

MANIPULATION OF INTERNET BROWSER WITH MULTIMODAL DEVICES
AND
INTRODUCTION OF MULTIMODAL INTERACTIVE FRAMEWORK C.O.A.L.S

by

VARCHALAMAS PETROS

B.A., Technological Educational Institute of Crete, 2015

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2015

Approved by:

Major Professor
Nikolaos Vidakis

Abstract

Humans interact with each other and with their environment in everyday life by utilizing the five basic senses as input modalities while using for output, modalities like sounds, gestures, gaze, facial and body expressions. In the other hand, when talking about human computer interaction, the existence of a difficulty of interaction between both sides, is always present.

Thus through Human Computer Interaction science (HCI), this kind of interaction tends to be more user friendly and operative through intuitive actions related to the frequent and natural human behavior. Multimodal interaction provides an alternative means of communication with an application, which is different enough from the traditional WIMP style. Despite the great amount of devices in existence, most applications make use of a limited set of modalities, most notably speech and touch.

This thesis describes the procedure to interact with Google Chrome browser through browser extensions, while performing hand and finger gestures and using them as multimodal input to the system from the Leap and the Kinect sensors. In addition, it is introduced the interactive multimodal framework C.O.AL.S., which enables the deployment of a vast variety of modalities. More specifically by fusing - combining different modalities from different sources or from a single source, in order to achieve the interpretation and extraction of semantic meaning from them which can vary according to the context, the task, the user and the time and finally to give the user the correct feedback according to the information gathered from the previous procedure.

Table of Contents

Abstract	ii
List of Figures	1
List of Tables	4
Acknowledgements.....	5
Chapter 1 - Introduction.....	6
Summary.....	7
Dissertation's overview	8
Research questions.....	9
Chapter 2 - Review and State of the Art	10
Multimodal Interactive Systems	10
Input Interpretation - Fusion Engines	12
Dialogue Management	14
Output Generation - Fission engines.....	15
Multimodal devices.....	16
Microsoft Kinect.....	18
Comparison between Kinect for Windows and Kinect for Xbox 360	18
Depth Sensor and Infrared light transmitter.....	20
RGB Camera.....	21
Microphones	21
Accelerometer	22
Led Light.....	23
Motorized Tilt	23
Available APIs for the Kinect.....	24
Leap Motion.....	25
The Leap device hardware overview	25
Software overview	26
Public use	28
A reference to other multimodal devices	29
Xtion Pro LIVE by Assus	29
MYO by Thalmic	30

Oculus Rift by Oculus VR.....	31
Google Glass by Google X.....	32
Plugins & Extensions.....	33
Plugins.....	33
Extensions.....	34
Blended Learning.....	36
Chapter 3 - Architectural Design & Description.....	39
C.O.A.L.S. Framework.....	39
Google Chrome Extensions.....	45
Chapter 4 - Approach of the Implementation.....	53
Loading the extension in Chrome.....	53
YouTube player Interaction with Leap Motion.....	55
Server side - Web Sockets Communication.....	55
Client side - Myscript.js walkthrough.....	56
Leap API.....	58
Interaction with YouTube page.....	61
Popup and Options Page.....	64
Browser Interaction with Microsoft Kinect.....	66
Server side – Communication & Data Initialization.....	66
Client side – Myscript.js and Popup.js walkthrough.....	69
Chapter 5 - Conclusion, Final Thoughts & Future Work.....	77
References.....	79

List of Figures

Figure 1 - Human Computer Interaction.....	10
Figure 2 - Classic peripherals	16
Figure 3 - Belkin nostromo n52te gamepad.....	17
Figure 4 - Xbox One Controller.....	17
Figure 5 - PlayStation 4 controller.....	17
Figure 6 - Nintendo Wii controller	17
Figure 7 - Kinect interior	18
Figure 8 - Kinect fusion in action	19
Figure 9 - Structure of IR light map.....	21
Figure 10 - Image from depth sensor.....	21
Figure 11 - RGB image from the Kinect	21
Figure 12 - Microphones inside the Kinect.....	22
Figure 13 - Kinect's Accelerometer coordinate system	22
Figure 14 - Kinect's accelerometer	23
Figure 15 - Kinect's LED light.....	23
Figure 16 – Kinect's Motorized Tilt.....	24
Figure 17 - The Leap device	25
Figure 18 - The internal parts of the Leap device.....	25
Figure 19 - Interaction area of the Leap sensor	26
Figure 20 - Leap's smaller observation area and high resolution.....	27
Figure 21 - Xtion Pro LIVE hardware	29
Figure 22 – Xtion Pro	29
Figure 23 – Xtion Pro LIVE	29
Figure 24 - MYO armband	30
Figure 25 - Oculus Rift DK1	31
Figure 26 - Oculus Rift DK2	32
Figure 27 - Google Glasses.....	32
Figure 28 - Multimodal human-machine interaction loop.....	39

Figure 29 - Multimodal system architectural design	40
Figure 30 - Modalities & Devices.....	41
Figure 31 - State transactions of the C.O.A.L.S. frame	43
Figure 32 –The modalities that can deliver these three basic modes.....	44
Figure 33 - Manifest in JSON format	50
Figure 34 - Background.html.....	51
Figure 35 - Popup.html	51
Figure 36 - Content script interaction	52
Figure 37 - Extensions in Chrome's options	53
Figure 38 - Extensions manager in Chrome	54
Figure 39 - Page action icon	54
Figure 40 - Browser action icon.....	54
Figure 41 - Web Sockets Communication	55
Figure 42 - Check Connection function.....	57
Figure 43 - Check focused Tab function.....	58
Figure 44 - Pointables information retrieved from Leap API.....	59
Figure 45 - Hands information retrieved from Leap API.....	59
Figure 46 – Pointables information retrieved by the Leap API.....	60
Figure 47 – Hands, Pointables and Fingers tracked.....	60
Figure 48 - Gestures provided by the Leap API	61
Figure 49 - Event triggered from circle gesture.....	62
Figure 50 - Get child elements and set them focused	63
Figure 51 - Popup Html page.....	64
Figure 52 - Extension's Options Page	65
Figure 53 - Web Socket Connection.....	67
Figure 54 - Device & Data stream initialization.....	68
Figure 55 - Server sends Blob & Json to the Client.....	69
Figure 56 - Drawing the user's skeleton on canvas.....	70
Figure 57 - Parsing retrieved Json data.....	71
Figure 58 - Getting skeleton's Id.....	71
Figure 59 - Get skeleton joints' names.....	71

Figure 60 - Specify joints to create gestures	72
Figure 61 - Functions called for gesture recognition	72
Figure 62 - Creation of right swipe gesture	73
Figure 63 - Creation of wave gesture.....	74
Figure 64 - Zoom in - out gesture and actions triggered.....	76

List of Tables

Table 1 - Input devices and their resulted actions.....	16
Table 2 - Multimodal - Complex gaming devices	17
Table 3 - Tokenization of the recognized data.....	41
Table 4 - Complete C.O.A.L.S. frame example.....	43
Table 5 - Actions triggered by gestures	75

Acknowledgements

I want to express deepest gratitude to my family and my friends who supported me in every possible way.

Chapter 1 - Introduction

It is a fact that over the years and due to the evolution of technology, the relationship between man and computer is becoming increasingly complicated and inherently multimodal. When referring to modality according to Bellik and Teil [1] it is “a concrete form of a particular communication mode” where **mode** is defined as the five human senses which constitute the receiving information and the multifarious ways of human expression such as gesture, speech, facial expressions and others which constitute the product information.

The interaction between them is studied from the scientific field of **Human Computer Interaction (HCI)** [2], taking consideration both sides of the human and the machines.

From the side of the machines, the interaction is related to the art of graphics, operating systems, programming languages, the inner material of the devices and probably their peripherals. On the other hand, what characterizes humans, is communication, linguistics, social sciences, psychology and the human factor. So, given the multidisciplinary nature of HCI, people with different scientific knowledge can contribute to its success.

The interaction between users and computers occurs inside the physical user interface (**Natural User Interface - NUI**) [3], [4], which is a multimodal interface that is easy to interact with and with the prospect of becoming invisible after its successful learning. This implies that the multimodal interaction facilitates a more natural communication between humans and systems in both input and output of data. But to make this possible, the right technology is needed, while also being affordable for the average user.

To achieve the detection of the human body motion, the equipment needed is more specialized and consequently more expensive, like digital sensors that provide high accuracy and efficiency in the conception of the machines. Their specialization, the difficulty to be controlled by anyone and in many cases their size, has the effect of turning these devices hardly accessible to the users.

Fortunately due to the technological evolution, devices with features as these that were previously mentioned and with affordable price were released recently to the public, from very big companies in the technology area, like *Kinect* by Microsoft, the *Xtion Pro LIVE* by Assus,

The *Leap* by Leap Motion, *MYO* by Thalmic, *Google Glass* by Google X and *Oculus Rift* by Oculus VR.

With their particular architecture, combining input from electromagnetic sensors, accelerometer sensors, gyroscope sensors and from devices such as RGB camera, microphone and the most important the depth camera, providing a remarkable motion tracking, giving the opportunity and the freedom to developers to evolve more human-centric applications, while giving a new impetus to what is so far known about the human machine interaction.

Summary

As technology evolves, so does everything that is related to it or tries to keep up at least. Inside the educational system the technological advances were always utilized in order learning to be more successful in many ways. So blended learning or technology-mediated instruction begins to exist since the '60s [5]. Through blended learning a student learns at least in part through delivery of content and instruction via digital and online media with some element of student control over time, place, path, or pace [6].

As part of the educational methods the games were always important and due the progress of both sometimes they can't be separated, especially when referring to preschool and younger children. Education through entertainment starts to be popular during the '90s and in general refers to any kind of education that also entertains even though it is usually associated with video games with educational aims [7]. These games are known as educational games or serious games.

Serious games are (digital) games used for purposes other than mere entertainment, allow learners to experience situations that are impossible in the real world for reasons of safety, cost, time, etc., but they are also claimed to have positive impacts on the players' development of a number of different skills.

With a series of games, which are developed to help kids with or without disabilities to improve their skills in reading and mathematics, the users can interact with the system using input modalities to receive information feedback through output modalities like sounds, graphics, speech synthesis or a combination of these.

More specifically, **the first aim of this thesis** is to present an interactive multimodal system that helps to achieve this natural human computer interaction, by **fusing**, meaning **combining** all these various and different **modalities** from **different sources or from a single source**, in order to achieve **the second aim**, meaning the **interpretation** and **extraction** of **semantic meaning** from them which can vary according to the context, the task, the user and the time. **The third aim**, is to give the user the correct **feedback** according to the information gathered from the previous procedure. Finally **the last aim** is to gather the different modalities as input to the system with the use of complex **multimodal devices/sensors** trying to minimize the use of single modality devices.

Through the next chapters the key components of the system will be presented as well as the way they are used in our approach alongside the topics that this thesis is part of.

Dissertation's overview

This dissertation consists of 5 chapters.

1st chapter → Brief introduction about the topics that concern this thesis, followed by research questions in order to specify more the area that will be covered.

2nd chapter → Review and the state of the art of the following topics:

- Multimodal Interactive Systems
- Multimodal Devices
- Blended Learning

3rd chapter → Description of the Architectural design of C.O.A.L.S framework, aiming to combine multiple modalities.

Description of the Architectural design of the extensions hosted by the Google Chrome browser.

4th chapter → Approach of the implementation providing code snippets with explanation.

5th chapter → The thesis ends with this chapter, which is composed of a general conclusion as well as perspectives and future work related to the subjects presented in here.

Research questions

In this section there are some questions cited, in order to guide the reader about the topics and their issues that will be presented in the next chapters.

- i. Could the educational methods be improved? How is this possible? How effective could it be?
- ii. What is multimodality and how it affects the applications?
- iii. Following this specific architectural design solves the problems of combining multimodal input and output in our application?
- iv. The multimodal devices used in this system, are a good choice?
- v. Are there any limits which should not be exceeded in our system and subsequently in our use case scenarios?

Chapter 2 - Review and State of the Art

Multimodal Interactive Systems

Multimodal Human–Computer Interaction (MMHCI) is a scientific field that lies at the crossroads of several research areas such as computer vision, artificial intelligence (AI), psychology, social sciences and many others. Through MMHCI there has been efforts to determine how computer technology can be more usable by people, which requires the understanding of three things like:

- The human factor meaning the users themselves,
- The machine factor meaning the system (the computer technology and its usability), and lastly
- The interaction between both of them.

By considering these aspects, it is obvious that MMHCI is a multidisciplinary science field since the designer of an interactive system should have expertise in a range of fields: psychology and cognitive science to understand the user’s perceptual, cognitive, and problem solving skills, sociology to understand the wider context of interaction, ergonomics to understand the user’s physical capabilities, graphic design to produce effective interface presentation, computer science and engineering to be able to build the necessary technology, etc. Thus, given the multidisciplinary nature of HCI, people with different scientific knowledge can contribute to its success.

The above statements lead to the conclusion that the interaction between human and machine is nothing more than elements understandable by the human, on data that understands and interprets the computer. But to create an interaction between these sides the created gap should

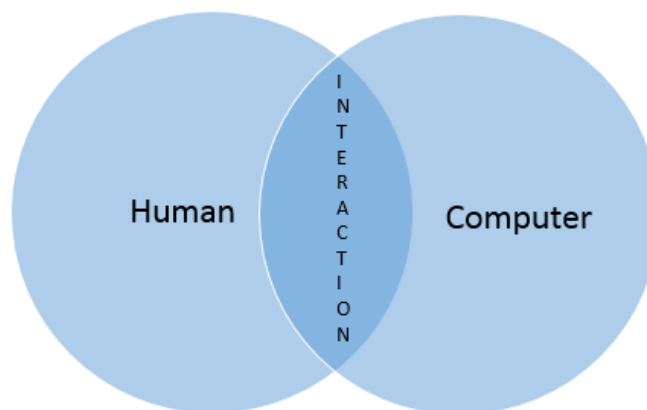


Figure 1 - Human Computer Interaction

be bridged. This bridging is obtainable by the user interface which is defined as: *the part of the human computer interaction that concerns the ways, the methods and the tools that implement the communicative interaction between the user entity and the computer system [4].*

Thus the user interface is consisted by elements which are part of the computer system by giving information about the hardware and the software parts. More precisely the information given to the users could be categorized as:

- System Input, allowing the users to control the system and
- System Output, allowing the system to bring up the results of the users' actions.

Referring to human computer multimodal interaction the term of modality is widely used and is referred to the combination of multiple modalities. These modalities mostly refer to the ways that the system responds to the inputs, i.e. communication channels [8]. The definition of these channels is inherited from human types of communication which are basically their senses like the sight, the hearing, the touch, the smell, and the taste. The possibilities for interaction with a machine include but are not limited to these types [9].

Thus, a multimodal system acts as a simplifier of human-computer interaction via two or more modes of input except the traditional widely used devices of the keyboard and the mouse. The number of supported input modalities, their types and the way in which they work together may vary widely from one multimodal system to another. Multimodal systems combine different combinations of speech, gesture, gaze, facial expressions and other non-conventional modes of input [10]. Thus multimodality can offer a number of advantages over traditional systems like a more natural and user-friendly experience, giving a more affordance to users and can contributing to system robustness.

A classic example of a multimodal system is the “***Put That There***” demonstration system [11] focusing on pointing and speech combined using speech/mouse, speech/pen [73], speech/gesture [75], or speech/ gaze tracking [74]. This system allowed the user to move an object into a new location on a map on the screen by saying “put that there” while pointing to the object itself then pointing to the desired destination. Since this pioneer work, multimodal interaction developers have strived to integrate more modalities, to refine hardware and software components and to explore limits and capabilities of multimodal interfaces. Another examples of multimodal applications systems are listed below:

- Smart Video Conferencing [12].
- Intelligent Homes/Offices [13].
- Driver Monitoring [14].
- Intelligent games [15].
- E-Commerce [16].
- Helping People with Disabilities [17].

In the following sections it will be analyzed the stages that manipulate the information retrieved from the system input in order to produce the desired output results.

Input Interpretation - Fusion Engines

Although an ideal multimodal HCI system should contain a combination of single modalities that interact correlatively, the practical boundaries and open problems in each modality oppose limitations on the fusion of different modalities.

User input interpretation is one of the most crucial aspects of multimodal interactive systems. This task can be a simple task, but tends to increase in complexity, in dependence with factors such as the number and type of modalities involved, the architectural design of the system, and user and context requirements.

As a system expands in terms of interaction modalities available, so does the amount and diversity of data received. For that reason, a mechanism that can correctly interpret all these data is needed. This mechanism must take into account, the way a user interacts with the system, the restrictions imposed by the user context and the user surroundings, and special confinements imposed by the application.

Multimodal fusion has been a research subject for the past two decades [1]. Sharma et. al. [18], considers three levels of fusion for incoming data:

- Data-level (or sensor-level) fusion,
- Feature-level fusion and
- Decision-level fusion.

Sanderson and Paliwal [19] define terms with similar meaning such as

- Pre-mapping,

- Midst-mapping and
- Post-mapping fusion.

Data-level fusion focuses on the fusion of either identical or tightly-linked types of multimodal data. An illustration of data-level fusion is the recording of a scene by multiple depth cameras, from different angles [20]. Data-level fusion barely deals with the semantics of the input data. However, it enriches or correlates data that is potentially processed by a higher-level fusion process. Data level fusion, works on the raw data, produced by the recognizers, which makes it very susceptible to noise or failures. For this reason, data-level fusion involves a very basic recognition, and/or noise filtering.

Feature-level fusion focuses on data that is already processed by filters. Fusion is applied on features extracted from the data, rather than the raw data. This kind of fusion is typically applied to closely-coupled modalities with possibly different representations. A classic example is presented in [21], where data modalities are speech that is recorded by a microphone, and lip movement, recorded by a camera. Specifically, two distinct, synchronized data streams of different modalities are used, and the goal is to achieve a more reliable speech recognition through information combining. Feature-level fusion works with pre-processed data, which makes it less susceptible to noise and failures than data-level fusion, although it conveys a lower level of information detail.

Decision-level fusion derives interpretations based on semantic information. As claimed in [22] this is the most common type of fusion. This fusion level correlates information coming from loosely coupled modalities, such as speech and gesture. Due to this fact, decision-level fusion is the most common fusion type in multimodal systems, and also the most versatile. Decision-level fusion includes processes such as merging of high-level information obtained by data- and feature-level fusion, modelling of human-computer dialogues and mutual disambiguation using information derived from feature-level fusion [23].

Dialogue Management

The dialogue management component is in charge of identifying the dialogue state, the action to communicate to the given application and/or to generate the message that will be returned through the fission engine.

Time constraints are crucial in multimodal systems, and all modalities should be properly time-stamped. When two modalities are used in parallel in a synergistic manner, it is important to know the order of the commands in order to perform the task accurately.

T. H. Bui [24] identifies four different approaches to dialogue management:

- **Finite state and frame state approaches.** In this approach, the dialogue structure is represented as a state machine. Frame-based models, an extension of finite-state models, use a slot-filling strategy in which a number of predefined information sources have to be gathered [25].
- **Information state-based approaches.** Human machine dialogue is described by following information states consisting of five main components, namely informational components, formal representations of the information components, dialogue moves, update rules and an update strategy [26].
- **Plan-based approaches.** Plan-based approaches are based on plan-based theories of communicative action and dialogue [25]. These theories claim that the speaker's speech act is part of a plan and that it is the listener's responsibility to identify and respond appropriately to this plan [27].
- **Collaborative agents-based approaches.** These approaches view dialogue as a collaborative process between intelligent agents. The agents work together to obtain a mutual understanding of the dialogue. This induces discourse phenomena such as clarifications and confirmations [28].

Output Generation - Fission engines

Another integral part of the framework is the multimodal fission module. Fission is responsible for generating and selecting the appropriate communication channels and output modalities according to the user profile and the environmental context.

Systems that combine output modalities evolved since the early nineties where graphics and text were combined (e.g. COMET [28]). Later, systems integrated more output modalities such as speech, 2D/3D animations and avatars (e.g. SmartKon [29], MIAMM [30]). More recent approaches include mobile personal assistants (e.g. Microsoft Cortana [31], Apple Siri [32]). Even though most applications use a limited repertoire of output modalities and for that reason straightforward fission techniques, dealing with the aforementioned output combinations, makes the presentations more complex and more difficult to coordinate.

According to Oviatt [33], fission engines should follow three tasks:

- **Message construction.** The information to be included, must be selected and structured in a proper manner. For example it is mandatory to decompose the semantic information generated by the dialogue manager into specific data to be presented to the user. The most prevalent approaches are the schema-based [34] and the plan based [35] approach.
- **Modality selection.** Following the message construction, the message must be adapted to the user profile and environmental context, and allocated to the appropriate output channels. The three prevalent approaches for the accomplishment of this goal are the rule based [36], the composite based [37], and the agent based approach [38].
- **Output coordination.** The final step of modality fission is the instantiation, which consists of getting the lexical and syntactic content and the modality attributes. Firstly, the concrete content of the message is chosen and subsequently the modality attributes, such as spatial and temporal parameters are established. For a more coherent and synchronized result, all used channels must be coordinated.

Multimodal devices

In computer science, an input device is any peripheral (piece of computer equipment) used to provide data to an information processing system such as the computer. There are many input devices but the ones which are widely used by the most computer user are the keyboard, the mouse, the microphone, the scanner and the RGB camera.



Figure 2 - Classic peripherals

Every device gives each own system input that result the execution of commands or actions such as:

Input Device	Resulted Actions
Keyboard	Typing, Control software with shortcuts (pressing a combination of keys)
Mouse	Control the cursor, Scrolling, Clicking
Microphone	Receiving sound input
Scanner	Scanning documents
RGB Camera	Receiving image or video input

Table 1 - Input devices and their resulted actions

Multimodal human-computer interaction has sought to provide alternative means of communication with an application which, as the time goes by, will be more and more natural to the users. Despite the great amount of devices in existence, most applications make use of a very

limited set of modalities, most notably speech and touch. Nevertheless input devices that have buttons as controls, like the mouse and the keyboard, can be combined into a single physical device that could be seen as a multimodal - complex device. Many gaming controllers can provide such a thing, like the following in the table below.




Multimodal - Complex <i>gaming</i> controllers for PC and console use	
	
Figure 3 - Belkin nostromo n52te gamepad	Figure 4 - Xbox One Controller
	
Figure 5 - PlayStation 4 controller	Figure 6 - Nintendo Wii controller

Table 2 - Multimodal - Complex gaming devices

Microsoft Kinect

On June 1st, 2009, Microsoft announced a project called Project Natal. The final product of this project, which made its appearance on the market in November 2010, was named Kinect [39] inspired from the words: kinetic (motor) and connect.

It is a motion sensing input device by Microsoft for Xbox 360 gaming console that uses the whole human body to control the games. After a while there was released to public an API in order to develop applications on Windows PC. The integrated depth sensor of the device, is designed and created by the Israeli company PrimeSense, Ltd [40].

From that time onwards, this device revolutionized not only in the field of video games but also in robotics, medicine and many other scientific sectors that can benefit from its useful capabilities.

It also a fact that the performance between ratio and price is quite high. An example of an expensive sensor is SwissRanger 4000, which had a cost of about \$ 10,000 while Kinect currently costs less than \$ 100, although its performance as a sensor is far superior to that of SwissRanger 4000 [41].

It is therefore logical two months after its release, Microsoft, have sold 8 million Kinect devices, giving it the title of the electronic device with the fastest sale in the Guinness book.

Comparison between Kinect for Windows and Kinect for Xbox 360

Both devices may seem identical in the exterior but they have a few but important differences such as:

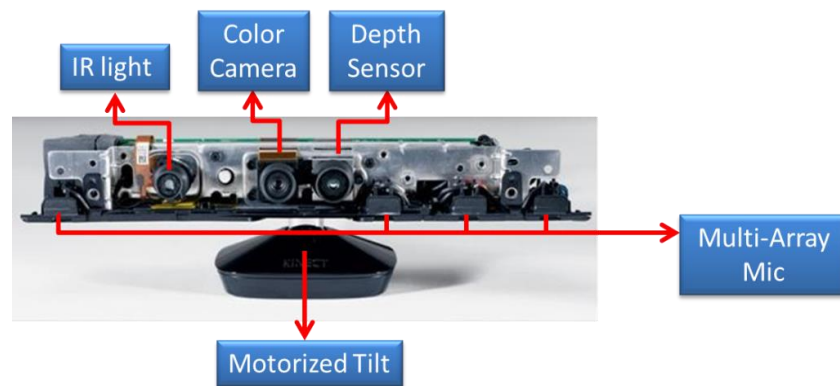


Figure 7 - Kinect interior

1. Near - mode tracking

It is enabled while the user is closer than 40 centimeters from the device, without losing any accuracy of the received information.

2. USB cable

It ensures reliability in a wide range of computers and improves coexistence with other USB peripherals.

3. Extensive camera settings

It provides additional settings to adjust the brightness, so that it can become better coordinated.

4. Kinect Fusion

Kinect Fusion provides 3D object scanning and model creation using a Kinect for Windows sensor. The user can paint a scene with the Kinect camera and simultaneously see, and interact with, a detailed 3D model of the scene. Although the Kinect Fusion can be run at interactive rates on supported GPUs, it also can run at non-interactive rates on a variety of hardware but allowing larger volume reconstructions.

Kinect Fusion in action, taking the depth image from the Kinect camera with lots of missing data and within a few seconds producing a realistic smooth 3D reconstruction of a static scene by moving the Kinect sensor around. From this, a point cloud or a 3D mesh can be produced. [42]

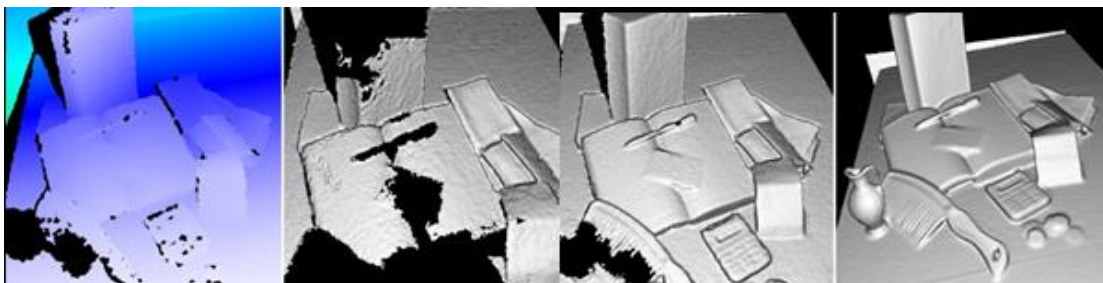


Figure 8 - Kinect fusion in action

5. Product price

- The Kinect for Windows costs near 190 euro.
- The Kinect for XBOX 360 is much cheaper, costing about less than 100 euro. From the other hand in order to connect with a pc, the purchase of a power adapter is needed.

6. Licensing

According to the correct use of the device, set by Microsoft, when a public presentation of an application takes place, the use of Kinect for Windows is essential and not of Kinect for Xbox 360, because otherwise it is illegal.

So the meaning for the big difference between the prices of the two devices, is the acquisition of the rights of use.

Depth Sensor and Infrared light transmitter

Light Coding Technology™ (Zalevsky, Z, et al. patent [43]) allows the Kinect device to create 3D depth maps of a scene also known as triangulation process in real time [44]. The laser source emits a single beam which is split into multiple beams from a diffraction grating, to create a stable pattern spots which are projected onto the surfaces (see Figure 9) and a CMOS image sensor (Complementary Metal Oxide Semiconductor) accepts the reflected rays. The PS1080 SoC chip (System on a Chip) made by PrimeSense that is contained inside the Kinect's system, controls the structure of light spots and processes the data from the CMOS sensor generating depth data in real time [45]. The maximum resolution of the depth image generated by the PS1080 (see Figure 10) is 640x480, with a frequency of 30 frames per second. At 2 meters distance from the sensor there is an accuracy of 3 mm in height and width and of 1 cm in depth. The range of proper operation is from 0.8 to 3.5 meters.

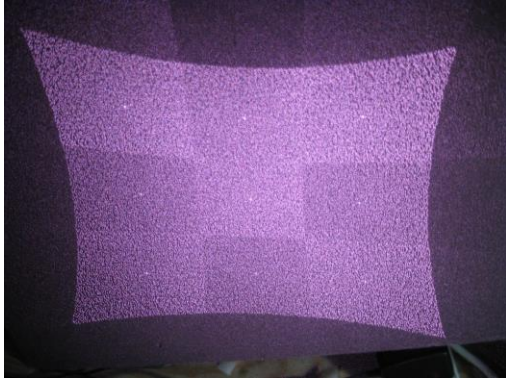


Figure 9 - Structure of IR light map



Figure 10 - Image from depth sensor

RGB Camera



Figure 11 - RGB image from the Kinect

The Kinect device also has a built-in color camera (Color CMOS - VNA38209015) with a maximum resolution of 1280x1024 in order to obtain the real picture beyond the depth map. The frame frequency of the camera is 30 fps and the image produced is good enough to be used in face, fingers and body recognition algorithms. Example image from RGB camera of the apparatus shown in Figure 11 - RGB image from the Kinect.

Microphones

Using an array of four microphones Kinect is able not only to receive sound but also to detect the angle of the source, measuring the sound field.

The description of the sound field can be done by using physical quantities such as the sound pressure and the particle velocity. By measuring these properties it is indirectly possible to obtain the direction of the sound source.

In Figure 12 - Microphones inside the Kinect, we see the position of the microphones in the device which separately processes each of the four channels which receive 16-bit audio sampling frequency equal to 16 kHz.

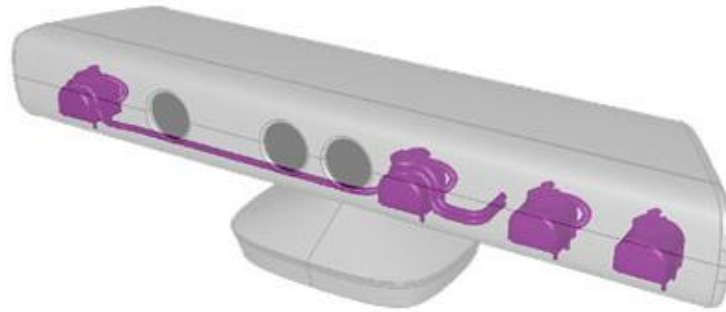


Figure 12 - Microphones inside the Kinect

Accelerometer

The accelerometer is a device which as perceived by the word itself, measures the acceleration. The acceleration is not necessarily coordinates' acceleration (change rate of velocity). Instead, the accelerometer sees the acceleration associated with the weight of any object.

The accelerometers are widely used in multiple applications, in industry and in science fields. Their most common use are inside tablet, computers, smartphones and digital cameras, so that the images on the screens always displayed upright.

The Kinect provides a three-axis accelerometer (KXSD9-1026) which provides the information of the position of the device relative to the gravity, of which the range is up to $2g$ - wherein g is the acceleration due to gravity.

The data returned by the accelerometer is a 3D vector showing the direction of gravity, in the form of a Vector4 (x , y ,

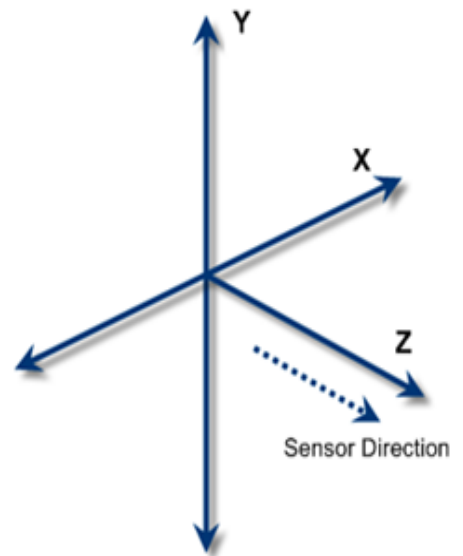


Figure 13 - Kinect's Accelerometer coordinate system

z, w) with the value of w to 0.0. Also the accelerometer is centered on the sensor, being a right hand coordinate system with the positive value of the z axis pointing in the direction facing the sensor (see Figure 14). The default value of the vector is set to (0, -1.0, 0, 0).

So thanks to the sensor data it is identified an "unusual" device orientation and by using the SDK of the device scenarios augmented reality could be achieved. This feature of Kinect has been widely used in the field of robotics.



Figure 14 - Kinect's accelerometer

Led Light

The Kinect device has a LED which has no particular utility other than to indicate the status of the device.

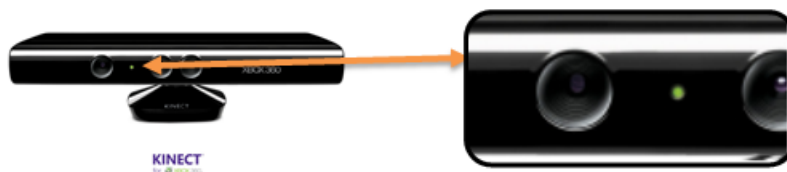


Figure 15 - Kinect's LED light

Motorized Tilt

The device also has a mechanical base, which is dynamically adjusted through the software. The movement of the base comes from a small motor to the size of a coin and from three fragile plastic gears, which are sensitive to heat and perhaps they are the weakest point of the device. The base gives the opportunity to the Kinect device to turn upwards or downwards by 27°.

However, this motor revocation mechanism requires more energy that can be provided through a USB port. For this reason, the device makes use of a special power cable (included in the package along with the sensor), which divides the link into separate USB and power cable. The required energy (12 Watt) is provided by a power adapter.



Figure 16 – Kinect’s Motorized Tilt

Available APIs for the Kinect

There are quite popular APIs to decode the data provided by the Kinect device which are free for public usage like the following:

- The official Microsoft’s SDK [46]
- OpenNI / NITE [47]
- Open Kinect Project [48]
- Libfreenect [49]
- CLNUI [50]
- Evoluce SDK [51]

Leap Motion

The Leap Motion is a company that develops advanced motion detection technology to human-computer interaction.

Originally inspired by the frustration around the 3D modeling, using the mouse and keyboard, they wanted to claim that accomplishing something in virtual reality, should be as easy as in real life.

Not giving the public movements since 2010, the company announced its first product, The Leap, on May 21, 2012 [52].



Figure 17 - The Leap device



Figure 18 - The internal parts of the Leap device

The Leap device hardware overview

From a hardware point of view, the Leap Motion sensor is quite simple [53]. The interior parts of the device (see Figure 18) consists of two cameras and three infrared LEDs. These track infrared light with a wavelength of 850 nanometers. It is quite obvious that this spectrum range is outside of humans' visible light spectrum.

Thanks to its wide angle lenses, the device has a large interaction space of eight cubic feet, which takes the shape of an inverted pyramid – the intersection of the binocular cameras' fields of view.

The Leap Motion Controller's viewing range is limited to roughly 1 meter above the controller, by 1 meter wide on each side (150° angle), by 1 meter deep on each side (120° angle) (see Figure 19). This range is limited by LED light propagation through space, since it becomes much harder to infer a hand's position in 3D beyond a certain distance. LED light intensity is

ultimately limited by the maximum current that can be drawn over the USB 3.0 micro-B connector.

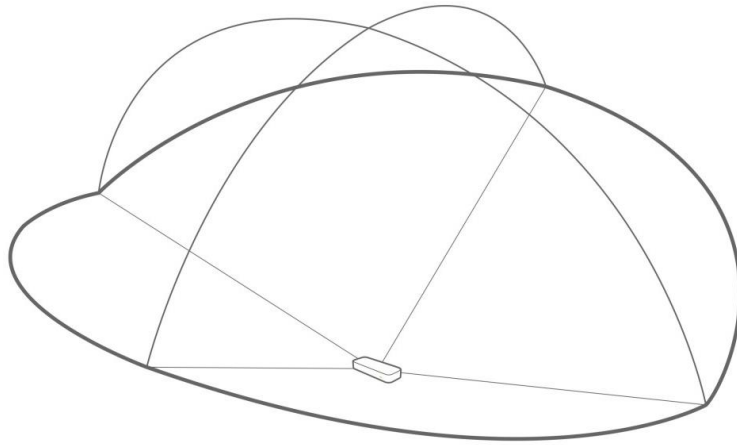


Figure 19 - Interaction area of the Leap sensor

Afterwards, the device's USB controller reads the sensor data into its own local memory and performs any necessary resolution adjustments. This data is then streamed via USB to the Leap Motion tracking software.

The data takes the form of a grayscale stereo image of the near-infrared light spectrum, separated into the left and right cameras. The only observable objects are those directly illuminated by the Leap Motion Controller's LEDs. However, incandescent light bulbs, halogens, and daylight will also light up the scene in infrared.

Software overview

Once the data of received image is streamed to the host computer, it's time for some heavy mathematical lifting. The Leap Motion Controller doesn't generate a depth map – instead it applies advanced algorithms to the raw sensor data.

The Leap Motion Service is the software on the host computer that processes the images. After compensating for background objects (such as heads) and ambient environmental lighting, the images are analyzed to reconstruct a 3D representation of what the device captures.

Next, the tracking layer matches the data to extract tracking information such as fingers and tools or anything that could be used as an extension of the hand in the size of a pen that could be used to point at something, just like a finger. The tracking algorithms used, interpret the

3D data and infer the positions of occluded objects. Filtering techniques are applied to ensure smooth temporal coherence of the data. The Leap Motion Service then feeds the results – expressed as a series of frames, or snapshots, containing all of the tracking data – into a transport protocol.

Through this protocol, the service communicates with the Leap Motion Control Panel, as well as native and web client libraries, through a local socket connection (TCP for native, Web Socket for web). The client library organizes the data into an object-oriented API structure, manages frame history, and provides helper functions and classes.

From there, the application logic ties into the Leap Motion input, allowing a motion-controlled interactive experience.

As shown in Figure 20 a hand with a tool is tracked. The spatial accuracy of a finger like object crossing within the tracking area, is about 0,01 mm (see). Because the picture is a bit misleading, the Leap device is below the pencil or pen shown in the following figure, with which the user "writes" and is located at a distance from the screen without touching it.



Figure 20 - Leap's smaller observation area and high resolution

The smaller observation area and with a higher resolution of the device, it is differentiated from the Kinect device, which is most suitable for monitoring and recognizing the whole body in a space, in the size of an area like a living room. In a demonstration, the Leap was

shown to perform tasks such as navigating a Web page by using pinch-to-zoom gestures on high-precision design maps, and manipulation of complex 3D data visualizations.

Public use

The Leap Motion Company has distributed thousands of free devices to the developers who are interested in creating applications for the device. The Leap Motion sensor was released to public in July 2013 [54], priced around 90 \$ USD.

One of the positive elements that came along with the device is the Airspace [55], electronic application store. This is a very big advantage for developers in terms of quick finding applications, but also for customers, so to ensure the acquisition of appropriate and compatible with Leap applications, to be sure. This movement from the Leap Motion, is an integral part of the company's strategy, since without a software using this unique special device, it is unlikely that someone will "stick" with it in the long run.

A reference to other multimodal devices

As mentioned in the introduction chapter there exist other complex devices too that combine modalities through different sensors and some of them provide this through a single device, except from these presented in the previous chapter.

Xtion Pro LIVE by Asus

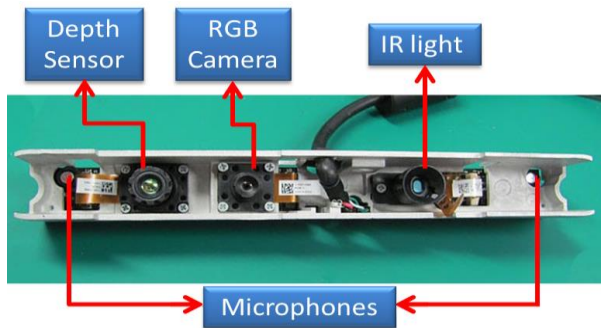


Figure 21 - Xtion Pro LIVE hardware

In late spring of 2011, the first depth sensing device of Asus was launched, the Asus Wavi Xtion Pro [56] (see Figure 22). Targeting purely the developers, it contained only the depth sensor of Primesense Ltd. Unfortunately this device hadn't the expected popularity and after a while Asus stopped its production and released an update of the device that combined both a depth sensor, an RGB camera and

microphones (see Figure 21). One big advantage over the Kinect of Microsoft was that the connection with a computer was only via a USB port without needing external power supply. The hardware of the Xtion pro LIVE sensor consists of one depth sensor, IR light transmitter, and RGB camera and two microphones (see Figure 23).



Figure 22 – Xtion Pro



Figure 23 – Xtion Pro LIVE

MYO by Thalmic

About two years ago, an unknown startup by the name of Thalmic Labs started to reveal its upcoming product, Myo (see Figure 24). Using a technique known as electromyography, this little armband was able to read electrical signals from the muscles in the user's forearm, map them to gestures made with the hand, and control other devices with those gestures.

This was (and still is) a radical departure from the way gesture commands are typically registered. Most gesture-control systems, like Microsoft's Kinect, still detect movements with cameras, which can be thrown off by poor lighting conditions, distance, and simple obstructions. By drawing gesture information directly from the user's arm muscles instead of a camera, Myo circumvents all these problems and also works with devices that don't have a camera in the first place.

To read the muscle activity in your forearm, Myo wraps it in eight different blocks, each of which contains a medical-grade EMG sensor. The armband also uses a three-axis gyroscope, three-axis accelerometer, and three-axis magnetometer to sense motion in any direction.

Muscle activity and motion readings are handled by an onboard ARM Cortex M4 Processor, which communicates with your devices via Bluetooth. For devices that don't have Bluetooth functionality built in, Myo also comes with a Bluetooth dongle that plugs into any USB port.



Figure 24 - MYO armband

Oculus Rift by Oculus VR.

The Rift Development Kit 1 (see Figure 25) uses a 7-inch (18 cm) screen with a significantly lower pixel switching time than the original prototype, reducing latency and motion blur when turning one's head quickly. The LCD is quite bright and the color depth is 24 bits per pixel. The field of view (FOV) is more than 90 degrees horizontal (110 degrees diagonal).

The resolution is 1280×800 (16:10 aspect ratio), which leads to an effective of 640×800 per eye (4:5 aspect ratio). However, since the Rift does not feature a 100% overlap between the eyes, the combined horizontal resolution is effectively greater than 640. The image for each eye is shown in the panel as a barrel distorted image that is then corrected by pincushion effect created by lenses in the headset, generating a spherical-mapped image for each eye.



Figure 25 - Oculus Rift DK1

In March 2014, Oculus announced the upcoming Development Kit 2 (DK2) (see Figure 26). This is a small refinement of the crystal cove prototype, featuring several key improvements over the first development kit, such as having a higher-resolution (960×1080 per eye) low-persistence OLED display, higher refresh rate, positional tracking, a detachable cable, and the omission of the need for the external control box.



Figure 26 - Oculus Rift DK2

Google Glass by Google X

Google Glass is a project started by Google that is intended to bring hands-free display technology to the general public. By utilizing voice commands, users can interact with their Google Glass device to get information from their phones, participate in Google+ Hangouts or to get information from the internet. With a wireless data connection, Google Glass adds an augmented-reality overlay to whatever you're looking at, automatically bringing up relevant information from various Google sources.

The user can wear the Glass as any pair of glasses, but it features an optical heads-up display that projects a display in front of their eyes. Interaction is by means of voice commands, accelerometer and gyroscope and a thin strip known as the “Touchpad” that runs alongside the right hand side of the frame.



Figure 27 - Google Glasses

Plugins & Extensions

In the early days of the World Wide Web, the first versions of HTML couldn't deliver fancy content like videos. Text, images, and links were pretty much the limit. Plug-ins were invented to work around the previous limitations of HTML and deliver more interactive content.

Browser plugins and extension development is the actual creation of a software for a specific browser because each browser type has its own architecture and APIs to build the extensions which requires different code and skills.

Nevertheless nowadays there are development frameworks which allows developers to build cross-browser extensions with only one code base and one API, eliminating the need to develop a different extension version for each one of the Browsers. Examples of those frameworks are:

- Crossrider development framework [57]
- Extension Maker [58]
- Kango cross-browser extension framework [59]
- Crossbrowser Free [60]

Plugins

As mentioned above the plug-in is an additional piece of software that specializes in processing particular types of content for example, users may download and install a plug-in like the widely used Adobe Flash Player [61] to view a web page which contains a video or an interactive game, or Java from Oracle in order to be able see java applets [62].The plug-in model we could say that defines a distinct space on the web page for the plug-in, then steps aside. The plug-in is free to operate inside that space, independent of the browser.

This independence means that a particular plug-in can work across many different browsers. However, that ubiquity also makes plugins prime targets for browser security attacks if they aren't up to date, lacking from the latest security fixes.

The plug-in model we use today is largely the one inherited from the web's early days, but the web community is now looking at new ways to modernize plugins in order their content

to be searchable, linkable, and can interact with the rest of the web page. More importantly, the collaboration of some browser vendors and plug-in providers protect users from security risks. For example, the Google Chrome and Adobe Flash Player teams have worked together to integrate Flash Player into the browser and with Chrome's auto-update mechanism it is ensured that the Flash Player plug-in is never out-of-date and always receives the latest security fixes and patches.

Many people confuse plugins and extensions in the browser world. They sound like they should be the same thing, but in fact they are very different.

A plugin is, quite simply, a third party library that “plugs in” to the browser that can be embedded inside a web page using an `<embed>` tag or a `<object>` tag. Many allow you to “talk” to them through JavaScript, though that's not required. **More importantly, a plugin affects only a specific page in which it is placed.**

More examples of common plugins include:

- Adobe (Macromedia) Flash [63]
- Microsoft Silverlight [64]
- Apple QuickTime [65]
- Adobe Reader [66]

Extensions

When browser extensions were first introduced, developers often had to build them in unusual programming languages or in heavy-duty mainstream languages like C++ hence this took a lot of work, time and expertise. More importantly while adding more code to the browser could have as a result security concerns, as it gave attackers more chances to exploit the browser. In addition to this issue, because of the bad coding sometimes the extensions were notorious for causing browser crashes, too.

Nowadays, most of the extensions are written in user friendly programming languages of the web like HTML, JavaScript and CSS as most web applications and pages they interact with. They're faster and easier to build, safer, and get better and better right along with the web standards they're built upon.

Extensions let the users add new features extending the capabilities of the browser, meaning customization with the most important browser features of each user, depending on their needs like allow the users to block advertisements, download videos from websites, integrate the browser with social networks and even add features from other browsers.

Another example of extension is the installation of a currency converter that shows up as a new button next to the browser's address bar and by clicking the button and it converts all the prices of the current web page into any currency the users specify. Extensions like this, let the users apply the same kind of functionality to every web page they visit.

Browser extensions can also act on their own, outside of web pages. An example of this scenario is an email notifier extension that exists on the browser toolbar and quietly checks for new messages in the users' email account and let them know when one arrives. In this case, the extension is always working in the background regardless what web page the users are looking at and without logging in to their email account in a separate tab or window.

Just as in the plugins, in the extensions filed the security issues exist too. While Web pages are constrained by the security model of the Web, extensions are not. As a result, a browser extension may not behave as described, and take action against the interest of the users that installed it. Such browser extensions are a form of Malware. For example some software downloads come with unwanted bundled programs that install browser extensions without a user's knowledge, while making it hard for the users to uninstall the add-on. However the crucial security difference of both plugins and extensions is that, extensions add features the user can use, while plug-ins add features websites can use.

Blended Learning

Over the years, pedagogical methods have evolved and consistently improved, compared to the educational systems of the past. A significant factor of this progress is unquestionably the increasing use of technology in teaching. Nowadays, most educators prefer to blend traditional teaching with interactive software, in order to achieve the maximum involvement of their students and to consolidate their learning.

“Blended learning” is a term which has yet to be universally defined. A sufficient description could be the following: “Blended learning is the process of using established teaching methods merged with Internet and multimedia material, with the participation of both the teacher and the students.”[6]

Still, there are a few matters in question regarding the process of creating blended learning environments [67] :

- Firstly, there is the issue of the importance of face to face interaction. Several learners have stated that they are more comfortable with the part of live communication in merged teaching methods, considering it more effective than the multimedia based part. Others are of the exact opposite opinion, which is that face to face instruction is actually not as required for learning, as it is for socialization. There are also those who believe that both live interaction and online or software material are of the same significance and it is the learner’s decision which of the two is more suitable for their educational needs.
- Secondly, there is the question of whether the learners opt for combined learning exclusively because of the flexibility and accessibility of the method, without taking into account if they are choosing the appropriate type of blended teaching. Also, there are doubts regarding the ability of the learners to organize their own learning without the support of an educator.
- Another matter, is the need for the instructors to dedicate a large amount of time to guide the learners and to equip them with the necessary skills in order to achieve their goals. In addition, the instructors need to continue being educated themselves to be able to meet the requirements of blended teaching.

- There is also the argument that schools which have integrated technology into the curriculum are mostly addressed and beneficial to people in a comparatively favorable position in terms of economic or social circumstances. This, however, is refuted by the fact that blended teaching methods are quite affordable and easily accessible. The quick distribution of the multimedia material used in this type of instructing, is considered an advantage, but the universality of the system raises the need to modify the provided material, in order to make it more culturally appropriate for each audience.
- Lastly, a continuous effort is being made to proceed to the new directions given by the novelties of technology while maintaining a low-priced production of educational material. The uninterrupted evolution of communication and information technology is making this effort rather strenuous for the developers of such teaching models.

Despite the difficulties faced when designing blended learning techniques, the advantages of combining face to face instructing with technology are many. Some of enormous importance [68] are:

1. The learner is intrigued by the procedure itself and as a result, the material being taught seems more interesting to them. This leads to the easier accomplishment of the educational goals that the teacher has set.
2. There is no limitation regarding the time or the place of the lesson. A student can attend remote classes being offered online, frequently having the opportunity to record and watch again the lesson.
3. The cost of the teaching process is greatly reduced. Every school unit that uses blended learning, has access to a variety of Internet and software material which would require a lot of time and effort to be independently produced, resulting in additional expenses.
4. In industrial applications of blended learning, it has been observed that the desired results have been achieved twice as fast as with the established instruction methods.

Regardless of how little official research has been done on blended learning, there are Universities that provide substantial information on the matter [68]. Stanford University is an institution with much experience in individually oriented programs for charismatic people. An issue which troubled the University was that a great percentage of their students (a bit less than half) would not finish the programs. They concluded that the fault was in the way the students were being taught, compared with the way they wished they would be taught. Thus, they decided

to include live eLearning in their programs, leading to a spectacular rise in the completion percentage, which then reached 94%. The factors that were credited to affect the scholars in such a positive manner were: (a) the scheduled live interaction, which provided inspiration and incentive to finish the assignment without delay, and (b) the access to communication with numerous teachers and fellow students, which gave a chance to experience superior forms of instructing and learning.

The example of Stanford University proves that joining customized educational material with live eLearning interaction could facilitate both the participation in the program and the completion of it. These findings, can motivate other institutions and companies to invest in self-paced teaching systems combined with multimedia resources.

However, blended learning should not be conceived solely as a method of enriching the class with technology or making the learning process more accessible and engaging. Its main purpose is to modify and adjust the teaching and learning interaction in order to upgrade it. It assists and enables the growth of critical thinking, creativity and cognitive flexibility [69].

Chapter 3 - Architectural Design & Description

C.O.A.L.S. Framework

Human-computer interaction can be represented in the form of a perception-action cycle, e.g. Norman's human action cycle [76]:

- Humans *perceive* the world with their senses,
- *Interpret* these perceptions,
- *Evaluate* those interpretations and
- Decide on some kind of *goal*, based on their evaluation of those interpretations.

This goal is transformed into an intention to act, this *intention* itself is then translated into a sequence of actions, which in turn is *executed*. From Norman's human action cycle and Nigay's

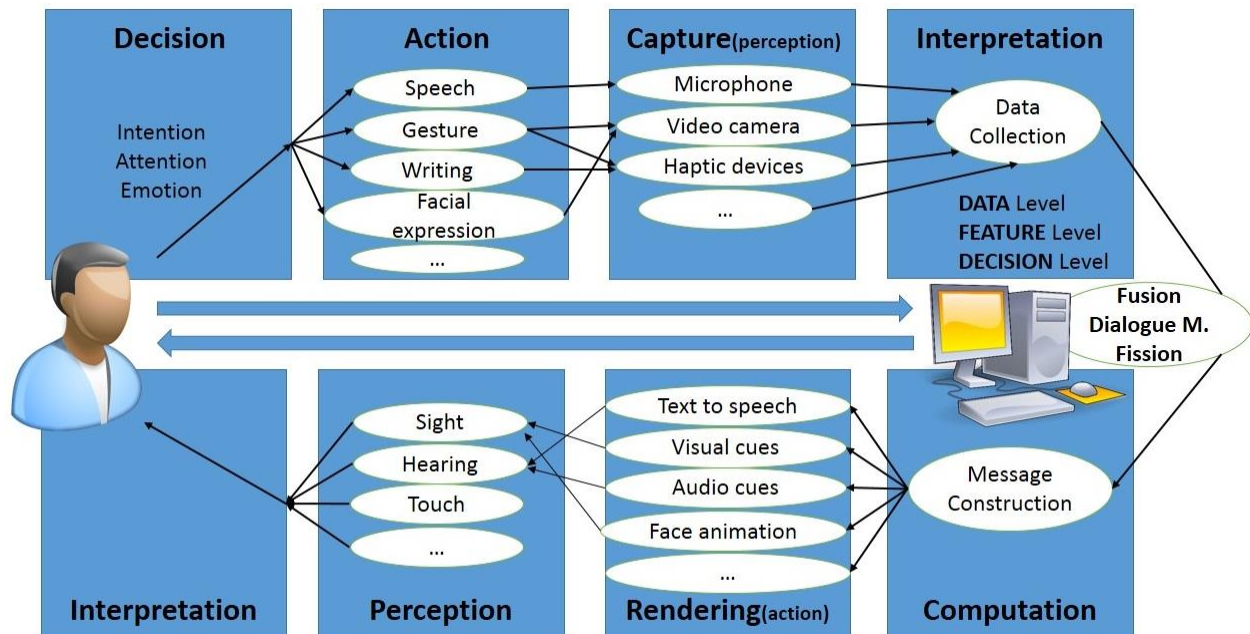


Figure 28 - Multimodal human-machine interaction loop

Pipe-Lines model [77] (a system side extension of Norman's cycle), the model of the multimodal human-machine interaction loop has been drawn [78] Figure 28.

Thus following the architectural design of multimodal systems, meaning the utilization of a Fusion engine, a Dialogue Manager and a Fission engine, the received data can be manipulated for the fulfillment of a purpose. Nevertheless the need of an intermediate framework is necessary in order to collect, reorganize and transform the user input into meaningful data, which will then be transferred into the parts of the architecture that were previously mentioned. Figure 29, demonstrates the general architecture of a multimodal system from the software perspective.

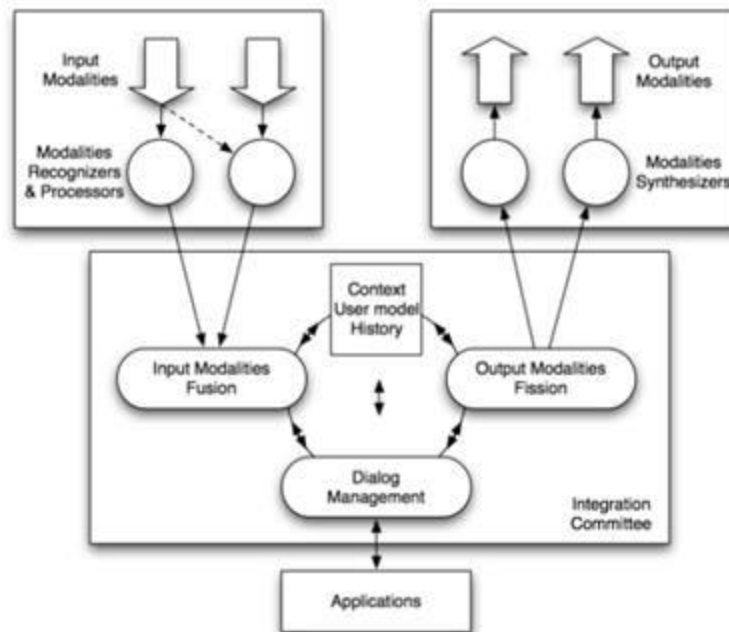


Figure 29 - Multimodal system architectural design

According to the above design, the upper left box includes the input modalities, modality recognizers and processors. The input modalities part, is practically the devices through which the system can receive the input from the users and is divided into

- Visual,
- Audio,
- Haptic and
- Motion modalities.

In Figure 30, these modalities are separated along with the devices that can be produced from. The modality recognizers and the processors are the software components that handle (recognize and process) the received data from the previous step.

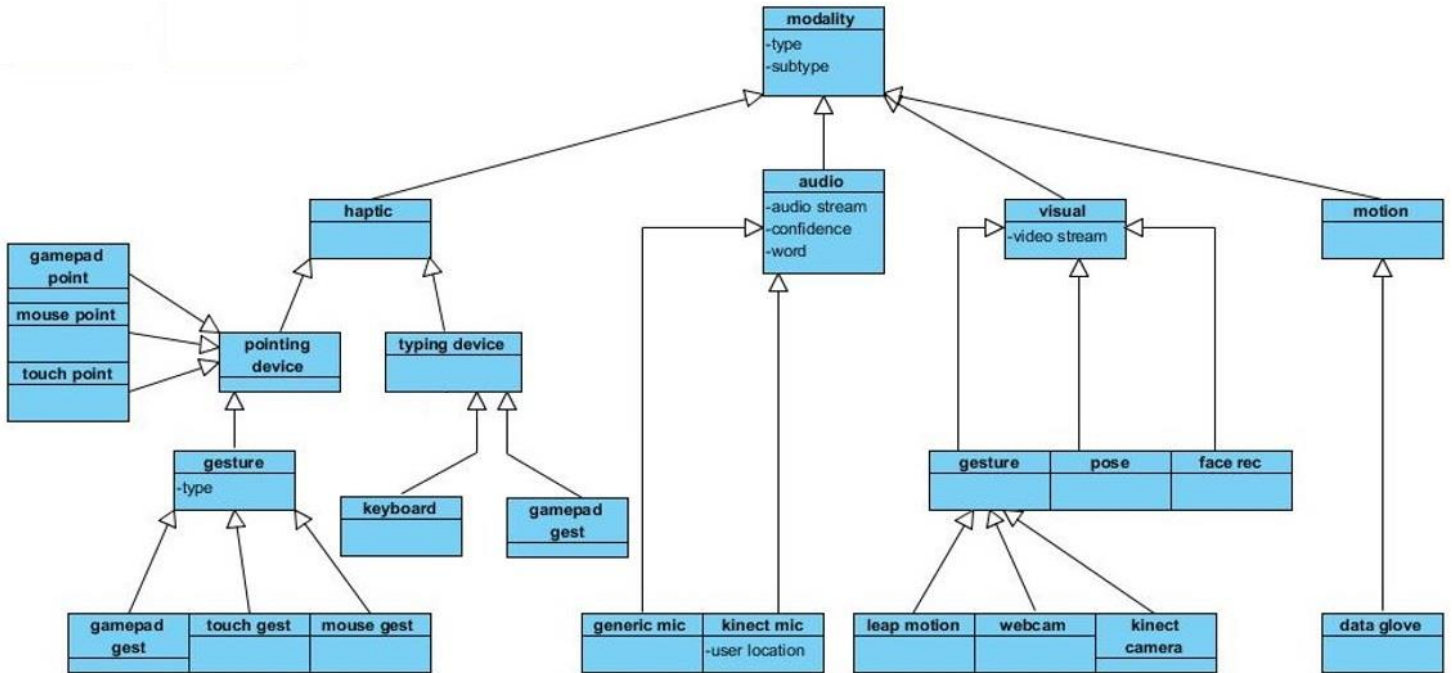


Figure 30 - Modalities & Devices

In addition with the recognition there is a tokenization part, for example:

Command / Word	Recognizers	Extra info
Put	Speech	-
This	Speech / Gesture	Type of Point[x, y, z]
There	Speech / Gesture	Type of Point[x, y, z]

Table 3 - Tokenization of the recognized data

After the previous steps, the part of the integration committee (bottom box of the Figure 29), is responsible for handling this multimodal input, integrate it under a common context and

communicate with both the application and the user. More precisely the parts of the integration committee are:

The Fusion engine:

- In *C.O.A.L.S. builder* each token takes a value and is packed in a “proto frame”, which is a markup file. More precisely these values - states are:
 - Command,
 - Object,
 - Attribute,
 - Location and
 - Selection

These values depend on the C.O.A.L.S vocabulary. This means, that with more semantic values, the tokens can be “described” better, even though the amount of the specific values are sufficient for the token description included in a simple sentence.

- The *C.O.A.L.S. state machine* then, receives any proto frame created from the previous step aiming to create a complete frame. The completion of the frame is achieved when there is a combination of sufficient tokens that their state, lead to the end of a sentence (see Figure 31). The minimum amount of the tokens inside the frame is least one with “Command” state, while the maximum amount is yet undefined because the size of the vocabulary could be larger. In case of arrival of another token with “Command” state, before the completion of the frame, the framework “resets” the procedure and restarts the creation of the frame.

Word	C.O.A.L.S. Vocabulary
Put	Command
This	Selection
[x, y, z]	Point
There	Location

[x, y, z]	Point
-----------	-------

Table 4 - Complete C.O.A.L.S. frame example

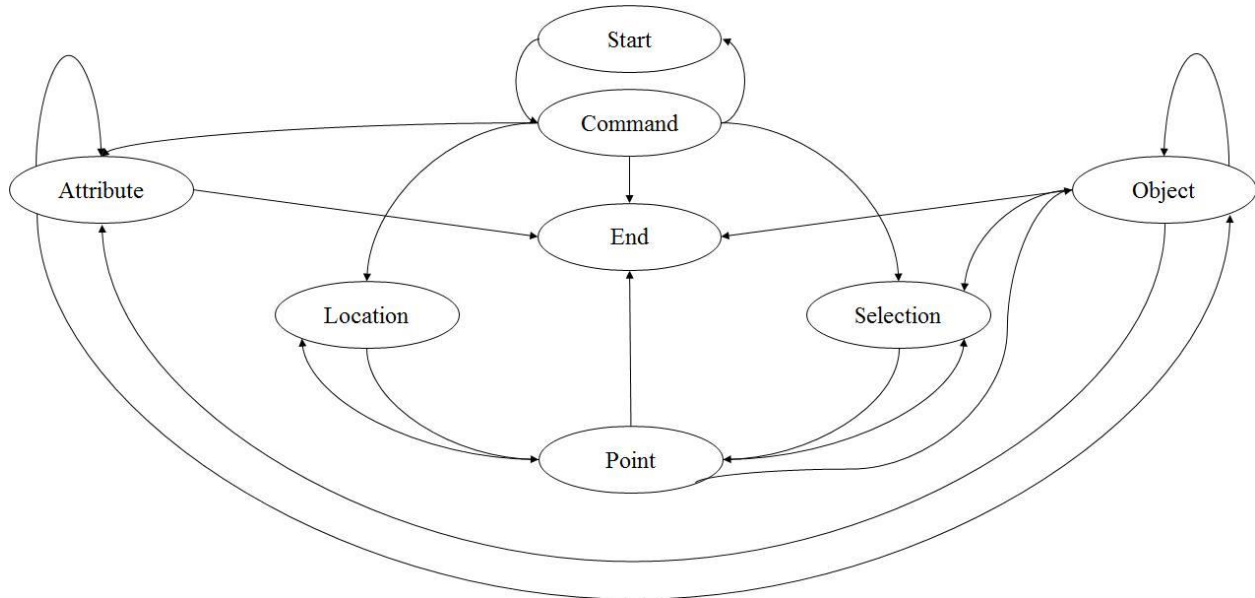


Figure 31 - State transactions of the C.O.A.L.S. frame

1. Dialogue Manager:

The dialogue manager is responsible for the creation of the “output frame”. Inside this component, there are:

- The sentence which is stored inside the complete frame. Using the application-actions mapping scheme, it is “translated” into applications’ functions.
- The appropriate message which after the subsequent implementation of the user’s actions, should inform them, passing through the fission engine.

2. Fission Engine:

This “output frame” holds information about the mode (audio, visual, touch, etc.) which occurs from the **Context Manager** and the data that will be delivered to the user, as feedback, after completing a task.

- **Message construction**

Gathering of the messages according to the user's actions, generated in the previous step.

- **Modality selection**

Each message can be delivered to the user with various ways. To choose the most adequate modality or a combination of modalities for the message delivery, a stable matching algorithm is used along with the information of the Context manager module.

```

"mode": "visual",
"modalities":
[
  {
    "name": "image",
    "candidates": [
      "image",
      "speech",
      "text"
    ]
  },
  {
    "name": "animation",
    "candidates": [
      "animation",
      "image",
      "speech",
      "text"
    ]
  },
  {
    "name": "text",
    "candidates": [
      "text",
      "speech"
    ]
  }
]

"mode": "audio",
"modalities":
[
  {
    "name": "sound",
    "candidates": [
      "sound",
      "text"
    ]
  },
  {
    "name": "speech",
    "candidates": [
      "speech",
      "text"
    ]
  }
]

"mode": "tactile",
"modalities":
[
  {
    "name": "vibration",
    "candidates": [
      "vibration",
      "speech",
      "sound"
    ]
  }
]

```

Figure 32 –The modalities that can deliver these three basic modes.

- **Coordination of the feedback**

For this final step, modalities are time synchronized, in order to demonstrate the feedback to the user in a natural manner. More precisely the concrete content of the message is chosen and subsequently the modality attributes are established. For a more coherent and synchronized result, all used channels must be coordinated.

Google Chrome Extensions

As mentioned in the previous sections (see Chapter 2 Plugins & Extensions) extensions allow the users to add functionality to an Internet browser without diving deeply into native code. The developers can create new extensions with core technologies that they are already familiar with from web development such as HTML, CSS, and JavaScript. More precisely, the aim of this chapter is to describe the way to create an extension in Google Chrome Browser, to interact with the YouTube player with the use of multimodal devices like the Leap and Kinect, in order to accomplish our case study.

Each extension has the following files:

- A manifest file
- One or more HTML files
- One or more JavaScript files
- Any other files the extension needs (image files)

Hence the very first thing needed to be created is a manifest file named *manifest.json*. This manifest is nothing more than a metadata file in JSON format that contains properties like the extension's name, the description, the version number and so on. More precisely, it will be used to inform Chrome about what the extension is going to do, what permissions it requires in order to do those things and what the most important files that it will use are. The manifest file used in this approach is shown in Figure 33

Manifest file Resources

More specifically the properties of the manifest file contain:

- ***“Manifest Version”***:
Extensions are simply bundles of resources, wrapped up with a *manifest.json* file that describes the package's contents. The format of this file is generally stable, but occasionally changes must be made to address important issues. The developers should specify which version of the manifest specification their package targets, by declaring a *manifest_version* key in their manifests. As

mentioned in the developers' page of Chrome [70] "*Consider manifest version 1 deprecated as of Chrome 18. Version 2 is not yet required, but we will, at some point in the not-too-distant future, stop supporting packages using deprecated manifest versions.*"

- **“Name”:**

Self-explanatory field (set extension's name).

- **“Description”:**

Self-explanatory field (set extension's description).

- **“Browser Actions”:**

- **Icon:**

To set the icon, the *default_icon* field of *browser_action* in the manifest is used, or the *browserAction.setIcon* method is called inside a JavaScript code snippet.

Browser action icons can be up to 19 dips (device-independent pixels) wide and high, but larger icons are resized to fit.

The icon can be set in two ways: using a static image or using the HTML5 canvas element. It is more common to use static images, but the canvas element is used in order to create more dynamic UIs.

Static images can be in any format WebKit [71] can display, including file formats like: BMP, GIF, ICO, JPEG, or PNG.

- **Popup:**

The popup appears when the user clicks the icon of the extension.

The popup can contain any HTML page and it's automatically sized to fit its contents. The logic of rendering the content of the popup is implemented by *popup.js* called inside the html code.

The HTML file is specified in the *default_popup* field of *browser_action* in the manifest, or called from the *browserAction.setPopup* method inside a JavaScript code snippet.

- **“Permissions”:**

The *chrome.permissions* API requests declared permissions at run time rather than install time, in order to give the chance to the users to understand why the

permissions are needed and grant only those that are necessary if they are optional.

Furthermore, permissions help limit the damage if the extension is compromised by a malware.

In this manifest file the required permissions are:

- **Tabs:**
chrome.tabs API is used to interact with the browser's tab system, like creating, modifying, and rearranging tabs in the browser. The use of most *chrome.tabs* methods and events can be made without declaring any permissions in the extension's manifest file. However, to access the url, title, or favIconUrl properties of *tabs.Tab*, the declaration of the "tabs" permission in the manifest is necessary.
- **Active Tab:**
Without *activeTab*, this extension would need to request full and persistent access to every web site. This is a lot of power to entrust to such a simple extension and if it is ever compromised, the attacker gets access to everything it had.
By contrast, an extension with the *activeTab* permission only obtains access to a tab following to an explicit user gesture. If the extension is compromised, the attacker will need to wait for the user to invoke the extension before obtaining access, while that access will only last until the tab is navigated or closed.
- **“Content Scripts”:**
Content scripts are JavaScript files that run in the context of web pages. By using the standard **Document Object Model (DOM)**, they can read details of the web pages the browser visits, or make changes to them.
Content scripts can indirectly use the *chrome.** APIs, get access to extension data, and request extension actions by exchanging *messages* with their parent extension.

- **Matches:**

Content script matching is based on a set of URLs defined by match patterns. A match pattern is essentially a URL that begins with a permitted scheme (http, https, file, or ftp, and that can contain '*' characters. The special pattern `<all_urls>` matches any URL that starts with a permitted scheme. Each match pattern has 3 parts:

- ***scheme*** — for example, http or file or *
- ***host*** — for example, www.google.com or *.google.com or *; if the scheme is file, there is no host part
- ***path*** — for example, /*, /foo*, or /foo/bar. The path must be present in a host permission, but is always treated as /*.

- **Js:**

Even though this field is optional, it is very important because this is the list of JavaScript files that will be injected into the matching pages and they will be injected in the order they appear in this array.

- **"Background":**

The background page is an HTML page that runs in the extension process. It exists as long as the extension does and only one instance of it at a time is active with the only of exception when the extension uses *incognito* [72] "split" mode, in which case a second instance is created for incognito windows. To specify which is the appropriate HTML file in the background page, the *page* property field is used (not in our case).

In a typical extension with a background page, the UI is implemented by dumb views. When the view needs a state, it requests the state from the background page.

When the background page notices a state change, the background page commands the views to update.

- **Scripts:**

This is simply the script that provides all actions in the background page.

- **Persistent:**

The persistent key is set to false in order to convert our background page to an event page that will be loaded when it is “needed” and unloaded when it goes idle again. Once it has been loaded, the event page will stay running as long as it is active.

- **“Options Page”:**

To allow users to customize the behavior of the extension, most of the times an options page is provided. If it is, a link will be provided from the extensions management page at *chrome://extensions*. Clicking the Options link opens a new tab redirecting to the options page. In our case, from the options page the user can see raw data retrieved from the multimodal device. As in the previous fields this page is an HTML page controlled by a JavaScript file.

```

{
  "manifest_version": 2,

  "name": "Youtube player interaction with Leap Motion",
  "description": "This extension allows the use of Leap Motion
controller to interact with Youtube player.",
  "version": "1.0.0",

  "browser_action": {
    "default_icon": "Ok-icon.png",
    "default_popup": "popup.html"
  },

  "permissions": [
    "tabs",
    "activeTab"
  ],

  "content_scripts": [
    {
      "matches": ["https://www.youtube.com/*"],
      "js": [
        "js/leap.min.js",
        "js/jquery-1.9.0.min.js",
        "js/myscript.js"
      ]
    }
  ],

  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },

  "options_page": "options.html"
}

```

Figure 33 - Manifest in JSON format

Extension's Architecture Overview

Many extensions have a **background page**, in many cases an invisible page that holds the main logic of the extension. In order to achieve the interaction between the web pages and the extension, the extension must use a content script. The Figure 34 shows the place of a browser action and a page action extension into the browser. Both extensions have background

pages, which are defined by the *background.html* file and have JavaScript code that controls their behavior in both windows.

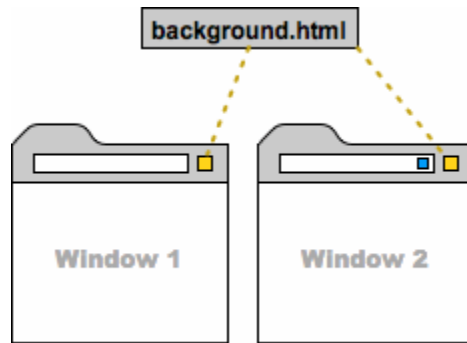


Figure 34 - Background.html

Extensions can contain ordinary HTML pages that display the extension's UI when the icon of the browser action is clicked. Usually, an extension can have an options page, which lets users customize how the extension works. The HTML pages inside an extension have complete access to each other's DOMs, and they can invoke functions on each other. The following figure shows the architecture of a browser action's popup.

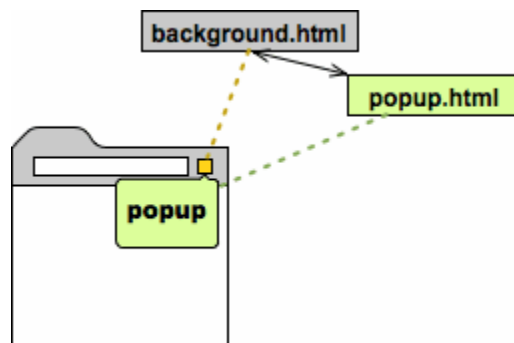


Figure 35 - Popup.html

If the extension needs to interact with web pages, then it needs a content script. A **content script** is a JavaScript file that is executed in the context of a page that has been loaded into the browser and it's not part of the extension it was packaged with, meaning its parent extension. Content scripts can read details of the web pages the browser visits, and they

can make changes to the pages. In the following figure, the content script can read and modify the DOM for the displayed web page. It cannot, however, modify the DOM of its parent extension's background page. In our case the actions and changes to the visited page are achieved through programmatic injection in order to get the needed data. For security reasons the results of this method are limited and the best way to surpass this problem is to use Google APIs, specifically for Chrome and YouTube.

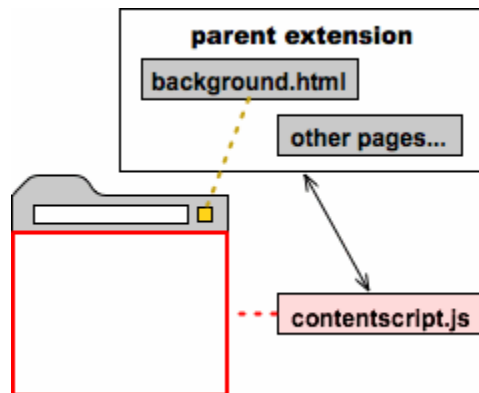


Figure 36 - Content script interaction

Chapter 4 - Approach of the Implementation

This chapter presents in detail the approach of the implementation created for the purpose of this thesis. More precisely, the creation of two internet browser extensions in order to control the pages that are visited from the user, with the use of multimodal devices like the Leap of Leap Motion and the Kinect of Microsoft. The decision to choose a specific browser for the creation of the extensions, was taken in order to avoid performance issues and support differences that internet browsers have among each other. Google Chrome is the choice because is one of the most used internet browser while supporting edge technologies like HTML5 and providing a JavaScript API to help the development of extensions and applications based on it.

Loading the extension in Chrome

There are a few ways to access the extension page in Google Chrome browser like:

- Typing to the address bar the exact link: “*chrome://extensions/*”.
- Finding the extensions from the “Customize and control Chrome” from the upper right corner of the browser (see Figure 37).

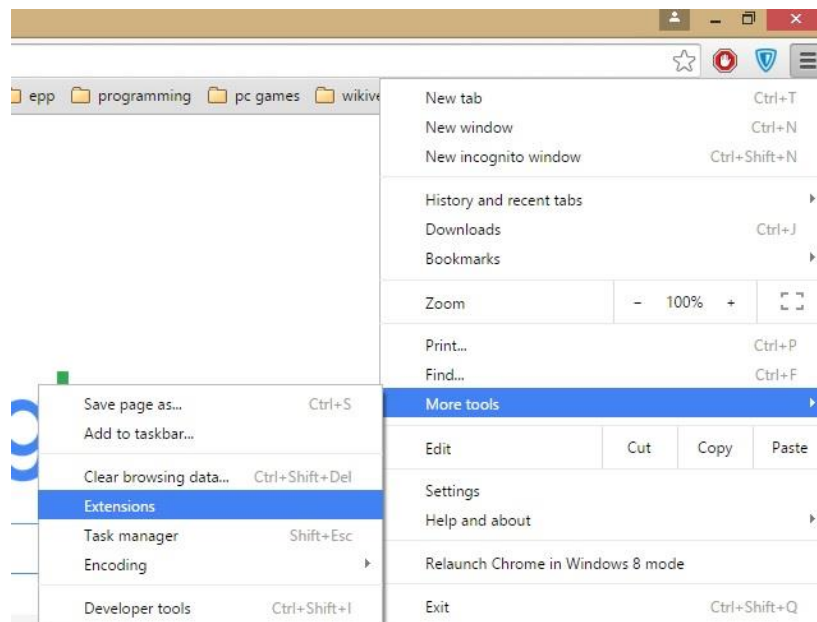


Figure 37 - Extensions in Chrome's options

After navigating to the extensions page, the user must download the extension that desires either from Google Chrome Web Store (<https://chrome.google.com/webstore/category/extensions?hl=en-US>) or loading their own from a specific directory of their PC, by clicking the button “Load unpacked extension...” from the upper left corner of the extensions’ page. To achieve the second way, the “Developer Mode” option must be checked as seen in Figure 38.

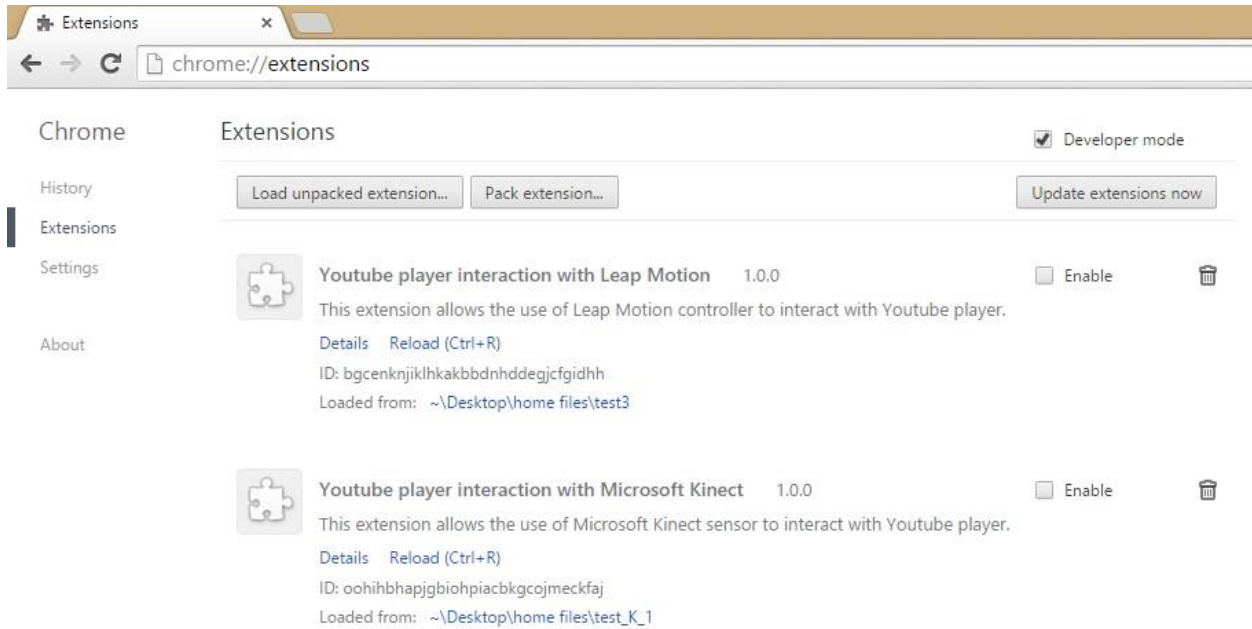


Figure 38 - Extensions manager in Chrome

After the installation of the extensions, the user must check the “enable” button in order to activate them. The activated extension appears in the top right corner of the browser with a slight difference. As mentioned in the previous chapter if the extension is refers to browser action it appears outside of the address bar and if it refers to a page action it is placed inside the address bar.



Figure 39 - Page action icon



Figure 40 - Browser action icon

YouTube player Interaction with Leap Motion

Leap motion device as mentioned in previous chapter is a sensor with great features and use, in many fields of computer science. More precisely, its small size combined with the high resolution hand/tool tracking, give to the user the opportunity to control the PC from closer distance rather than using the Kinect sensor. For this reason the created extension, handles more detailed DOMs which from a bigger distance would be harder to interact with.

Server side - Web Sockets Communication

```
var ws;
// Create the socket with event handlers
function connectToWebSocket() {
    // Create and open the socket
    ws = new WebSocket("ws://localhost:6437/v4.json");
    // On successful connection
    ws.onopen = function(event) {
        var enableMessage = JSON.stringify({enableGestures: true});
        ws.send(enableMessage); // Enable gestures
        var backgroundMessage = JSON.stringify({background: true});
        ws.send(backgroundMessage); // Get frames in background
        console.log("open");
    };
    // On message received
    ws.onmessage = function(event) {
        var obj = JSON.parse(event.data);
        var str = JSON.stringify(obj, undefined, 2);
        if(obj.id){
            console.log("Frame data for " + obj.id);
        } else {
            console.log("message " + event.data);
        }
    };
    // On socket close
    ws.onclose = function(event) {
        ws = null;
        console.log("close");
    }
    // On socket error
    ws.onerror = function(event) {
        console.log("error");
    };
}
connectToWebSocket();
```

Figure 41 - Web Sockets Communication

The Leap Motion controller provides tracking data through a Web Socket server. The Web Socket server listens to port 6347 on the localhost domain (<http://127.0.0.1:6437>). Any client application, including Web clients that can make a Web Socket connection can access the Leap Motion tracking data in the form of JSON-formatted messages. From the above code snippet (see Figure 41) it is presented the way the Web Socket connection is established which is included inside the `“leap.min.js”`.

Client side - Myscript.js walkthrough

From the client side, this is the main file/script that handles the logic of the extension. In this section will be presented the actions made through the user input along with screenshots in order to be more understandable. From figure, inside the `manifest.json`, this is the last script called after the `“leap.min.js”` as mentioned in the previous subsection and `“jquery-1.9.0.min.js”`, as order matters inside the array of `“content scripts”` → `“js”`.

The script begins with the initialization of some variables as usual, with the most important the `“controllerOptions”` variable which is used as argument later in the `“Leap.loop”` function, in order the tracking of the gestures to be enabled.

```
var controllerOptions = {enableGestures: true};
```

Afterwards the *function* `“init()”` is called to check for two things:

- Sometimes even after successfully establishing the connection, issues may still exist and the connection can be lost, resulting the Leap to be restarted and establish a new connection.

From the API provided by Google Chrome a message is sent to the `background.js` and if the connection is lost, the badge text of the extension is updated, as well the user is prompted to restart the device and refresh the visited page (YouTube in our case) as seen in Figure 42.

```

function check_connection()
{
    now = new Date().getTime() / 1000;
    if(now - last_poll > connection_lost_after)
    {
        clearInterval(connection);

        try {
            chrome.runtime.sendMessage({ connection: 'lost' },
            function(response) {
                console.error('Connection to Leap Motion Lost. Restart Leap
                Motion and Refresh Page.');
```

```

                $('body').append('<div
                class="leap_motion_connection" style="display:
                none;"></div>');

                $('.leap_motion_connection').html('<b>ATTENTION:</b>
                Connection to Leap Motion Lost. Restart Leap Motion and
                Refresh Page.').css({
                    'position': 'fixed',
                    'top': '0',
                    'left': '0',
                    'width': '100%',
                    'color': '#222',
                    'text-align': 'center',
                    'height': '30px',
                    'z-index': '1000',
                    'line-height': '30px',
                    'background-color': '#9AC847'
                }).fadeIn('slow');
```

```

                $('.leap_motion_connection').click(function() {
                $(this).fadeOut('slow'); });
            });
        }
        catch(error) {
            console.error(error.message);
        }
    }
}

```

Figure 42 - Check Connection function

- The second thing to check is, if the current tab has focus and only then, run this extension on the active tab. Once again through the Chrome API a message is sent to the *background.js* in order to receive the information needed like the active tab and the page's URL.

The other task that is made through this function, is to update the current page after the user reloads the extension, because an error is produced, which is fixed by this page refresh (see Figure 43).

```
function check_focus()
{
    try {
        chrome.runtime.sendMessage({ tab_status: 'current' },
function(response) {
            if(response.active && window.location.href ==
response.url && document.hasFocus())
            {
                tab_has_focus = true;
            }
            else
            {
                tab_has_focus = false;
            }
        });
    }
    catch(error) {
        if(error.message.indexOf
('Error connecting to extension') !== -1)
        {
            document.location.reload(true);
        }
        else
        {
            console.error(error.message);
        }
    }
}
```

Figure 43 - Check focused Tab function

Leap API

“Leap” is the global namespace inside the Leap API. In addition to the classes in the API, the Leap namespace contains the function of the “Leap.loop(options, callback)”.

The loop() function sets up the Leap controller and Web Socket connection and invokes the specified callback function on a regular update interval. Moreover, the developer can create their own Controller() object.

Each frame of data from the Web Socket server contains JSON defining a frame. The attributes of a frame in the JSON include information about the collected data like the examples below:

```
"pointables": array of Pointable objects

  "bases": the 3 basis vectors for each bone, in index order,
            wrist to tip, (array of vectors).
  "btipPosition": the position of the tip of the distal phalanx
                  as an array of 3 floats.
  "carpPosition": the position of the base of metacarpal bone as
                  an array of 3 floats.
  "dipPosition": the position of the base of the distal phalanx
                  as an array of 3 floats.
  "direction": array of floats (vector)
  "extended": boolean (true or false)
  "handId": integer
  .
  .
  .
```

Figure 44 - Pointables information retrieved from Leap API

```
"hands": array of Hand objects
  "armBasis": the 3 basis vectors of the arm (array of vectors)
  "armWidth": float
  "confidence": float
  "direction": array of floats (vector)
  "elbow": array of floats (vector)
  "grabStrength": float
  "id": integer
  "palmNormal": array of floats (vector)
  .
  .
  .
```

Figure 45 - Hands information retrieved from Leap API

```

"pointables": array of Pointable objects
  "bases": the 3 basis vectors for each bone, in index order, wrist
           to tip, (array of vectors).
"btipPosition": the position of the tip of the distal phalanx as an
                array of 3 floats.
"carpPosition": the position of the base of metacarpal bone as an
                array of 3 floats.
"dipPosition": the position of the base of the distal phalanx as an
                array of 3 floats.
"direction": array of floats (vector)
"extended": boolean (true or false)
"handId": integer
.
.
.

```

Figure 46 – Pointables information retrieved by the Leap API

Therefore while the information of the retrieved data is received, it can also be manipulated as seen fit. The most important variables that are used to fulfil our case scenario, are the hands, the fingers and the pointables of the user that are tracked as well the gestures produced by them. An example of the code snippet is seen below in Figure 47.

```

for (var i = 0; i < frame.hands.length; i++) {
  var hand = frame.hands[i];

  // IDs of pointables associated with this hand
  if (hand.pointables.length > 0) {
    var fingerIds = [];
    for (var j = 0; j < hand.pointables.length; j++) {
      var pointable = hand.pointables[j];
      fingerIds.push(pointable.id);
    }
    if (fingerIds.length > 0) {
      handString += "Fingers IDs: " + fingerIds.join(", ") + "<br
/>";
    }
  }
}
else {
  handString += "No hands";
}

```

Figure 47 – Hands, Pointables and Fingers tracked


```

if (frame.gestures.length > 0) {
  for (var i = 0; i < frame.gestures.length; i++) {
    var gesture = frame.gestures[i];
    switch (gesture.type) {
      case "circle":
        gestureString += "center: " + vectorToString(gesture.center)
          + " mm, \n"
          + "normal: " + vectorToString(gesture.normal, 2) +
            ", \n"
          + "radius: " + gesture.radius.toFixed(1) + " mm,
            \n"
          + "progress: " + gesture.progress.toFixed(2) + "
            rotations \n\n";
        break;
      case "swipe":
        gestureString +=
          "start position: " + vectorToString(gesture.startPosition) + "
            mm, \n"
          + "current position: " +
            vectorToString(gesture.position) + " mm, \n"
          + "direction: " + vectorToString(gesture.direction, 1) + ", \n"
          + "speed: " + gesture.speed.toFixed(1) + " mm/s \n\n";
        break;
      case "screenTap":
        console.log("screenTap");
        break;
      case "keyTap":
        gestureString += "position: " +
          vectorToString(gesture.position) + " mm \n\n";
        break;
      default:
        gestureString += "unkown gesture type \n\n";
    }
  }
}
else {
  gestureString += "No gestures \n\n";
}

```

Figure 48 - Gestures provided by the Leap API

Interaction with YouTube page

This particular extension has any interaction effect only with the YouTube page as mentioned before and noted inside the “*manifest.json*” file. More precisely inside the part of the code that handles the detected gestures, there also can be found the actions that will be triggered accordingly. An example of the action triggered by the circle gesture is shown below.

```

case "circle":
    //get the video DOM
    var x = document.getElementsByClassName("video-stream html5-
main-video")[0];
    //
    // check if the circle gesture is made clockwise or not
    //
    var clockwise = false;
    var pointableID = gesture.pointableIds[0];
    var direction = frame.pointable(pointableID).direction;
    var dotProduct = Leap.vec3.dot(direction, gesture.normal);

    if(gesture.state == "stop"){
        if (dotProduct > 0){
            clockwise = true;
            //increase volume
            if( x.volume < 0.8){
                x.volume = x.volume + 0.1;
            }
        }
        else{
            //decrease volume
            if(x.volume > 0.1){
                x.volume = x.volume - 0.1;
            }
        }
    }
    break;

```

Figure 49 - Event triggered from circle gesture

To be more specific, at this point will be presented the actions that the user can make to interact with the YouTube page after generating a gesture:

- **“Circle Gesture”**:
When the circle gesture is tracked, the video element of the DOM (visited page) is retrieved. If the circle gesture was made clockwise or not then the volume of the player is increased or decreased.
- **“Swipe Gesture”**:
When the swipe gesture is tracked, the “moveNextElement()” function is called (see Figure 50).
With this function the child nodes of the controls player’s panel are retrieved and with each swipe gesture the focus moves to the next child element.

For more precision, the direction of the swipe gesture is checked and the action is made accordingly to that too.

```
//  
// Set focus to next element of the control list of Youtube  
//  
function moveNextElement() {  
  
    var controls = document.getElementsByClassName("ytp-chrome-  
controls")[0];  
    var sizeOfControls = controls.childNodes.length;  
  
    if(controlsCNT < sizeOfControls){  
        focusedKey = controls.childNodes[controlsCNT];  
        focusedKey.focus();  
        console.log("focused element " + focusedKey.className);  
        controlsCNT++;  
    }  
    else{  
        controlsCNT = 0;  
    }  
}
```

Figure 50 - Get child elements and set them focused

- **“Key Tap Gesture”:**
Based on the swipe gesture, if a button is focused, when the key tap gesture is tracked, then this button is clicked and provides the same action as it was clicked with the mouse.
- **“Screen Tap Gesture”:**
Upon performing the screen tap gesture, the generated action is to quickly change the size of the video without to focus its button with the swipe gesture and click it with the key tap gesture.

Every action produced happens by injecting code inside the page of YouTube. Code injection has its limits though, for example the key press of the fullscreen button is not allowed. Hence especially in very secure internet sites like the YouTube, in order to fully interact with the DOM it is preferred to take advantage of its API provided by Google.

Popup and Options Page

As mentioned in chapter 3, when the user clicks on the icon of the extension, a popup window appears, which in our case provides a link that redirects the user to the options page (see Figure 51). This isn't exactly a page that gives the user the opportunity to modify any option, however they can preview an aggregation of the retrieved data in detail (see Figure 52).

Furthermore they can interrupt the input stream by displaying the values of the variables as provided by the API, for the given instance. Additionally to that, the user has the opportunity to pause once again the flow of data, when a gesture is tracked by the sensor. Apparently after any of these actions, the user can resume the procedure and keep track of the new values.

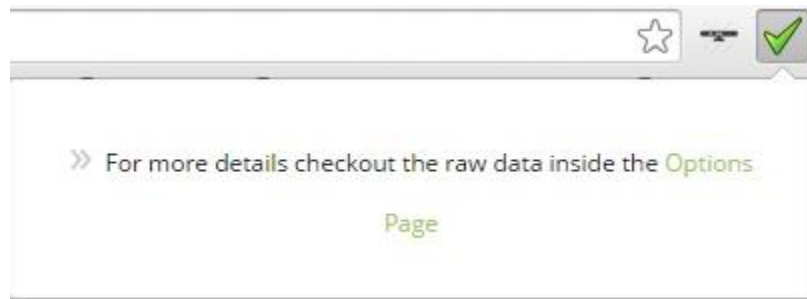


Figure 51 - Popup Html page

Leap Motion Data

Pause on gesture

Frame data:

Frame ID: 3308
Timestamp: 20634297463 μ s
Hands: 1
Fingers: 5
Tools: 0
 Gestures: 2
 Translation: (4.1, -2.8, -0.7) mm
 Rotation axis: (0.10, -0.18, 0.98)
 Rotation angle: 0.09 radians
 Scale factor: 0.99

Hand data:

Hand ID: 1
Type: left hand
Direction: (0.35, 0.12, -0.93)
Palm position: (-76.3, 204.3, 32.5) mm
Grab strength: 0
Pinch strength: 0.0141069
Confidence: 0.530812
Arm direction: (0.5, 0.8, -0.4)
Arm center: (-158.5, 113.6, 117.7)
Arm up vector: (-0.3, 0.6, 0.8)
Translation: (4.1, -2.8, -0.7) mm
Rotation axis: (0.1, -0.2, 1.0)
Rotation angle: 0.09 radians
Scale factor: 0.99
Fingers IDs: 10, 11, 12, 13, 14

Finger and tool data:

Pointable ID: 10
Type: Thumb
Belongs to hand with ID: 1
Classified as a finger
Length: 44.7 mm
Width: 17.4 mm
Direction: (0.40, -0.15, -0.90)
Extended?: false
Metacarpal bone
Center: (-76.9, 191.9, 83.7)
Direction: (0.7, 0.1, -0.7)
Up vector: (0.6, 0.4, 0.7)
Proximal phalanx bone
Center: (-59.5, 192.9, 71.4)
Direction: (0.8, 0.0, -0.6)
Up vector: (0.5, 0.4, 0.8)
Intermediate phalanx bone
Center: (-36.2, 191.8, 45.7)
Direction: (0.4, -0.2, -0.9)
Up vector: (0.9, 0.3, 0.3)
Distal phalanx bone
Center: (-29.9, 187.0, 22.8)
Direction: (0.0, -0.3, -1.0)
Up vector: (1.0, 0.2, -0.0)
Tip position: (-29.2, 185.0, 18.1) mm

Pointable ID: 11
Type: Index finger
Belongs to hand with ID: 1
Classified as a finger
Length: 50.4 mm
Width: 16.6 mm
Direction: (0.23, -0.58, -0.78)
Extended?: true
Metacarpal bone
Center: (-64.4, 208.9, 47.7)
Direction: (0.5, -0.0, -0.9)
Up vector: (0.1, 1.0, 0.0)
Proximal phalanx bone
Center: (-43.7, 200.4, 4.2)
Direction: (0.3, -0.4, -0.9)
Up vector: (0.2, 0.9, -0.4)
Intermediate phalanx bone
Center: (-36.3, 187.1, -20.1)
Direction: (0.2, -0.6, -0.8)
Up vector: (0.2, 0.8, -0.5)
Distal phalanx bone
Center: (-32.6, 176.0, -33.3)
Direction: (0.2, -0.7, -0.7)
Up vector: (0.3, 0.7, -0.6)
Tip position: (-31.8, 175.1, -36.6) mm

Pointable ID: 12
Type: Middle finger
Belongs to hand with ID: 1
Classified as a finger
Length: 57.5 mm
Width: 16.3 mm
Direction: (-0.23, -0.57, 0.79)
Extended?: false
Metacarpal bone
Center: (-77.6, 211.3, 41.6)
Direction: (0.4, -0.0, -0.9)
Up vector: (-0.1, 1.0, -0.1)
Proximal phalanx bone
Center: (-63.3, 190.2, 11.7)
Direction: (0.1, -1.0, -0.1)
Up vector: (0.3, 0.2, -0.9)
Intermediate phalanx bone
Center: (-63.1, 162.9, 19.0)
Direction: (-0.2, -0.6, 0.8)
Up vector: (0.3, -0.8, -0.5)
Distal phalanx bone
Center: (-68.6, 154.9, 36.1)
Direction: (-0.3, -0.1, 0.9)
Up vector: (0.1, -1.0, -0.1)
Tip position: (-70.4, 155.2, 40.9) mm

Pointable ID: 13
Type: Ring finger
Belongs to hand with ID: 1
Classified as a finger
Length: 55.3 mm
Width: 15.5 mm
Direction: (-0.13, -0.53, 0.84)
Extended?: false
Metacarpal bone
Center: (-92.1, 211.2, 38.6)
Direction: (0.2, -0.0, -1.0)
Up vector: (-0.2, 1.0, -0.1)
Proximal phalanx bone
Center: (-82.0, 192.0, 10.8)
Direction: (0.2, -1.0, -0.1)
Up vector: (0.2, 0.1, -1.0)
Intermediate phalanx bone
Center: (-79.8, 167.0, 19.1)
Direction: (-0.1, -0.5, 0.8)
Up vector: (0.3, -0.8, -0.5)
Distal phalanx bone
Center: (-83.2, 160.0, 36.9)
Direction: (-0.2, -0.1, 1.0)
Up vector: (0.2, -1.0, -0.0)
Tip position: (-84.7, 161.4, 41.7) mm

Pointable ID: 14
Type: Pinky finger
Belongs to hand with ID: 1
Classified as a finger
Length: 43.3 mm
Width: 13.8 mm
Direction: (0.23, -0.52, 0.82)
Extended?: false
Metacarpal bone
Center: (-106.2, 206.3, 37.5)
Direction: (0.1, 0.0, -1.0)
Up vector: (-0.3, 0.9, -0.0)
Proximal phalanx bone
Center: (-99.0, 192.7, 10.8)
Direction: (0.3, -0.9, -0.1)
Up vector: (-0.1, 0.1, -1.0)
Intermediate phalanx bone
Center: (-92.3, 174.1, 15.7)
Direction: (0.2, -0.5, 0.8)
Up vector: (0.2, -0.8, -0.6)
Distal phalanx bone
Center: (-89.7, 168.9, 30.0)
Direction: (0.1, -0.1, 1.0)
Up vector: (0.3, -0.9, -0.1)
Tip position: (-89.4, 168.8, 33.4) mm

Gesture data:

Gesture ID: 18, type: circle, state: start, hand IDs: 1, pointable IDs: 11, duration: 0 μ s, center: (-20.6, 216.4, -33.5) mm, normal: (-0.35, -0.14, 0.93), radius: 39.8 mm, progress: 0.79 rotations
Gesture ID: 18, type: circle, state: update, hand IDs: 1, pointable IDs: 11, duration: 8719 μ s, center: (-21.1, 216.0, -33.4) mm, normal: (-0.34, -0.13, 0.93), radius: 39.9 mm, progress: 0.82 rotations

Figure 52 - Extension's Options Page

Browser Interaction with Microsoft Kinect

Microsoft Corporation released Kinect sensor that was one of the first single devices that could combine multiple sources of system input like sound, video and image from RGB camera and from Depth camera. Due to the minimum distance from the sensor (near 800mm) and in order to have a variety of actions, different from these of the Leap Motion sensor, the actions that will be presented in this section concern not only the visited pages but also the browser itself. Apparently, the same architectural design and restrictions apply to this case too, since all operations are made through an extension of Chrome.

This project consists of two sub-projects:

- A server-side WPF application written in C#, which uses Kinect SDK and
- A client-side web page displaying the skeleton joints on an HTML5 canvas (popup.html) while performing gestures actions in another html page (visited page).

Server side – Communication & Data Initialization

The server application's job is straightforward: detect the users' joints, pack the data and send them to the client using Web Sockets. The Web Socket server listens to port 8181 or on the localhost domain (//localhost:8181) as seen in the code snippet of Figure 53.

- ***Web Sockets – Initialization of the connection:***

```

private static void InitializeConnection()
{
    var server = new WebSocketServer("ws://localhost:8181");

    server.Start(socket =>
    {
        socket.OnOpen = () =>
        {
            _clients.Add(socket);
        };

        socket.OnClose = () =>
        {
            _clients.Remove(socket);
        };

        socket.OnMessage = message =>
        {
            switch (message)
            {
                // ....
            }
        };
    });
}

```

Figure 53 - Web Socket Connection

- **Device and Data stream initialization:**

In Figure 54, the code starts with the initialization of the device with the one that is found first. In the case that a sensor is available the script continues with the initialization of the data streams like:

- The Color stream
- The Depth stream
- The Skeleton stream and
- The event listeners

```

private static void InitilizeKinect ()
{
    var sensor =
    KinectSensor.KinectSensors.SingleOrDefault ();

    if (sensor != null)
    {
        sensor.ColorStream.Enable ();
        sensor.DepthStream.Enable ();
        sensor.SkeletonStream.Enable ();

        sensor.AllFramesReady += Sensor_AllFramesReady;

        _coordinateMapper = sensor.CoordinateMapper;

        sensor.Start ();
    }
}

```

Figure 54 - Device & Data stream initialization

Without diving inside the code, in each data stream class that is enabled as previously seen, there is a Serialize function through which data is packed into a **Binary Large Object (Blob)**. These blobs, in our case are images, audio or multimedia objects, which are sent to the client side through the web sockets.

However the part of the code that is responsible to store the skeleton data, inside the Open skeleton frame function, is not a blob but a JSON file with a number of properties that can be manipulated by the client as seen fit (see Figure 55).


```

static void Sensor_AllFramesReady(object sender,
    AllFramesReadyEventArgs e){
    using (var frame = e.OpenColorImageFrame()){
        //...
        var blob = frame.Serialize();
        foreach (var socket in _clients){
            socket.Send(blob);
        }
    }
    using (var frame = e.OpenDepthImageFrame()){
        //...
        var blob = frame.Serialize();
        foreach (var socket in _clients){
            socket.Send(blob);
        }
    }
    using (var frame = e.OpenSkeletonFrame()){
        //...
        if (users.Count > 0){
            string json =
users.Serialize(_coordinateMapper, _mode);
            foreach (var socket in _clients){
                socket.Send(json);
            }
        }
    }
}
}
}
}

```

Figure 55 - Server sends Blob & Json to the Client

Client side – Myscript.js and Popup.js walkthrough

In the client's side, JavaScript is used in order to receive the server's data so the user can manipulate the visited page, as well as having a visual representation of their skeleton that is drawn in an html canvas in the "*popup.html*".

Popup.html

Upon loading the extension, its icon appears in the upper right corner of the browser. When the user clicks on the icon, a popup window shows up, informing them with a "Status" of connection, meaning if it was successfully or not established via Web Sockets between them and the server. Below this, an HTML canvas exists, previewing a static image, until the connection is established. So when the server sends the new data the static image is replaced by the skeleton of the user.

Myscript_k.js & popup.js

The common think between those two scripts, is the Web Socket connection check and the parsing of the JSON file sent from the server with the user's skeleton information. Except these, the rest of the scripts and their job, are different.

With the “*popup.js*” the received data from the server, using “*socket.onMessage()*” function, can be displayed on the canvas as mentioned before, as long as a user is tracked.

```
// SKELETON DATA
// Get the data in JSON format.
var jsonObject = JSON.parse(event.data);

//clear the context of the canvas before drawing
context.clearRect(0, 0, canvas.width, canvas.height);

///
/// Display the skeleton joints.
///
for (var i = 0; i < jsonObject.skeletons.length; i++) {
    for (var j = 0; j <
jsonObject.skeletons[i].joints.length; j++) {
        var joint = jsonObject.skeletons[i].joints[j];
        ///
        /// Draw
        ///
        context.beginPath();
        context.arc(joint.x, joint.y, 10, 0, Math.PI * 2,
true);
        context.closePath();
        context.fill();
    }
}
```

Figure 56 - Drawing the user's skeleton on canvas

From the other hand, the “*myscript_k.js*” is responsible for the recognition of the hand gestures, generated by the user, as well as the actions that are triggered accordingly. The flow of the script code is as follows:

1. Checking if the current tab has focus and only then run the extension.(Figure 43)
2. Parsing of the JSON file to get skeleton info.

```
// Get the data in JSON format.  
var jsonObject = JSON.parse(event.data);
```

Figure 57 - Parsing retrieved Json data

3. Manipulating the received data and store what is needed:

a. The skeleton's Id

```
///  
///  
///  
for (var i = 0; i < jsonObject.skeletons.length; i++) {  
    for (var j = 0; j < jsonObject.skeletons[i].id.length; j++) {  
        var jointID = jsonObject.skeletons[i].id[j];  
        skeletonIdArray.push(jointID);  
    }  
}
```

Figure 58 - Getting skeleton's Id

b. The name of the tracked skeleton joints.

```
///  
///  
///  
for (var i = 0; i < jsonObject.skeletons.length; i++) {  
    for (var j = 0; j < jsonObject.skeletons[i].joints.length; j++){  
        var jointName = jsonObject.skeletons[i].joints[j].name;  
        jointNameArray.push(jointName);  
    }  
}
```

Figure 59 - Get skeleton joints' names

c. Specify the skeleton joints needed in order to create the hand gesture check while converting them into 3D Vectors objects.

```

for (var i = 0; i < jsonObject.skeletons.length; i++) {
  for (var j = 0; j < jsonObject.skeletons[i].joints.length; j++) {
    switch (jsonObject.skeletons[i].joints[j].name) {
      case "wristright":
        var wristRight_Vector3D =
          new Vector(jsonObject.skeletons[i].joints[j].x,
                    jsonObject.skeletons[i].joints[j].y,
                    jsonObject.skeletons[i].joints[j].z);

        break;
      case "wristleft":
        var wristLeft_Vector3D =
          new Vector(jsonObject.skeletons[i].joints[j].x,
                    jsonObject.skeletons[i].joints[j].y,
                    jsonObject.skeletons[i].joints[j].z);

        break;
      //...
      //...
      //...
    }
  }
}

```

Figure 60 - Specify joints to create gestures

d. Check for user's performed hand gestures and trigger their events.

```

///
/// Detect Gestures and trigger actions
///

waveGesture(elbowRight_Vector3D, wristRight_Vector3D,
            wristLeft_Vector3D, head_Vector3D);

setZoomPage(shoulderCenter_Vector3D, wristRight_Vector3D,
            wristLeft_Vector3D, head_Vector3D,
            shoulderRight_Vector3D, shoulderLeft_Vector3D);

```

Figure 61 - Functions called for gesture recognition

Creating the Hand Gestures and the triggered actions

In contrast with the Leap API that provides a number of hand or finger gestures, the API provided for Kinect, along with the SDK 1.8, gives the developer the opportunity to manipulate the retrieved data, with no predefined hand or body gestures. This is the reason why, for our approach, a number of simple hand gestures were created, in order to interact with the Chrome

browser or the visited page. As shown in Figure 61 two function are called to detect the performed hand gestures by the user and trigger some browser actions accordingly.

Wave – Swipe Gesture

This function is responsible to detect whether the user swipes with their right hand to any direction in (x, y) axis or make a “wave” gesture. The arguments needed for the function are four skeleton joints, such as:

- The right elbow,
- The right wrist,
- The left wrist and
- The head.

Since all these points are 3D vectors, the value of their position in (x, y) axis can be stored and manipulated separately. More precisely, to create a swipe gesture, the following checks should be made (see Figure 62):

- the wrist should be above or below the elbow (y axis check),
- the wrist should move either right or left of the elbow (x axis check).

To avoid conflict with other hand gestures, another check must be performed, meaning whether the other hand from the one that produces the swipe gesture, is higher than the head (y axis check). Whenever these checks succeed, then the swipe gesture is complete in each direction.

```
///  
/// wristR above elbowR  
///  
if (joint1_vector.y > joint2_vector.y) {  
    ///  
    /// wristR right of the elbowR  
    ///  
    if (joint1_vector.x < joint2_vector.x) {  
        ///  
        /// left wrist above head  
        ///  
        if(joint3_vector.y < joint4_vector.y) {  
////////////////////////////////////  
//////////////////////////////////// ADD SWIPE ACTION //////////////////////////////////////  
////////////////////////////////////  
        } } }  
} } }
```

Figure 62 - Creation of right swipe gesture

The left swipe gesture, or both gestures performed by the left hand, can be created by following the same technique.

The easiest way to create a wave gesture, is to consider it as a combination of many swipe gestures. For this reason, after the completion of a swipe gesture, a “*wave counter*” variable increases its value by one and when it reaches a specific number, then the wave gesture is completed. In addition to the previous counter check, to avoid gesture conflicts, a timer is also created in order to measure the time between the first performed swipe gesture and the completion of the wave gesture. The logic behind this thought, is that the natural wave gesture is performed quickly enough, separating it from distinct swipe gestures (see Figure 63).

```
// start right swipe gesture
waveCounter++;
/// Start action timer in order to avoid gesture conflict
if (start_action == 0) {
    start_action = new Date().getTime();
}
//.....
// start left swipe gesture
waveCounter++;
//.....
if (waveCounter == 4) {
    var end_action = new Date().getTime();
    //convert milliseconds to seconds
    var offset = (end_action - start_action) / 1000;
    waveCounter = 0;
    ///
    /// if the offset lasts less than 2.5 seconds, in order to
    distinguish from swipe gestures
    ///
    if (offset < 2.5) {
        ///////////////////////////////////////////////////////////////////
        /////////////////////////////////////////////////////////////////// ADD WAVE ACTION ///////////////////////////////////////////////////////////////////
        ///////////////////////////////////////////////////////////////////
    }
}}
```

Figure 63 - Creation of wave gesture

The actions triggered by the hand gestures, are achieved through code injection, as mentioned in the previous chapter of Leap motion sensor and are demonstrated in the table below:

Hand Gestures	Triggered Actions
Wave	Reload page
Wrist above elbow	
Right Swipe	Play html5 video
Left Swipe	Pause html5 video
Wrist below elbow	
Right Swipe	Increase video's volume
Left Swipe	Decrease video's volume

Table 5 - Actions triggered by gestures

Zoom In and Zoom Out the visited page

With this function, the user can increase or decrease the zoom level of the visited page's document body (see Figure 64). As arguments four skeleton joints are used:

- Spine,
- Right wrist,
- Left wrist and
- Head

The checks that must succeed are:

- The distance of both wrists from the spine joint, must exceed the 30cm, in order to continue (z axis check).
- Calculate the distance between both wrists.

After calibrating the variable holding the value of this distance then the zoom level of the document's body changes accordingly. In addition to that, this variable is limited to the browser's minimum or maximum zoom level value.

The last gesture inside this script that must be check is, whether both wrists of the user exceed the height of their head (y axis check) and if succeeded then restore the page's default zoom level.

```
spine_wristR_dist = joint1_vector.z - joint2_vector.z;
spine_wristL_dist = joint1_vector.z - joint3_vector.z;
wristR_wristL_dist = joint2_vector.x - joint3_vector.x;

if((spine_wristR_dist > 0.3 ) && (spine_wristL_dist > 0.3) ){
    currentZoom = wristR_wristL_dist - 70;
    if( (currentZoom <= maxPageZoom) || (currentZoom >=
minPageZoom) ){
        document.body.style.zoom = currentZoom.toString() + "%";
        if(currentZoom > maxPageZoom ){
            document.body.style.zoom = maxPageZoom.toString() + "%";
        }
        else if(currentZoom < minPageZoom){
            document.body.style.zoom = minPageZoom.toString() + "%";
        }
    }
}
if( (joint4_vector.y > joint2_vector.y) && (joint4_vector.y >
joint3_vector.y) ){
    document.body.style.zoom = defaultZoom.toString() + "%";}
```

Figure 64 - Zoom in - out gesture and actions triggered

Chapter 5 - Conclusion, Final Thoughts & Future Work

Ambient intelligence or ubiquitous computing is attempting to embed modern technology into education, work and mostly in everyday life. Even though the relationship between man and computer is becoming increasingly complicated and inherently multimodal, many computer science fields try to make this interaction more natural and invisible at the same time. The term multimodal refers to the combination of multiple and different modalities and to the ways that the system responds to the received inputs which originate from human types of communication.

Hence through this dissertation, the beneficial use of an interactive multimodal framework is presented, in order to successfully handle a complicated system with various multimodal inputs, such as voice commands, hand gestures and touch gestures.

More precisely C.O.A.L.S. framework achieves the previously stated goals, firstly by fusing – combining different modalities, secondly by interpreting the previous fusions while extracting semantic meaning from them and thirdly by providing the user the right feedback, according to the preceding procedures.

Furthermore, the final aim of this thesis, is to gather different modalities as input to the system with the use of complex multimodal sensors such as the Microsoft Kinect and the Leap Motion. These sensors collaborate with the internet browser through a client – server based communication model. Even though these devices and their input are part of an interactive cross-platform system, in this thesis it was preferred to present some use case scenarios, through Google Chrome browser extensions. This choice was made because devices other than personal computers and laptops, such as portable devices (tablets and smartphones) cannot connect with the already mentioned sensors.

For future work there are enough thoughts that could be materialized. Regarding the framework, much more descriptions and actions could be integrated inside the vocabulary which is in the fusion engine and the applications – actions table in the dialogue manager, in order to cover a wider field of commands in each application.

Concerning the browser extensions and the multimodal sensors, after using the existing programming code with a specific application, the enrichment of the detected actions made by the users is a good start point to begin from, in order to fulfill more use case scenarios.

Furthermore, cross browser scripting could be a good idea to follow, in order to avoid a single

browser selection. However the implementation is quite difficult along with incompatibilities among the browsers. Moreover in the near future if the connectivity issues between external multimodal sensors and portable devices cease to exist, the use of the browser extensions might be less useful and will probably be avoided, even though their development might continue as a separate application in collaboration with other ones.

References

- [1] Bellik and D. Teil, “Définitions Terminologiques pour la Communication Multimodale,” presented at the Proceedings of interface Hommemachine (IHM), 1992.
- [2] Sears A. and Jacko J., “The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications.”, 2007.
- [3] Moore Christian, "New Community Open". NUI Group Community, 2006.
- [4] Varchalamas Petros, “Development of a Natural User Interface and creation of a Context Information Manager, using Microsoft Kinect.”, 2013.
- [5] Josh Bersin, “The Blended Learning Book: Best Practices, Proven Methodologies and Lessons Learned.”
- [6] Norm Friesen, “Report: Blended Learning.”, 2012
- [7] Tarja Susi, Mikael Johannesson, Per Backlund, “Serious Games – An Overview”, University of Skövde, Sweden, 2007
- [8] C.S. Wasson, “System Analysis, Design, and Development: Concepts, Principles, and Practices”, John Wiley & Sons, Hoboken 2006.
- [9] Karray, Fakhreddine; Alemzadeh, Milad; Saleh, Jamil Abou; Arab, Mo Nours, “Human-Computer Interaction: Overview on State of the Art”, Waterloo, Canada 2008
- [10] S. Oviatt, “Multimodal interfaces”, in J.A. Jacko and A. Sears (eds), The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Application, Lawrence Erlbaum Associates, Mahwah 2003.
- [11] R.A. Bolt, “Put-that-there: voice and gesture at the graphics interface”, Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques, Seattle, Washington, United States, pp 262-270 (1980).
- [12] I. McCowan, D. Gatica-Perez, S. Bengio, G. Lathoud, M. Barnard and D. Zhang, “Automatic analysis of multimodal group actions in meetings”, IEEE Transactions on PAMI, 27(3), pp 305-317 (2005).
- [13] S. Meyer and A. Rakotonirainy, “A Survey of research on context-aware homes”, Australasian Information Security Workshop Conference on ACSW Frontiers, pp 159-168 (2003).
- [14] P. Smith, M. Shah and N.D.V. Lobo, “Determining driver visual attention with one camera”, IEEE Transactions on Intelligent Transportation Systems, 4(4), pp 205-218 (2003).

- [15] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*, MIT Press, Cambridge (2003).
- [16] Y. Arafa and A. Mamdani, "Building multi-modal personal sales agents as interfaces to E-commerce applications", *Proceedings of the 6th International Computer Science Conference on Active Media Technology*, pp 113-133 (2001).
- [17] Y. Kuno, N. Shimada and Y. Shirai, "Look where you're going: a robotic wheelchair based on the integration of human and environmental observations", *IEEE Robotics and Automation*, 10(1), pp 26-34 (2003).
- [18] L. Nigay and J. Coutaz, "Multifeature Systems: The CARE Properties and Their Impact on Software Design," in *Multimedia Interfaces: Research and Applications*, chapter 9, 1997.
- [19] S. Oviatt, "Advances in robust multimodal interface design," *IEEE Computer Graphics and Applications*, vol. 23, no. 5, pp. 62–68, Sep. 2003.
- [20] L. Nigay and J. Coutaz, "A Design Space for Multimodal Systems: Concurrent Processing and Data Fusion," in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, New York, NY, USA, 1993.
- [21] E. Petajan, B. Bischoff, D. Bodoff, and N. M. Brooke, "An Improved Automatic Lipreading System to Enhance Speech Recognition," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 1988.
- [22] C. Sanderson and K. Paliwal, *Information Fusion and Person Verification Using Speech & Face Information*, 2002.
- [23] L. Zhang, J. Sturm, D. Cremers, and D. Lee, "Real-Time Human Motion Tracking using Multiple Depth Cameras," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012.
- [24] T. H. Bui, "Multimodal Dialogue Management - State of the art," Jan-2006.
- [25] P. Cohen, "Dialogue Modeling," *Dialogue Modeling*, 1997.
- [26] D. R. Traum and S. Larsson, "The Information State Approach to Dialogue Management," in *Current and New Directions in Discourse and Dialogue*, J. van Kuppevelt and R. W. Smith, Eds. Springer Netherlands, 2003, pp. 325–353.
- [27] G. E. Churcher, E. S. Atwell, and C. Souter, "Dialogue Management Systems: a Survey and Overview," 1997
- [28] D. G. Novick and K. Ward, "Mutual Beliefs of Multiple Conversants: A Computational Model of Collaboration in Air Traffic Control," in *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C., 1993, pp. 196–201.

- [29] N. Reithinger, J. Alexandersson, T. Becker, A. Blocher, R. Engel, M. Löckelt, J. Müller, N. Pflieger, P. Poller, M. Streit, and V. Tschernomas, “Smartkom - adaptive and flexible multimodal access to multiple applications,” in IN PROC. OF THE 5TH INT. CONF. ON MULTIMODAL INTERFACES, 2003, pp. 101–108
- [30] N. Reithinger, D. Fedeler, A. Kumar, C. Lauer, E. Pecourt, and L. Romary, “Miamm — A Multimodal Dialogue System Using Haptics,” in Advances in Natural Multimodal Dialogue Systems, J. C. J. van Kuppevelt, L. Dybkjær, and N. O. Bernsen, Eds. Springer Netherlands, 2005, pp. 307–332
- [31] “Cortana”, Microsoft, “<http://windows.microsoft.com/en-us/windows-10/getstarted-what-is-cortana>”.
- [32] “Siri”, Apple, “<http://www.apple.com/ios/siri>”.
- [33] S. Oviatt, “The Human-computer Interaction Handbook,” J. A. Jacko and A. Sears, Eds. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 2003, pp. 286–304
- [34] M. Fasciano and G. Lapalme, “Intentions in the Coordinated Generation of Graphics and Text from Tabular Data,” Knowledge and Information Systems, vol. 2, no. 3, pp. 310–339, Aug. 2000.
- [35] C. Duarte, “Design and Evaluation of Adaptive Multimodal Systems,” PhD Thesis, Universidade de Lisboa, 2007.
- [36] J. Bateman, T. Kamps, J. Kleinz, and K. Reichenberger, “Towards Constructive Text, Diagram, and Layout Generation for Information Presentation,” Computational Linguistics, vol. 27, no. 3, pp. 409–449, Sep. 2001.
- [37] M. Fasciano and G. Lapalme, “Intentions in the Coordinated”.
- [38] Y. Han and I. Zukerman, “A Mechanism for Multimodal Presentation Planning Based on Agent Cooperation and Negotiation,” Hum.-Comput. Interact., vol. 12, no. 1, pp. 187–226, Mar. 1997.
- [39] Microsoft Kinect Official WebPage, “www.microsoft.com/en-us/kinectforwindows”
- [40] PrimeSense Ltd, “The PrimeSensor(TM) Reference Design 1.08.”
- [41] IR Camera Sensor SwissRanger 4000,
“<http://www.adept.net.au/cameras/Mesa/SR4000.shtml>”
- [42] Kinect Fusion, Kinect for Windows, (SDK 1.7,1.8), “<https://msdn.microsoft.com/en-us/library/dn188670.aspx>”
- [43] Depth Sensor patent, ZALEVSKY, Z. et al.,
“<http://patentscope.wipo.int/search/en/detail.jsf?docId=WO2007043036&recNum=1&maxRec=&office=&prevFilter=&sortOption=&queryString=&tab=PCT+Biblio>”

- [44] K. Khoshelham. Accuracy Analysis of Kinect Depth Data, (2011).
- [45] Z. S. A. M. A. e. a. Zalevsky, «Method and System for Object Reconstruction». USA 2006
- [46] MICROSOFT KINECT SDK, ”<http://www.microsoft.com/en-us/kinectforwindows/develop/developer-downloads.aspx>”
- [47] OpenNI Nite , “<http://openni.org/>”
- [48] Open Kinect Project, “http://openkinect.org/wiki/Main_Page”
- [49] OPENKINECT Main Page,” http://openkinect.org/wiki/Main_Page”
- [50] CL NUI Platform. Code Laboratories, “<http://codelaboratories.com/kb/nui>”
- [51] EVOLUCE SDK, “<http://www.evolute.com/en/software/sdk-for-kinect.php>”
- [52] Leap Motion - The Leap, “<https://www.leapmotion.com/>”
- [53] Leap motion sensor. How does it work?, “<http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work>”
- [54] Etherington, Darrell, "Leap Motion Controller Ship Date Delayed Until July 22, Due To A Need For A Larger, Longer Beta Test", 2013
- [55] Airspace download center, “<https://www.leapmotion.com/apps>”
- [56] Asus - Xtion Pro Official WebPage, “http://www.asus.com/Multimedia/Xtion_PRO/”
- [57] Crossrider framework, “<http://crossrider.com/developers>”
- [58] Extension Maker, “<http://extensionmaker.com>”
- [59] Kango framework, “<http://kangoextensions.com/about.html>”
- [60] Crossbrowser, “<http://crossbrowser.com>”
- [61] Adobe Flash Player, “<http://www.adobe.com>”
- [62] Java, “<https://www.oracle.com/java/index.html>”
- [63] Adobe (Macromedia) Flash, “https://en.wikipedia.org/wiki/Adobe_Flash_Player”
- [64] Microsoft Silverlight [“<https://www.microsoft.com/silverlight/what-is-silverlight>”]
- [65] Apple Quicktime [“<http://www.apple.com/quicktime/what-is/>”]
- [66] Adobe Reader [“<https://helpx.adobe.com/reader/faq.html>”]

- [67] Bonk, C. J. & Graham, C. R. (Eds.). (in press). Handbook of blended learning: Global Perspectives, local designs. San Francisco, CA: Pfeiffer Publishing, Chapter 1.1, Blended learning systems : Definition, current trends, and future directions
- [68] A White Paper: Achieving Success with Blended Learning, By Harvi Singh and Chris Reed, Centra Software
- [69] Blended learning: Uncovering its transformative potential in higher education, D. Randy Garrison*, Heather Kanuka Learning Commons, Room 525, Biological Sciences Building, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada, 13 February 2004
- [70] Google Chrome developers page,
“https://developer.chrome.com/extensions/manifest/manifest_version”
- [71] WebKit, “<https://www.webkit.org/>”
- [72] Extension’s Background page in Incognito Mode,
“<https://support.google.com/chrome/answer/95464?hl=en>”
- [73] Cohen, P. R., Johnston, M., McGee, D., Oviatt, S., Pittman, J., Smith, I., Chen, L., Clow, J. 1997. QuickSet: multimodal interaction for distributed applications. In Proceedings of the Fifth ACM international Conference on Multimedia, Seattle, USA, pp. 31-40, (1997).
- [74] Koons, D., Sparrell, C., Thorisson, K. 1993. Integrating simultaneous input from speech, gaze, and hand gestures. In M. Maybury (Ed.), Intelligent Multimedia Interfaces. Cambridge, MA: MIT Press, pp. 257- 276 (1993).
- [75] Mugellini, E., Lalanne, D., Dumas, B., Evequoz, F., Gerardi, S., Le Calvé, A., Boder, A., Ingold, R., Abou Khaled, O. 2009. Memodules as tangible shortcuts to multimedia information. In Denis Lalanne, Jürg Kohlas eds. Human Machine Interaction, LNCS 5440, Springer-Verlag, Berlin/Heidelberg, pp. 103-132 (2009).
- [76] Norman, D.A. 1998. The Design of Everyday Things. New York: Basic Book (1988).
- [77] Nigay, L. 1994. Conception et modélisation logicielles des systèmes interactifs: application aux interfaces multimodales. PhD dissertation.
- [78] Bruno Dumas, “Frameworks, description languages and fusion engines for multimodal interactive systems.”, 2010