MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Gesture based navigation system for Google Maps using Leap Motion controller

LICENSE THESIS

Graduate: **Marius Cristian MĂNĂSTIREANU**

Supervisor: **Lect. Dr. Eng. Adrian GROZA**

2014

DEAN,	HEAD OF DEPARTMENT,
Prof. dr. eng. Liviu MICLEA	**Prof. dr. eng. Rodica POTOLEA**

Graduate: **Marius Cristian MĂNĂSTIREANU**

LICENSE THESIS TITLE

1. **Project proposal:** *The aim of this project is to design and implement a JavaScript library that should be used in combination with Google Maps JavaScript API V3 and the Leap Motion controller in order to provide the end users a better user experience while navigating in Google Maps and Street View by eliminating the interaction with the mouse and keyboard and giving the end users full control and full freedom of movement introducing hand and finger movements in the air space.*

2. **Project contents:** *Introduction, Project Objectives, Bibliographic research, Analysis and theoretical foundation, Detailed design and implementation, Testing and validation, User's manual, Conclusions, Bibliography*

3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**: Lect. Dr. Eng. Adrian Groza

5. **Date of issue of the proposal:** November 1, 2013

6. **Date of delivery:** September 11, 2014

Graduate: ______________________________

Supervisor: ______________________________

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY

OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind
autenticitatea lucrării de licenţă

Subsemnatul Mănăstireanu Marius Cristian legitimat cu CI seria KX nr. 668359 CNP 1910421125784 autorul lucrării "Navigation in Google Maps using Leap Motion controller and Google Maps JavaScript API V3" elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea Computer Science din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Septembrie a anului universitar 2013-2014, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

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

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

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

Data   Nume, Prenume

_______________________   ___________________________________

Semnătura

Table of Contents

Chapter 1. Introduction

In this chapter I will present you the project context of the license thesis together with the motivation for choosing this field of study, followed by a specification of the precise domain of the license thesis.

1.1. Project context

When talking about future in technology often we have a vision, similar with the science fiction movies, of supercomputers controlled by sweeping and swaying the hands in the air. The reality is not that far actually; current trends in technology have pointed towards a world where humans and machines interact in such fashion: screens can now recognize faces and fingerprints, the glasses are not what used to be thanks to Google Glasses and there exists various devices that are able to interpret your body movement and transpose it in the virtual world. At this moment, the latest user interfaces are dominated by touch, but as we can see, things are changing fast, other technologies appear, and the future is pointing us in new directions.

This emerging trend that is mentioned here can be defined as gesture recognition. At this moment there exists applications in this domain which use the hands and fingers movement in order to control the machines; most of the application are present in the consumer electronic devices area, where humans can change the channel or adjust the volume of their smart TVs with just one simple move in the air, or they can swipe between pages of one recipe on their tablet while cooking and their hands are too busy and dirty to touch the device. Another field of interest where this kind of applications started to develop and emerge is the healthcare domain; there exists on the market software for surgeons that use the Microsoft's Kinect controller in order to be able to manipulate digital images needed in the time of the surgeries while maintaining the environment clean and sterile.

As stated before, most of the applications at this time, are developed for the home consumer electronic devices and gaming industry, but soon it will extend to other fields of interest, such automotive, automation or healthcare area. The age of computer classical mice and their pads might soon come to an end. With gesture-control enabled by devices like the Leap Motion, Xbox Kinect and Myo fast emerging, and touchscreens taking over just about everything else, the spot next to the keyboard might soon be empty.

Gesture recognition is an emerging technology that has the potential to revolutionize the way humans interact with machines; not only in gaming and automotive applications but also in day-to-day activities on the home front. Standalone devices like the Leap Motion controller will spread the awareness of this new revolutionary human machine interaction technology.

Even if the future of this technology is a bit hard to predict, because as with any other technology, it is constantly changing and depends also on other factors, a market report [1] written by a market research consulting company from the United States of America states that the total gesture recognition and touch-less sensing market is expected to reach \$22.04 billion by 2020 at a double digit CAGR[1] from 2013 till 2020, having in mind that the market value in 2012 was approximately \$2.2 billion (Figure 1.1). Therefore

[1] Compound annual growth rate (CAGR) is a business and investing specific term for the geometric progression ratio that provides a constant rate of return over the time period.

a growth of approximately 10 times is expected for the market of gesture recognition and touch-less sensing devices in a period of 8 years.

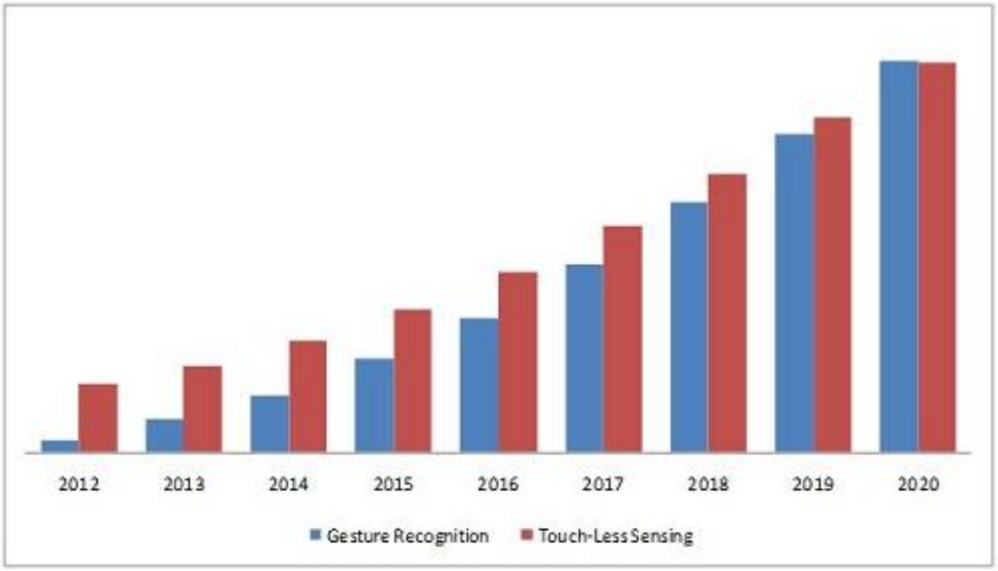


Figure 1.1 Gesture recognition and touch-less sensing market research (2012-2020) (taken from [1])

1.2. Motivation

My work will be focused on the technology that the Leap Motion controller offers and together with the API provided by Google, for their products, Google Maps and Street View in order to develop an application that aims to enhance the user experience of browsing locations in Google Maps.

What motives my decision to focus my research in the gesture recognition domain is the fact that this field of study was not explored much before, since this technologies are quite new on the market, leaving space to a lot of unknown factors.

The gesture recognition is an emerging technology that has the potential to revolutionize the way humans interact with the machines. The possibility to work with such a device that gives total freedom of move to the user and removes any other dependency (such as a mouse or a keyboard) is priceless.

The main speaker of a Los Angeles based web design conference focused on the future of interaction design, Christopher Noessel, mentioned during his presentation: “I think that the Leap Motion millimeter-wave gesture recognition is going to be super promising. Now that we can read fingers it’s going to get a lot more interesting as far as gesture recognition, and we’ll be able to maybe even sort of meet sci-fi at what sci-fi has been promising for that sort of thing” [2].

1.3. Paper structure

The paper is structured under eight chapters and aims to inform the reader and to provide him a full understanding of the project context, objectives, and related work in this field of study, how the solution was designed, built and tested and finally how it can be used.

Chapter 1 identifies the project context as well as the domain of study together with the motivation of this choice.

Chapter 2 specifies the project objectives, what should the final output of the work be, how the product will act in different situations and what are the requirements of the software and hardware involved.

Chapter 3 presents the bibliographic research that I did in the gesture recognition domain, what are the related works in this field (both from the software and hardware point of view), what the Leap Motion controller can do, how it is built and how it works.

Chapter 4 presents a conceptual system architecture on which the product implementation will be based. A brief description and presentation about LeapJS JavaScript library can be found and also the proposed algorithms that will be used during the implementation of the application.

Chapter 5 presents the detailed system architecture and the design patterns that were taken into consideration while writing the implementation. An overview on the JavaScript files will be presented as well as an overview about the general development guidelines that should be used when extending the library's functionality or maintaining the code.

Chapter 6 presents an analysis on Leap Motion's accuracy, precision and reliability in both static and dynamic scenarios. An overview on how precise and accurate are the measurements in terms of special positioning and gesture recognition is presenter.

Chapter 7 will summarize the (minimum) system requirements that a user needs in order to run the application, a manual for the developers, on how to install and use the library and a manual for the end user explaining how to use the application and its features.

Chapter 8 will present the conclusions. During this chapter I will give an overview about the results that I achieved and my contributions to this field of study and project, as well as a sections in which I will describe what further improvements can be done for the final product.

Chapter 2. Project Objectives

In this chapter I will present the project's objectives, what should the final output of the work be, how the product will act in different situations, what are the requirements of the software and hardware and what can it do and how it should do it.

2.1. Overview

The main goal of this project is to provide a JavaScript library, ready to go, that would act as a wrapper over the Google Maps and Street View products developed by Google and will handle the interaction between the user and the map through the use of a Leap Motion controller.

The project might be used as it is, simplifying the interaction between the end user and the Google's products mentioned above, providing a better user experience, or it can be integrated within any website that has a Google Map present.

2.2. Project specifications

2.2.1. Project features

The library will give full freedom of movement to the end users that will try to navigate through Google Maps and Street View. With simple hand gestures that will take place above the Leap Motion controller, the user will be able to interact with the digital world.

While in Google Maps, the user will be able to:

- Move the map
- Zoom in and out the map
- Switch to Street View

While in Street View, the user will be able to:

- Move from one panorama to another, both forward and backwards
- Rotate 360° the camera
- Switch to Google Maps

A help screen will be available.

2.2.2. What and How

In this sub-chapter I will go through each of the previously defined features (*what*) and will describe *how* they should be handled. Basically this is the place where the project specifications are refined, in such a way for everyone to understand how the application should act; the users should be able to use the application without any problem.

One of the best rules of usability is defined, as Steve Krug states, "don't make me think" [3]. The way the interaction is done using the Leap Motion controller should be straight forward and intuitive. The end user should be able to use the library as it is, without spending too much learning the gestures and how they will affect the flow of the application.

As stated before, a help screen will be available all the time. The user should press the *H* key in order to open it or to close it. The navigation between screens while the help window is opened will be done using swipe gestures.

2.2.2.1. Google Maps

Spreading the hand and moving on the x-y axis above the Leap Motion controller will force the map to move together with it (see Figure 2.1).

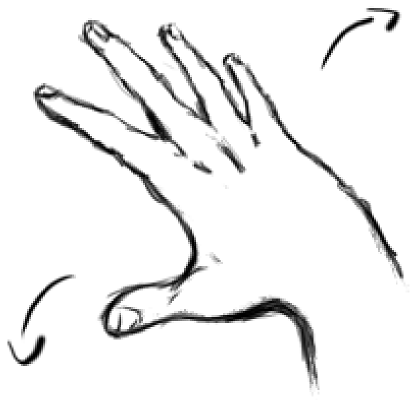


Figure 2.1 A hand which is spread

There exists a safe space in the center of the screen, where if the hand (represented by a pointer on the screen) will be placed there the map will not move. As the pointer (hand) moves to the edges of the screen, the map will start to move together with it. The distance of the pointer related to the center and edges of the map will affect the acceleration of the animation moving the map (see Figure 2.2).

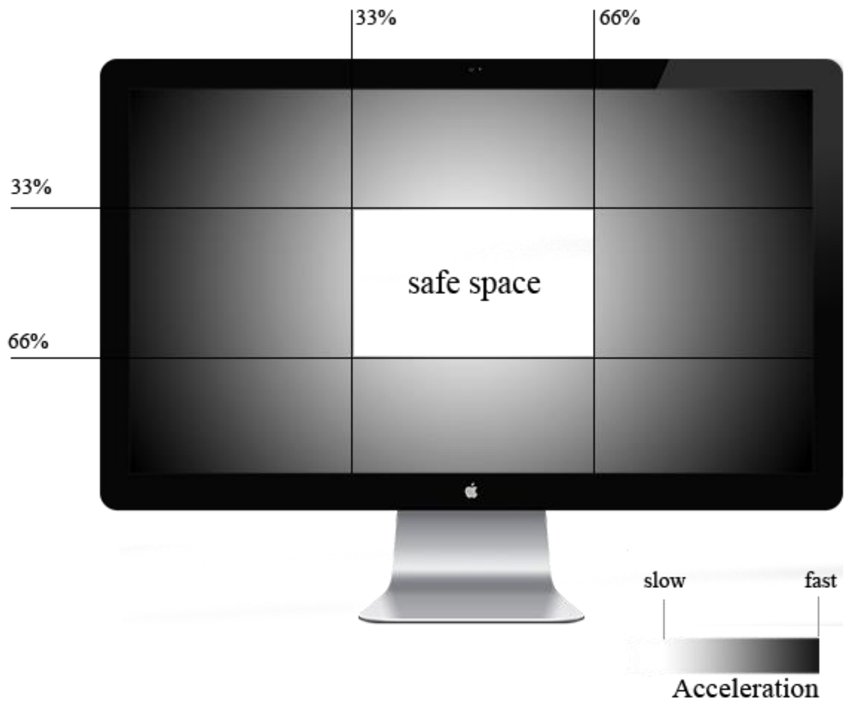


Figure 2.2 Navigation in Google Maps safe space and acceleration

Zooming in and out will be done by performing a circle gestures clockwise and counter-clockwise. A circle gesture is defined as a single finger tracing a circle (see Figure 2.3). When the user will perform a clockwise circle (note: only one finger should be extended while performing this gesture), the map should zoom in, while when a counter-clockwise circle gesture is performed and recognized by the Leap Motion software, the map should zoom out.

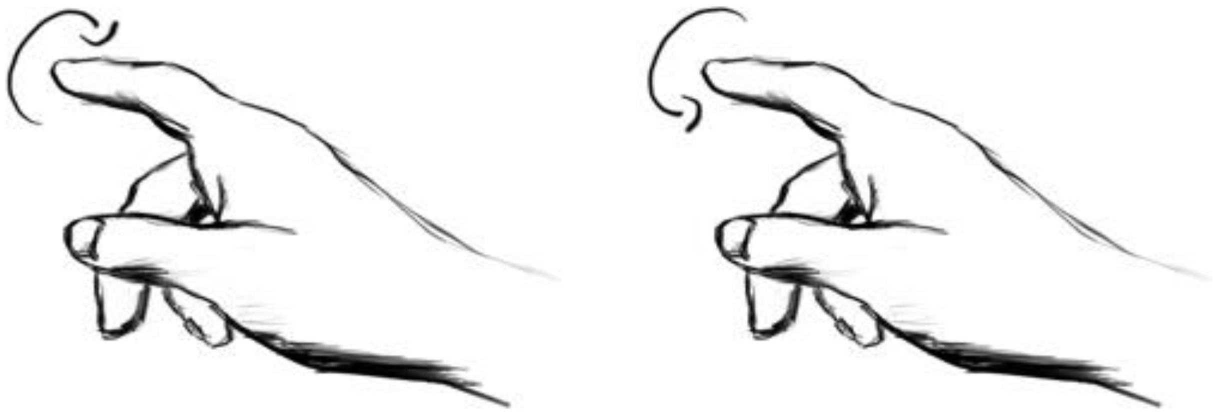


Figure 2.3 Circle gestures

To switch between Google Maps and Street View, the end user will perform a screen tap gesture (see Figure 2.4). A tapping movement by the finger as if tapping a vertical computer screen (moving forward on the z axis and then backwards) describes a Screen Tap gesture. Note: when performing a screen tap gesture only one finger of the hand should be extended.

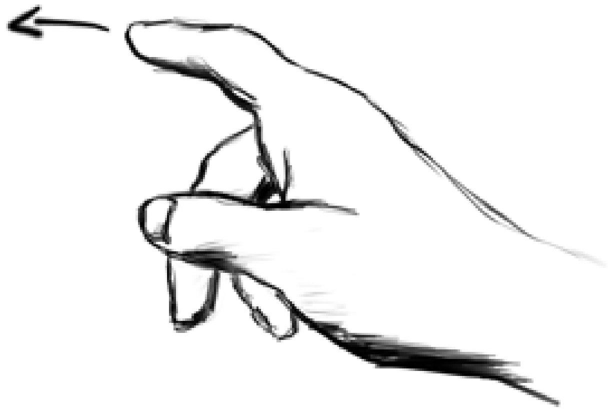


Figure 2.4 Screen tap gesture

2.2.2.2. Street View

The movement between one panorama to another will be done using circle gestures (see Figure 2.3). When a clockwise circle gesture is performed the panorama (point of view) will be moved to the next one available in front of the user. If a counter-clockwise circle gesture is performed, the panorama will be switched to another one which is available in the back of the user's position. Note: as before, only one finger should be extended while performing the circle gesture.

In order to rotate the camera's angle 360°, swipe gestures will be performed (see Figure 2.5). One swipe gesture, depending on the direction, will be able to move the camera on the N, W, S, E axis.

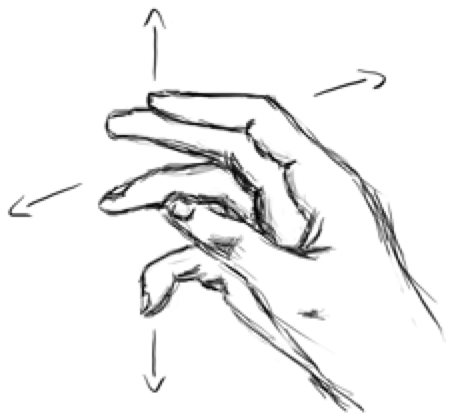


Figure 2.5 Swipe gestures

To switch back to Google Maps (close Street View), a screen tap gesture will be performed.

2.3. Requirements

In this sub-chapter I will present the functional and non-functional requirements for this project.

2.3.1. Functional requirements

The final product, the JavaScript library should be developed in such a way that it can be included in any web application without needing other dependencies.

The library will act like a wrapper above Google Maps and Street View, providing a way of interaction between the user and Google's products with the use of the Leap Motion controller.

The users should be able to navigate through the map by pointing to different locations with the hand spread in the air, zoom in and out the map by performing circle gestures and switch between Google Maps and Street View by performing screen tap gestures. While in Street View, the users should be able to easily move between two locations by performing clockwise and counter-clockwise circle gestures and they should be able to rotate the camera 360°.

2.3.2. Non-functional requirements

The final product will be a JavaScript library that can be included in any web application. The library will contain dependencies to jQuery, LeapJS and Google Maps JavaScript API V3.

The library should meet all Google's and Leap Motion's Terms of Service. Moreover, the library should be aligned with the web standards of the World Wide Web Consortium (W3C).

The library should be free to use (open source) and available for download at a public internet source.

If the intent is to be open source, and therefore used by other developers too, the code used to build the library should be easy to read, maintain and scale. Every function should be well documented in order to let other developers to easily use it, understand it and modify it.

The library should be fully compatible between all the modern browsers (Google Chrome, Mozilla Firefox, Safari and Internet Explorer 10 and higher).

The minimum system requirements should only be affected by the Leap Motion controller:

- AMD Phenom™ II or Intel® Core™ i3 or better
- 2 GB RAM
- USB 2.0 Port
- Internet Connection (also needed by Google Maps API).

Lastly, from the efficiency and performance point of view, the loading time of the library should be less than one second at a speed of 20Mbps, in order to not affect the performance of the applications in which will be used.

2.4. Use case specification

Let us assume that a user will use this application in order to find a specific location on the map, enter street view and navigate near the destination.

The basic and alternate flow are also presented in Figure 2.5.

2.4.1. Terminology

Actor – the end user of the application.
System – the application itself.

2.4.2. Preconditions

The actor should have:

- Internet connection
- Leap Motion controller up and running

The actor closed the help screen which appears at the first run of the application.

2.4.3. Basic flow

Use case start

This use case starts when the actor wants to find a specific location on the map.

1. The actor will spread his hand and navigate through the map
2. The system will respond to its gestures and move the map accordingly.
3. The actor will perform clockwise circle gestures in order to zoom in on a specific location.
4. The system will respond to the gestures and will zoom in the map.
5. The actor will perform a screen tap gesture to enter street view near its destination.
6. The system will open street view at the specified point
7. The actor will perform circle and swipe gestures in order to navigate in street view mode to reach his destination.
8. The system will respond to the gestures performed and will act accordingly.

Use case end

This use case ends when the actor finds its specific location.

2.4.4. Alternate flow

1. The system will not recognize the actor's gestures.
   This case can occur at steps 1, 3, 5, 7. The system will do nothing.
2. The system will not find any Street View panorama available at the location the user pointed.
   This case can occur at step 5.

The system will open a JavaScript alert notifying the user that there is no Street panorama available in a range of 150 meters from the latitude and longitude specified.

2.4.5. Postconditions

The actor found the destination needed.

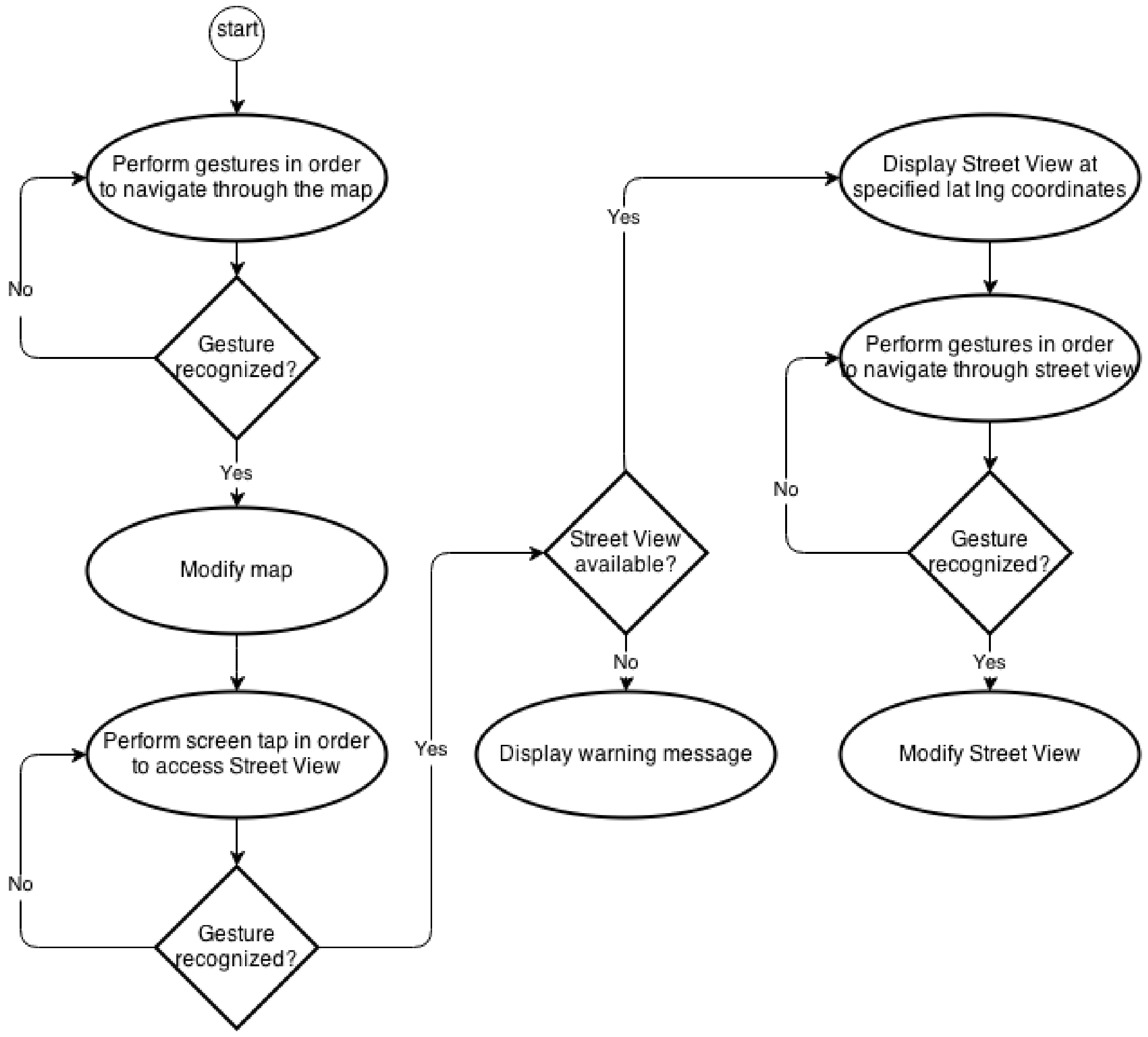

start
Perform gestures in order to navigate through the map
Gesture recognized?
No
Yes
Modify map
Perform screen tap in order to access Street View
Gesture recognized?
No
Yes
Street View available?
Yes
No
Display warning message
Display Street View at specified lat lng coordinates
Perform gestures in order to navigate through street view
Gesture recognized?
No
Yes
Modify Street View

Figure 2.6 Flow diagram of the Use Case

2.5. Project goal

The project's end goal is to deliver a library that acts as a wrapper over Google Maps and Street View that should facilitate the interaction between the end user and the map or street view through the use of the Leap Motion controller.

The library should be open source, therefore free to use, lightweight, easy to use and fast.

Chapter 3. Bibliographic Research

In this chapter I will present my research related to what other similar solutions do exists related to my work at this moment on the market, I will present the way the Leap Motion Controller is build, how it works and how reliable it is and finally I will present my research results in the gesture recognition devices and sensors field (trying to accentuate what other devices except the Leap Motion are present on the market).

3.1. Related work

This section of the thesis briefly reviews related work in the areas directly associated with Leap Motion Controller and Google Maps API. Finally, conclusions will be drawn in order to accentuate the pluses and the contribution of this thesis in the fields mentioned above.

3.1.1. Leap Motion and Google Earth

One of the most known application that is available in this field is the *Google Earth* plugin developed by Google themselves. The plugins allows the user to travel the world through a virtual glove and view satellite imagery, maps, terrain and 3D buildings [4]. Even though the solution provided by Google is quite unique and comes with a lot of features, the plugin comes with some downsides and some flaws.

From the beginning we can see that the plugin is way too sensitive to use. Trying to control the application out of the box can be a disaster since manipulating the space can be quite difficult. The earth can get into a state when is constantly spinning, the motion speed while using the plugin is varying too much because of the high sensibility and sometimes the movements are too large, fast or out of control. Another downside of Google Earth plugin is that you need to have installed Google Earth (which is a software that requires relatively high system settings, such as: 2GB+ free hard disk space, a network speed of 768 Kbits/sec and so on [5]).

3.1.2. Leap Motion and Hyperlapse (by Teehan+Lax Labs)

The developers from Teehan+Lax Labs [6] have come up with a solution that is able to create hyper-lapse[2] videos from Google Street View panoramas. Because of the fact that the original code of the hyperlapse.js is available on GitHub I investigated their solution: two locations must be defined, using Google Maps' public Routing API a route is being computed between that two locations and all the panoramas available between this two locations are downloaded and cached into the browser memory and then stitched up together using GSVPano.js. After all the images are downloaded, using Three.js the movement between frames is being implemented.

The only big problem here is that this innovative library for Google Maps Street View, Hyperlapse.js, **breaks one of the Google Maps' terms of agreement**, which is: "you must not use the Products in a manner that gives you or any other person access to

[2] A hyper-lapse is an exposure technique in time-lapse photography, in which the position of the camera is being changed between each frame in order to create a tracking shot. Definition taken from: http://en.wikipedia.org/wiki/Hyperlapse.

mass downloads or bulk feeds of any Content" (more information can be found under 2(d) in the Google Maps/Earth Terms of Service [7]).

The app is entirely written in Javascript using LeapJS and a modified version of Hyperlapse.js [8] and is not public. There exists only an online video on YouTube platform demoing the application.

3.1.3. Conclusions

As we can see from the presented related work that exists at this moment on the market (August 2014), there exists only two applications in this domain, which have involved both the Leap Motion controller and the Google Maps API, one of which is not public (and also breaks Google's Terms of Service) and another one which is similar, but intended to use with another product – not Google Maps, but Google Earth, which requires some software dependencies such as the application itself.

Therefore, we can see there is a niche that can be explored. The presence of the flaws and lack of portability of the Google Earth solution and no plugins available for the actual Google Maps or Street View gives me the opportunity to develop an application that will have human interaction without the necessary use of the keyboard or mouse, through the support given by the Leap Motion controller.

3.2. Leap Motion controller

The *Leap Motion controller* is a computer hardware 3D sensor device that supports hand and finger motions as input, but it does not require any hand contact leaving the hand navigate free into the air. The *Leap Motion Controller* senses how you naturally move your hands and lets you use your computer in a whole new revolutionary way. The *Leap Motion controller* (Figure 3.1) is sleek, light (45 grams) and has relatively small dimensions, being 80 mm long, 12.7 mm tall and having a width of 30 mm [9].

The Leap Motion Controller represents a revolutionary input device for gesture-based human-computer interaction. It allows for the precise and fluid tracking of multiple hands, fingers, and small objects in free space with sub-millimeter accuracy.

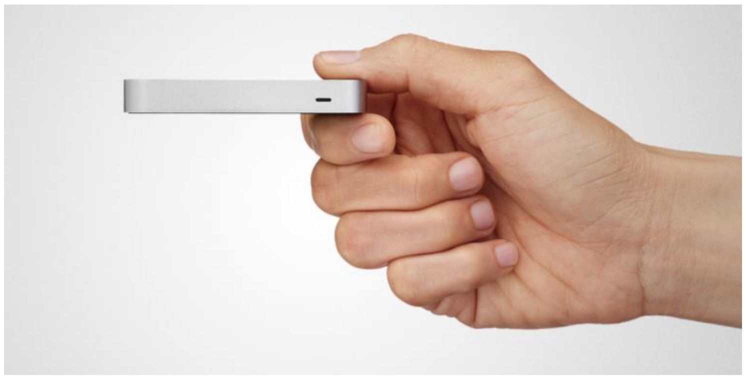


Figure 3.1 The *Leap Motion controller* (taken from www.leapmotion.com)

3.2.1. Leap Motion hardware architecture

The *Leap Motion controller* is an input peripheral device with an USB connector. The device’s architecture is rather simple, it uses two monochromatic IR cameras and three infrared LEDs (see Figure 3.2). Therefore, the controller can be categorized as an optical tracking system based on the stereo vision principle. The Leap software analyzes the objects observed in the device’s field of view. It recognizes hands, fingers, and tools, reporting discrete positions, gestures, and motion.

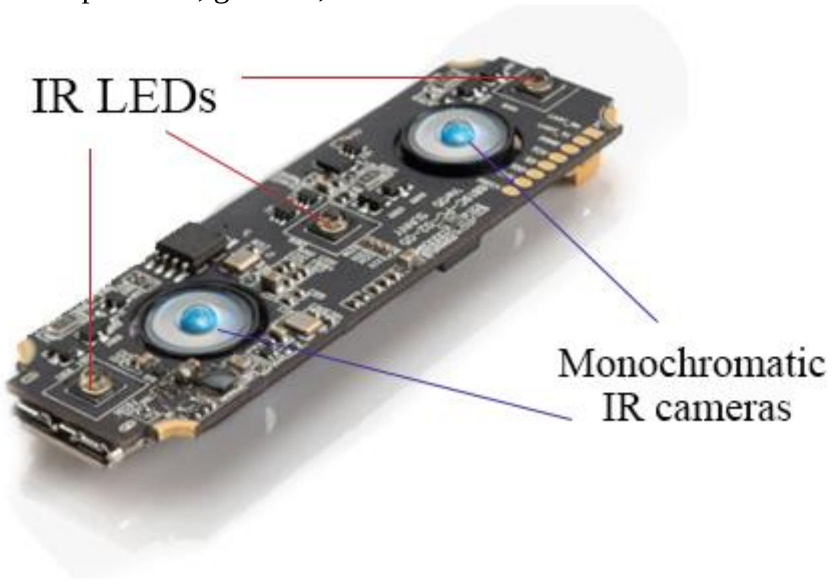


Figure 3.2 Leap Motion hardware architecture
(original image taken from www.designboom.com)

The *Leap Motion controller* scans a region in the shape of an inverted pyramid centered at the device’s center and extending upwards – this space where the hands can be detected is also called the interaction box (see Figure 3.3). The effective range of the controller extends from approximately 25 to 600 millimeters above the device.

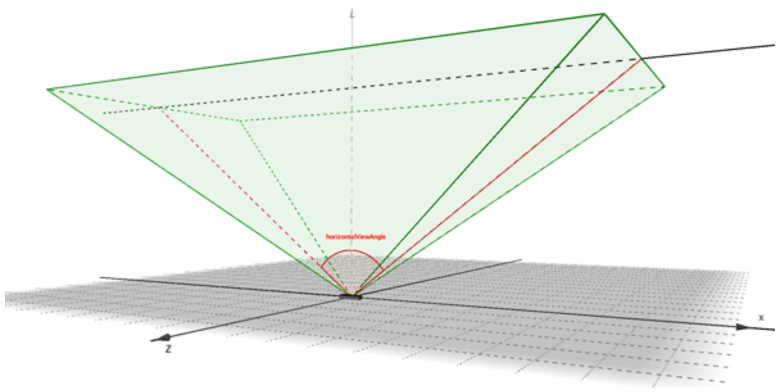


Figure 3.3 *Leap Motion*’s field of view (taken from www.leapmotion.com)

3.2.2. Leap Motion system architecture

The Leap Motion software runs as a service. The software connects the input device (Leap Motion controller) over the USB hub with the computer. The Leap Motion service receives motion tracking data from the hardware device. Over that, the SDK provides two varieties of API for getting the Leap Motion data:

- A native interface (a dynamic library that can be used to create new Leap-enabled applications – the library can be linked in C++, ObjectiveC, Java, C# or Python)
- A WebSocket interface (together with the JavaScript client library LeapJS allows the creation of Leap-enabled web applications)

Because of the nature of my application proposal, I will focus on the WebSocket interface and try to explain how it works.

The Leap Motion service runs a WebSocket server on the localhost domain at the port 6437. The WebSocket interface provides motion data (captured from the Leap Motion controller) in the form of JSON messages. A JavaScript client library, as stated before, is available and can be used in order to consume the JSON messages and present the tracking data as regular JavaScript objects.

This interface is intended to be used in web applications, but it can be accessed through any kind of application that can establish a WebSocket connection.

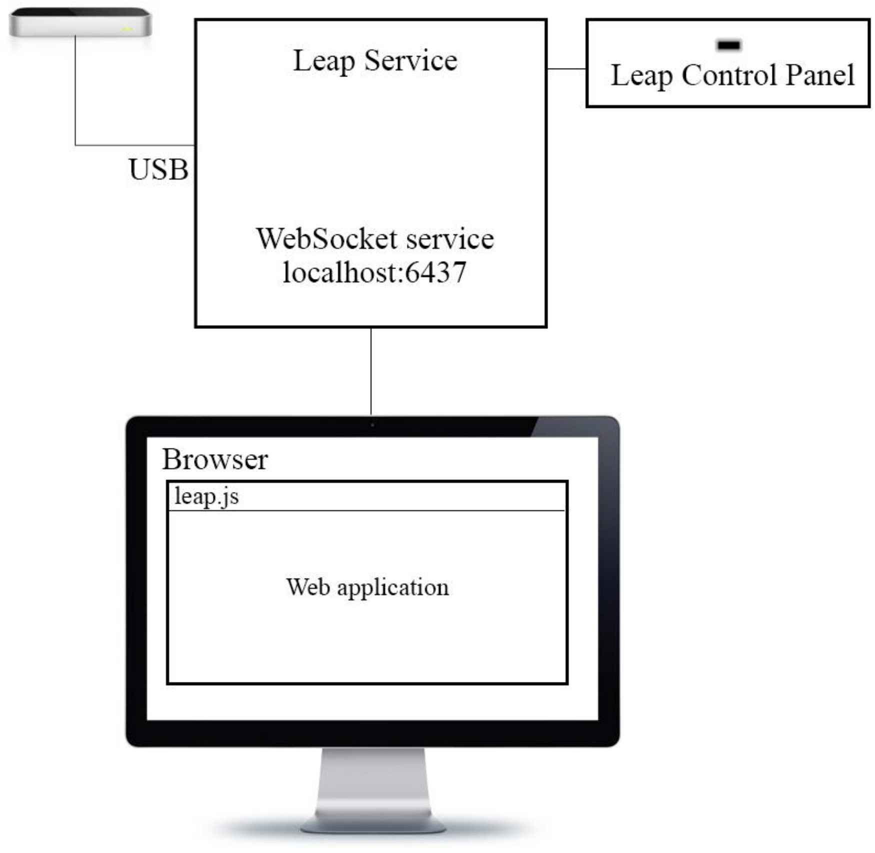


Figure 3.4 Leap Motion WebSocket interface

The Leap Motion service provides a WebSocket server listener at localhost:6437 address. The server sends tracking data as JSONs messages while it receives data from the hardware device. The leap.js client JavaScript library it is designed in such way to establishes the connection to the WebSocket server and to consume the JSON messages; the library should be included and used in web applications [10].

3.2.3. Spatial positioning – an overview on precision and reliability

The study's goals on which this sub-chapter is based was to analyze the precision and reliability of the *Leap Motion Controller* in static and dynamic conditions and to determine its suitability as an economically attractive finger/hand and object tracking sensor [11].

Two types of measurements were performed within the experiment, under two experimental conditions:

- Static conditions: acquisition of a limited number of static points in space (3,000 measurements were taken of 37 positions)
- Dynamic conditions: tracking of moving objects with constant inter-object distance within the calibrated space (119,360 measurements were taken in an attempt to cover the estimated useful sensory space of the controller).

The results of the study, in the *static scenario*, are presented in Table 3.1. As we can see, the standard deviation is less than 0.5 mm at all times, therefore we can say that the controller is quite precise and accurate in determining the static positioning in space.

Table 3.1 Results of static study (taken from [11])

	Standard deviation position	x Axis (std_x)	y Axis (std_y)	z Axis (std_z)	Spatial position (std)
Minimal std	std(mm)	**0.0081**	0.0093	0.015	0.013
	location(x, y, z) (cm)	(0, 30, 0)	(-10,-10,-5)	(0, 20, -5)	(0, 15, 0)
Maximal std	std(mm)	0.39	**0.49**	0.37	0.38
	location(x, y, z) (cm)	(-20, 20,0)	(-20, 30, 0)	(-20, 30, 0)	(-20, 30,0)

The lowest standard deviation (0.0081 mm) was measured on the x axis at 30 cm above the controller, while the highest standard deviation (0.49 mm) was measured on the y axis at the leftmost and topmost positions [11]. In terms of spatial positioning, the results show a minimal standard deviation of 0.013mm (when the object is positioned right above the controller at 15mm height), while the maximum standard deviation of 0.38 mm was measured at 20cm on the left hand side of the controller and 30cm height.

The set of measurements in the *dynamic scenario* revealed that the accuracy of the controller drops when the objects move away from the sensor, therefore resulting in an inconsistent performance of the controller.

Previous similar tests were done using a robotic arm, both in static and dynamic scenarios, in order to measure the precision of the sensor [12]. The study from 2013 was focused on an early version of the Leap Motion Controller taking into account the so-called tremor (defined as an involuntary and approximately rhythmic movement of muscles).

Depending on the human age, the tremor value can be in the range of 0.4mm ± 0.2mm (for young adults) to 1.1mm ± 0.6 mm (for older adults) [13]. The study from 2013 has revealed even bigger standard deviations, therefore this proves that the Leap software has improved from earlier phases by now, and even though this deviation occurs, the study has shown that comparable controllers in the same price range (e.g., the Microsoft Kinect) were not able to achieve the accuracy that the Leap Motion has [12].

From the studies presented above, we can draw the conclusion that the accuracy of the Leap Motion controller decreases when the objects are moving away from the sensor and when moving to the far left or right of the controller. However, having a relatively low standard deviation (less than 0.5 mm at all times [11]), the controller can be considered, in my opinion, a very good tool, with a high quality/price ratio, for personal use, but not as a professional tracking system.

3.3. Other gesture recognition sensors and devices

The gesture recognition is an emerging technology that has the potential to revolutionize the way humans interact with the machines. Together with the evolution of touchless sensing industry, the demand of the market in the past years for gesture recognition devices and sensors has increased and therefore we can find a lot of devices on the market in different shapes and sizes.

In this sub-chapter I will enumerate and briefly describe what other important 3D sensors devices are available on the market, beside the Leap Motion controller.

3.3.1. Wii (by Nintendo)

In the domain of gesture recognition, one of the first accurate and commercially viable solutions was the Nintendo Wii controller. The Wii controller, released in 2006, looks like a TV remote and was designed as a game controller. The primary control of the movement is the controller itself, containing solid-state accelerometers that let it sense full 3D gestures patters, such as:

- Tilting and rotation up, down, left and right
- Rotating along the main axis (as with a screwdriver)
- Acceleration up, down, left and right
- Acceleration towards the screen and away [14]

Because of the fact that the WiiMote can operate as a separate device, it was used in many applications that worked with a computer.

Compared with the Leap Motion Controller, the WiiMote can detect hand gestures in the air field, but it has no information about the fingers position, alignment and gestures.

3.3.2. Kinect (by Microsoft)

The Kinect (see Figure 3.5), developed by Microsoft and released in 2010, comes as an add-on sensor for the Xbox 360 gaming console. The controller has visual and auditory (voice recognition) inputs and includes a 3D depth-sensing camera which gives the opportunity to the developers to incorporate, acquire and recognize full body gestures of multiple users at a time. Moreover, having an open SDK allows the developers to integrate this controller in other fields also, allowing it to detect other objects than human bodies.

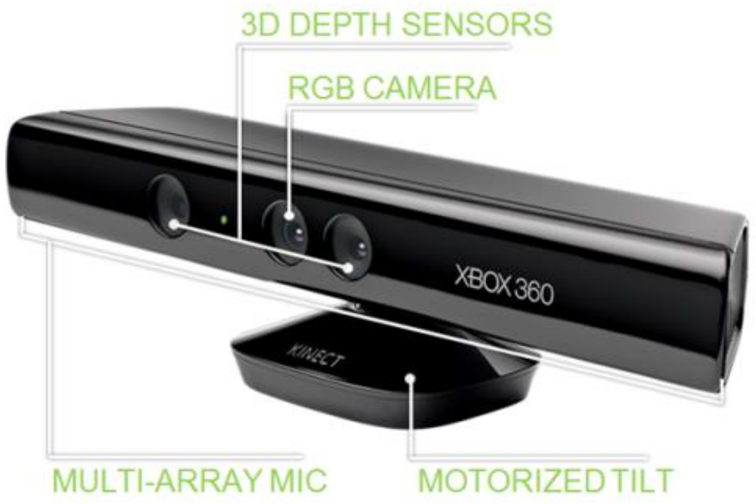


Figure 3.5 Microsoft Kinect (taken from www.generationrobots.com)

Gathering massive amounts of data from motion-capture in real-life scenarios, the Kinect developers processed that data using a machine-learning algorithms and they were able to map the data to models representing people of different ages, body types, genders and clothing. With select data, developers were able to teach the system to classify the skeletal movements of each model, emphasizing the joints and distances between those joints [15].

However, in comparison with the Wii controller, the Microsoft Kinect allows the full body gesture recognition, instead of only hand movement recognition, but compared with the Leap Motion controller it has no precision at the level of fingers.

3.3.3. Xtion PRO Live (by Asus)

The Xtion PRO Live is an alternative to Microsoft's Kinect. It is capable of registering:

- gesture recognition by tracking people's hand motions, having some predefined poses such as: push, click, circle, wave,
- (whole) body recognition,
- RGB detection: the controller is able to capture full RGB colored images allowing developers to design security systems, digital signage and many more,
- audio signals, allowing for voice recognition, video conference (together with the RGB detection) and so on [16].

In comparison with Microsoft's product, Kinect, the Xtion Live has no motor, therefore the user should manually position the sensor, the controller is also a less popular device and therefore a lack of applications available on the market can be noticed.

3.3.4. PrimeSense Carmine

PrimeSense Carmine is an alternative for the Xtion Live (by Asus), but manufactured under a different brand name. The only difference at some point was that the PrimeSense Carmine had a better USB 3.0 compatibility, but the problem was solved with a firmware update for the Asus device, making both devices equivalent [17].

3.3.5. RealSense (by Intel)

Similar with the Microsoft's Kinect, Asus' Xtion Live and PrimeSense Carmine the RealSense technology developed by Intel is based on the depth sensors. The camera provides to the PCs and tablets a 3D vision, while the voice technologies incorporated can send audio information to the affiliated devices.

The device can:

- analyze facial images (by tracking multiple faces, identifying the eyes, mouth and nose),
- detect and track hands and fingers (up to 10 simultaneous fingers, and 8 gestures predefined, such as thumbs up, dynamic waving of the hand and so on, but it also has the possibility to process raw depth data, similar with the Leap Motion controller and SDK),
- recognize speech
- subtract the background (from the user's body, for instance),
- track objects within the camera range and draw CG images on real world scenarios [18].

3.3.6. PointTouch (by PointGrab)

PointTouch, by PointGrab, is designed for consumer electronic devices (such as personal computers, tablets, TVs and others). The device will create a virtual touch space between the devices and the user, so they can point directly at anything on the screen in order to access it. The interesting part of the technology is that PointTouch does not rely only on the hand position, but it computes the line of sight based on a highly accurate 3D finger and face position instead (see Picture 3.6) [19].

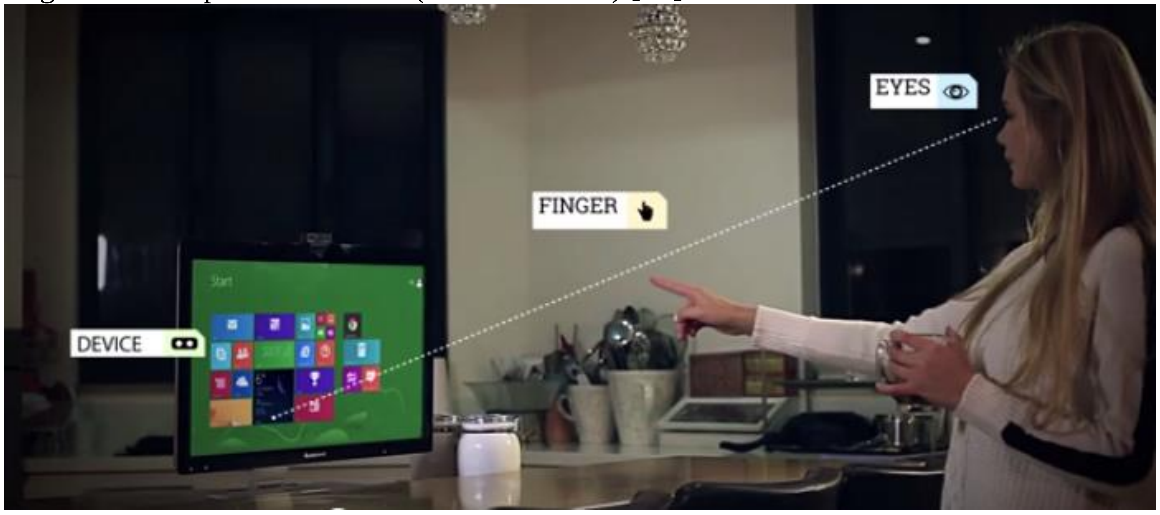


Figure 3.6 PointTouch eye-finger-device system (taken from www.pointgrab.com)

3.3.7. Elliptic Labs

The engineers from Elliptic Labs developed a unique, innovative product that is able to deliver touchless gestures for multi-platform human interaction by using ultra sounds.

The technology behind it works in a similar way the bats are adapting to the environment. Ultrasounds signals are sent through the air from the speakers integrated in the devices, then they bounce against the user's hands and fingers back to the device where are recorded by the microphones provided by the sensor.

The SDK provided by Elliptic Labs contains a few predefined hand and finger gestures, such as selection (screen tap), horizontal and vertical scroll (using swipes), drop down menus (using vertical swipes), rotate (using circle gestures) and trajectory control (using swipes).

Because of the fact that the technology involved in the gesture recognition process uses sound waves, it gives the opportunity to use the full 3D space available at 180° in front of the device (sensor) without any other restrictions. Using ultra sounds, the sensor's size is really small, enabling easier integration in any devices (it can fit in your monitor or tablet frame). Another plus of this technology is that it uses a small fraction of power in comparison with other 3D recognition technologies like depth sensors, IR cameras and so on [20].

3.3.8. Myo (by Thalmic Labs)

The Myo, developed by Thalmic Labs, is a revolutionary armband device that lets you use the electrical activity in your muscles to wirelessly control your computer, phone, and other favorite digital technologies (see Figure 3.7).

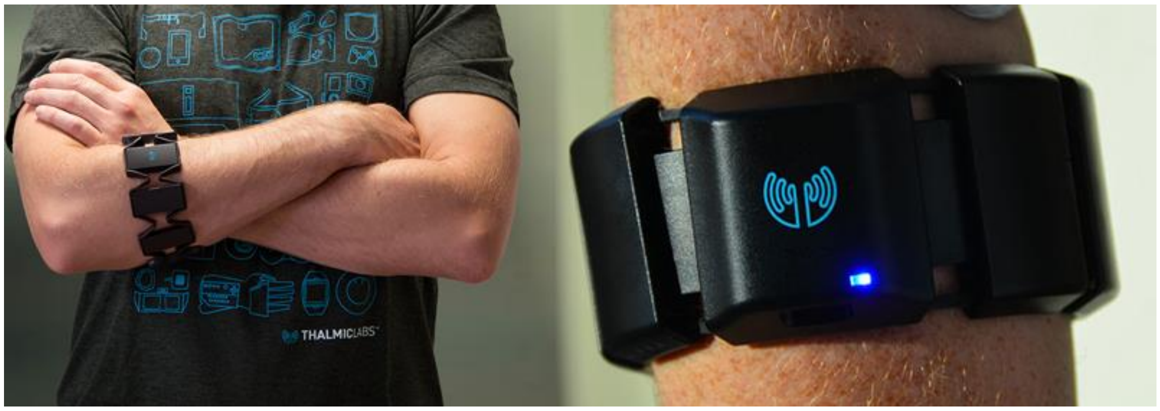


Figure 3.7 The Myo armband (original images taken from www.thalmic.com)

Using EMG sensors[3], the Myo armband measures the electrical activity from the user's muscles in order to detect what gestures he is making. Having a relatively low price (150$ - August 2014), the Myo armband can revolutionize the way people can interact with the digital world, leaving them full movement capability (the device's area of interaction is only restricted by the range of the Bluetooth emitter, which can be up to 100 meters), but

[3] Electromyography (EMG) is a technique for evaluating and recording the electrical activity produced by skeletal muscles.

also can revolutionize the life of impaired people helping, not necessarily to reproduce the movement of the hand, but to transmit data to a computer, smartphone, or any other device that has bluetooth sensors, and help them interact with the digital world in a much more simpler way [21].

Unfortunately, the device, at this moment (August 2014), is only available for pre-ordering, the shipping beginning from September 2014.

3.3.9. Nimble (by Intugine)

Nimble is a device, developed in India, which uses a ring and a sensor combination (see Figure 3.8) in order to detect gestures. Compared with the Leap Motion controller, the device requires the ring to be present on your hand, while using Leap Motion controller your hand is completely free, but limited to the interaction box available; the Nimble sensor can detect interaction up to 4.5 meters away from the device.

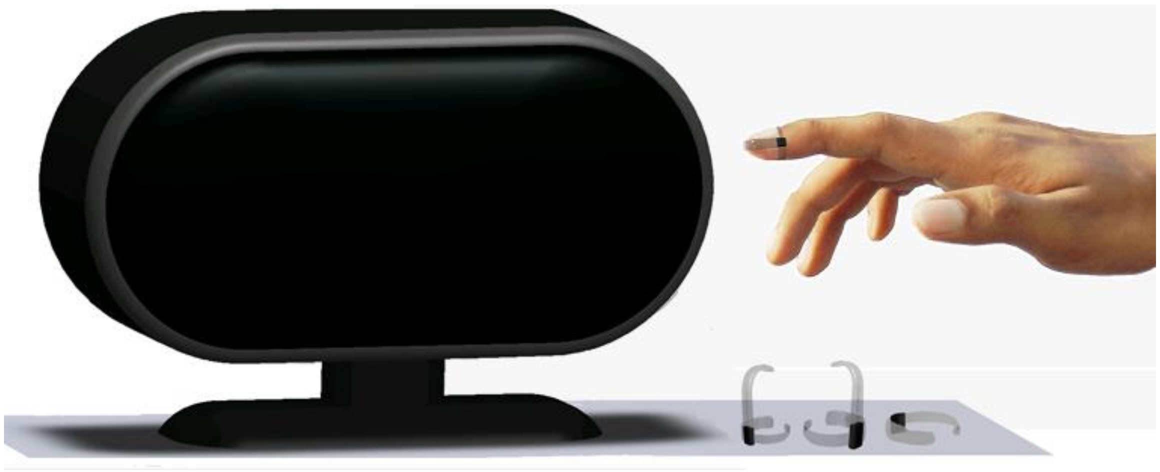


Figure 3.8 Intugine Nimble sensor and ring (taken from www.intugine.com)

The intent is to integrate the Nimble device not only with the computers, but also with smart TVs, home automation devices and game consoles. We can say that Nimble effectively can replace not only the mouse and the keyboard but also the remote controllers or gaming consoles.

At this moment, the Nimble controller is only available for pre-ordering and it will be released in the last quarter of 2014, therefore is not accessible at this moment on the market [22].

3.3.10. Conclusions

The market and field of gesture recognition devices, sensors and software is increasing at this moment, as we speak. The gesture recognition is an emerging technology that has the potential to revolutionize the way humans interact with the machines (computer, tablets, smartphones, TVs and so on). As we can see from the previously presented devices, at this moment there are quite a few options in this field of technology, each of them providing similar solutions (what), but coming up with different approaches (how).

Chapter 4. Analysis and Theoretical Foundation

In this chapter I will explain the operating principles of the implemented application. I will describe my solution from a theoretical point of view presenting a conceptual system architecture. I will briefly present and describe the way the LeapJS library works and what data is received from the Leap Motion controller, and finally I will present the proposed algorithms that will be used in the implementation.

4.1. Conceptual system architecture

In this sub-chapter I will present the conceptual architecture of the application. As you can see from Figure 4.1 all the components developed within this project lay at the client side. The desire was that the final product to be a JavaScript library, therefore all the code that is in scope should lay in the browser.

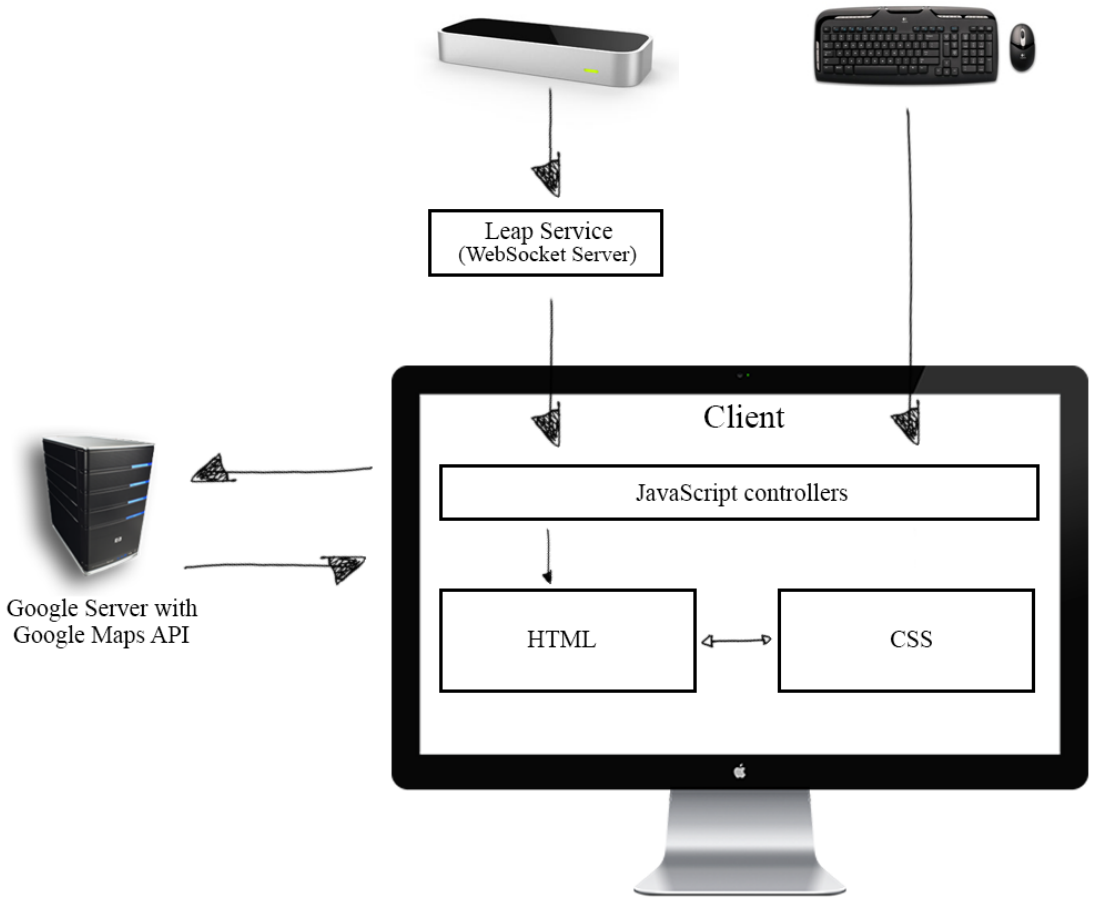


Figure 4.1 Conceptual system architecture

The system can take input from the Leap Motion controller, the keyboard and the mouse (though not recommended since the scope of the project is to eliminate the use of the keyboard and mouse) and also from the Google Maps API.

The interaction between the Leap Motion controller and our system is done through the help of the LeapJS JavaScript library, which connects to the WebSocket server located at localhost:6437 (where the Leap Motion hardware device sends data) and consumes the messages which are delivered in the form of JSON messages.

The interaction between our system and Google's data is done through the Google Maps JavaScript API V3. The applications sends requests to Google's servers and the servers respond by modifying the map which lays and is rendered on the client side, in the browser.

JavaScript controllers are defined and implemented with the scope of:

- Detecting, interpreting and handling Leap Motion data (frames and gestures) – LeapController.js
- Generic control over the application, handling events – Controller.js
- Interacting with the Google Map and Street View – MapsController.js

The final product being delivered as a JavaScript library, one of the main nonfunctional requirements was to keep the library size as low as possible in order to be able to load as fast as possible, not affecting the performance of the application in which will be used. In order to do so, I tried to think the architecture of the application in such way to keep the code as clean as possible, without complications, without unnecessary code, keeping only the strictly necessary code in scope. This is the reason why the architecture of the application is so clean and simple.

4.2. LeapJS library

LeapJS[4] is the Leap Motion JavaScript framework developed and publicly shared with the public by the employees of Leap Motion. Even though the LeapJS is maintained by Leap Motion, developers can contribute by requesting a pull and submitting their solution to be reviewed by the Leap Motion team.

4.2.1. Leap Motion communication protocol

First of all, I would like to talk about how the Leap Motion hardware device sends the data that is detected in its field of view to the computer and then to the client (browser). The Leap Motion controller provides a protocol for communicating the data received in the real world. The *leapd* is a service which interprets data coming from the Leap Motion controller and reconstructs hand position. The hand data is made available via a WebSocket server within *leapd*. The web socket communication is done through a local service on the machine on which the Leap Motion controller is connected. The client establishes a connection to the *leapd* deamon by connecting to the 6347 port, in this specific case this is done through the help of LeapJS library.

Thea *leapd* service sends to the client:

- Events – informing the client about changing the state of the device (e.g. connection status)
- Frames – JSON blocks of data detected by the Leap Motion device; the frame structure can be read from Table 4.1

[4] LeapJS on GitHub: https://github.com/leapmotion/leapjs

Table 4.1 A Leap Motion *frame* structure

Field name	Type	Description
id	Float	A unique identifier of the frame
timestamp	Integer	Timestamp when the frame was detected
hands	Hand[]	An array of Hand objects; See Table 4.2
interactionBox	InteractionBox	The dimensions of the measurable area of the Leap Motion Controller hardware; See Table 4.3
pointables	Pointable[]	An array of all the pointables detected from all hands in the interaction box: fingers, pens, sticks etc
gestures	Gesture[]	An array of Gesture objects currently detected; See Table 4.4
r	Float[][]	(rotation) a 4x4 matrix reflecting overall motion of detected objects; For internal use
s	Float[]	(scale) a 3 values array of floats indicating relative size of the space occupied by detected objects; For internal use
t	Float[]	(translation) a 3 values array of the net relative motion of detected objects; For internal use

Table 4.2 Structure of *Hand* object

Field name	Type	Description
Hand.id	Integer	The identity of the hand
Hand.direction	Float[]	Euler angles of the hand (3 floats)
Hand.palmNormal	Float[]	Euler angles of the palm (3 floats)
Hand.palmPosition	Float[]	The position of the center of the hand, in millimeters, relative to the Controller hardware (3 floats)
Hand.palmVelocity	Float[]	The change in position of the palm relative to the last measured position of the hand (3 floats)
Hand.sphereCenter	Float[]	The position of an imaginary ball being held by the hand, in millimeters, relative to the Controller hardware (3 floats)
Hand.sphereRadius	Float	The distance from the imaginary ball to your palm

Table 4.3 Structure of *InteractionBox* object

Field name	Type	Description
InteractionBox.center	Float[]	The position, in millimeters, of the origin relative to the Controller hardware (3 floats)
InteractionBox.size	Float[]	The extend of the Controller detection area, in millimeters. The three values are width, height and depth.

Table 4.4 Structure of *Gesture* object

Field name	Type	Description
Gesture.center	Float[]	The position, in millimeters, of the gesture. For circular gestures it is the position of the center of the gesture
Gesture.duration	Integer	The number in microseconds that the gesture has been maintained
Gesture.handIds	Integer[]	The ids of the hands making the gestures
Gesture.pointableIds	Integer[]	The ids of the pointables making the gestures
Gesture.radius	Float	For the circle gesture, the approximate distance in millimeters between the center and the pointable (or the mean location of the pointables)
Gesture.state	String	• start • update • stop
Gesture.type	String	• circle • swipe • keyTap • screenTap

More information about the Leap Motion system architecture can be found under the bibliographic research, sub-chapter *Leap Motion controller*.

4.2.2. An overview on LeapJS

The Leap Motion API measures the physical real world quantities with the specified measurements units from Table 4.5.

Table 4.5 Leap Motion measurements units

Quantity	Unit
Distance	Millimeters
Time	Microseconds (=0.001Millisecond)
Speed	Millimeters/Second
Angle	Radians

The LeapJS library connects to the WebSocket server, as specified before, and brings to the client useful information detected in the leap frame (see previous chapter).

The version 2 of the API, which was used during the implementation of this product, also called *Skeletal tracking* provides additional information about hands and fingers (from the ones specified in the previous sub-chapter) and also improves the overall tracking data.

The additional features include:

- Confidence rating about hand, fingers and tools recognition
- Right/Left handedness identification
- Digits identification
- Position and orientation of finger bones
- Grip factors (pinching or grasping)
- Individual finger reporting
- Finger extended or not.

Since 28th of August, the Leap Motion API is also able to provide raw sensor images (see Figure 4.2). The image data contains the measured IR brightness values and the calibration data required to correct for the complex lens distortion.

More information can be found on leap motion's documentation website [23].

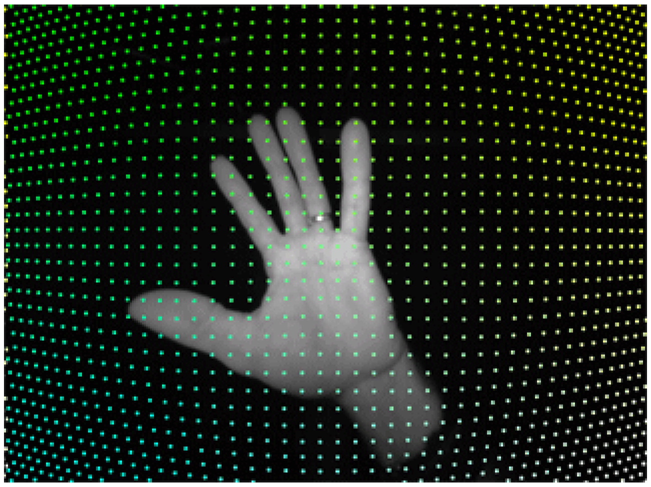


Figure 4.2 A raw sensor image with superimposed calibration points (taken from https://developer.leapmotion.com/)

4.3. Proposed algorithms

4.3.1. Mapping real world coordinates to virtual coordinates

One of the first challenges that this application gives to the developers is how to map the real-world coordinates, detected and registered by the Leap Motion controller in the virtual world, to the appropriate application-defined coordinates system.

First of all we must understand how the Leap Motion software works and what the coordinate system is it using (see Figure 4.3).

The Leap motion controller provides coordinates in units of real world millimeters in form of a vector (x, y, z). The values are given in millimeters and represent the deviation from the center of the Leap Motion controller. The origin is located at the top center of the hardware.

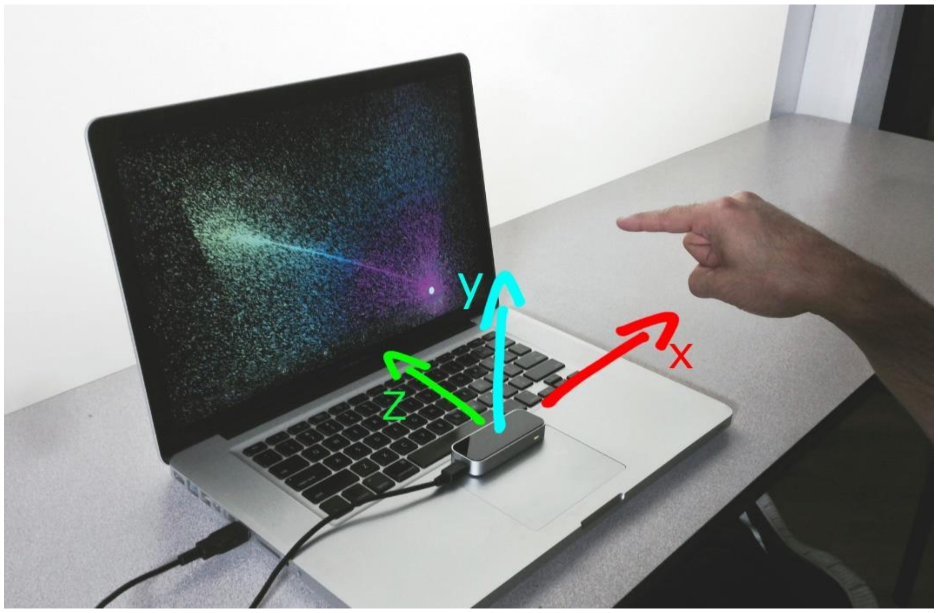


Figure 4.3 Leap Motion controller's coordinate system
(original image taken from www.snibbestudio.com)

For the purpose of this application, the mapping of the Leap Motion data will only take in consideration the x and y axis. In my case, the origin of the application coordinate will be at the top left corner of the window, with y values increasing downward. The Leap Motion controller, however, has the y values increase upwards, so basically the y axis have to be flipped.

The Leap Motion controller offers an interaction box which is represented as a rectilinear area within the Leap Motion field of view (remember Figure 3.3). The Leap software guarantees that if any movement is done inside the interaction box it will be detected, therefore the mapping should be done with respect to the interaction box, not to the entire field of view that the Leap Motion controller has. The size of the interaction box is determined by the Leap Motion field of view and the users' interaction height setting (from the control panel) [24].

Having all the information gathered, we can move forward and define the steps of the algorithm.

First of all, the points detected by the Leap Motion controller have to be normalized. Therefore a transformation will be done, in such way that the coordinates from the Leap Motion frame (which are measured in millimeters) will be converted in the range [0…1], such that the minimum value of the interaction box will map to the value of 0, while the maximum value of the interaction box will map to 1.

Finally, the values in the interval [0…1] obtained at the previous step must be mapped to the application's height and width (see Figure 4.4).

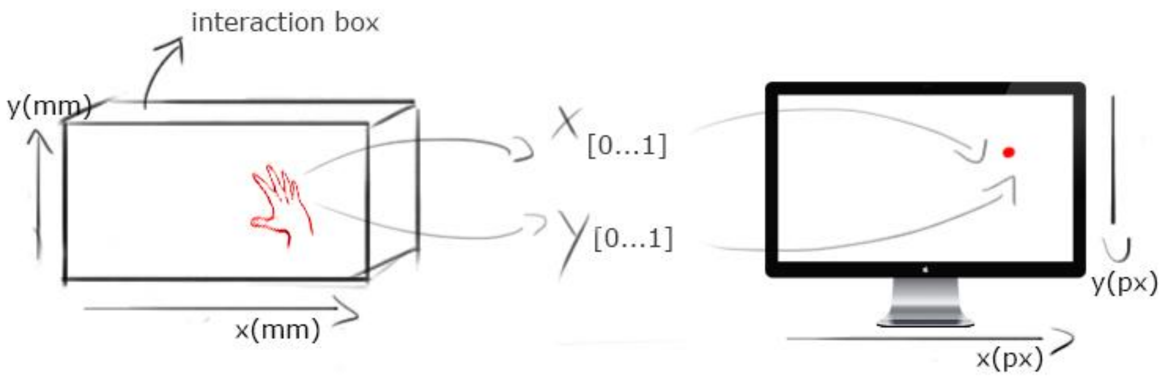


Figure 4.4 Mapping real world coordinates into digital world coordinates

Wrapping it up, this algorithm is essential in order to map the real world points, provided in millimeters by the Leap Motion controller, into the digital world, represented by pixels.

4.3.2. Converting pixels to latitude and longitude values

In order to perform some actions while using the application, often there is the case when pixels should be converted into latitude and longitude values.

First, we must understand how Google Maps works and what their coordinate system is. Whenever the Google Maps API needs to translate a location from the real world to a location on the map, it needs to first translate the latitude and longitude values into a "world" coordinate. This translation is accomplished using a map projection. Google Maps is using the Mercator projection [25].

The map at 0 zoom level is considered as a single tile. The world coordinates are relative to the pixel coordinates at zoom 0. Using the projection, latitude and longitude values are converted to pixel position on the base tile.

World coordinates in Google Maps are measured from the Mercator projection's origin (which is the northwest corner of the map at 180 degrees longitude and approximately 85 degrees latitude) and increase in the x direction towards east and in the y direction towards the south (see Figure 4.5). Because the basic Mercator Google Maps tile is 256x256 pixels, the usable world coordinate space is {0-256}, {0-256}.

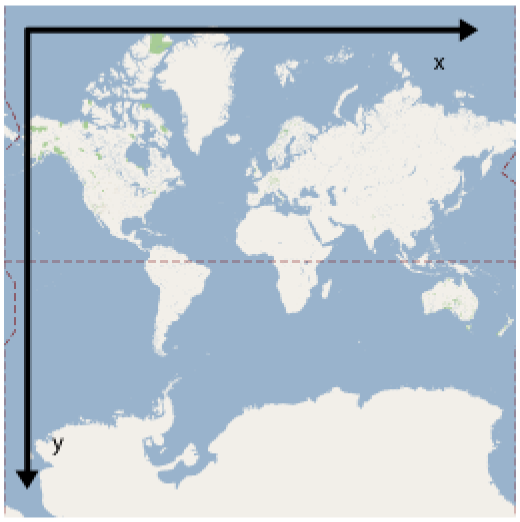


Figure 4.5 Google Maps World coordinates system (taken from https://developers.google.com)

Therefore, we can say that world coordinates reflect absolute locations on a given projection, but we need to translate these into pixel coordinates in order to determine the pixel offset at a given zoom level. This can be done with the following formula:

$$\text{pixelCoordinate} = \text{worldCoordinate} * 2^{\text{zoomLevel}}$$

Knowing all this, we must find a way to convert pixel coordinates (points from the screen) into Google Points and then into Google LatLng values.

Having the x and y screen pixel coordinates, and knowing the area on which the map is spread on the screen (the div containing the map), we must compute the percentage of the x and respectively y coordinated related to the div containing the map (i.e. a point on the center of the screen will have x percentage 50% and y percentage 50%, while a point at the rightmost point of the screen will have the x and y percentage 100%).

Having this computer, we move one step forward and we will retrieve the latitude and longitude values of the far extremities points of the visible map. The Google Maps JavaScript API V3 gives the possibilities to the developers to retrieve the latitude and longitude values of the north-east and south-west map boundaries. We need to convert this values (latitude and longitude, which are computed in degrees) into Google Points, using the Maps API.

At this point we are able to compute the new coordinates by applying a few computations and transformations; for longitude we will:

- subtract from the right boundary of the map the left boundary,
- multiply it by the percentage where the x coordinate was on the screen related to the container in which the map is placed, and
- add back the left boundary;

Finally, we need to convert the computed values, which are expressed at this moment as Google Points, using the Maps API, into Latitude and Longitude values.

4.3.3. Simulating map movement acceleration while moving the hand pointer to the far ends of the screen

In the project specifications it is written that when the user is in Google Maps mode, by spreading its hand and moving it on the x-y axis in the real world, will allow the map to move according to the real world movements. In the middle of the screen there should be a safe space, where, if the hand will point, there the map should not move. Also, while moving to the edges of the screen, the speed with which the map will move will increase, simulating acceleration.

In order to achieve this behavior I chose to split the application size (in my case the screen size) in three sections, both horizontally and vertically, each of them consisting of 33% of the screen (see Figure 4.6).

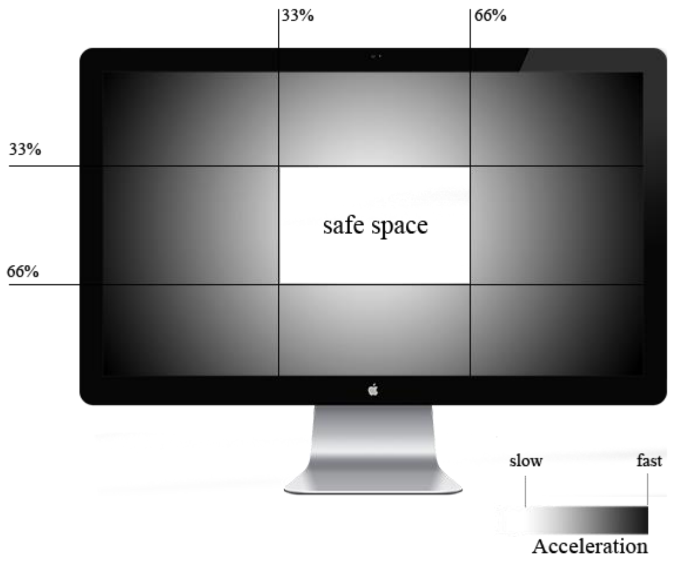


Figure 4.6 Screen division and acceleration areas

What I wanted is to have a piecewise linear function that maps the values of the screen pixel(x, y) into a duple representing acceleration (a_x, a_y), where a ϵ [-10, 10]. The mapping should be done in the following way:

- [0, 33%] → [-10, 0],
- [33%, 66%] → 0,
- [66%, 100%] → [0, 10].

Finding a linear function ***f*** that maps one interval ***[a, b]*** to another interval ***[c, d]***, in the following way ***f(a) = c*** and ***f(b) = d***, having ***a ≠ b***, can be done in a few simple steps.

First, we will define and apply the map f_1 that shifts the initial endpoint of the interval ***[a, b]*** to the origin. Therefore f_1 can be easily defined as $f_1(x) = x - a$, which will map the interval ***[a, b]*** onto ***[0, b – a].***

Next, we will define and apply the map f_2 that scales the interval ***[0, b – a]*** so that its right endpoint becomes ***d – c.*** Therefore $f_2(x) = \frac{d-c}{b-a} x$. Note that the length of the image interval is the same as the length of the interval ***[c, d]***, problem that will be addressed in the next step. Also this is the place where $a \neq b$ is required.

Finally, we will define and apply the map f_3 that shifts the interval ***[0, d – c]*** onto ***[c, d]***. In order to do so, f_3 should look like this: $f_3(x) = x + c$.

Wrapping it up, we will want to put all this maps together in order to achieve the result of mapping one interval [a, b] to another one [c, d].

$$f(x) = f_3(f_2(f_1(x))) = \frac{d-c}{b-a}(x-a) + c$$

Now that we have a linear function ***f*** that maps one interval ***[a, b]*** to another interval ***[c, d],*** in the following way $f(a) = c$ and $f(b) = d$, having $a \neq b$, we can put it all together and build our final function ***g(x).*** A graphic of the function can be viewed in Figure 4.7.

$$g(x) = \begin{cases} f(x),\ x \in [0, 33\%],\ a = 0,\ b = 33\%,\ c = -10,\ d = 0 \\ 0, \quad x \in [33\%, 66\%] \\ f(x),\ x \in [66\%, 100\%],\ a = 66\%,\ b = 100\%,\ c = 0,\ d = 10 \end{cases}$$

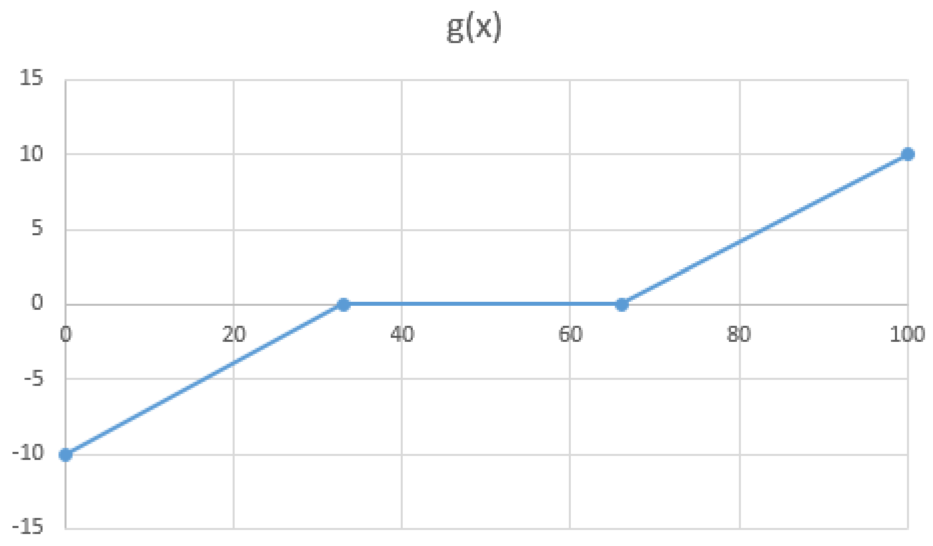


Figure 4.7 The graphic for function g(x)

Chapter 5. Detailed Design and Implementation

In this chapter I will document the developed application in such a way that it can be maintained and developed later. In the first part I will present the system architecture, going through all the modules and the way they interact with each other, describing the flow of events in the application, and later on I will continue with a detailed explanation of how does each component work and how it is possible for the developers to extend the functionality.

5.1. System architecture

As we can see from Figure 5.1, the system architecture is rather simple. The goal of the project was to keep it as simple as possible, because of performance issues.

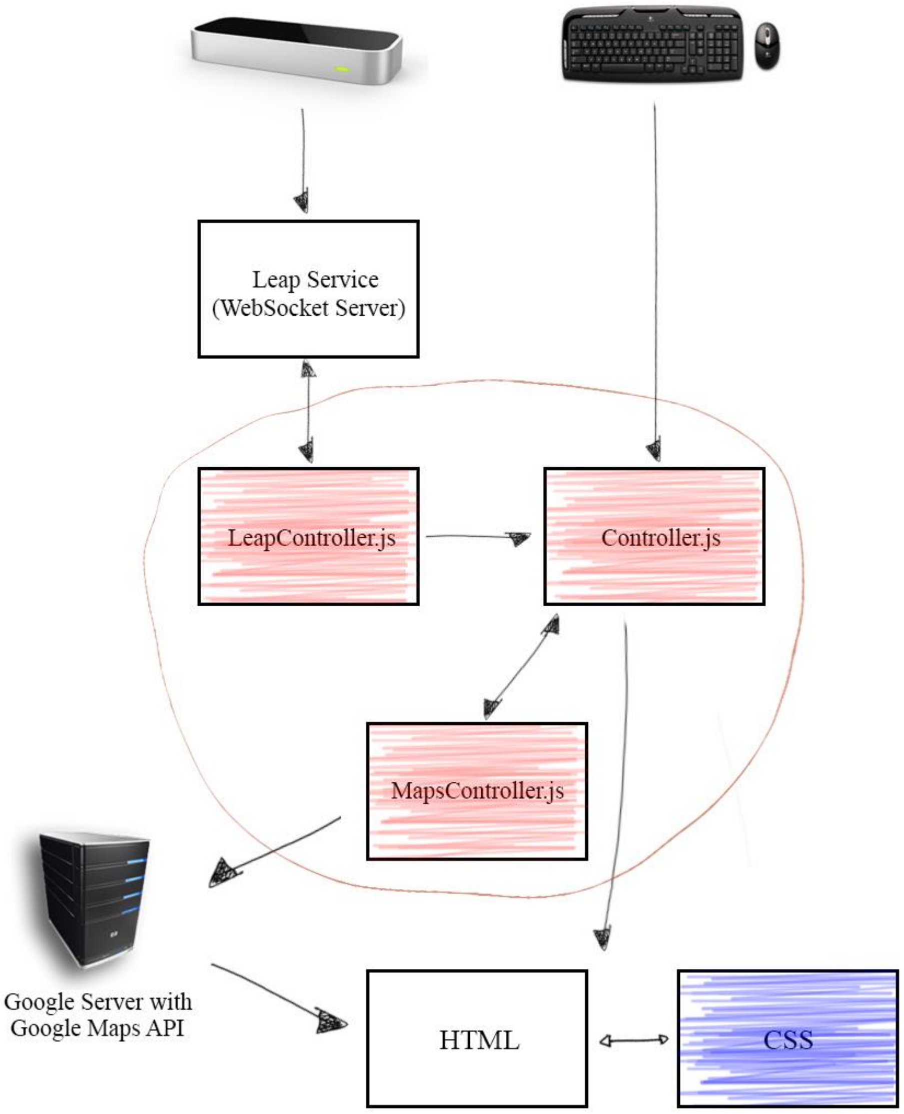


Figure 5.1 System architecture

The components that are colored are in the scope of this project and documentation. With the red color we have the JavaScript controllers, which all together form the JavaScript library that is the goal of this project, and as we can see from Figure 5.1, with blue color we have the CSS files that will closely interact with the HTML files that the developers will have.

5.1.1. Low coupling and high cohesion

One important thing that was taken into consideration while designing the system architecture was to have a very well defined domain model and a very good overview on the separation of concepts; the main *design pattern* that was taken into consideration was the low coupling and high cohesion.

I will take a moment to express what I mean by this; coupling, in this case, refers to the degree to which the different modules (in my case classes) depend on each other. The purpose was to keep the coupling as low as possible, therefore the modules should be as much independent in between as possible; trying to achieve low coupling, leads to the fact that the degree to which the elements of a module (in my case classes) belong together should be as high as possible, therefore related code and methods should be close to each other, binding all the related code together.

Having this in mind while developing the application I divided the code based on logic elements, the similarity of the domains and the similarities in functionality. Therefore, the outcome of this was to divide the code into three controllers (that will be explained in detail a bit later during this chapter).

The *LeapController.js* is the place where we will loop over the Leap frames and process the data (refine the data received, compute hand position, detect gestures).

The *Controller.js* is the place where all the application logic is handled.

The *MapsController.js* is closely responsible to interact with Google Maps and Street View through the help of the Google Maps JavaScript API V3.

5.1.2. Flow of events

To have a better overview about this project, I will describe the flow of events in the application.

As stated before, the main input source which is in the scope of the project is the data that comes from the Leap Motion controller. Of course, mouse and keyboard interactions were not completely eliminated.

First things first, our application will have to connect to the WebSocket server through which the Leap Motion controller sends frame data and will listen to the data that is received. Data will be processed and depending on the current state of the application a call to Google Maps’ API will be made.

More explanations are done in the following step-by-step description and as well as in Figure 5.2.

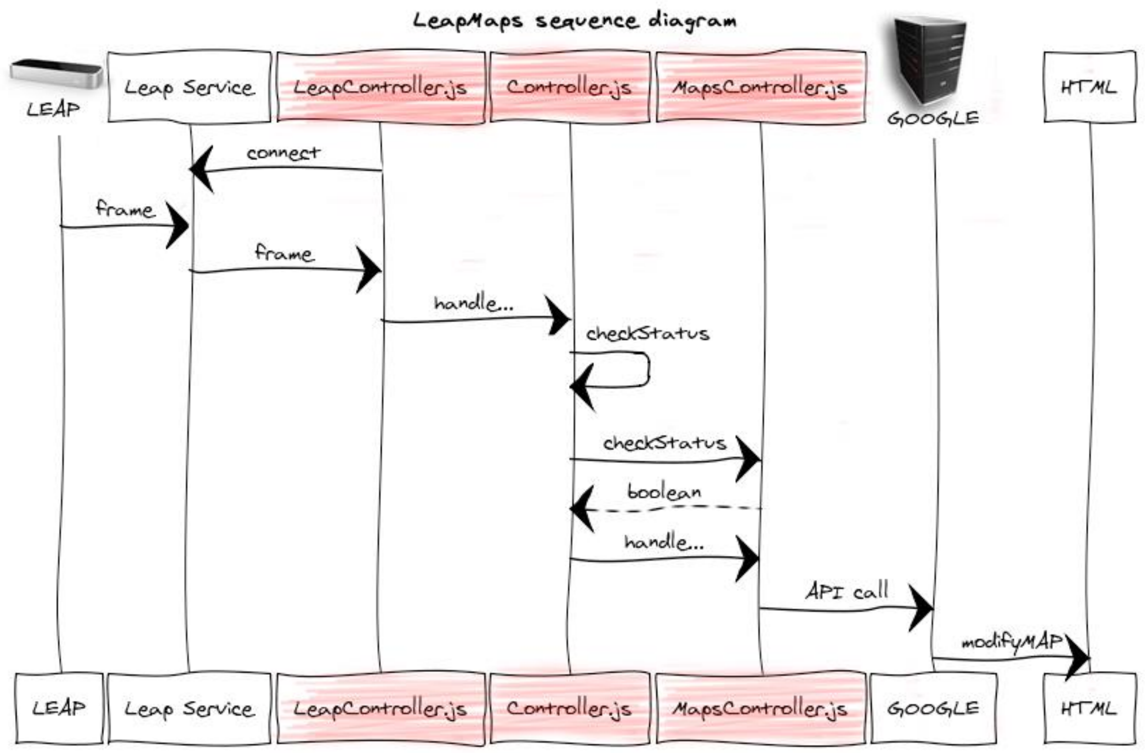


Figure 5.2 LeapMaps sequence diagram

LeapController.js

1. Connect to localhost:6437 (Leap Service) and listen to messages (through the help of LeapJS library)
2. Loop over the frames
3. Handle frame data:
    a. Detect hand position
    b. Detect gestures
        i. Refine the detected gesture
    c. Call the *Controller* to handle the events detected

Controller.js

4. Check for current status:
    a. If HELP screen is displayed (not shown in Figure 5.2)
        i. Handle gesture (not shown in Figure 5.2)
        ii. Alter HTML (not shown in Figure 5.2)
    b. If HELP screen is NOT displayed
        i. Check for current status of the Map object (call the *MapsController)*
        ii. Depending on status:
            1. Application logic (computations, algorithms, etc.)
            2. Call *MapsController* with appropriate data

MapsController.js

5. Depending on the call from the *Controller*
    a. Convert pixels to latitude/longitude values
    b. Move map

c. Zoom map
d. Switch between Google Maps and Street View
e. Move in Street View
f. Rotate the camera in Street View.

5.2. LeapController.js

The scope of the *LeapController.js* file is to connect to the WebSocket server through which the Leap Motion controller sends data, with the help of LeapJS, loop over the frames that are received from the server, detect hand position in real world and map it to the virtual world, detect gestures, refine the detected gestures, apply computations on the data received and lastly calling the *Controller.js* with appropriate data in order to handle the events and interact with the system.

In Figure 5.3 we can see the internal structure of the *LeapController* and the functions that are defined.

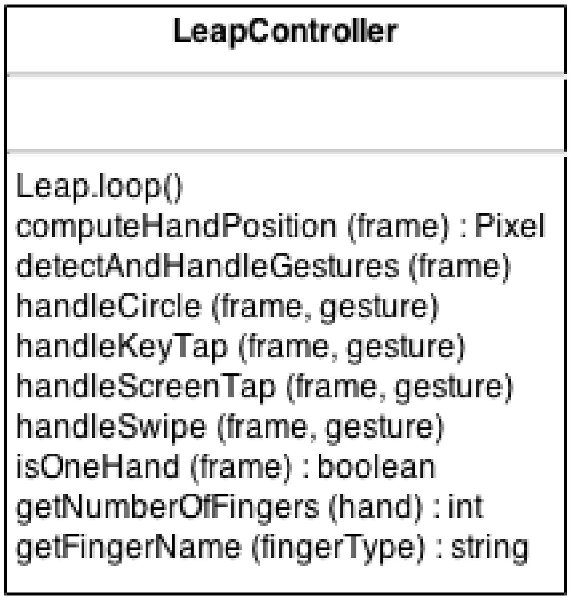

LeapController
Leap.loop()
computeHandPosition (frame) : Pixel
detectAndHandleGestures (frame)
handleCircle (frame, gesture)
handleKeyTap (frame, gesture)
handleScreenTap (frame, gesture)
handleSwipe (frame, gesture)
isOneHand (frame) : boolean
getNumberOfFingers (hand) : int
getFingerName (fingerType) : string

Figure 5.3 LeapController

Leap.loop() is the function where the LeapJS will connect to the WebSocket server to which the Leap Motion controller sends data and will loop over the frames received from the server.

The function *computeHandPosition* will map the position of the hand detected in real world coordinates by the Leap Motion controller (in millimeters) to digital coordinates (i.e. Pixel – x and y coordinates on the screen). The algorithm with which this mapping is achieved can be found under Chapter 4 - Mapping real world coordinates to virtual coordinates.

If a gesture will be present in the current frame, this gesture will be detected in *detectAndHandleGestures* function. Based on which gesture is detected, one and only of the following four functions will be called.

HandleCircle function is called when a circle gesture is detected. In the scope of this function is to detect if the circle gesture was clockwise or counter-clockwise; this is achieved by computing a dot product with the direction vector and the gesture normal vector. Finally, the *Controller* is being called with the appropriate direction of the circle gesture.

HandleKeyTap function is called when a key tap gesture is detected. The real world location where the gesture happened is mapped to the digital coordinates system, therefore a pixel is computed and the *Controller* is being called with this data.

HandleScreenTap function is called when a screen tap gesture is detected. The functionality of this function is quite similar with the one from the key tap gesture. The real world location where the gesture happened is mapped to the digital coordinates system, therefore a pixel is computed and the *Controller* is being called with this data.

HandleSwipe function is called when a swipe gesture is detected. A check is made in order to find out if the gesture's state is "stop"; if so, the direction of the swipe is computed here

- Horizontal
    - From left to right
    - From right to left
- Vertical
    - From top to bottom
    - From bottom to top

The *Controller* will be called with the correct direction of the swipe.

IsOneHand is a helper function that detects if in the current frame is one and only one hand detected.

GetNumberOfFingers function will iterate through a hand object and will count how many fingers are extended and will return that value.

The function *getFingerName* will return a string representation (the finger's name) of a finger based on its Leap Motion defined *fingerType.* Possible values:

- Thumb
- Index
- Middle
- Ring
- Pinky

5.3. Controller.js

The main application logic is done inside the *Controller.js* file. Here we subscribe some listeners to a few events, the events from the *LeapController* are received here, and the interaction with the *MapController* is done at this level. All checks regarding the state of the application at the current moment are done inside this controller.

In Figure 5.4 we can observe the structure of the controller, the local variables and the functions that are defined.

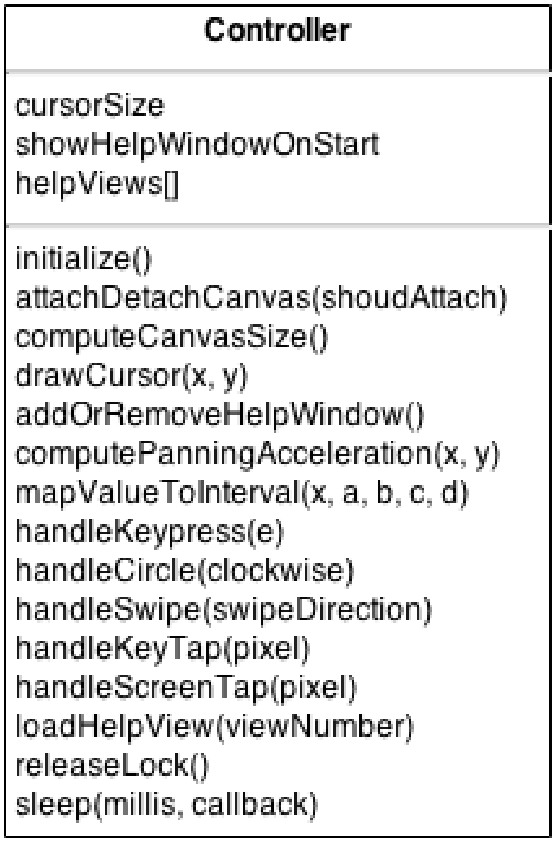


Figure 5.4 Controller

In the next part of this sub-chapter I will go through the functions that are defined and explain what they do and how they do it and also explain the local variables.

As we can see from Chapter 8, the User’s manual, before we initialize the library we can set and twist a few things.

When the hand is detected by the Leap Motion controller a cursor is shown on the screen; we can change its size by modifying the value of *cursorSize* (which by default is 5 pixels).

The next thing that is configurable is the possibility to choose to show or not to show the help screen when the application loads. This feature can be accessed by modifying the Boolean *showHelpWindowOnStart* which by default is set to *true.*

In the *helpViews* array, the path to the html views for the help screen are stored.

In order to enable the library we have to call the *initialize()* function (preferably on page load). Here is the place where event listeners are registered for

- the keyboard events (we will be interested especially in the *[h]* key, the arrows and *[esc]* key),
- resize events (each time the browser will be resized, we need to recomputed the canvas size where the cursor is drawn),
- document ready (when the document is ‘ready’ we will compute the allowed space in which the map can be displayed , then initialize the *MapsController* and finally show the help window if the option is not disabled by the user).

The *attachDetachCanvas* function will attach or detach a canvas where the custom cursor of the Leap Motion will be displayed and will hide the mouse. This function is automatically called by the *LeapController* when a hand is detected in a frame.

ComputeCanvasSize function will compute the size for the canvas that is used to display the cursor.

When the leap motion detects a hand, as stated before, a cursor will be displayed on the screen, this thing is done inside the function *drawCursor.*

The function *addOrRemoveHelpWindow* will interact with the HTML files and add or remove a modal help window on top of the map. Modal in this case means that the window has to be closed in order to perform other actions; the focus is only on the window opened and will remain as that until the modal window is closed (removed).

ComputePanningAcceleration function maps the hand position, given by the arguments x and y (representing pixels on the screen), onto the [-15, 15][5] domain which represents an offset from the current position of the center of the map (the center of the map being 0). After the new position is computed, this function calls the *MapsController* in order to pan the map to the new destination. Based on the offset of the hand position (cursor) from the center of the screen, the map is panned with less pixels or more pixels, simulating therefore an acceleration in the movement. As o note, it is wise to add that there is a safe space in the center of the screen, which is defined by a ratio of 1/3 of the width and height of the canvas in which the map is displayed. If the cursor will be placed in the safe space, the map will not pan.

The next function defined, *mapValueToInterval*, is a helper function that allows the feature from the previous function to be achieved. Given a value, x (the value for which the function should be computed), *a* and *b* defining the domain [a, b], and *c* and *d* defining the range [c, d], the function will return the value f(x), where the function *f* is a linear function that maps the given value from the given domain [a, b] to the given range [c, d]. The algorithm with which this is achieved can be found under Chapter 4 - Simulating map movement acceleration while moving the hand pointer to the far ends of the screen.

HandleKeyPress is registered as an event handler for the keyboard events. In this function we will define what the following keys does:

- *[H]* key - show/hide window screen,
- *[ESC]* key – hide window screen
- *[←] key (left arrow)* – goes to the previous help screen
- *[→] key (right arrow)* – goes to the next help screen.

[5] It was initially thought of [-10, 10], mentioned before, in chapter 4, but in practice it was proved that this was not enough, so I had to stretch the interval.

HandleCircle function will handle a circle gesture. This function will be called from the *LeapController*. Depending on the current application state, the function can zoom in or out the map, or to move the current position (switch the panorama) in street view.

HandleSwipe function is called from the *LeapController*. Depending on the application state, this can

- Switch between help screens, while the help screen is on
- Rotate the camera 360 degrees, while in Street View (to check what the current state of the map is, the *Controller* will call a method to *MapsController*).

HandleKeyTap adds or removes a marker from the map. At this moment this function is out of scope and not used. The call from *LeapController* is never done.

HandleScreenTap function will handle a screen tap gesture detected in *LeapController*. This function will delegate the responsibility to the *MapsController* to switch between Google Maps and Street View.

The function *loadHelpView* will load a HTML file from the path *views/help/* into the modal help window that is displayed over the map.

ReleaseLock and *Sleep* functions were introduced because of a bug. When performing a swipe gesture, in order to switch from a help screen to another, the Leap Motion controller will notify the detection of a swipe gesture in every frame. This means that in the time when the end user will perform a swipe gesture, most probably at least 20 swipe gestures will be reported. Therefore I needed to implement a mechanism to block the switching of help views for a predefined time. At this moment, this is the process flow:

- Detect swipe gesture
- Block the possibility to switch the help screen for 250ms by calling the *sleep* function
- Call *releaseLock* function after 250ms (this is done automatically, because the *sleep* function that I implemented allows the developer to specify a callback function) and enable switching the help screens again.

5.4. MapsController.js

The interaction between the library that I created and the map itself it's done at this level. In the *MapsController* we have defined some local variables related to the map and some basic settings that can affect the map, as well as some functions that make the interaction with the map possible. In order to create, render and modify the map I used the Google Maps JavaScript API V3.

In Figure 5.5 we can see the internal structure of the *MapsController*.

In the next part of this sub-chapter I will go through the functions that are defined and explain what they do and how they do it and also explain the local variables.

Before initializing the main *Controller*, we can modify some variables that will affect how the map will look like. As we can see from the picture below, a *map* is defined as local variable; this holds a Google Map object and should not be modified. However, at this moment when the map is rendered, the center of the map is placed in Cluj-Napoca; this can be modified by modifying the *lat* and *lng* variables with the desired location. Another thing that can be modified is the *zoom* level at which the map is rendered.

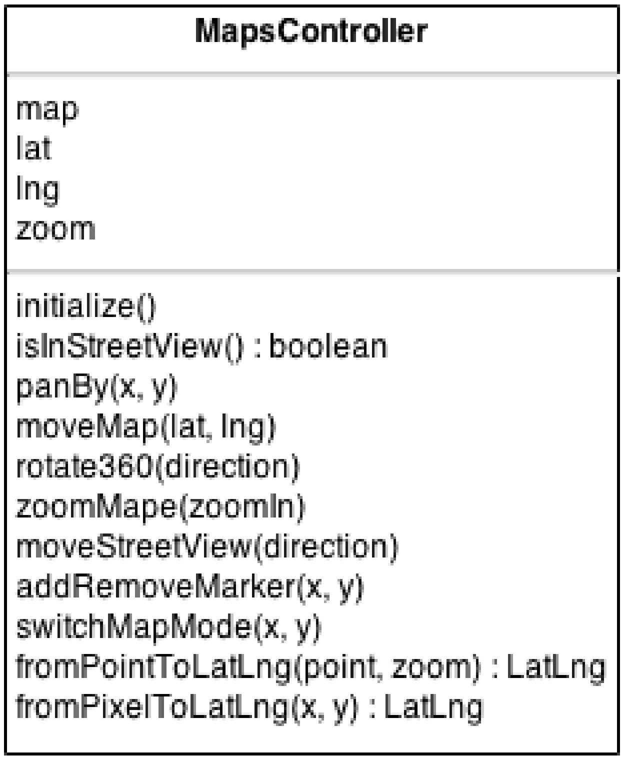


Figure 5.5 MapsController

The function *initialize()* is called from the *Controller* at the moment of initializing. In this function the map is created in the *map-canvas* div of the html with the properties defined above.

IsStreetView function returns true if the current state of the map is Street View, or false otherwise (i.e. the map is in Google Maps mode).

The function *panBy* changes the center of the map by the given distance in pixels. If the distance is less than both the width and height of the map, the transition will be smoothly animated.

The function *moveMap* will move the center of the map at the given position by the new latitude and longitude values.

Rotate360 function will rotate the map by the given direction.

The function *zoomMap* can be used to control the zoom level of Google Maps or Street View. Therefore you can zoom in or zoom out by passing a Boolean as argument. If the Boolean value is true, than the map will zoom in, otherwise it will zoom out.

While in Street View the user can navigate through panoramas by performing circle movements. This functionality is implemented in the function *moveStreetView*. A Boolean is passed as argument; if the value is true, the panorama which is in from of the user will be loaded (if available), if false, the panorama available at 180 degrees will be rendered (the panorama from behind). The way in which the next position of the panorama is chosen is based on the available panoramas and the current heading of the camera. Basically the list of available panoramas is parsed and if the current heading of the camera matches the heading of an available panorama (± 40 degrees) the panorama will be switched.

AddRemoveMarker is currently out of scope. It should add or remove a marker given the position on the screen. A pixel coordinate is passed as argument, the pixel is converted into LatLng object and a marker should be added or removed at that specific position.

SwitchMapMode function will switch between Google Maps and Street View. The function accepts as argument a pixel coordinate. When switching from Google Maps to Street View, the pixel coordinate is converted into LatLng object and a panorama will be rendered at that location; if no panorama is available at that specific position or at a radius of 150 meters, an alert message will be displayed in the UI informing the end user this. While in Street View, if this function is called, the Street View's visibility will be set to false and therefore Google Maps will be available again.

The function *fromPointToLatLng* is a helper function that converts a Google Point object into a Google LatLng object. This is done through the help of Google Maps API.

The function *fromPixelToLatLng* converts the given pixel on the screen (defined by x and y values) into a Google LatLng object. The algorithm with which this is achieved is described under Chapter 4 - Converting pixels to latitude and longitude values.

5.5. General guidelines

This sub-chapter is designed for the developers and has the aim to give general guidelines about the possible further development.

In the process of extending the functionality of the application, two things have to be respected.

5.5.1. Flow of events

The flow of events was designed in such a way that all application logic and decisions should be done in the main *Controller*.

All events that are related to the Leap Motion, such as frame processing, refinement of data, gesture detection and so on, should be treated in *LeapController*. The *LeapController* should NOT interact directly with the *MapsController* (or even with the map itself) or with the HTML files; after the detection of events and processing the data, it is the *Controller*'s responsibility to decide what to do in that situation.

The *MapsController* is designed to interact directly with Google Maps API. Computation related to maps elements should be done here. No other application logic, such as decision making, interaction with the Leap Motion data or HTML files should be done here.

5.5.2. Separation of concerns

This second point is very closely related to the previously one. Theoretically speaking, a concern is a set of information that affects the code of the application.

Therefore, the main idea is to separate the code in different components in such way that if something changes (such as Leap Motion's API or Google Maps API) the required changes will be as minimal as possible. Having this separation of concerns in place, makes the application a modular one.

The value added of this separation of concerns is simplifying the development and maintenance of the library. When concerns are well separated, individual sections can be developed and updated independently. In such way it is much more easily to later improve or modify one section of code without having to know all the implementation details of other sections of code, and without having to make corresponding changes to those sections.

The intent was to keep the architecture as simple as possible for further development and maintenance of the library. It is really important that the two points from above will be taken into consideration while developing the application and extending the functionality.

Chapter 6. Testing and Validation

In this chapter I will present an analysis about Leap Motion's precision and reliability speaking in terms of gesture recognition accuracy, as well as an analysis about the performance of the proposed and developed library (I will discuss about the size of the project, frames per second and browser compatibility).

6.1. Leap Motion precision and reliability

6.1.1. Gesture recognition

As we can see from the bibliographic research chapter, the Leap Motion Controller has very good precision and accuracy, but when it comes to Gesture recognition, the Leap software has some flaws.

The Leap Motion software recognizes certain movement patterns as gestures which could indicate a user intent or command. The following movement patters are recognized by the Leap Motion software:

- *Circle* (a single finger tracing a circle),
- *Swipe* (A long, linear movement of a finger),
- *Key Tap* (A tapping movement by a finger as if tapping a keyboard key) and
- *Screen Tap* (a tapping movement by the finger as if tapping a vertical computer screen) [26].

The software lacks accuracy in detecting the right movement at the right time, but does not lack of accuracy and precision in terms of hand positioning. To illustrate this issue manual tests were done. I, as a user, I intended to perform 50 gestures of each type mentioned above, and I noted down the results that the Leap Motion has given and have built charts from it.

NOTE: The results can vary from one user to another. The Leap software does not have a learning algorithm implemented, but the user can learn the patterns that the Leap software recognizes and try to mimic them.

6.1.1.1. *Circle gesture*

A circle gesture is defined as a single finger tracing a circle. For this test, I intended to perform 50 circle gestures (both clockwise and counter-clockwise) positioning my hand in different parts of the interaction box above the controller. Out of 50 gestures performed, 45 were detected as *circles* and 5 were detected as *key taps* (see Figure 6.1).

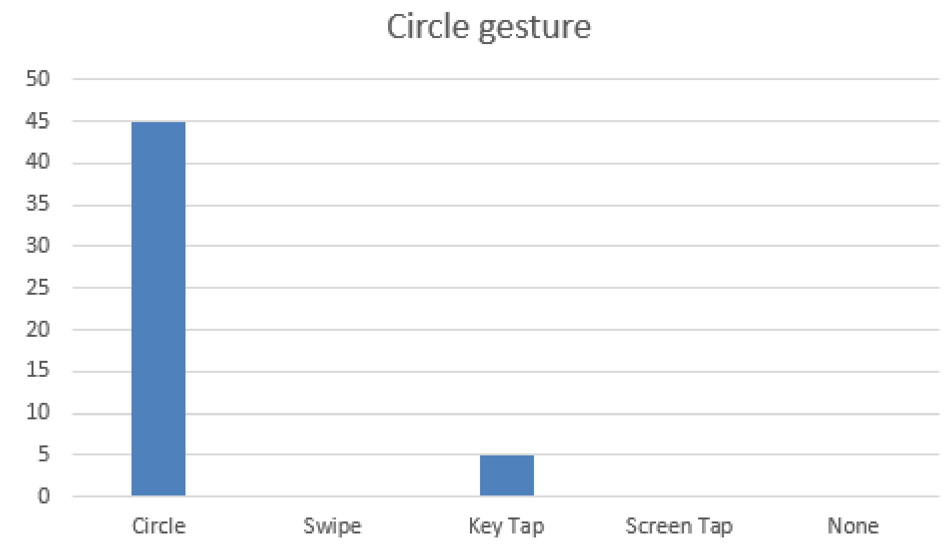


Figure 6.1 Circle gesture statistics

6.1.1.2. *Swipe gesture*

A swipe is defined as a long linear movement of a finger or hand. This gesture is highly used in mobile application industry (e.g. unlocking the screen, navigating through screens or photos, etc.).

Out of 50 gestures performed, only 28 were detected as swipe gestures, 10 were not detected at all, and the rest were detected as either key taps or circle gestures (see Figure 6.2).

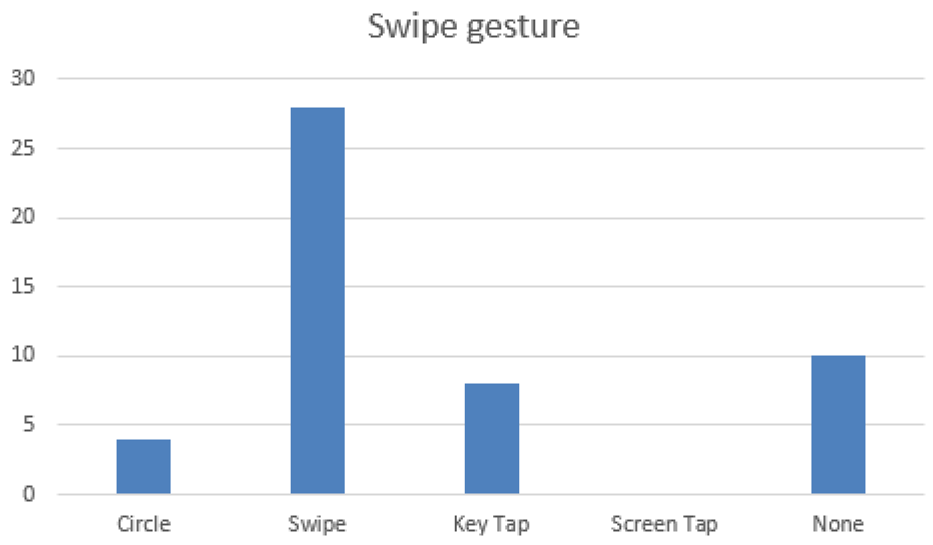


Figure 6.2 Swipe gesture statistics

6.1.1.3. Key Tap gesture

The Key Tap gesture is defined as a movement representing a finger tapping downwards and back upwards as if tapping a keyboard key. As we can see from the image below (Figure 6.3), the Key Tap gesture is the gesture which is the most easily detected by the Leap software, out of 50 gestures performed, 48 were registered as key taps and two gestures were not recognized.

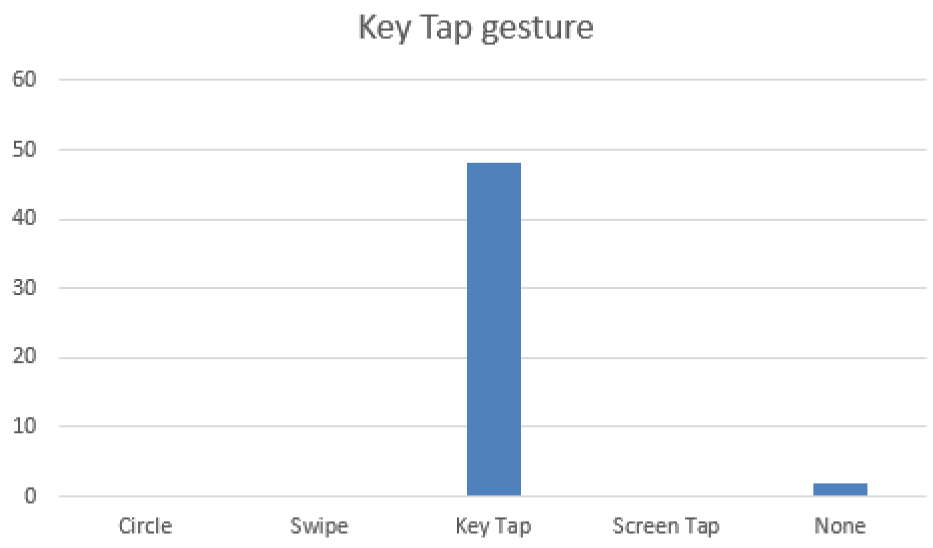


Figure 6.3 Key Tap gesture statistics

6.1.1.4. Screen Tap gesture

A tapping movement by the finger as if tapping a vertical computer screen (moving forward on the z axis and then backwards) describes a *Screen Tap* gesture.

I have to mention that it took me quite some time to learn the gesture, at the very beginning of the implementation the project, I had problems reproducing the gesture, most of the times the Leap software not recognizing gesture interaction while trying to perform a screen tap.

Therefore, in the Figure 6.4, we can notice that out of 50 gestures performed, 35 were detected as screen taps, 10 gestures were not detected, three of them were classified as being circle gestures and one as a key tap.

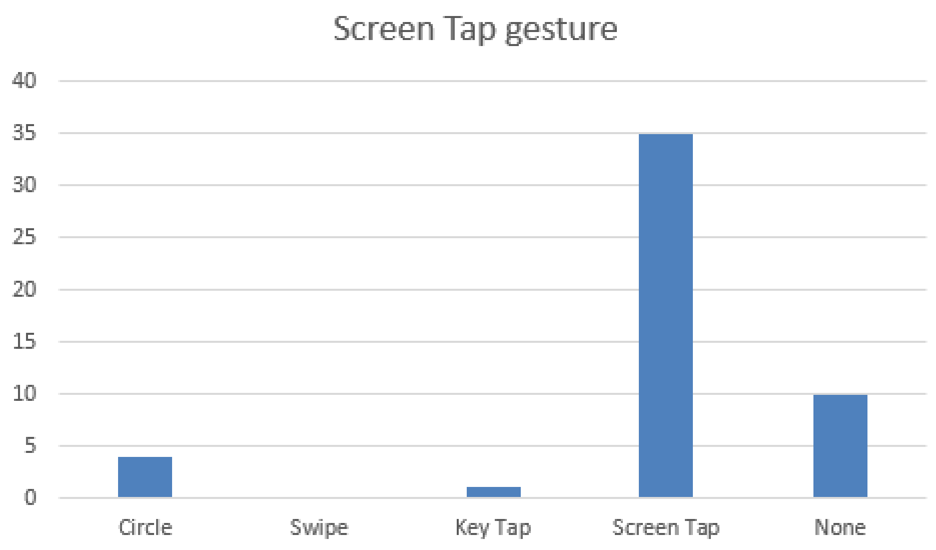


Figure 6.4 Screen Tap statistics

In conclusion, after I have performed the manual tests, summing it all up, in 78% of the cases the gesture was correctly detected and classified, 11% of times the gesture was misclassified and also 11% of the times the gesture was not detected at all (see Figure 6.5)

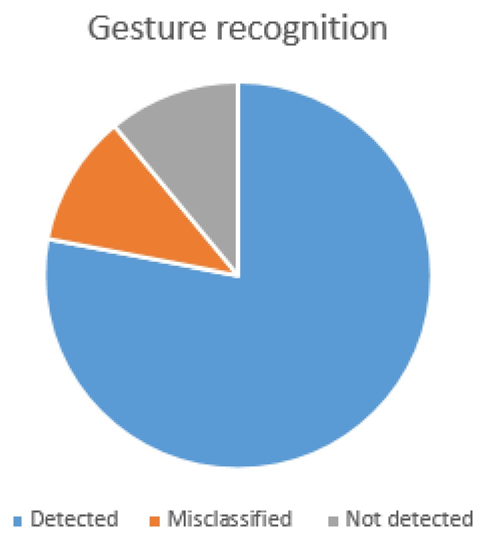


Figure 6.5 Gesture recognition statistics

I would say that the Leap Motion Controller is a very good option for detecting the spatial position in terms of 3D in the interaction box that is available above the controller itself, but not quite such a good option, at the moment when the document is written, for detecting movement patterns. Even if 78% is a quite high number, there still are approximately 2 times out of 10 when the controller, and implicitly the product developed will not act as it should. The lack of gesture recognition (11%) is not quite concerning from my point of view, but detecting other gesture than performed in 11% of the times is quite concerning because there could be other implementations available for different gestures and then this will result in an unexpected and unwanted behavior of the product developed.

The Leap Motion Controller's lack of artificial intelligence involvement in the software (i.e., not being able to adapt with the user interaction and to learn its behavior) could force the developers to give up at all the gesture implementation and integration in the product that is being developed in order to avoid performance issues and unwanted behavior that might occur because of the lack of accuracy in detecting the right moves at the right time.

6.2. Performance

6.2.1. Library size

First of all I would like to discuss about the size of the project. The application was designed in such way to be included as a library over the Google Maps.

All the JavaScript files (my files together with the leap.js library) together with the CSS files sum up to 379. At the first sight you would say that this is an acceptable size, but in the case of web development the best practices recommend to minify the sources (JavaScript files, CSS files, HTML files) in order to obtain the smallest size possible. The smaller the size, the faster the load will be.

UglifyJS2[6] was used to minify the sources by removing the spaces, comments, unused code, unreachable code, converting the if-else statements into one line statements and so on; the final size of all the libraries together is 87 KB.

Using a simple online download calculator we can see that at an average speed of 20 Mbps (which matches the average internet speed around the globe) the files will be downloaded under 1 second.

6.2.2. Frames per Second

When using such an application the intent is to have smooth animations in the UI that are appealing and could be easily followed without disturbing the user's attention. The transactions from one position to another on the map, as well as rotating the camera in Street View should be smooth and without any lag.

Because I could not find any automated tools that can provide an average FPS (frames per second) over a predefined period of time I used Google Chrome's FPS tool and measured the activity over one minute of interaction with the application in both Google Maps and Street View, first without the library included (therefore using mouse and keyboard interaction), then with the library included (using the Leap Motion controller).

[6] UglifyJS2: https://github.com/mishoo/UglifyJS2

During the monitoring phase of the FPS variance 20 samples were taken and graphs were plotted.

First test was done while navigating in Google Maps, as we can see from Figure 6.6, the values were quite close. We can see plotted with the blue line the FPS samples taken while using the mouse and keyboard as input devices, while with orange we can see the samples taken while interacting with the Leap Motion controller.

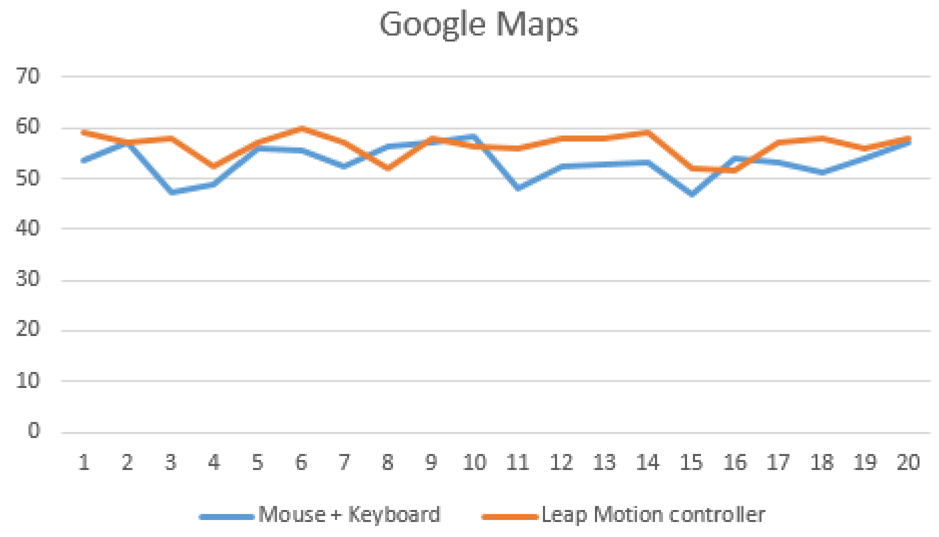


Figure 6.6 Google Maps FPS variance

Even if the values are quite similar and close to each other, we can notice a slightly improvement with the Leap Motion implementation. This can be justified by the fact that I used another implantation than Google uses for moving the map. Moreover, the WebSocket service through which the Leap Motion controller communicates with the browser sends information, in form of *frame* JSON messages, as often as every 10ms. The browser only repaints the page every 16ms (60Hz) at best. The LeapJS controller generates animation frames based of the browser's *animation.loop*, therefore the "request" to render the page will not occur between other requests, generating non smooth transactions between frames, instead will "request" to render the page at the same time when the browser should.

The second test was done while navigating in Street View (see Figure 6.7). As in the previous graphic we can notice that the blue line represents the FPS variance while using the mouse and the keyboard, while with orange the values from the Leap Motion controller are plotted. In this case we can notice that the difference between the two cases presented is quite big.

Even if, from my point of view, rotating the camera through swipe gestures is not that accurate, smooth and therefore fun to use than while using the mouse, we can clearly see from this test that this technique implemented in my work that rotates the camera using the Leap Motion controller provides higher frames per second than the solution given by Google.

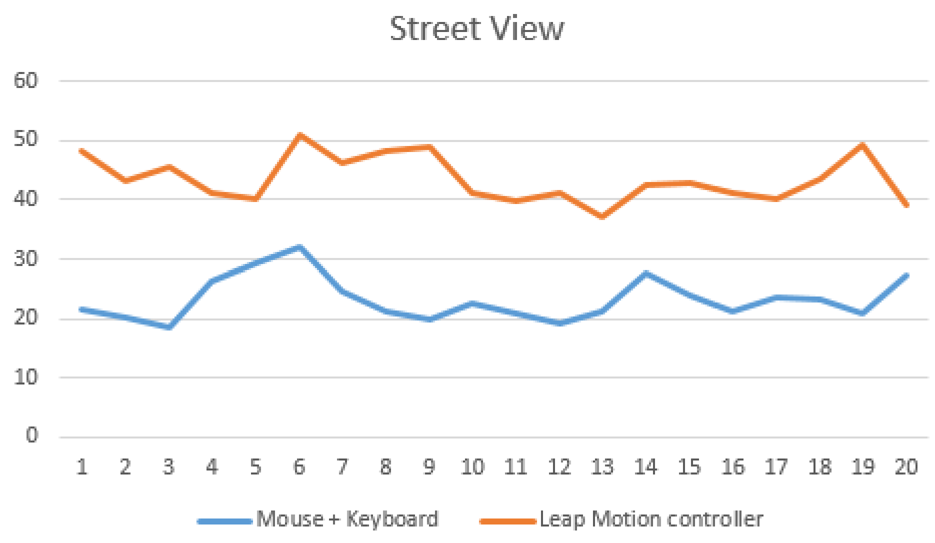


Figure 6.7 Street View FPS variance

Wrapping it up, we can clearly see from this two FPS tests that using the implementation provided in this project, we can obtain a slightly higher performance, from the frames per second points of view.

6.2.3. Browser compatibility

As stated in the non-functional requirements the software should be compatible to all modern browsers.

The application was tested on:

- Google Chrome 36.0
  - No compatibility issues were found
- Mozilla Firefox 31.0
  - No compatibility issues were found
- Internet Explorer 11.0
  - When running the application a warning message appears that says that “Internet Explorer has restricted this webpage from running scripts or ActiveX controls that could access your computer.”
  - The user can continue by clicking “Allow blocked content”
    - The initial help window cannot be closed. It will constantly appear on the screen, even if the application will act like it is missing.

Chapter 7. User’s manual

This chapter will present the system requirements, both from hardware and software point of view, for the application to run within the standards, a manual for developers, describing how to include and use the library, as well as a manual for the end users which explains how to use the application and describes its features.

7.1. System requirements

In order to run the application you will need:

- A computer with Internet connection and USB 2.0 port,
- A Leap Motion controller,
- Leap Motion software installed (v2.0 or better)
- 2 GB RAM (or more).

Depending on the computer manufacturer or operating system you can consult Table 7.1.

Table 7.1 Minimum system requirements[7]

Mac	Windows	Linux
Mac OS X 10.7 Lion (or newer)	Windows 7 (or newer)	Ubuntu 12 (or newer)
Intel Core i3 (or better)	AMD Phenom II or Intel Core i3 (or better)	AMD Phenom II or Intel Core i3(or better)

[7] Leap Motion controller minimum requirements: https://www.leapmotion.com/setup

7.2. Manual for developers

This sub-chapter is intended to be used by developers. It contains instructions on how to include and use the leapmaps JavaScript library and what known issues exists at this moment.

7.2.1. How to install

In order to benefit of the library's features one must include it, together with the CSS files in your html file where the Google Map is rendered. Besides this two files that need to be deployed on the server and included in the html, additional files should also be deployed.

The **leapmaps.zip** contains all the files that you need:

- leapmaps.min.js or leapmaps.dep.js
- leapmaps.css
- /views/ directory
- /img/ directory

7.2.1.1. JavaScript library

The leapmaps JavaScript library is available under two versions.

1. The library without dependencies

```
<script src="js/leapmaps.min.js"></script>
```

Note: if you chose to include the leapmaps library without the dependencies, you will have to manually include jQuery and LeapJS (v2.0 or higher).

2. The leapmaps library together with dependencies to jQuery and LeapJS.

```
<script src="js/leapmaps.dep.min.js"></script>
```

7.2.1.2. CSS file

The CSS file must be also included:

```
<link rel="stylesheet" type="text/css" href="css/leapmaps.css">
```

7.2.1.3. Other dependencies

The two following directories should be deployed on the server

- /views/ (which contains under /help/ four html files representing the different views for the help screen)
- /img/ (which contains the images that are used in the help screen)

Once you deploy all the files found in the leapmaps.js on the server and include the JavaScript and CSS files in your application you are good to go. Please consult the next sub-chapter in order to find out how to use the library.

7.2.2. How to use

First of all I would like to stress some points that you should know when using Google Maps on your web applications.

In order for your application to be able to load a Google Map, you first need to obtain an API Key. Having an API key does not only allow you to render the map, but also allows you to monitor your application's Maps API usage.

To create an API key follow this steps:

1. Visit the APIs Console at https://code.google.com/apis/console and log in with your Google Account.
2. Click the Services link from the left-hand menu.
3. Activate the Google Maps JavaScript API v3 service.
4. Click the API Access link from the left-hand menu. Your API key is available from the API Access page, in the Simple API Access section. Maps API applications use the Key for browser apps [27].

Other important things that you need to know before starting to use google maps are its usage limits[8] and terms of service[9].

In order to include the Google Maps JavaScript API V3 paste the following lines in your html file.

```
<script type="text/javascript"
    src="https://maps.googleapis.com/maps/api/js?key=API_KEY"></script>
```

On the page where the Google Map will be placed and rendered please follow this coding standards and names presented below:

```
<div id="wrapper">
    <div id="map-canvas"></div>
</div>
```

In order to initialize the Controllers involved in the library, to add event listeners and to initialize the Leap Motion controller, include the following lines in the head of the html file:

```
<script>
    Controller.initialize();
</script>
```

At this moment, when loading the page, the library will force the rendering of the map centered at the latitude and longitude of Cluj-Napoca at a zoom level of 10.

If you want to change this you can do it by accessing lat and lng local variables from the MapsController, as well as the zoom variable. Do this before the initialization of the main Controller.

[8] Google Maps Usage Limits: https://developers.google.com/maps/documentation/javascript/usage#usage_limits

[9] Google Maps Terms of Service: http://maps.google.com/help/terms_maps.html

```
MapsController.lat = YOUR_LAT_VALUE;
MapsController.lng = YOUR_LNG_VALUE;
MapsController.zoom = YOUR_ZOOM_VALUE;
```

When the page is loaded, the application will show a help information screen with instructions on how to use the application. If you want to disable this feature, before initializing the Controller, paste the following line:

```
Controller.showHelpWindowOnStart = false;
```

7.2.3. Known issues

While using Google Chrome, during the development phase, if the application is running locally (therefore is not deployed on a server and accessed remotely), HTML templates for the help window are not loaded.

An error message can be observed in the console: "XMLHttpRequest cannot load file *[filename]*. Origin null is not allowed by Access-Control-Allow-Origin.". This is a known issue[10].

In order to avoid this issue, the user can passing an argument to Google Chrome at runtime. Use: chrome.exe --allow-file-access-from-files.

[10] Google Chrome's known issue of accessing local files from another local file: http://stackoverflow.com/questions/4208530/xmlhttprequest-origin-null-is-not-allowed-access-control-allow-origin-for-file

7.3. Manual for end users

This sub-chapter is intended to be used by end users. It contains instructions on how the application works and what interactions are needed in order to have full control of Google Maps and Street View through the help of the Leap Motion controller.

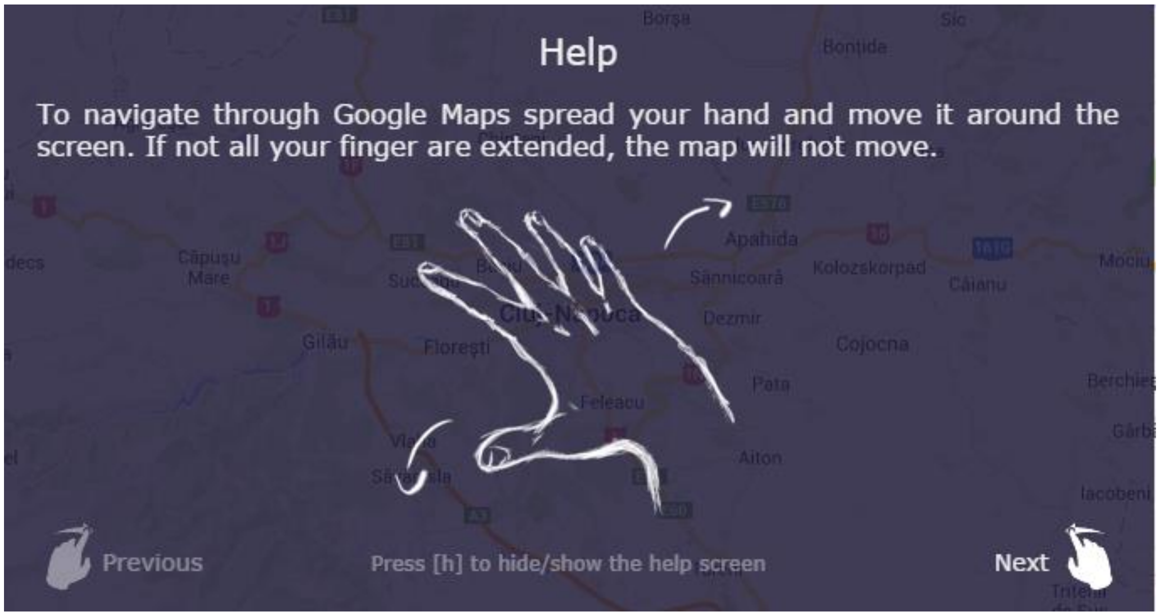
Once you will run the application the first thing that you will notice is that a Google Maps screen is loaded and on top of it, a help screen is displayed.

The help screen contains multiple pages with valuable information about how to use the application. You can navigate through pages using swipe gestures. Performing a swipe gesture from right to left will move to the next frame; moving back to the previous frame can be done by performing a swipe gesture from left hand side to the right hand side.

The help screen can be always easily closed or accessed by pressing the *H* key from the keyboard.

The first help screen (see Figure 7.1) will let you know what the basic movements that need to be done in order to navigate in Google Maps are. Spreading the hand and moving on the x-y axis above the Leap Motion controller will move the map in the direction where the pointer on the screen is (it will change the map's center).

There exists a safe space in the center of the screen, where if the hand will be placed, the map will not move. As the pointer (hand) moves to the edges of the screen, the map will start to move together with it. The distance of the pointer related to the center and edges of the map will affect the acceleration of the animation moving the map.


Help

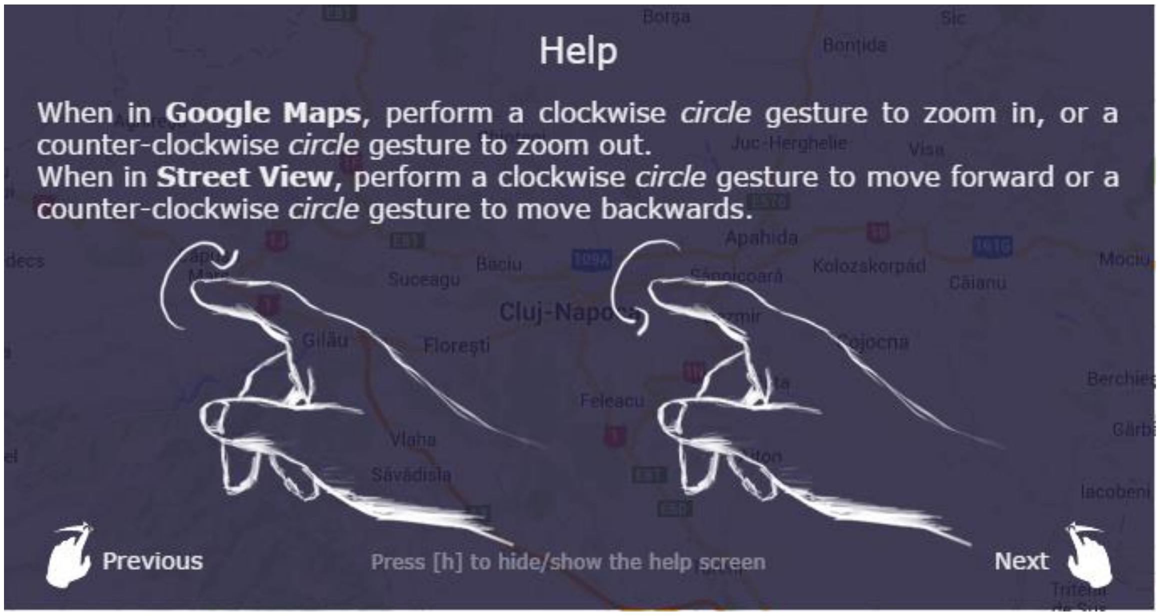
To navigate through Google Maps spread your hand and move it around the screen. If not all your finger are extended, the map will not move.

Figure 7.1 First help screen. Navigation through Google Maps

The next help screen (see Figure 7.2) will let the user know what a *circle* gesture is, how it can be performed, and how that will affect the application depending on the status of the Google Map.

If the user is navigating in Google Maps the circle gestures will zoom in or zoom out the map, while in Street View a clockwise circle gesture will move the camera forward and a counter-clockwise circle gesture will move the camera backwards.

Note: In order for the gestures to be properly recognized, extend only one finger when performing circles.



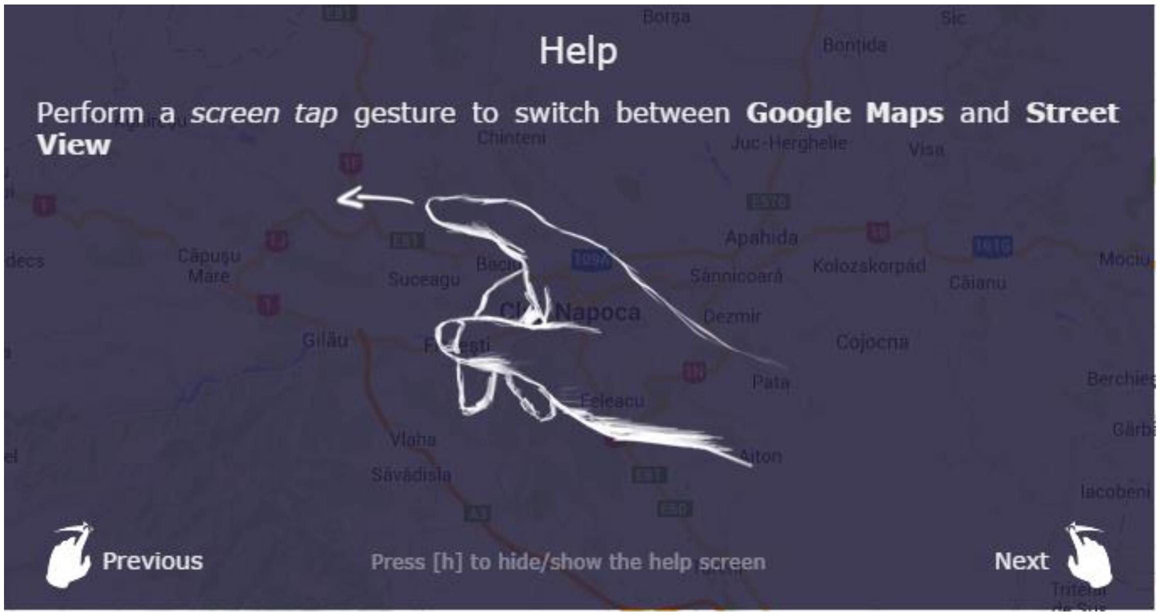
Help

When in **Google Maps,** perform a clockwise *circle* gesture to zoom in, or a counter-clockwise *circle* gesture to zoom out.
When in **Street View,** perform a clockwise *circle* gesture to move forward or a counter-clockwise *circle* gesture to move backwards.

Figure 7.2 Second help screen. Circle gestures – zooming in and out the Google Map and moving forwards and backwards in Street View

The third help screen will present the way a user can switch between Google Maps and Street View. To switch between Google Maps and Street View, the end user will perform a screen tap gesture. A tapping movement by the finger as if tapping a vertical computer screen (moving forward on the z axis and then backwards) describes a Screen Tap gesture (see Figure 7.3).

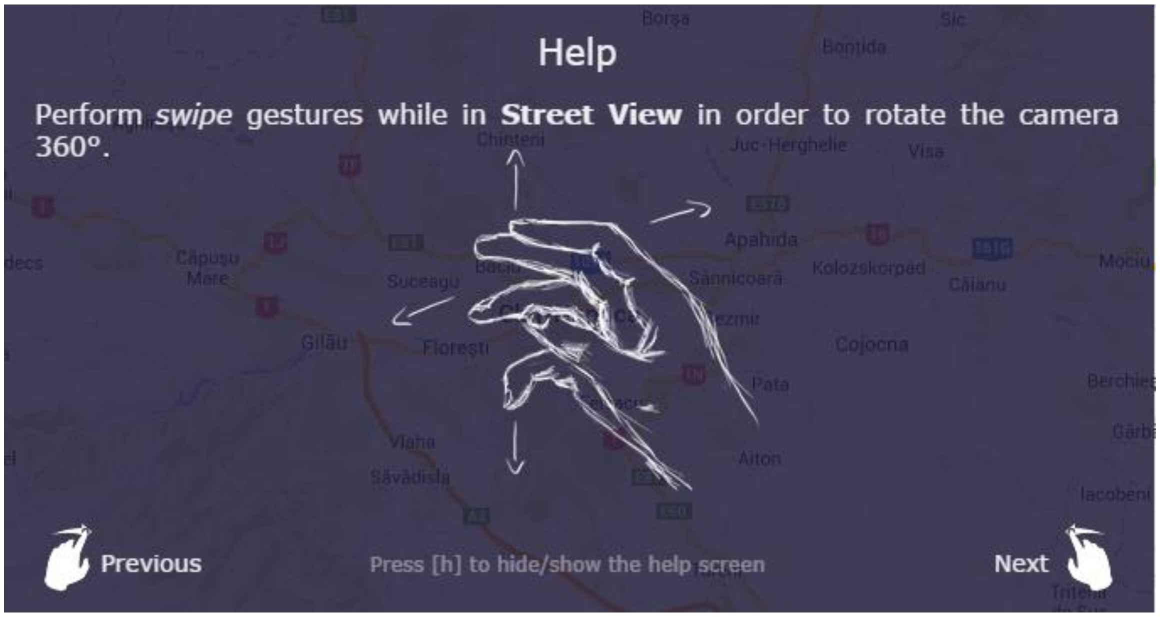
Note: when performing a screen tap gesture only one finger of the hand should be extended.


Help

Perform a *screen tap* gesture to switch between **Google Maps** and **Street View**

Figure 7.3 Third help screen. Switching between Google Maps and Street View

The last help screen (Figure 7.4) will let you know what are the movements that need to be done while in Street View in order to rotate the camera around. Swipe gestures are better recognized if more than one finger is extended.


Help

Perform *swipe* gestures while in **Street View** in order to rotate the camera 360°.

Figure 7.4 Last help screen. Rotate the camera 360° while in Street View

Chapter 8. Conclusions

In this chapter I will summarize my work in this field while working on this specific project, analyze the results that were achieved and briefly describe what other further improvements can be done.

8.1. Results achieved

The main goal of this project was to enhance the user experience while navigating in Google Maps in a unique way by removing the interaction with the keyboard and mouse and introducing a new way of human-to-computer interaction using the Leap Motion controller, giving the end user full control and full freedom of hand and finger movements. At the same time, the second goal of the project was to provide an easy to use JavaScript library that can be used by any developer. The aim of the second goal was to keep it as simple as possible, making the integration in their software as smooth as possible, without producing much change in the existing code.

In the end, I managed to deliver a fully functional ready to use JavaScript library that can be integrated in any web application. As we can see from the test results from Chapter 6, the performance of the application will not drop by using this library, keeping the frames per second at least as without it.

The navigation in Google Maps is quite smooth, the rendering of the map is done without lag and the precision is quite high. In Street View, the end user could easily navigate through panoramas back and forth without having to struggle too much. Rotating the camera, while in Street View, using the swipe movements is not that smooth unfortunately.

8.2. Further improvements

At this moment there are some things that can be enhanced or other functionalities that can be added to the current library.

The biggest flaw which, from my point of view, the library has, is the fact that while in Street View, rotating the camera is not that accurate and smooth as I would like. So therefore, I think that implementing the camera rotation in Street View using swipe gestures is not the best option. A better solution for this it would be to rotate the camera in a similar way with which panning the map is implemented in Google Maps mode giving more precision, accuracy and freedom of movement. This can be achieved by using GSVPano.js for downloading all the tiles that compose one panorama, stitching them together, without the use of Google Maps API and implementing the rotation of camera using Three.js. The amount of effort for implementing this solution was out of scope in this pilot project.

At this moment there exists no zooming functionality while the user is navigating in Street View. I think this will be another important feature that has to be present. Lacking zoom functionality in Street View is not such a big flaw, but it takes away functionality from the original Google Maps implementation, which can be frustrating at some point.

The current representation of the pointer, is a black square of 5 pixels width. This could be modified in something more user friendly, such as a hand, a pointer or a tool.

Maybe a modern approach could be considered for the interaction with between the Leap Motion events, Google Maps API and the HTML files. At this moment each help view is represented by a separate HTML file which is loaded in the help window div whenever a swipe gesture is detected. AngularJS framework could be an option, for example, for switching the help views with its in ui-routing.

At this moment the help window is built in by me, an alternative could be to introduce a CSS framework, such as Bootstrap (from Twitter), and use one of the predefined popup windows. On a first look this will improve the compatibility between browsers, it will make the design responsive and will probably enhance the user experience, but on the other hand, this change will attract even more dependencies that our library will need, so the size of the project will be bigger, therefore the performance lower.

Chapter 9. Bibliography

[1] Markets and Markets, "Gesture recognition & Touch-less sensing market (2013-2020)," Markets and Markets, Dallas, U.S., 2014.

[2] L. Swenson, "The Future of Interaction Design With Christopher Noessel and Maggie Hendrie," 11 July 2013. [Online]. Available: http://www.mediacontour.com/the-future-of-interaction-design-with-christopher-noessel-and-maggie-hendrie/. [Accessed August 2014].

[3] S. Krug, Don't make me think. A common sense approach to web usability, Berkeley, California, USA: New Riders Publishing, 2006.

[4] Google, "Google Earth plugin for Leap Motion," Google, [Online]. Available: https://airspace.leapmotion.com/apps/google-earth/weblink. [Accessed August 2014].

[5] Google, "Google Earth System requirements," Google, [Online]. Available: https://support.google.com/earth/answer/20701?hl=en. [Accessed August 2014].

[6] Teehan+Lax Labs, "Teehan+Lax - Defining Experience," Teehan+Lax Labs, [Online]. Available: http://www.teehanlax.com/. [Accessed August 2014].

[7] Google, "Google Maps/Earth Terms of Service," Google, [Online]. Available: http://maps.google.com/help/terms_maps.html. [Accessed August 2014].

[8] Teehan+Lax Labs, "Driving Google Street View with Leap Motion," Teehan+Lax Labs, [Online]. Available: http://youtu.be/1-lXnyFm_Wc. [Accessed August 2014].

[9] Leap Motion, "Leap Motion: Technical Specifications," Leap Motion, 2014. [Online]. Available: https://www.leapmotion.com/product.

[10] Leap Motion, "Leap Motion System architecture," Leap Motion, [Online]. Available: https://developer.leapmotion.com/documentation/skeletal/javascript/devguide/Leap_Architecture.html. [Accessed August 2014].

[11] J. Guna, G. Jakus, M. Pogačnik, S. Tomažič and J. Sodnik, "An Analysis of the Precision and Reliability of the Leap Motion Sensor and Its Suitability for Static and Dynamic Tracking," *Sensors,* no. 14, pp. 3702-3720, 2014.

[12] F. Weichert, D. Bachmann, B. Rudak and D. Fisseler, "Analysis of the Accuracy and Robustness of the Leap Motion Controller," *Sensors,* no. 13, pp. 6380-6393, 2013.

[13] M. M. Sturman, D. E. Vaillancourt and D. M. Corcos, "Effects of Aging on the Regularity of Physiological Tremor," *Journal of Neurophysiology,* vol. 93, pp. 3064-3074, 2005.

[14] How Stuff Works, "How the Wii Works," How Stuff Works, [Online]. Available: http://electronics.howstuffworks.com/wii.htm. [Accessed August 2014].

[15] How Stuff Works, "How Microsoft Kinect Works," How Stuff Works, [Online]. Available: http://electronics.howstuffworks.com/microsoft-kinect.htm. [Accessed August 2014].

[16] Asus, "Asus Xtion PRO Live," Asus, [Online]. Available: http://www.asus.com/Multimedia/Xtion_PRO_LIVE/. [Accessed August 2014].

[17] iPiSoft, "Depth Sensors Comparison," iPiSoft, [Online]. Available: http://wiki.ipisoft.com/Depth_Sensors_Comparison. [Accessed August 2014].

[18] Intel, "Intel RealSense," Intel, [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html. [Accessed August 2014].

[19] PointGrab, "PointGrab," PointGrab, [Online]. Available: http://www.pointgrab.com/. [Accessed August 2014].

[20] Elliptic Labs, "Elliptic Labs," Elliptic Labs, [Online]. Available: http://www.ellipticlabs.com/. [Accessed August 2014].

[21] Thalmic Labs, "Myo armband," Thalmic Labs, [Online]. Available: https://www.thalmic.com/en/myo/. [Accessed August 2014].

[22] Intugine, "Intugine Nimble," Intugine, [Online]. Available: http://intugine.com/. [Accessed August 2014].

[23] Leap Motion, "Leap Motion API Overview," Leap Motion, [Online]. Available: https://developer.leapmotion.com/documentation/skeletal/javascript/devguide/Leap_Overview.html. [Accessed August 2014].

[24] Leap Motion, "Coordinate Systems," Leap Motion, [Online]. Available: https://developer.leapmotion.com/documentation/skeletal/csharp/devguide/Leap_Coordinate_Mapping.html. [Accessed August 2014].

[25] Google, "Map types," Google, [Online]. Available: https://developers.google.com/maps/documentation/javascript/maptypes#WorldCoordinates. [Accessed August 2014].

[26] Leap Motion, "Leap Motion JavaScript API," Leap Motion, [Online]. Available: https://developer.leapmotion.com/documentation/skeletal/javascript/. [Accessed August 2014].

[27] Google, "Google Maps JavaScript API V3 Getting started," Google, [Online]. Available: https://developers.google.com/maps/documentation/javascript/tutorial. [Accessed August 2014].