UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Reconocimiento del alfabeto dactilológico de la lengua española de signos basado en el dispositivo Leap Motion

Autor: Gilberto Naranjo García
Tutor(es): D. Valentín De Armas Sosa
D. Félix B. Tobajas Guerrero
Fecha: Diciembre 2019

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Reconocimiento del alfabeto dactilológico de la lengua española de signos basado en el dispositivo Leap Motion

HOJA DE FIRMAS

Alumno/a:	Gilberto Naranjo García	Fdo.:	NARANJO GARCIA GILBERTO - 54107675Z Firmado digitalmente por NARANJO GARCIA GILBERTO - 54107675Z Fecha: 2019.12.04 13:33:13 Z
Tutor/a:	D. Valentín De Armas Sosa	Fdo.:	DE ARMAS SOSA VALENTIN - 52856177S Firmado digitalmente por DE ARMAS SOSA VALENTIN - 52856177S Fecha: 2019.12.05 12:01:07 Z
Tutor/a:	D. Félix B. Tobajas Guerrero	Fdo.:	TOBAJAS GUERRERO FELIX BERNARDO - 43765233M Firmado digitalmente por TOBAJAS GUERRERO FELIX BERNARDO - 43765233M Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-43765233M, givenName=FELIX BERNARDO, sn=TOBAJAS GUERRERO, cn=TOBAJAS GUERRERO FELIX BERNARDO - 43765233M Fecha: 2019.12.05 10:53:57 Z

Fecha: Diciembre 2019

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Reconocimiento del alfabeto dactilológico de la lengua española de signos basado en el dispositivo Leap Motion

HOJA DE EVALUACIÓN

Calificación: .....

Presidente Fdo.:

Secretario Fdo.:

Vocal Fdo.:

Fecha: Diciembre 2019

IUMA
t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

ABSTRACT

The world health organisation estimates that there is more than a thousand million people living in the world with some kind of disability. Specifically, five percent of those people, around three hundred and sixty million is considered to have an audition deficiency. Nationally the figure of people is estimated to be around a million, from whom ten percent use the sign language as a main communication means. Accordingly, the main objective of this master's thesis was to create and develop an interactive platform that could translate the Spanish dactylological language into written language in real time.

Initially a study had to be carried out in order to determine the hardware/software resources that would be used in the development of the platform. On the other hand, a study of the Spanish dactylological language was also carried out, with the objective of identifying the best parameters to discriminate between the different symbols that compose the language. The development of the platform consisted in the generation of a C/C++ written code, unifying all the previous studies in a single reconnaissance platform. Hence the code was developed to extract the main parameters and transform them into accountable data, a Support Vector Machine (SVM) was utilized to classify the symbols and therefore ultimately identifying the corresponding written letter.

During the development of the platform a series of experiments where designed and executed to validate the parameters extracted according to the code and the classification made by the SVM algorithm. The results derived from these experiments support the decision of the selected parameters, as well as, backing up the use of the support vector machine to produce the letter corresponding to the symbol of the sign language. As evaluation metrics the use of the accuracy of the system, was selected among others. Comparing the results amid themselves to obtain better results within the execution of the different experiments.

RESUMEN

Según la Organización Mundial de Salud (*OMS*) se estima que actualmente existen más de mil millones de personas en el mundo con algún tipo de discapacidad. En cuanto a la discapacidad auditiva, a nivel nacional dicha cifra se establece en torno a un millón de personas, cuya gran mayoría utiliza el lenguaje de signos para comunicarse diariamente. En ese sentido el objetivo principal del desarrollo de este trabajo fin de máster consistió en la creación y desarrollo de una plataforma interactiva con la que poder realizar una traducción del alfabeto dactilológico español. Para llevar a cabo dicho objetivo se realizó un estudio de los diferentes símbolos que componen la *LSE*, así como de los sistemas *Hardware/Software* actuales para determinar la idoneidad del mismo. Existe una gran diversidad en cuanto a sistemas para la extracción de datos se refiere, por ello a partir del dispositivo seleccionado se realizó un estudio de los parámetros que discriminarán los símbolos del lenguaje dactilológico entre sí. Para posteriormente generar un código descrito en *C/C++* que englobase lo previamente dicho en una única plataforma de reconocimiento. El mismo se lleva a cabo mediante la integración de el algoritmo de clasificación *SVM*.

Durante el desarrollo de la plataforma se llevaron a cabo diferentes experimentos con el fin de validar el tipo de parámetros extraídos y la clasificación según el *SVM*. Los resultados derivados de los mismo respaldan la utilización de tanto los parámetros extraídos a partir del controlador *HW* como de la utilización de este tipo de clasificador. Como métrica de evaluación se estableció el valor del *accuracy*, comparando los resultados entre sí para obtener una mejora del mismo, a lo largo de la ejecución de los diferentes experimentos.

AGRADECIMIENTOS

A mis padres por el apoyo incondicional a lo largo de toda mi vida, tanto en lo bueno como en lo malo, gracias a ellos este Trabajo Fin de Máster ha sido posible. A mi hermano y hermana por haber estado siempre ahí cuando lo necesitaba.

A mis tutores Dr. Félix Tobajas y Dr. Valentín de Armas, por su esfuerzo y dedicación que han demostrado tanto a lo largo del desarrollo del Trabajo Fin de Grado como el mostrado para completar este.

A todos mis compañeros de carrera y máster, gracias a ellos he aprendido valiosas lecciones de vida. A mis amigos más cercanos que me enseñaron que siempre hay que tirar para adelante.

A Andrea que siempre estará ahí.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ACRONIMOS

API	*Application Programming Interface*
CI	*Circuito Integrado*
DLL	*Dynamic Link Library*
EMG	*Electromiografía*
FPS	*Frame per second*
GUI	*Graphic User Interface*
HTTP	*Hyper Text Transfer Protocol*
IDE	*Integrated Development Kit*
IUMA	*Instituto Universitario de Microelectrónica Aplicada*
JSON	*JavaScript Object Notation*
LMC	*Leap Motion Controller*
LMS	*Leap Motion Service*
LSE	*Lengua de Signos Española*
OMS	*Organización Mundial de la Salud*
RBF	*Radial Basis Function*
SDK	*Software Development Kit*
SDL	*Spanish Dactylological Language*
SVC	*Support Vector Classification*
SVM	*Support Vector Machine*
SVR	*Support Vector Regression*
TFM	*Trabajo Fin de Máster*
ULPGC	*Universidad de Las Palmas de Gran Canaria*
USB	*Universal Serial Bus*

Capítulo 1. Introducción

El primer capítulo se centra en introducir los diferentes aspectos de los temas que se engloban el presente Trabajo Fin de Máster.

1.1. Introducción

Según la Organización Mundial de la Salud (*OMS*) se estima que más de mil millones de personas viven en todo el mundo con alguna forma de discapacidad [1]. Estas cifras son aún más preocupantes debido al aumento de la prevalencia que demuestra. De estos mil millones de personas, un 5% (360 millones) padece pérdida de audición, mientras que se considera que para 2050, una de cada diez personas la padecerán [2]. Más concretamente, la referencia sobre población con discapacidad auditiva en España se cifra en torno a un millón de personas, de las cuales un 10% se comunica principalmente mediante lengua de signos [3].

El alfabeto dactilológico es un sistema alfabético, aunque no es internacional, por lo que cada país tiene el suyo [4]. En este sentido, a lo largo de los años se han desarrollado varias tecnologías que pretendían solventar el problema de la deficiencia auditiva, siendo cada solución específica para cada región. Por ello, en este TFM, se pretende desarrollar una plataforma capaz de reconocer el alfabeto dactilológico español. Los sistemas de reconocimiento actuales se han desarrollado a partir de diferentes formas de captura de información, entre las cuales se encuentran los que se basan en el reconocimiento por imágenes, tanto de profundidad como de forma de manos y figuras [5], los desarrollos que hacen uso de sensores externos para la captura de información [6] y por último, los basados en la recopilación de datos multimodo [7], los cuales utilizan varias técnicas de recolección de información.

En concreto, el desarrollo de este TFM se basará en el procesamiento de imágenes para la extracción de información relevante mediante la integración del dispositivo *Leap Motion* [8][9]. Este dispositivo es capaz de proporcionar información detallada sobre la posición de cada mano detectada en el espacio de trabajo del controlador. A su vez, mediante el uso de un ordenador, es capaz de reconocer la posición de los dedos de las manos con una precisión de 0.01 mm. Esta detección permite el reconocimiento de un *framework* de los dedos y posibilita la extracción de información, como la posición de los huesos de los dedos, o el ángulo de flexión que forman éstos. El dispositivo soporta una frecuencia de transmisión de hasta 100 Hz [10]. Su introducción en 2013 ha proporcionado el bagaje de este, con una amplia librería de proyectos desarrollados a partir de esta tecnología.

La empresa desarrolladora, proporciona un SDK que permite la extracción de datos, a partir de los cuales se pretende desarrollar el *software* encargado del reconocimiento de símbolos para posteriormente desarrollar un clasificador basado en *SVM* y así posibilitar el proceso de reconocimiento y clasificación de símbolos. El uso de este clasificador es capaz de mejorar la tasa de reconocimiento de símbolos en un 80% [11].

1.2. *Antecedentes*

Existe una gran variedad de dispositivos en cuanto a sensores para la monitorización de la posición de las manos se refiere. Estos pueden dividirse en tres categorías principales: los dispositivos cuya captura se basa en sistemas ópticos, los basados en sistemas hápticos, y los últimos en sensores multimodales.

- Sistemas Ópticos

Los sistemas ópticos son aquellos que se fundamentan en la necesidad de utilizar una cámara para la monitorización de las posiciones de la mano. Estos sistemas, por lo general, funcionan en tres capas: detección, seguimiento e identificación. La capa de detección se encarga de identificar y extraer características visuales a través la cámara. Esto se realiza mediante la segmentación de la imagen y la identificación de características, colores o formas. La capa de seguimiento realiza una asociación espaciotemporal entre las

características extraídas, permitiendo la detección y localización de una mano, por ejemplo. Por último, la capa de identificación se encarga de realizar la definición y etiquetado de los datos previamente extraídos. [12]

- Sistemas Hápticos

Los sistemas basados en la tecnología háptica son aquellos que hacen uso de sensores hápticos y electromiográficos (*EMG*) para la captura de datos. Existe una gran variedad de este tipo de sensores, los cuales se utilizan principalmente para el monitoreo de presión, fuerzas o señales eléctricas musculares. En general, estos tipos de sensores no se comercializan habitualmente debido a la complejidad del *hardware* que integran, la mayoría de este tipo de dispositivos son hechos a medida. El sistema háptico más utilizado se basa en el guante con sensores flexibles, sin embargo, estos resultan ser considerablemente caros y la precisión de los sensores llega a decrecer con el tiempo. [12]

- Sistemas Multimodales

Los sistemas multimodales hacen referencia a la combinación de sensores ópticos y hápticos para la captura y extracción de datos. Este tipo de sistemas destaca por su complejidad, lo cual se debe al enfoque dual que lo caracteriza. La principal ventaja que presentan estos sistemas es la doble comprobación de los datos extraídos, por lo que en general aumenta la fiabilidad de estos sensores. No obstante, la elevada complejidad de los sistemas dificulta su comercialización. [12]

El dispositivo *HW* empleado en el desarrollo de este TFM se engloba dentro de los sistemas ópticos, ya que realiza la captación de datos, que posteriormente serán utilizados para el reconocimiento de los símbolos del alfabeto dactilológico, a través de las múltiples cámaras que posee el mismo.

1.3. Alfabeto dactilológico Español

A lo largo del tiempo y de forma natural se han creado lenguas de signos como respuesta creativa a la limitación sensorial de esta deficiencia [13]. Sin embargo no fue hasta 2007 que se reconoció oficialmente la lengua española de signos. [14] El alfabeto dactilológico español es un lenguaje natural completo con ciertas características lingüísticas. El alfabeto se expresa a través del movimiento gestual, siendo la principal vía de comunicación para las personas con deficiencia auditiva, aunque actualmente se encuentra bastante extendido entre la población que no padece esta deficiencia.

A continuación, se muestra el alfabeto dactilológico oficial español.

Figura 1.1.: Alfabeto dactilológico español. Fuente: CNSE

En la Figura 1.1, se muestra la lengua de signos española, en ella se puede observar que de los 30 símbolos de los que se compone este lenguaje, 18 se realizan de forma estática y los restantes 12 requieren de algún movimiento de la mano para su expresión. Debido a esto se planteó realizar la captura el ángulo de flexión de los dedos de la mano como parámetro discriminativo entre símbolos. A su vez se puede comprobar como existe una alta similitud entre símbolos, como es el caso de las letras “F” y “T”, en las que diferencias únicamente por la posición del dedo pulgar.

Existen múltiples letras cuyos símbolos son iguales y únicamente difieren al incorporar un movimiento. Un ejemplo podría ser la “R” o la “RR”, por ello fue necesaria la introducción de algún método de detección de movimiento con el fin de discriminar entre este tipo de letras. Asimismo, las letras “I”, “Y” y “Z” se basan en la misma posición de la mano, con ligeras diferencias en el dedo meñique. Específicamente, la diferencia entre la letra “I” y la letra “Y” se basa en la flexión del mismo dedo.

1.4. Objetivos

El objetivo principal de este Trabajo de Fin de Máster consiste el desarrollo de una plataforma *hardware/software* que permita la interpretación del alfabeto dactilológico de la Lengua de Signos Española (*LSE*) a partir del dispositivo *Leap Motion*.

Para satisfacer este objetivo general, será necesario desarrollar las librerías que permitan realizar, tanto la interacción del dispositivo *Leap Motion* con la máquina en la que se esté ejecutando, como la captura de información pertinente para la detección del alfabeto dactilológico español.

Para el desarrollo de la plataforma mencionada se identifican tres funciones clave:

- Detección de símbolos, fundamentalmente a partir del dispositivo *Leap Motion*.
- Clasificación de los símbolos detectados basada en SVM (*Support Vector Machine*).
- Procesamiento de los símbolos detectados.

1.5. Peticionario

Actúa como peticionario del presente Trabajo Fin de Máster (*TFM*) el Instituto Universitario de Microelectrónica Aplicada (*IUMA*) como requisito indispensable para la obtención del título de Máster Universitario en Tecnologías de Telecomunicación de la Universidad de Las Palmas de Gran Canaria (*ULPGC*), tras haber superado con éxito las asignaturas especificadas en el Plan de Estudios.

1.6. Estructura del documento

El presente documento se encuentra dividido en dos partes diferenciadas: Memoria y Bibliografía. La Memoria se ha estructurado en 7 capítulos, además de las referencias bibliográficas consultadas para la elaboración del presente *TFM*.

- **Parte I. Memoria.**
  - **Capítulo 1. Introducción.**

Se realiza una primera toma de contacto con los conceptos básicos que se abordan en este *TFM*, incluyéndose también los objetivos, el peticionario, y la estructura general del documento.

  - **Capítulo 2. Componentes *HW/SW*.**

Se describen las herramientas necesarias para el desarrollo del sistema. Por una parte, se detalla el componente *hardware* principal de la plataforma, además de específicamente el *software* empleado.

  - **Capítulo 3. Clasificación.**

El tercer capítulo describe los diferentes métodos de clasificación existentes, haciendo especial hincapié en el algoritmo *SVM*, que será utilizado para la clasificación de los símbolos en el desarrollo de la plataforma.

  - **Capítulo 4. Desarrollo de la plataforma *HW/SW*.**

En el cuarto capítulo se explican los pasos seguidos en la realización de distintos experimentos con el objetivo de diseñar un sistema completo inicial, incluyéndose también la base teórica sobre la que se sustenta el reconocimiento de símbolos estáticos.

- **Capítulo 5. Plataforma *HW/SW* final.**

En el quinto capítulo se detalla el diseño de la versión final de la plataforma *HW/SW*, incluyendo el reconocimiento de símbolos dinámicos y la implementación de funcionalidades. Se especifica también la utilización del algoritmo de clasificación *SVM*.

- **Capítulo 6. Validación y resultados.**

En el sexto capítulo se presenta un resumen de los resultados obtenidos a partir de la validación experimental del sistema completo, así como las limitaciones existentes.

- **Capítulo 7. Conclusiones y líneas futuras.**

Finalmente, en el séptimo capítulo se presentan las conclusiones y las propuestas de líneas futuras.

– **Parte II. Bibliografía**

En la segunda parte se abarcan las referencias y documentos consultados para el desarrollo de este *TFM*.

Capítulo 2. Componentes *HW/SW*

El segundo capítulo se enfoca en La explicación de los componentes tanto *Hardware* como *Software* utilizados en el desarrollo del presente *TFM*.

2.1. Dispositivo Hardware

En esta sección se describen los aspectos básicos del dispositivo *Hardware* empleado para el desarrollo de la plataforma, correspondiente al controlador *Leap Motion*.

2.1.1. Leap Motion

Leap Motion Controller (LMC) es un dispositivo de interfaz humano-ordenador basado en gestos [16]. Es capaz de capturar y rastrear la posición de manos y dedos en tiempo real, en un espacio tridimensional, y con una precisión de 0.01 milímetros. El dispositivo es capaz de detectar objetos en un rango de 1 metro gracias a su campo de visión, el cual es de aproximadamente 150 grados en su parte más ancha. En la Figura 2.1, se puede observar una representación del espacio de interacción del dispositivo.

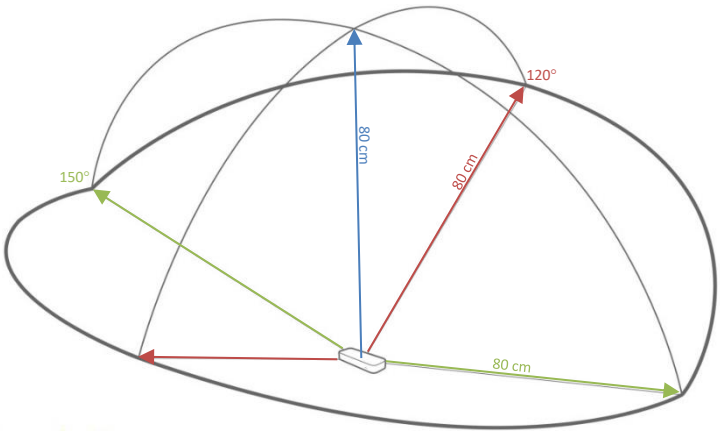


Figura 2.1.: Rango de interacción Leap Motion

El rango efectivo de captura de información de *LMC* se encuentra directamente sobre las cámaras de este, y abarca un rango hasta 80 centímetros [15]. Sin embargo, existe una restricción a aplicar, ya que el dispositivo tiene una orientación predeterminada, lo que implica que en caso de que el dispositivo se coloque en alguna otra orientación que la establecida, no será capaz de detectar el cambio, y por consiguiente ajustarse al mismo [16]. Como se puede observar en la Figura 2.2, existen dos posiciones para la colocación del dispositivo: una es en una superficie plana, y necesariamente conectado mediante USB al ordenador, y la otra es montado sobre unas gafas de realidad virtual con el fin de capturar el movimiento de las manos. Esto ha sido posible gracias a la introducción del *software Orion*, el cual permite la interacción del dispositivo con algunas gafas de realidad virtual como *Oculus Rift* o *HTC Vive* [16].

Figura 2.2.: Posicionamiento del Leap Motion [17]

El *hardware* del dispositivo *Leap Motion* se compone de tres LED infrarrojos utilizados para iluminar el campo de visión del dispositivo. Dos cámaras infrarrojas monocromáticas con una separación de 4 centímetros entre sí, las cuales capturan imágenes a un *framerate* desde 50 hasta 200 *fps*, siempre dependiendo del puerto USB al que esté conectado. A su vez, el dispositivo incorpora el circuito integrado (CI) *Macronix 25L320E* [18][19]. Este CI proporciona 32Mbits de espacio de almacenamiento, que se utiliza para almacenar el *firmware* para el controlador USB. El último circuito integrado que se incluye en el dispositivo es el controlador USB *CYUSB3014-BZXC* [19][20] de la empresa *Cypress Semiconductor*, que forma parte de la familia *FX3 SuperSpeed USB 3.0* [21]. A continuación, en la Figura 2.3, se observa una vista esquemática de las dimensiones del dispositivo *Leap Motion*.

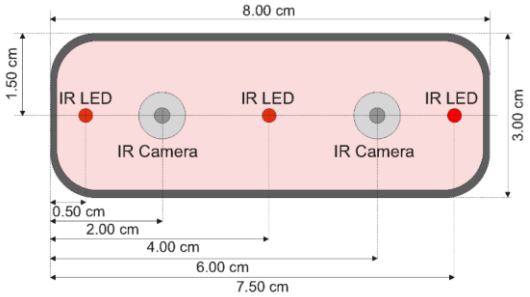


Figura 2.3.: Vista esquemática del dispositivo Leap Motion [24]

2.1.1.1. SDK

La empresa *Leap Motion Inc.* dispone de un *kit* de desarrollo *software* descargable directamente a través de su página web oficial [22]. Este *SDK,* denominado *Leap_Motion_Developer_kit* y en su versión 4.0.0, incluye el *software* utilizado como panel de control para el dispositivo, así como una carpeta donde se encuentra el grueso del *kit*, incluyendo todas las herramientas necesarias para comenzar con el desarrollo de aplicaciones basadas en este controlador.

La *API* incluida en el *SDK* se divide en estructuras de datos, para así poder englobar distintos tipos en una misma estructura. Esto es de gran utilidad en el caso de *Leap Motion,* debido a la necesidad de crear un tipo específico de datos como puede ser el objeto *Hand*, a partir de tipos diferentes que componen la misma, como sería los objetos *Fingers* o *Bones.*

Entre los archivos incluidos con el *SDK*, se encuentra la librería principal empleada para el desarrollo de la plataforma la cual se denomina `LeapC.h`. Esta se organiza de forma jerárquica, por lo que la entidad superior es el *Frame*, que a su vez engloba las manos disponibles en la entidad. De manera similar, los dedos se encuentran englobados en la entidad *Hand*. La estructura de datos que define las características de los dedos también incluye otra estructura que contiene los huesos de la mano, por lo que una vez más, y de forma similar, la organización de los dedos de la mano se integra dentro de una estructura superior. La entidad *Bone* describe los parámetros de definirán los huesos.

En la Tabla 1, se observan las principales entidades a utilizar para el desarrollo de esta aplicación basada en *Leap Motion*. Se añade una pequeña descripción de cada una de ellas, acompañada de la estructura de datos correspondiente para cada entidad.

Tipo	Descripción	Estructura de datos
Frame	Objeto raíz, el cual permite el acceso a todas las entidades monitoreadas en un instante de tiempo.	LEAP_TRACKING_EVENT
Image	Acceso a los datos en bruto de las cámaras y a los parámetros de calibración para *LMC.*	LEAP_IMAGE
Hand	Principal entidad extraíble, que a su vez contiene el resto de las entidades asociadas a la mano.	LEAP_HAND
Arm	El brazo adjunto a la mano, que permite la extracción de características.	LEAP_BONE
Palm	Contiene las propiedades asociadas a la palma de la mano.	LEAP_PALM
Finger (Digits)	Entidades que representan los dedos de una mano.	LEAP_DIGIT
Bone	Entidades que representan los huesos de cada dedo la mano.	LEAP_BONE

Tabla 1.: Estructuras principales de datos

Para llevar a cabo este desarrollo de forma precisa, es necesario entender el modo de funcionamiento del controlador, el cual provee de un sistema de coordenadas cartesiano, como el que se observa en la Figura 2.4. Se puede observar que las coordenadas de origen se encuentran directamente sobre el cristal superior del dispositivo. El sistema de coordenadas posee una precisión milimétrica. Por ejemplo, si la posición de un dedo se encuentra en una coordenada tal que *(x, y, z) = [0, 50, 0]*, significa que el dedo en cuestión se encuentra a 5 cm sobre la parte superior del controlador. [23]

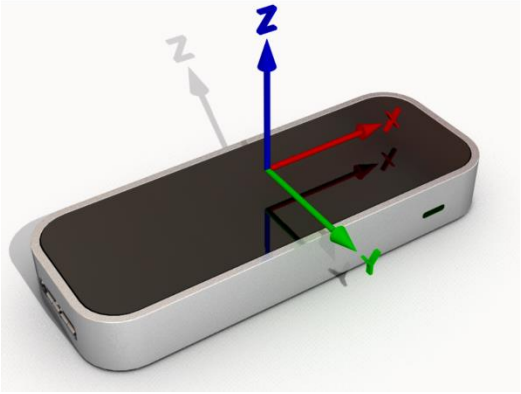


Figura 2.4.: Modelo 3D del sistema de coordenadas Leap Motion [24]

El sistema de coordenadas del controlador está basado en el sistema de coordenadas de la mano derecha, donde la parte positiva de los ejes *X* e *Y* apuntan hacia la derecha y hacia arriba. Sin embargo, la parte positiva del eje *Z* apunta hacia el usuario del controlador. En este sistema el eje de rotación positivo se realiza de forma contraria a la aguja de reloj.

2.1.2. Arquitectura del sistema

El software propio que utiliza el controlador, no provee de un canal de comunicación directo entre los datos extraíbles del dispositivo con otras aplicaciones. Por ello, es necesario la utilización de un *software* ejecutándose como servicio en *Windows* o como *Daemon* en Mac. Este servicio se denomina *Leap Motion Service (LMS).* Mediante la ejecución en segundo plano de este servicio, se consigue crear un puente de comunicación entre *LMC* y los programas que pretenden utilizar los datos extraídos del controlador.

El acceso a los datos extraídos se realiza mediante la utilización del *SDK* proporcionado por el desarrollador. En este, se incluyen dos *API*, las cuales están formadas por un conjunto de funciones generales y herramientas para la programación de aplicaciones *software,* basándose en la extracción de datos del controlador. Las interfaces de programación incluidas son para desarrollo de forma nativa y a través de *web socket*.

En *Windows,* la interfaz de programación nativa la aporta la utilización de la librería cargada de forma dinámica (*DLL*), propia del dispositivo *Leap Motion*, la cual contiene las clases que a su vez contienen los métodos y estructuras de datos pertinentes para la

generación de aplicaciones. El uso de la librería permite que las aplicaciones *software* puedan realizar la extracción de datos a través de *LMS*. El dispositivo soporta una gran variedad de lenguajes de programación, como puede ser *Objective-C, C++, Python, Java*, aparte de incluir un *plugin* para su utilización en motores de juego como *Unity 3D* y *Unreal Engine*. La gran variedad de lenguajes de programación soportados aporta un amplio abanico de posibilidades a la hora del desarrollo de aplicaciones basadas en este dispositivo.

La extracción de datos por parte de aplicaciones de terceros se debe realizar forzosamente a través de *LMS*. El servicio recibe los paquetes de datos a través del bus *USB* para su procesado y posteriormente los envía a las aplicaciones que se estén ejecutando este servicio. Por defecto, *LMS* únicamente envía los datos capturados a programas que se estén ejecutando en primer plano, si bien, aquellos programas ejecutándose en segundo plano también pueden solicitar datos al *LMS*.

El servicio de *Leap Motion* se configura a través de un programa propio del controlador, el cual está compuesto por un panel de control que se puede encontrar en la barra de tareas de *Windows*. Cabe destacar que el programa de control de servicio se ejecuta de forma independiente al *LMS*, permitiendo la configuración del mismo. En la Figura 2.5, se puede observar un esquema de la interfaz nativa del *Leap Motion*.

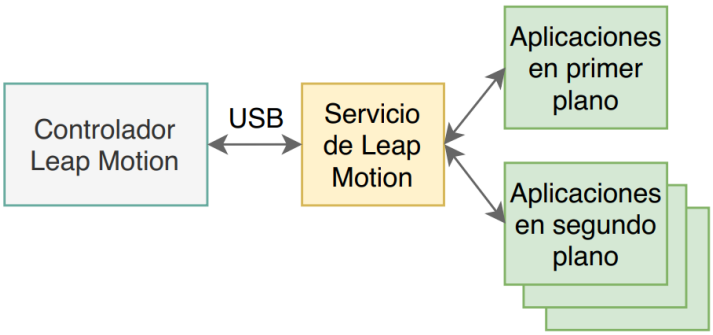


Figura 2.5.: Esquema Interfaz nativa [26]

La segunda interfaz de programación (*API*) está basada en *web socket*, en la que se pretende realizar la extracción de datos a través de servidores *web* y navegadores *web*. La comunicación entre cliente-servidor se realiza mediante la utilización del protocolo de comunicación *HTTP*. El servicio de *Leap Motion* utiliza un *web socket* en *localhost* para

realizar el envío de los datos extraídos del controlador al servidor web. El control y configuración de la comunicación vía *web socket* se realiza mediante la utilización de archivos *JSON* (*JavaScript Object Notation*), que se define como un tipo de formato específico para el intercambio de datos, con una amplia utilización en cuanto a comunicaciones servidor-cliente [25].

Por otra parte, el navegador *web* que actúa como cliente debe realizar las peticiones correspondientes para la obtención de los datos extraídos de controlador. La conexión por parte del cliente se realiza mediante la utilización de un *JavaScript client library* denominada `leap.js`, la cual recibe los archivos *JSON* y reproduce los datos como objetos de *JavaScript*. El cliente posee la capacidad de realizar el envío de mensajes de configuración al servidor *web*, como se muestra a continuación en la Figura 2.6.

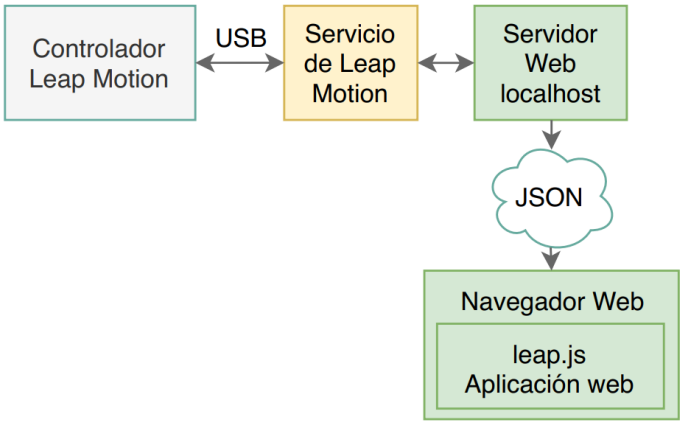


Figura 2.6.: Esquema Interfaz WebSocket [26]

2.1.3. Adquisición de datos:

El dispositivo *Leap Motion* rastrea la posición de las manos para posteriormente extraer los datos asociados a dicha posición en cada instante. Los datos son presentados en forma de objeto *Frame*. Este objeto se compone de una serie de entidades que son monitoreadas, como ejemplo, se pueden mencionar las manos, dedos o huesos. En la Figura 2.7, se observa cómo a partir del objeto *Frame* se realiza la extracción de las imágenes y manos, así como la estructura interna de la *API*.

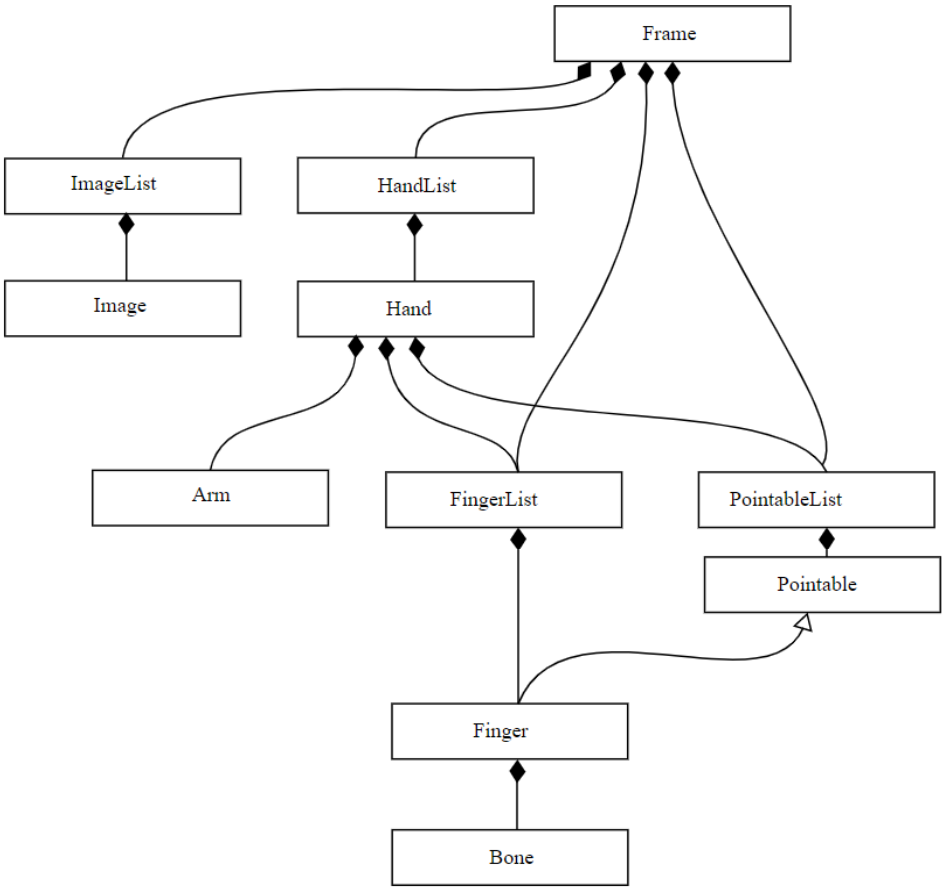


Figura 2.7.: Tracking model [27]

La mano es la principal entidad monitoreada por el dispositivo. Esta se representa con la clase *Hand*, que es la encargada de proporcionar el acceso a los datos extraíbles de las manos, principalmente la posición de las mismas, así como otros datos relacionados. A su vez, la utilización de esta clase permite la extracción de datos referentes, tanto al brazo adyacente de la mano detectada, como a los dedos que componen dicha mano.

Debido a que el dispositivo está basado en sistemas ópticos, existen ciertas circunstancias en las que el controlador no es capaz de monitorear correctamente la posición de las manos. Cuando el controlador no sea capaz de tener contacto directo con los dedos, éste supondrá la posición de los mismos, pudiendo llevarse a cabo una extracción errónea de datos en dichas posiciones de la mano. Las predicciones se basan en observaciones realizadas previamente para, en este caso, estimar la posición de los dedos.

El dispositivo *Leap Motion* está capacitado para el reconocimiento de objetos que posean características similares a los dedos de la mano. Estos objetos serían cilíndricos y alargados, como por ejemplo un lápiz. Sin embargo, debido a los objetivos de desarrollo del presente *TFM*, no será necesario el monitoreo de los mismos. La extracción de datos de los dedos se realiza a través del objeto *Hand*, que se extrae a su vez de cada *Frame*. Los dedos se definen a través de la clase *Fingers,* y estos se encuentran organizados en un *array* que los contiene y permite su utilización. Están estructurados por tipo, según el nombre de cada dedo, comenzando por el pulgar y finalizando por el meñique.

La entidad *Bone* contiene las definiciones de las principales características de los huesos de los dedos, que a su vez se encuentran englobados en la entidad *Fingers*. La extracción de datos se ha de realizar obligatoriamente a partir de un objeto *Hand*. De forma similar, los huesos de la mano se organizan en un *array* que contiene y ordena los huesos desde la base de la mano hasta la punta de los dedos como se describió previamente.

Una vez realizada la extracción de datos correspondiente para cada caso, el *software* propio de *LMC* realiza la combinación de sendos datos con un modelo interno. El modelo representa la anatomía de una mano humana. La utilización de dicho modelo pretende mejorar la monitorización ayudando a ajustar correctamente la extracción de datos al modelo, y así evitar condiciones en las que, por ejemplo, se pudiera monitorear incorrectamente la posición de la mano. Dicho modelo se basa en la representación mostrada en la Figura 2.8, donde se puede observar un resumen de la estructura de una mano en *Leap Motion*.

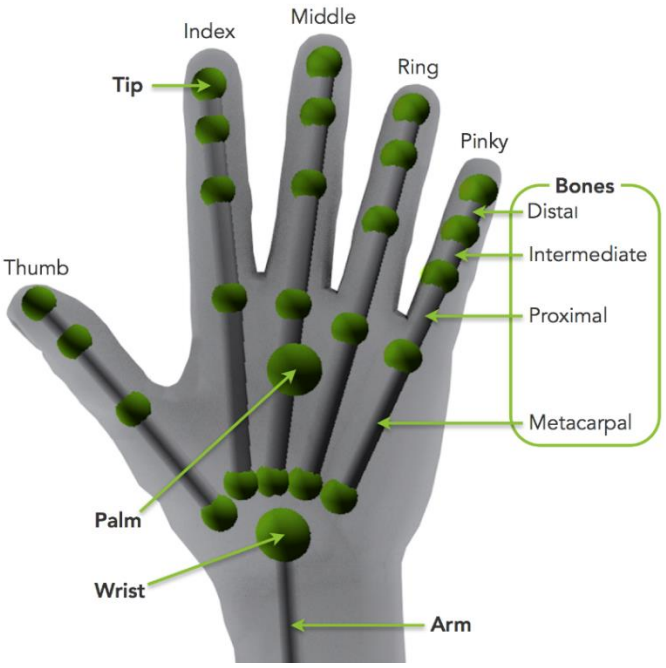


Figura 2.8.: Jerarquía de una mano [28]

Cabe destacar que el dispositivo *Leap Motion* está capacitado para la monitorización de más de una mano al mismo tiempo. Sin embargo, no se recomienda, ya que la calidad del monitoreo de las manos puede decrecer con cierta significancia.

2.2. Herramientas Software

En este apartado se introducen las herramientas *software* empleadas en el desarrollo del sistema. Las herramientas empleadas son el programa *CMake,* y *Visual Studio*.

2.2.1. CMake

La gran diversidad de lenguajes de programación hace que resulte poco eficiente la compilación de código a través de diferentes plataformas. El método actual requiere de un compilador para cada tipo de entorno, resultando en un proceso poco eficiente que, además, no fomenta la reutilización de código. [29]

El *software CMake,* de la empresa *KITWARE,* se basa en una herramienta de código abierto que controla el proceso de compilación. Al contrario que en otros sistemas, *CMake* se diseñó para ser usado en conjunción a el *software* de compilación nativo. Para la

"construcción" de programas, se deben utilizar archivos de configuración simples ubicados en el directorio raíz, denominados `CMakeLists.txt`.

El funcionamiento detrás de la herramienta también es distinto a cualquier *software* de compilación, ya que *Cmake* no "construye" los archivos programados en sí. En vez de esto, el *software* genera otro tipo de archivos, los cuales pueden ser `makefiles` o archivos de proyecto dependiendo de la plataforma nativa en la que se ejecuta. Para el caso específico del desarrollo del presente *TFM*, los archivos de proyecto que se generan son los pertenecientes a *Visual Studio*, ya que el sistema nativo utilizado será *Windows*. [29]

El objetivo que se pretende cumplir con la utilización de esta herramienta se basa en la simplificación de la compilación multiplataforma. En lugar de ser necesaria la utilización de archivos de compilación individuales para cada plataforma, *Cmake* únicamente necesita de un conjunto de archivos de entrada para llevar a cabo el proceso de compilación en cualquier plataforma.

El *software* se puede ejecutar, tanto a través de la línea de comandos, como a través de una interfaz gráfica de usuario. Las interfaces permiten a un usuario realizar la configuración del programa de forma visual, almacenando dicha configuración en un archivo caché. En la Figura 2.9, se observa esta interfaz gráfica.

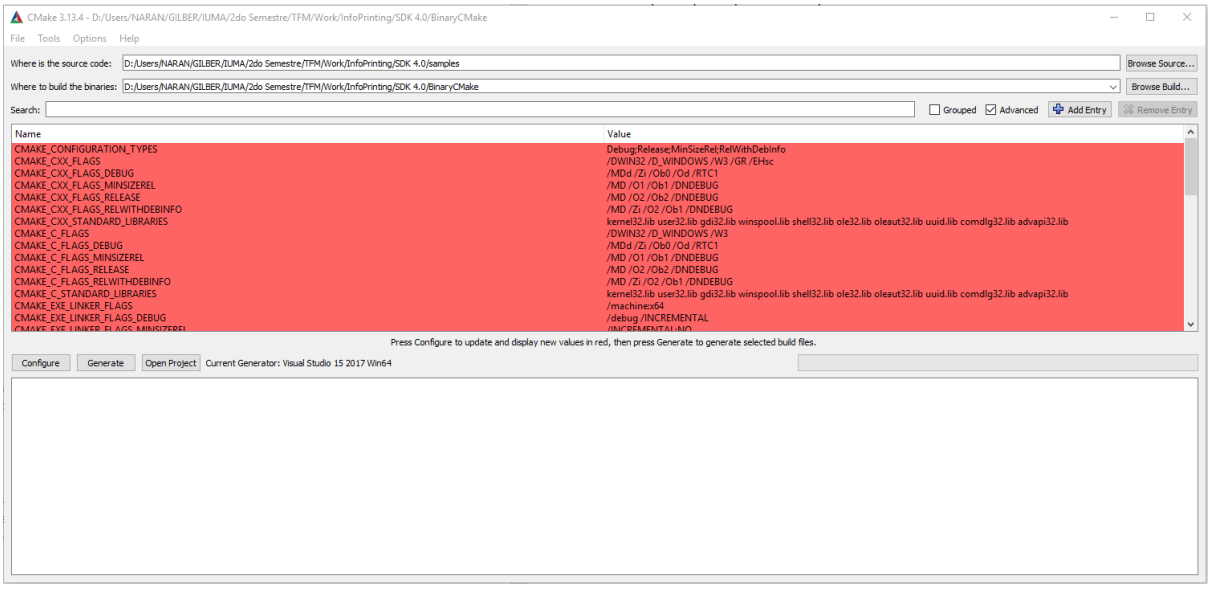


Figura 2.9.: CMake GUI

2.2.2. Visual Studio

El *software* principal de desarrollo es *Visual Studio,* de la empresa *Microsoft*. La elección de la utilización del programa para el desarrollo del *TFM* viene dada por el uso del programa *CMake*, el cual, como se comentó previamente, exporta los archivos de salida directamente en forma de proyecto de *Visual Studio*.

Los entornos de desarrollo integrado (*IDE*) se basan en programas multifuncionales que aportan servicios para facilitar el desarrollo de aplicaciones *software*. En general, estos entornos únicamente proveen de un editor de código y un depurador, si bien, el *IDE* que *Visual Studio* aporta incluye un compilador, así como herramientas de finalización de código y de diseño gráfico. En la Figura 2.10, se muestra el entorno de desarrollo del programa, señalando algunas funciones básicas del mismo.

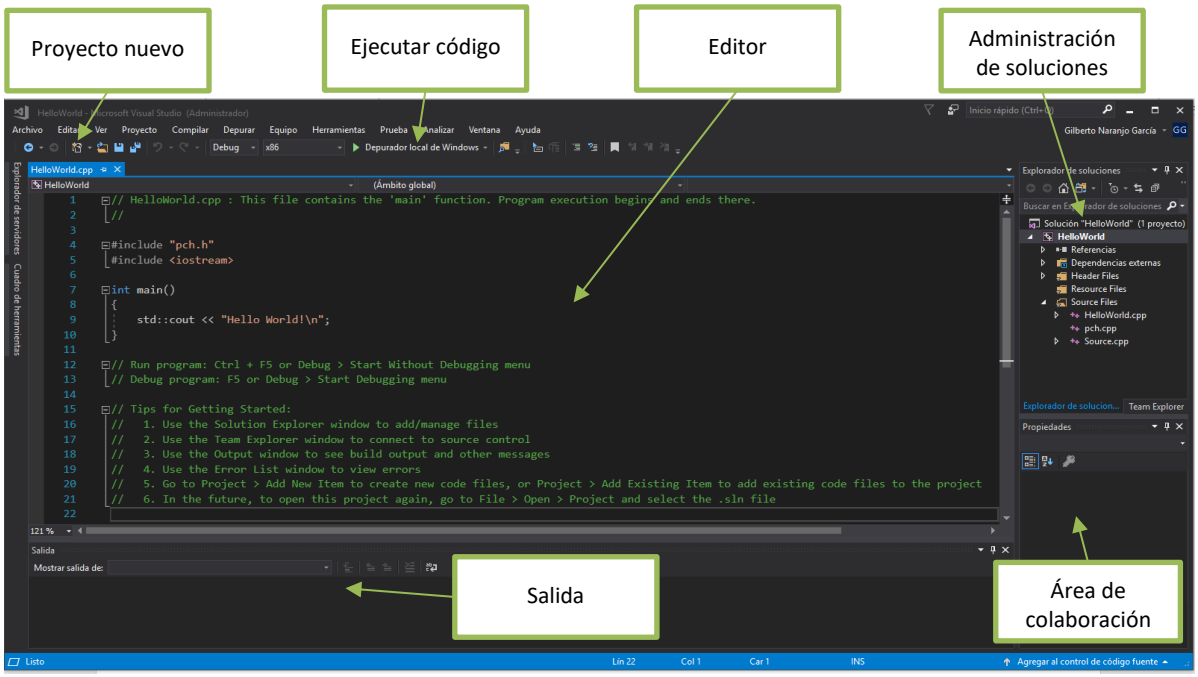


Figura 2.10.: Visual Studio IDE

Como se observa en la Figura 2.10, el administrador de soluciones se encuentra en el panel derecho del *IDE* y permite a un usuario navegar y administrar los archivos, carpetas y soluciones que estén en uso en se momento. En la parte inferior derecha se encuentra el área de colaboración que proporciona la capacidad de compartir documentos para su desarrollo, así como permitir la utilización de tecnologías como *Git* para el control de versiones.

La parte central del *IDE* se reserva para el desarrollo de código en sí. Este apartado provee de la capacidad de la visualización de archivos de código para llevar a cabo su edición, así como el desarrollo de interfaces gráficas, como pudiera ser la ventana de una aplicación *software*. En la parte inferior se puede apreciar el cuadro del diálogo de salida. En este se puede observar la salida tanto del proceso de compilación como de la depuración, así como incluye una ventana de símbolo de sistema para realizar las actividades pertinentes.

En la parte superior se encuentran algunos de los botones atajos más importantes para el desarrollo de las aplicaciones. Estos son el botón de creación de un nuevo proyecto y otro para la depuración y ejecución del código que se esté desarrollando. Cabe destacar que el *IDE* incluye características de productividad, como es el módulo *IntelliSense,* que permite la visualización de ayudas sobre el código en vivo.

Capítulo 3. Clasificación

En este capítulo se realiza una descripción de los diferentes métodos de clasificación, con el fin de comprender las principales características que componen cada método. Se hace hincapié en el estudio de las máquinas de soporte de vectores (*SVM*), ya que será el implementado en el presente TFM.

3.1. Tipos de clasificación

La clasificación se basa en un proceso por el que, dado un conjunto de datos (*data set*) extraídos en bruto, se realiza una serie de operaciones matemáticas para catalogar las instancias que componen el mismo. Las principales técnicas de clasificación pueden dividirse en tres tipos: clasificación supervisada, no supervisada y semi supervisada.

Se dice que un sistema utiliza clasificación supervisada cuando se parte de un *dataset* que contenga instancias que han sido identificadas y etiquetadas a priori, por lo que el clasificador realizará una predicción basada en los atributos que definen las características de los datos que se pretendan clasificar. A su vez, este tipo de clasificación tiene una ventaja añadida, ya que el modelo generado puede ser evaluado y así realizar una comprobación de la calidad de la clasificación mediante la utilización de parámetros específicos, como puede ser *accuracy*.

Por ende, la clasificación no supervisada es aquella en la que las instancias no se han etiquetado previamente y se basa en el agrupamiento (*clustering*) de diferentes datos. Se puede afirmar que en este tipo de clasificación se realiza una formación en grupos cuyas instancias compartan características comunes. Además, se fijan los requisitos inicialmente y el resultado no puede ser evaluado.

La clasificación semi supervisada combina la utilización de técnicas, tanto clasificación supervisada como no supervisada. En ella, se utilizan instancias previamente etiquetadas y no etiquetadas a lo largo del proceso de entrenamiento del clasificador. El desarrollo de este tipo de clasificación se llevó a cabo principalmente debido a que el etiquetado de instancias resulta considerablemente costoso y resulta imposible en ciertas aplicaciones [31].

En la Tabla 2 se muestra un resumen de algunos de los diferentes tipos de métodos de clasificación. En general, en minería de datos los métodos supervisados son los más utilizados.

Método	Descripción	
***K-Nearest Neighbour* (*K-NN*)**	Este método de clasificación mide distancias con distintas métricas para agrupar instancias, por ejemplo, la distancia Euclídea o el coeficiente Jaccard. El mismo no genera un modelo, utiliza el *data set* entero como referencia para clasificar instancias nuevas. La letra K, es un numero entero que indica la cantidad de vecinos que el algoritmo tendrá en consideración para la clasificación.	Supervisado
***Artificial Neural Network* (*ANN*)**	El método de clasificación pretende emular el funcionamiento del cerebro humano. A partir de un *set* de entrada, la red neuronal realiza una distribución capa a capa, el camino a seguir dentro de la red se define la estimulación de neuronas a partir de los nodos anteriores. Para los algoritmos de propagación inversa, al llegar a la última capa se realiza una comprobación entre el *set* de entrada y salida, en caso de que hubiera diferencias, se calcularía un error que posteriormente se propagaría por la red cambiando el valor del peso de cada nodo.	

***Support Vector Machine* (*SVM*)**	El algoritmo de clasificación *SVM* se basa en el principio teórico de aprendizaje estadístico. El mismo se engloba dentro de los clasificadores de margen máximo, ya que *SVM* genera un hiperplano de margen máximo para ordenar las instancias en grupos con características similares. A partir de un modelo generado. Este algoritmo posee una gran eficiencia a la hora de adaptarse a instancias con un amplio número de atributos.	
Centroid Classifier	Al contrario que con el método K-NN, este algoritmo realiza una media aritmética de vectores para averiguar el centroide de cada grupo de instancias. A partir de ello, se genera un modelo que será posteriormente usado para la clasificación de instancias de prueba. Un algoritmo típico es el *K-means* el cual emplea la K como referencia de la cantidad de clases en las que se quiere clasificar. En general, es uno de los algoritmos más fáciles de implementar.	
***Principal Component Analysis* (*PCA*)**	El análisis de componentes principales es un método que principalmente se utiliza para la reducción de dimensiones o extracción características. Se basa en un procedimiento matemático que utiliza la transformación ortogonal para convertir una serie de instancias, que en principio pudieran tener alguna correlación entre ellas, a instancias incorreladas denominadas componentes principales.	No Supervisado
***Gaussian Mixture Model* (*GMM*)**	El modelo de mezcla Gaussiano se basa en un modelo estadístico con el que se estima de forma paramétrica la distribución de variables aleatorias, haciendo uso de la distribución normal Gaussiana.	

Tabla 2.: Tabla-resumen de algoritmos de clasificación [30] [32] [33] [34]

En el desarrollo del presente *TFM* se selecciona la máquina de vectores de soporte para realizar la clasificación de los símbolos del alfabeto dactilológico español. Esto se debe a que este tipo de clasificador aporta ciertas ventajas específicas, como, por ejemplo, que una vez obtenida una solución, esta sea única o global. Otra ventaja específica radica en la simplificación del número de parámetros de ajuste disponible en este tipo de clasificador.

3.2. Algoritmos SVM

Las primeras redes de vectores de soporte fueron introducidas por Vladimir Vapnik y sus colaboradores en 1995. A partir de las mismas se ha desarrollado diferentes enfoques hacia las máquinas de vectores de soporte.

SVM se incluye dentro de los algoritmos de clasificación no supervisada, en las cuales se conoce la identidad de cada instancia de entrada. Para este tipo de algoritmo las instancias de entradas se dividen en dos *sets* de datos independientes, uno para realizar el entrenamiento de la máquina, y otro para la realización de pruebas de verificación.

El proceso de entramiento se basa en la utilización de un *set* de datos (*train set*) que define instancias según una serie de atributos, para realizar el entrenamiento de la máquina y que así, se genere un modelo único capaz de identificar las diferentes instancias del *set*. Por otra parte, el proceso de testeo se lleva a cabo mediante el uso de un *set* de datos especifico (*test set*), el cual contiene instancias diferentes al *set* anterior, pero organizadas de la misma forma. A partir de lo comentado, se pretende categorizar las instancias del *test set* según el entrenamiento realizado previamente.

En general, para la utilización de este algoritmo es necesario realizar un procesado de los datos que serán utilizados como *data set*. Más específicamente, *SVM* requiere que los datos de entrada se proporcionen en forma de vectores de números reales, por lo que, si se pretendiese realizar la clasificación de instancias con atributos categóricos, estos deberían ser previamente convertidos a valores numéricos. A su vez, la realización de un escalado previo de las instancias se considera fundamental para obtener un alto grado de precisión en la clasificación. La ventaja principal que se aporta mediante el escalado de datos se basa en evitar la disparidad entre los rangos numéricos que describen las instancias. [35]

SVM pertenece a la familia de clasificadores lineales, lo cual se debe al uso de hiperplanos de separación óptimos empleados en *set* de datos con instancias separables. A su vez, los hiperplanos óptimos se definen como una función de decisión lineal con un margen de separación máximo entre los vectores de las clases que se pretende clasificar. Como se observa en la Figura 3.1, la construcción del hiperplano únicamente hace uso de algunas instancias del *set* de entrenamiento, en concreto, las instancias ubicadas en las fronteras, para la creación de los márgenes de las clases. Dichos márgenes se denominan vectores de soporte. [36]

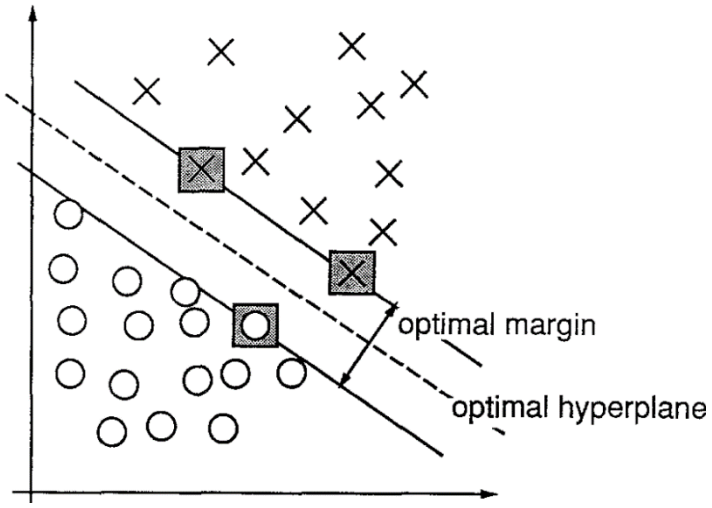


Figura 3.1.: Ejemplo de separación de clases mediante Hiperplano óptimo

A pesar de que *SVM* se considera un algoritmo perteneciente a la familia de los clasificadores lineales, en la práctica la mayoría de las aplicaciones de este algoritmo no presentan linealidad, lo que quiere decir que los *sets* de datos que se utilizarán para la generación del modelo no son separables linealmente. Es por ello por lo que se requiere de la utilización de *kernels*. Mediante la utilización de estas funciones se consigue trabajar con algoritmos complejos que permiten, a su vez, la utilización del espacio transformado de alta dimensionalidad con patrones de datos no lineales. [37] Algunas de las funciones *kernel* más utilizadas en *SVM* son:

- Lineal: $K(x_i, x_j) = x_i^T x_j$ (1)
- Polinomial-homogénea: $K(x_i, x_j) = (x_i * x_j)^n$ (2)
- Función base radial (RBF): $K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \gamma > 0$ (3)
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma < x_i, x_j > +\tau)$ (4)

La clasificación que se realiza en este *TFM* presenta una serie de complicaciones asociadas al problema que se pretende solucionar. En este caso, se pretende clasificar 30 clases diferentes, correspondientes con cada símbolo del alfabeto dactilológico español. Las instancias para clasificar no pueden separarse linealmente, por lo que es necesario la utilización de una función *kernel* para llevar a cabo la clasificación de forma precisa, seleccionándose la función base radial (*Radial Basis Function, RBF*) (*3*). La elección de esta función se basa en las ventajas que presenta con respecto a algunos otros *kernels*, como, por ejemplo, la capacidad de trabajar con un *set* de datos no separables linealmente. Otra ventaja que aporta esta función es que necesita de un número menor de parámetros de ajuste que en otros casos. [35] La selección de esta función *kernel* se basa en el estudio previo realizado en [39].

Existen dos parámetros de ajuste relacionados con la función *RBF*: *C* y *Gamma*. Ambos parámetros juegan un papel crucial en el rendimiento del algoritmo de clasificación y por ello deben ser calibrados correctamente. El uso de estos parámetros sin calibrar puede conducir a una sobreadaptación del modelo con respecto al *set* de datos, lo que causaría que el modelo generado se ajustase fielmente al conjunto de datos de entrenamiento y en consecuencia, dicho modelo no generalizaría ante nuevas instancias, únicamente predeciría correctamente las instancias del *set* de entrenamiento.

El parámetro *Gamma* pertenece implícitamente a la función del *kernel* y se define como una medida de influencia que una única instancia de entrenamiento ejerce sobre la función. Se dice que para valores bajos de *Gamma* el hiperplano de decisión corta a través de las instancias de una forma más lineal, mientras que para valores altos del mismo el hiperplano, se ajustará mejor a las instancias. Para la generación del hiperplano con valores bajos de *Gamma* se tendrán en cuenta las instancias más alejadas del mismo, mientras que, para valores altos, las instancias más próximas al hiperplano tendrán un mayor peso. En la Figura 3.2, se muestra un ejemplo de valores de *Gamma* diferentes.

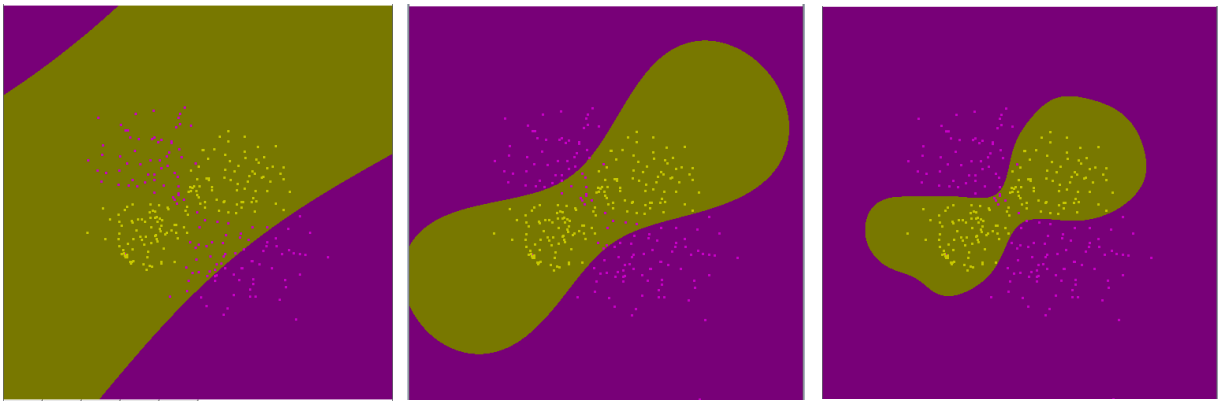


Figura 3.2.: Ejemplo de clasificación con kernel RBF y gamma: 1, 10, 100

Por otro lado, el parámetro *C* se define como el compromiso entre un hiperplano ajustado correctamente a cada instancia, lo que podría producir sobre adaptación por parte del modelo, o un hiperplano con un margen óptimo, con lo que se podría diferenciar mejor entre una clase de instancia y otra. Comúnmente se conoce como la cantidad de error y establece la proporción de instancias mal clasificadas correctamente. Cuanto mayor sea el valor de *C*, menor es la cantidad de instancias mal clasificadas que se toleran, mientras cuanto menor sea el valor, se tolera mayor cantidad de instancias mal clasificadas, y por consiguiente se suaviza el hiperplano de separación. En la Figura 3.3, se muestra una imagen con valores diferentes del parámetro *C*.

Figura 3.3.: Ejemplo de clasificación con *kernel* RBF y C: 10, 1000, 100000

Existen multitud de técnicas para realizar una búsqueda y obtención de los parámetros *C* y *Gamma* idóneos para cada caso de clasificación, como por ejemplo *cross-validation*. Sin embargo, para el desarrollo específico de este *TFM* se parten de valores preestablecidos, los cuales son: $C = 100$ y $\gamma = 0.0001$ [39].

3.3. LIBSVM

La creación del *software LIBSVM* por parte de Chih-Jen Lin et al., ha supuesto la introducción de una herramienta de utilización y desarrollo libre, mediante la cual se pretende facilitar la implementación de este tipo de algoritmos en aplicaciones actuales. Esta librería incluye archivos para la implementación del mismo dentro de una aplicación de terceros, además de incluir programas precompilados para la ejecución de ciertas de funciones a través de la línea de comandos.

La librería de soporte se describe como una herramienta simple, eficiente y fácil de usar, para la clasificación y regresión basada en *SVM*. La misma se desarrolló para soportar distintos tipos de tareas de aprendizaje dentro de las máquinas de vectores de soporte. Específicamente, *LIBSVM* soporta:

- *SVC*: *Support Vector Classification.*
- *SVR: Support Vector Regression.*
- *One-class SVM.*

SVC hace referencia a la clasificación basada en *SVM*. Este tipo soporta tanto la ordenación de dos clases como de multiclase. En este sentido, ya que para el desarrollo del *TFM* se pretende realizar la clasificación de 30 símbolos, se implementará la función de clasificación de diferentes clases. *LIBSVM* implementa el método "*one-against-one*" para la clasificación multiclase. Este método se basa en la construcción de $k * (k-1)/2$ clasificadores, donde cada uno de ellos se construye a partir de instancias de entrenamiento de dos clases elegidas a entre las k clases que componen el método. [38]

El paquete de *LIBSVM* se descarga a través de su página web oficial y sigue una estructura basada en:

- *Directorio principal*: En este subdirectorio se pueden encontrar los programas base y datos de muestra implementados en C/C++. En particular el archivo `svm.cpp`, el cual implementa los algoritmos de entrenamiento y prueba.
- *Subdirectorio de la herramienta*: En este subdirectorio se encuentran las herramientas para la comprobación del formato de los datos y la selección de parámetros para el clasificador.

- *Otros subdirectorios*: Los restantes subdirectorios contienen los archivos binarios precompilados y las interfaces a otro tipo de lenguaje o *software*.

[38]

El archivo `svm.cpp` que se encuentra en el directorio principal incluye todas las funciones y algoritmos para la realización del entrenamiento y prueba del clasificador. Las principales subrutinas incluidas para llevar a cabo ambas tareas son *svm_train* y *svm_predict*. En el mismo directorio también se encuentran los archivos que implementan la funcionalidad de *SVM* a través de la línea de comandos, correspondientes a los archivos son `svm-scale.c`, `svm-train.c` y `svm-predict.c`.

El archivo `svm-scale.c` se define como una herramienta para realizar el escalado de instancias que se encuentran en los *sets* de datos de entrada. A su vez, el archivo `svm-train.c` contiene las funciones para realizar el entrenamiento del clasificador y, a través de la ejecución de este archivo, posibilita de la creación de diferentes modelos según la selección de los valores de los parámetros de ajuste. El archivo `svm-predict.c` se compone de las funciones principales para realizar la clasificación de instancias, a partir de un modelo previamente generado. Cabe destacar que todas las funcionalidades presentes en estos archivos se recogen en la librería `svm.h`, y son descritas en el archivo `svm.cpp` con el fin de realizar la integración del sistema en otra aplicación. [39]

Algunas de las funciones más importantes declaradas y desarrolladas en el archivo `svm.cpp`, son las siguientes:

- **char svm_check_parameter():** Esta función realiza una comprobación del rango de los parámetros. Al realizar la revisión de parámetros, esta función debe ser invocada previamente a la ejecución de *svm_train()*. Si la función devuelve *NULL*, los parámetros se encuentran en un rango viable.
- **svm_model svm_train():** Esta función se utiliza para la construcción del modelo que regirá la clasificación, según los parámetros de ajuste. Asimismo, la función devuelve el modelo *SVM* generado, el cual se almacena en la estructura de datos *svm_model*.
- **svm_predict():** Esta función realiza el proceso de clasificación o regresión sobre un vector *X* de instancias de prueba, a partir de un modelo previamente generado. Para un modelo de clasificación la función devuelve una predicción de la clase de *X*.

En la Figura 3.4, se muestra la función correspondiente a svm_*predict*. En ella se evidencia que para el tipo de clasificación *ONE_CLASS*, *EPSILON_SVR* y *NU_SVR* los bytes asignados por la función *Malloc()* son únicamente 1. Así mismo se puede observar que dicha función hace uso de otra, *svm_predict_values()*.

```
double svm_predict(const svm_model *model, const svm_node *x)
{
    int nr_class = model->nr_class;
    double *dec_values;
    if(model->param.svm_type == ONE_CLASS ||
       model->param.svm_type == EPSILON_SVR ||
       model->param.svm_type == NU_SVR)
        dec_values = Malloc(double, 1);
    else
        dec_values = Malloc(double, nr_class*(nr_class-1)/2);
    double pred_result = svm_predict_values(model, x, dec_values);
    free(dec_values);
    return pred_result;
}
```

Figura 3.4.: Función svm_predict() del archivo svm.cpp

La función *svm_predict_values()* se encarga de dar valores de decisión sobre un vector de instancias de *test*, a partir de un modelo específico. En la Figura 3.5, se muestra dicha función, donde se puede observar que para su utilización con un modelo de clasificación con n clases, se generarán $n * (n-1)/2$ valores de decisión en el array *dec_values*. El número n de clases puede determinarse a través la invocación a la función *svm_get_nr_class()*. [40]

```
double svm_predict_values(const svm_model *model, const svm_node *x, double* dec_values)
{
    int i;
    if(model->param.svm_type == ONE_CLASS ||
       model->param.svm_type == EPSILON_SVR ||
       model->param.svm_type == NU_SVR)
    {
        double *sv_coef = model->sv_coef[0];
        double sum = 0;
        for(i=0;i<model->l;i++)
            sum += sv_coef[i] * Kernel::k_function(x,model->SV[i],model->param);
        sum -= model->rho[0];
        *dec_values = sum;

        if(model->param.svm_type == ONE_CLASS)
            return (sum>0)?1:-1;
        else
            return sum;
    }
    else
    {
        int nr_class = model->nr_class;
        int l = model->l;

        double *kvalue = Malloc(double,l);
        for(i=0;i<l;i++)
            kvalue[i] = Kernel::k_function(x,model->SV[i],model->param);

        int *start = Malloc(int,nr_class);
        start[0] = 0;
        for(i=1;i<nr_class;i++)
            start[i] = start[i-1]+model->nSV[i-1];

        int *vote = Malloc(int,nr_class);
        for(i=0;i<nr_class;i++)
            vote[i] = 0;
```

Figura 3.5.: Extracto de la función svm_predict_values()

Como se comentó previamente, *LIBSVM* posibilita la ejecución de las funciones declaradas en dicha librería a partir del terminal de *Windows*. Específicamente, los archivos pre compilados `svm-scale.c`, `svm-train.c` y `svm-test.c`, son los encargados de realizar los procesos de: escalado, entrenamiento y prueba, respectivamente. Para la utilización de los mismos a través de la línea de comandos se debe realizar la generación de los archivos. Por ello mediante la utilización del comando *make* en el directorio correspondiente, se generan los archivos `svm-scale.exe`, `svm-train.exe` y `svm-test.exe`.

El proceso de escalado para un *data set* específico se implementa a través del archivo `svm-scale.exe`. Para el empleo de esta función se deben seguir las instrucciones suministradas por el desarrollador de la librería, por lo que el comando a utilizar debe seguir la estructura: $svm - scale\ [opciones]\ dataSet_{entrada} > trainSet_{salida}$, donde las opciones que permiten el ajuste de características en esta función son:

- *-l* lower: límite inferior de escalado en eje *X*, por defecto -1.
- *-u* upper: límite superior de escalado en eje *X*, por defecto 1.
- *-y y_lower y_upper*: limistes de escalado en eje *Y*, por defecto sin escalado.

- *-s save_filename*: guardado de parámetros de escalado en *save_filename*.
- *-r restore_filename*: restaurado de parámetros de escalado de *restore_filename*.

La opción *-s* se utiliza para la generación de un archivo que contendrá el rango de valores para cada atributo. Este archivo se reutiliza posteriormente en el escalado del *set* de datos de entrenamiento. A su vez, los archivos denominados $dataSet_{entrada}$ y $trainSet_{salida}$ hacen referencia a los archivos de datos con extensión *txt*. Por una parte, el *set* de datos de entrada debe contener las instancias ordenadas de forma ascendente, según el formato establecido. Este tipo de clasificación se basa en incluir primero una etiqueta que describa cada instancia, seguida por un índice que describa el atributo que se empleará, y el valor de dicho atributo. La separación entre índices se representa con un espacio. Tanto el *data set* de entrada como el de salida deben seguir el formato que se ha descrito y que visualmente se representaría de la siguiente forma:

$$< etiqueta > < indice1 >:< valor1 > < indice2 >:< valor2 > \cdots$$

El proceso de entramiento del clasificador a partir del cual se genera un modelo que podrá ser posteriormente utilizado para predecir la clase de las instancias en el *set* de datos de prueba, debe seguir el formato establecido previamente. Al igual que con la función anterior, debe seguir el formato: $svm - train\ [opciones]\ trainSet_{escalado}\ [modelo]$, donde algunas las opciones más significativas son:

- *-s svm_type*: selección del tipo de *SVM*, por defecto 0 (*C-SVC*).
- *-t kernel_type*: selección del tipo de función del *kernel*, por defecto 2 (*RBF*).
- *-g gamma*: selección del parámetro gamma, por defecto $1/nº\ atributos$.
- *-c cost*: selección del parametro *C*, por defecto 1.
- *-v n*: modo *cross validation* con n iteraciones.

En este caso, las dos primeras opciones describen el tipo de *SVM* y el *kernel* que se desea utilizar mientras que las opciones *-g* y *-c* se usan para establecer los parámetros de ajuste del clasificador. Con la selección de la opción *-v* se divide el *data set* correspondiente en *n* partes iguales y se le aplica el método validación cruzada, para obtener el valor de *accuracy*. La sentencia posterior hace referencia al *set* de datos de entramiento ya

previamente escalado, y a continuación se debe incluir el nombre del modelo que se pretende generar.

Una vez obtenido el modelo que describe el clasificador, ya se pueden realizar predicciones con el mismo. El archivo `svm-predict.exe` contiene las funciones necesarias para llevar a cabo la clasificación de datos. Para la ejecución del archivo se debe seguir, una vez más, el formato aportado por el desarrollador. Por lo que el formato que se debe seguir para realizar la ejecución se corresponde con el que se muestra a continuación: $svm - predict\ [opciones]\ testSet_{escalado}\ modelo\ archivo_{salida}$. Donde las opciones que pueden ser incluidas son:

- *-b probability_estimates*: activar (1) o desactivar (0) la predicción de estimaciones de probabilidad, por defecto 0.

Para llegar a realizar la clasificación de un *set* de datos correctamente se debe tener en cuenta que el clasificador hace uso de un *set* de datos de prueba que debe ser escalado previamente, por lo que $testSet_{escalado}$ hace referencia a dicho archivo de datos. Cabe destacar que las etiquetas numéricas incluidas en el archivo de *test*, únicamente son utilizadas para el cálculo de parámetros de verificación, como es *accuracy*, o en el cálculo de errores. Posteriormente, en la función se debe incorporar un modelo valido generado a partir de la función expuesta anteriormente. La predicción de un *set* de datos genera a su vez un archivo de salida, donde se contempla la predicción realizada. A partir de este archivo se genera el valor del parámetro *accuracy* o precisión del sistema.

Capítulo 4. Desarrollo de la plataforma Inicial

Una vez presentados los componentes, tanto *hardware* como *software*, y el método de clasificación seleccionado, en el presente capítulo se presenta el enfoque inicial para el desarrollo de la plataforma que pretende realizar el reconocimiento del alfabeto dactilológico de la lengua de signos española.

4.1. Selección de datos

El *SDK* incluido con el dispositivo *Leap Motion* incorpora una serie de ejemplos que pretenden servir como elemento de ayuda para comenzar el desarrollo *software*. La parte inicial del desarrollo de la plataforma consistió en determinar los parámetros que pudieran ser de interés para discriminar entre los diferentes gestos asociados al alfabeto dactilológico de la lengua de signos española.

La selección de datos se llevó a cabo en base a la información disponible sobre los parámetros extraíbles a partir del dispositivo *Leap Motion*, para posteriormente integrarla en un código cuya función principal se basaba en la impresión por pantalla de dichos datos y de esa manera poder realizar una verificación de la concordancia de los mismos. Cabe destacar que el *SDK* incluido únicamente permite la extracción de la rotación de los dedos de la mano en forma de cuaterniones, que serán introducidos posteriormente. Así, en la Tabla 3, se muestran los parámetros considerados inicialmente en el desarrollo de la plataforma.

Dato	Descripción	Estructura de datos
bones.rotation	Rotación de los huesos de la mano.	LEAP_QUATERNION
palm.normal	Eje normal a la palma de la mano	LEAP_VECTOR
palm.direction	Vector unitario que apunta desde la posición de la palma hacia los dedos.	LEAP_VECTOR
palm.orientation	Un cuaternión que representa la orientación de la palma.	LEAP_QUATERNION
yaw	Rotación en el eje Y.	LEAP_VECTOR
pitch	Rotación en el eje X.	LEAP_VECTOR
roll	Rotación en el eje Z.	LEAP_VECTOR

Tabla 3.: Parámetros extraíbles de PrintingVariables.c

En versiones anteriores del *SDK* de *Leap Motion* se incluían funciones para el cálculo del *raw*, *pitch* o *roll* que, junto con la información de la posición de los dedos y de la palma de la mano, representan los parámetros que se consideraron en primer lugar con el fin de poder clasificar los gestos correspondientes al alfabeto dactilológico de la lengua española. Sin embargo, en el actual se partió de parámetros básicos para construir una función matemática que describiese la rotación en los distintos ejes de coordenadas del dispositivo. *Yaw* se define como el ángulo entre la parte negativa del eje *Z* y la proyección del vector en el plano *X-Z*, mientras que el *pitch* se define como el ángulo entre la parte negativa del eje *Z* y la proyección del vector al plano *Y-Z*. Por su parte, *Roll* se define como el ángulo que forma la parte negativa del eje Y y la proyección del vector sobre el plano *X-Y*. [41] En la Figura 4.1, se muestran los gráficos representativos de los tres parámetros de rotación indicados.

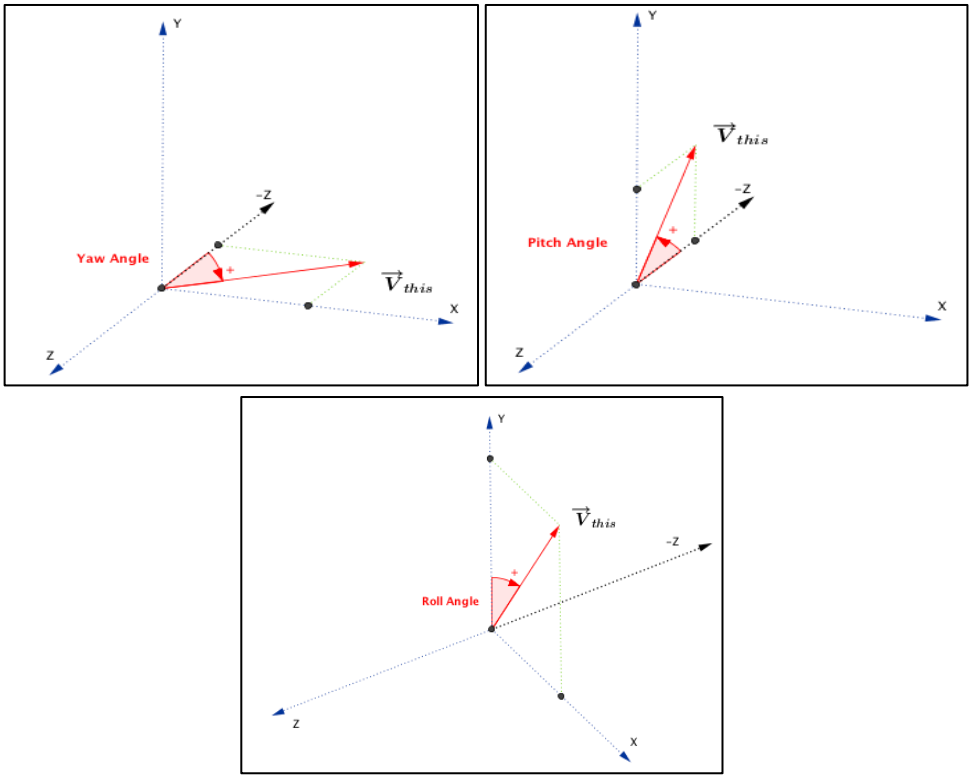


Figura 4.1.: Yaw, Pitch & Roll [41]

Una vez realizada la selección de los datos que se utilizarían en la primera versión de la aplicación *software* fue necesaria la extracción de los mismos. Por ello, en la Figura 4.2, se observa el archivo `PollingSample.c` desarrollado en el que simula la función *loop* mediante la ejecución de un bucle infinito. En cada iteración del bucle se realiza una llamada a la función *GetFrame()* para, en caso de que existiese un nuevo *frame*, actualizar el mismo y extraer algunos datos básicos. En vez de utilizar las funciones *callback*, el archivo hace uso de la función propia con el fin de que resultase más conveniente en situaciones donde no existiese un bucle *loop*.

```c
#include "LeapC.h"
#include "ExampleConnection.h"
int64_t lastFrameID = 0; //The last frame received
int main(int argc, char** argv) {
  OpenConnection();
  while(!IsConnected)
    millisleep(100); //wait a bit to let the connection complete
  printf("Connected.");
  LEAP_DEVICE_INFO* deviceProps = GetDeviceProperties();
  if(deviceProps)
    printf("Using device %s.\n", deviceProps->serial);
  for(;;){
    LEAP_TRACKING_EVENT *frame = GetFrame();
    if(frame && (frame->tracking_frame_id > lastFrameID)){
      lastFrameID = frame->tracking_frame_id;
      printf("Frame %lli with %i hands.\n", (long long int)frame->tracking_frame_id,
      for(uint32_t h = 0; h < frame->nHands; h++){
        LEAP_HAND* hand = &frame->pHands[h];
        printf("    Hand id %i is a %s hand with position (%f, %f, %f).\n",
               hand->id,
               (hand->type == eLeapHandType_Left ? "left" : "right"),
               hand->palm.position.x,
               hand->palm.position.y,
               hand->palm.position.z);
      }
    }
  } //ctrl-c to exit
  return 0;
}
//End-of-Sample
```

Figura 4.2.: PollingSample.c

Como se observa en la Figura 4.2, la función principal del archivo de ejemplo se encuentra integrada en la librería `ExampleConnection.h` cuyas funciones se declaran en el archivo `ExampleConnection.c`. El archivo de ejemplo utiliza esta librería de la misma forma que se utiliza la función *callback*. Sin embargo, hace uso de funciones propias, como son *GetFrame()* o *setFrame()*, como se puede observar en la Figura 4.3. Estas funciones, entre otras, son utilizadas para escribir las estructuras de datos recibidas en variables globales que permitan el acceso a la mismas directamente desde la aplicación. Mediante la utilización de *mutex* se previene que ambos hilos accedan al mismo recurso a la vez.

```c
void setFrame(const LEAP_TRACKING_EVENT *frame){
  LockMutex(&dataLock);
  if(!lastFrame) lastFrame = malloc(sizeof(*frame));
  *lastFrame = *frame;
  UnlockMutex(&dataLock);
}
/** Returns a pointer to the cached tracking frame. */
LEAP_TRACKING_EVENT* GetFrame(){
  LEAP_TRACKING_EVENT *currentFrame;

  LockMutex(&dataLock);
  currentFrame = lastFrame;
  UnlockMutex(&dataLock);

  return currentFrame;
}
```

Figura 4.3.: Función setFrame() en ExampleConnection.c

A partir del código de ejemplo se diseñó un archivo denominado `printingVariablesConsole.c`. En este se realiza una impresión por pantalla de los parámetros recogidos. En la Figura 4.4, se muestra un extracto de la función *main()* del código correspondiente.

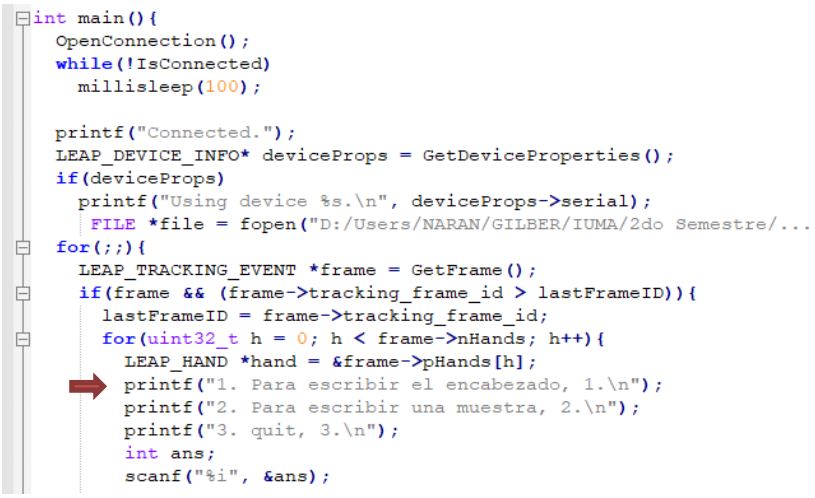


Figura 4.4.: Extracto de la función main() del archivo PrintingVariablesConsole.c

Debido a que en el archivo de ejemplo original se implementaba un bucle infinito, fue necesaria la introducción de un menú para controlar la cantidad de muestras que se escribían al archivo externo. En este se implementaron dos funcionalidades adicionales, primero la de escribir un encabezado para una mejor visualización de los datos extraídos, y segundo la de realizar la extracción de muestras en sí.

4.2. Extracción de datos

Una vez comprobada la correspondencia de los datos, se adaptó el código para realizar la impresión en un archivo *.txt* independiente, en el cual quedaran reflejados los parámetros que inicialmente se consideraron de interés para la extracción de datos.

Como se muestra en la Figura 4.5, la extracción de los datos se lleva a cabo mediante llamadas a los *array* o vectores que contienen las estructuras de datos. En este caso, el *array digits[k]* hace referencia a los dedos de la mano, así como el *array bones[l]* define los huesos dentro de los dedos de la mano. Los vectores *v[n]* y *v[o]* contienen las coordenadas en formato [*X*, *Y*, *Z*], del vector normal y dirección de la palma de la mano, respectivamente.

Debido a que tanto la rotación de los huesos como la orientación de la palma se engloba en la estructura de datos *LEAP_QUATERNION*, tanto el array *v[m]* como *v[p]* definen la rotación en el formato especifico de cuaternión, el cual es [*W*, *X, Y, Z*].

```
for(int k=0; k<5; k++){
  for(int l=1; l<4; l++){
    for(int m=0; m<5; m++){
      float this = hand->digits[k].bones[l].rotation.v[m];
      if(this >= 0){
        printf("%F", this);
        printf( "%F,  ", this);
      }else{
        printf( "%F, ", this);
      }
    }
  }
}
for(int n=0; n<4;n++){
  printf( "%F, ", hand->palm.normal.v[n]);
}
printf("%3s", "|");
 for(int o=0; o<4;o++){
printf( "%F, ", hand->palm.direction.v[o]);
}
printf("%3s", "|");
for(int p=0; p<5;p++){
  printf( "%F, ", hand->palm.orientation.v[p]);
}
printf("%3s", "|");
printf("%F", atan2(hand->palm.direction.x, -hand->palm.direction.z));
printf("%5s","|");
printf("%F", atan2(hand->palm.direction.y, -hand->palm.direction.z));
printf( "%5s","|");
printf("%F", atan2(hand->palm.direction.x, -hand->palm.direction.y));
```

Figura 4.5.: Extracto función Main() en PrintingVariablesConsole.c

La extracción de parámetros *yaw*, *pitch* y *roll* se realizó mediante la ecuación: $atan2()$ de la librería *Math.h*. Esta función calcula la arcotangente entre los argumentos de la misma. Como inicialmente se planteó incluir únicamente los parámetros asociados a la mano, los argumentos de la función parten de las componentes del vector dirección de la palma de la mano.

Una vez realizado el código, se comprobó la validez de los datos extraídos. En la Figura 4.6, se muestra un extracto de archivo de salida en formato .txt, para algunas de las pruebas iniciales realizadas. Este archivo se corresponde con la realización el símbolo asociados a la letra “A”.

```
|Indice                                                                                      |
|Proximal                                |Medio                                |Distal                                   |
|x          y          z          w          |x          y          z          w          |x          y          z          w          |
|0.209658,  -0.605813, -0.249326, 0.725859,  |-0.059455, -0.682494, 0.326460,  0.651223,  |-0.157058, -0.772101, 0.198166,  0.583029,  |
|-0.412727, -0.067449, 0.685782,  0.595659,  |-0.690884, 0.060490,  0.181625,  0.697160,  |0.990515,  0.012627,  -0.116954, -0.071003, |
|-0.407552, -0.152027, 0.642367,  0.630993,  |-0.662677, 0.082589,  0.144574,  0.730162,  |0.986565,  -0.031924, -0.106346, -0.119840, |
|-0.397989, -0.150631, 0.632843,  0.646858,  |-0.669413, 0.088057,  0.137212,  0.724780,  |0.986211,  -0.048256, -0.098682, -0.123775, |
|-0.388797, -0.131313, 0.647687,  0.641947,  |-0.655242, 0.075076,  0.137784,  0.738943,  |0.982682,  -0.013595, -0.114790, -0.144828, |
|-0.347286, -0.137000, 0.624954,  0.685606,  |-0.671391, 0.100113,  0.147186,  0.719408,  |0.983728,  -0.050669, -0.115742, -0.127731, |
|-0.433273, -0.093212, 0.678179,  0.586225,  |-0.706695, 0.060273,  0.170767,  0.683951,  |0.995239,  -0.019562, -0.080363, -0.051550, |
|-0.463600, -0.086527, 0.694875,  0.542897,  |-0.716672, 0.064240,  0.180044,  0.670700,  |0.993052,  0.015242,  -0.115624, -0.015728, |
|-0.363673, 0.012400,  0.664485,  0.652724,  |-0.612067, 0.114537,  0.161533,  0.765612,  |0.965404,  -0.010753, -0.176302, -0.191827, |
|-0.388946, -0.060573, 0.665094,  0.634588,  |-0.653109, 0.078910,  0.156638,  0.736672,  |0.986944,  -0.014936, -0.116018, -0.110716, |
|-0.376736, -0.124258, 0.645064,  0.653087,  |-0.661296, 0.105061,  0.152914,  0.726820,  |0.980862,  -0.050407, -0.125406, -0.140150, |
|-0.480620, -0.125783, 0.642550,  0.583363,  |-0.728373, 0.037619,  0.132642,  0.671166,  |0.998799,  -0.030480, -0.028073, -0.026149, |
|-0.406315, -0.004621, 0.628916,  0.662836,  |-0.697501, 0.067632,  0.119084,  0.703376,  |0.988003,  -0.011433, -0.103041, -0.114463, |
|-0.405230, -0.029452, 0.642599,  0.649606,  |-0.695660, 0.055106,  0.135837,  0.703255,  |0.988256,  0.004037,  -0.103187, -0.112634, |
|-0.404794, -0.029080, 0.647590,  0.644921,  |-0.682674, 0.057654,  0.138146,  0.715226,  |0.986807,  0.003919,  -0.106780, -0.121634, |
|-0.409378, -0.030623, 0.641681,  0.647856,  |-0.698586, 0.052987,  0.131078,  0.701419,  |0.989974,  -0.000435, -0.094046, -0.105390, |
|-0.401041, -0.088715, 0.646175,  0.643237,  |-0.687168, 0.036816,  0.139467,  0.712035,  |0.991669,  -0.007229, -0.065927, -0.110425, |
|-0.444249, -0.130635, 0.639250,  0.613951,  |-0.674688, 0.024017,  0.120875,  0.727742,  |0.993217,  0.006282,  -0.055538, -0.101958, |
|-0.380455, -0.025940, 0.680198,  0.626029,  |-0.687922, 0.039954,  0.173152,  0.703694,  |0.987509,  0.030708,  -0.112231, -0.106245, |
|-0.429727, -0.038686, 0.631793,  0.643953,  |-0.678951, 0.057031,  0.113944,  0.723042,  |0.987484,  0.013806,  -0.108727, -0.113417, |
```

Figura 4.6.: Archivo de salida a partir del código PrintingVariables.c

Concretamente en la Figura 4.6 se muestran los cuaterniones extraídos en su formato específico. Estos se asocian a la rotación de los hueso proximal, medio y distal para cada dedo de la mano. Como se puede observar, los datos asociados a los cuaterniones varían de forma significativa en algunos casos, teniendo en cuenta que se trata de un *data set* que describe una única letra. Esta variación, junto con el elevado número de parámetros que se deberían tener en cuenta y la complejidad para visualizar los datos, dificultan la implementación de los cuaterniones a la hora de realizar la clasificación de los símbolos.

4.3. Cuaterniones

Los cuaterniones se definen como una herramienta para realizar rotaciones de vectores en 3D. En *Leap Motion* las rotaciones de los huesos de la mano se representan mediante la teoría de los cuaterniones. Un cuaternión es un sistema compuesto de cuatro valores, propuestos inicialmente por Hamilton en 1844. [42] A pesar de que originalmente se plantearon para describir transformaciones en 3D, su uso no está muy extendido, lo cual se debe principalmente a la dificultad que presenta a la hora de entender su significado físico.

Para comprender los cuaterniones es necesario introducir una base algebraica de su composición. En general, como extensión de los números complejos, los cuaterniones $q(q_0, \boldsymbol{q})$ se componen de un escalar q_0 y un vector $\boldsymbol{q} = (q_1,\ q_2,\ q_3)$. Estos se representan en de la siguiente forma: $q(q_0, \boldsymbol{q}) = (q_0,\ q_1,\ q_2,\ q_3) = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$, donde $\boldsymbol{i}, \boldsymbol{j}$, $\boldsymbol{k}$ son valores imaginarios. [42]

Entre las propiedades que presentan los cuaterniones se destacan las dos principales para el desarrollo de este *TFM*. Debido a que el producto de los números imaginarios satisface la propiedad $i^2 = j^2 = k^2 = ijk = -1$, los cuaterniones heredan dicha propiedad de no conmutación, por lo que el producto entre estos se define como:

$$pq = (p_0 + p_1\boldsymbol{i} + p_2\boldsymbol{j} + p_3\boldsymbol{k})(q_0 + q_1\boldsymbol{i} + q\boldsymbol{j} + q_3\boldsymbol{k}) \quad (5)$$

$$= p_0q_0 - \boldsymbol{p}\cdot\boldsymbol{q} + p_0\boldsymbol{q} + q_0\boldsymbol{p} + \boldsymbol{p}\times\boldsymbol{q} \quad (6)$$

Siendo la función (6) una simplificación extraída a partir de la función (5). La segunda propiedad que se debe tener en cuenta es la normalización de los cuaterniones para su empleo en las matrices de rotación, cumpliéndose que:

$$|q|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \quad (7) \quad [44]$$

En la Figura 4.7 se muestra una representación gráfica de tres cuaterniones. El cuarto eje $\boldsymbol{m}$, el cual solapa al eje correspondiente a $\boldsymbol{i}$, se incorpora para poder representar el valor escalar.

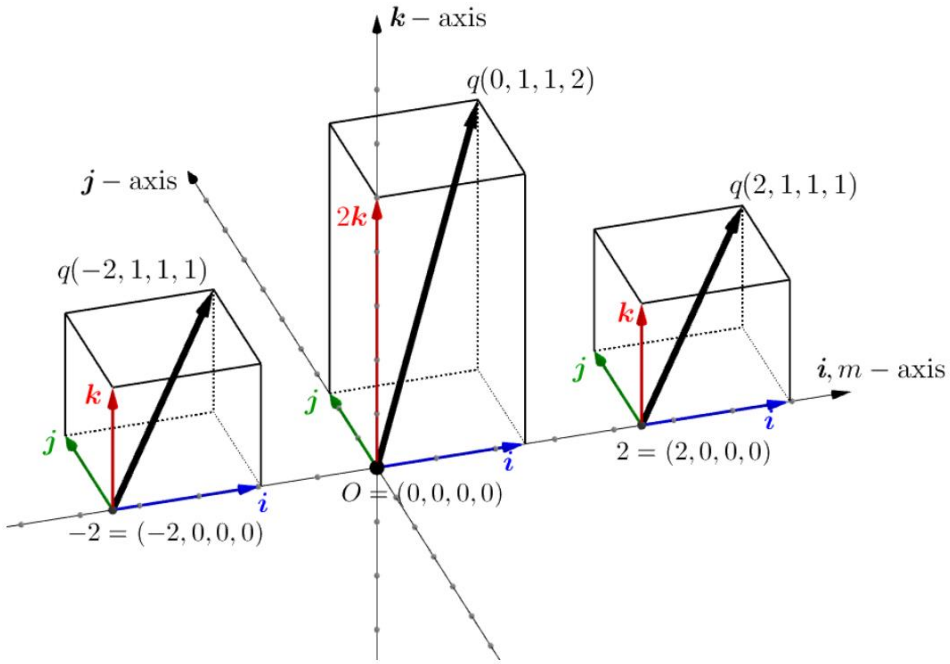


Figura 4.7.: Representación de cuaterniones: q(-2,1,1,1), q(0,1,1,2) y q(2,1,1,1) [42]

Como se ha comentado previamente, el dispositivo *Leap Motion* únicamente permite extraer la rotación de los huesos de los dedos de la mano en forma de cuaternión. Tras realizar las pruebas de correspondencia de los datos extraídos, así como tener en consideración los problemas que representaban, se concluyó que se debía aplicar un sistema de transformación de cuaterniones. Por ello se optó por introducir una matriz de rotación basada en la relación entre ángulos de Euler.

Las rotaciones en 3D pueden ser representadas a partir de matrices ortogonales de 3x3, denominadas matrices de rotación. Sin embargo, se debe tener en cuenta el teorema de rotación de Euler, en el cual se especifica que cada rotación en tres dimensiones se define a partir de un vector unitario en forma de $\boldsymbol{n} = (n_x, n_y, n_z)$ y un ángulo θ. En general, esta definición se organiza según el formato de cuaterniones, con el fin de que el producto entre ellos genere un cuaternión correspondiente a la composición de rotaciones de una matriz. Los parámetros de Euler definen la relación entre cuaterniones y la teoría de ángulos, describiéndose como: $q_0 = \cos{}^{\theta}/_{2}$, $q_1 = n_x \sin{}^{\theta}/_{2}$, $q_2 = n_y \sin{}^{\theta}/_{2}$, $q_3 = n_z \sin{}^{\theta}/_{2}$. [43]

A partir de los parámetros anteriores se establece que la matriz de rotación ortogonal de 3x3 puede ser representada en términos de cuaterniones, y viene dada por la forma:

$$R(q) = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix}$$

Como se observa en la matriz de rotación $R(q)$, esta se compone de nueve elementos, de los cuales cuatro son independientes debido a la característica ortogonal que presenta. Estos elementos de la matriz se basan en polinomios cuadráticos de variables y no en funciones trigonométricas. [43] La propiedad establecida en la ecuación (7) posibilita la simplificación de la matriz de rotación. Por ello se reorganizó la matriz $R(q)'$ para satisfacer dicha propiedad, estableciendo la matriz de rotación empleada en el desarrollo de la aplicación *software*, de la siguiente forma:

$$R(q)' = \begin{pmatrix} 1 - 2(q_2^2 + q_3^2) & -2(q_0q_3 - q_1q_2) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 1 - 2(q_1^2 + q_2^2) \end{pmatrix}$$ [45]

4.4. Reconocimiento de símbolos estáticos

A partir de los *data set* generados se planteó realizar el cambio de algunos de los parámetros que se pretendían extraer. Por ello fue necesario llevar a cabo una reorganización del código inicialmente desarrollado. Con la introducción de la matriz de rotación se pretendía incluir un nuevo parámetro para el reconocimiento de símbolos estáticos, correspondiente al ángulo de flexión de los dedos de la mano.

Una vez obtenida la matriz de rotación a partir de los cuaterniones, fue posible realizar la extracción de los vectores base de la matriz. Los vectores base se definen como un subconjunto linealmente independiente de vectores extraídos a partir de los vectores originales. Estos se describen como vectores de dirección en los ejes de coordenadas [*X*, *Y*, *Z*], por lo que al realizar la extracción del vector *Y*, según el eje de coordenadas del dispositivo *Leap Motion*, se hallaría el vector dirección para una falange especifica. A partir de una operación matemática sería posible realizar la extracción del ángulo que se encuentra en el vector dirección de una falange y el mismo de la falange siguiente, tal y como se muestra en la Figura 4.8.

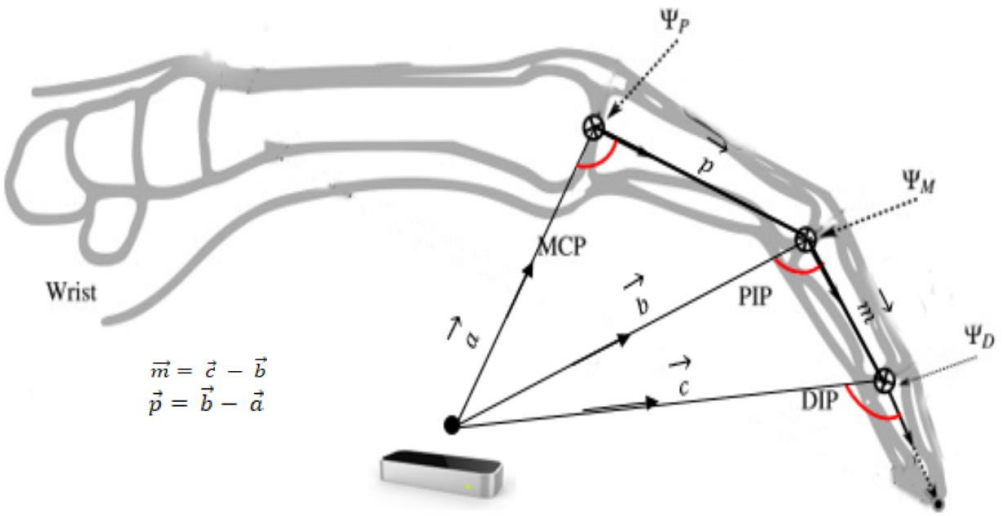


Figura 4.8.: Representación de ángulos de flexión para Leap Motion [46]

En la reorganización de dicho código, se excluyeron algunos de los parámetros que inicialmente se consideraron para realizar el reconocimiento, como son: los ángulos *yaw*, *pitch* y *roll*. La decisión de retirar este parámetro se fundamenta principalmente en la capacidad de extraer el ángulo de flexión para cada dedo la mano. A su vez se introdujo el parámetro *is_extended*. Este parámetro se encuentra englobado en la estructura de datos *LEAP_DIGIT*. El mismo devuelve un valor lógico correspondiente a si un dedo de la mano se encuentra completamente extendido o flexionado. Tras la exclusión de algunos parámetros se determinaron los datos que se recogen en la Tabla 4, que se muestra a continuación.

Dato	Descripción	Estructura de datos
bones.rotation	Rotación de los huesos de la mano.	LEAP_QUATERNION
Flexión del dedo	Parámetro del nivel de doblez de los dedos.	FLOAT
digits.is_extended	Ángulo de flexión de cada dedo.	LEAP_DIGIT

Tabla 4.: Datos extraíbles PrintingVariablesFlex.c

A partir del archivo PrintingVariablesConsole.c se realizó una adaptación que se denominó `PrintingVariablesFlex.c`. En este caso se hace uso de lo previamente descrito y como se muestra en la Figura 4.9, se realiza la extracción de los vectores base. Cabe destacar que, según el alfabeto dactilológico español, la flexión de los dedos para la mayoría de las representaciones de los símbolos se realiza a la altura de la junta denominada *PIP* (Figura 4.8), la cual se encuentra entre la falange proximal e intermedia. Por ello se decidió realizar únicamente la extracción del ángulo entre dichas falanges. A continuación, en la Figura 4.9, se muestra la porción correspondiente a la transformación de parámetros y posterior extracción de los vectores bases.

```
for(int k=0; k<5; k++){//Fingers
quat.v[0] = hand->digits[k].bones[1].rotation.x;
quat.v[1] = hand->digits[k].bones[1].rotation.y;
quat.v[2] = hand->digits[k].bones[1].rotation.z;
quat.v[3] = hand->digits[k].bones[1].rotation.w;
//Quaterniones a Matriz
vector.v[0] = (float)(1.0f-(2.0f*quat.v[1]*quat.v[1]) - (2.0f*quat.v[2]*quat.v[2]));
vector.v[1] = (float)(2.0*quat.v[0]*quat.v[1] - 2.0f*quat.v[2]*quat.v[3]);
vector.v[2] = (float)(2.0f*quat.v[0]*quat.v[2] + 2.0f*quat.v[1]*quat.v[3]);
vector1.v[0] = (float)(2.0f*quat.v[0]*quat.v[1] + 2.0f*quat.v[2]*quat.v[3]);
vector1.v[1] = (float)(1.0f - 2.0f*quat.v[0]*quat.v[0] - 2.0f*quat.v[2]*quat.v[2]);
vector1.v[2] = (float)(2.0f*quat.v[1]*quat.v[2] - 2.0f*quat.v[0]*quat.v[3]);
vector2.v[0] = (float)(2.0f*quat.v[0]*quat.v[2] - 2.0f*quat.v[1]*quat.v[3]);
vector2.v[1] = (float)(2.0f*quat.v[1]*quat.v[2] + 2.0f*quat.v[0]*quat.v[3]);
vector2.v[2] = (float)(1.0f - 2.0f*quat.v[0]*quat.v[0] - 2.0f*quat.v[1]*quat.v[1]);
matriz.m[0] = vector;
matriz.m[1] = vector1;
matriz.m[2] = vector2;
//BasisVectors de Quaterniones
vectorX.v[0] =  matriz.m[0].v[0];
vectorX.v[1] =  matriz.m[1].v[0];
vectorX.v[2] =  matriz.m[2].v[0];
vectorY.v[0] = matriz.m[0].v[1];
vectorY.v[1] = matriz.m[1].v[1];
vectorY.v[2] = matriz.m[2].v[1];
vectorZ.v[0] = matriz.m[0].v[2];
vectorZ.v[1] = matriz.m[1].v[2];
vectorZ.v[2] = matriz.m[2].v[2];
```

Figura 4.9.: Extracción de vectores base en PrintingVariablesFlex.c

Como se puede observar en la Figura 4.9, se crea la variable *quat* para almacenar los valores de rotación correspondientes al hueso proximal de cada dedo. Una vez almacenados estos valores se realiza la transformación de cuaternión a matriz de rotación. Mediante la utilización de la variable *matriz* propia de *Leap Motion*, se almacena en una matriz de 3x3 los componentes de los cuaterniones, para posteriormente realizar la extracción de los vectores base correspondientes a cada eje, a partir de la matriz de rotación. El proceso descrito se ha de realizar por segunda vez con la intención de extraer los vectores bases correspondientes a *bones[2]* que representan la falange intermedia de los huesos de la mano.

Para llevar a cabo la extracción del ángulo a partir de sendos vectores, fue necesaria la introducción de varias funciones de apoyo. En la Figura 4.10 se muestra un extracto con la declaración de dichas funciones.

```
//MagnitudeSquared
float MagnitudeSquared(LEAP_VECTOR vector){
  float ans = (float)((vector.v[0]*vector.v[0] +
                       vector.v[1]*vector.v[1] +
                       vector.v[2]*vector.v[2]));
  return ans;
}
//Dot
float dot(LEAP_VECTOR firstVector, LEAP_VECTOR secondVector){
  float ans = (float)((firstVector.v[0]*secondVector.v[0]) +
                      (firstVector.v[1]*secondVector.v[1]) +
                      (firstVector.v[2]*secondVector.v[2]));
  return ans;
}
//AngleTo
double AngleTo(LEAP_VECTOR firstVector, LEAP_VECTOR secondVector){
  double denom = MagnitudeSquared(firstVector) *
                 MagnitudeSquared(secondVector);
  if(denom <= EPSILON){
    return 0.0f;
  }
  double val = dot(firstVector, secondVector) / sqrt(denom);
  if(val >= 1.0f){
    return 0.0f;
  }else if(val <= -1.0f){
    return  M_PI;
  }
  return acos(val);
}
```

Figura 4.10.: Declaración de funciones en PrintingVariablesFlex.c

La función *AngleTo* que se muestra en la *Figura* 4.10, es la encargada de realizar la extracción del ángulo a partir de vectores. A su vez, esta función emplea las funciones d*ot* y *MagnitudeSquared* para llevar a cabo el proceso de extracción del ángulo correspondiente. La función *dot* se emplea para realizar el producto escalar de dos vectores diferentes,

mientras que la función *MagnitudeSquared* se emplea para determinar la magnitud al cuadrado o distancia de un vector específico. Para realizar la extracción del ángulo, la función *AngleTo* se basó en el cociente entre vectores, el cual se fundamenta a partir de la expresión matemática (8):

$$\cos^{-1}\left(\frac{\vec{p}\cdot\vec{m}}{|\vec{p}|\cdot|\vec{m}|}\right) \qquad (8)$$

Mediante el uso de esta función se calcula el ángulo que se encuentra en el plano formado por ambos vectores, siendo el menor de sendos ángulos conjugados el devuelto por la función. En la Figura 4.11 se muestra el extracto del código correspondiente a la función, así como a la implementación del parámetro *is_extended*.

```
    //AngleTo:
    float angle = (float)(AngleTo(vectorY, vectorYY)*RAD_TO_DEG);
    printf("Angulo (DEG): %f.", angle);
    printf("\n");
    fprintf(file, "%f, ", angle);
   }
   //is_extended:
   for(int m=0;m<5;m++){//Fingers
     fprintf(file, "%lu ,", hand->digits[m].is_extended);
   }
}else if(ans == 3){
   return 0;
}else{
   printf("Introduzca opcion valida");
   printf("\n");
}
```

Figura 4.11.: Implementación AngleTo de PrintingVariablesFlex.c

En la Figura 4.11, se observa cómo a su vez fue necesario realizar un *cast* de la función *AngleTo* con el fin de poder almacenar el ángulo de respuesta en un valor *float*, que posteriormente se imprimiría en un archivo de texto, generando así los *dataset* necesarios para realizar el proceso de clasificación. Se continuó con la implementación del parámetro *is_extended*, por lo que fue necesario incluirla dentro de un bucle *for* que recorriese todos los dedos de la mano, y verificar así el estado de flexión de los mismos. El último apartado que se encuentra en la Figura 4.11 corresponde con la tercera opción del menú creado para realizar el control del código, esta opción únicamente se utiliza para la terminar la ejecución del programa.

Para ejecutar la función *AngleTo* de forma satisfactoria es necesario incluir un parámetro que permita realizar la transformación de radianes a grados. Por ello se introdujo *RAD_TO_DEG*, parámetro fue definido al comienzo del código desarrollado, como se muestra en la Figura 4.12. Sin embargo, la función *AngleTo* hace uso de los parámetros *EPSILON* y *M_PI*, los cuales también se definen al comienzo del archivo. Como se muestra a continuación, estas tres definiciones representan valores numéricos.

```
#define M_PI 3.14159265358979323846
#define EPSILON 1.192092896e-07f
#define RAD_TO_DEG 57.295779513f
```

Figura 4.12.: Definiciones de PrintingVariables.c

A partir de la ejecución del código `PrintingVariablesFlex.c` se compusieron varios *data set*. Por una parte, como comprobación de los datos que se extrajeron en el proceso de captura, y por otro lado, para la generación de los archivos de entrada específicos utilizados en el clasificador *SVM*. En la Figura 4.13, se muestran ambos *sets* de datos.

Pulgar (DEG)	|Indice (DEG)	|Medio (DEG)	|Anular (DEG)	|Menique (DEG)	|is_extended|
(38.456734)	(84.373260)	(86.946243)	(85.860359)	(85.900192)	| 0 | 0 | 0 | 0 | 0 |
(35.776955)	(85.055557)	(85.090683)	(82.773613)	(79.280373)	| 0 | 0 | 0 | 0 | 0 |
(34.840733)	(81.239677)	(82.288773)	(80.903999)	(78.192879)	| 0 | 0 | 0 | 0 | 0 |
(35.293751)	(80.834702)	(81.004822)	(80.695885)	(78.667183)	| 0 | 0 | 0 | 0 | 0 |
(37.125454)	(85.607117)	(85.308235)	(83.304642)	(79.081497)	| 0 | 0 | 0 | 0 | 0 |
(41.193790)	(80.735847)	(83.764748)	(84.640709)	(81.895699)	| 0 | 0 | 0 | 0 | 0 |
(40.762943)	(80.771347)	(83.841675)	(83.848480)	(79.574150)	| 0 | 0 | 0 | 0 | 0 |
(41.199371)	(80.835724)	(83.788055)	(83.400093)	(77.385376)	| 0 | 0 | 0 | 0 | 0 |
(41.083416)	(84.046288)	(85.378014)	(83.932739)	(82.296677)	| 0 | 0 | 0 | 0 | 0 |
(42.605537)	(78.868446)	(80.635063)	(80.488327)	(75.664993)	| 0 | 0 | 0 | 0 | 0 |

```
0 1:38.456734 2:84.373260 3:86.946243 4:85.860359 5:85.900192 6:0 7:0 8:0 9:0 10:0
0 1:35.776955 2:85.055557 3:85.090683 4:82.773613 5:79.280373 6:0 7:0 8:0 9:0 10:0
0 1:34.840733 2:81.239677 3:82.288773 4:80.903999 5:78.192879 6:0 7:0 8:0 9:0 10:0
0 1:35.293751 2:80.834702 3:81.004822 4:80.695885 5:78.667183 6:0 7:0 8:0 9:0 10:0
0 1:37.125454 2:85.607117 3:85.308235 4:83.304642 5:79.081497 6:0 7:0 8:0 9:0 10:0
0 1:41.193790 2:80.735847 3:83.764748 4:84.640709 5:81.895699 6:0 7:0 8:0 9:0 10:0
0 1:40.762943 2:80.771347 3:83.841675 4:83.848480 5:79.574150 6:0 7:0 8:0 9:0 10:0
0 1:41.199371 2:80.835724 3:83.788055 4:83.400093 5:77.385376 6:0 7:0 8:0 9:0 10:0
0 1:41.083416 2:84.046288 3:85.378014 4:83.932739 5:82.296677 6:0 7:0 8:0 9:0 10:0
0 1:42.605537 2:78.868446 3:80.635063 4:80.488327 5:75.664993 6:0 7:0 8:0 9:0 10:0
```

Figura 4.13.: Archivos de salida para PrintingVariablesFlex.c

En la *Figura* 4.13 se muestran los archivos de salida correspondientes a la extracción de datos a partir de la realización del gesto asociado al símbolo "A" en el alfabeto dactilológico de la lengua de signos española. Como se puede comprobar el *set* de datos consta de diez atributos, entre los que se encuentra en ángulo de flexión para cada dedo de la mano y la implementación del parámetro *is_extended*, igualmente para cada dedo de la mano.

El proceso de extracción de parámetros se llevó a cabo para todos los símbolos estáticos del alfabeto dactilológico español, siendo en total dieciocho. En la generación de los mismos se tuvo en cuenta la forma correcta de la posición de las manos para el gesto asociado a cada símbolo, por lo que se fueron comprobando los *sets* de datos conforme se iban generando, con el fin de cerciorase de que las instancias recogidas no tuvieran ningún error en sus parámetros. En caso de detectar alguna instancia considerada como errónea se procedía a descartar la misma y generar una nueva instancia. En la Tabla 5 se muestran los símbolos estáticos del alfabeto dactilológico de la lengua de signos española, así como en correlación con las etiquetas empleadas para describir los símbolos del abecedario. Dichas etiquetas deben seguir un orden ascendente atendiendo a los requisitos de la librería *LIBSVM*.

Etiqueta	Símbolo	Etiqueta	Símbolo
0	A	***9***	M
1	B	***10***	N
2	C	***11***	O
3	D	***12***	P
4	E	***13***	Q
5	F	***14***	R
6	I	***15***	S
7	K	***16***	T
8	L	***17***	U

Tabla 5.: Símbolos estáticos

Una vez extraídos los parámetros, se llevó a cabo la preparación de los mismos, para posteriormente llevar a cabo el proceso de clasificación de los *data set*. Se deben realizar una serie de pasos previos a la clasificación en sí. En primer lugar, es necesario realizar la división del *set* de datos original, a un *set* de datos de entrenamiento y otro de prueba. En general esta división se implementa a partir de algún tipo de algoritmo como *k-fold cross-validation*, aunque también está ampliamente aceptado utilizar el 70% de las muestras para el entrenamiento del clasificador y el 30% restantes para verificar que se clasifican correctamente las instancias. Por ello, se optó por realizar la división en este formato, utilizándose siete de las diez muestras obtenidas para realizar el entrenamiento del algoritmo de clasificación y tres muestras para comprobar dicha clasificación.

Así, en primer lugar es necesario realizar un escalado de los datos para adecuarlos a los requisitos del algoritmo, así como para evitar atributos con mayores rangos numéricos que otros, es decir para mantener su homogeneidad. En general el escalado transforma los datos de las instancias correspondientes en un rango de [-1, +1]. Para llevar a cabo el escalado de los datos fue necesaria la ejecución del archivo *svm-scale* a través de la línea de comandos. Este archivo pre compilado, que se incluye con la librería *LIBSVM*, permite realizar el escalado de datos de forma directa a través de la ejecución de un simple comando. En la Figura 4.14, se muestra la ejecución del comando especifico en un terminal de *Windows*.

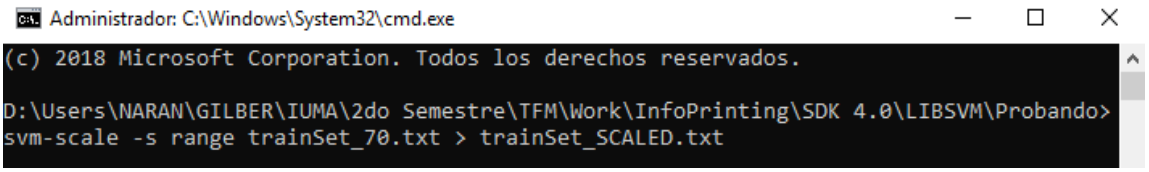
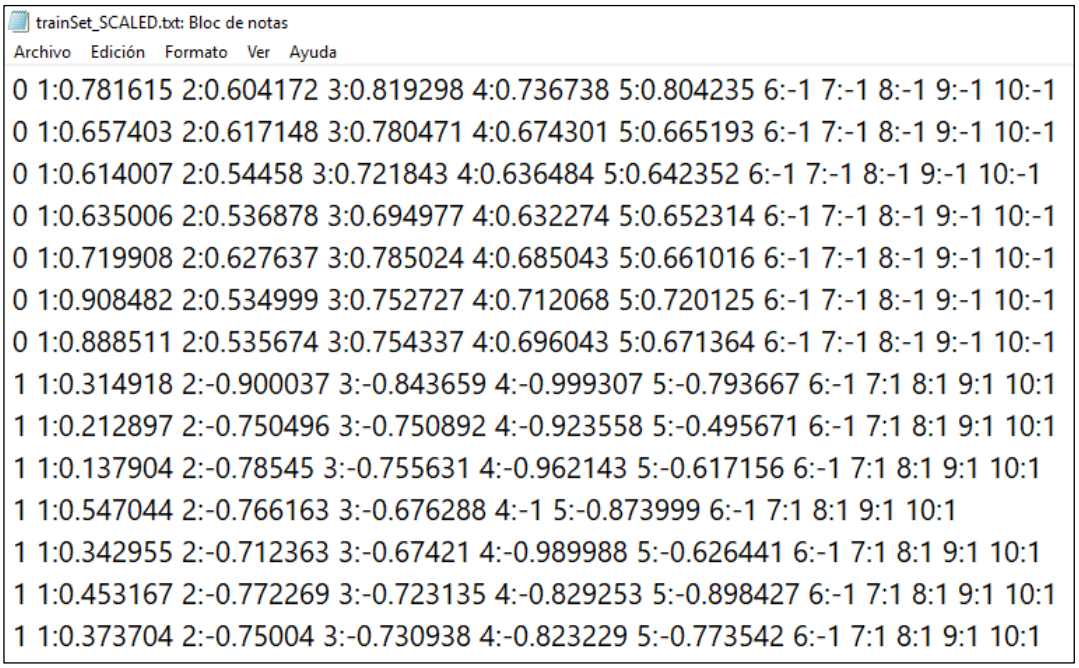


Figura 4.14.: Ejecución svm-scale.exe

A partir de la ejecución anterior, se realiza el escalado de las instancias que se encuentran en un archivo de texto denominado `trainSet_70`, según los parámetros recogidos en el archivo. El archivo de entrada se renombra a `trainSet_SCALED.txt`. A continuación, en la Figura 4.15, se muestra dichos archivos ya escalados. A pesar de que únicamente se muestra el *set* para la letra “A” y “B”, el archivo contiene los parámetros extraídos para cada símbolo estático. Como se observa en la Figura 4.15, el *set* de

entrenamiento consta de siete instancias para cada símbolo del abecedario. Como se ha comentado previamente, la ejecución de esta función genera factores de escalado, que se almacenan en el archivo denominado *range* para posteriormente ser utilizados en el escalado de datos.


trainSet_SCALED.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
0 1:0.781615 2:0.604172 3:0.819298 4:0.736738 5:0.804235 6:-1 7:-1 8:-1 9:-1 10:-1
0 1:0.657403 2:0.617148 3:0.780471 4:0.674301 5:0.665193 6:-1 7:-1 8:-1 9:-1 10:-1
0 1:0.614007 2:0.54458 3:0.721843 4:0.636484 5:0.642352 6:-1 7:-1 8:-1 9:-1 10:-1
0 1:0.635006 2:0.536878 3:0.694977 4:0.632274 5:0.652314 6:-1 7:-1 8:-1 9:-1 10:-1
0 1:0.719908 2:0.627637 3:0.785024 4:0.685043 5:0.661016 6:-1 7:-1 8:-1 9:-1 10:-1
0 1:0.908482 2:0.534999 3:0.752727 4:0.712068 5:0.720125 6:-1 7:-1 8:-1 9:-1 10:-1
0 1:0.888511 2:0.535674 3:0.754337 4:0.696043 5:0.671364 6:-1 7:-1 8:-1 9:-1 10:-1
1 1:0.314918 2:-0.900037 3:-0.843659 4:-0.999307 5:-0.793667 6:-1 7:1 8:1 9:1 10:1
1 1:0.212897 2:-0.750496 3:-0.750892 4:-0.923558 5:-0.495671 6:-1 7:1 8:1 9:1 10:1
1 1:0.137904 2:-0.78545 3:-0.755631 4:-0.962143 5:-0.617156 6:-1 7:1 8:1 9:1 10:1
1 1:0.547044 2:-0.766163 3:-0.676288 4:-1 5:-0.873999 6:-1 7:1 8:1 9:1 10:1
1 1:0.342955 2:-0.712363 3:-0.67421 4:-0.989988 5:-0.626441 6:-1 7:1 8:1 9:1 10:1
1 1:0.453167 2:-0.772269 3:-0.723135 4:-0.829253 5:-0.898427 6:-1 7:1 8:1 9:1 10:1
1 1:0.373704 2:-0.75004 3:-0.730938 4:-0.823229 5:-0.773542 6:-1 7:1 8:1 9:1 10:1
```

Figura 4.15.: trainSet_SCALED.txt

Una vez obtenido el archivo escalado, se procede a la generación del modelo *SVM* a partir de la ejecución del comando *svm-train* a través del terminal de *Windows*, como se observa en la Figura 4.16. Mediante la ejecución de este comando, se obtiene el modelo que será utilizado para realizar la predicción de las instancias contenidas en el *set* de datos de prueba. Los parámetros *Gamma* y de *C* utilizados para la generación del modelo fueron suministrados previamente [39], por lo que en una primera aproximación se emplearan los mismos para generar el modelo.

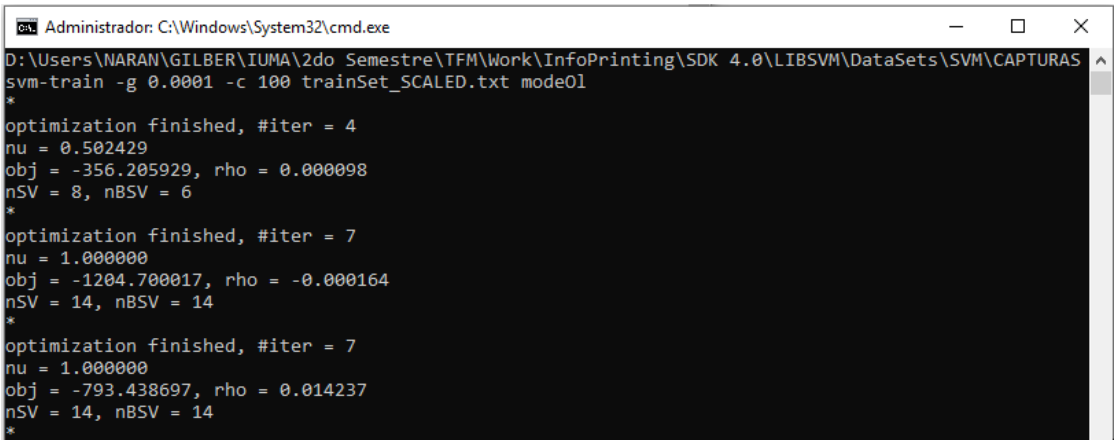


Figura 4.16.: Ejecución de svm-train.exe

Para debe realizar el proceso de escalado tanto para el *set* de datos de entrenamiento como el de prueba. Por ello mediante la utilización del comando *svm-scale* se adaptan las instancias del archivo de texto `testSet_30`. El escalado de este archivo genera un archivo de salida denominado `testSet_SCALED.txt`, que se muestra en la Figura 4.17.

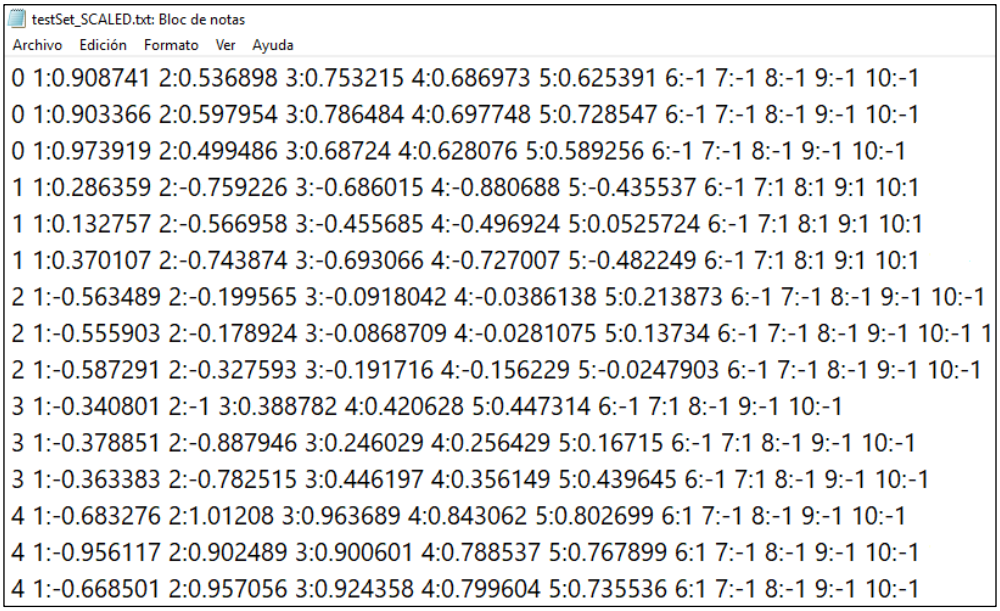


Figura 4.17.: testSet_SCALED.txt

Como se observa en la Figura 4.17, el archivo se compone de tres instancias por cada símbolo del abecedario. Aunque se muestran únicamente los cuatro primeros símbolos del *set* de datos, representado las letras “A”, “B”, “C”, “D” y “E” el archivo contiene los datos escalados para cada símbolo del alfabeto. Así mismo se puede observar como el rango de escalado de los valores se establece en el rango de [-1, +1] como previamente se estableció.

Finalmente, para llevar a cabo el reconocimiento de los símbolos estáticos se debe ejecutar el archivo `svm-predict.exe`. Al ejecutar este archivo, como se muestra en la Figura 4.18, a partir de un *set* de entrada y un modelo, se genera un archivo de salida que contiene la clasificación de las instancias.

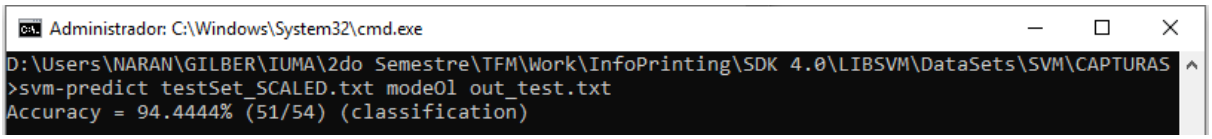


Figura 4.18.: Ejecución del archivo `svm-predict.exe`

Como se observa en la Figura 4.18, la precisión de la clasificación a partir del modelo denominado *modeOl* es de 94.4%. Esta precisión se considera aceptable para haberse dado en el primer intento de clasificación, sin embargo, existen varios métodos para lograr aumentar la precisión de la misma, que se introducirán posteriormente.

Capítulo 5. Plataforma HW/SW final

En este quinto capítulo se recoge la plataforma final desarrollada en la que se engloban todos los componentes *Hardware* y *Software* utilizados para el desarrollo de este TFM, aportando la capacidad de reconocimiento de todos los símbolos del alfabeto dactilológico de la lengua de signos española.

5.1. Selección de datos

Una vez finalizada la plataforma de reconocimiento de símbolos estáticos y para cumplir con los objetivos de desarrollo de este TFM, se pretendió realizar una plataforma de reconocimiento final, en la cual se englobasen tanto los símbolos estáticos como dinámicos del alfabeto dactilológico de la lengua de signos española. Por ello, una vez más se llevó a cabo una selección de datos que en un principio fuesen discriminantes entre los símbolos dinámicos.

Partiendo de los datos seleccionados en el desarrollo de la plataforma *HW/SW* inicial, fue necesario incorporar un nuevo tipo de datos que permitiese la detección de movimiento en la mano. Para ello, se optó inicialmente por añadir la desviación estándar del parámetro *palm.velocity*, que proporciona la tasa de cambio de la posición de la mano en milímetros por segundo. Por otra parte, debido a la gran similitud entre ciertos símbolos donde el movimiento del dedo meñique se considera crítico para la discriminación entre símbolos, tal y como se especificó en el primer capítulo de este *TFM*, fue necesaria la introducción de un parámetro especifico capaz de detectar el movimiento de este dedo. Una vez más, se optó por añadir la desviación estándar del ángulo de flexión del dedo meñique, quedando los parámetros extraíbles como se muestran en la Tabla 6, para una primera aproximación.

Dato	Descripción	Estructura de datos
bones.rotation	Rotación de los huesos de la mano.	LEAP_QUATERNION
Flexión del dedo	Parámetro del nivel de doblez de los dedos.	FLOAT
digits.is_extended	Ángulo de flexión de cada dedo.	LEAP_DIGIT
palm.velocity Standard Deviation	Tasa de cambio de la posición de la mano en milímetros por segundo.	LEAP_VECTOR
Pinky Flex Angle Standard Deviation	Tasa de cambio del ángulo de doblez del dedo meñique.	FLOAT

Tabla 6.: Datos extraíbles para PrintingVariablesDEF.c

Para el cálculo de la desviación estándar de la velocidad de una mano detectada, se partió del parámetro extraíble correspondiente a la velocidad de la mano (*palm.velocity*). Posteriormente se le aplicaron operaciones matemáticas con el fin de poder observar la desviación típica de la velocidad. En caso de que hubiese alguna desviación en los resultados, se podría afirmar que hay movimiento de la mano. Por otro lado, para el cálculo de la deviación estándar del movimiento del dedo meñique, se partió del cálculo del ángulo de doblez de los dedos de la mano. Así la transformación matemática indicada se consigue obtener la desviación estándar, y con ello se pudo comprobar la variación del ángulo y por consiguiente detectar, en caso de que lo hubiera, el movimiento del dedo meñique.

Una vez obtenidos los parámetros que se usarán para el reconocimiento de los símbolos dinámicos del alfabeto dactilologico de la lengua de signos española, fue necesaria la creación de un archivo de código capaz de extraer este tipo de información. Por ello, se realizó una adaptación del archivo `PrintingVariablesFlex.c` expuesto anteriormente para satisfacer esta demanda, denominándose este nuevo archivo como `PrintingVariablesStandard.c` como se muestra en la sección de código representada en la Figura 5.1.

```
//Velocidad de la palma
for(int m=0;m<VectorPos;m++){
  PalmVelocitySum.v[m] = 0;
  for(int n=0;n<SAMPLES;n++){
    PalmVelocitySum.v[m] = PalmVelocitySum.v[m] + PalmVelocityBuffer[m][n];
  }
  PalmVelocityMean.v[m] = PalmVelocitySum.v[m] / (float)SAMPLES;
  float sqDevSum = 0;
  for(int o=0;o<SAMPLES;o++){
    sqDevSum += (float)pow(PalmVelocityMean.v[m] - (PalmVelocityBuffer[m][o]), 2);
  }
  stDev.v[m] = (float)sqrt(sqDevSum /(float)SAMPLES);
  stDev.v[m] = stDev.v[m] / 10;
}
if(stDev.v[0] > 8.9118155 || stDev.v[1] > 6.5256798 || stDev.v[2] > 4.3590943 ){
  fprintf(dataSet, " %i:100.0", iteraciones);
  iteraciones++;
}else{
  fprintf(dataSet, " %i:0.0", iteraciones);
  iteraciones++;
}
```

Figura 5.1.: Cálculo de movimiento de la palma en PrintingVariablesStandard.c

Tal y como se observa en la Figura 5.1, el cálculo de la desviación típica requirió la declaración de un *array* de vectores con el fin de almacenar la velocidad de la mano en los tres ejes de dicho vector [X, Y, Z]. Inicialmente, se acumula la suma de valores en el *array PalmVelocitySum* para posteriormente calcular la media de los valores y almacenarlos en el *array PalmVelocityMean*, cuya declaración es idéntica al *array* anterior. Una vez obtenida la media, se procede a realizar el cálculo de la diferencia entre los valores instantáneos y medios, para finalmente calcular la raíz cuadrada de cada valor y obtener así la desviación típica del movimiento de la mano. Cabe destacar que, debido a que la variable de velocidad de la mano se mide en milímetros/segundos, habrá una desviación típica mínima asociada al movimiento intrínseco que posee la mano al realizar un símbolo. Por ello, se establecen unos valores umbrales para cada eje de coordenadas.

Por otro lado, el cálculo de la desviación estándar del movimiento del dedo meñique se basa en un patrón similar al comentado previamente. Como se puede observar en la Figura 5.2, se mantiene el mismo método de cálculo de la desviación estándar con la ligera diferencia de que el cálculo realizado será únicamente en un solo eje de coordenadas.

```
//Movimiento del meñique
PinkyAngleSum = 0;
for(int n=0;n<SAMPLES;n++){
  PinkyAngleSum = PinkyAngleSum + PinkyAngleBuffer[n];
}
PinkyAngleMean = PinkyAngleSum / (float)SAMPLES;
float PinkysqDevSum = 0;
for(int o=0;o<SAMPLES;o++){
  PinkysqDevSum += (float)pow(PinkyAngleMean - (PinkyAngleBuffer[o]), 2);
}
PinkySTDEV = (float)sqrt(PinkysqDevSum / (float)SAMPLES);/
PinkySTDEV = PinkySTDEV / 10;
if(PinkySTDEV < 1.042565f){
  fprintf(dataSet, " %i:0.0", iteraciones);
  iteraciones++;
}else{
  fprintf(dataSet, " %i:100.0", iteraciones);
  iteraciones++;
}
```

Figura 5.2.: Cálculo de movimiento del dedo meñique en PrintingVariablesStandard.c

Como se comentó previamente, para el cálculo de la desviación típica del ángulo del dedo meñique, únicamente se establece en el eje de coordenadas [0, Y, 0]. Esto se debe principalmente a que, como se mencionó en capítulos anteriores de este *TFM*, el dispositivo *Leap Motion* posee una posición preestablecida en la que se deben colocar las manos, por lo que resultó necesario obtener únicamente el ángulo de doblez de los dedos en dicho eje de coordenadas. En consecuencia, se realiza el cálculo de la desviación típica en dicho eje, estableciendo un único valor umbral para la detección de movimiento del dedo meñique.

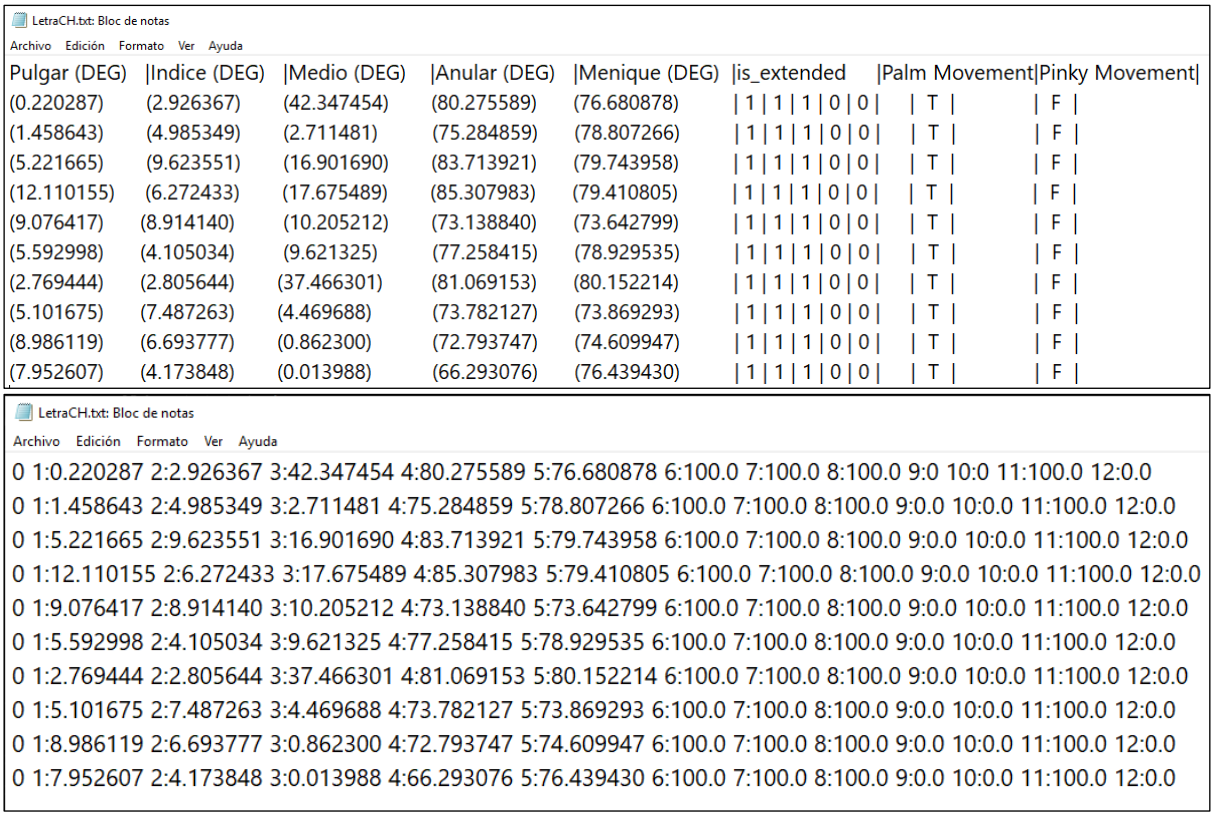
5.2. Reconocimiento de símbolos dinámicos

Una vez desarrollado el código capaz de obtener los parámetros deseados, se continuó con la ejecución del mismo y extracción de los *data set* correspondientes. En el desarrollo de la plataforma final se plantea el reconocimiento de todos los símbolos del alfabeto dactilológico de la lengua de signos española, aunque inicialmente se comprobó la correspondencia de los *set* de datos generados únicamente para los símbolos dinámicos. En la Tabla 7 se muestran los símbolos dinámicos del alfabeto, así como una correlación las etiquetas empleadas para describir los mismos.

Etiqueta	Símbolo	Etiqueta	Símbolo
0	CH	***6***	RR
1	G	***7***	V
2	H	***8***	W
3	J	***9***	Y
4	LL	***10***	Z
5	Ñ		

Tabla 7.: Símbolos dinámicos

A partir de la ejecución del código `PrintingVariablesStandard.c`, se realizó la generación de los *data sets* correspondientes a los símbolos dinámicos del alfabeto dactilológico de la lengua de signos española, tal y como se observa en la Figura 5.3. Al igual que en el desarrollo de la plataforma inicial, se tuvo en cuenta el correcto posicionamiento de las manos para cada símbolo. De esta forma se iban comprobando conforme se iban generando. En caso de detectar alguna instancia considerada como errónea, se procedía a descartar la misma y generar una nueva instancia.


LetraCH.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Pulgar (DEG)	|Indice (DEG)	|Medio (DEG)	|Anular (DEG)	|Menique (DEG)	|is_extended	|Palm Movement|	Pinky Movement|
(0.220287)	(2.926367)	(42.347454)	(80.275589)	(76.680878)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(1.458643)	(4.985349)	(2.711481)	(75.284859)	(78.807266)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(5.221665)	(9.623551)	(16.901690)	(83.713921)	(79.743958)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(12.110155)	(6.272433)	(17.675489)	(85.307983)	(79.410805)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(9.076417)	(8.914140)	(10.205212)	(73.138840)	(73.642799)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(5.592998)	(4.105034)	(9.621325)	(77.258415)	(78.929535)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(2.769444)	(2.805644)	(37.466301)	(81.069153)	(80.152214)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(5.101675)	(7.487263)	(4.469688)	(73.782127)	(73.869293)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(8.986119)	(6.693777)	(0.862300)	(72.793747)	(74.609947)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |
(7.952607)	(4.173848)	(0.013988)	(66.293076)	(76.439430)	| 1 | 1 | 1 | 0 | 0 |	| T |	| F |

LetraCH.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

0 1:0.220287 2:2.926367 3:42.347454 4:80.275589 5:76.680878 6:100.0 7:100.0 8:100.0 9:0 10:0 11:100.0 12:0.0
0 1:1.458643 2:4.985349 3:2.711481 4:75.284859 5:78.807266 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:5.221665 2:9.623551 3:16.901690 4:83.713921 5:79.743958 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:12.110155 2:6.272433 3:17.675489 4:85.307983 5:79.410805 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:9.076417 2:8.914140 3:10.205212 4:73.138840 5:73.642799 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:5.592998 2:4.105034 3:9.621325 4:77.258415 5:78.929535 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:2.769444 2:2.805644 3:37.466301 4:81.069153 5:80.152214 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:5.101675 2:7.487263 3:4.469688 4:73.782127 5:73.869293 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:8.986119 2:6.693777 3:0.862300 4:72.793747 5:74.609947 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0
0 1:7.952607 2:4.173848 3:0.013988 4:66.293076 5:76.439430 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0

Figura 5.3.: Archivos de salida para PrintingVariablesStandard.c

A partir de la ejecución del código especificado anteriormente, se generan los *sets* de datos que se observan en la Figura 5.3. Como se puede observar, se incorporaron los parámetros de movimiento de la mano y de movimiento del dedo meñique, tal y como se especificó anteriormente. Por otro lado, el primer *data set* que se observa corresponde al utilizado para la comprobación de la correspondencia de los valores obtenidos en el proceso de captura, mientras que el segundo *data set* que se observa se corresponde a un archivo con el formato específico para su utilización en el clasificador *SVM*.

Una vez obtenidos los *data set* necesarios para todos los símbolos del alfabeto, se debe seguir el mismo proceso que el descrito en el capítulo anterior. Inicialmente, el *set* de datos generado debe ser escalado, una vez más en un rango comprendido entre [-1, +1]. Para ello, en la Figura 5.4 se muestra la sentencia correspondiente al escalado del archivo de entrenamiento generado, una vez más mediante la utilización del archivo precompilado `svm-scale.exe`.

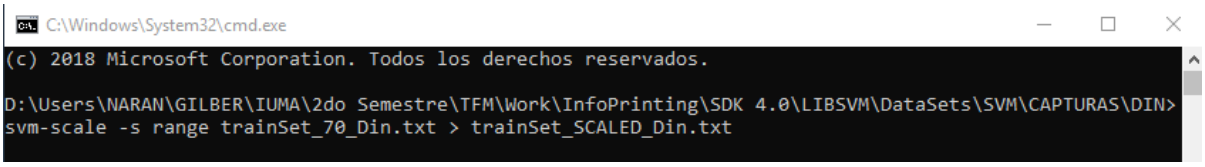


Figura 5.4.: Ejecución de `svm-scale.exe` para símbolos dinámicos

Mediante la ejecución de esta sentencia se escalan los valores contenidos en el archivo `trainSet_70_Din.txt` y se almacenan en el archivo denominado `trainSet_SCALED_Din.txt`, atendiendo a las especificaciones previamente expuestas. En la Figura 5.5 se muestra dicho archivo ya escalado.

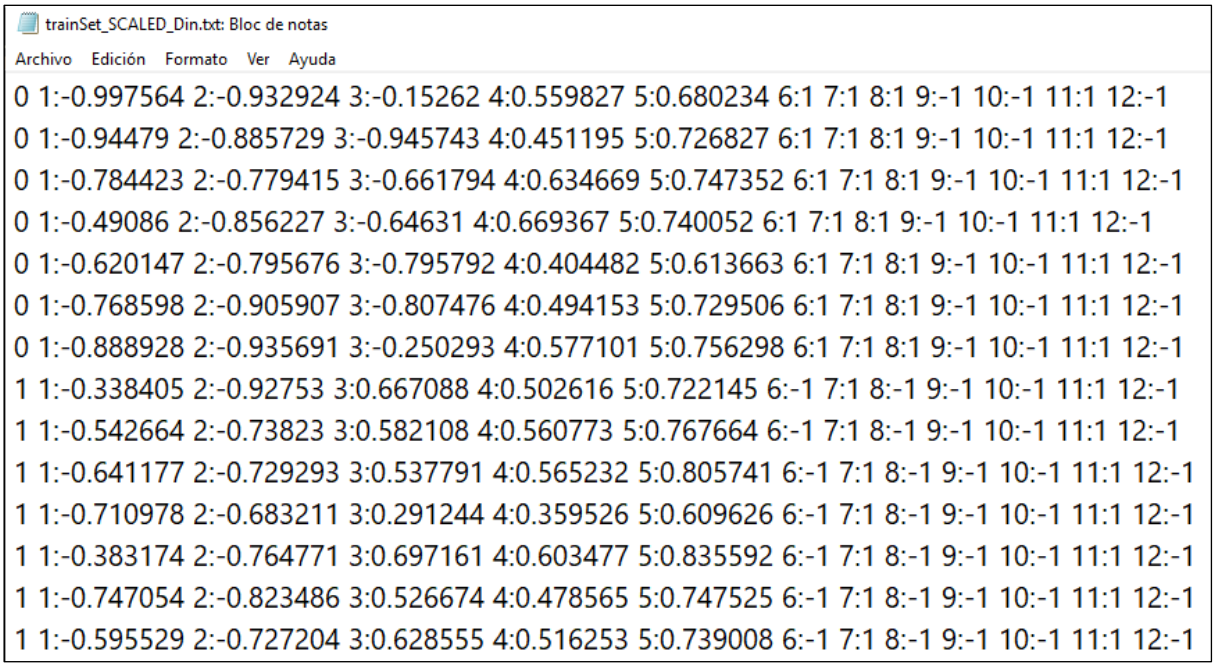


Figura 5.5.: trainSet_SCALED_Din.txt

Como se observa en la Figura 5.5, el *data set* ya escalado se compone de 7 instancias para cada símbolo dinámico de la lengua, a pesar de que únicamente se visualicen los correspondientes a los símbolos “CH” y “G”. Una vez obtenido el *set* de entrenamiento escalado, se procede con la generación del modelo SVM que realizará la clasificación de las muestras. Así en la Figura 5.6 se muestra la ejecución de la sentencia correspondiente al archivo `svm-train.exe`.

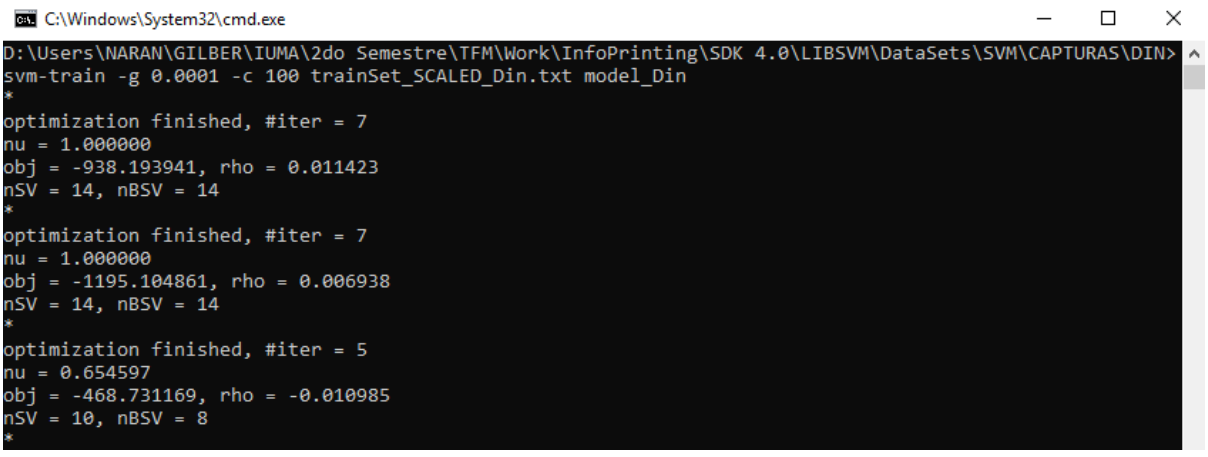


Figura 5.6.: Ejecución de svm-train.exe para símbolos dinámicos

En *LIBSVM* es necesario que todos los *sets* de datos que se vayan a utilizar en el clasificador sean escalados previamente, por ello se realiza el escalado del archivo `testSet_30_Din.txt` y se almacena en el archivo `testSet_SCALED_Din.txt`, como se puede observar en la Figura 5.7 que se muestra a continuación.

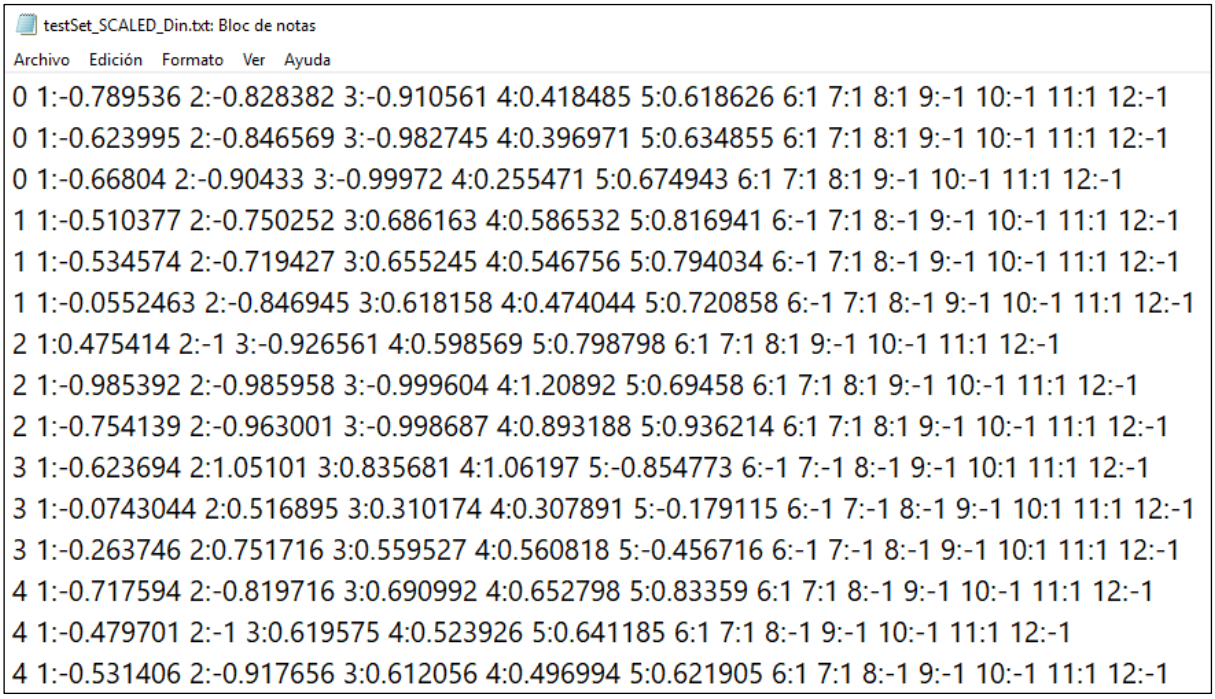


Figura 5.7.: testSet_SCALED_Din.txt

Como se observa en la Figura 5.7, el *data set* se compone de 3 instancias para cada símbolo, aunque una vez más, únicamente se observan las instancias correspondientes a los símbolos “CH”, “G”, “H”, “J”, si bien este archivo contiene las instancias para todos los símbolos dinámicos. Con el *set* de prueba ya escalado se procede a la predicción de los símbolos, por ello se hace uso del archivo `svm-predict.exe`, que con la sentencia adecuada genera el valor de *accuracy* correspondiente a la clasificación, como se observa en la Figura 5.8.

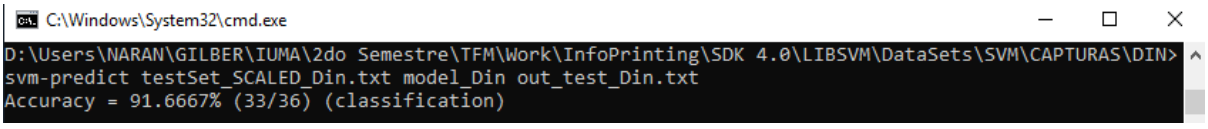


Figura 5.8.: Ejecución de svm-predict.exe para símbolos dinámicos

La clasificación de las instancias da un resultado de *accuracy* del 91,6%, como se observa en la Figura 5.8. Fallando principalmente en la clasificación del símbolo correspondiente a la letra "W", la cual agrupo incorrectamente como el símbolo correspondiente a la letra "V". Este resultado es inferior al esperado, por lo que se planteó la modificación de introducir un parámetro adicional para discriminar con mayor eficacia entre los símbolos dinámicos.

Para el alfabeto dactilológico de la lengua de signos española existe una clara diferencia entre los símbolos donde la palma de la mano se encuentra enfocada hacía el dispositivo *Leap Motion*, o símbolos en los cuales la palma de la mano se encuentra invertida. Por esto y por lo expuesto anteriormente, se optó por la introducción del parámetro *palm.normal*. Este tipo de datos proporciona el eje normal a la palma de la mano. En la Tabla 8 se muestra un resumen de los parámetros finales incorporados al desarrollo de la plataforma desarrollada.

Dato	Descripción	Estructura de datos
bones.rotation	Rotación de los huesos de la mano.	LEAP_QUATERNION
Flexión del dedo	Parámetro del nivel de doblez de los dedos.	FLOAT
digits.is_extended	Ángulo de flexión de cada dedo.	LEAP_DIGIT
palm.velocity Standard Deviation	Tasa de cambio de la posición de la mano en milímetros por segundo.	LEAP_VECTOR
Pinky Flex Angle Standard Deviation	Tasa de cambio del ángulo de doblez del dedo meñique.	FLOAT
palm.normal	Eje normal a la palma de la mano	LEAP_VECTOR

Tabla 8.: Datos extraídos para la plataforma final

Una vez más, como se observa en la Tabla 8, el parámetro finalmente introducido hace referencia al eje normal de la palma de la mano en forma de vector. Tal y como se comentó previamente, únicamente sería necesario extraer la componente correspondiente al eje [0, Y, 0] ya que se busca discriminar entre símbolos donde la palma de la mano se

encuentre apuntando hacia la superficie del dispositivo *Leap Motion*, o símbolos donde la palma se encuentre invertida. En la Figura 5.9 se muestra el código correspondiente al a introducción del parámetro mencionado.

```
//PALM Normal
normal = hand->palm.normal;
if(normal.v[1] < 0 ){
  fprintf(dataSet, " %i:0.0", iteraciones);
  iteraciones++;
}else{
  fprintf(dataSet, " %i:100.0", iteraciones);
  iteraciones++;
}
```

Figura 5.9.: Implementación de palm.normal en PrintingVariablesStandard.c

En la Figura 5.9 se observa cómo se almacena en la variable *normal* el vector asociado a el eje normal de la palma de la mano. Posteriormente mediante la utilización de el condicionante *if*, se decide si la orientación del eje Y es inferior o superior a cero. En caso de que el valor recogido fuese inferior a cero, indicaría que el eje normal se encuentra apuntando hacia el controlador, mientras que en caso contrario indicaría que la palma de la mano se encuentra invertida con respecto al controlador.

Con la introducción de este nuevo parámetro fue necesaria la comprobación del resultado de la clasificación, por lo que al igual que en validaciones anteriores, se siguieron los pasos descritos por el desarrollador de la librería *LIBSVM*. Inicialmente se realiza el escalado del *set* de entrenamiento generado por el código `PrintingVariablesStandard.c`, a continuación se genera un modelo a partir de la utilización del archivo `svm-train.exe`, posteriormente se realiza el escalado del *test set* para su adecuar el rango de valores a los especificados por *SVM*, y finalmente se realiza la predicción de dicho *set* de testeo mediante la utilización del archivo `svm-predict.exe`, obteniendo en este caso un resultado de *accuracy* como el que se observa en la Figura 5.10.

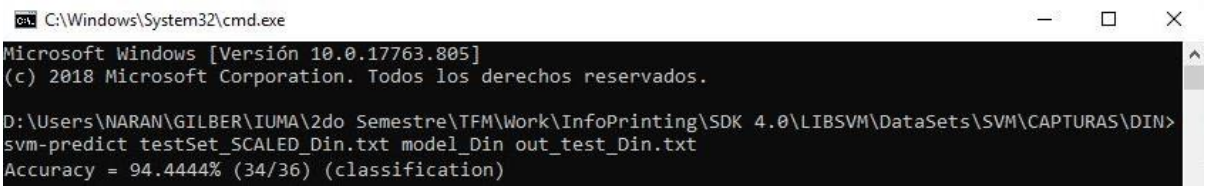


Figura 5.10.: Ejecución de svm-predict.exe para palm.normal

Como se puede observar en la Figura 5.10, con la introducción del nuevo parámetro se mejora ligeramente el valor correspondiente al *accuracy* de la clasificación, demostrando que la introducción del nuevo parámetro mejora la discriminación entre los símbolos de la lengua de signos. En este caso se agrupo erróneamente los símbolos correspondientes a la letra "W". Esta precisión se considera aceptable para haberse dado en el primer intento de clasificación de símbolos dinámicos, sin embargo, posteriormente se introducirán los métodos para mejorar dicho nivel.

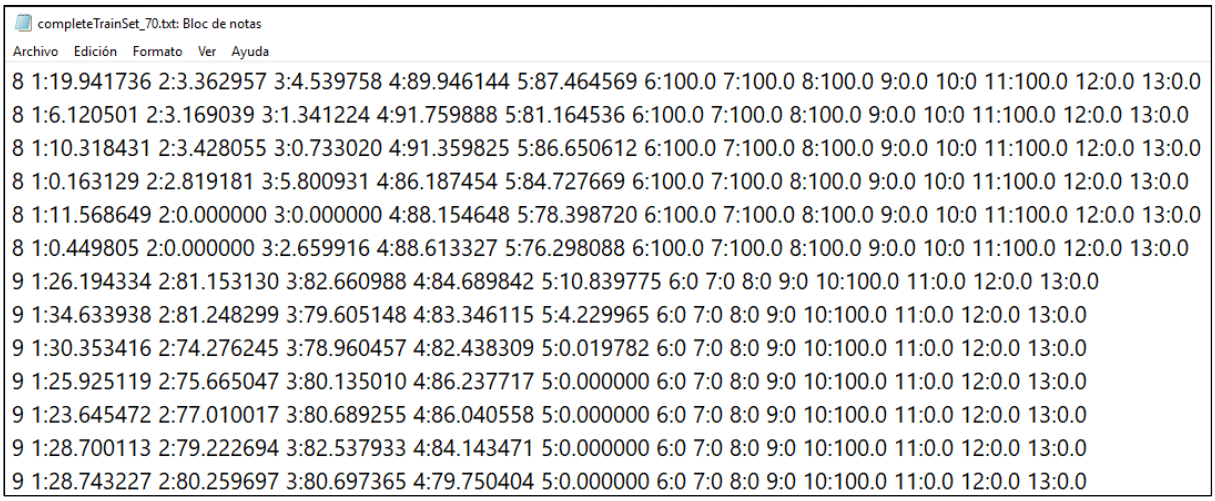
5.3. Reconocimiento de símbolos del lenguaje

Para satisfacer el objetivo global de este *TFM*, se continuo con el desarrollo de la plataforma que integrase, tanto el reconocimiento de símbolos dinámicos como estáticos, por lo cual se llevó a cabo una unificación de *data sets* con el fin de obtener un banco de datos superior con el que entrenar al clasificador. En la Tabla 9 se muestran los símbolos, tanto estáticos como dinámicos del alfabeto, así como una correlación de las etiquetas empleadas para describir cada símbolo de la lengua de signos española. Dichas etiquetas deben seguir un orden ascendente atendiendo a los requisitos de la librería *LIBSVM*.

Etiqueta	Símbolo	Etiqueta	Símbolo	Etiqueta	Símbolo
0	A	***10***	J*	***20***	R
1	B	***11***	K	***21***	RR*
2	C	***12***	L	***22***	S
3	CH*	***13***	LL*	***23***	T
4	D	***14***	M	***24***	U
5	E	***15***	N	***25***	V*
6	F	***16***	Ñ*	***26***	W*
7	G*	***17***	O	***27***	X*
8	H*	***18***	P	***28***	Y*
9	I	***19***	Q	**29**	Z*
* Símbolos dinámicos.					

Tabla 9.: Relación etiqueta símbolo del alfabeto dactilológico español.

Con la unificación del *set* de datos, también se pretende verificar la correcta clasificación de instancias con el sistema pertinente. Mediante la realización de las pruebas correspondientes, se obtiene el valor del *accuracy* del sistema completo. En la Figura 5.11 se muestra el *set* de datos correspondiente al entrenamiento del clasificador.

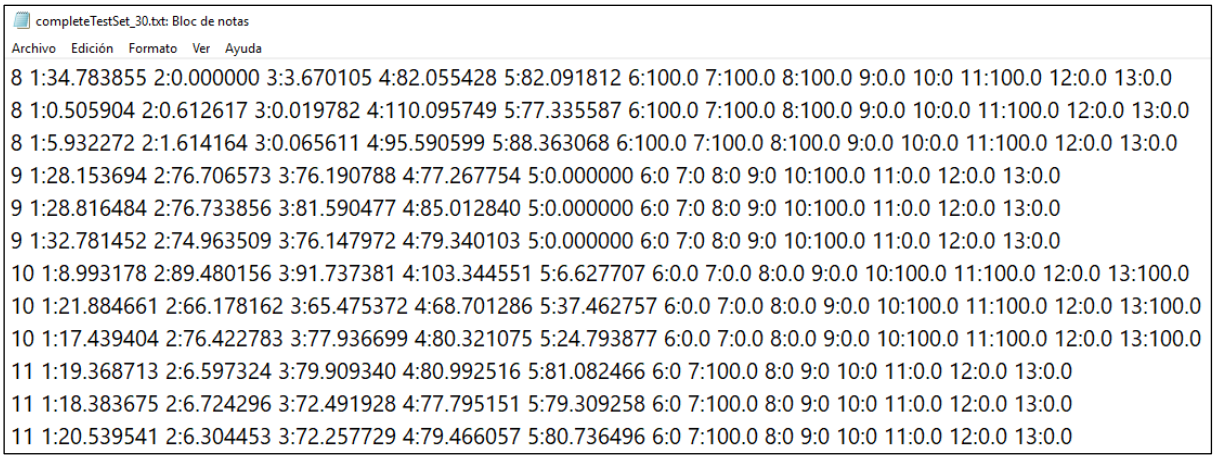

completeTrainSet_70.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
8 1:19.941736 2:3.362957 3:4.539758 4:89.946144 5:87.464569 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
8 1:6.120501 2:3.169039 3:1.341224 4:91.759888 5:81.164536 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
8 1:10.318431 2:3.428055 3:0.733020 4:91.359825 5:86.650612 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
8 1:0.163129 2:2.819181 3:5.800931 4:86.187454 5:84.727669 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
8 1:11.568649 2:0.000000 3:0.000000 4:88.154648 5:78.398720 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
8 1:0.449805 2:0.000000 3:2.659916 4:88.613327 5:76.298088 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
9 1:26.194334 2:81.153130 3:82.660988 4:84.689842 5:10.839775 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:34.633938 2:81.248299 3:79.605148 4:83.346115 5:4.229965 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:30.353416 2:74.276245 3:78.960457 4:82.438309 5:0.019782 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:25.925119 2:75.665047 3:80.135010 4:86.237717 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:23.645472 2:77.010017 3:80.689255 4:86.040558 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:28.700113 2:79.222694 3:82.537933 4:84.143471 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:28.743227 2:80.259697 3:80.697365 4:79.750404 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
```

Figura 5.11.: CompleteTrainSet_70.txt

Como se puede observar en la Figura 5.11, el *set* de datos de entrenamiento se compone de 7 instancias y 13 atributos que describen cada símbolo, incluyendo el parámetro asociado al vector normal a la palma de la mano. En el extracto del *set* de datos completo se observan las instancias correspondientes a las letras “H” y “J”. Cabe destacar que el símbolo correspondiente a la letra “H” se ejecuta de forma dinámica mientras que el símbolo correspondiente a la letra “J” se ejecuta de forma estática, lo cual se observa fácilmente analizando el atributo número 11 que hace referencia al parámetro de la desviación estándar del movimiento de la mano. Para el caso del movimiento dinámico se establece este valor a 100.0, mientras para el caso de movimiento estático el mismo valor corresponde a 0.0. Por otro lado, en la Figura 5.12 se observa el *set* de datos que corresponde al archivo de prueba, que se muestra a continuación.


completeTestSet_30.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

8 1:34.783855 2:0.000000 3:3.670105 4:82.055428 5:82.091812 6:100.0 7:100.0 8:100.0 9:0.0 10:0 11:100.0 12:0.0 13:0.0
8 1:0.505904 2:0.612617 3:0.019782 4:110.095749 5:77.335587 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0 13:0.0
8 1:5.932272 2:1.614164 3:0.065611 4:95.590599 5:88.363068 6:100.0 7:100.0 8:100.0 9:0.0 10:0.0 11:100.0 12:0.0 13:0.0
9 1:28.153694 2:76.706573 3:76.190788 4:77.267754 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:28.816484 2:76.733856 3:81.590477 4:85.012840 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
9 1:32.781452 2:74.963509 3:76.147972 4:79.340103 5:0.000000 6:0 7:0 8:0 9:0 10:100.0 11:0.0 12:0.0 13:0.0
10 1:8.993178 2:89.480156 3:91.737381 4:103.344551 5:6.627707 6:0.0 7:0.0 8:0.0 9:0.0 10:100.0 11:100.0 12:0.0 13:100.0
10 1:21.884661 2:66.178162 3:65.475372 4:68.701286 5:37.462757 6:0.0 7:0.0 8:0.0 9:0.0 10:100.0 11:100.0 12:0.0 13:100.0
10 1:17.439404 2:76.422783 3:77.936699 4:80.321075 5:24.793877 6:0.0 7:0.0 8:0.0 9:0.0 10:100.0 11:100.0 12:0.0 13:100.0
11 1:19.368713 2:6.597324 3:79.909340 4:80.992516 5:81.082466 6:0 7:100.0 8:0 9:0 10:0 11:0.0 12:0.0 13:0.0
11 1:18.383675 2:6.724296 3:72.491928 4:77.795151 5:79.309258 6:0 7:100.0 8:0 9:0 10:0 11:0.0 12:0.0 13:0.0
11 1:20.539541 2:6.304453 3:72.257729 4:79.466057 5:80.736496 6:0 7:100.0 8:0 9:0 10:0 11:0.0 12:0.0 13:0.0

Figura 5.12.: CompleteTestSet_30.txt

El *data set* de prueba se compone de 3 instancias para cada símbolo del lenguaje, tal y como se observa en la Figura 5.12. Una vez más, las instancias recogidas en este *set* de datos se corresponden a las letras “H”, “I”, “J” y “K”, que observando el atributo número 11 del *set* se establece que las letras “H” y “J” se ejecutan de forma dinámica, mientras que las letras “I” y “K” se ejecutan de forma estática. Ampliando más se puede observar que según el atributo número 13, correspondiente con la normal a la palma de la mano, se encuentra a un valor lógico alto para la letra “J”, evidenciando que dicho símbolo se ejecuta con la palma de la mano invertida con respecto al dispositivo *Leap Motion*.

Una vez obtenidos los *sets* de datos completos, se procede a realizar el proceso establecido por la librería *LIBSVM*. En primer lugar, escalar los valores de el *data set* de entrenamiento, para así poder realizar la generación del modelo correspondiente al archivo precompilado `svm-train.exe`, y en segundo lugar escalar el archivo `CompleteTestSet_30.txt` para así poder realizar la clasificación de instancias y obtener un valor para el parámetro *accuracy*. En la Figura 5.13 se observa la ejecución del archivo `svm-predict.exe` para el s*et* de datos completo.

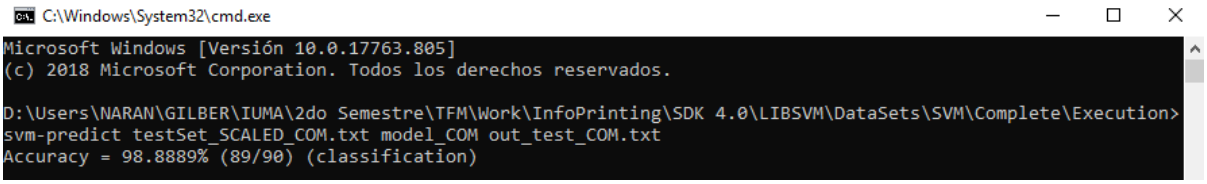


Figura 5.13.: Ejecución svm-predict.exe con model_COM

Como se observa en la Figura 5.13, utilizando el modelo denominado *model_COM*, el cual hace referencia al modelo generado con el *set* de datos completo, se alcanza un valor de *accuracy* del 98.8%, habiendo clasificado correctamente 89 de las 90 instancias que componen el *set* de datos de prueba.

Sin embargo, este resultado es ligeramente inferior al esperado ya que, para cumplimentar el objetivo general de este *TFM*, se requiere de un sistema de clasificación que aporte un valor de *accuracy* del 100%, haciendo referencia a una correcta clasificación de todas las instancias. Por ello se procede a variar los parámetros de ajuste que posee este tipo de clasificador. Como se expuso en capítulos anteriores, estos son los parámetros *C* y *gamma*. Para llevar a cabo este tipo de ajuste es necesario volver a generar el modelo, ya que dichos parámetros se especifican con la generación del mismo. En la Figura 5.15 se muestra la ejecución correspondiente a `svm-train.exe`.

Figura 5.14.: Ejecución svm-train.exe con C y gamma modificados

Con la ejecución de la sentencia que se observa en la Figura 5.14, se genera un modelo nuevo denominado *model_COMP*. Una vez obtenido dicho modelo se procede con la comprobación del nivel de *accuracy*, mediante la ejecución del archivo `svm-predict.exe`, como se observa en la Figura 5.15, que se muestra a continuación.

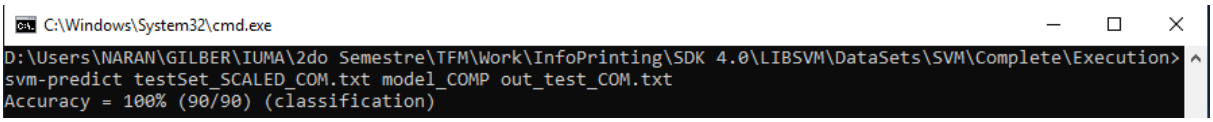


Figura 5.15.: Ejecución svm-predict.exe con model_COMP

Como se observa en la Figura 5.15, con el ajuste de los parámetros del clasificador se modifica el nivel de *accuracy*. En este caso el valor obtenido se corresponde con una clasificación completa de todos los símbolos del alfabeto de la lengua de signos española, por lo que se procede a establecer los nuevos valores de *C* y *gamma* para el *SVM*. En la Tabla 10 se recogen los nuevos valores de los parámetros de ajuste.

C	$Gamma\ (\gamma)$
90	0.0005

Tabla 10.: C y Gamma

5.4. Integración del clasificador SVM

La generación de un modelo que sea capaz de realizar una correcta clasificación de todos los símbolos del alfabeto dactilológico de la lengua de signos española da paso a la integración del mismo clasificador en la plataforma que se ha desarrollado a lo largo del *TFM*. Dicha integración pretende proveer a la plataforma de reconocimiento de la capacidad de clasificar símbolos de forma *online* y así actuar de traductor.

Para la realización de la plataforma de reconocimiento *online* se debía emular el proceso descrito por el desarrollador de la librería *LIBSVM*, por lo tanto, se optó por la integración de las diferentes funcionalidades que componen el proceso de clasificación. En primer lugar, se comenzó con la integración del código correspondiente al escalado de valores, tal y como se muestra en la Figura 5.16.

```
void scaleValues(float buffer[INSTANCE][NODE][INDEX]){
  if(alreadyScaled){
    printf("Ya se ha realizado el es escalado de datos.\n");
  }else{
    alreadyScaled = TRUE;
    for(int i = 0; i < INDEX; i++){
      scaledNodeBuffer[0][0][i] = buffer[0][0][i];
      scaledNodeBuffer[0][1][i] = scaleOutput((int)buffer[0][0][i],buffer[0][1][i]);
      printf("|   COMPLETADO: ");
      printf("\033[1;34m");
      printf("%i%%.\n", (int)(8.333333333333333333 * i));
      printf("\033[0m");
    }
  }
}
```

Figura 5.16.: Función scaleValues

Para realizar el escalado de los valores obtenidos se hace uso de la función *scaleValues*. Esta función recibe como parámetros el *array* que hace las veces de *buffer* para el almacenamiento de los datos recogidos a partir del dispositivo *Leap Motion*, en el que se almacenan, tanto los índices que corresponden a cada atributo que definen los símbolos, como el valor de dicho atributo. Principalmente, la función recorre dicho *buffer* de datos mediante la utilización de un *for* hasta *INDEX* que representa un numero finito correspondiente al número de atributos que se utilizan. A su vez, en cada iteración se hace uso de una función auxiliar para realizar de forma explícita el escalado de valores, denominada *scaleOutput*, como se muestra en la Figura 5.17.

```
float scaleOutput(int index, float value){
  float answer;
  if(value == range[index][0]){
    answer = LOWER;
  }else if(value == range[index][1]){
    answer = UPPER;
  }else{
    answer = LOWER + (UPPER - LOWER) * (value - range[index][0]) /
    (range[index][1] - range[index][0]);
  }
  return answer;
}
```

Figura 5.17.: Función scaleOutput

La función recibe como parámetro de entrada el índice correspondiente al atributo que se pretende escalar, así como el valor de dicho atributo. A partir de esto, se realiza el escalado de los valores según lo establecido en el *array range*. El mismo se compone de los márgenes máximos y mínimos entre los que se debe escalar cada valor correspondiente a cada atributo. Las variables *UPPER* y *LOWER* se definen como el margen superior (*UPPER*) e inferior (*LOWER*) que debe alcanzar el valor que será escalado. Para este caso particular los valores se corresponden con [-1, +1]. Principalmente la función compara el valor que se le pasa como parámetro con el valor almacenado en *range* para, en caso de corresponderse con el valor mínimo, devolver *LOWER* como valor escalado, así para el valor máximo. En caso de que el valor se encuentre entre ese margen, se procede a escalar el valor según la función matemática (9):

$$x = (b - a)\frac{x - \min(x)}{\max(x) - \min(x)} + a \qquad (9)$$

Donde las variables se corresponden con lo mostrado en la Tabla 11, que se expone a continuación.

Variable	***Variable***
$\boldsymbol{x}$**: answer**	$\min(x)$: **range[index][0]**
$\boldsymbol{b}$**: *LOWER***	$\max(x)$: **range[index][1]**
$\boldsymbol{a}$**: *UPPER***	

Tabla 11.: Correspondencia scaleOutput

Debido a que se pretendía desarrollar una plataforma de reconocimiento que utilizara el mismo modelo previamente generado para llevar a cabo la clasificación de símbolos, se consideró realizar la incorporación de una opción con la que se permitiese cargar otro modelo diferente al sistema. Esta función se denominó *loadModel*, y se introduce como se muestra en la Figura 5.18.

```
void loadModel(char* modelname){
  if(strcmp(modelname, previousmodelName) == 0 && !model){
    model = svm_load_model(modelname);
    loadmodel = FALSE;
  }else if(strcmp(modelname, previousmodelName) == 0 && model){
    printf("El modelo ya ha sido previamente cargado...\n");
  }else{
    if(!svm_load_model(modelname)){
      printf("Modelo no cargado, compruebe el nombre... \n");
    }else{
      if(model){
        svm_free_model_content(model);
      }
      model = svm_load_model(modelname);
      loadmodel = FALSE;
      printf("Modelo cargado satisfactoriamente... \n");
      *previousmodelName = modelname;
    }
  }
}
```

Figura 5.18.: Función loadModel

La función *loadModel* recibe por parámetros el nombre del modelo que se pretende cargar. Debido a que en la implementación final del sistema se parte de un modelo precargado, inicialmente se realiza una comprobación entre el nombre del modelo que se ha cargado previamente (*previousmodelName*) con el nombre del modelo que se pretende cargar (*modelName*). Principalmente, la función se encarga de comprobar si ya se ha cargado previamente un modelo, si ha habido algún error a la hora de cargar el modelo, o de cargar un modelo nuevo almacenándolo en la variable *model*.

Una vez realizado el escalado de los valores que se pretenden clasificar, fue necesaria la integración de una función con capacidad de realizar el reconocimiento de símbolos. En la Figura 5.19 se muestra el código correspondiente a la función *predict* desarrollada.

```
void predict(struct svm_model *model, float buffer[INSTANCE][NODE][INDEX]){
  if(!model){
    printf("No hay ning%cn modelo cargado...\n",(char)163);
    return;
  }
  node = (struct svm_node *) realloc(node, 13*sizeof(struct svm_node));
  int i;
  for(i = 0; i<INDEX; i++){
    printf("|   COMPLETADO: ");
    printf("\033[1;34m");
    printf("%i%%.\n", (int)(8.333333333333333333 * i));
    printf("\033[0m");
    node[i].index = (int) buffer[0][0][i];
    node[i].value = (double) buffer[0][1][i];
  }
  node[i].index = -1;
  predictedLabel = svm_predict(model, node);
  recognizedLetter((int)predictedLabel);
}
```

Figura 5.19.: Función predict

Para la realización de la predicción de los símbolos, fue necesario que la función recibiese por parámetros el *buffer* en el cual se encuentran almacenados los valores que previamente han sido escalados, así como el modelo que previamente ha sido cargado en el sistema. Inicialmente la función realiza una comprobación de si efectivamente se encuentra cargado un modelo con el que poder realizar una predicción. La función recorre mediante un bucle *for* similar al utilizado en las funciones anteriores, el *buffer* de entrada para, en cada iteración, transferir el valor de las variables a la estructura de datos correspondiente, que en este caso se corresponde con *svm_node*. Según especificaciones del desarrollador de la librería, cada instancia que representa a un símbolo debe finalizar en índice -1. La predicción se almacena en la variable *predictedLabel*, que posteriormente se pasará como parámetro a una función auxiliar denominada *recognizedLetter*. Esta función auxiliar se muestra en la Figura 5.20.

```
void recognizedLetter(int class){
  if(class == 3 ){//CH
    printf( "|--------------------------------------|\n"
      "|*  Se ha reconocido la letra: " );
    printf("\033[1;32m");
    printf("  %c%c", (char)67 ,(char)72);
    printf("\033[0m");
    printf( "      |\n"
      "|--------------------------------------|\n" );
  }else if(class == 13){//LL
    printf( "|--------------------------------------|\n"
      "|*  Se ha reconocido la letra: " );
    printf("\033[1;32m");
    printf("  %c%c", (char)76 ,(char)76);
    printf("\033[0m");
    printf( "      |\n"
      "|--------------------------------------|\n" );
  }else if(class == 16){//Ñ
    printf( "|--------------------------------------|\n"
      "|*  Se ha reconocido la letra: " );
    printf("\033[1;32m");
    printf("   %c", (char)21);
    printf("\033[0m");
    printf( "     |\n"
      "|--------------------------------------|\n" );
  }else if(class == 21){//RR
    printf( "|--------------------------------------|\n"
      "|*  Se ha reconocido la letra: " );
    printf("\033[1;32m");
    printf("  %c%c", (char)82 ,(char)82);
    printf("\033[0m");
    printf( "      |\n"
      "|--------------------------------------|\n" );
  }else{
    printf( "|--------------------------------------|\n"
      "|*  Se ha reconocido la letra: " );
    printf("\033[1;32m");
    printf("   %c", labels[class]);
    printf("\033[0m");
    printf( "     |\n"
      "|--------------------------------------|\n" );
  }
}
```

Figura 5.20.: Función recognizedLetter

Como se comentó previamente, la función *recognizedLetter* recibe por parámetro el valor numérico correspondiente a un símbolo específico. El objetivo principal detrás del desarrollo de esta función es tratar aquellos símbolos que no pueden ser reconocidos directamente, ya sea por que contienen caracteres especiales, o porque se componen de 2 letras. De esta forma, dependiendo del valor de la variable *class* se procede a imprimir por pantalla la letra correspondiente a dicho valor numérico. Para ello se hace uso de un *array* denominado *labels* que contiene todas las letras del del alfabeto dactilológico de la lengua de signos española.

5.5. Plataforma Final

La implantación de la plataforma final se llevó a cabo a partir de lo desarrollado previamente. A partir de la incorporación del clasificador en el desarrollo de la plataforma fue necesaria la introducción de algún método de control para la ejecución del programa. Por ello, se realizó de un menú, así como la implementación de algunas funciones de apoyo para el correcto funcionamiento del sistema.

En la Figura 5.21 se muestra el código correspondiente a la primera parte de la función *captureSamples*, que se desarrolló para englobar el proceso de captura de datos en una única función. A pesar de que el desarrollo de esta función se basa en el mismo principio de captura de datos que ya se utilizó previamente, se realizaron las modificaciones correspondientes para la adaptación de la plataforma final.

```
void captureSamples(void){
  alreadyScaled = FALSE;
  int msec = 0;
  clock_t timeBefore = clock();
  for (int WritePos = 0; WritePos < SAMPLES; WritePos++){
    frame = GetFrame();
    if (frame && (frame->tracking_frame_id > lastFrameID)){
      lastFrameID = frame->tracking_frame_id;
      for (uint32_t h = 0; h < frame->nHands; h++){
        hand = &frame->pHands[h];
        quat.v[0] = hand->digits[4].bones[1].rotation.x;
        quat.v[1] = hand->digits[4].bones[1].rotation.y;
        quat.v[2] = hand->digits[4].bones[1].rotation.z;
        quat.v[3] = hand->digits[4].bones[1].rotation.w;
        firstVector = quat2basis(quat);
        quat1.v[0] = hand->digits[4].bones[2].rotation.x;
        quat1.v[1] = hand->digits[4].bones[2].rotation.y;
        quat1.v[2] = hand->digits[4].bones[2].rotation.z;
        quat1.v[3] = hand->digits[4].bones[2].rotation.w;
        secondVector = quat2basis(quat1);
        angleY = (float)(AngleTo(firstVector, secondVector) * RAD_TO_DEG);
        PinkyAngleBuffer[WritePos] = angleY;
        Velocity.v[0] = hand->palm.velocity.v[0];
        Velocity.v[1] = hand->palm.velocity.v[1];
        Velocity.v[2] = hand->palm.velocity.v[2];
        printf("|   COMPLETADO: ");
        printf("\033[1;34m");
        printf("%i%%.\n", (int)(3.45 * WritePos));
        printf("\033[0m");
        for (int i = 0; i < VectorPos; i++){
          PalmVelocityBuffer[i][WritePos] = Velocity.v[i];
          quat.v[i] = 0.0f;
          quat1.v[i] = 0.0f;
          Velocity.v[i] = 0.0f;
        }
        quat.v[3] = 0.0f;
        quat1.v[3] = 0.0f;
        angleY = 0;
      }
    }
    millisleep(100);
  }
```

Figura 5.21.: Función captureSamples, parte 1

La función *captureSamples* no recibe ningún parámetro de entrada, ya que únicamente se utiliza para la captura de datos; así mismo, no devuelve ninguna variable. En esta primera parte de la función se recogen y almacenan los datos correspondientes para el cálculo de la desviación estándar, ya que para el mismo es necesario que se capturen varias muestras. En la Figura 5.21 se muestra cómo se invoca a la función *GetFrame* y se almacena su resultado en la variable *frame*. De esta forma se captura la trama completa que contiene los datos de interés. A continuación, se extraen los datos correspondientes a las manos y se almacenan en la variable *hand*, de la cual se extraen directamente los cuaterniones correspondientes a la rotación de los dedos de la mano. Esta sección del código se engloba dentro de un bucle *for* con el fin de controlar la cantidad de muestras que se recogen, y tras una serie de operaciones matemáticas, previamente explicadas, se almacena el valor del ángulo correspondiente en el *buffer* denominado *PinkyAngleBuffer*. Posteriormente se procede con la captura de los valores asociados al movimiento de la mano, almacenando los resultados en el *buffer* denominado *PalmVelocityBuffer*.

En la Figura 5.22 se muestra el código correspondiente a la segunda parte de esta función. Este extracto del código se focaliza en la captura de los valores denominados instantáneos, de forma que al contrario que con la primera sección de la función, esta sección únicamente se ejecuta una sola vez para cada dedo de la mano. A su vez se ejecutan las funciones correspondientes al cálculo de la desviación típica para los valores previamente almacenados.

```
    if (index != 0){
      index = 0;
    }
    //InstantFingers
    for (int j = 0; j < FINGERS; j++){
      quat2.v[0] = hand->digits[j].bones[1].rotation.x;
      quat2.v[1] = hand->digits[j].bones[1].rotation.y;
      quat2.v[2] = hand->digits[j].bones[1].rotation.z;
      quat2.v[3] = hand->digits[j].bones[1].rotation.w;
      firstVector = quat2basis(quat2);
      quat3.v[0] = hand->digits[j].bones[2].rotation.x;
      quat3.v[1] = hand->digits[j].bones[2].rotation.y;
      quat3.v[2] = hand->digits[j].bones[2].rotation.z;
      quat3.v[3] = hand->digits[j].bones[2].rotation.w;
      secondVector = quat2basis(quat3);
      angleY = (float)(AngleTo(firstVector, secondVector) * RAD_TO_DEG);
      nodeBuffer[iteraciones][0][index] = (float)(index + 1);
      nodeBuffer[iteraciones][1][index] = angleY;
      index++;
    }
    //is_Extended
    for (int j = 0; j < FINGERS; j++){
      if (hand->digits[j].is_extended){
        nodeBuffer[iteraciones][0][index] = (float)(index + 1);
        nodeBuffer[iteraciones][1][index] = 100.0;
        index++;
      }
      else{
        nodeBuffer[iteraciones][0][index] = (float)(index + 1);
        nodeBuffer[iteraciones][1][index] = 0.0;
        index++;
      }
    }
    //Velocidad de la Palma
    for (int m = 0; m < VectorPos; m++){
      PalmVelocitySum.v[m] = 0;
      for (int n = 0; n < SAMPLES; n++){
        PalmVelocitySum.v[m] = PalmVelocitySum.v[m] + PalmVelocityBuffer[m][n];
      }
      PalmVelocityMean.v[m] = PalmVelocitySum.v[m] / (float)SAMPLES;
      float sqDevSum = 0;
      for (int o = 0; o < SAMPLES; o++){
        sqDevSum += (float)pow(PalmVelocityMean.v[m] - (PalmVelocityBuffer[m][o]), 2);
      }
      stDev.v[m] = (float)sqrt(sqDevSum / (float)SAMPLES);
      stDev.v[m] = stDev.v[m] / 10;
    }
    if (stDev.v[0] > 8.9118155 || stDev.v[1] > 6.5256798 || stDev.v[2] > 4.3590943){
      nodeBuffer[iteraciones][0][index] = (float)(index + 1);
      nodeBuffer[iteraciones][1][index] = 100.0;
      index++;
    }
    else{
      nodeBuffer[iteraciones][0][index] = (float)(index + 1);
      nodeBuffer[iteraciones][1][index] = 0.0;
      index++;
    }
```

Figura 5.22.: Función captureSamples, parte 2

Como se observa en la Figura 5.22, se procede a la extracción de la rotación de cada dedo a través de los cuaterniones, para posteriormente aplicar las transformaciones matemáticas pertinentes y así obtener el ángulo de flexión para cada dedo de la mano. El resultado se almacena directamente en el *buffer* denominado *nodeBuffer*, que se encarga de almacenar los valores que serán posteriormente escalados y utilizados para la predicción de símbolos. El proceso de captura se ejecuta secuencialmente obteniendo primero los valores

de los ángulos de los dedos, para a continuación obtener los valores correspondientes a si cada dedo se encuentra extendido o no, mediante un proceso similar al anterior donde el código se engloba en un bucle *for* que recorre las iteraciones hasta la variable *FINGERS*, la cual representa un valor numérico correspondiente al número de dedos. Finalmente, en esta sección del código se realizan los cálculos pertinentes, para la obtención de la desviación típica de los valores recogidos con anterioridad. La tercera parte de la función, que se muestra en la Figura 5.23, se centra en el cálculo de la desviación típica el movimiento del dedo meñique y en la obtención del vector normal a la palma de la mano.

```
        //Movimiento del meñique
        PinkyAngleSum = 0;
        for (int n = 0; n < SAMPLES; n++){
          PinkyAngleSum = PinkyAngleSum + PinkyAngleBuffer[n];
        }
        PinkyAngleMean = PinkyAngleSum / (float)SAMPLES;
        float PinkysqDevSum = 0;
        for (int o = 0; o < SAMPLES; o++){
          PinkysqDevSum += (float)pow(PinkyAngleMean - (PinkyAngleBuffer[o]), 2);
        }
        PinkySTDEV = (float)sqrt(PinkysqDevSum / (float)SAMPLES);
        PinkySTDEV = PinkySTDEV / 10;
        if (PinkySTDEV < 1.042565f){
          nodeBuffer[iteraciones][0][index] = (float)(index + 1);
          nodeBuffer[iteraciones][1][index] = 0.0;
          index++;
        }
        else{
          nodeBuffer[iteraciones][0][index] = (float)(index + 1);
          nodeBuffer[iteraciones][1][index] = 100.0;
          index++;
        }
        //PalmNormal
        normal = hand->palm.normal;
        if (normal.v[1] < 0){
          nodeBuffer[iteraciones][0][index] = (float)(index + 1);
          nodeBuffer[iteraciones][1][index] = 0.0;
          index++;
        }
        else{
          nodeBuffer[iteraciones][0][index] = (float)(index + 1);
          nodeBuffer[iteraciones][1][index] = 100.0;
          index++;
        }
        iteraciones++;
      }
    }
    clock_t difference = clock() - timeBefore;
    msec = difference * 1000 / CLOCKS_PER_SEC;
    printf("Se tardo %i segundos en completar el proceso de captura. \n"
    ,(int)(msec / 1000));
  }
}
```

Figura 5.23.: Figura captureSamples, parte 3

En la Figura 5.23, y como se especificó anteriormente, se observa que en primer lugar se realizan las formulaciones pertinentes para el cálculo de la desviación típica correspondiente al movimiento del dedo meñique, almacenando los resultados en *nodeBuffer*. A continuación, se obtiene el vector normal a la palma de la mano como ultimo atributo del *set* de datos que se pretende construir, almacenando los resultados una vez en el *buffer* especifico. Cabe destacar que la última parte de la función se utiliza para el cálculo del tiempo de ejecución del proceso de captura, mostrando por pantalla la cantidad de tiempo consumida en ejecutar este recurso.

Para realizar el control de plataforma se diseñó un menú, que se incluyó en la función denominada de la misma manera. En la Figura 5.24 se muestra el código correspondiente a esta función.

```
int menu(int i){
    int option = 9;
    if(i == 7){
      option = 0;
      return option;
    }
    printf(
      "|---------------------------------------|\n"
      "|                  MEN%c                 |\n"
      "|---------------------------------------|\n"
      "| 1. Instrucciones de uso.              |\n"
      "| 2. Reconocer un s%cmbolo.             |\n"
      "| 3. Recoger muestras de la mano.       |\n"
      "| 4. Escalar valores.                   |\n"
      "| 5. Realizar una predicci%cn.          |\n"
      "| 6. Cargar otro modelo.                |\n"
      "| 0. Salir.                             |\n"
      "|---------------------------------------|\n"
      "|*Se debe introducir un valor num%crico.|\n"
      "|---------------------------------------|\n"
    ,(char)233, (char)161, (char)162, (char)130);

    while( scanf_s("   %d", &option, MAXMENU)){
      if(option > 0 || option < 6){
        return option;
      }else if(!isdigit(option)){
        return 0;
      }else{
        printf("Opci%cn incorrecta...\n",(char)162);
        printf("Escoja otra opci%cn.\n",(char)162);
      }
    }
}
```

Figura 5.24.: Función menú

En este caso, la función obtiene como parámetro de entrada un valor numérico. Esto se debe a la necesidad de terminar el programa en ciertos casos específicos. Principalmente la función se encarga de mostrar por pantalla un menú y mediante el uso de la sentencia *scanf_s* capturar un valor numérico como respuesta del usuario. Como se observa la opción de entrada se engloba dentro de un bucle *while*, puesto que se pretende que el usuario únicamente seleccione una opción dentro del rango permitido, arrojando un mensaje de error en caso de que dicha opción se encuentre fuera del rango. Aunque se especifica que únicamente se deben introducir valores numéricos, en caso de que un usuario introdujese algún carácter, el programa terminaría de inmediato.

Para la utilización del programa se desarrolló un manual de instrucciones que se incluye en el código de la plataforma de reconocimiento, mostrado en la Figura 5.25. La función denominada *manual* únicamente imprime por pantalla el manual de instrucciones.

```
void manual(void){
  printf(
      "|---------------------------------------|\n"
      "|          Instrucciones de uso         |\n"
      "|---------------------------------------|\n"
    "|---------------------------------------|\n"
      "|1.- Posicionar mano antes de capturar.|\n"
    "|2.- Capturar muestras.                 |\n"
    "|3.- Realizar escalado de valores.      |\n"
    "|4.- Realizar una predicci%cn.           |\n"
    "|---------------------------------------|\n"
    "|* La opci%cn 2. realiza el proceso de  |\n"
    "|* forma automatica.                    |\n"
    "|---------------------------------------|\n"
    "|* Adicionalmente se puede realizar la |\n"
    "|* carga de otro modelo. El mismo debe |\n"
    "|* estar contenido en la carpeta de la |\n"
    "|* soluci%cn.                           |\n"
      "|---------------------------------------|\n"
      "|---------------------------------------|\n"
    ,(char)162, (char)162, (char)162);
}
```

Figura 5.25.: Función manual

A continuación, en la Figura 5.26, se muestra el código correspondiente a la primera parte de la función *main* del código desarrollado. Inicialmente en esta sección del código se realiza la conexión con el dispositivo *Leap Motion* a través de la ejecución de la función *OpenConnection* la cual se incluye en la librería denominada *LeapC.h* y que se aporta con la instalación del software que controla el *Leap Motion*.

```
int main(int argc, char** argv) {
  OpenConnection();
  while(!IsConnected)
    millisleep(100);
  deviceProps = GetDeviceProperties();
  if(!deviceProps){
    printf("Controlador no detectado...\n");
    printf("Conecte el controlador.\n");
    while(!deviceProps){
      deviceProps = GetDeviceProperties();
      millisleep(100);
    }
  }else{
    printf("Conectado. \n");
    printf("Dispositivo: %s.\n", deviceProps->serial);
  }
  if(loadmodel){
    loadModel(modelName);
    loadmodel = FALSE;
  }
```

Figura 5.26.: Función main, parte1

Posteriormente, se realiza la extracción de los parámetros específicos del controlador y se almacenan en la variable *deviceProps*. A su vez, mediante la utilización de un bucle *while*, se espera a que dichos parámetros se extraigan correctamente para cerciorarse de que la conexión con el dispositivo se completa satisfactoriamente. Como se expuso previamente, al iniciar la plataforma se realiza el precargado de un modelo que debe estar contenido en la carpeta de ejecución de la plataforma. Mediante la ejecución de la función *loadModel* se realiza la carga de dicho modelo.

En la Figura 5.27 se muestra el código correspondiente a la segunda parte de la función *main*. Inicialmente se almacena el valor numérico de la respuesta de la función *menu* en la variable *choice*. Dependiendo del valor de dicha variable, se ejecutan las funciones correspondientes a esa opción. Al estar la sección englobada en un bucle *while*, se mantiene la ejecución del mismo hasta que la variable *choice* sea 0, indicando la finalización del programa.

```
int choice;
choice = menu(NULL);
while(choice != 0 ){
  switch(choice){
    case 1:
      system("cls");
      manual();
      break;
    case 2:
      system("cls");
      printf(
        "|--------------------------------------|\n"
        "|       2. Reconocer un s%cmbolo.       |\n"
        "|--------------------------------------|\n"
        "|* Recuerde posicionar la mano sobre el|\n"
        "|* controlador para capturar muestras. |\n"
        "|--------------------------------------|\n"
      ,(char)161);
      millisleep(1000);
      captureSamples();
      printf("Ya puede retirar la mano...\n");
      millisleep(1000);
      scaleValues(nodeBuffer);
      predict(model, scaledNodeBuffer);
      break;
    case 3:
      system("cls");
      printf(
        "|--------------------------------------|\n"
        "|    3. Recoger muestras de la mano.   |\n"
        "|--------------------------------------|\n"
        "|* Recuerde posicionar la mano sobre el|\n"
        "|* controlador para capturar muestras. |\n"
        "|--------------------------------------|\n"
      );
      millisleep(1000);
      captureSamples();
      break;
    case 4:
      system("cls");
      printf(
        "|--------------------------------------|\n"
        "|         4. Escalar valores.          |\n"
        "|--------------------------------------|\n"
      );
      scaleValues(nodeBuffer);
      break;
    case 5:
      system("cls");
      printf(
        "|--------------------------------------|\n"
        "|     5. Realizar una predicci%cn.      |\n"
        "|--------------------------------------|\n"
      ,(char)167);
      predict(model, scaledNodeBuffer);
      break;
    case 6:
      system("cls");
      printf(
        "|--------------------------------------|\n"
        "|        6. Cargar otro modelo.        |\n"
        "|--------------------------------------|\n"
      );
      loadmodel = TRUE;
      printf("Introduzca el nombre del modelo. \n");
      fflush(stdin);
      scanf_s("   %s", modelName, MAXINPUT);
      loadModel(modelName);
      break;
  }
  printf("\n\n");
  choice = menu(NULL);
}
system("cls");
printf(
        "|--------------------------------------|\n"
        "|              0. Salir.               |\n"
        "|--------------------------------------|\n"
        "|* Hasta la pr%cxima...                 |\n"
        "|--------------------------------------|\n"
      ,(char)162);
millisleep(1000);
system("exit");
CloseConnection();
return 0;
}
```

Figura 5.27.: Función main, parte 2

Finalmente, en la Figura 5.28 se muestra el código correspondiente a la definición y declaración de variables que se utilizan a lo largo de la ejecución del programa en la plataforma.

```
#undef __cplusplus
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <string.h>
#include <math.h>
#include <time.h>
#ifdef _WIN32
#include <Windows.h>
#else
#include <unistd.h>
#endif
#include "LeapC.h"
#include "ExampleConnection.h"
#include "svm.h"
#define M_PI 3.14159265358979323846
#define EPSILON 1.192092896e-07f
#define RAD_TO_DEG 57.295779513f
#define FINGERS 5
#define SAMPLES 30
#define VectorPos 3
#define INDEX 13
#define UPPER 1
#define LOWER -1
#define NODE 2
#define INSTANCE 1
#define MAXINPUT 10
#define MAXMENU 1
#define CLASS 30
int index = 0;
int64_t lastFrameID = 0;
bool alreadyScaled;
int iteraciones;
LEAP_TRACKING_EVENT *frame;
LEAP_HAND *hand;
LEAP_QUATERNION quat, quat1, quat2, quat3;
LEAP_DEVICE_INFO *deviceProps;
LEAP_VECTOR vector, vector1, vector2, vectorX, vectorY, vectorZ, firstVector, secondVector,
            PalmVelocitySum, PalmVelocityMean, stDev, Velocity, normal;
LEAP_MATRIX_3x3 matriz;
float PalmVelocityBuffer[VectorPos][SAMPLES];
float PinkyAngleBuffer[SAMPLES];
float PinkyAngleSum, PinkyAngleMean, PinkySTDEV, angleY;
struct svm_node *node;
struct svm_model *model;
float nodeBuffer[INSTANCE][NODE][INDEX];
float scaledNodeBuffer[INSTANCE][NODE][INDEX];
double exampleBuffers[1][NODE][INDEX];
float maximum[INDEX], minimum[INDEX], prediction[INDEX];
float range[INDEX][NODE];
char modelName[MAXINPUT] = {"model"};
char *previousmodelName[MAXINPUT] = {"model"};
double predictedLabel;
char labels[CLASS];
```

Figura 5.28.: Definición y declaración de variables

Capítulo 6. Validación y resultados

En este sexto capitulo se recoge la validación completa del sistema desarrollado, así como los resultados obtenidos a partir del proceso de validación, así como los aspectos a considerar que existen a la hora de implementar la plataforma *HW/SW* final.

6.1. Validación funcional

La validación funcional de la plataforma final desarrollada en el presente Trabajo Fin de Máster se llevó a cabo a partir de la ejecución de las distintas opciones contempladas. Por simplificación, se añadió la segunda opción del menú, correspondiente a reconocer un símbolo, con el fin de realizar la ejecución de las opciones de captura de datos y de escalado de los valores capturados, así como realizar una predicción acorde a los valores recogidos. Estas opciones se implementan a su vez de forma individual a través de las opciones del menú número 3, 4 y 5. Por ello, la verificación de la plataforma se divide en dos partes, por un lado, la ejecución de la segunda opción, y por otro la ejecución individual paso a paso, como se muestra a continuación.

En primer lugar, en la Figura 6.1 se muestra el menú diseñado para la plataforma final de reconocimiento, en esta se engloban todas las opciones que se pueden ejecutar a través del mismo. Como se observa, al comienzo de la ejecución de la plataforma se carga un modelo previamente contenido en la carpeta de ejecución del programa.

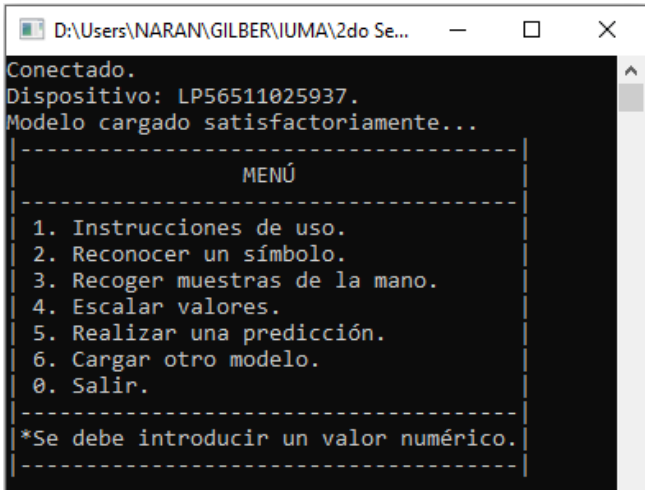


Figura 6.1.: Ejecución menú

A continuación, en la Figura 6.2, se muestra la ejecución de la primera opción del menú, correspondiente a las instrucciones de uso de la plataforma de reconocimiento.

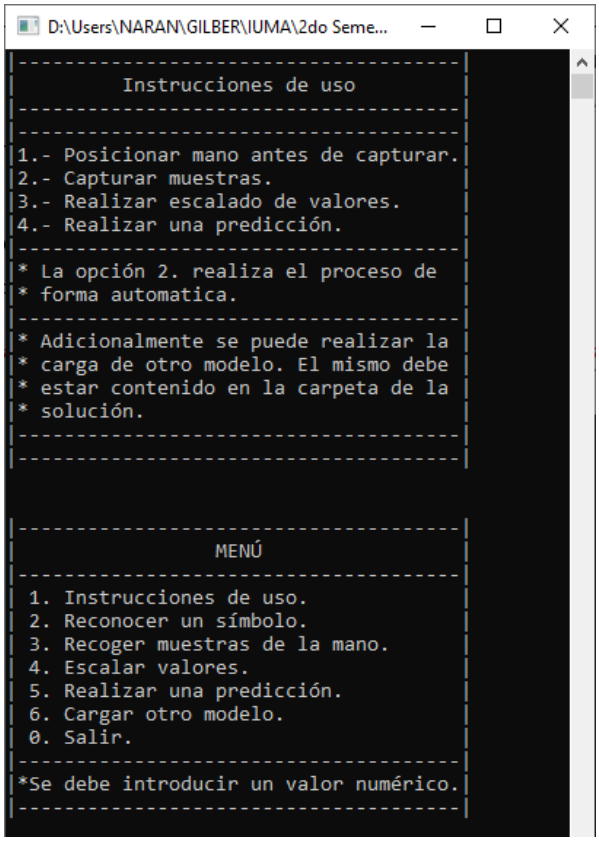


Figura 6.2.: Ejecución instrucciones de uso

Como se puede observar en la Figura 6.2, con cada ejecución de una sección del menú, el mismo reaparece en la parte inferior, omitiéndose por simplificación en las siguientes capturas del proceso de validación.

Para llevar a cabo la traducción de un símbolo es necesario seguir el orden mostrado en las instrucciones de uso de la plataforma. Por ello, en la Figura 6.3 se muestra la ejecución de la tercera opción del menú, correspondiente a la captura de los datos de las manos a través del dispositivo *Leap Motion*.

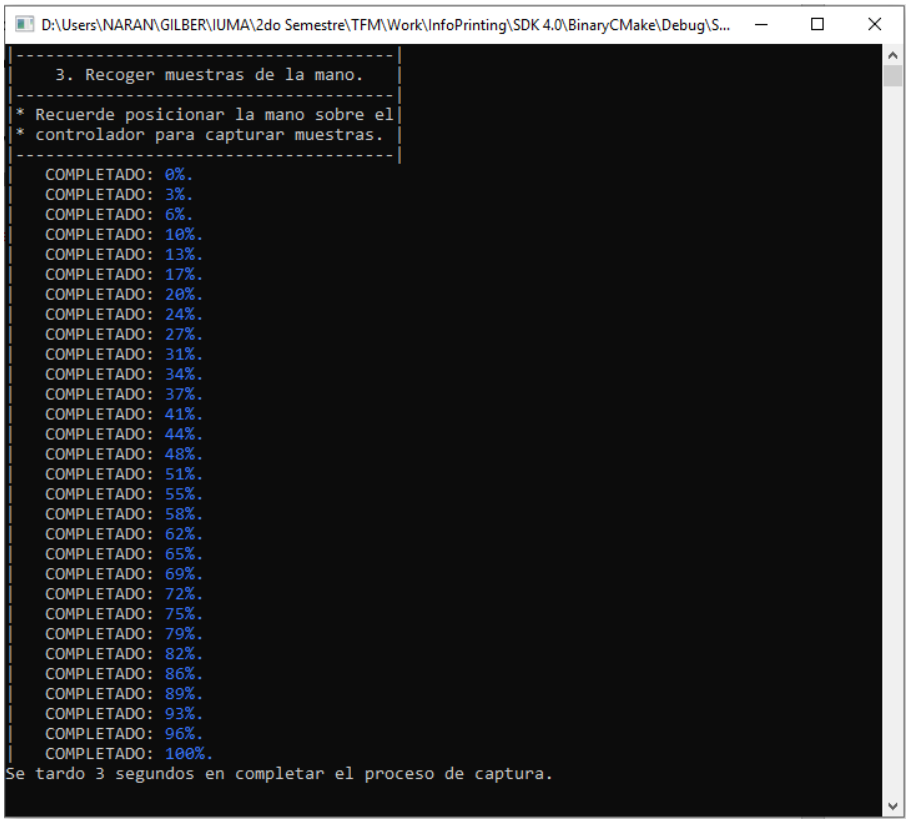


Figura 6.3.: Ejecución Recoger muestras

Como se puede observar en la Figura 6.3, se realizó la introducción de un temporizador para calcular el tiempo que consume la ejecución de esta opción, siendo para este caso de 3 segundos. Una vez realizada la captura de datos, y como se ha mencionado previamente, se procede a escalar los valores recogidos. En la Figura 6.4 se muestra la ejecución correspondiente a dicha opción.

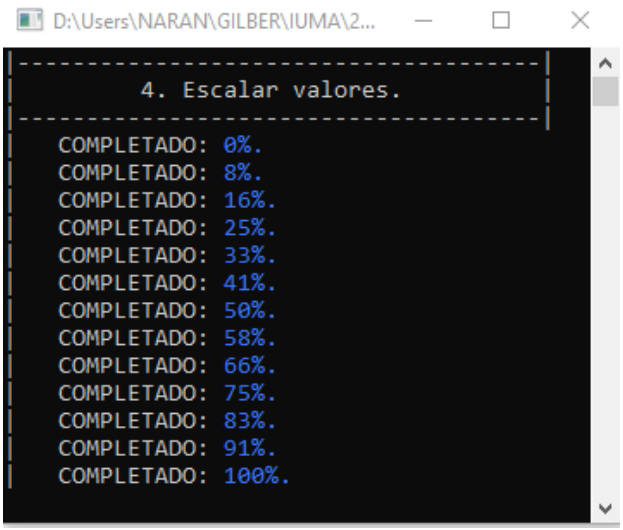


Figura 6.4.: Ejecución escalar valores

La tarea de escalado de valores se considera ejecutada cuando alcanza un porcentaje de completado correspondiente al 100%. Una vez realizado este proceso, se continuaría con la ejecución de la quinta opción del menú, correspondiente a la realización de una predicción a partir de los valores capturados y escalados, como se muestra en la Figura 6.5.

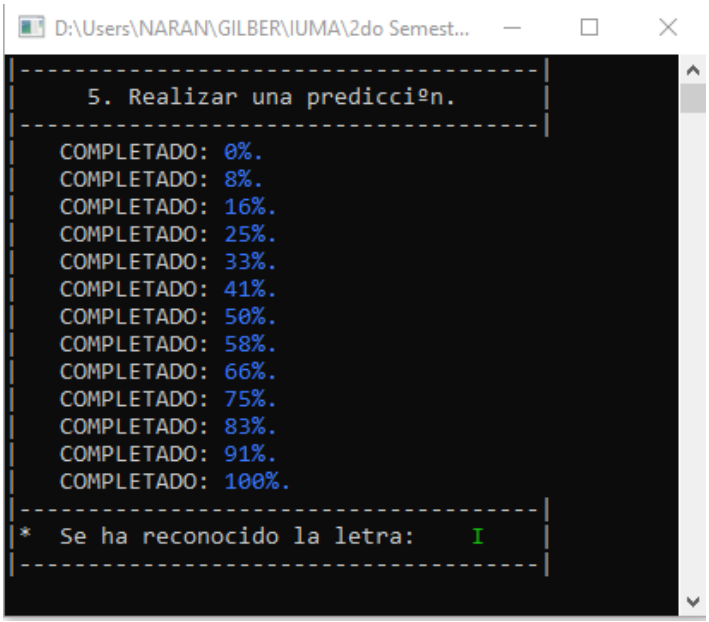


Figura 6.5.: Ejecución realizar una predicción

Con la ejecución de la opción mencionada, se muestra por pantalla la letra correspondiente al símbolo que se tradujo. Como se puede observar, en este caso particular corresponde al a letra “I”.

Como opción adicional se incluyó la posibilidad de cargar un modelo distinto. Así, en la Figura 6.6 se muestra la ejecución de la sexta opción del menú. Para este caso es necesario la introducción del nombre del modelo que se pretende incluir en la plataforma. Para ello, dicho modelo debe estar contenido dentro de la carpeta donde se encuentra el archivo ejecutable de la plataforma.

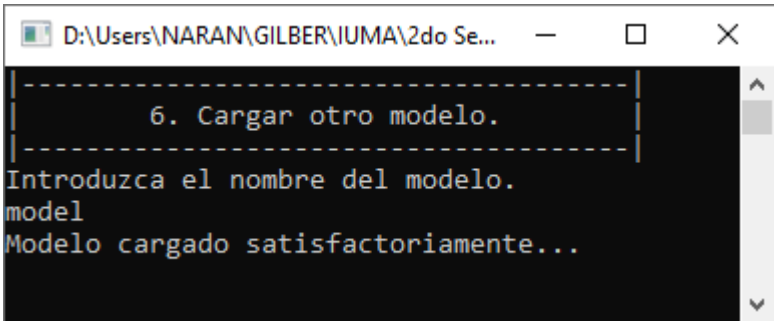


Figura 6.6.: Ejecución cargar otro modelo

A continuación se muestra la ejecución de la segunda opción del menú. Como se expuso previamente, esta opción realiza los 3 procesos principales necesarios para la traducción de los símbolos del alfabeto dactilológico de la lengua española de signos en un único proceso. En la Figura 6.7 se muestra la ejecución correspondiente a dicha opción. Cabe destacar que, al igual con la opción de predecir un resultado, al finalizar devuelve por pantalla la letra correspondiente al símbolo identificado, que para este caso se corresponde con la letra “E”.

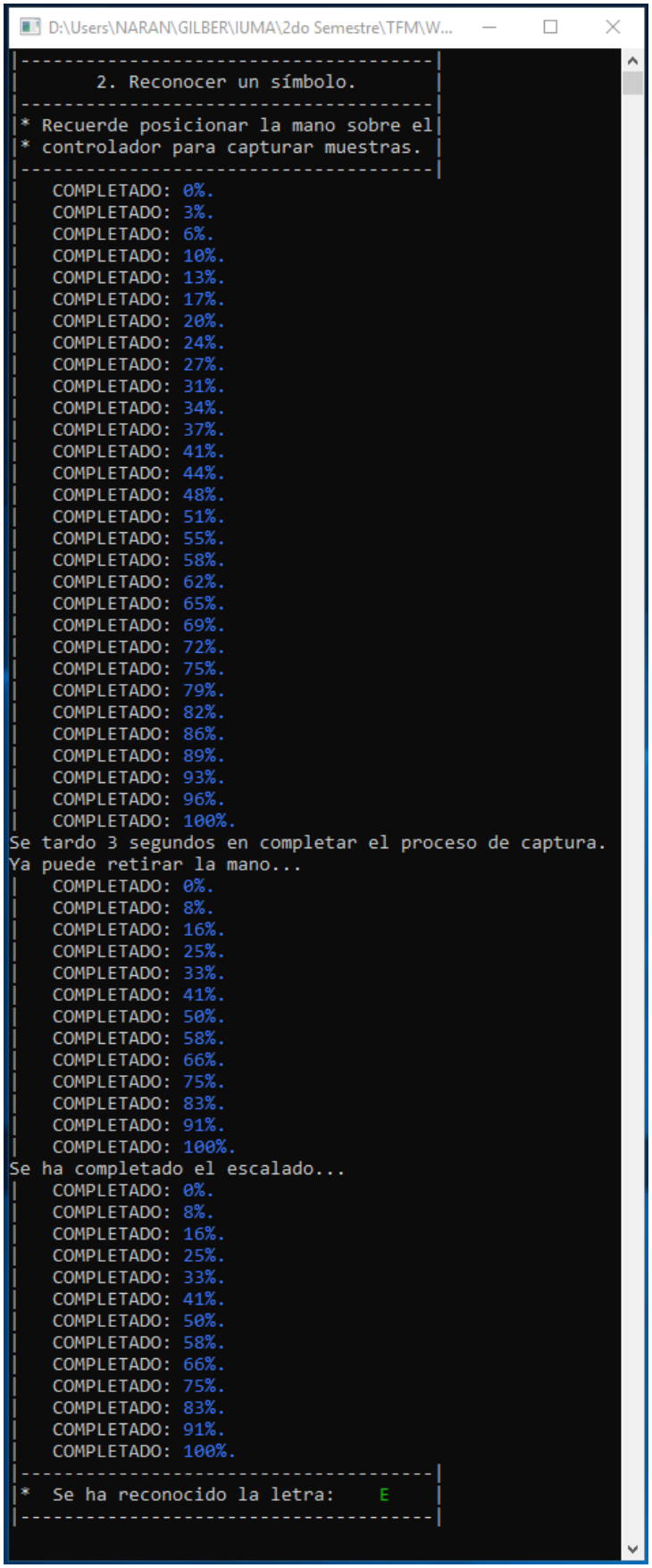


Figura 6.7.: Ejecución reconocer un símbolo

Finalmente, en la Figura 6.8 se muestra la ejecución de la última opción del menú, la cual se corresponde con la terminación del programa.

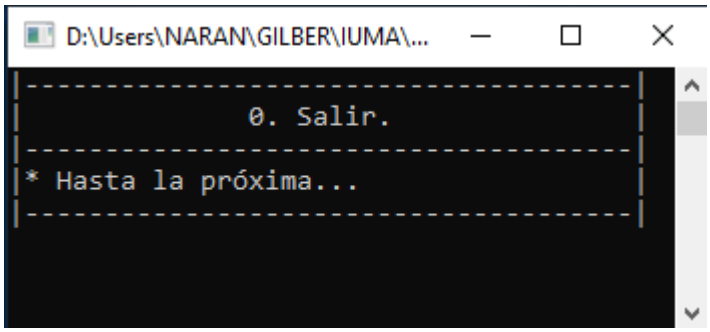


Figura 6.8.: Ejecución Salir

6.2. Resultados

El rendimiento de los clasificadores se mide a través de un nivel de puntuación en forma de porcentaje que representa el número de aciertos. Normalmente, este porcentaje mide el rendimiento total del clasificador y, como se ha mencionado con anterioridad, se denomina *accuracy*. Para el caso específico de este tipo de clasificador, dicho valor se obtiene a partir de la fórmula (10) que se observa a continuación:

$$Accurracy\ (\%) = \frac{aciertos}{total * 100} \qquad (10)$$

A su vez, la forma más común de representar dicho valor de aciertos es a través de las matrices de confusión. Las matrices de confusión se componen de una matriz de dimensión n x n, donde n representa el número de clases que se pretenden clasificar, en la columna principal se observan las instancias reales, mientras que en la fila principal se representan los valores predichos. De esta forma se establece una manera eficaz y visual con la que poder estudiar los resultados de una clasificación de instancias.

En esta sección del capítulo se presentan las matrices de confusión implementadas a partir de los experimentos realizados a lo largo del desarrollo de la plataforma completa. Inicialmente se comenzó con la ejecución de algoritmo de clasificación para el *set* de datos correspondiente a los símbolos estáticos del lenguaje, generando la matriz de confusión que se muestra en la Figura 6.9.

	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

Figura 6.9.: Matriz de confusión para símbolos estáticos

Como se observa en la Figura 6.9, el algoritmo *SVM* clasificó correctamente 51 de las 54 instancias disponibles, fallando principalmente en la detección del simbolo “M”, lo cual se traduce en un valor de *accuracy* del 94,4%. Por otro lado, en la Figura 6.10 se muestra la matriz de confusión correspondiente a la ejecución del clasificador para el reconocimiento de símbolos dinámicos.

	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	2	0	0	0	0	1	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

Figura 6.10.: Matriz de confusión para símbolos dinámicos

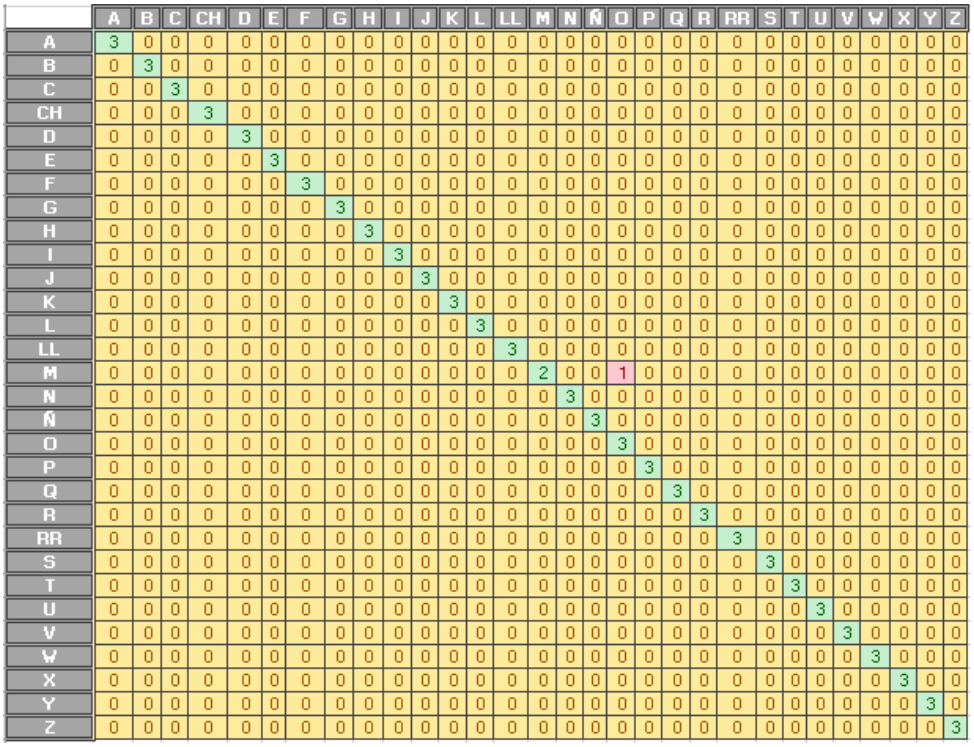
Para la clasificación de símbolos dinámicos, el algoritmo reconoció satisfactoriamente 33 de las 36 instancias disponibles, fallando en la detección del símbolo “W”, el cual clasificó incorrectamente 2 veces como el símbolo “V”. Así mismo, el algoritmo clasificó incorrectamente 1 vez el símbolo “G”. En este caso el valor de *accuracy* reflejado corresponde al 91,6%.

En la Figura 6.11 se muestra la matriz de confusión correspondiente a ejecución de los símbolos dinámicos con el nuevo parámetro incorporado correspondiente a *palm.normal*.

	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

Figura 6.11.: Matriz de confusión para símbolos dinámicos + palm.normal

Como se observa, en este caso el algoritmo únicamente clasificó incorrectamente el símbolo correspondiente a la letra “W”, corrigiendo uno de los fallos que anteriormente se obtuvieron, y mejorando ligeramente las prestaciones del sistema. En este caso se clasificaron correctamente 34 de las 36 instancias disponibles, cuyo valor de *accuracy* corresponde con un 94,4%. Una vez finalizado con el reconocimiento de los símbolos dinámicos se llevó a cabo la clasificación de todos los símbolos del alfabeto dactilológico de la *LSE*, como se muestra en la Figura 6.12.



	A	B	C	CH	D	E	F	G	H	I	J	K	L	LL	M	N	Ñ	O	P	Q	R	RR	S	T	U	V	W	X	Y	Z
A	3	0	0	0	0	0	0	0	0	0	0	0	0	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	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	0	0	0	0	0	0	0	0	0	0	0	0
CH	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LL	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ñ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
RR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3

Figura 6.12.: Matriz de confusión para símbolos completos

Al unificar ambos *sets* de datos se logra reducir la cantidad de errores cometidos por el algoritmo *SVM* al clasificar, como se puede observar en la Figura 6.12. En este caso el algoritmo únicamente clasificó de forma errónea el valor correspondiente al símbolo “M” y además 1 sola vez. Específicamente se realizó la clasificación correcta de 89 de las 90 instancias disponibles, lo que representa un valor de *accuracy* del 98,8%. Aunque este valor se considere bastante elevado, para el desarrollo de este *TFM* se requería de un *accuracy* del 100%, por lo que se procedió a modificar los parámetros de ajuste del clasificador, resultando en una matriz de confusión como la que se muestra en la Figura 6.13.

	A	B	C	CH	D	E	F	G	H	I	J	K	L	LL	M	N	Ñ	O	P	Q	R	RR	S	T	U	V	W	X	Y	Z
A	3	0	0	0	0	0	0	0	0	0	0	0	0	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	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	0	0	0	0	0	0	0	0	0	0	0	0
CH	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
LL	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ñ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
RR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3

Figura 6.13.: Matriz de confusión para símbolos completos con parámetros de ajuste

Finalmente, se puede observar que al realizar la modificación del parámetro de ajuste del clasificador *SVM* se alcanza el nivel de *accuracy* deseado. Específicamente se realiza la correcta clasificación de las 90 instancias disponibles, lo que representa un valor de *accuracy* correspondiente al 100%.

6.3. Consideraciones

A pesar de haber cumplimentado satisfactoriamente todos los objetivos propuestos para el desarrollo del presente *TFM* existen varias consideraciones a tener en cuenta, tanto por parte de la utilización del algoritmo de clasificación desarrollada, como del uso del controlador *Leap Motion*.

En primer lugar, se parte de la base de que para realizar una clasificación se debe poseer un banco de datos suficientemente grande como para realizar todas las pruebas pertinentes. En el caso específico del desarrollo de este *TFM* se parte de un banco de datos de 10 instancias por símbolo. Esto se debe principalmente a que dicho banco de datos tuvo

que ser generado de forma propia, así como por la limitación de tiempo existente a la hora de realizar el desarrollo del *TFM.* Sin embargo, como primera aproximación se considera suficiente. Las pruebas pertinentes mencionadas con anterioridad se basan en ir reduciendo la cantidad de muestras del *set* de datos de testeo, para así averiguar el número mínimo de instancias necesario para conseguir una correcta clasificación, entre otras pruebas realizables.

En cuanto al dispositivo *Leap Motion*, al utilizar cámaras de detección por infrarrojos, estas se ven considerablemente afectadas por la irradiación de la luz solar, permitiendo que el controlador capture valores anómalos que pueden llevar a errores a la hora de realizar la clasificación de un símbolo. En este sentido, se considera de vital importancia la realización de los símbolos de forma correcta, así como mantener la mano sobre el dispositivo el tiempo requerido para la correcta captura de los datos. Por otro lado, según indica el fabricante, el dispositivo tiene 2 posiciones fijas, lo que limita la utilización del controlador. Previamente a la realización del proceso de captura es necesario que el usuario coloque sobre el dispositivo *Leap Motion* la mano de forma abierta, con el fin de cerciorarse de que el dispositivo detecta correctamente la mano, para posteriormente realizar el símbolo que se pretende traducir. Cabe destacar que existe un límite tecnológico en cuanto al uso de este dispositivo, ya, que en cuanto alguno de los dedos de la mano sale del campo visual del controlador, este intenta compensarlo estimando la posición de los dedos, lo cual puede conllevar una captura de valores erróneos, afectando a la detección de los símbolos.

Capítulo 7. Conclusiones y líneas futuras

El séptimo y último capítulo se centra en las conclusiones obtenidas a partir del desarrollo de la plataforma de reconocimiento del alfabeto dactilológico de la lengua de signos española, y las líneas futuras que pueden derivar del mismo.

7.1. Conclusiones

En vista de los resultados favorables obtenidos en cada uno de los experimentos realizados, se puede afirmar que con el presente Trabajo Fin de Máster se ha logrado el objetivo de desarrollar un prototipo de plataforma HW/SW, capaz de reconocer los símbolos pertenecientes al alfabeto dactilológico de la lengua española de signos y a partir del dispositivo *Leap* Motion representarlos de forma escrita.

En concordancia con los objetivos establecidos, se ha llevado a cabo una evaluación de los diferentes métodos de captación de datos, además de realizar un estudio del alfabeto dactilológico de la *LSE*, a partir del cual se han implementado los parámetros que permitiesen una mayor discriminación entre los mismos símbolos. Dichos parámetros son extraíbles a partir del *kit* de desarrollo (*SDK*) proporcionado por el fabricante del controlador *Leap Motion*, que se utilizó para la captura de datos. Una vez realizado el estudio, y determinados los parámetros que se utilizarían en la generación de los *data set*, se procedió a la generación de un código descrito en C que realizase la captura de dichos parámetros. Así mismo, se llevó a cabo un estudio sobre los distintos métodos de predicción de instancias, para lo cual, se llevó a cabo la implementación de la librería *LIBSVM*, y con ello, la incorporación de un algoritmo con el que poder realizar la clasificación de cada símbolo. Finalmente, se englobó todo en una única plataforma de traducción.

Además, gracias a su diseño, el sistema de captación de imágenes permite al usuario interactuar con la plataforma de una forma sencilla. Este factor favorece en gran medida a aquellas personas que sufren algún tipo de discapacidad auditiva, ya que solamente se

requiere ejecutar unos simples comandos para lograr la traducción del símbolo deseado a lenguaje escrito. Por otra parte, la plataforma se diseñó para tener la menor latencia posible, tomando solamente unos pocos segundos desde que se capturan los datos referentes a la posición de la mano, hasta que se realiza la traducción.

Finalizado el proceso de implementación de la plataforma HW/SW, y tras la experiencia adquirida durante la realización de la misma, se pueden establecer las siguientes conclusiones:

- La amplia documentación disponible *online* constituye una buena fuente de ayuda. Así mismo, los foros del *Leap Motion* han resultado de gran utilidad a la hora de resolver dudas específicas.
- El *kit* de desarrollo del dispositivo *Leap Motion* proporciona al usuario un sistema potente a la par que eficaz para la extracción de parámetros a partir del controlador, permitiendo la integración del mismo en múltiples aplicaciones.
- Debido a que la plataforma se desarrolló como código abierto, se da la posibilidad de realizar modificaciones sobre el código, permitiendo la implementación de nuevas funcionalidades.

7.2. Líneas futuras

Una vez alcanzados los objetivos propuestos para el desarrollo de este TFM, se procede a proponen las siguientes líneas futuras para el desarrollo de la plataforma:

- Integrar la función de generación de modelos, denominada *svm-train*, con el fin de proveer de autonomía a la plataforma de reconocimiento de símbolos.
- Incluir nuevos símbolos de la lengua de signos española que representen palabras completas.
- Implementar la funcionalidad de comenzar el reconocimiento de un símbolo al detectar que se ha posicionado una mano sobre el controlador.
- Integrar en la plataforma la capacidad de visualización de la posición de las manos, para así aportar robustez al sistema.

Bibliografía

[1] OMS | Informe mundial sobre la discapacidad. [Online]. Available: http://www.who.int/disabilities/world_report/2011/es/. [Accessed: March 2018]

[2] OMS | Sordera y pérdida de la audición. [Online] Available: http://www.who.int/mediacentre/factsheets/fs300/es/. [Accessed: March 2019]

[3] Ministerio de Educación y Ciencia, "Población con discapacidad auditiva." [Online] Available: http://ares.cnice.mec.es/informes/17/contenido/19.htm. [Accessed: March 2019]

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

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

[6] "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]

[7] 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.

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

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

[10] 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].

[11] 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].

[12] Gunawardane P, Medagedara N. "Comparison of Hand Gesture inputs of Leap Motion Controller & Data Glove into a Soft Finger". Proceedings - 2017 IEEE 5th International Symposium on Robotics and Intelligent Sensors, IRIS 2017. ISBN: 9781538613429. [Online] Available: https://ieeexplore.ieee.org/document/8250099 [Accessed October 2019].

[13] CNSE | Confederación estatal de personas sordas. "¿Qué es la lengua de signos?". [Online] Available: http://www.cnse.es/lengua.php [Accessed October 2019].

[14] BOE | Boletín Oficial del Estado. "Ley 27/2007, de 23 de octubre, por la que se reconocen las lenguas de signos españolas y se regulan los medios de apoyo a la comunicación oral de las personas sordas, con discapacidad auditiva y sordociegas.".

[Online] Available: https://www.boe.es/buscar/act.php?id=BOE-A-2007-18476 [Accessed October 2019].

[15] Alex Colgan. "How Does the Leap Motion Controller Work". [Online] Available: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/ [Accessed October 2019].

[16] 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].

[17] Figura Posicionamiento del Leap Motion. [Online] Available: https://mocap.reallusion.com/iclone-motion-live-mocap/leap-motion.html [Accessed October 2019].

[18] 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].

[19] 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].

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

[21] 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].

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

[23] Leap Motion Developer. "Coordinate System". [Online] Available: https://developer-archive.leapmotion.com/documentation/cpp/devguide/Leap_Coordinate_Mapping.html [Accessed October 2019].

[24] Daniel Bachmann, Frank Weichert, Gerhard Rinkenauer. "Review of Three-Dimensional Human-Computer Interaction with Focus on the Leap Motion Controller". TU Dortmund University. 2018 [Online] Available: https://doi.org/10.3390/s18072194 [Accessed October 2019].

[25] A. A. Abd El-Aziz, A. Kannan. "JSON Encryption". Anna University. 2014 [Online]. Availble: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6921719 [Accessed October 2019].

[26] Leap Motion Developer. "System Architecture". V3. [Online] Available: https://developer-archive.leapmotion.com/documentation/cpp/devguide/Leap_Architecture.html [Accessed October 2019].

[27] Leap Motion Developer. "Tracking Model". [Online] Available: https://developer-archive.leapmotion.com/documentation/cpp/devguide/Leap_Coordinate_Mapping.html [Accessed October 2019].

[28] Alex Colgan. "Hand Hierarchy". [Online] Available: http://blog.leapmotion.com/wp-content/uploads/2014/08/hand-hierarchy.png [Accessed October 2019].

[29] Bill Hoffman, David Cole, John Vines. "Software Process for Rapid Development of HPC Software Using CMake". 2009 DoD High Performance Computing Modernization Program Users Group Conference. [Online] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5729493 [Accessed October 2019].

[30] Bougacha A, Boughariou J, Ben Slima M, Ben Hamida A, Ben Mahfoudh K, Kammoun O, Mhiri C. "Comparative study of supervised and unsupervised classification methods: Application to automatic MRI glioma brain tumors segmentation". 2018 4th International

Conference on Advanced Technologies for Signal and Image Processing, ATSIP 2018. [Online] Available: https://ieeexplore.ieee.org/document/8364463 [Accessed October 2019].

[31] Liu J, Li M, Wang J, Wu F, Liu T, Pan Y. "A survey of MRI-based brain tumour segmentation methods". [Online] Available: https://ieeexplore.ieee.org/document/6961028 [Accessed October 2019].

[32] Besimi N, Çiço B, Besimi A. "Overview of data mining classification techniques: Traditional vs. parallel/distributed programming models". 2017 6th Mediterranean Conference on Embedded Computing, MECO 2017 - Including ECYPS 2017, Proceedings. [Online] Available: https://ieeexplore.ieee.org/document/7977126 [Accessed October 2019].

[33] Fengxi Song, Zhongwei Guo, Dayong Mei. "Feature selection using principal component analysis". [Online] Available: https://ieeexplore.ieee.org/document/5640135 [Accessed October 2019].

[34] Kaur R, Himanshi E. "Face Recognition using Principal Component Analysis". IEEE International Advance Computing Conference. 2015. [Online] Available: https://ieeexplore.ieee.org/document/7154774 [Accessed October 2019].

[35] Hsu C, Chang C, Lin C. "A Practical Guide to Support Vector Classification". 2003. [Online] Available: https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf [Accessed October 2019].

[36] Cortes C, Vapnik V, Saitta L. "Support-Vector Networks Editor". Machine Leaming 1995 vol: 20 pp: 273-297. [Online] Available: https://dl.acm.org/citation.cfm?id=218929 [Accessed October 2019].

[37] Marti A. Hears. "SVMs-a practical consequence of learning theory". [Online] Available: https://www.semanticscholar.org/paper/SVMs-%E2%80%94-a-practical-consequence-of-learning-theory-Sch%C3%B6lkopf/00eb744cdf12275e512703dcc7e74c7435bd0d62 [Accessed October 2019].

[38] 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].

[39] 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.

[40] "Readme LIBSVM". [Online] Available: https://www.csie.ntu.edu.tw/~r94100/libsvm-2.8/README [Accessed October 2019].

[41] Leap Motion Developer. "Vector". [Online] Available: https://developer-archive.leapmotion.com/documentation/cpp/api/Leap.Vector.html?proglang=cpp [Accessed October 2019].

[42] 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].

[43] Soheil Sarabandi, Federico Thomas. "Accurate Computation of Quaternions from rotation matrix". [Online] Available: http://www.iri.upc.edu/files/scidoc/2068-Accurate-Computation-of-Quaternions-from-Rotation-Matrices.pdf [Accessed October 2019].

[44] MURTY S. CHALLA, JAY G. MOORE, DANIEL J. ROGERS. "A Simple Attitude Unscented Kalman Filter: Theory and Evaluation in a Magnetometer-Only Spacecraft Scenario". [Online]

Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7460892 [Accessed October 2019].

[45] Euclidean Space. "Conversion Quaternion to Matrix". [Online] Available: http://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToMatrix/ [Accessed October 2019].

[46] 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].