Recognition of the Spanish Sign language alphabet based on the Leap motion Controller

Gilberto Naranjo García, Valentín de Armas Sosa, Félix B. Tobajas Guerrero.
Institute for Applied Microelectronics (IUMA)
University of las Palmas Gran Canaria (ULPGC), Spain
{gnaranjo, armas, tobajas}@iuma.ulpgc.es

***Abstract*—The World Health Organization estimates that there is around 360 million people who suffer from audition deficiency. In a national level, there is around 1 million people, whom 10% of those use the sign language to communicate. Accordingly, the main objective of this paper is to design and develop a unique platform with capabilities of translating from the Spanish sign language alphabet into written text. To do so, it was necessary to carry out studies to determine the parameters that would be extracted in order to discriminate between the different symbols of the language and by means of a support vector machine, ultimately identify the corresponding letter to the symbol presented.**

Keywords: OMS, Spanish sign language, Leap Motion controller, Hardware/Software platform, Support Vector Machines.

I. INTRODUCTION

The Spanish dactylological language (*SDL*) is considered to be an alphabetical system, although it is not international and therefore each country has its own. [1] On this sense, among years, there has been many technological developments which aimed to solve the audition deficiency, being each solution specific to the region of development. The recognition systems developed are based in different means of data capture, being the most common those which use images to sense the depth and capture the position of hands and figures. [2] Other systems use external sensors in order to capture data, [3] and there is also those based in multimodal sensing [4] which use a combination of techniques to achieve the data recollection.

In a more specific approach, this paper is based on the development of a platform which uses image processing to extract relevant data from the *Leap Motion Controller (LMC)*. [5] The device is able to provide detailed information about the position of each hand detected within its field of view. [6] The software that operates the controller is able to recognize the position of the fingers with a precision of 0.01 mm, bearing a transmission frequency of up to 100 Hz. [7] Alongside with the hardware presented, the company developed a Software Developer Kit (*SDK*) which allow for the extraction of the specified data. Therefore, a code had to be developed in order to enable the extraction of the parameters and subsequently develop a classifier based on Support Vector Machines (*SVM*), that would realize the identification of symbols. The use of this type of classifiers can increase the recognition rate of each symbol up to 80% [8].

II. LEAP MOTION CONTROLLER

The *LMC* is a controller based in gestures [9] for human-computer interaction. It has the capability of tracking hand and finger position in real time, enclosed in a tridimensional space and with a precision of 0.01 mm. It has a working range of 1 m due to its vision field which is of approximately 150º in its widest part. The effective range for data capture is directly above the controller, however, it comes with a preestablished working position, implying that if the device is not positioned within the orientation of its working position, the controller won't be able to detect the change and adapt to it. [10] There are two preestablished working positions: the first implies the controller being laid over a flat surface and necessarily connected to a computer via USB. Whilst the other position is mounted to virtual reality glasses, this is possible thanks to an update of the software's hardware control, which allows the interaction with glasses such as *Oculus rift*. [11]

A. Hardware

The *LMC* is composed of 3 infrared LED, mainly used to lighten up the vision field of the controller. Two monochromatic infrared cameras with a 4 cm separation between themselves, being capable of capturing a framerate within the range of 50 up to 200 fps, depending onto which port is connected. The controller also incorporates the Macronix 25L320E [12][13] integrated circuit to store the firmware of the controller, this is a 32 Mbit serial NOR flash. Alongside with a FX3 SuperSpeed USB 3.0 controller. [14] In this case the USB CYUSB3014-BZXC [13][15] from *Cypress Semiconductor*.

B. SDK

The Software Developer Kit is directly downloadable from its official webpage. [16] Within the kit is also included an Application Programming Interface (*API*) which divides into data structures. These structures are divided into objects that represent *Hand* and are composed of other objects representing the *Fingers* and *Bones* that compose each detected hand.

Within the files included in the kit there is a main library that was employed in the development of the software side of the application. This is named *LeapC.h* and its organized in a hierarchical way, being the superior entity, the *Frame*

captured with the cameras. The *Frame* entity encompasses the *Hands* entity in the same way that *Fingers* are contained in *Hands*. As such demonstrated in Figure 1.

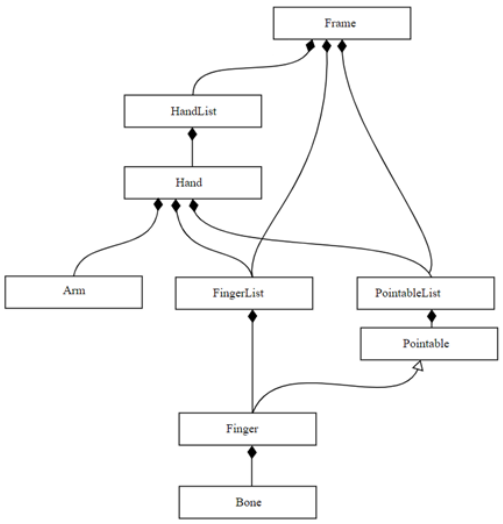

Figure 1. Leap Hierarchy.

III. SVM

There are numerous different options when it comes to classification. From those, the *SVM* algorithm was proposed for this paper. This is mainly based on the fact that once the classification is made the solution will be global and unique, also this type of classifier allows for a simplification of the adjustment parameters that are available.

A. LIBSVM

The classifier was introduced by means of *LBISVM* [17], which is a library specifically developed to simplify the adaptation of this kind of algorithms into other specific developments, such as the one presented in this paper. The library is described as simple, efficient and easy to use tool, that supports: *SVC* (Support Vector Classification), *SVR* (Support Vector Regression), *One-class SVM*.

In this case, the option chosen corresponds to the *SVC*, which allows for multiclass classification, by applying the "one-against-one" method. Accordingly, this paper focuses on the classification of 30 different symbols. Because the symbols are not lineally separable it was necessary to use a kernel function named *RBF* (Radial Base Function), this function includes two main tuning parameters that allow for a correct calibration of the classifier. The parameters are known as: *C* and *Gamma*. Due to previous studies, such as [18], the starting value for these two adjustment parameters is the one showed in Table I.

TABLE I. INITIAL *C* AND *GAMMA*

C	***Gamma***
100	0.0001

Once the package is downloaded from the developer's webpage [17], the package includes different files. The main ones utilized in this paper are the main library files that describe the use of the classifier along with some specific functions and some precompiled executables to be used through the windows command line. The library files included are: *svm.h* and *svm.cpp*, these files are to be used in the development of a project which aims to include the classifier within its execution. In the same way these include some specific functions that are required to carry out a classification with this type of algorithm, being those: *scale*, *train* and *predict*.

On the other hand, the precompiled files which are included provide the ability of executing those specific commands directly through *Windows Terminal*. The files included are: *svm-scale.exe*, *svm-train.exe* and *svm-predict.exe*, along some other ones which are not used in the development of this work. This type of files are the ones used to make the experiments and therefore check the variations in classification alongside the development of the work.

IV. INITIAL DEVELOPMENT

The initial step towards the implementation of the platform was to decide over the parameters that could be extracted from the LMC's SDK. Initially a code was written in C/C++, which main objective was to dump the results into a log file, giving the opportunity of analyzing those extracted parameters in a more in-depth manner. A pull back is that the SDK only allowed for the extraction of *Finger* rotation data in form of quaternions.

A. Quaternions

Quaternions are a way of describing 3D rotations, although they where introduced in 1844 [19] its use is no very extended due to the difficulty they present when it comes to visualization. A quaternion $q(q_0, \boldsymbol{q})$ is composed of a scalar q_0 and a vector $\boldsymbol{q} = (q_1, q_2, q_3)$. This way of repressing the rotation is not useful for the development of the project and therefore a transformation to Euler's rotation had to be applied. On this sense, a rotation matrix had to be introduced in order to obtain the rotation of the *Fingers* along Euler's theorem. The rotation matrix utilized is as follows in Figure 2.

$$R(q)' = \begin{pmatrix} 1 - 2(q_2^2 + q_3^2) & -2(q_0q_3 - q_1q_2) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 1 - 2(q_1^2 + q_2^2) \end{pmatrix}$$

Figure 2. Rotation matrix.

B. *Static symbol recognition*

When obtained the rotation matrix, it was possible to extract the angle of rotation of each finger in degrees by applying the mathematic transformation of multiplying vectors. In Figure 3, a representation of how to obtain the angle corresponding to the flexion of each *Bone* is shown.

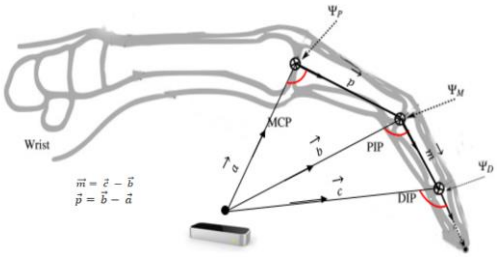

Figure 3. Flex angle. [20]

As already mentioned, to obtain the angle in degrees it was necessary to apply the formula shown in Figure 4.

$$\cos^{-1}\left(\frac{\vec{p}\cdot\vec{m}}{|\vec{p}|\cdot|\vec{m}|}\right)$$

Figure 4. Math transformation.

Once obtained the flex angle of each *Finger*, it was possible to finally establish the parameters that would be extracted for the static symbol recognition, and those are as follows in Table II.

TABLE II. STATIC PARAMETERS

Parameter	***Description***	***Data Structure***
bones.rotation	Finger rotation.	LEAP_QUATERNION
Flex angle	Flex level of each finger.	FLOAT
digits.is_extended	Whether a finger is considered extended or not.	LEAP_DIGIT

The extraction of parameters was dumped into a log file, that later would be used in combination with precompiled binaries included in the *LIBSM*. By means of these and following the strict procedure the developer of the library includes, a classification could be made. The main objective here was to obtain the parameters that could discriminate the most between the different symbols, and therefore simplify the classification, making it easier for the algorithm to group correct instances. This first experiment provided a sufficient *accuracy* level for the classification, validating the selected parameters for the static symbol recognition.

C. *Dynamic symbol recgonition*

When a high enough classification was obtained for the static symbols, the focus was set in finding some extra parameters that could discriminate specifically the dynamic symbols of the *SDL*.

Due to the specific movement of the hand for this type of symbols the parameter *palm.Velocity* had to be introduced, this identifies the velocity of the palm of a detected *Hand* within the system. On the other hand, there are great similarities between some dynamic letters of the Spanish sign language alphabet, and therefore another parameter had to be introduced. In this case it was necessary to control the movement of the pinky *Finger*. It was decided to introduce the calculus of the standard deviation for both the velocity of the palm of the *Hand* and the pinky *Finger* flex angle. Accordingly, if there was deviation above a threshold level, previously established for each parameter, it was considered that the result demonstrated the movement of the *Hand* and the pinky *Finger*.

Hereafter, in Table III, the parameters selected for the recognition of the dynamic symbols is presented.

TABLE III. DYNAMIC PARAMETERS

Parameter	***Description***	***Data Structure***
bones.rotation	Finger rotation.	LEAP_QUATERNION
Flex angle	Flex level of each finger.	FLOAT
digits.is_extended	Whether a finger is considered extended or not.	LEAP_DIGIT
Palm.Velocity Standar Deviation	Rate of change of a *Hand.*	LEAP_VECTOR
Pinky flex angle Standard Deviation	Rate of change of pinky flex angle.	FLOAT

The results derived from the extraction of the parameters shown above, were dumped into a log file, once again. From this newly created file, was applied the procedure of verification stablished previously, this is, by means of the precompiled headers in the *LIBSVM*, carry out the classification of the dynamic symbols. Following, the procedure stablished by the developer of the library and *accuracy* value was obtained. In this case the obtained level was not sufficient. This triggered another study to find out another parameter that could help solve this situation.

In the Spanish dactylological language there is a clear differentiation between symbols executed with the palm focused towards the *LM* controller and those which the palm is directly opposite to the device. Therefore, the parameter that was later introduces was *palm.normal*, this parameter obtains the normal vector of a *Hand*, allowing to differentiate between whether a palm is focused downwards or upwards.

Once again, in Table IV, the final parameters that would be used in the development of the complete system, are presented.

TABLE IV. COMPLETE PARAMETERS

Parameter	***Description***	***Data Structure***
bones.rotation	Finger rotation.	LEAP_QUATERNION
Flex angle	Flex level of each finger.	FLOAT

Parameter	*Description*	*Data Structure*
digits.is_extended	Whether a finger is considered extended or not.	LEAP_DIGIT
Palm.Velocity Standar Deviation	Rate of change of a *Hand*.	LEAP_VECTOR
Pinky flex angle Standard Deviation	Rate of change of pinky flex angle.	FLOAT
Palm.normal	Normal axis of the palm of a *Hand*.	LEAP_VECTOR

With the introduction of the new parameter the *accuracy* level for the classification of the dynamic symbols upgraded, reaching a level that could be suitable for the development of the translating platform.

V. Final Development

To continue with the development of the project, it was necessary to test the completed datasets which were generated for the 30 symbols that compose the *SDL*. Once both the static and dynamic symbols data sets where unified, the experiment was carried out, with expectations of proving the correct classification of the complete dactylological alphabet. This experiment was relatively similar to the one corresponding to the classification of the complete dynamic dataset, being the only main differentiation that for the first experiment, only the dynamic symbol dataset was used whilst for this experiment the complete dataset was going to be used.

The *accuracy* level for this experiment was 98.8% of correct classifications. For this specific case, the result was not satisfactory, this is mainly due to the fact that for the development of the real-time translation platform, it was necessary to obtain an accuracy level of 100%, corresponding to a correct classification of all the different symbols of the *SDL*. Therefore, it was implied that a correction of the tuning parameters of the classifier had to be applied.

The parameters to be tuned are those related to the kernel function utilized in the classifier, this are: *C* and *Gamma*. A study was carried out in order to settle the correct value for these parameters. Finally, it was found out that the parameters had to be tuned as it can bee seen in Table V.

TABLE V. *Final C and Gamma*

C	*Gamma*
90	0.0005

After tuning the parameters, the same experiment was carried out. In this case the *accuracy* level obtained correspond with complete correct classification of the whole dataset. This validates the use of the tuned parameters.

VI. Final Platform

When the complete classification of the whole dataset was obtained correctly, the main focus of the project was to create a unique platform that could encompass all the previously stated, as well as make real time classification of the *DSL*.

Along the execution of each experiment, a code had to be developed in order to obtain the extraction of the parameters of interest. The code that initially extracted only parameters for static symbols was developed into another that could not only extract the required information for all the symbols of the dactylological language, but also included the algorithm for classification.

In order to control the platform a menu had to be introduced, displaying the different options that are available to the user, from this menu different specific functions can be called and utilized. These functions are comprised of the instructions and math transformations required to carry out, from the capturing and extraction of the desired information to the classification of the same, and therefore the obtention of the letter corresponding to the presented symbol.

In Figure 5, it can be observed the menu function for the final platform.

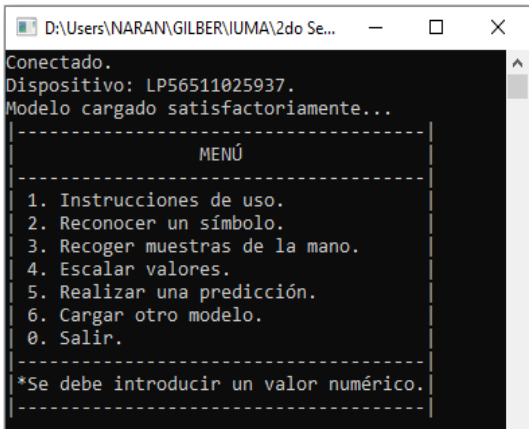

Figure 5. Menu function

Hereafter in Figure 6, an example of the execution of the system can be observed.

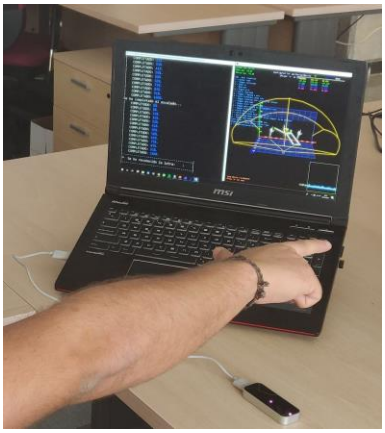

Figure 6. Complete Execution

VII. RESULTS

In this section the results of each classification realized through the development is presented. The main evaluation criteria for classifiers is through the *accuracy* level, which represents the total number of correct classifications made. Visually this can also be represented by means of a confusion matrix, in which a matrix of *n* * *n* dimension is created, where *n* represents the number instances. The main column represents the real instances whilst on the main row, the predicted instances are represented. This establishes and effective visually attractive way of verifying the results of the classification procedure.

Accordingly, the initial experiment was the one corresponding to the classification of the static symbols. As result of the experiment the confusion matrix observed in Figure 7 is obtained.

	A	B	C	D	E	F	I	K	L	M	N	O	P	Q	R	S	T	U
A	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
M	0	0	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3

Figure 7. Static Symbols

In this case the classifier correctly grouped 51 of the 54 available instances, this result corresponds to a 94.4% of *accuracy*. The next experiment carried out was the classification of the dynamic symbols, after the *palm.normal* parameter was introduced. In Figure 8, the results are shown.

	CH	G	H	J	LL	Ñ	RR	V	W	X	Y	Z
CH	3	0	0	0	0	0	0	0	0	0	0	0
G	0	3	0	0	0	0	0	0	0	0	0	0
H	0	0	3	0	0	0	0	0	0	0	0	0
J	0	0	0	3	0	0	0	0	0	0	0	0
LL	0	0	0	0	3	0	0	0	0	0	0	0
Ñ	0	0	0	0	0	3	0	0	0	0	0	0
RR	0	0	0	0	0	0	3	0	0	0	0	0
V	0	0	0	0	0	0	0	3	0	0	0	0
W	0	0	0	0	0	0	0	2	1	0	0	0
X	0	0	0	0	0	0	0	0	0	3	0	0
Y	0	0	0	0	0	0	0	0	0	0	3	0
Z	0	0	0	0	0	0	0	0	0	0	0	3

Figure 8. Dynamic Symbols

For this case the algorithm classified correctly 34 of the 36 possible instances. Once more the *accuracy* level according to this classification corresponds with the 94.4%. The experiment carried out was with the complete dataset, including both the static and dynamic symbols of the *SDL*. In Figure 9 the results are shown.

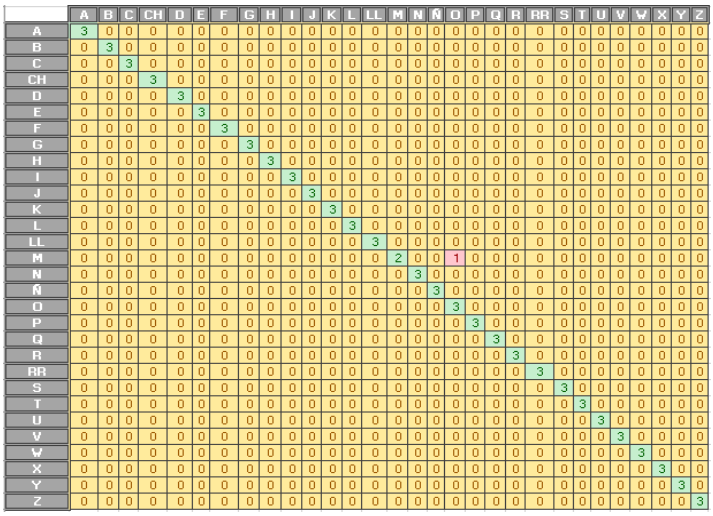

Figure 9. Complete Symbols

Once both datasets where combined and the experiment carried out, there was a reduction of the classification errors, for this specific case, the algorithm correctly arranged 89 of the 90 possible symbols, establishing an *accuracy* level of 98.8%. As mentioned before, although this result is pretty good, it is not sufficient for the development of the platform. Therefore, the *C* and *Gamma* parameters of the kernel function where tuned, resulting as it shows in Figure 10.

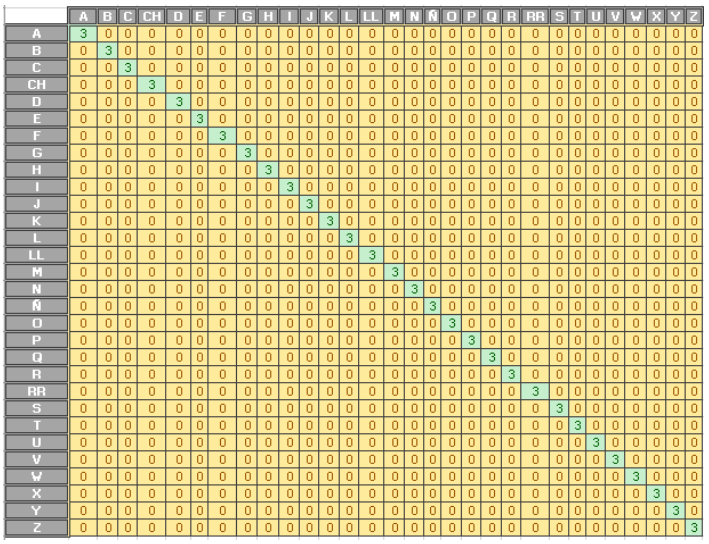

Figure 10. Complete Symbols & tuning

Figure 10 shows a complete correctly classified confusion matrix, this corresponds to an *accuracy* level of 100%, achieving finally the objective of classifying all instances correctly. This, in combination with the development of a unique platform that carries out the complete process, comprises the main objective of the development of the paper.

VIII. CONCLUSIONS

According to the results obtained in each experiment, it can be established that this work has achieved the main goal of creating and developing an interactive platform, based on the *Leap Motion Controller*, which can translate from the Spanish sign laguage alphabet into written text.

The platform was designed to be the most user friendly as possible, making it pretty easy to use it. On the other hand, it was also degined to have the least possible latency, taking only a few seconds to complete the whole process.

REFERENCES

[1] Fundación Once, "Datilológico". [Online] Available: http://ares.cnice.mec.es/informes/17/contenido/19.htm. [Accessed: March 2019]

[2] Marcus V. Lamar, Md. Shoail Bhuiyan, Akira Iwata. "Hand alphabet recognition using morphological PCA and neural networks Neural Networks"; IJCNN '99. (1999).

[3] "Un guante inteligente que traduce el lenguaje de signos a texto y audio." 2015. [Online] Available: https://descubrearduino.com/un-guante-inteligente-que-traduce-el-lenguaje-de-signos-a-texto-y-audio/. [Accessed: March 2019]

[4] H. Liu, Z. Ju, X. Ji, C. Chan and M. Khoury, "Human Motion Sensing and Recognition", 1st ed. Berlin, Heidelberg: Springer. Berlin Heidelberg, 2017, pp. 1-64. ISBN: 978-3-662-53692-6.

[5] Leap Motion, 201. [Online] Available: http://www.leapmotion.com/. [Accessed: March 2019]

[6] Mischa Spiegelmock. "Leap Motion Development Essentials"; Packt Publishing (2013). ISBN-10: 1849697728.

[7] Michał Nowicki, Olgierd Pilarczyk, Jakub Wasikowski, Katarzyna Zjawin. "Gesture Recognition library for Leap Motion controller". Poznan, Polonia, 2014. [Online] Available: http://www.cs.put.poznan.pl/wjaskowski/pub/theses/LeapGesture_BScThesis.pdf [Accessed October 2019].

[8] Makiko Funasaka, Yu Ishikawa, Masami Takata, and Kazuki Joe. "Sign Language Recognition using Leap Motion Controller". Nara, Japan (2015). [Online] Available: https://pdfs.semanticscholar.org/68ef/18393db775cccfe55d2e806b40a95dd53f31.pdf [Accessed October 2019].

[9] Jaagrup Irve. "Gesture Evaluation for Leap Motion". Tallinn 2015. [Online] Available: https://digi.lib.ttu.ee/i/file.php?DLID=3579&t=1 [Accessed October 2019].

[10] Jaagrup Irve. "Gesture Evaluation for Leap Motion". Tallinn 2015. [Online] Available: https://digi.lib.ttu.ee/i/file.php?DLID=3579&t=1 [Accessed October 2019].

[11] Jaagrup Irve. "Gesture Evaluation for Leap Motion". Tallinn 2015. [Online] Available: https://digi.lib.ttu.ee/i/file.php?DLID=3579&t=1 [Accessed October 2019].

[12] Candemir Orsan. "What's Inside? – Vol 1: Leap Motion". 2014. [Online] Available: https://medium.com/@candemir/taking-things-apart-vol-1-leap-motion-36adaa137a0a [Accessed October 2019].

[13] Macronix International Co. "Serial NOR Flash MX25L3206E Specifications". [Online] Available: http://www.macronix.com/en-us/products/NOR-Flash/Serial-NOR-Flash/Pages/spec.aspx?p=MX25L3206E&m=Serial%20NOR%20Flash&n=PM1568 [Accessed October 2019].

[14] Cypress Semiconductor. "Leap Motion Selects Cypress's EZ-USB® FX3™ Solution for Controller Components". 2013. [Online] Available: http://www.cypress.com/?rID=74083 [Accessed October 2019].

[15] Macronix International Co. "Serial NOR Flash MX25L3206E Specifications". [Online] Available: http://www.macronix.com/en-us/products/NOR-Flash/Serial-NOR-Flash/Pages/spec.aspx?p=MX25L3206E&m=Serial%20NOR%20Flash&n=PM1568 [Accessed October 2019].

[16] Cypress Semiconductor. "EZ-USB FX3 *CYUSB3014-BZXC* Specifications". [Online] Available: https://www.cypress.com/part/cyusb3014-bzxc [Accessed October 2019].

[17] Leap Motion Developer. "Get Started". [Online] Available: https://developer-archive.leapmotion.com/get-started [Accessed October 2019].

[18] Chang C, Lin C. "LIBSVM: A library for support vector machines". National Taiwan University. 2011. [Online] Available: https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf [Accessed October 2019].

[19] Claudia R. Rivero. "Plataforma para la interpretación del alfabeto dactilológico de la lengua de signos española basadas en dispositivos IoT". Universidad de Las Palmas de Gran Canaria. 2018.

[20] JONGCHAN BAEK, HAYEONG JEON, GWANGJIN KIM, SOOHEE HAN. "Visualizing Quaternion Multiplication". [Online] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7930375 [Accessed October 2019].

[21] P.D.S.H. Gunawardane, Nimali T. Medagedara. "Comparison of Hand Gesture inputs of Leap Motion Controller & Data Glove in to a Soft Finger". [Online] Available: https://ieeexplore.ieee.org/document/8250099 [Accessed October 2019].