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) _____, legitimat(ă) cu _____ seria _____ nr. _____ CNP _____, autorul lucrării _____

_____

_____

elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, 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 _____

Semnătura

Contents

Chapter 1 Introduction - Project Context **1**
1.1 Virtual reality . . . 1
1.1.1 History . . . 1
1.1.2 Modern technology . . . 2
1.2 Gesture recognition . . . 2
1.3 Reactive programming . . . 3

Chapter 2 Project Objectives and Specifications **4**
2.1 Introduction . . . 4
2.2 Positioning . . . 4
2.2.1 Problem statement . . . 4
2.2.2 Product Position Statement . . . 5
2.3 Stakeholder and User Descriptions . . . 5
2.3.1 Stakeholder summary . . . 5
2.3.2 User summary . . . 5
2.3.3 User environment . . . 6
2.3.4 Summary of key stakeholder or user needs . . . 6
2.3.5 Alternatives and competetion . . . 6
2.4 Product overview . . . 6
2.4.1 Product perspective . . . 7
2.4.2 Assumption and dependencies . . . 7
2.5 Product features . . . 7
2.6 Other product requirements . . . 8

Chapter 3 Bibliographic research **10**
3.1 Gestures . . . 10
3.2 Leap Motion . . . 10
3.2.1 Hardware . . . 10
3.2.2 Sofware . . . 11
3.3 Reactive Extensions . . . 12
3.3.1 Why use observables? . . . 12
3.4 Title . . . 12

3.5 Other title 12

Chapter 4 Analysis and Theoretical Foundation **14**
4.1 Title 14
4.2 Other title 14

Chapter 5 Detailed Design and Implementation **15**

Chapter 6 Testing and Validation **16**
6.1 Title 16
6.2 Other title 16

Chapter 7 User's manual **17**
7.1 Title 17
7.2 Other title 17

Chapter 8 Conclusions **18**
8.1 Title 18
8.2 Other title 18

Bibliography **19**

Appendix A Relevant code **20**

Appendix B Other relevant information (demonstrations, etc.) **21**

Appendix C Published papers **22**

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, militry 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.

1.3 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.

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. The details of how the Fluent Motion fulfills these needs are detailed in the use-case and supplementary specifications.

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 Leap-Motion'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.

2.4.1 Product perspective

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

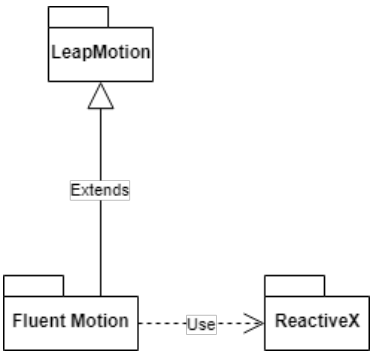


Figure 2.1: Fluent Motion architectural diagram

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 Gestures

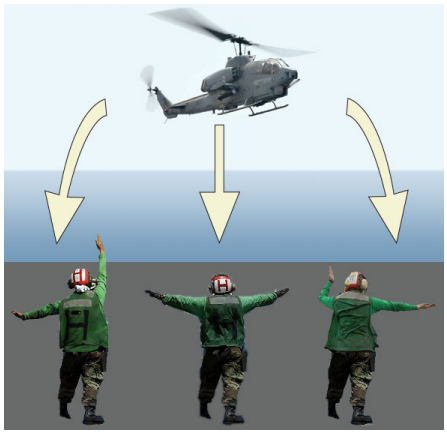


Figure 3.1: Military air marshallers use hand and body gestures to direct flight operations aboard aircraft carriers

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. [3]

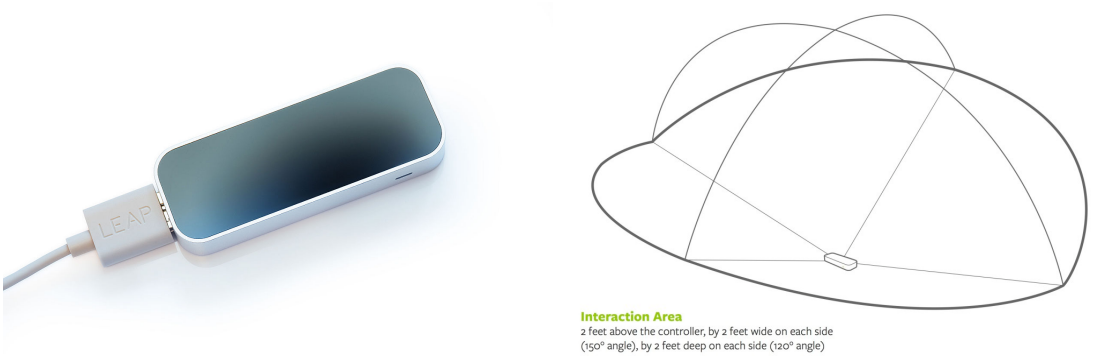
Gestures can be of two types: *informative (passive)* or *communicative (active)*.

3.2 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.

3.2.1 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. [4]



(a) The LeapMotion controller (b) The interaction area

Figure 3.2: The LeapMotion system

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 3.2b). 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 3.3.

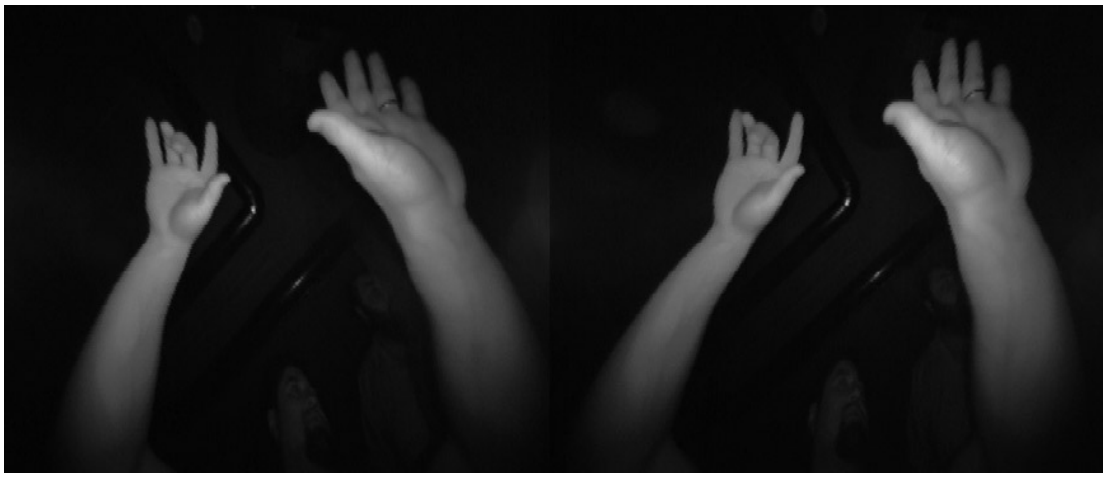


Figure 3.3: Leap Motion raw data

3.2.2 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 com-

mon misconceptions - instead it applies sophisticated algorithms to the raw sensor information.

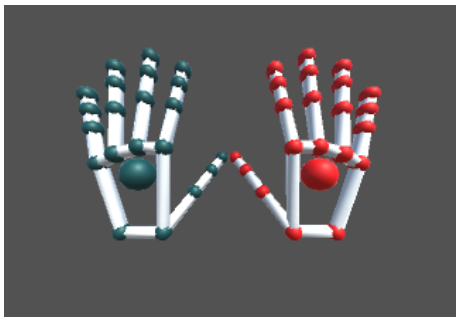


Figure 3.4: Capsule hands

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.

3.3 Reactive Extensions

ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences. [5]

It extends the observer pattern to support data and/or event sequences and provides operators that help you compose sequences declaratively while abstracting issues such as low-level threading, synchronization, thread security, concurrent data structures, and non-blocking I/O.

Table 3.1 shows how observables integrate in the programming world.

	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

3.3.1 Why use observables?

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 very much liberty.

3.4 Title

3.5 Other title

Iterable

```
getDataFromLocalMemory()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed"
})
  .forEach({ println "next => " + it })
```

Observable

```
getDataFromNetwork()
  .skip(10)
  .take(5)
  .map({ s -> return s + " transformed"
})
  .subscribe({ println "onNext => " + it
})
```

Figure 3.5: Iterable vs Observable

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.

4.1 Title

4.2 Other title

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

[4] 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/

[5] 2019. [Online]. Available: http://reactivex.io/intro.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