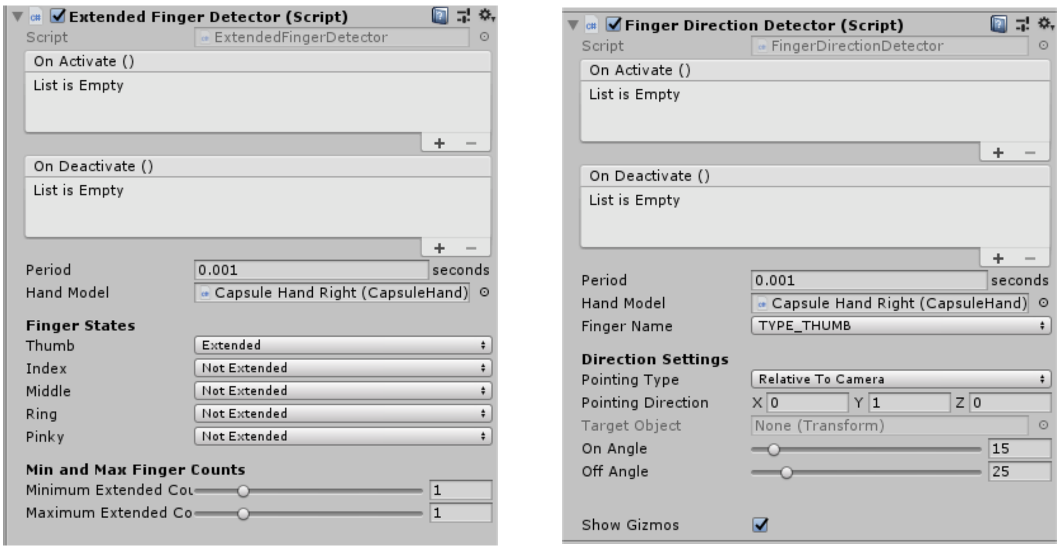


Figure 3.7: Leap Motion bones

- **Finger Pointing Detector** - configured to detect that the thumb is pointing up (*Vector3(0, 0, 1)*) relative to the horizon (figure 3.8b)



(a) Leap Thumb extended detector

(b) Leap Thumb pointing up detector

Figure 3.8: Leap Motion basic detectors for the "thumbs-up" gesture

- **And Logic Gate** - to combine the other two detectors and have callbacks (C# scripts) attached to it (figure ref 3.9)

This approach requires adding three components to a game object and referencing the first two detectors (Finger Extended Detector and Finger Pointing Detector) from the

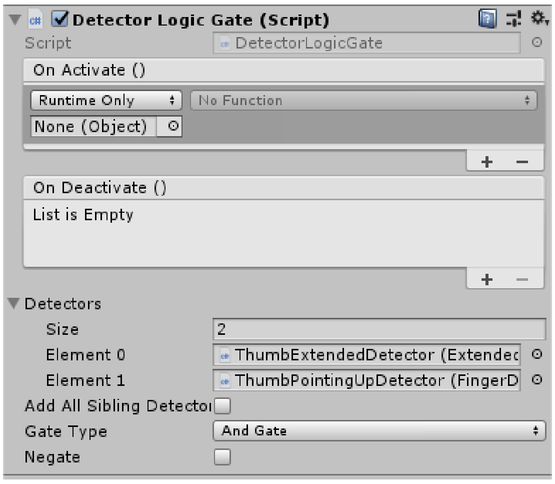


Figure 3.9: Leap Thumb pointing up detector by combining the detectors 3.8a and 3.8b

Logic Gate. This can quickly get out of hand when requiring a high number of combined gestures. The logic gate detector is also highly coupled to the other two detectors, and any change in the base detectors (conditions, renaming or, the worse, moving) will come with a change in the logic gate detector. Even though, with this approach, the basic detectors are reusable, the logic gates usually are not.

This issues fuel the need for a higher level API that is flexible and easily extendable, and which has high reusability. The API should adhere to a programming model which reacts to change rather than query components for changes so that is gives high performance.

3.5 Reactive programming

Reactive programming is a declarative programming paradigm concerned with asynchronous data streams and the propagation of change. With this paradigm it is feasible to easily express static (e.g. lists) or dynamic (e.g. events) information streams and to also indicate that an implied dependency remains within the related implementation model, which promotes the automatic propagation of the altered information stream.

Examples of Reactive Programming include hardware description languages (HDLs), such as VHDL or Verilog, in which changes are modeled as they propagate through a circuit. As a manner to optimize the development of dynamic user interfaces and virtually-real-time system animation, reactive programming has been suggested.

3.5.1 Reactive Extensions

ReactiveX is a powerful library for asynchronous and event-based programming. It is an implementation of the observer pattern meant for event-driven programming. It

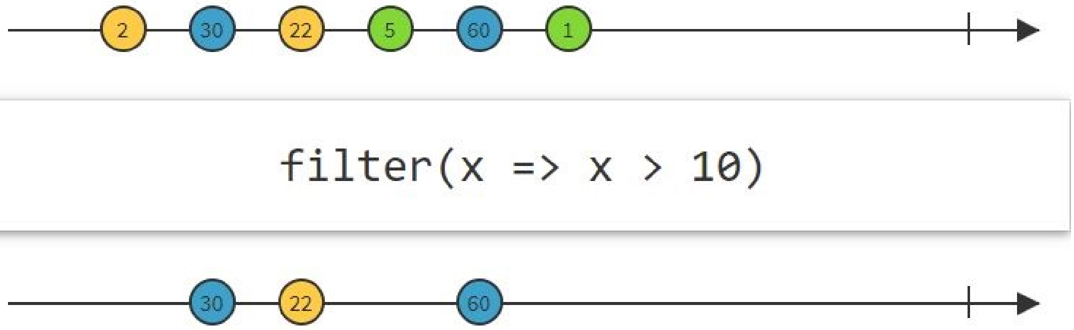


Figure 3.10: Example of a RX operator

also extends the observer pattern with operators that allow the user to compose sequences declaratively without worrying about low-level concerts (such as mutlithreading and the problems that come with it).

Figure 3.10 shows how an operator works on an observable. In the example, the operator is *filter*. *Filter* takes as input a predicate, a function that maps a value to a boolean (true or false). So, from the source observable [2, 30, 22, 5, 60, 1], by filtering the elemnts greater than 10, we are left with only [30, 22, 60]. Note that the elements are emitted in the same order that they were in the source, almost instantly. The vertical line at the end represents the end of the observable stream. One can attach a callback to that, called *OnComplete*.

The main data structure used by ReactiveX is *Observables*. As stated on their intro page:

> You can think of the Observable class as a “push” equivalent to Iterable, which is a “pull.” With an Iterable, the consumer pulls values from the producer and the thread blocks until those values arrive. By contrast, with an Observable the producer pushes values to the consumer whenever values are available. This approach is more flexible, because values can arrive synchronously or asynchronously. (ReactiveX intro)

Code snippets 3.1 and 3.2 show the resemblance between the iterable and observable. One might say that the only difference is the call to *subscribe* instead of *forEach*. While, indeed, both of the code snippets produce the same result, the real difference is the data flow.

In the *forEach* example, the thread is blocked until 15 elements arrive from the *getDataFromNetwork* call (first 10 are skipped, then only 5 are processed by the *map*).

In the *subscribe* example, the only delay in the thread's execution is the creation of the observable stream, after which other instructions are executed. When data arrives from the *getDataFromNetwork*, the thread which created the observable is interrupted and data is processed.

```
GetDataFromLocalMemory()
  .Skip(10)
  .Take(5)
  .Select(s -> $"{s} transformed")
  .ForEach(s -> Console.WriteLine($"next -> {s}"));
```

Listing 3.1: Iterable

```
GetDataFromNetwork()
  .Skip(10)
  .Take(5)
  .Select(s -> $"{s} transformed")
  .ForEach(s -> Console.WriteLine($"onNext -> {s}"));
```

Listing 3.2: Observable

ReactiveX observables are intended to be ***composable, flexible***and ***less opitionated***. These provide a huge advantage over structures like Java *Futures* or C# *Awaitables*, because it removes the need for ambiguous nesting of callbacks.

RX Observables also offer three methods for of flow control - *OnNext*, *OnError* and *OnCompleted* - which give the programmer a high degree liberty. Table 3.1 shows how observables integrate in the programming world, at the crossroads of asynchronous multiple items data streams.

	single items	multiple items
synchronous	T GetData	IEnumerable<T> GetData
asynchronous	Awaitable<T> GetData	Observable<T> GetData

Table 3.1: Observable position in multiple items and asynchronous world

Chapter 4

Analysis and Theoretical Foundation

Together with the next chapter takes about 60% of the whole paper

The purpose of this chapter is to explain the operating principles of the implemented application. Here you write about your solution from a theory standpoint - i.e. you explain it and you demonstrate its theoretical properties/value, e.g.:

- used or proposed algorithms
- used protocols
- abstract models
- logic explanations/arguments concerning the chosen solution
- logic and functional structure of the application, etc.

YOU DO NOT write about implementation.

YOU DO NOT copy/paste info on technologies from various sources and others alike, which do not pertain to your project.

Chapter 5

Detailed Design and Implementation

Together with the previous chapter takes about 60% of the paper.

The purpose of this chapter is to document the developed application such a way that it can be maintained and developed later. A reader should be able (from what you have written here) to identify the main functions of the application.

The chapter should contain (but not limited to):

- a general application sketch/scheme,
- a description of every component implemented, at module level,
- class diagrams, important classes and methods from key classes.

Chapter 6

Testing and Validation

About 5% of the paper

6.1 Title

6.2 Other title

Chapter 7

User's manual

In the installation description section your should detail the hardware and software resources needed for installing and running the application, and a step by step description of how your application can be deployed/installed. An administrator should be able to perform the installation/deployment based on your instructions.

In the user manual section you describe how to use the application from the point of view of a user with no inside technical information; this should be done with screen shots and a stepwize explanation of the interaction. Based on user's manual, a person should be able to use your product.

7.1 Title

7.2 Other title

Chapter 8

Conclusions

About. 5% of the whole
Here your write:

- a summary of your contributions/achievements,
- a critical analysis of the achieved results,
- a description of the possibilities of improving/further development.

8.1 Title

8.2 Other title

Bibliography

[1] 2019. [Online]. Available: https://www.etymonline.com/search?q=virtual

[2] A. Artaud, *The Threatre and its Double*, 1938.

[3] "Virtual and augmented reality: The next big computing platform?" *Goldman Sachs*.

[4] K. D. et al., "Virtual reality for rehabilitation in parkinson's disease," *Cochrane Database of SYstematic Reviews*, vol. 2016, pp. 1–50, 2016.

[5] K. Adam, *Gesture: Visible Action as Utterance.* Cambridge University Press, 2004.

[6] R. M. K. et al., *Lexical Gestures and Lexical Access: A Process Model.* Cambridge University Press, 2001, pp. 261–283.

[7] A. Colgan, "How does the leap motion controller work?" 2014. [Online]. Available: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/

[8] L. M. Inc., "Leapmotion developer - using the tracking api," 2019. [Online]. Available: https://developer-archive.leapmotion.com/documentation/python/devguide/Leap_Guides2.html

Appendix A

Relevant code

```
 /** Maps are easy to use in Scala. */
object Maps {
  val colors = Map("red" -> 0xFF0000,
                   "turquoise" -> 0x00FFFF,
                   "black" -> 0x000000,
                   "orange" -> 0xFF8040,
                   "brown" -> 0x804000)
  def main(args: Array[String]) {
    for (name <- args) println(
      colors.get(name) match {
        case Some(code) =>
          name + " has code: " + code
        case None =>
          "Unknown color: " + name
      }
    )
  }
}
```

Appendix B

Other relevant information (demonstrations, etc.)

Appendix C

Published papers