Universidad Autónoma de Yucatán

Maestría en Ciencias de la Computación

Título:

Reconocimiento de señas de las manos a través de técnicas de aprendizaje automático

I.S.C. Javier Armando Jiménez Villafaña

Asesora de tesis:
Dra. Anabel Martín González

Resumen

La Lengua de Señas Mexicana (LSM) es la lengua de la comunidad mexicana de sordo-mudos, que consiste en una serie de signos gestuales articulados con las manos y acompañados de expresiones faciales, mirada intencional y movimiento corporal, dotados de función lingüística; forma parte del patrimonio lingüístico de dicha comunidad, y es tan rica y compleja en gramática y vocabulario como cualquier lengua oral.

Actualmente, en México no existen sistemas que, de manera automática, sean capaces de reconocer el LSM. Por otra parte, los sistemas existentes fuera de México, realizan la tarea del reconocimiento a través de procesamiento digital de imágenes, o video con cámaras convencionales RGB, lo cual es costoso en tiempo computacional.

En este trabajo, se presenta un sistema para reconocer un conjunto de señas del LSM, formadas por una sola mano, e identificar el símbolo gesticulado mediante técnicas de aprendizaje automático, así como filtrado, discretización y extracción de características a partir de imágenes capturadas por un sensor de profundidad.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Una de las formas más naturales de interacción entre los seres humanos es la comunicación hablada. Cuando existen personas que por impedimentos físicos o naturales quedan imposibilitados de comunicarse verbalmente con otros seres humanos a su alrededor sucede entonces que estas personas comienzan a tener problemas para integrarse a la sociedad. Según el INEGI [15], en la sociedad mexicana, hasta el año 2010, había aproximadamente 1.18 millones de personas con impedimentos auditivos o del habla. Estos números crecen, tanto en zonas rurales como en la ciudad.

1.1. La Lengua de Señas Mexicana

La Lengua de Señas Mexicana (LSM) es la única forma de comunicación para la gente sordomuda. Normalmente, las personas sordomudas no pueden comunicarse con la gente oyente sin ayuda de un intérprete de la lengua de señas. Esta falta de interacción provoca el aislamiento de las personas sordomudas dentro de la sociedad. Es por esta razón que un sistema computacional que reconozca automáticamente el lenguaje de señas sería muy útil para facilitar la comunicación entre sordomudos y oyentes.

Todos los lenguajes de señas del mundo, incluyendo el LSM, están compuestos de señas, hechas principalmente con las manos y a veces incorporan otras partes del cuerpo. Para realizar las señas existe una mano dominante (en el caso de que la seña se haga únicamente con una mano), y una mano no dominante (en el caso de que la seña este hecha con las dos manos). Las señas pueden ser estáticas o con movimiento. Las señas representan una palabra completa, como se presenta en la figura 1.1. En caso de no existir la palabra dentro del conjunto de señas del LSM, se deletrea la palabra en español a

través de las señas del abecedario del LSM, como se presenta de acuerdo a la figura 1.2. En el lenguaje de señas existen vocales, consonantes, números, preposiciones, entre otros.

Figura 1.1: Ejemplos de Palabras del abecedario del LSM. De izquierda a derecha, se aprecian las palabras Manzana, Nuera y Tarde [22].

Figura 1.2: Ejemplos de letras del abecedario del LSM, de izquierda a derecha, se aprecian las letras M, N y T [22].

La principal motivación para realizar este proyecto de tesis, es el creciente número de personas con discapacidad auditiva o del habla y el escaso número de intérpretes de la lengua de señas necesarios para poder entender a esta población y la necesidad de no segregar a un grupo social por falta de medios para comunicarnos con ellos.

El problema desde un punto de vista computacional reside en el hecho de que actualmente en México no existe un sistema que, de manera automática y en tiempo real interprete la lengua de señas mexicana. Los sistemas existentes fuera de México, se basan en guantes electrónicos, o cámaras de video con imágenes a color, y en ambos casos el coste de componentes digitales y el tiempo computacional, representan mayores problemas para la creación de

dicho sistema.

La propuesta de solución que se presenta en este trabajo es la creación de un sistema que, de manera automática, pueda interpretar correctamente un subconjunto de señas de la lengua de señas mexicana; esto con el objetivo de que una persona con discapacidad auditiva o del habla pueda hacer uso del sistema, y sus gesticulaciones con las manos puedan ser correctamente interpretadas al idioma español y una persona hablante pueda entender su significado.

1.2. Contribuciones

- La principal contribución de este trabajo es la realización de un sistema automático de identificación de un subconjunto de señas del LSM mediante un sensor de profundidad RGBD y la extracción características Haar en 3D.

- Otra importante contribución de este trabajo es haber desarrollado un sistema robusto a ruidos e interferencias lumínicas, que consiste en uno de los principales problemas en el tratamiento e identificación de patrones en imágenes.

- También en este trabajo se presenta una manera novedosa de ajuste y centrado de imágenes a profundidad ya que al momento de elaborar este trabajo, no se encontró ninguna biblioteca compatible con esta tarea.

- Finalmente, una versión paralelizable del algoritmo de aprendizaje automático AdaBoost, se realizó para el entrenamiento de los patrones extraídos de cada una de las tercias de imágenes para agilizar el método y reducir los tiempos de entrenamiento, lo cual resultó bastante útil utilizando el auge de los procesadores multinúcleo.

1.3. Organización de la tesis

A continuación, se menciona brevemente la organización de este trabajo de tesis en los siguientes párrafos.

Capítulo 1.- Introducción. En este capítulo abordamos la justificación de este trabajo, así como la motivación para realizar la tarea del reconoci-

miento de un subconjunto de señas del LSM.

Capítulo 2.- Estado del arte. En este capítulo hablamos de todos los trabajos previos realizados hasta la fecha en cuanto al reconocimiento de las lenguas de señas en algunas de sus variantes, como por ejemplo la Lengua de Señas Indú, y la Lengua de Señas Americana.

Capítulo 3.- Marco teórico. En este capítulo hablamos de todos los conceptos y herramientas computacionales utilizados en este proyecto, describiendo a los algoritmos de extracción de características, los algoritmos de aprendizaje automático y la morfología matemática para el procesamiento de imágenes.

Capítulo 4.- Metodología. En este capítulo abordamos a profundidad el cómo se realizó este proyecto, desde la creación de la base de datos de señas, el preprocesamiento de las imágenes y la extracción de características Haar que sirven de entrada para el algoritmo de reconocimiento, y finalmente la identificación de imágenes crudas en tiempo real.

Capítulo 5.- Resultados. En este capítulo mostraremos todos los resultados obtenidos del experimento, así como las medidas de verosimilitud para medir la validez del sistema.

Capítulo 6.- Conclusiones. En este capítulo se resume la problemática, los obstáculos encontrados a lo largo del proyecto y finalmente el trabajo futuro a realizar.

Capítulo 2

Estado del arte

De acuerdo a Mitra [25] el reconocimiento gesticular es una tarea compleja que podemos dividir en dos categorías principales y son el reconocimiento de señas estáticas y el reconocimiento de señas dinámicas.

En el trabajo de Aditha et al. [2], los autores realizan un sistema de reconocimiento de la Lengua de Señas Hindú basado en imágenes sin usar marcadores. Ellos resolvieron el reconocimiento utilizando una cámara de video común. No utilizaron ningún tipo de marcadores o guantes para las manos, y obtuvieron imágenes RGB[1] por separado de las señas estáticas. Las imágenes capturadas en formato RGB fueron transformadas a un formato YCbCr[2] como se muestra en la ecuación 2.1.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cr &= 128 + 0.5R - 0.418G - 0.081B \\ Cb &= 128 - 0.168R - 0.331G + 0.5B \end{aligned} \tag{2.1}$$

Después de la conversión a YCbCr, se transformó la imagen a una escala binaria. Seguidamente, se realizó una transformación de distancia, que genera una imagen en escala de grises a partir de la imagen binaria previa, utilizando la distancia euclidiana entre dos pixeles, como se muestra en la ecuación 2.2.

$$d_e(P;Q) = \sqrt{(x-u)^2 + (y-v)^2} \tag{2.2}$$

[1]En inglés Red, Green, Blue, y en español rojo, verde y azul. Es la composición del color en términos de la intensidad de los colores primarios de la luz.

[2]Y representa la componente de luminosidad y las señales Cb y Cr son los componentes de crominancia, la diferencia de azul y diferencia de rojo, respectivamente.

En donde P, es el primer punto con coordenadas en $(x,\ y)$; el segundo punto Q tiene sus coordenadas en $(u,\ v)$, y la distancia euclidiana de $(P;Q)$ es la raíz cuadrada de la suma de las diferencia entre las componentes de ambos puntos. Después, obtuvieron los coeficientes de la proyección de la transformada de distancia de la imagen en escala de grises, y de esto último obtuvieron un vector fila (R) y un vector columna (C) que representan únicamente la forma de la mano de la imagen de entrada. Después, para eliminar el ruido restante aplicaron el método de los descriptores de Fourier, descrito en la ecuación 2.3:

$$\begin{gathered} U_n = \frac{1}{N}\sum_{t=0}^{N-1} R(t)exp(\frac{-j2\Pi nt}{N}),\ donde\ n\ =\ 0,1,2,3...N-1 \\ -j\ es\ un\ valor\ imaginario \\ V_n = \frac{1}{N}\sum_{t=0}^{N-1} C(t)exp(\frac{-j2\Pi nt}{N}),\ donde\ n\ =\ 0,1,2,3...N-1 \\ N\ es\ la\ cantidad\ de\ elementos\ del\ vector\ R\ y\ del\ vector\ C \end{gathered} \tag{2.3}$$

El paso anterior genera un vector de características normalizado por cada seña. Este vector está formado por los coeficientes de Fourier U_n y V_n que se obtuvieron de los descriptores que se muestran en la ecuación 2.3, y en el que cada elemento $x_i \in X$ es dividido entre la primera componente, también llamada componente DC. Esta componente existe en cada uno de los vectores de coeficientes de Fourier es decir existe una componente DC para el vector de coeficientes U_n y una componente DC para el vector de coeficientes V_n y en ambos casos es la primera posición del vector ordenado de manera descendente. Cada elemento es una muestra en dominio de la frecuencia, la cual representa la forma de la mano. De este par de vectores se extraen características especiales llamadas $Momentos\ Centrales$, y son un conjunto de valores que caracterizan una probabilidad de distribución. Existen cuatro de estos momentos centrales y su explicación es la siguiente, el momento central de orden cero μ_0 tiene un valor igual a 1, el primer momento central μ_1 tiene un valor igual a 0. El segundo momento central, también llamado varianza, es usualmente denotado por σ^2, en donde σ, representa la desviación estándar de una distribución. Este momento lo vemos calculado en la ecuación 2.4.

$$\begin{gathered} \mu_2 = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2,\ donde\ x_i, \\ es\ la\ muestra\ con\ valor\ i\ =\ 1,2,3...\ N \end{gathered} \tag{2.4}$$

El tercer momento central μ_3 es la oblicuidad, también es una medida de asimetría de la probabilidad de distribución de una variable con un valor real aleatorio, que puede tener un valor positivo o negativo. La fórmula de la oblicuidad se muestra en la ecuación 2.5.

$$\mu_3 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^3, \ donde\ x_i, \\ es\ la\ muestra\ con\ valor\ i\ =\ 1, 2, 3...\ N \tag{2.5}$$

El cuarto momento central μ_4 es la curtosis, y también es una medida de puntiagonalidad o "*Peakedness*" de una probabilidad de distribución. Es una característica de distribución de un conjunto de valores de datos con tamaño finito N. La fórmula de la curtosis se muestra en la ecuación 2.6.

$$\mu_4 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^4, \ donde\ x_i, \\ es\ la\ muestra\ con\ valor\ i\ =\ 1, 2, 3...\ N \tag{2.6}$$

Todos estos son los que valores forman el vector de entrada para el reconocimiento de los signos. Finalmente, la clasificación utiliza una red neuronal que es alimentada con este vector de características y se entrena con 360 imágenes de un conjunto de entrenamiento, y es probada con 180 imágenes de un conjunto de prueba, arrojando una precisión arriba del 90 %.

Yi Li [21] propone en su trabajo reconocer cuáles dedos están levantados para reconocer la seña hecha con la mano, como se observa en la figura 2.1. Yi Li resolvió el problema, primero mediante el algoritmo de k-medias para saber si hay más de una mano haciendo un símbolo, si es así, entonces sólo se trabaja con una sola mano. Después el algoritmo de "Graham scan"[3] es usado para obtener la forma convexa de la mano; luego con el algoritmo de tres puntos se obtienen los bordes y el alineamiento de los dedos; seguidamente con el cálculo de ángulos entre las falanges extendidas se le pone nombre a los dedos.

Una vez que los dedos son identificados con sus nombres, se almacenan los nombres de los dedos, y la correspondencia de sus vectores. El ángulo

[3]El método de Graham (Graham scan) es un método de cálculo computacional de la envolvente convexa de un grupo finito de puntos en el plano de complejidad $O(n\ log\ n)$.

A, que es la distancia angular que separa a los dedos, que se forma entre los dos vectores $V_1(x_1, y_1)$ y $V_2(x_2, y_2)$ es calculado como se muestra en la ecuación 2.7.

$$A = \frac{\overrightarrow{V_1} \cdot \overrightarrow{V_2}}{||\overrightarrow{V_1}||\ ||\overrightarrow{V_1}||} \tag{2.7}$$

Finalmente, los momentos centrales, junto con el valor angular entre los dedos son los datos que ingresan a un clasificador de tres capas, el cual, primero cuenta los dedos, y después los identifica y etiqueta con su nombre. Si estos datos son suficientes para reconocer la seña, el reconocimiento acaba ahí; si no, se pasa a un tercer nivel en el clasificador, en el cual mediante una comparación de ángulos, nos permite etiquetar el símbolo. En su trabajo Yi Li reporta un reconocimiento de al menos un 80 % en señas para una sola mano; y para la misma seña pero utilizando ambas manos, el reconocimiento se aumenta hasta un 90 %.

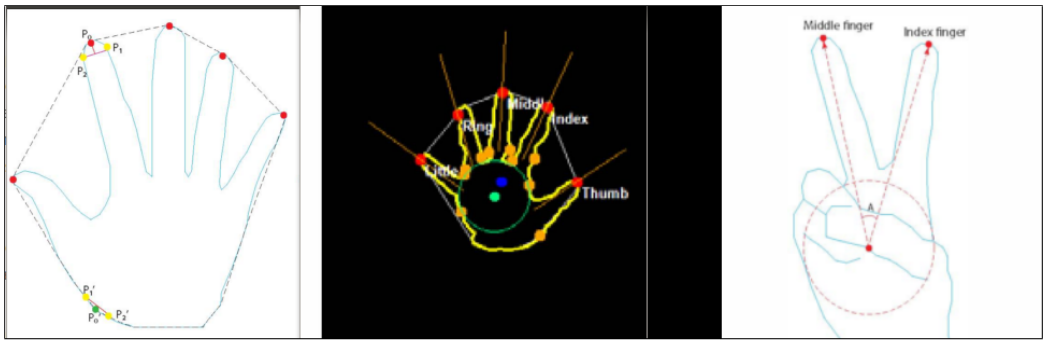


Figura 2.1: Proceso de reconocimiento de la Lengua de Señas China seguido por Yi Li [21].

Por otro lado, Chikkanna [3], busca en su trabajo detectar gestos móviles o "Spotting". En su trabajo propone la localización y seguimiento de las manos. Como primer paso, consigue la localización y seguimiento obteniendo un punto central de referencia, el cual rastrea en todo momento utilizando la fórmula del centroide mostrada en la ecuación 2.8.

$$(X_c, Y_c) = (\frac{x_1 + x_2}{2}; \frac{y_1 + y_2}{2}) \tag{2.8}$$

En donde X_c, Y_c es el punto central de la mano, y se calcula usando (x_1, y_1), que se ubican en la muñeca de la mano, y otro punto (x_2, y_2) que es

el punto final hacia lo alto de la mano.

En el siguiente paso, después de este seguimiento, se obtiene una ruta de la seña, la cual es una consecución de puntos almacenados que forma una curva, como se muestra en la figura 2.2.

Figura 2.2: Ruta no suavizada que realiza el gesto móvil [21].

A esta ruta se le hace un suavizado, como se muestra en la figura 2.3, esto para prevenir cambios inesperados dentro de la trayectoria. Este suavizado se muestra en la ecuación 2.9.

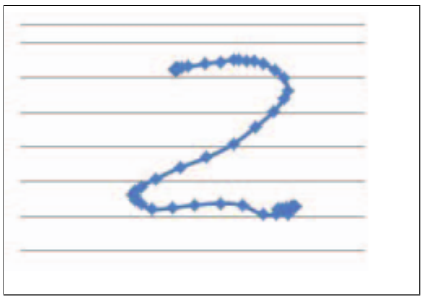


Figura 2.3: Ruta suavizada que realiza el gesto móvil [21].

$$(x_s, y_s) = (\frac{x_{t-1} + x_t + x_{t+1}}{3}; \frac{y_{t-1} + y_t + y_{t+1}}{3}) \tag{2.9}$$

En donde (x_s, y_s) es el punto suavizado, (x_{t-1}, y_{t-1}) es el punto previo, (x_t, y_t) es el punto actual, (x_{t+1}, y_{t+1}) es el punto siguiente.

A continuación, se hace una extracción de características como la posición dada por la ecuación 2.10.

$$length = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{2.10}$$

En donde (x_1, y_1), es el punto actual, y (x_2, y_2) es el punto anterior

La característica de la orientación de acuerdo al centroide de la ruta (θ_1), la orientación entre dos puntos consecutivos (θ_2), y la orientación entre un punto inicial y el punto actual de la seña (θ_3), se calculan usando la ecuación 2.11.

$$\theta_t = tan^{-1}(\frac{y_2 - y_1}{x_2 - x_1}) \tag{2.11}$$

Y la tercera característica básica, es la velocidad (V_t), la cual juega un importante papel durante la fase de reconocimiento de la seña. La velocidad es calculada utilizando la ecuación 2.12.

$$V_t = \sqrt{(\frac{x_t - x_{t-1}}{t})^2 + \frac{y_t - y_{t-1}}{t})^2)} \tag{2.12}$$

En donde x_t, es la posición actual del gesto, x_{t-1}, es la posición anterior del gesto, y t es el tiempo total del recorrido de la ruta de la mano al realizar el gesto.

El cuerpo se divide en zonas llamadas momentos, en los cuales se determina dónde comienza y termina un gesto móvil como se observa en la figura 2.4. Este momento del gesto es la cuarta característica necesaria para el reconocimiento. Todas estas características, es decir la ruta realizada por la mano, la orientación, la velocidad junto con momento del gesto sirven de entrada para que el clasificador, el cual se basa en HCRF[4], realice de manera correcta la tarea del reconocimiento del gesto móvil.

[4]En inglés Hidden Conditional Random Fields y Campos Ocultos Condicionales Aleatorios en español

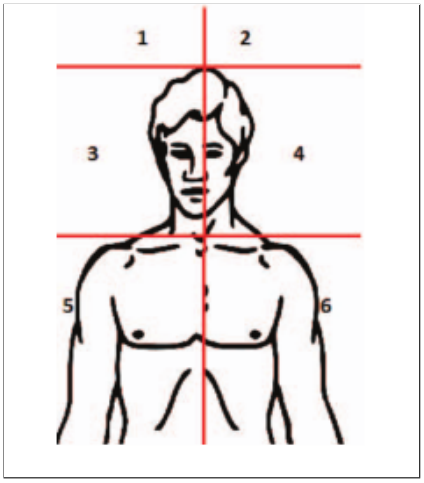


Figura 2.4: División del cuerpo en momentos [3].

Cada gesto dispone de 65 imágenes, 40 para el entrenamiento y 25 para la prueba. Los resultados arrojan un 95 % de precisión al clasificar el conjunto test, y un 90 % de precisión clasificando en tiempo real.

En el trabajo de Samirelons et al. [29] se busca reconocer la forma de la mano para etiquetarla y clasificarla como una seña, pero en este reconocimiento proponen hacerlo sin pasar por la etapa de filtrado de imágenes. La propuesta de solución comienza con la adquisición de imágenes, a las cuales se les procesa para obtener una firma (*signature*). Una vez extraídas las firmas, estas se procesan en una versión modificada de una PCNN[5], que trabaja con dos entradas como se muestra en la figura 2.5.

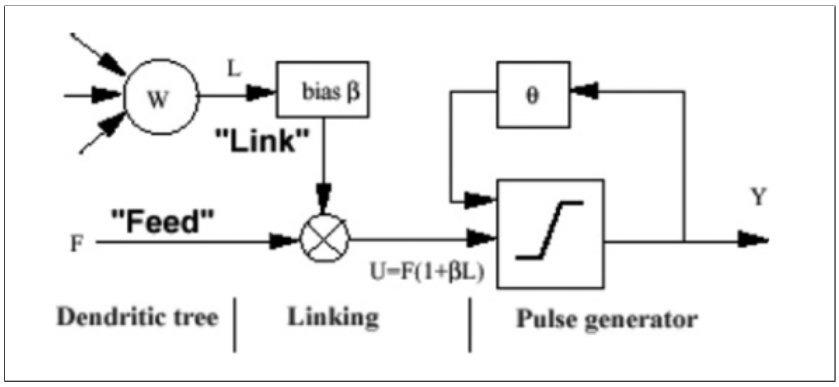


Figura 2.5: Diagrama de una red modificada PCNN [29].

[5]En inglés Pulse Coupled Neural Network, Red Neuronal de Pulso Conjunto en español

La primera entrada, la entrada $feeded$ o de alimentación, se alimenta de estímulos externos, los cuales dependen de la intensidad del pixel; la entrada *linking* o de enlace depende de los estímulos internos que son las relaciones con las neuronas con radio de alimentación cercanos a ella. Cada neurona en la PCNN, puede ser descrita como se observa en las ecuaciones 2.13 y 2.14.

$$L(i) = L(i-1)e(\alpha L) + VL(R * Ysur(i-1)) \tag{2.13}$$

$$F(i) = S + F(i-1)e(\alpha F) + VF(R * Ysur(i-1)) \tag{2.14}$$

En donde $L(i)$ es la entrada *linking* potencial, $F(i)$ es la entrada $feeding$ potencial, y S representa la intensidad de un elemento pixel de la imagen dada.

$$U(i) = F(i)[1 + \beta L(i)] \tag{2.15}$$

$$\begin{aligned} \theta(i) &= \theta(i-1)e - (\alpha q) + V\theta Yout(i-1) \\ &\quad donde: \\ U(i) > \theta &=> Yout = 1(Regla\ de\ disparo) \\ de\ otra\ manera &=> Yout = 0 \end{aligned} \tag{2.16}$$

$U(i)$ como se muestra en 2.15, es el potencial de activación de una neurona. $\theta(i)$, que se muestra en 2.16, es el potencial umbral de una neurona e (i) es el paso de la iteración actual. Los parámetros (αL), (αF) y (αq) son coeficientes que decaen, (β) es el coeficiente *linking* y los parámetros (VL) y (VF) son coeficientes de *linking* y el umbral potencial.

Como primer paso para clasificar cada gesto, se genera una firma de la imagen con la menor cantidad de características para mejorar el reconocimiento en imágenes grandes. Una vez obtenida la firma, en el paso dos se le aplica la DFT[6], para obtener los máximos coeficientes (K) que son los descriptores de rotación, traslación, y escala invariante conocidos como descriptores de Fourier. Finalmente, estas características que se obtienen a través

[6]En inglés Discrete Fourier Transformation, en español Transformada Discreta de Fourier.

de los descriptores de Fourier, sirven de entrada para el clasificador MLP[7], de 11 neuronas de entrada, 18 neuronas ocultas y un número n de neuronas finales, entrenadas por cuando menos 12 horas. Este método neutraliza las condiciones de luz en las que fue tomada la imagen de la seña y arrojan una precisión de reconocimiento de un 90 %.

En el trabajo de Mekala et al. [23] se busca reconocer las señas de las manos mediante un método basado en procesamiento de imágenes que ubica, centra, y la sigue la mano. En su propuesta lidian con el problema de la oclusión, el comienzo y final de un símbolo, y el posicionamiento de la cámara. En su propuesta capturan una secuencia de video con una cámara convencional, la posición inicial de la captura es manual. Después, se hace un preprocesamiento del video sin editar. El primer paso que usan es aplicar un filtro de medianas que se encarga de remover el ruido no deseado de la imagen, el segundo paso es la eliminación del fondo de la imagen dejando únicamente la mano, esto último mediante un filtro gausiano. El tercer paso que aplican es la extracción de características. De la mano sacan dos puntos de interés POI's[8]. Así obtienen de la mano cinco puntas de los dedos, y un centro de la mano TP[9], como se observa en la figura 2.6. Extraen después un vector de 55 características, que se componen de cinco puntas de dedos, cuatro elementos del vector de movimiento, seis elementos de la secuencia y los 40 restantes son elementos de la transformada de Wavelet.

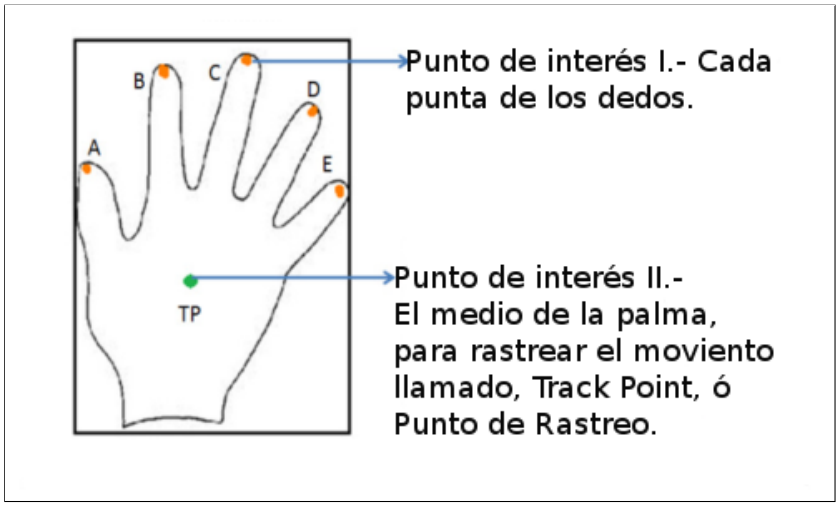


Figura 2.6: Puntos de rastreo de la mano [23].

[7]en inglés Multi Layer Perceptron, en español Perceptrón Multicapa
[8]En inglés Point Of Interest, en español Puntos de Interés.
[9]En inglés Track Pointen español Punto de Rastreo.

De ahí toman los descriptores de Fourier, como se muestra en la ecuación 2.17.

$$z(k) = \sum_{k=0}^{n-1} p(k) exp(j\frac{2\pi lk}{n}) \quad (l = 0, 1, ...n-1) \tag{2.17}$$

Donde, z es la transformada de *Fourier* de p, y también es la expresión de una secuencia de puntos en el dominio de la frecuencia, y que representan la curva del objeto. Después mediante una CNN, en inglés Combinational Neural Network, en español Red Neuronal Combinacional, reducen el espacio de búsqueda. El bloque de entrenamiento es de 26 letras que no requieren vector de movimiento y la letra Z y la J que sí requieren del vector de movimiento. En conclusión, el algoritmo de Mekala et al, provee una precisión de arriba del 90 % y una ventaja de alta velocidad de procesamiento que permite clasificaciones en tiempo real. Las etapas del proceso de Mekala se pueden observar en la figura 2.7.

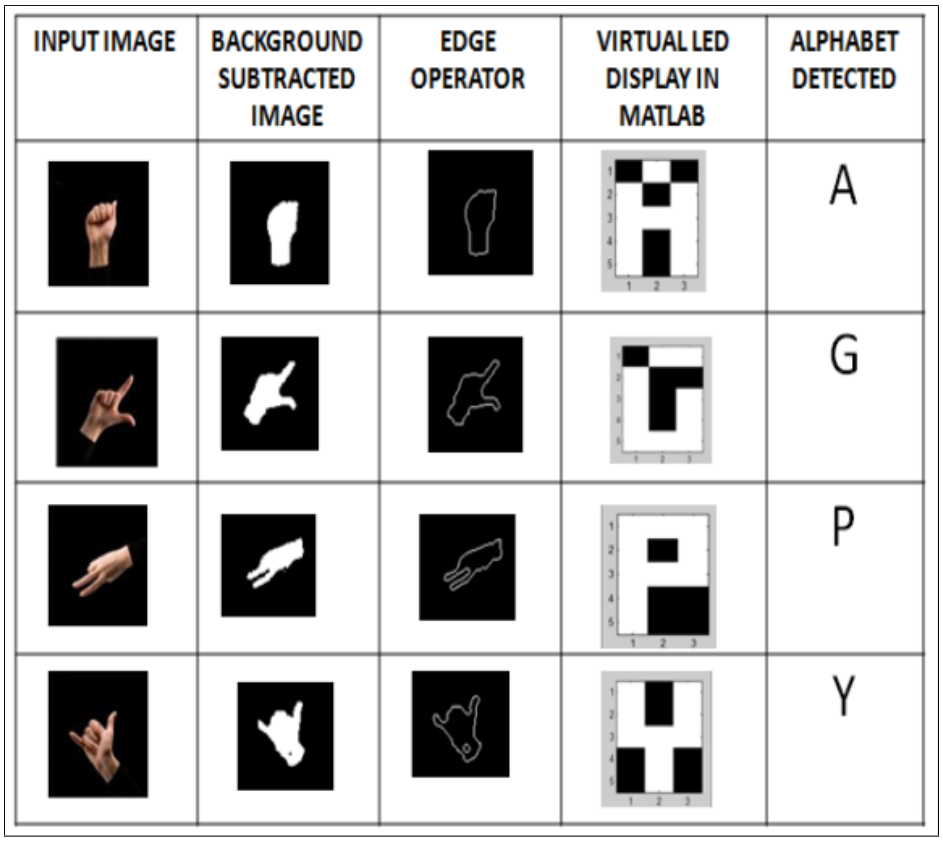


Figura 2.7: Etapas del proceso de identificación, de gestos de la Lengua de Señas Indú, seguido por Mekala [23].

Capítulo 3

Marco teórico

En este capítulo se describen las diferentes herramientas utilizadas en la arquitectura del sistema de reconocimiento de señas, así como las técnicas de procesamiento de imágenes, de aprendizaje automático y de optimización empleadas.

3.1. Sensor de profundidad

El sensor de profundidad de Microsoft, también llamado Kinect, fue desarrollado para controlar e interactuar con la consola de videojuegos Xbox 360 sin necesidad de tener contacto físico con un control de videojuegos tradicional. Funciona mediante un sensor de profundidad, un micrófono de múltiples matrices y un procesador personalizado que ejecuta el software patentado que proporciona captura de movimiento de todo el cuerpo en 3D, reconocimiento facial y capacidades de reconocimiento de voz. Podemos observar una imagen del dispositivo en la figura 3.1.

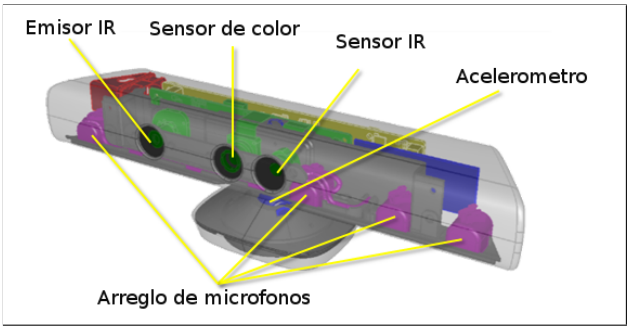


Figura 3.1: Sensor Kinect de Microsoft [1].

Lo importante de este sensor son las características en su sistema de cámaras y las dimensiones de las imágenes capturadas que son de dos tipos 320 × 240 pixeles con 16 bits de profundidad a 30 fps y 640 × 480 pixeles con 32 bits de color a 30 fps, un campo de visión horizontal a 57 grados, un campo de visión vertical a 43 grados, un rango de inclinación física a ± 27 grados, y un rango de de profundidad del sensor de uno a cinco metros.

El conjunto de bibliotecas para el control del sensor Kinect es el The Kinect for Windows Software Development Kit (SDK), en su versión 1.8, compilado y ejecutado en el IDE Microsoft Visual Studio, en su versión 2010 SP2 con licencia académica. El Kinect SDK es completamente gratuito para desarrollo y se puede descargar desde la pagina oficial de Microsoft[1].

3.2. Morfología matemática

En los últimos años, la Morfología Matemática (MM) ha tenido una gran aplicación en el tratamiento de imágenes digitales cuyas formas irregulares impiden descripción alguna por las formas geométricas existentes. Esto hace que los cálculos de la geometría analítica sean dificultosos y es entonces donde entra la MM y el Procesamiento Digital de Imágenes (PDI). Si consideramos la imagen como una matriz de puntos entonces podemos realizar mediciones de áreas, perímetros y fórmulas derivadas con gran precisión utilizando matemáticas discretas [27].

La MM es una teoría basada en conceptos de geometría, álgebra, y teoría de conjuntos. Su aspecto clave reside en el Elemento Estructurador (EE) un conjunto completamente conocido y de geometría definida. La MM tiene cuatro operaciones básicas, Dilatación, Erosión, Apertura, y Cierre. Cada operación tiene su respectivo trato en imágenes binarias y en escala de grises [38].

La MM, también se refiere al estudio de las figuras y ha sido utilizada para interpretar la estructura o forma de objetos en imágenes. Una imagen puede considerarse formada por conjuntos (regiones) de pixeles, como se observa en la figura 3.2, y por tanto, pueden aplicarse herramientas de la Teoría de Conjuntos.

[1] http://www.microsoft.com/en-us/download/details.aspx?id=40278

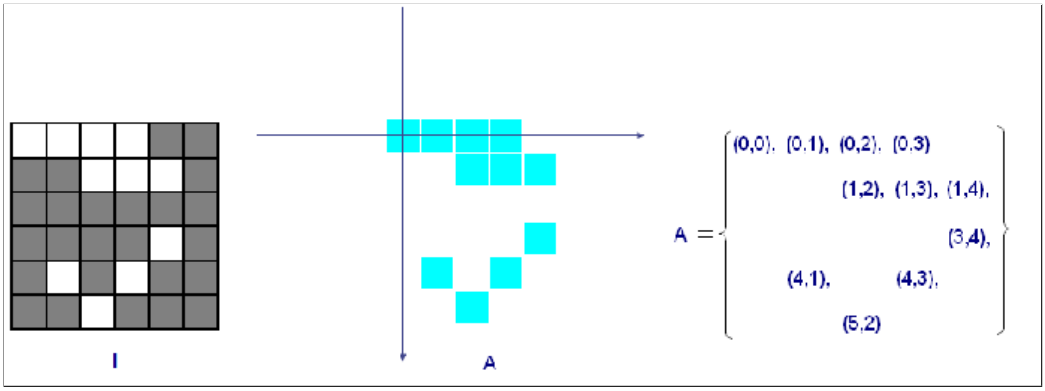


Figura 3.2: Imágenes representadas como conjuntos de pixeles.

Los operadores se crean al hacer interactuar la imagen con el elemento estructurador. Pueden aplicarse diferentes operadores para analizar las propiedades morfológicas de las regiones. Al igual que la imagen, el elemento estructurador tiene origen, y longitudes conocidas como se muestra en la figura 3.3:

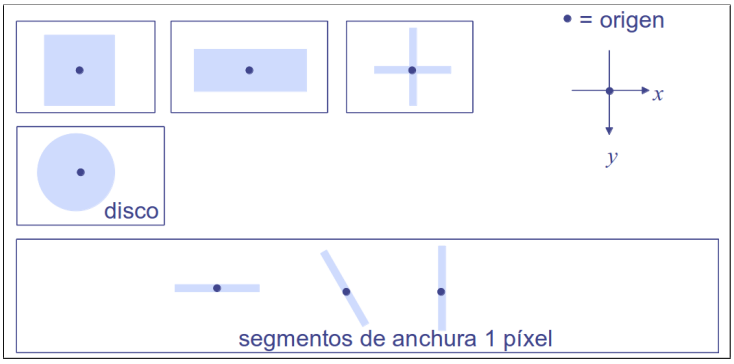


Figura 3.3: Elemento estructurador.

Por citar algunos ejemplos, las operaciones de morfología matemática son muy utilizadas en imágenes binarias para la eliminación de regiones pequeñas que muchas veces son originadas por el ruido de pixeles difusos, el relleno de pequeños agujeros en regiones, la extracción de determinados rasgos de la imagen, y para la descomposición de figuras con formas complejas en sus partes más significativas, eliminado aquellas no relevantes.

Sean f y g dos imágenes en niveles de gris, con dominios D_f y D_g, respectivamente. La operación de erosión tiene como finalidad remover los pixeles excedentes en los bordes exteriores y filtra hacia el interior de la imagen [35]. La erosión de la imagen f, por el elemento estructurador g, denotada por $\epsilon_g(f)$ se define como muestra la ecuación 3.1:

$$\epsilon_g(f_{s,t}) = max_{(s+x,t+y)\in D_f;(x,y)\in D_g} f(s+x,t+y) - g(x,y) \qquad (3.1)$$

La operación de erosión también se emplea para separar regiones débilmente unidas o para eliminar pequeños detalles. Tras la erosión quedan únicamente las formas de mayor tamaño, como se observa en la figura 3.4.

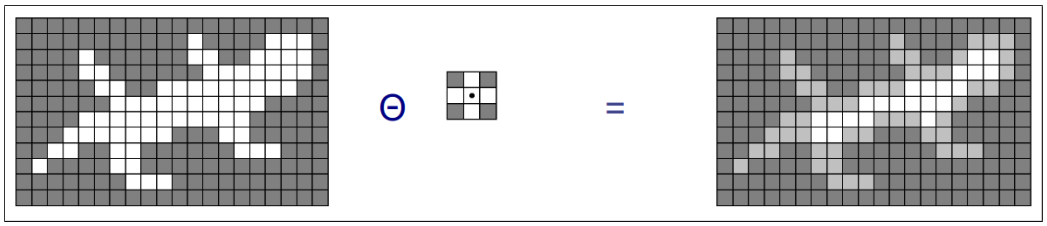


Figura 3.4: Ejemplo de una imagen binaria erosionada.

La operación de dilatación se encarga de filtrar hacia el exterior de la imagen [35]. La dilatación de la imagen f por el elemento estructurador g, denotado por $\delta_g(f)$ se define en la ecuación 3.2:

$$\delta_g(f_{s,t}) = max_{(s-x,t-y)\in D_f;(x,y)\in D_g} f(s-x,t-y) + g(x,y) \qquad (3.2)$$

La operación de dilatación también resulta muy útil para rellenar agujeros o para unir regiones próximas que en la imagen se han podido separar por una deficiente binarización, como se observa en la figura 3.5.

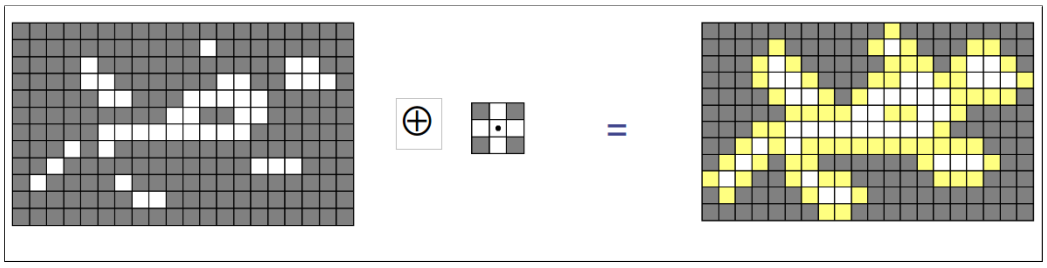


Figura 3.5: Ejemplo de una imagen binaria dilatada.

Las operaciones de erosión y dilatación pueden utilizarse combinadamente para llevar a cabo operaciones de filtrado. La dilatación, seguida de erosión, se utiliza para rellenar agujeros espurios, por eso se le llama cierre, y la erosión, seguida de dilatación, se emplea para eliminar pixeles aislados y desconectar regiones débilmente unidas.

La operación de apertura es útil para eliminar detalles luminosos y pequeños en relación al elemento estructurador, quedando el resto de la imagen prácticamente sin modificaciones. La apertura es una erosión seguida de una dilatación con el mismo elemento estructurador. La apertura morfológica de una imagen f por el elemento estructurador g es definida por la ecuación 3.3.

$$\gamma_g(f_{s,t}) = \delta_g(\epsilon_g(f)) \tag{3.3}$$

La operación de cierre morfológico es útil para eliminar detalles obscuros y pequeños en relación al elemento estructurador. El cierre es una dilatación seguida de una erosión con el mismo elemento estructurador. El cierre morfológico de una imagen en niveles de gris f por el elemento estructurador g es definido por la ecuación 3.4.

$$\phi_g(f_{s,t}) = \epsilon_g(\delta_g(f)) \tag{3.4}$$

Aunque la Morfología Matemática originalmente fue concebida para imágenes binarias, también puede usarse para imágenes en RGB e imágenes en escala de grises [31].

3.3. Imagen integral

El conjunto de características que servirán de entrada para nuestro análisis (características Haar), son áreas rectangulares sobre ciertas zonas de la imagen que deseamos analizar, y que son rápidamente computadas usando una representación intermedia de la imagen la cual llamaremos imagen integral. La imagen integral (II) en la locación (x, y) contiene la suma inclusiva de todos los pixeles de izquierda a derecha, y de arriba hacia abajo hasta el punto (x, y) actual, como se muestra en la ecuación 3.5.

$$ii(x, y) = \sum_{x'<=x, y'<=y} i(x', y') \tag{3.5}$$

En donde $ii(x, y)$ representa la imagen integral e $i(x', y')$ el valor de la intensidad del pixel (x', y') en la imagen original. Una vez calculada la imagen integral esta puede ser evaluada a cualquier escala o ubicación en tiempo constante.

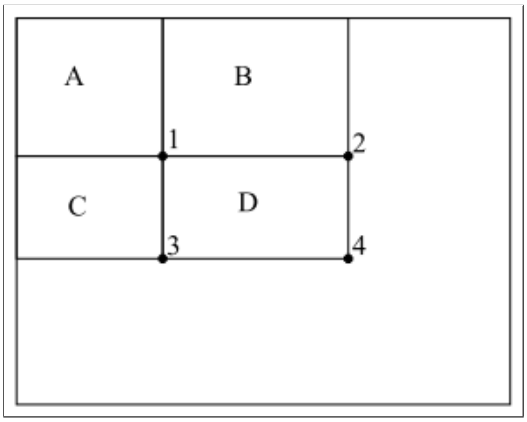


Figura 3.6: La suma de los pixeles dentro de un rectángulo D computada únicamente con cuatro puntos [33].

Usando la imagen integral, cualquier suma rectangular puede ser computada utilizando cuatro puntos de la II, como se muestra en la figura 3.6. El valor de la imagen integral en la posición 1 es la suma de los pixeles en el rectángulo A. El valor en la posición 2 es $A + B$, en la posición 3 es $A + C$ y en la posición 4 es $A + B + C + D$. Y la suma dentro de D puede ser computada como $4 + 1 - (2 + 3)$.

3.4. Características Haar 2D

El uso del conjunto de funciones base Haar Wavelet ha sido una de las propuestas más extendidas y eficaces para extraer la información de la textura que describe una clase de objetos [26]. Las características Haar han sido empleadas en los últimos años para la detección de rostros [6, 14, 12, 24], para la detección y clasificación de especies de insectos [18], y tambien algunas de sus variantes poligonales para detección de matriculas de autos, y rocas de la superficie marciana [28], entre otros. Esta representación codifica en una imagen las diferencias de las intensidades promedio entre dos regiones rectangulares adyacentes, capturando las similitudes estructurales entre las instancias de la clase de objetos. Motivados por esta idea, Viola & Jones [34] adoptaron el uso de las Haar Wavelet y desarrollaron las llamadas características Haar. Una característica Haar considera regiones rectangulares adyacentes en un lugar específico dentro de una ventana de detección en una imagen, suma las intensidades de los pixeles en estas regiones y calcula la diferencia entre ellas. Luego, esta diferencia se utiliza para clasificar las subregiones candidatas en una imagen. Dentro de cualquier ventana de detección el número total de características Haar (conjunto exhaustivo) es

muy grande, mucho mayor que el número de pixeles. Con el fin de garantizar una clasificación rápida, el proceso de aprendizaje debe excluir a una gran mayoría de estas características y centrarse en un conjunto pequeño de características críticas o más significativas para la clasificación.

En la figura 3.7 se muestra el conjunto base de características Haar en dos dimensiones (2D), formado por tres tipos de estas características, utilizando dos rectángulos, tres rectángulos, y cuatro rectángulos. Para el caso de dos rectángulos, el valor representa la diferencia entre la suma de los pixeles dentro de las dos regiones rectangulares que tiene el mismo tamaño y forma y pueden estar horizontal o verticalmente adyacentes. Las características de tres rectángulos calculan la suma dentro de los dos rectángulos exteriores y a este valor se le resta la suma en el rectángulo central. Por último, las características de cuatro rectángulos calculan la diferencia entre pares diagonales de rectángulos.

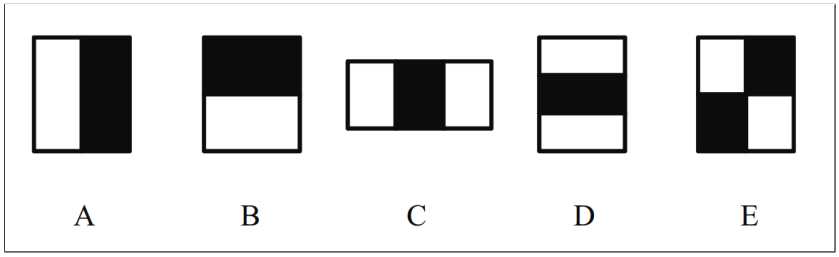


Figura 3.7: Características Haar: a) vertical de dos barras, b) horizontal de dos barras, c) vertical de tres barras, d) horizontal de tres barras, e) diagonal de cuatro barras.

Para calcular rápidamente y a varias escalas las características Haar se introduce la representación de la imagen conocida como imagen integral (ver sección 3.3). La imagen integral puede calcularse a partir de una imagen con pocas operaciones por pixel. La imagen integral en el punto (x, y) se define como la suma de los valores de los pixeles que se encuentran por encima y a la izquierda de dicho punto en la imagen original[34], como se observó previamente en la ecuación 3.5.

Dado que el conjunto exhaustivo de características Haar en cada ventana de búsqueda en la imagen es muy amplio, el método propuesto por Viola & Jones, se utiliza el algoritmo AdaBoost[7], para seleccionar un pequeño número de características distintivas de todas estas posibles combinaciones.

3.5. Características Haar 3D

Las características Haar en tres dimensiones (3D) son características extraídas de un volumen en el orden del tiempo y del espacio. Pueden ser vistas como filtros cúbicos con volumen. Un ejemplo de estas características tridimensionales puede ser observado en la figura 3.8

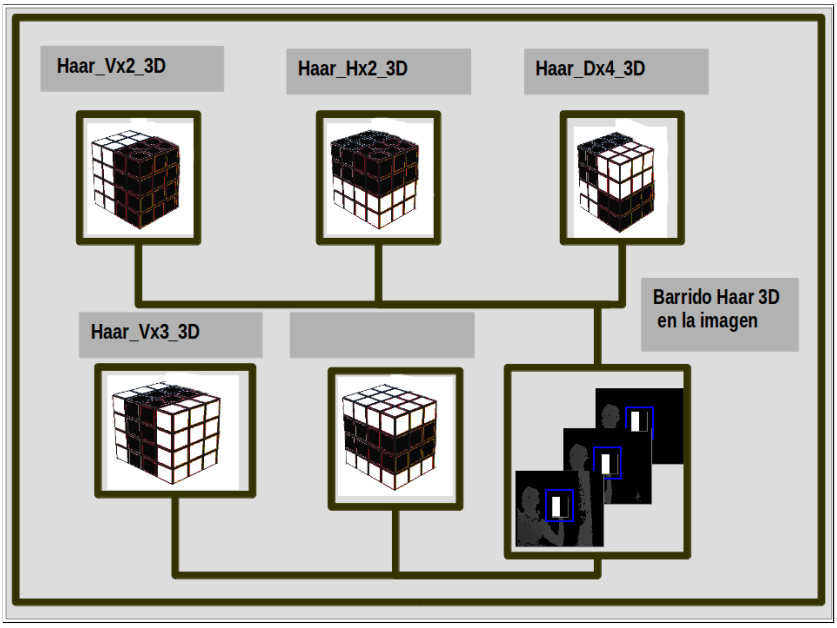


Figura 3.8: Formas básicas de características Haar en 3D.

A través de estas características se puede obtener más información de múltiples imágenes, y aunque es posible tomar un número N de cuadros tridimensionales, muchos de ellos reportarían información redundante y el costo en tiempo computacional se incrementa mientras mayor es la cantidad de imágenes tomadas en un mismo tiempo. Un ejemplo práctico de su uso lo presentan Cui et al. [4] en su trabajo donde necesitan identificar la existencia de peatones en una ciudad transitada. Cui et al., toman diversos cuadros extraídos de una secuencia de video, y aplican las características Haar en 3D para obtener información acerca del movimiento y la posible trayectoria de un peatón, suponiendo que exista alguno. También se ha utilizado en para calcular volúmenes de modelos CAD tridimensionales generados por computadora como lo muestra el trabajo de Yamamoto et al. [37]. Y para generar reconstrucciones tridimensionales utilizando segmentos de video de múltiples cámaras para recrear un cascaron volumétrico de una persona [5, 8] entre otros.

3.6. Ada Boost

El método de aprendizaje automático supervisado *Ada Boost*, junto con las características *Haar*, son comúnmente usados debido a su bajo costo computacional, por ejemplo, en tarea de identificación de intrusos en redes como lo presenta el trabajo de Weiming Hu et al. [11] donde usan como características para cada conexión el protocolo de transmisión de datos, el contenido relacionado con la conexión, y el tráfico entre ambas computadoras muestreado cada dos segundos, y en monitoreo de anomalías de red en general [20, 39]. En la detección de tumores mamarios en las mastografías [32, 13, 30]. Por último ejemplo citaremos el trabajo de Xiaofei Ji et al. [16], en donde los autores buscan reconocer las acciones humanas usando características espacio-temporales como descriptores de movimiento y para detectar peatones en un ambiente transitado [19].

AdaBoost logra entrenar conjuntos de clasificadores débiles que al ser conectados de forma consecutiva, a manera de cascada, son capaces de detectar características y patrones en las imágenes de objetos particulares [17].

Cada uno de esos clasificadores débiles consta de dos o tres rectángulos calculados a partir de una imagen integral de valores $I(x, y)$ como se observa en la figura 3.9. Como podemos observar a manera de ejemplo, el conjunto de rectángulos asignados a un clasificador débil extraído en la imagen, no precisamente tiene que ser el mismo para el segundo clasificador débil.

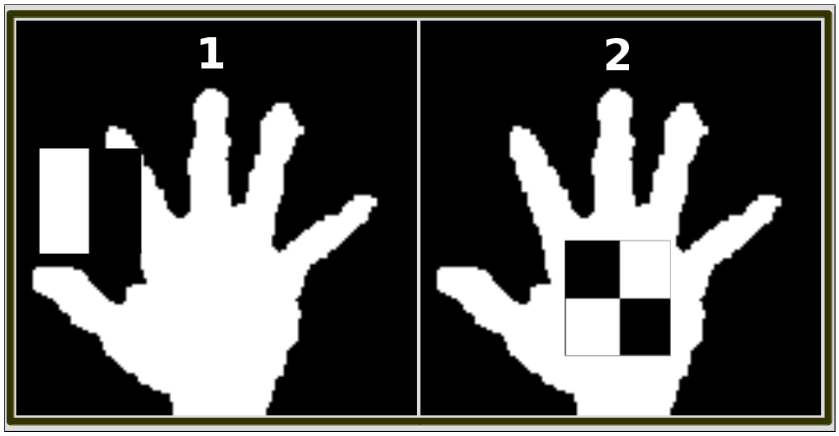


Figura 3.9: De izquierda a derecha, ejemplos de un primer y segundo clasificador debil.

Calcular cada rectángulo R, que forma parte de un clasificador débil, es obtenido con la ecuación 3.6, donde $I'(x_0, y_0)$, $I'(x_0, y_1)$, $I'(x_1, y_0)$ y $\mathrm{I}'(x_1, y_1)$,

son valores de la imagen integral correspondientes a las esquinas del rectángulo.

$$R = I'(x_1, y_1) + I'(x_0, y_0) – I'(x_0, y_1) - I'(x_1, y_0) \tag{3.6}$$

donde $x_0 < x_1;\ y_0 < y_1$.

Para usos prácticos, en sistemas de reconocimiento de objetos en tiempo real es indispensable la inmediatez de los métodos elegidos para analizar e interpretar características [9]. El esquema propuesto por Viola y Jones [33], es un método idóneo para realizar tareas de identificación y etiquetado de clases, porque se adapta suficientemente bien a cualquier sistema computacional, inclusive en aquellos sistemas embebidos, donde la capacidad y los recursos de computo son limitados, realizando de manera satisfactoria tareas de reconocimiento y etiquetado de clases [10].

El pseudocódigo del algoritmo AdaBoost lo podemos ver a continuación en el algoritmo 1.

Algorithm 1 Ada Boost

Require: $(x_1, y_1), ..., (x_m, y_m)$ donde $x_i \in X, y_i \in \{-1, +1\}$.

for all $t \in T$ **do**
  for $i = 1; i <= M; i++$ **do**
    $D_i = 1/m$
  end for
  Evaluar cada clasificador débil $h_t(x)$
  Hasta encontrar el criterio h_t con el menor error ϵ_t
  Donde $\epsilon_t = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
  Establezca un $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  for $i = 1; i \leq M; i++$ **do**
    $D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(xi))}{Z_t}$
    donde Z_t es un factor de normalización, tal que $\sum_{i=1}^{M} D_{t+1}(i) = 1$
  end for
end for
Salida de la hipótesis final:
$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$
return $H(x)$

Para entrenar con el algoritmo AdaBoost usando una distribución de probabilidad D_t y obtener un clasificador débil h_t, normalmente se muestrean los ejemplos de entrenamiento usando D_t (muestreo por importancia) y de todos las hipótesis o clasificadores h_t escogemos el clasificador cuyo error ϵ_t sea mínimo.

Para cada clasificador débil $h_t \in T$, en cada iteración obtenemos un valor de confianza o importancia de opinión α_t. Esta importancia de opinión α_t incrementa de manera proporcional mientras menos sean los errores de clasificación del clasificador débil h_t. También por cada iteración obtenemos un porcentaje de error ϵ_t, que es el error asociado a h_t y viene dado por la ecuación 3.7, el cual aumenta de manera proporcional mientras mayores sean los errores de clasificación de nuestro clasificador débil h_t.

$$\epsilon_t = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)] \tag{3.7}$$

Es decir, sumar el peso del ejemplo, cada vez que la hipótesis actual falle al predecir la clase esperada y_i. Ahora bien, la importancia o valor α_t de h_t surge de intentar optimizar dicho error y viene dado por la ecuación 3.8:

$$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) \tag{3.8}$$

Para actualizar la distribución de pesos D hay que señalar que inicialmente, cuando $T = 1$, todos los ejemplos son igualmente probables. En las siguientes iteraciones, es más probable seleccionar los ejemplos más difíciles (los que hacen fallar al clasificador) para obtener un clasificador que se enfoque en corregir los errores de su predecesor. Esta actualización de pesos viene dada por la ecuación 3.9

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(xi))}{Z_t} \tag{3.9}$$

Esto lo veremos aclarado con el siguiente ejemplo. Supóngase que en la primera ronda de clasificación todos los ejemplos son equiprobables, es decir que cada uno de ellos tiene igual importancia en ser clasificado correctamente o de manera incorrecta, y como se ilustra en la figura 3.10 todos los símbolos tienen el mismo tamaño. Es decir, cada uno de ellos tiene un peso de castigo igual a $\frac{1}{m}$, donde m es la cantidad de ejemplos de entrenamiento de nuestro

sistema. Como en esta ronda todavía no se ha modificado ningún peso, cualquier clasificador débil o hipótesis h_t, puede ser la mejor siempre y cuando tenga un error inferior a arrojar una moneda al aire, es decir, menor a $\frac{1}{2}$, y sea el menor de todos los errores de las hipótesis de cada columna. A este clasificador de la primera ronda, lo utilizaremos para actualizar los pesos en la siguiente iteración D_{t+1}.

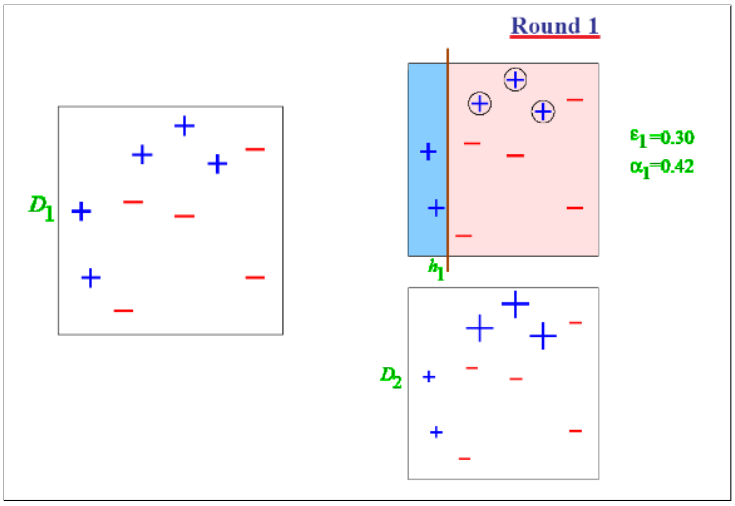


Figura 3.10: Primer clasificador h_t.

Para la segunda ronda, vemos que el tamaño entre los símbolos ya no es igual, como se observa en la figura 3.11 habiendo unos de mayor tamaño que otros. Esto se debe a la actualización de pesos generada por el criterio anterior ht, el cual penaliza a los ejemplos en los cuales falló en la tarea de clasificación, asignándoles un peso D mayor, disminuyendo el peso de los ejemplos clasificados correctamente, al menos en la ronda inmediata siguiente. Nuestro nuevo clasificador débil h_{t+1}, se concentrará entonces en clasificar aquellos ejemplos que no pudo clasificar nuestra anterior hipótesis h_t, y de nueva cuenta se tomará al clasificador con el menor porcentaje de error ϵ_t, y se calculará la importancia α de este nuevo clasificador, y se recalcularán los pesos de la siguiente ronda D_{t+2}.

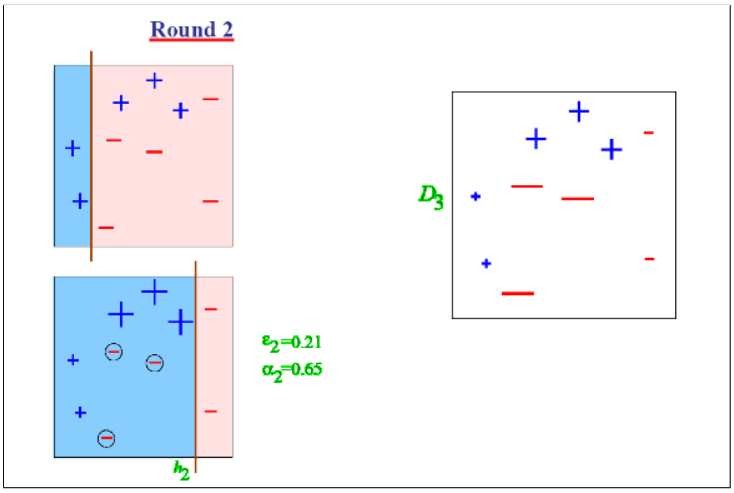


Figura 3.11: Segundo clasificador h_t.

Si observamos en la figura 3.12, los tamaños de cada símbolo volvieron a modificarse, debido a que el nuevo clasificador h_t, reevaluó los pesos de cada ejemplo, incrementando los ejemplos mal clasificados y reduciendo el peso de los correctamente clasificados. Este recalculo de pesos, es en base al clasificador h_t anterior, y en un proceso iterativo de T clasificadores débiles, cada clasificador por separado tiene un margen de error ϵ_t, y una importancia en su opinión α.

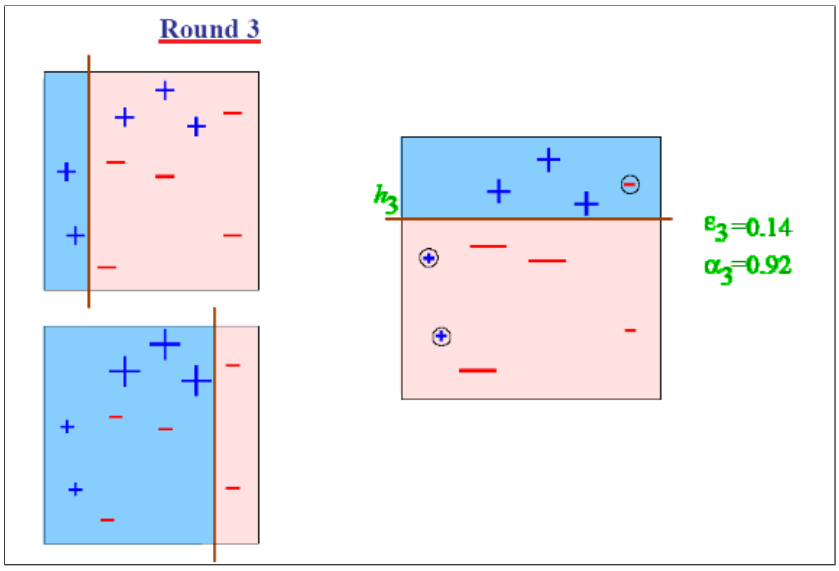


Figura 3.12: Tercer clasificador h_t.

Al juntar cada una de las opiniones de cada clasificador generamos una

hipótesis final o clasificador fuerte H_T, y finalmente las clases son correctamente etiquetadas, como podemos observar en la figura 3.13.

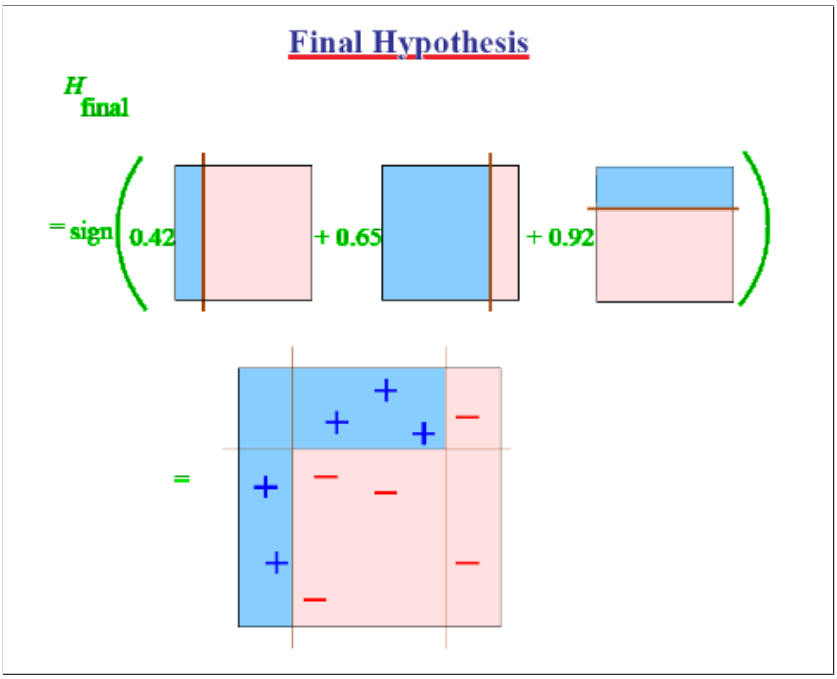


Figura 3.13: Hipótesis o clasificador fuerte final H_T.

3.7. Concurrencia

En el simple y más básico nivel de entendimiento, la concurrencia trata acerca de dos o más procesos separados que suceden al mismo tiempo. Podemos entrar la concurrencia como un proceso natural de la vida. Por ejemplo podemos caminar y hablar al mismo tiempo, o realizar diferentes acciones con cada mano, y por supuesto cada persona puede vivir su propia vida independientemente de otra.

Cuando hablamos de concurrencia en términos de computadoras, podríamos decir que un simple sistema podría realizar múltiples tareas independientes en paralelo, en lugar de secuencialmente, o una después de otra, aunque este no es un fenómeno nuevo. Los sistemas operativos multitarea que permiten a una simple computadora ejecutar múltiples aplicaciones al mismo tiempo a través de intercambio de tareas han sido comunes desde hace muchos años, y servidores de gama alta con múltiples procesadores que permiten una ver-

dadera concurrencia han estado disponibles durante más tiempo [36].

Lo que es nuevo, es el aumento de la prevalencia de los equipos que realmente pueden ejecutar múltiples tareas en paralelo en lugar de sólo dar la ilusión de hacerlo.

Históricamente, la mayoría de las computadoras siempre han tenido un solo procesador, con un simple núcleo de procesamiento, y esto en algunas computadoras de hoy en día todavía sigue presente. Este tipo de máquinas realmente sólo pueden realizar una tarea al mismo tiempo, pero pueden cambiar entre tareas muchas veces por segundo. Realizando una porción de una tarea, pausar, y entonces realizar otra porción de otra tarea, esto pareciera dar una apariencia de tareas que ocurren de manera concurrente. A este proceso se le llama *Task Switching* o intercambio de tareas, como se ilustra en la figura 3.14.

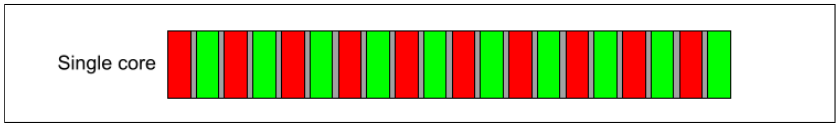


Figura 3.14: Intercambio de tareas [36].

Ahora bien, volviendo a la concurrencia, en la actualidad las computadoras contienen múltiples procesadores y se han utilizado para los servidores y las tareas de computación de alto rendimiento y para los equipos basados en procesadores con más de un núcleo en un sólo chip (procesadores multi núcleo) y son cada vez más comunes en las máquinas de escritorio. Si tienen varios procesadores o núcleos múltiples dentro de un procesador (o ambos), estos equipos son capaces de genuinamente correr más de una tarea en paralelo. Llamamos a esta concurrencia, concurrencia de hardware [36].

La figura 3.15 muestra un escenario idealizado de una computadora con exactamente dos tareas para hacer, cada uno dividido en 10 partes de igual tamaño. En una máquina de doble núcleo, cada tarea puede ejecutarse independiente mente en un núcleo.

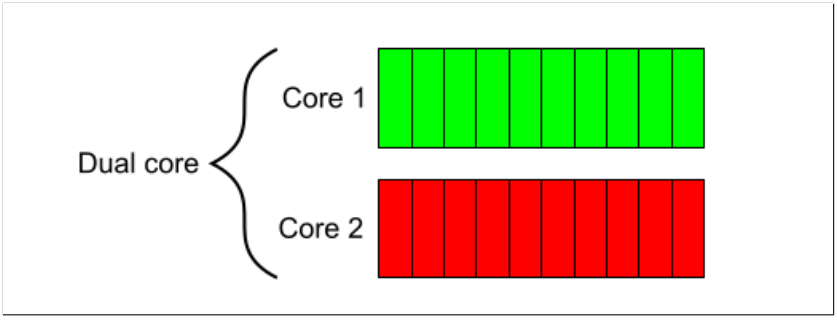


Figura 3.15: Concurrencia de hardware [36].

Hay dos razones principales para utilizar la concurrencia en una aplicación: separación de tareas y rendimiento. De hecho, que son prácticamente las únicas razones para utilizar la concurrencia; todo lo demás se reduce a uno u otro (o incluso ambos) cuando se busca optimizar un proceso para que se realice lo suficientemente bien y en un buen tiempo, y no sólo por una razón tan trivial como la de "yó programo de manera concurrente, porque se puede y porque quiero".

Es igual de importante saber cuándo no usar concurrencia como lo es saber cuándo hay que utilizarla. Fundamentalmente, la única razón para no utilizar la concurrencia es cuando el beneficio no vale la pena ante el costo. Usando código concurrente es más difícil de entender en muchos casos, por lo que hay un costo intelectual directo a escribir y mantener código multiproceso, y la complejidad adicional también puede dar lugar a más errores. A menos que el potencial sea lo suficientemente grande o la separación de tareas lo suficientemente claras para justificarlas, de otra manera no debe usarse la concurrencia.

3.8. La concurrencia y Ada Boost

Adaboost es un algoritmo que cumple, con los requisitos básicos para ser paralelizable. Visualicemos el proceso de selección de los clasificadores débiles o hipótesis h_t. En cada ronda de búsqueda de clasificadores h_t, cada una de las columnas es evaluada, elemento por elemento con un criterio de umbral mayor ó menor al elemento de la fila x evaluado actualmente, y que es comparado contra las X filas de cada columna, para al final tener un candidato a clasificador h_t por cada columna como se muestra en la figura 3.16.

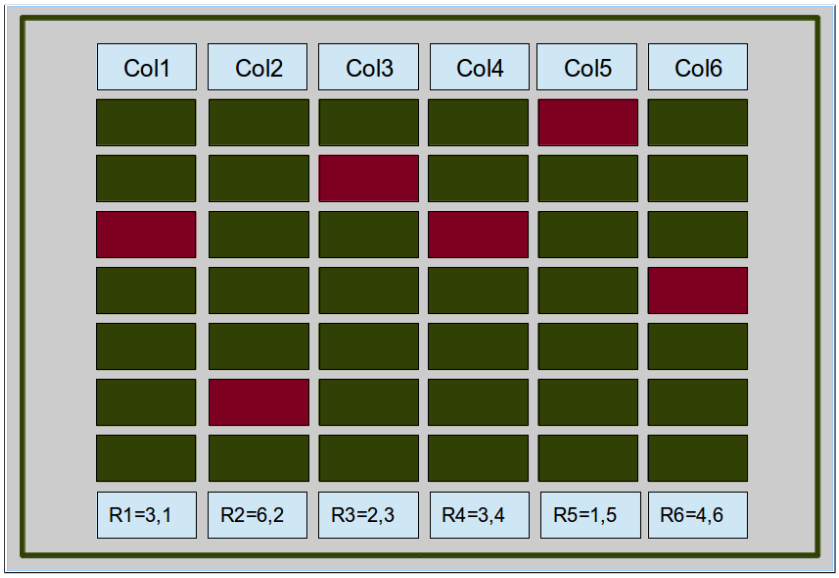


Figura 3.16: Proceso de selección de candidatos h_t secuencial.

Al final de este proceso, cada uno de los candidatos, es comparado contra todos los otros Y candidatos de cada columna, y al final de esta comparación es seleccionado el mejor clasificador con el menor error e_t como se ilustra en la figura 3.17, en la cual es el elemento $R5 = 1,5$ con un error $e_t = 0.0526$. Todo este proceso en un tiempo constante T por cada ronda.

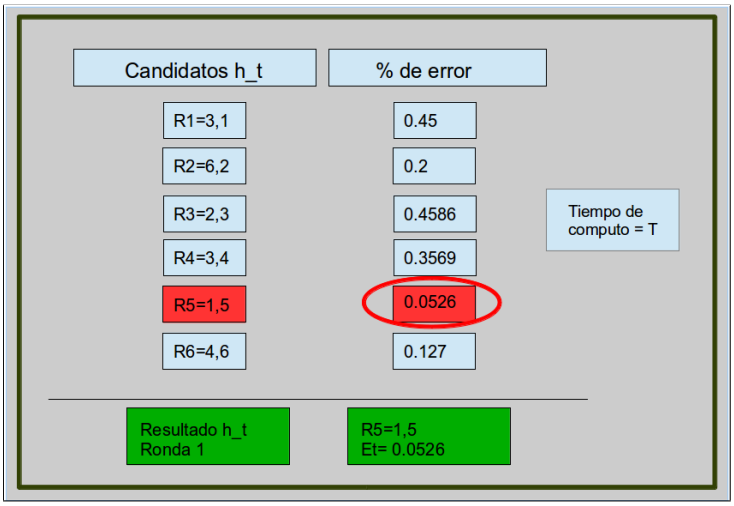


Figura 3.17: Ejemplo de selección final h_t entre los candidatos a clasificador débil.

Ahora bien, este mismo proceso pensémoslo de manera concurrente. Imaginemos que cada color en las columnas representa un hilo de un mismo

proceso, que se ejecuta a la par junto con otros N hilos, que realizan la misma tarea de búsqueda de candidatos a clasificador débil h_t. Como se ilustra en la figura 3.18, vemos los colores Magenta, Amarillo y Verde que representan conceptualmente tres hilos dentro de un mismo proceso de búsqueda. La idea general es que cada hilo busca y almacena a sus clasificadores candidatos, según el número de columnas asignadas a este hilo. Teniendo finalmente igual número de candidatos tanto como columnas hubiese, siempre y cuando cada candidato a clasificador débil supere el umbral de error $h_t < 0.5$.

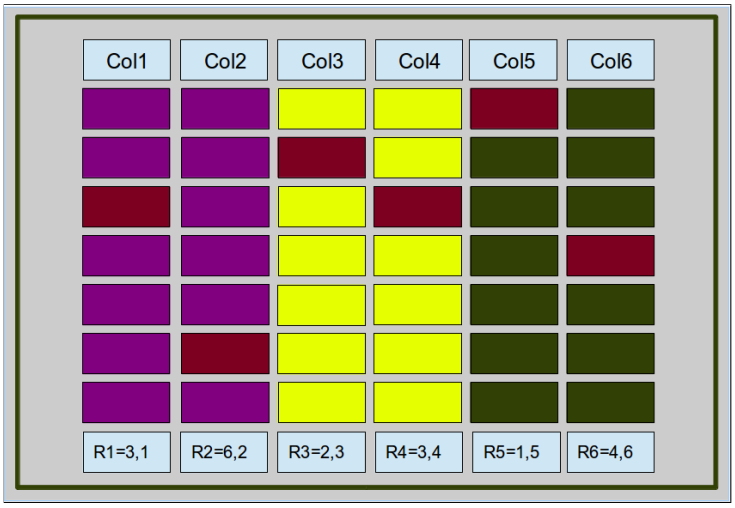


Figura 3.18: Búsqueda de candidatos a clasificador débil de manera concurrente.

Este proceso paralelizado asigna a cada hilo un número de columnas por bloque, evalúa entonces tantos bloques como hilos se hayan creado. Y por cada hilo nos devuelve al mejor candidato a clasificador débil por bloque. Volviendo al mismo ejemplo que se ilustra en la figura 3.19, si tuviéramos tres hilos, entonces tendríamos tres bloques de los cuales surgirían tres candidatos a clasificador débil, de los cuales en una etapa final de búsqueda para el clasificador débil final h_t escogeremos al candidato con menor error e_t.

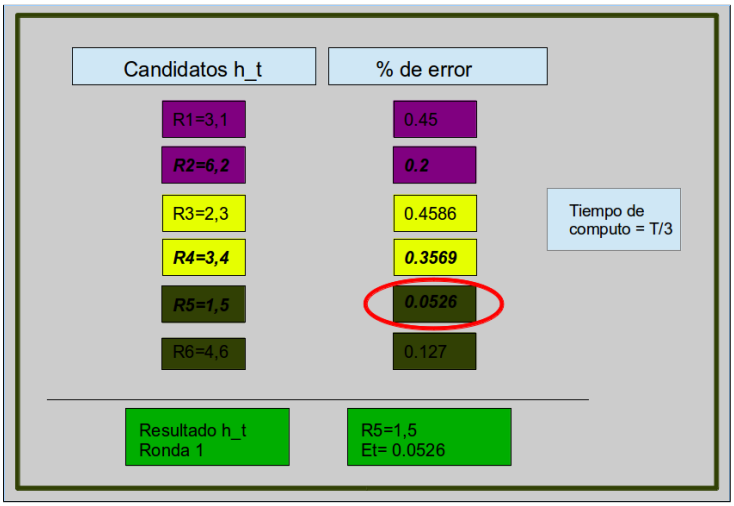


Figura 3.19: Selección de candidatos por bloques.

Y de nueva cuenta obtenemos a nuestro mejor clasificador débil de la ronda uno, el cual es el elemento $R5 = 1,5$ con un error $e_t = 0.0526$, que como podemos observar es exactamente igual que en el proceso secuencial, a diferencia de que en esta iteración de búsqueda, el tiempo fue de $T/3$ debido a la ejecución concurrente de tres hilos realizando la misma tarea.

Capítulo 4

Metodología

En esta sección hablaremos acerca de la arquitectura del sistema. El sistema se compone de tres etapas básicas para el reconocimiento de gestos. Como primera etapa está la captura de las letras gesticuladas por el usuario con el sistema Kinect. La segunda etapa consiste en el preprocesamiento de la imagen para su reconocimiento. La tercera y última etapa consiste en la identificación del gesto y el despliegue de su símbolo en un monitor de computadora.

4.1. Arquitectura del sistema

El sistema consiste en una computadora (PC) conectada a un sensor Kinect que captura las imágenes, una PC adicional donde se procesan las imágenes y un monitor donde se despliega el símbolo alfanumérico del gesto reconocido como puede apreciarse en la figura 4.1. Para la parte de captura de imágenes en este experimento hemos utilizado el sistema operativo Windows 8 versión Ultimate de 64 bits en su versión Desktop. Para la parte de reconocimiento y procesamiento de imágenes hemos utilizado la distribución GNU/Linux Ubuntu 12.04 LTS (nombre clave Precise Pangolin - Pangolín Preciso) basada en el Kernel Linux 3.19, en su versión Unity Desktop de 64 bits.

Para el procesamiento y manejo de imágenes capturadas por el sensor Kinect se utilizó la suite de visión por computadora OpenCV, en su versión 2.4.9 para GNU/Linux, la cual es totalmente libre y gratuita para propósitos tanto académicos como comerciales.

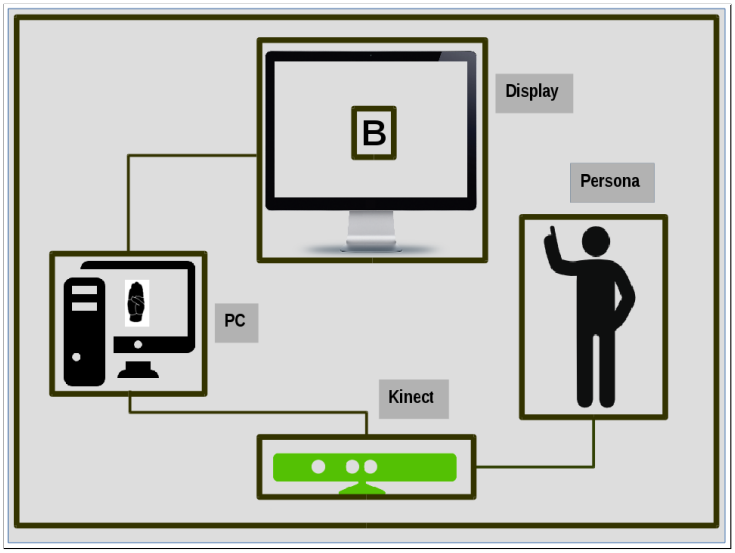


Figura 4.1: Arquitectura del sistema.

4.2. Base de datos de imágenes

4.2.1. Descripción de los participantes

El número de participantes voluntarios fue de cien personas, y fueron escogidos de manera aleatoria entre una población estudiantil universitaria, de entre 18 a 30 años, de ambos géneros, con proporciones corporales bastante variadas. Cabe recalcar que para este experimento no se contempló ninguna persona con carencia de falanges o dedos. Fuera de esto, las características fisonómicas de los participantes fueron de lo más variadas, manos grandes, pequeñas, delgadas, gruesas, en fin un mosaico de características bastante amplio para robustecer el sistema lo más posible.

4.2.2. Condiciones lumínicas y de distancia

En este experimento se tomaron las fotos en diversas situaciones lumínicas, por ejemplo, en ventanas con iluminación natural, por las mañanas, las tardes y las noches; además de algunas tomadas en cubículos cerrados con luz controlada y en todas las situaciones. Las imágenes de todos los participantes fueron capturadas sin ningún problema.

La captura de imágenes se realizó a tres distintas profundidades. La última profundidad en promedio fue de 90 cms, la segunda profundidad de 88.5

cm, y la tercera profundidad se tomó a 87 cm, a una altura no superior a 1.20 mts de distancia sobre el suelo. Las manos de los participantes se procuraron colocar en un ángulo recto, y lo más alejado del cuerpo posible.

4.2.3. Formato y dimensiones

Las imágenes crudas que nos provee el sensor Kinect vienen originalmente en forma de tres matrices de valores comprendidos entre el cero y el 255, y se procesan para exportarlas en formato PNG con un ancho de 640 pixeles, por un alto de 480 pixeles, divididas en tres canales RGB. Adicionalmente a ello se agregó un recuadro guía en color azul, como se muestra en la figura 4.2, en donde el participante ubica la mano como una referencia, ya que al no haber un marco guía en las primeras capturas, a los participantes les era complicado ubicar la mano en una misma posición a pesar de ser por un periodo corto de tiempo.

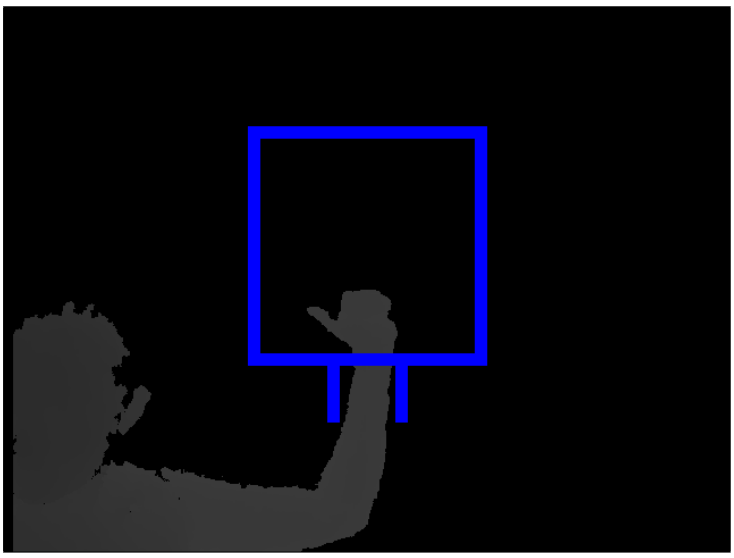


Figura 4.2: Cuadro guía.

4.2.4. Cantidad de gestos por usuario

Cada usuario realizó diez gestos distintos, las letras A, B, C, D y E, y los gestos numéricos Uno, Dos, Tres, Cuatro, y Cinco como se observa en la figura 4.3, en una sola toma, y variando la profundidad de captura del sensor Kinect, aproximadamente 1.5 cm, para obtener imágenes de la seña en diferentes profundidades. También se agregó un gesto de control llamado MANO_MUESTRA, esta mano se capturó para tener una medida estándar

de alto y ancho para cada sujeto, lo que facilita la tarea de comparación entre manos de diversos tamaños. En números totales, cada usuario realizó diez gestos alfanuméricos, y a cada gesto se le capturaron tres distintas profundidades, formando una base de datos de 3,000 imágenes únicas.

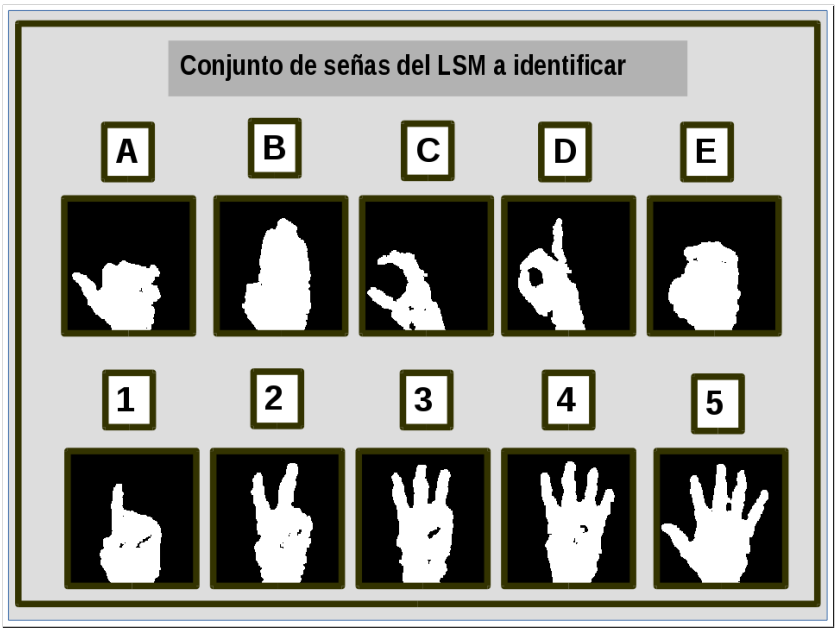


Figura 4.3: Conjunto de diez señas a identificar.

4.3. Preprocesamiento de imágenes

4.3.1. Selección del área de la mano

Como habíamos mencionado con anterioridad en la sección 4.2.3, al momento de realizar la captura de imágenes con los participantes, se graficó en pantalla un área cuadrada de color azul con dos barras para delimitar la zona donde el usuario debe centrar la mano. La utilidad del área cuadrada azul es ser un marco de referencia para facilitar, al capturar las imágenes debido a su posición relativa en una misma sección de la pantalla en todo momento. El proceso es bastante simple, en cada imagen tomada la única constante es el área azul del cuadrado, y todo el resto puede ser fácilmente fondo, o parte del cuerpo del participante. Puesto que todo lo que resta del cuerpo y del fondo, que no sea mano, no nos interesa, entonces simplemente recortamos el área exacta contenida dentro del cuadrado azul y tenemos la mano de los participantes de manera inmediata, como se ilustra en la figura 4.4.

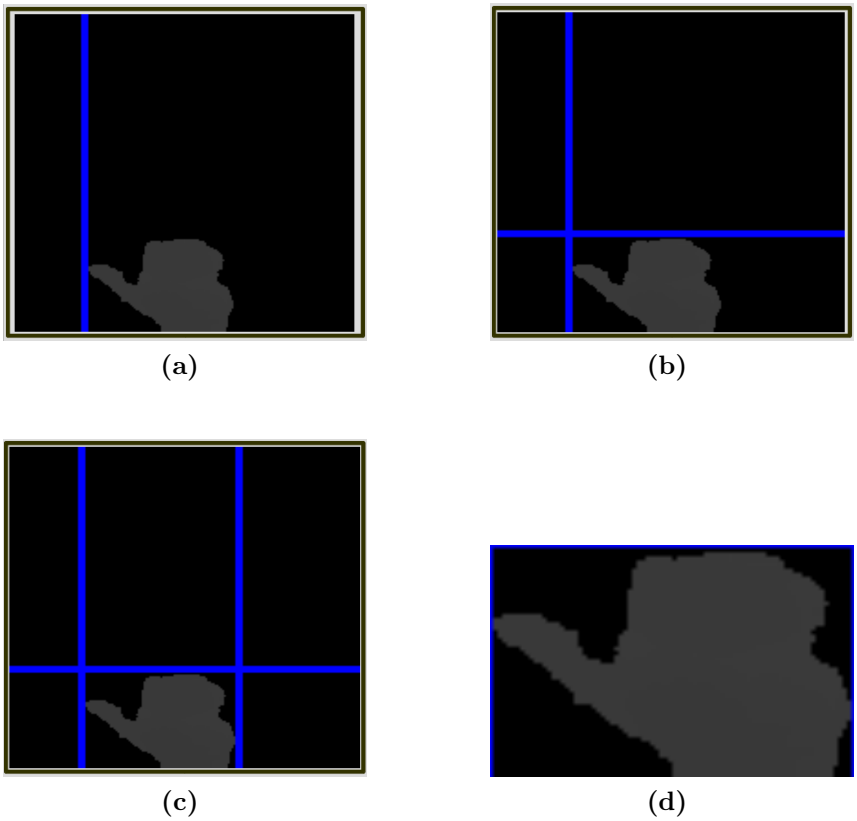
Figura 4.4: Recorte del área donde se ubica la mano.

El recorte exacto de la mano tiene dos propósitos en nuestro sistema: el saber las dimensiones exactas de la mano de cada participante cada vez que realiza un gesto, y, el poder centrar posteriormente la mano en una zona equivalente para todas las demás manos que no sean del mismo participante. El primer paso es ubicar los bordes externos de la mano para realizar el corte exacto. Después de encontrar los bordes donde comienza el área de la mano se extrae la región que forma la mano. Para encontrar estos bordes, primero recorremos de manera vertical las columnas, hasta encontrar el primer pixel de izquierda a derecha en nuestra imagen como se observa en la figura 4.5a.

De manera iterativa, ahora buscamos al primer pixel de arriba hacia abajo buscando las puntas superiores de los dedos como se muestra en la figura 4.5b.

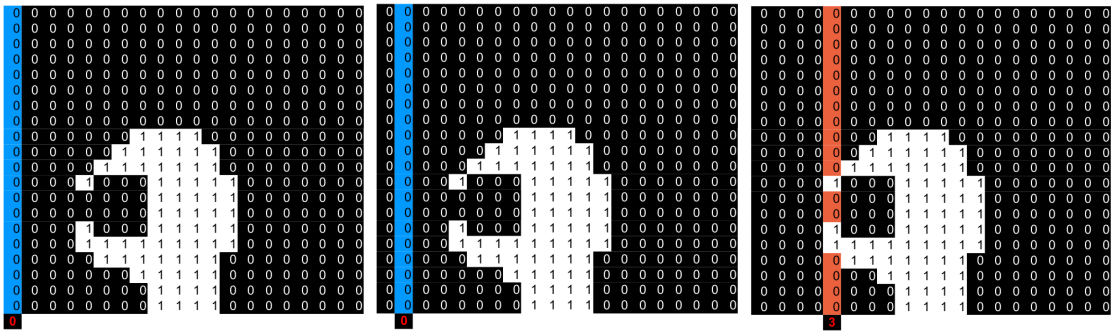
Finalmente se busca el primer pixel en la mano distinto de cero de derecha a izquierda en la mano. De esta manera se obtiene el recorte exacto de la mano (como se observa en la figura 4.5c), y obtenemos un recorte exacto únicamente del área donde se encuentra la mano, ya que el fondo u otras áreas aledañas son descartadas, como se observa en la figura 4.5d.

Todas estas búsquedas las optimizamos apoyándonos en la imagen integral de M pixeles de *Ancho*, y N pixeles de *Alto*. Para este proceso utilizamos un rectángulo de 1 pixel de ancho y N pixeles de alto para encontrar el borde izquierdo de la mano como se observa en la figura 4.6. Esta búsqueda recorre sumando toda el área contenida dentro del rectángulo, y se detiene en la coordenada M cuyo valor de sumatoria es diferente de 0.



(a) **(b)** **(c)** **(d)**

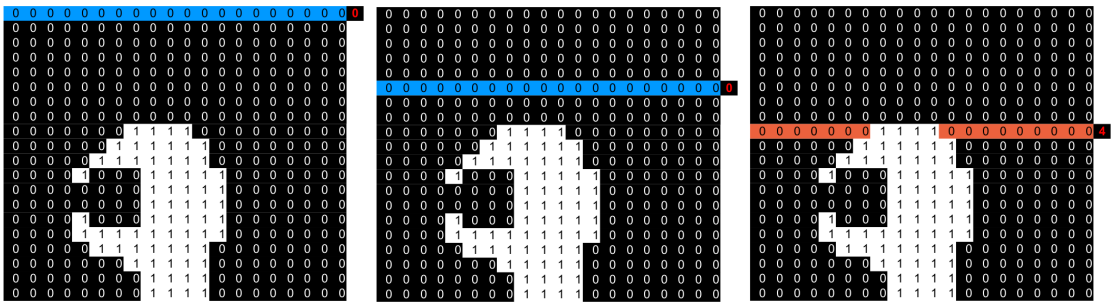
Figura 4.5: Busqueda de: a) Primer pixel del borde izquierdo de la mano, b) Primer pixel del borde superior de la mano, c) Primer pixel del borde derecho de la mano, d) Mano recortada de manera exacta sin excedentes de fondo.



(a) **(b)** **(c)**

Figura 4.6: Sumatoria de áreas verticales para obtener el borde izquierdo de la mano.

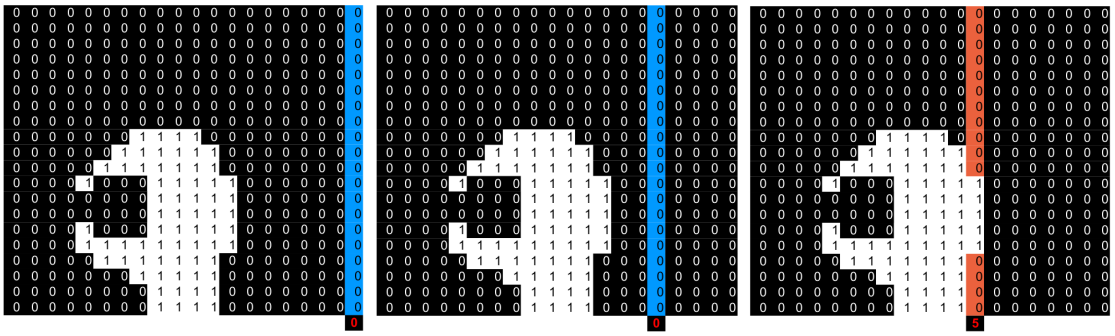
Para encontrar el borde superior donde inicia la mano esta operación la repetimos con un rectángulo de 1 pixel de alto, y M pixeles de ancho, comenzando en la parte superior de la imagen, la recorremos de manera descendente hasta encontrar la primera sumatoria $\neq 0$ y entonces obtenemos la coordenada N del borde superior de la imagen 4.7.



(a) **(b)** **(c)**

Figura 4.7: Sumatoria de áreas horizontales para obtener el borde superior de la mano.

Por último, el borde derecho donde termina la mano, es hallado de forma similar al borde izquierdo. Recorremos la imagen de derecha a izquierda con un rectángulo de un pixel de ancho y N pixeles de alto, y nos detenemos en la primera sumatoria del área rectangular $\neq 0$ como se observa en la figura 4.8 y entonces tenemos la segunda coordenada M donde termina el borde derecho de la mano.



(a) **(b)** **(c)**

Figura 4.8: Sumatoria de áreas horizontales para obtener el borde superior de la mano.

Para el centrado uniforme de todas las manos, se utilizo de referencia una imagen de la mano extendida del sujeto (ver figura 4.9). Dicha imagen

la denominamos MANO_MUESTRA. Esta seña se utiliza para medir el alto y ancho máximos que puede alcanzar la mano del individuo. El objetivo de tomar esta muestra es con fines de calibración para que ningún gesto quede desproporcionadamente alto o ancho en comparación a los demás gestos realizados por otros participantes. Como se muestra en la figura 4.9, observamos la mano muestra centrada. En la mano muestra se extraen las siguientes medidas.

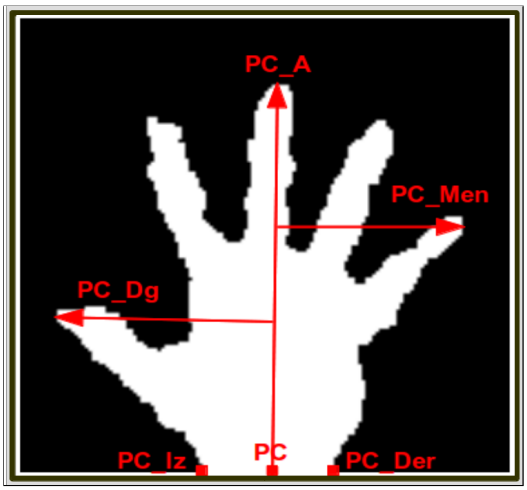


Figura 4.9: Medidas para la mano muestra.

- Punto izquierdo de la muñeca PC_Iz. Esta medida define el punto exacto donde comienza la parte izquierda de la muñeca.
- Punto derecho de la muñeca PC_Der. Esta medida define el punto exacto donde comienza la parte derecha de la muñeca.
- Punto central de la muñeca PC. Esta medida se define como la distancia exacta entre ambos puntos de la muñeca y se puede calcular como: $\frac{PC\_Iz+PC\_Der}{2}$.
- Punto central de la muñeca a dedo gordo PC_Dg. Esta medida se define como la distancia entre el punto central de la muñeca hasta el primer pixel donde comienza la mano de una persona de izquierda a derecha.
- Punto central de la muñeca a dedo meñique PC_Men. Esta medida se define como la distancia entre el punto central de la muñeca hasta el primer pixel donde comienza la mano de una persona de derecha a izquierda.

- Punto más alto de la mano a hasta la muñeca PC_A. Esta medida se define como la distancia entre el borde superior de la mano hasta el punto central de la muñeca, es decir el alto total de la mano.

Para obtener *Alto* y *Ancho* finales de nuestra imagen recortada y centrada, se realizan las siguientes comparaciones descritas en los algoritmos 2, 3.

Algorithm 2 Busqueda del ancho óptimo MANO_MUESTRA

```
1: if PC_Dg >= PC_Men then
2:     Ancho = PC_Dg * 2
3: else
4:     Ancho = PC_Men * 2
5: end if
```

Algorithm 3 Busqueda del alto óptimo MANO_MUESTRA

```
1: if PC_A >= Ancho then
2:     Alto = PC_A
3:     Ancho = PA_A
4: else
5:     Alto = Ancho
6: end if
```

El algoritmo 2, se encarga de encontrar que distancia de *Ancho* es la óptima para centrar nuestra imagen. Para obtener esta medida de *Ancho* se compara si dos veces la distancia del dedo gordo (PC_Dg), al punto central de la muñeca PC, es mayor a dos veces la distancia del dedo meñique (PC_Men) al punto central de la muñeca; si es así, entonces el *Ancho* es dos veces la distancia PC_Dg, de otra manera el *Ancho* es dos veces la distancia PC_Men.

El algoritmo 3, encontrará que distancia de *Alto* es la óptima para centrar la imagen. Esta medida compara si el alto actual de la imagen PC_A es mayor o igual al *Ancho* calculado previamente en el algoritmo 2, de ser afirmativa la respuesta entonces el *Ancho* es igual al alto actual PC_A; de otra manera el *Alto* es igual al *Ancho* del algoritmo 2. Con estas medidas se recorta y centra la MANO_MUESTRA.

4.3.2. Centrado y escalado

Los puntos extraídos de la muestra en el apartado anterior, nos servirán de referencia para centrar y ajustar todas las imágenes capturadas del mismo

participante.

Al capturar cada uno de los gestos, correspondientes a las letras o números, extraeremos también las medidas PC_Iz, PC_Der, PC, PC_Dg, PC_Men, PC_A, *Ancho* y *Alto*; y se realiza una comparación de medidas para establecer los altos y anchos que se utilizarán para centrar las imágenes. Una vez calculados el Ancho y Alto de las imágenes de todos los gesto volvemos a verificar, ahora entre los *Ancho_C* y *Alto_C* de cada imagen de cada gesto, y los *Ancho* y *Alto* de la Mano_Muestra.

Ancho imagen del gesto consecutivo = *Ancho_C*
Alto imagen del gesto consecutivo = *Alto_C*

Algorithm 4 Busqueda del ancho óptimo muestras consecutivas

```
1: if Ancho > Ancho_C then
2:     AnchoFinal = Ancho
3: else
4:     AnchoFinal = Ancho_C
5: end if
```

Algorithm 5 Busqueda del alto óptimo muestras consecutivas

```
1: if Alto > Alto_C then
2:     AltoFinal = Alto
3: else
4:     AltoFinal = Alto_C
5: end if
```

El algoritmo 4 se encarga de encontrar que distancia de ancho es la mayor, y por tanto la óptima entre el *Ancho* de la Mano_Muestra, y el *Ancho_C* de la imagen del gesto calculado actualmente, que puede ser cualquiera de los diez símbolos alfanuméricos (A,B,C,D,E,Uno,Dos,Tres,Cuatro ó Cinco). Si el *Ancho* de la Mano_Muestra, es mayor al *Ancho_C* del gesto actual, entonces el *AnchoFinal* es igual al *Ancho* de la Mano_Muestra; de otra manera el *AnchoFinal* es igual al *Ancho_C* del gesto actual.

El algoritmo 5 se encarga de encontrar que distancia de alto es la óptima entre el *Alto* de la Mano_Muestra, y el *Alto_C* de la imagen del gesto calculado actualmente. Si el *Alto* de la Mano_Muestra, es mayor al *Alto_C* del gesto actual, entonces el *AltoFinal* es igual al *Alto* de la Mano_Muestra; de

otra manera el $AltoFinal$ es igual al $Alto\_C$ del gesto actual.

Ahora con el $AnchoFinal$ y el $AltoFinal$ óptimos para centrar las imágenes de los símbolos del mismo usuario, necesitamos hacer una última comparación, ya que necesitamos una imagen de dimensiones cuadráticas $N \times N$ y por lo general $AnchoFinal \neq AltoFinal$ lo que genera una imagen no cuadrática de $M \times N$.

Algorithm 6 Busqueda del alto y ancho óptimo finales para las muestras consecutivas

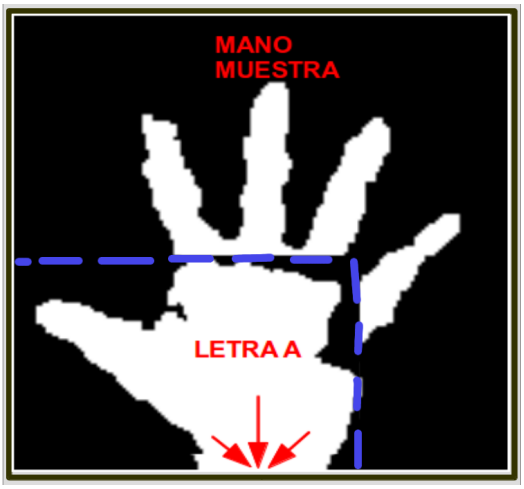
```
1: if AltoFinal > AnchoFinal then
2:     AnchoFinal = AltoFinal
3: else
4:     AltoFinal = AnchoFinal
5: end if
```

El algoritmo 6, realiza una comparación entre el $AltoFinal$ y el $AnchoFinal$, para obtener una imagen de dimensiones cuadráticas. Si el $AltoFinal$ es mayor al $AnchoFinal$, entonces se substituye el $AnchoFinal$ y ahora toma el valor del $AltoFinal$; de otra manera $AltoFinal$ se substituye y toma el valor del $AnchoFinal$.

Y de esta manera obtenemos las medidas cuadráticas $N \times N$ de la imagen que contendrá el gesto.

Después de este procedimiento, para centrar cada imagen en la misma posición relativa, calculamos el centro del cuadro contenedor dividiendo $PC\_C = \frac{Ancho}{2}$ y calculamos la resta del $Desplazamiento = (PC\_Dg – PC\_C)$ que necesita moverse la mano para quedar su punto central de la muñeca sobre puesto al punto central del cuadrado contenedor de la imagen. Ubicamos entonces el recorte exacto de la mano sumando en las coordenadas Y o distancia horizontal del $Desplazamiento$ a la distancia faltante para ubicar el punto central de la muñeca PC en el centro de la imagen. Con esta sumatoria los puntos centrales de la muñeca PC de la Mano_Muestra como los de las imágenes de los gestos consecutivos del mismo sujeto quedan ubicados en la misma posición dentro del centro de la imagen como se muestra en la figura 4.10.

(a) (b)



(c)

Figura 4.10: Centrado de imágenes: a) Mano_Muestra, b) Imagen de la seña A, c) Seña A sobrepuestra a Mano_Muestra.

4.3.3. Morfología matemática

Después de obtener una imagen de dimensiones cuadradas $N \times N$, la imagen binarizada tenia demasiadas irregularidades, como protuberancias o pixeles excedentes, así como zonas inconexas o huecos. Para clarificar o limpiar la imagen y dejarla con una apariencia mas regular recurrimos al método de la Morfología matemática, para mejorar la forma de la imagen. Al momento de aplicar la morfología matemática seleccionamos dos operadores básicos que combinados, realizan una tercera forma de morfología matemática. En este caso utilizamos primero la erosión para eliminar cualquier borde irregular en la figura, así como partes que no se ubican totalmente dentro de la

mano (ver figura 4.11a).

Después aplicamos la operación dilatación para expandir posibles áreas que hayan quedado inconexas debido a la erosión. Esta combinación de operadores morfológicos nos dan como resultado a la operación apertura, que como se puede observar en la figura 4.11b veremos que la forma se ve mucho más regular que la imagen inicial.

Después de todo este preprocesamiento, para realizar un cálculo manejable de características Haar, se reescalan las imágenes preprocesadas a una medida estándar de 30 pixeles × 30 pixeles como se muestra en la figura 4.11c. Se probó también una medida de 24 × 24 pixeles, pero la forma básica de cada gesto se perdía por la baja resolución de la imagen.

4.4. Extracción de características

4.4.1. Características Haar 2D

El proceso de extracción de características Haar comienza exactamente con la imagen escalada a 30 × 30 pixeles. En esta parte creamos una imagen integral de la imagen escalada de 30 × 30 pixeles que ya está recortada y centrada. Computamos tres formas básicas de las características Haar. El primer par de características es de tipo Haar vertical con dos barras (ver figura 3.7 a), y el otro es el Haar vertical de tres barras(ver figura 3.7 c).

Para este cálculo, se recorrió de manera semi exhaustiva la imagen, en cada ronda los desplazamientos se incrementan con dos pixeles de manera horizontal, dos pixeles de manera vertical, así como un crecimiento en cada barra de tres pixeles de manera vertical, y el crecimiento horizontal fue de un pixel de manera constante por cada ronda de cálculo.

El segundo par de características Haar calculadas fueron las horizontales con dos barras (ver figura 3.7 b) y tres barras horizontales (ver figura 3.7 d). En este cálculo, de igual forma, el recorrido fue semi exhaustivo, desplazando dos pixeles horizontales, y dos pixeles verticales por cada recorrido, así como un crecimiento de tres pixeles en las columnas de cada característica en cada iteración, y un crecimiento constante de un pixel en las filas por cada ronda de cálculo. Finalmente, la última clase de característica Haar computada fue la de tipo diagonal (ver figura 3.7 e), aunque ésta a diferencia de las otras características Haar si se recorrió de manera exhaustiva, debido a que la for-

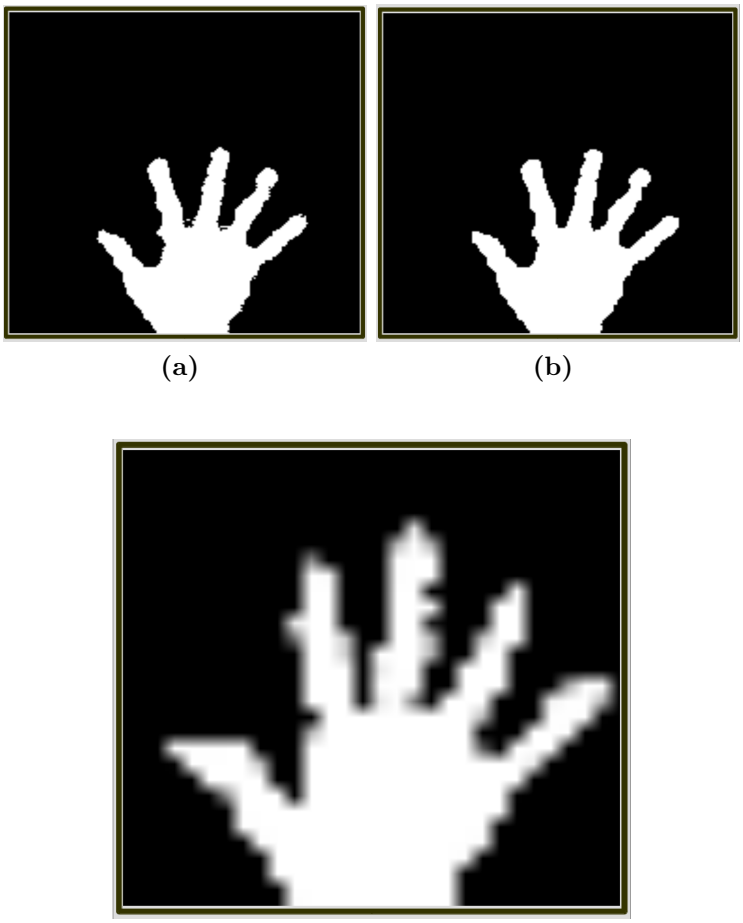


Figura 4.11: Morfología matemática aplicada a las imágenes: a) Imagen sin morfología, b) Imagen con apertura morfológica, c) Imagen reescalada.

ma de la característica se expande mas aceleradamente que sus predecesoras.

En cada ronda de cálculo, el desplazamiento fue incrementándose de un pixel en el desplazamiento vertical, y un pixel en el desplazamiento horizontal, y el incremento del tamaño de la característica por cada cuadro, fue del doble del tamaño inicial, es decir, si inicialmente la característica tenía una dimensión de 4 $\times$ 4 pixeles, divididos en cuatro regiones, en la siguiente iteración cada región mediría dos pixeles de alto por dos pixeles de ancho, es decir, 4 pixeles por región, lo que nos daría un total de 16 pixeles por característica dividido en cuatro regiones. Cada una de las cinco formas de características Haar se muestran en la imagen 4.12

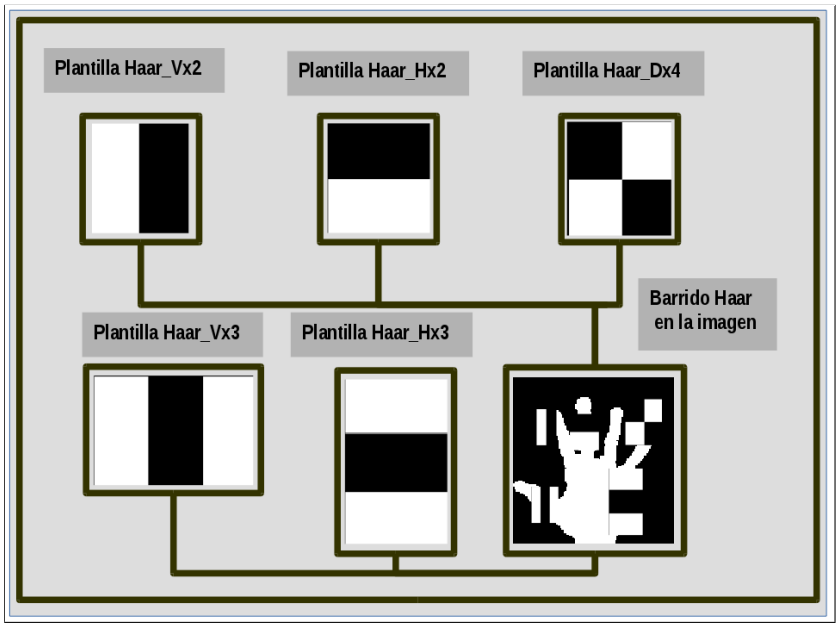
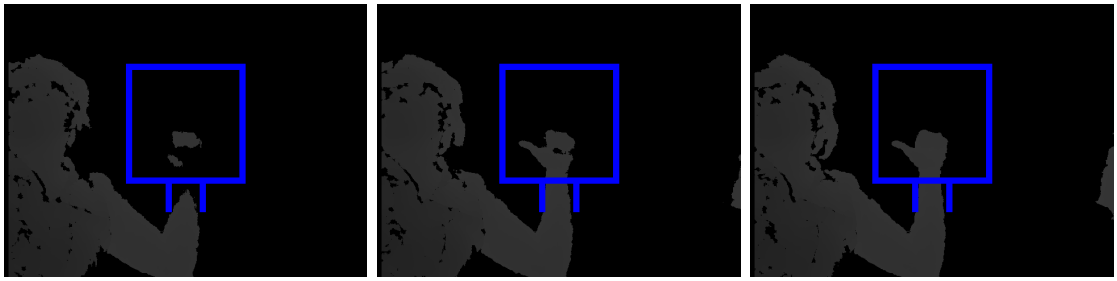


Figura 4.12: Extracción de características Haar.

Para la extracción de características Haar en 2D se utilizo únicamente la imagen de mayor profundidad (4.13c) con respecto al sensor Kinect. De esta manera el vector de características Haar 2D lo constituyen un total de 45,809 elementos y es este mismo el vector que sirve de entrada para entrenar al algoritmo de aprendizaje automático Ada Boost.



(a) **(b)** **(c)**

Figura 4.13: Imágenes tomadas a profundidad: a) Imagen a prundidad uno 87 cm, b) Imagen a prundidad dos 88.5 cm, c) Imagen a prundidad tres 90 cm.

4.4.2. Creación de valores Haar en 3D

Los vectores Haar que se implementarán para el entrenamiento final de nuestro sistema, utilizarán cada uno de los ejemplos de letras y números

gesticulados por nuestros participantes, a excepción del gesto muestra. En cada uno de estos vectores se encuentra la información de las tres imágenes a tres profundidades diferentes tomadas en la misma muestra del sujeto. Por cada profundidad se extrajeron 45,809 características, y finalmente de las tres profundidades iniciales se obtienen 137,427 características. Para obtener una característica Haar en 3D, es necesario calcular un volumen integral VI, como se muestra en la figura 4.14.

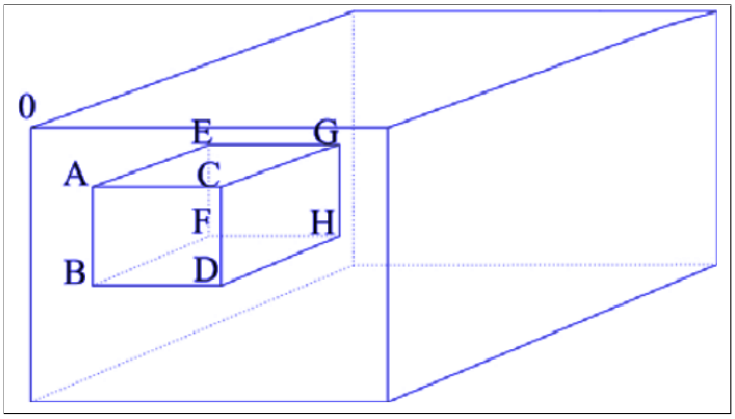


Figura 4.14: Cuadro ilustrativo de un volumen integral [4].

La suma de los pixeles dentro de la caja cúbica $ABCDEFGH$, que corresponde a un par de imágenes, puede ser computada mediante ocho operaciones de suma y resta. La suma queda dada como se muestra en la ecuación 4.1.

$$VI(H) - VI(D) - VI(F) - VI(G) + VI(B) + VI(C) + VI(E) - VI(A) \tag{4.1}$$

donde 0 *es el origen.*

Para calcular las tres profundidades de una seña, reemplazamos en la ecuación las profundidades que se muestra en la tabla 4.1, en la ecuación 4.1 y obtenemos las nuevas profundidades en 3D que son un total de 274,854 características por cada vector combinando tres características Haar 2D (profundidades uno, dos, tres), y tres características Haar 3D (profundidades cuatro, cinco, seis).

4.5. Reconocimiento

Para entrenar cada clasificador fuerte que identifique un solo signo, se siguió la metodología del Adaboost planteada previamente en el capítulo

Profundidad	Abarca	Descripción
1	Casi nada	La distancia es la menor (figura 4.13a)
2	Mitad	La distancia es la menor + 1.5 cm (figura 4.13b)
3	Completa	La distancia es la menor + 3 cm (figura 4.13c)
4	Combina 3+2	Suma de la ultima y segunda profundidades
5	Combina 2+1	Suma de la segunda y primera profundidades
6	Combina 1+2+3	Suma de todas las profundidades

Tabla 4.1: Descripción de profundidades.

tres. Aunque de hecho el algoritmo AdaBosst es virtualmente invulnerable al sobre ajuste u *overfitting*, se opto por generar bloques con índices en desorden tanto para los elementos train, como para los elementos test, es decir un ordenamiento aleatorio tanto de elementos positivos como negativos son presentados al algoritmo Adaboost para comenzar el entrenamiento. Como se había mencionado anteriormente, para comprobar la eficacia del algoritmo es utilizado el método de validación cruzada, con la modalidad ONE LEAVE OUT como se ve en la figura 4.15, o uno fuera, es decir, por cada bloque se toma a intervalos conocidos un elemento para agregarlo al bloque de test y es inmediatamente eliminado del bloque train. Por ejemplo si existen 200 ejemplos para entrenar el signo de la letra A, y necesitamos el 20 % de dichos signos, entonces tomamos los 200 ejemplos divididos entre los 40 elementos necesarios y tenemos un intervalo de cinco elementos. O bien, comenzamos a tomar ejemplos desde el índice 0 de y avanzamos con incremento de cinco, “0,5,10,15”... etc., hasta formar nuestro grupo de 20 elementos test positivos, 20 elementos test negativos, y el 80 % restante osea 160 elementos conforman el bloque train.

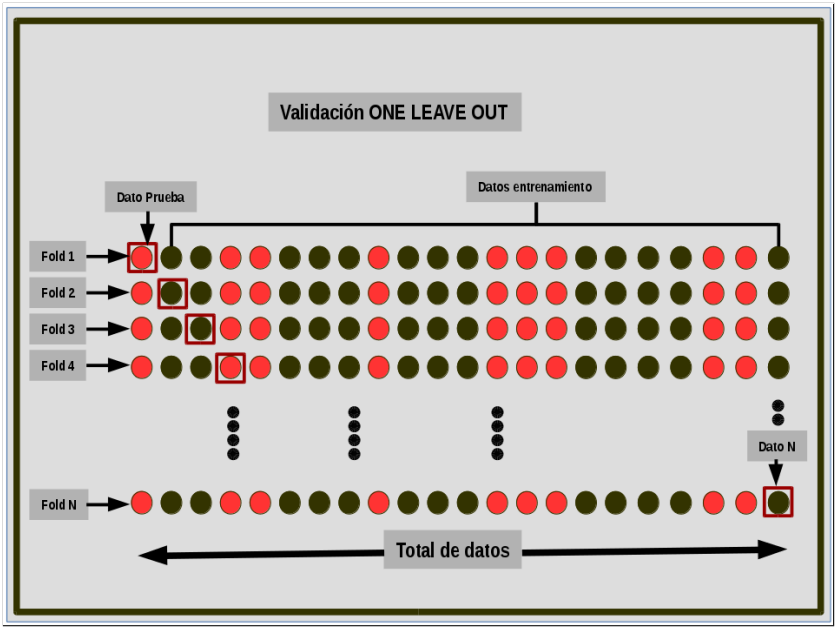


Figura 4.15: Validación One Leave Out.

Cada clasificador es generado del bloque train. El proceso se divide en 2 etapas, la generación de cada uno de los T clasificadores débiles, en este caso $T = 30$, y la generación del clasificador fuerte que es la combinación de los 30 t clasificadores débiles. En la primera etapa, como se había explicado en el marco teórico, se envía una por una todas las 274,854 columnas que conforman las características de los signos, y cuando menos una vez cada uno de los 100 elementos de cada columna es probado como una hipótesis h óptima, descartando todas aquellos ejemplos cuyo error de clasificación t sea mayor a 0.5, es decir, peor a un lanzamiento de moneda. De nueva cuenta, al hacer cálculos, la enorme cantidad de procesos comparativos que necesitan realizarse para obtener un solo clasificador débil de los 30 necesarios:

- 274,854 características Haar.
- 100 ejemplos de cada característica Haar, que cuando menos se probaran 1 vez como hipótesis "h".
- 30 clasificadores débiles.
- 100 ejemplos contra los que se comprobará la hipótesis "h".

En general, tenemos aproximadamente 2,748,540,000 comparaciones por ciclo, una cifra interesante si hablamos de cálculos lineales para obtener un solo clasificador t, y 82,456,200,000 si queremos a los 30 clasificadores débiles

que en conjunto forman al clasificador fuerte T. Como habíamos comentado en el capítulo anterior, podríamos utilizar la versión lineal, o no paralelizada del Adaboost para encontrar a los 30 clasificadores débiles y tardar un tiempo lineal "tl". Pero al analizar a fondo este algoritmo, cumple con tres condiciones básicas para ser candidato a una paralelización

1. Cada cálculo es una combinación de operaciones atómicas.

2. Los resultados de cada cálculo son mutuamente excluyentes de otro resultado.

3. No existe orden de precedencia para presentar resultados, siempre y cuando estén completos.

En este trabajo se presentó una versión paralelizada del algoritmo Adaboost, la cual redujo el tiempo de cómputo "tl" drásticamente, y el número de procesos paralelizables depende únicamente del número de núcleos que poseamos en nuestro computador, que para este caso fueron cuatro núcleos. El proceso de paralelización sigue un esquema básico y unas reglas generales dependiendo del número de procesos ligeros o hilos que deseemos para agilizar el cálculo. Supongamos pues que tenemos las 274,854 columnas de características, para realizar un computo paralelo primero dividimos las 274,854 columnas entre el número de procesos ligeros o hilos:

$$columnas/hilos = 274,854/4 = 68713.5000$$

columnas	hilos	elementos por bloque
274,854	4	68713.5000

Tabla 4.2: Columnas y elementos por bloque.

y tomamos la parte entera de la división, entonces el "*incremento general*" es igual a 68,713.

Para saber de que columna a que columna calculara cada hilo a su mejor hipótesis "h", el primer intervalo comenzara en cero, y su límite superior, sera cero más el "*incremento general*", el siguiente intervalo tendrá como límite inferior al anterior límite superior, y el límite superior sera el límite inferior actual más el incremento general, y así sucesivamente hasta completar los "n" bloques para cada hilo, como lo indica la tabla 4.3.

Hilo	límite inferior	límite superior	incremento
1	0	68712	68713
2	68713	137425	68713
3	137426	206138	68713
4	206139	Columnas (274,854)	El último bloque termina en "n".

Tabla 4.3: Bloques por cada hilo.

Salvo en el último hilo, en donde el límite superior toma a la cantidad de columnas o características que estemos utilizando.

Lo que hacemos es en pocas palabras es buscar por cada ronda dentro de los N bloques de columnas, a los N mejores criterios o hipótesis "h", para almacenarlos en una estructura que contendrá tantos candidatos como N número de hilos hayamos ejecutado, y entre todos los candidatos seleccionara al mejor de todos ellos, y ese sera nuestro mejor clasificador débil "t" por cada ronda. La segunda etapa consta de combinar cada una de las opiniones de los "t" clasificadores débiles, para formar un único clasificador fuerte "T" por signo.

Una vez completados los 30 ciclos de búsqueda para los clasificadores débiles, cada uno de ellos es almacenado para comprobar la precisión del sistema en el bloque prueba y para repetir posteriormente el experimento. El almacenamiento cumple con la siguiente regla de ordenamiento que se observa en la tabla 4.4.

fila	valorCriterio	errorCriterio	polaridad	columna	alfa
94	-13692	0.01875	0	37283	1.97882

Tabla 4.4: Almacenamiento de clasificadores.

En donde:

- Fila.- Representa al ejemplo de donde se tomo la característica Haar.
- Valor Criterio.- Es el valor Haar de la característica tomada como clasificador débil "t".
- Error Criterio.- Es el promedio ponderado de ejemplos mal clasificados por el clasificador débil "t".

- Polaridad.- Este valor representa al mismo clasificador débil, pero evaluado de dos maneras. El 0 representan que todos los valores menores o iguales al clasificador etiquetan al valor deseado. En caso contrario el valor 1 representa que todos los valores mayores o iguales al clasificador etiquetan al valor deseado.

- Columna.- Representa la manera de recrear la característica Haar, es decir la coordenada x e y de donde proviene, que tipo de característica es, y las dimensiones de la misma.

- Alfa.- Es la importancia del clasificador débil "t".

De esta forma exportamos todos los clasificadores débiles, que en conjunto forman al clasificador fuerte "T" por cada una de las cinco rondas de entrenamiento-test y tomamos los clasificadores de la ronda con el menor error posible.

4.5.1. Reconocimiento y etiquetado del signo

Para realizar la clasificación, necesitamos recuperar toda la información previamente generada a lo largo del sistema. Como primer paso recuperamos los mejores T clasificadores débiles. El segundo paso es leer todas las combinaciones de características Haar, y como fueron creadas. Al crear cada característica Haar, vertical-horizontal de una y dos barras, y diagonales, cada combinación posible fue almacenada para recrear, futuramente, el valor Haar que necesitemos para las imágenes en tiempo real que ahora toca analizar. Y como tercer paso simplemente asociamos la columna de donde proviene el "t"-ésimo clasificador débil, con la fila que recrea al valor Haar y recreamos nuestro valor Haar para la nueva imagen.

Para la extracción de características Haar en tiempo real, primero aplicamos todo el preprocesamiento a la imagen cruda, recorte de la imagen, centrado, aplicamos morfología matemática, y reescalamos a una escala de 30×30 pixeles, pero ya no calcularemos todo el bloque de características Haar por imagen, solo las 30 que nos indican los clasificadores débiles. El procedimiento que se sigue, es ubicar la forma de la característica Haar y extraer este valor Haar a la imagen cruda en tiempo real.

Con las nuevas características extraídas a la imagen en tiempo real y almacenadas en un vector, mandamos dicho vector a la función signo utilizada que nos devolverá un valor binario comprendido entre -1 o +1. Para esta ultima parte que nos dirá a que signo pertenece la imagen hay que hacer

mención de lo que se envía y se recibe como resultado. Por cada imagen cruda en tiempo real procesada, entran 10 vectores, uno por cada letra, y cada vector contiene las 30 características Haar extraídas igualmente en tiempo real. Cada uno de esos vectores corresponde a su debido conjunto de clasificadores T. Para la salida de este experimento, nos enfocamos en dos versiones de criterios para obtener un resultado que en apariencia parece el mismo, pero con diferencia en exactitud, y estas dos versiones corresponden a los criterios estricto, y relajado.

4.5.2. Criterios de reconocimiento.

Para el **criterio estricto**, una vez recibidos los 10 vectores de valores Haar, son calculados de forma binaria los resultados de la función signo, y arrojando un **1** cuando los valores Haar activan la función con un resultado igual o mayor a cero, y arrojando un **0** cuando los valores Haar quedan por debajo del umbral "0". Ahora bien, en un criterio estricto, solo un conjunto de valores Haar debe activar la función signo, dando como resultado una sumatoria de valores, multiplicados por la importancia de su criterio un valor positivo mayor o igual a cero, uno y solo uno debe activarse. En caso de activarse más de un signo, o de obtener dos o más resultados positivos binarios, el algoritmo estrictamente desconoce el signo y envía un signo de interrogación para no cometer errores de etiquetado. Este criterio, tiene un número prácticamente nulo de falsos positivos, pero por otra parte, tenemos muchos signos que no son etiquetados ni de manera correcta o incorrecta simplemente son ignorados.

Para el **criterio relajado**, igualmente recibimos 10 vectores de valores Haar, pero utilizamos entonces los valores continuos de la función signo, así como la letra a la que pertenecen y los almacenamos en un vector. Después de calculados todos los valores para los 10 vectores, ordenamos de manera descendente estos valores, y tomamos el valor de la función signo cuyo valor sea el más alto. Esto nos indica en cuestión de semejanza que signo se parece más a la imagen cruda en tiempo real, y para efectos prácticos siempre obtenemos la respuesta con mayor similitud a la imagen cruda. Este criterio, no es infalible obviamente, pero siempre garantiza el mejor resultado con respecto a la imagen analizada y nos da un resultado en cada análisis, no dejando sin etiqueta ninguna imagen cruda.

Después de escoger el criterio que más nos resulte útil para el sistema, en nuestro caso utilizamos el criterio relajado, el gesto es identificado, y una símbolo alfanumérico es mostrado en pantalla en tiempo real, concluyendo

la última etapa de etiquetado, y el sistema en si.

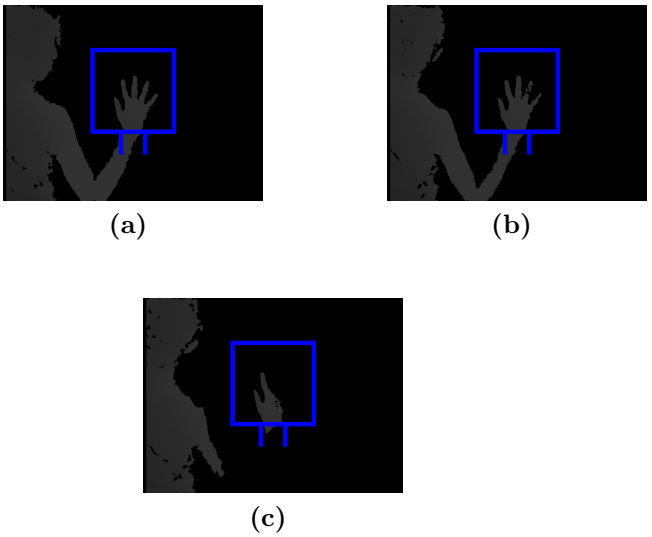
Capítulo 5

Resultados

En este capítulo se presentan los resultados del reconocimiento de cada una de las señas alfanuméricas seleccionadas para este trabajo de reconocimiento de la Lengua de Señas Mexicana utilizando características Haar 2D y 3D. Así mismo, se presentan los resultados obtenidos del preprocesamiento de las imágenes.

5.1. Tratamiento digital de imágenes

En la figura 5.1 se puede apreciar un ejemplo de captura de las tres profundidades con el sensor kinect de una de las 10 señas del LSM.



(a) **(b)**

(c)

Figura 5.1: Imágenes a profundidad: a) Imagen a una profundidad de 90 cm, b) Imagen a una profundidad de 87.5 cm, c) Imagen a una profundidad de 86 cm

En la figura 5.2 se muestran los resultados del preprocesamiento para las señas del LSM.

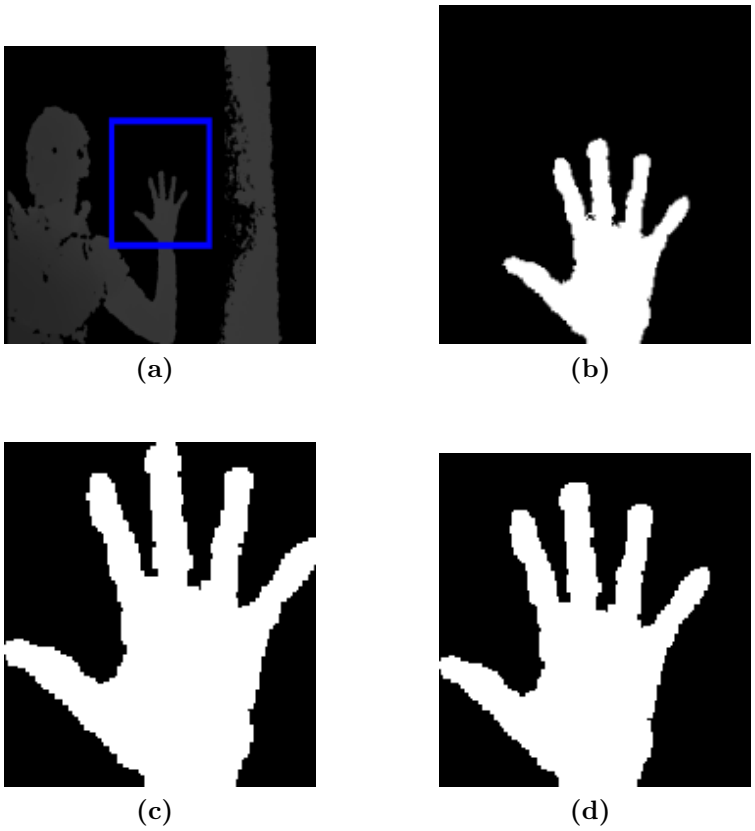


Figura 5.2: a) Imagen en profundidad original, b) Imagen de la mano recortada binarizada sin morfologia, c) Imagen de la mano recortada de forma exacta, d) Imagen de la mano centrada.

El proceso de mejora digital con morfología matemática queda como se muestra en la figura 5.3 de arriba a abajo, y de izquierda a derecha, representa una imagen cruda en tiempo real, la erosión de la imagen, su posterior dilatación, y finalmente una imagen con la combinación de estos dos operadores. El resultado es una imagen de 30 $\times$ 30 pixeles con apertura morfológica.

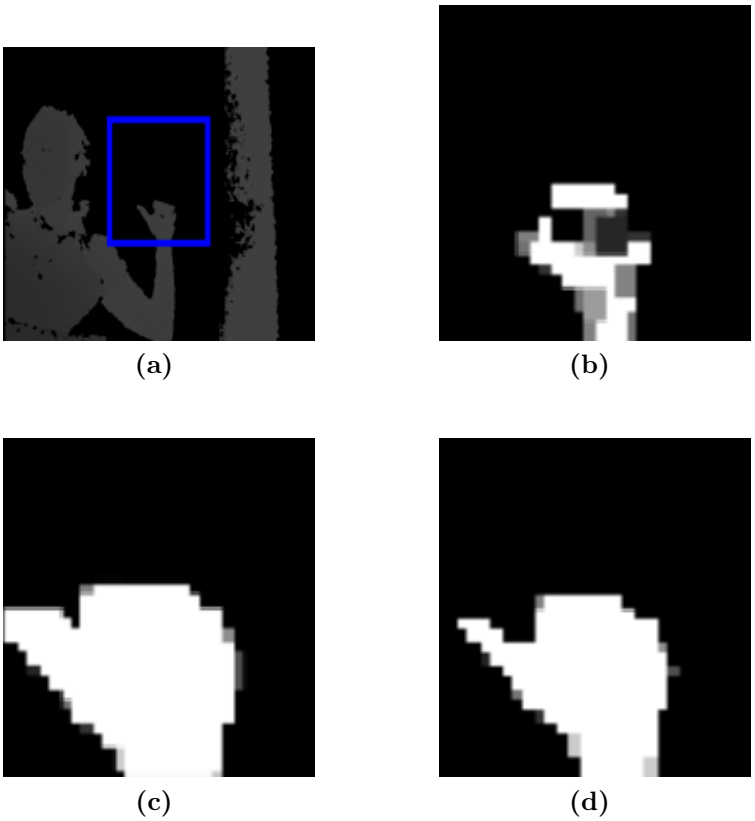


Figura 5.3: a) Imagen en profundidad original, b) Erosión morfológica, c) Dilatación morfológica, d) Apertura morfológica.

5.2. Base de datos

Al utilizar el algoritmo de Adaboost, requerimos un bloque de ejemplos positivos y negativos. Como la naturaleza del algoritmo procesa los datos es de forma binaria, se utilizan únicamente dos tipos de valores -1 y +1. En nuestro caso los ejemplos positivos (+1) serán el signo a identificar, y los ejemplos negativos serán todo el conjunto de signos diferentes al signo en cuestión a identificar (-1).

Para demostrar la efectividad del sistema es necesario ponerlo a prueba mediante la validación cruzada que es una técnica, como su nombre lo indica, para validar un sistema cuando no se disponen suficientes ejemplos. Para llevar a cabo estas pruebas se divide todo el conjunto de ejemplos en subgrupos, los cuales en cada etapa de validación serán excluidos uno a uno, tomándolos como grupos de prueba o "test", y entrenando con el resto de ejemplos o grupo "train", hasta completar las N pruebas dependiendo de los N bloques

en los que se ha dividido es espacio muestra total de ejemplos. Para nuestra pruebas de validación cruzada dividimos entonces todos los ejemplos en dos segmentos, 80 % para el bloque de entrenamiento y 20 % para el bloque prueba como se ve en la figura 5.4, realizando un total de cinco rondas de validación cruzada por cada signo, y en total 50 rondas de validación cruzada para validar los 10 signos.

Figura 5.4: Base de datos del sistema.

Para tener un punto de comparación acerca de las características Haar 2D contra las características Haar 3D se utilizaron dos bases de datos con el mismo esquema de validación cruzada.

La primera base de datos es la de características Haar 2D únicamente, que cuenta con 100 ejemplos de cada letra con 45,809 valores Haar, pero utilizando únicamente la tercera profundidad de cada tercia de imágenes, es decir la profundidad que mayor información aporta a cada gesto.

La segunda base de datos es la de características Haar 2D únicamente, consta de 100 ejemplos de cada letra con 274,854 valores Haar, pero utilizando todas las distancias profundidad de cada tercia de imágenes, y 3 combinaciones de ellas por cada gesto.

5.3. Reconocimiento de señas alfanuméricas

En esta sección se muestran los resultados de reconocimiento de cada una de las letras, primero la tabla con las medidas de verosimilitud, seguida de el top 5 de las mejores plantillas Haar utilizadas para cada letra. Finalmente la gráfica correspondiente al porcentaje total de reconocimiento de todo el sistema.

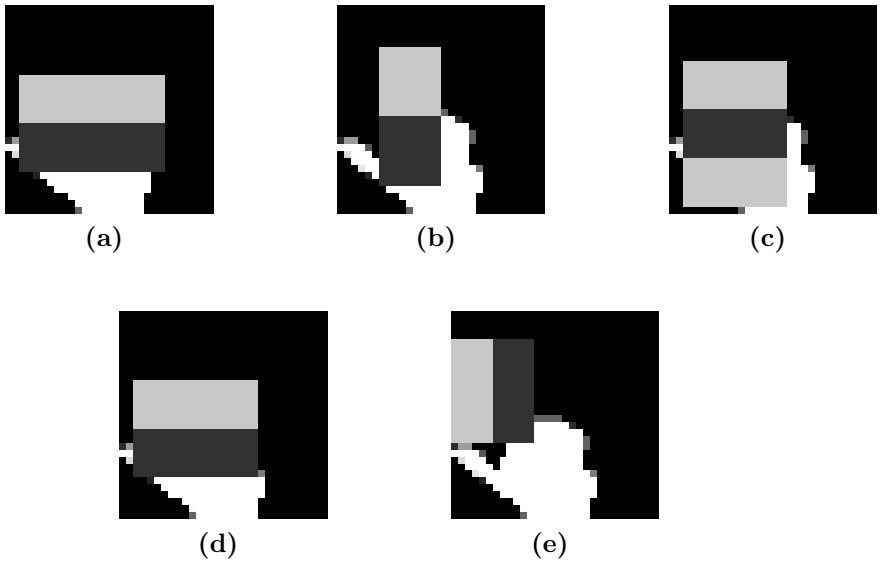


Figura 5.5: Top Cinco de mejores plantillas para la letra A: a) Haar 2 barras horizontal de 14 filas × 21 columnas, b) Haar 2 barras horizontal de 20 filas × 9 columnas, c) Haar 3 barras horizontal de 21 filas × 15 columnas, d) Haar 2 barras horizontal de 14 filas × 18 columnas, e) Haar 2 barras vertical de 15 filas × 12 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	92.50	90.00	95.00	92.31	94.74	90.00
2	97.50	100.00	95.00	97.56	95.24	100.00
3	97.50	100.00	95.00	97.56	95.24	100.00
4	95.00	90.00	100.00	94.74	100.00	90.00
5	100.00	100.00	100.00	100.00	100.00	100.00

Tabla 5.1: Reconocimiento de la letra **A** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	95.00	95.00	95.00	95.00	95.00
2	92.50	100.00	85.00	93.02	86.96	100.00
3	100.00	100.00	100.00	100.00	100.00	100.00
4	95.00	90.00	100.00	94.74	100.00	90.00
5	97.50	100.00	95.00	97.56	95.24	100.00

Tabla 5.2: Reconocimiento de la letra **A** con caracteristicas Haar **3D**.

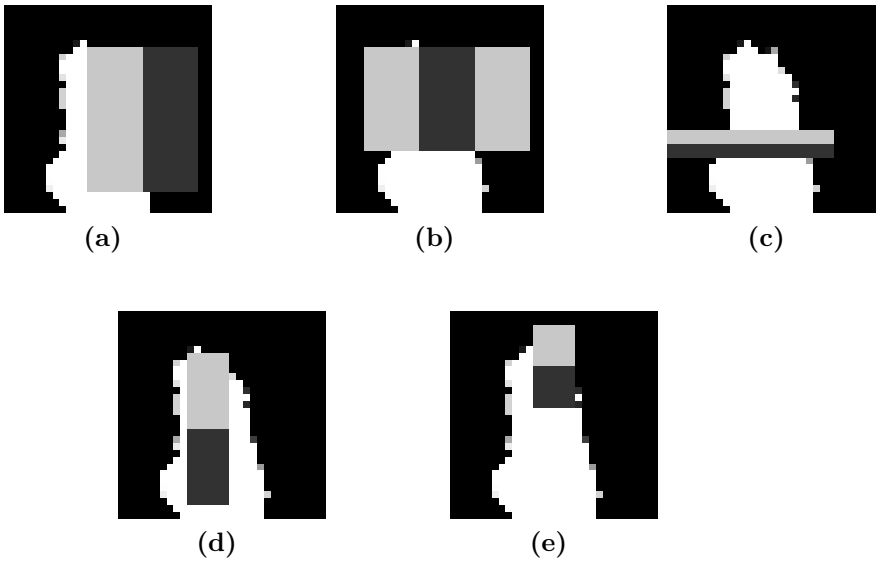


Figura 5.6: Top Cinco de mejores plantillas para la letra B: a) Haar 2 barras vertical de 21 filas × 16 columnas, b) Haar 3 barras vertical de 15 filas × 24 columnas, c) Haar 2 barras horizontal de 4 filas × 24 columnas, d) Haar 2 barras horizontal de 22 filas × 6 columnas, e) Haar 2 barras vertical de 12 filas × 6 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	90.00	100.00	94.74	100.00	90.00
2	95.00	90.00	100.00	94.74	100.00	90.00
3	95.00	100.00	90.00	95.24	90.91	100.00
4	82.50	100.00	65.00	85.11	74.07	100.00
5	97.50	95.00	100.00	97.44	100.00	95.00

Tabla 5.3: Reconocimiento de la letra **B** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	95.00	95.00	95.00	95.00	95.00
2	95.00	90.00	100.00	94.74	100.00	90.00
3	100.00	100.00	100.00	100.00	100.00	100.00
4	95.00	100.00	90.00	95.24	90.91	100.00
5	100.00	100.00	100.00	100.00	100.00	100.00

Tabla 5.4: Reconocimiento de la letra **B** con caracteristicas Haar **3D**.

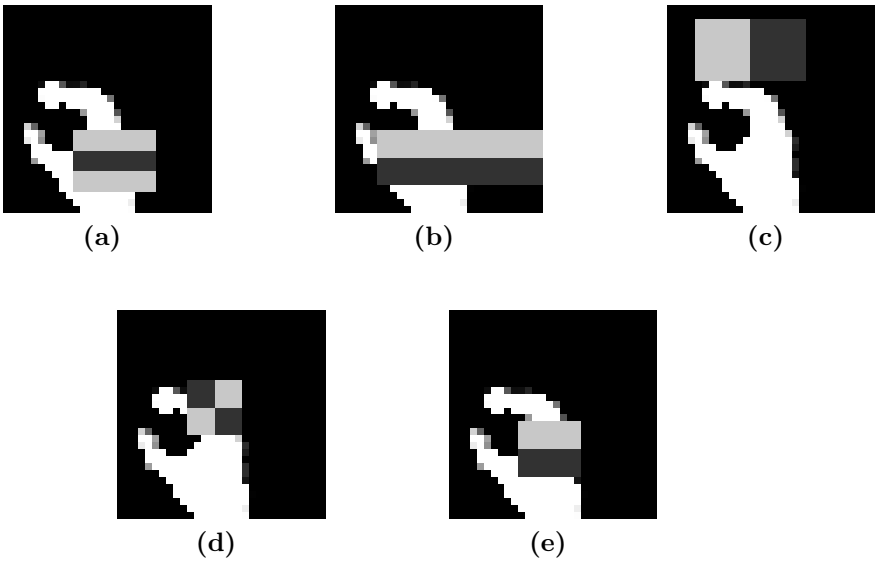


Figura 5.7: Top Cinco de mejores plantillas para la letra C: a) Haar 3 barras horizontal de 9 filas × 12 columnas, b) Haar 2 barras horizontal de 8 filas × 24 columnas, c) Haar 2 barras vertical de 9 filas × 16 columnas, d) Haar 4 barras diagonal de 8 filas × 8 columnas, e) Haar 2 barras horizontal de 8 filas × 9 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	90.00	100.00	94.74	100.00	90.00
2	95.00	95.00	95.00	95.00	95.00	95.00
3	97.50	100.00	95.00	97.56	95.24	100.00
4	95.00	100.00	90.00	95.24	90.91	100.00
5	92.50	95.00	90.00	92.68	90.48	95.00

Tabla 5.5: Reconocimiento de la letra **C** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	90.00	100.00	94.74	100.00	90.00
2	100.00	100.00	100.00	100.00	100.00	100.00
3	97.50	95.00	100.00	97.44	100.00	95.00
4	95.00	95.00	95.00	95.00	95.00	95.00
5	95.00	100.00	90.00	95.24	90.91	100.00

Tabla 5.6: Reconocimiento de la letra **C** con caracteristicas Haar **3D**.

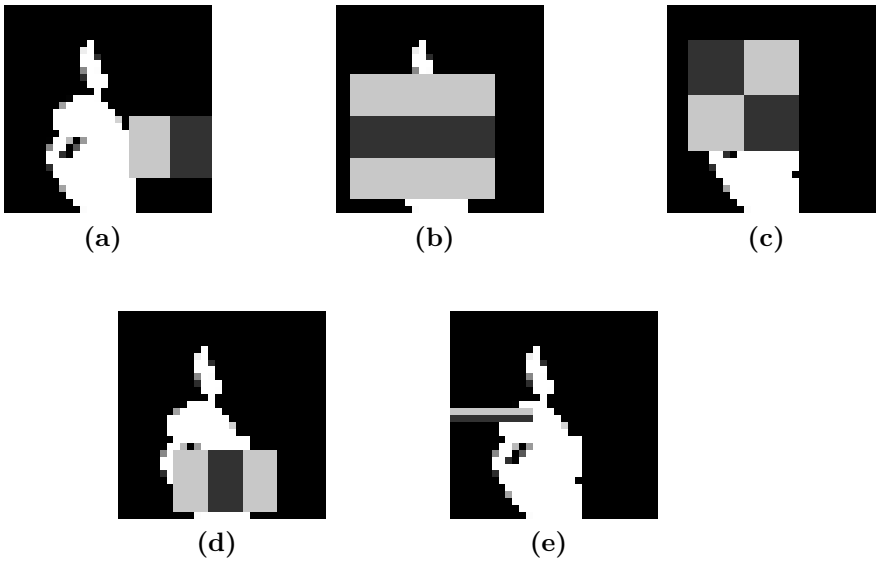


Figura 5.8: Top Cinco de mejores plantillas para la letra D: a) Haar 2 barras vertical de 9 filas × 12 columnas, b) Haar 3 barras horizontal de 18 filas × 21 columnas, c) Haar 4 barras diagonal de 16 filas × 16 columnas, d) Haar 3 barras vertical de 9 filas × 10 columnas, e) Haar 2 barras horizontal de 2 filas × 12 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	95.00	100.00	97.44	100.00	95.00
2	92.50	100.00	85.00	93.02	86.96	100.00
3	95.00	95.00	95.00	95.00	95.00	95.00
4	100.00	100.00	100.00	100.00	100.00	100.00
5	90.00	90.00	90.00	90.00	90.00	90.00

Tabla 5.7: Reconocimiento de la letra **D** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	100.00	100.00	100.00	100.00	100.00	100.00
2	92.50	95.00	90.00	92.68	90.48	95.00
3	92.50	85.00	100.00	91.89	100.00	85.00
4	97.50	100.00	95.00	97.56	95.24	100.00
5	97.50	100.00	95.00	97.56	95.24	100.00

Tabla 5.8: Reconocimiento de la letra **D** con caracteristicas Haar **3D**.

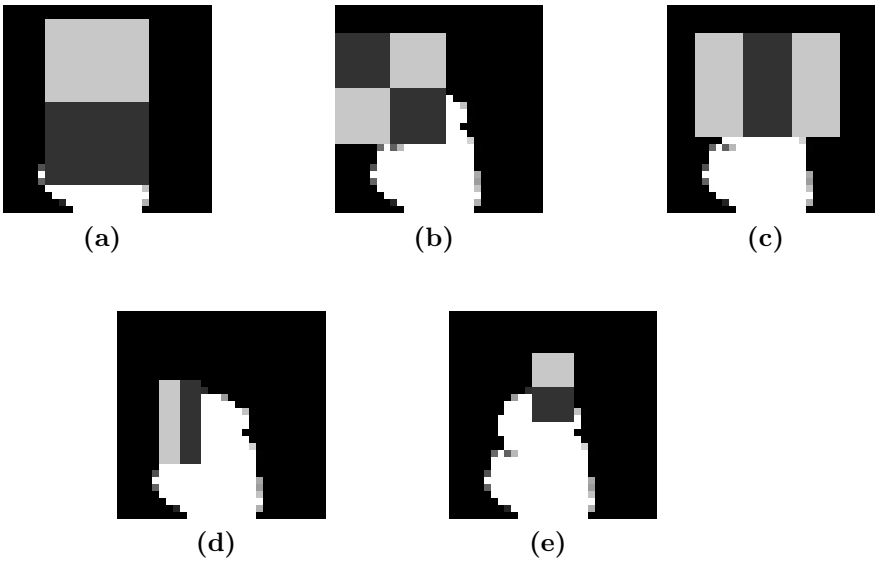


Figura 5.9: Top Cinco de mejores plantillas para la letra E: a) Haar 2 barras horizontal de 24 filas × 15 columnas, b) Haar 4 barras diagonal de 16 filas × 16 columnas, c) Haar 3 barras vertical de 15 filas × 14 columnas, d) Haar 2 barras vertical de 12 filas × 6 columnas, e) Haar 2 barras horizontal de 10 filas × 6 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	100.00	95.00	97.56	95.24	100.00
2	95.00	100.00	90.00	95.24	90.91	100.00
3	95.00	95.00	95.00	95.00	95.00	95.00
4	90.00	85.00	95.00	89.47	94.44	85.00
5	85.00	75.00	95.00	83.33	93.75	75.00

Tabla 5.9: Reconocimiento de la letra **E** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	100.00	100.00	100.00	100.00	100.00	100.00
2	95.00	100.00	90.00	95.24	90.91	100.00
3	95.00	95.00	95.00	95.00	95.00	95.00
4	87.50	85.00	90.00	87.18	89.47	85.00
5	80.00	75.00	85.00	78.95	83.33	75.00

Tabla 5.10: Reconocimiento de la letra **E** con caracteristicas Haar **3D**.

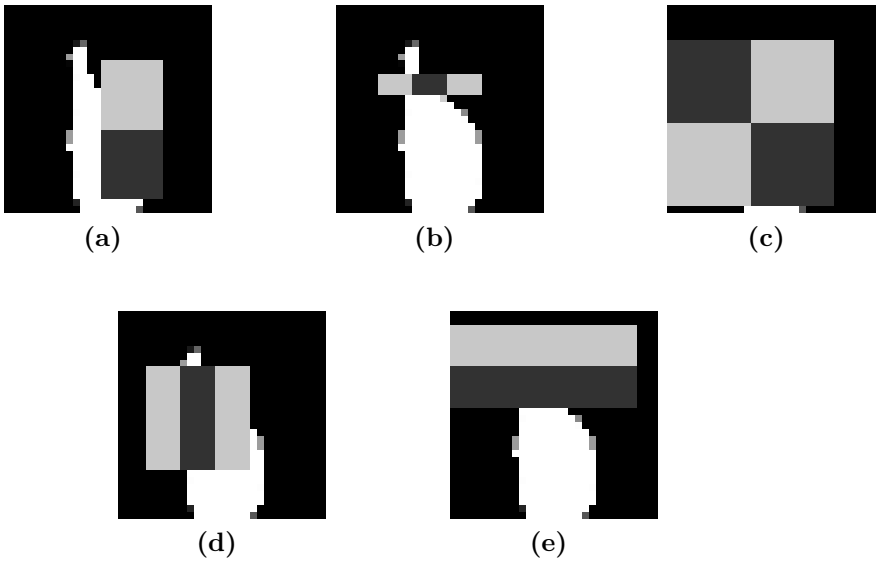


Figura 5.10: Top Cinco de mejores plantillas para el número Uno: a) Haar 2 barras horizontal de 20 filas × 9 columnas, b) Haar 3 barras vertical de 3 filas × 10 columnas, c) Haar 4 barras diagonal de 24 filas × 24 columnas, d) Haar 3 barras vertical de 15 filas × 15 columnas, e) Haar 2 barras horizontal de 12 filas × 27 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	100.00	95.00	97.56	95.24	100.00
2	100.00	100.00	100.00	100.00	100.00	100.00
3	95.00	95.00	95.00	95.00	95.00	95.00
4	90.00	80.00	100.00	88.89	100.00	80.00
5	97.50	100.00	95.00	97.56	95.24	100.00

Tabla 5.11: Reconocimiento del número **Uno** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	100.00	95.00	97.56	95.24	100.00
2	100.00	100.00	100.00	100.00	100.00	100.00
3	92.50	100.00	85.00	93.02	86.96	100.00
4	82.50	65.00	100.00	78.79	100.00	65.00
5	97.50	95.00	100.00	97.44	100.00	95.00

Tabla 5.12: Reconocimiento del número **Uno** con caracteristicas Haar **3D**.

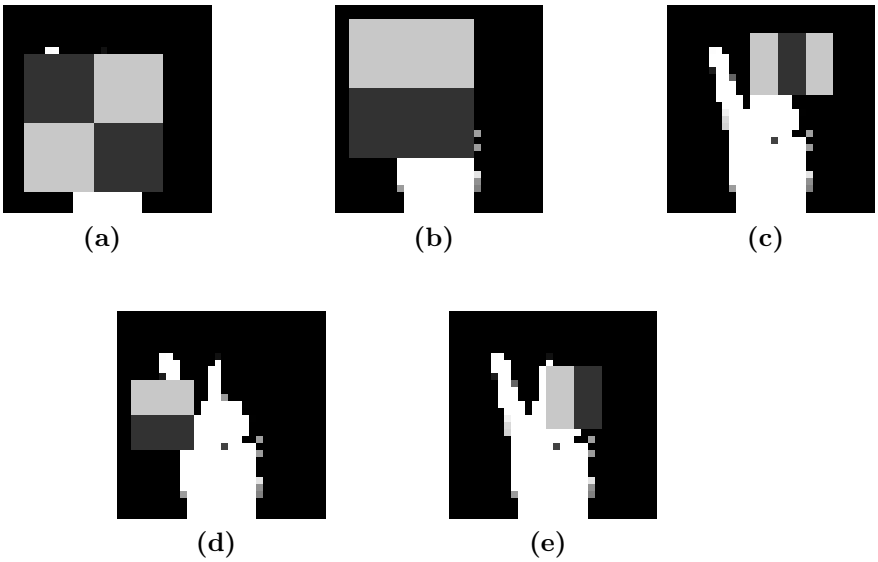


Figura 5.11: Top Cinco de mejores plantillas para el número Dos: a) Haar 4 barras diagonal de 20 filas × 20 columnas, b) Haar 2 barras horizontal de 20 filas × 18 columnas, c) Haar 3 barras vertical de 9 filas × 8 columnas, d) Haar 3 barras horizontal de 15 filas × 9 columnas, e) Haar 2 barras vertical de 9 filas × 8 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	95.00	100.00	97.44	100.00	95.00
2	87.50	95.00	80.00	88.37	82.61	95.00
3	85.00	80.00	90.00	84.21	88.89	80.00
4	100.00	100.00	100.00	100.00	100.00	100.00
5	92.50	95.00	90.00	92.68	90.48	95.00

Tabla 5.13: Reconocimiento del número **Dos** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	95.00	100.00	97.44	100.00	95.00
2	85.00	95.00	75.00	86.36	79.17	95.00
3	92.50	95.00	90.00	92.68	90.48	95.00
4	100.00	100.00	100.00	100.00	100.00	100.00
5	87.50	95.00	80.00	88.37	82.61	95.00

Tabla 5.14: Reconocimiento del número **Dos** con caracteristicas Haar **3D**.

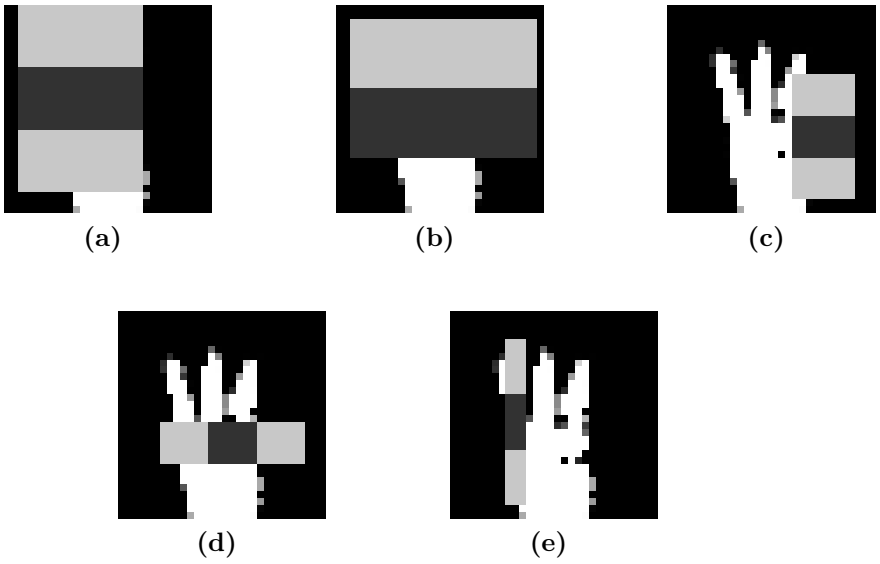


Figura 5.12: Top Cinco de mejores plantillas para el número Tres: a) Haar 3 barras horizontal de 27 filas × 18 columnas, b) Haar 2 barras horizontal de 20 filas × 27 columnas, c) Haar 3 barras horizontal de 18 filas × 9 columnas, d) Haar 3 barras vertical de 6 filas × 21 columnas, e) Haar 3 barras horizontal de 24 filas × 3 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	95.00	100.00	97.44	100.00	95.00
2	92.50	100.00	85.00	93.02	86.96	100.00
3	87.50	95.00	80.00	88.37	82.61	95.00
4	85.00	85.00	85.00	85.00	85.00	85.00
5	85.00	90.00	80.00	85.71	81.82	90.00

Tabla 5.15: Reconocimiento del número **Tres** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	95.00	95.00	95.00	95.00	95.00
2	92.50	100.00	85.00	93.02	86.96	100.00
3	87.50	85.00	90.00	87.18	89.47	85.00
4	97.50	100.00	95.00	97.56	95.24	100.00
5	92.50	95.00	90.00	92.68	90.48	95.00

Tabla 5.16: Reconocimiento del número **Tres** con caracteristicas Haar **3D**.

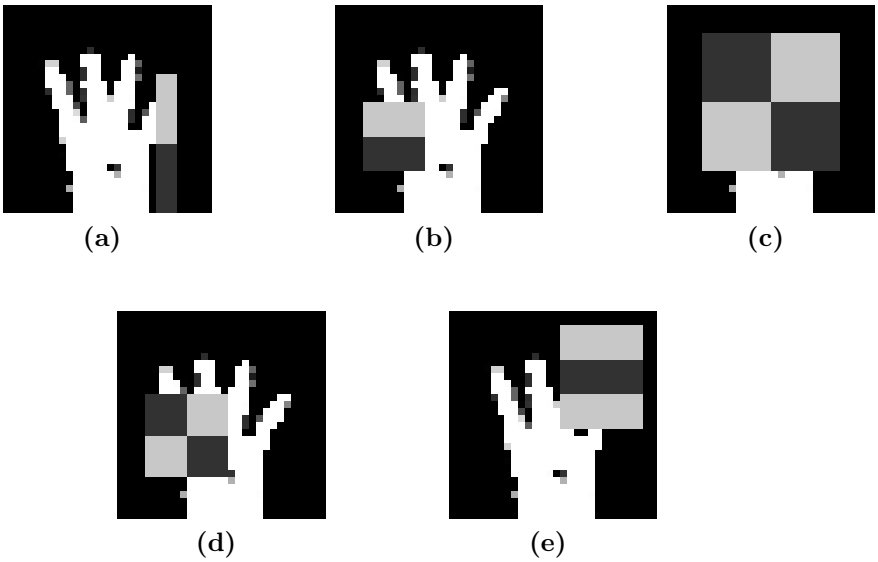


Figura 5.13: Top Cinco de mejores plantillas para el número Cuatro: a) Haar 2 barras horizontal de 20 filas × 3 columnas, b) Haar 2 barras horizontal de 10 filas × 9 columnas, c) Haar 4 barras diagonal de 20 filas × 20 columnas, d) Haar 4 barras diagonal de 12 filas × 12 columnas, e) Haar 3 barras horizontal de 15 filas × 12 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	92.50	100.00	85.00	93.02	86.96	100.00
2	85.00	85.00	85.00	85.00	85.00	85.00
3	92.50	95.00	90.00	92.68	90.48	95.00
4	90.00	85.00	95.00	89.47	94.44	85.00
5	95.00	100.00	90.00	95.24	90.91	100.00

Tabla 5.17: Reconocimiento del número **Cuatro** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	95.00	95.00	95.00	95.00	95.00	95.00
2	87.50	85.00	90.00	87.18	89.47	85.00
3	92.50	95.00	90.00	92.68	90.48	95.00
4	92.50	90.00	95.00	92.31	94.74	90.00
5	92.50	100.00	85.00	93.02	86.96	100.00

Tabla 5.18: Reconocimiento del número **Cuatro** con caracteristicas Haar **3D**.

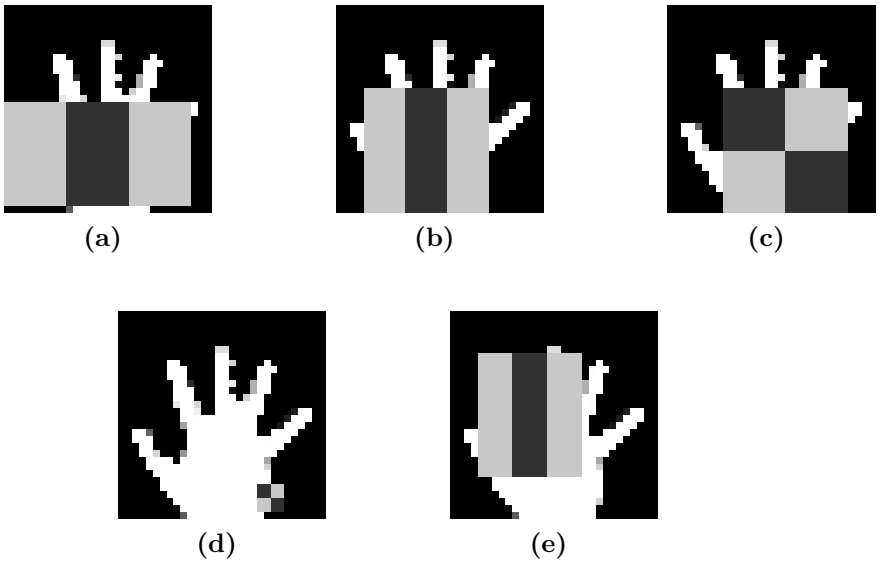


Figura 5.14: Top Cinco de mejores plantillas para el número Cinco: a) Haar 3 barras vertical de 15 filas × 27 columnas, b) Haar 3 barras vertical de 18 filas × 18 columnas, c) Haar 4 barras diagonal de 18 filas × 18 columnas, d) Haar 4 barras diagonal de 4 filas × 4 columnas, e) Haar 3 barras vertical de 18 filas × 15 columnas.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	100.00	100.00	100.00	100.00	100.00	100.00
2	100.00	100.00	100.00	100.00	100.00	100.00
3	97.50	95.00	100.00	97.44	100.00	95.00
4	97.50	95.00	100.00	97.44	100.00	95.00
5	95.00	95.00	95.00	95.00	95.00	95.00

Tabla 5.19: Reconocimiento del número **Cinco** con caracteristicas Haar **2D**.

Fold	Rec	Sens	Spec	F1score	Prec	Recall
1	97.50	100.00	95.00	97.56	95.24	100.00
2	95.00	100.00	90.00	95.24	90.91	100.00
3	100.00	100.00	100.00	100.00	100.00	100.00
4	97.50	95.00	100.00	97.44	100.00	95.00
5	95.00	95.00	95.00	95.00	95.00	95.00

Tabla 5.20: Reconocimiento del número **Cinco** con caracteristicas Haar **3D**.

5.3.1. Gráficas de las medidas de verosimilitud

En esta sección se presentan las gráficas de los resultados ponderados en las medidas de verosimilitud de todo el sistema. En color rojo se presentan los resultados utilizando únicamente características Haar 2D, y en color verde se presentan los resultados de las características Haar 3D.

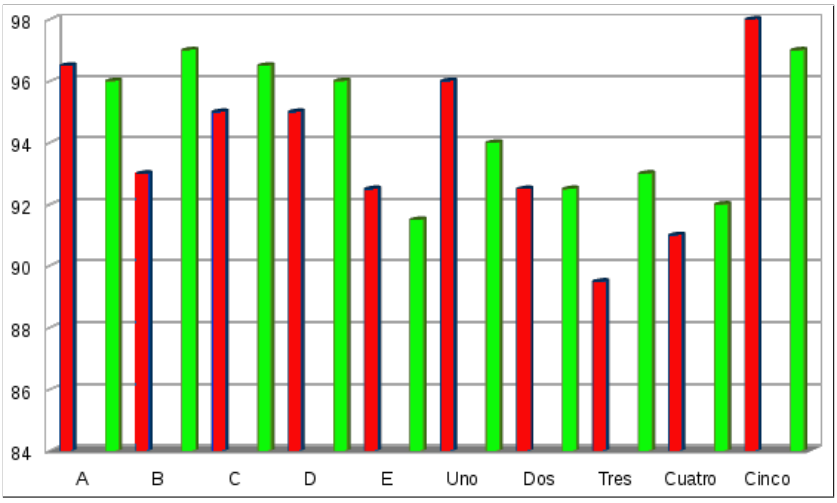


Figura 5.15: Grafica del porcentaje de reconocimiento general en 2D y 3D.

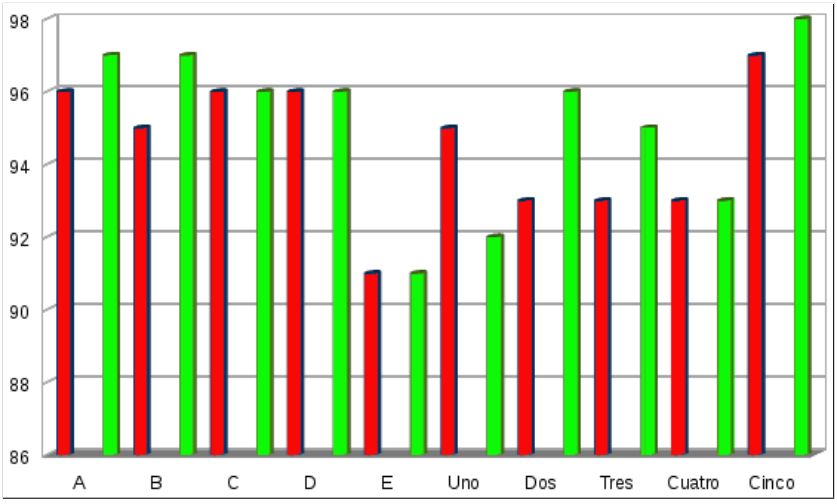


Figura 5.16: Sensibilidad, tasa de verdaderos positivos en 2D y 3D.

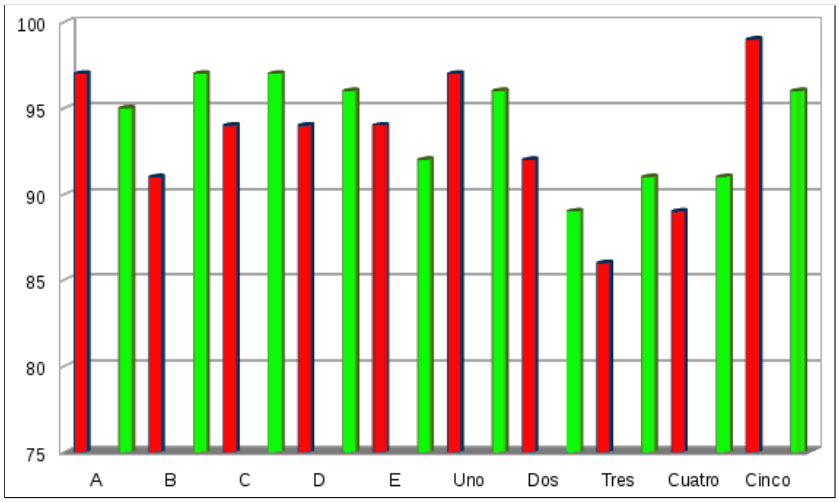


Figura 5.17: Especificidad, tasa de verdaderos negativos en 2D y 3D.

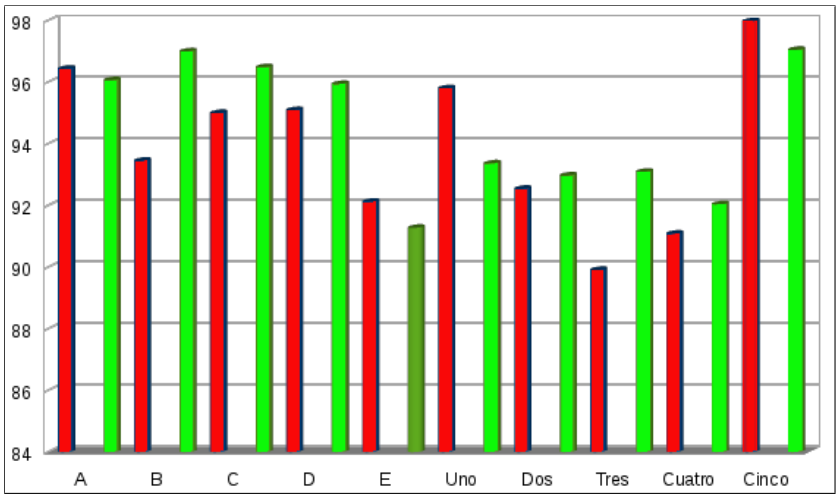


Figura 5.18: F1Score, exactitud del sistema en 2D y 3D.

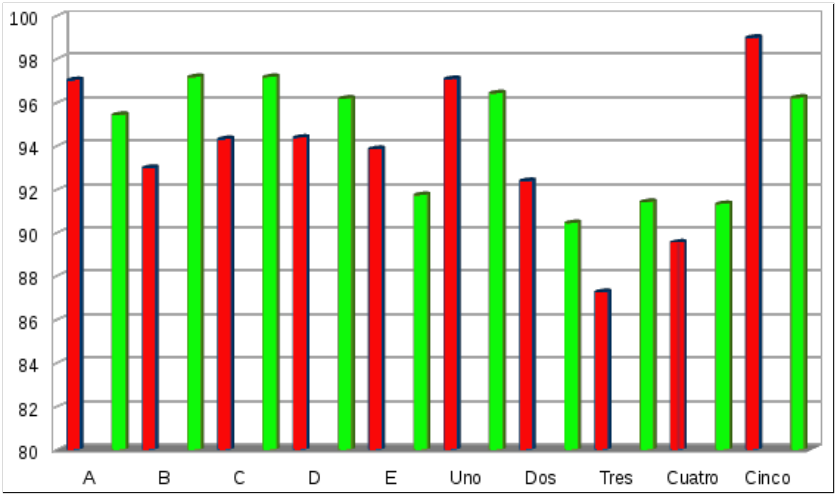


Figura 5.19: Precisión, valores positivos predichos en 2D y 3D.

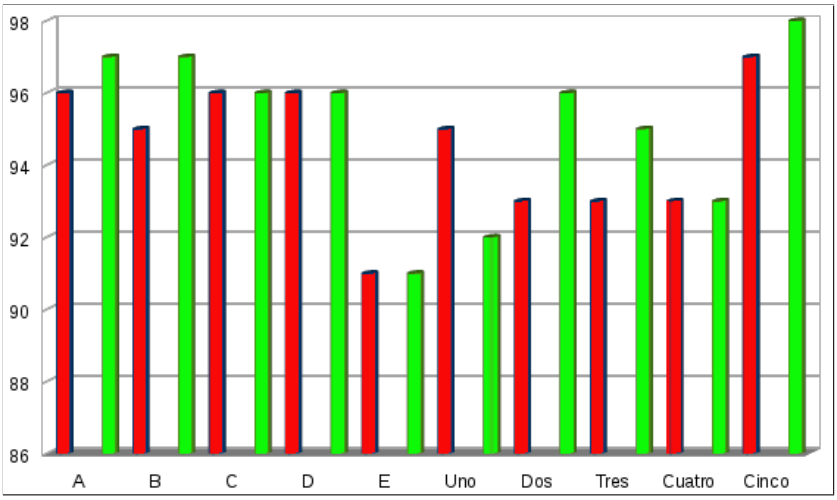
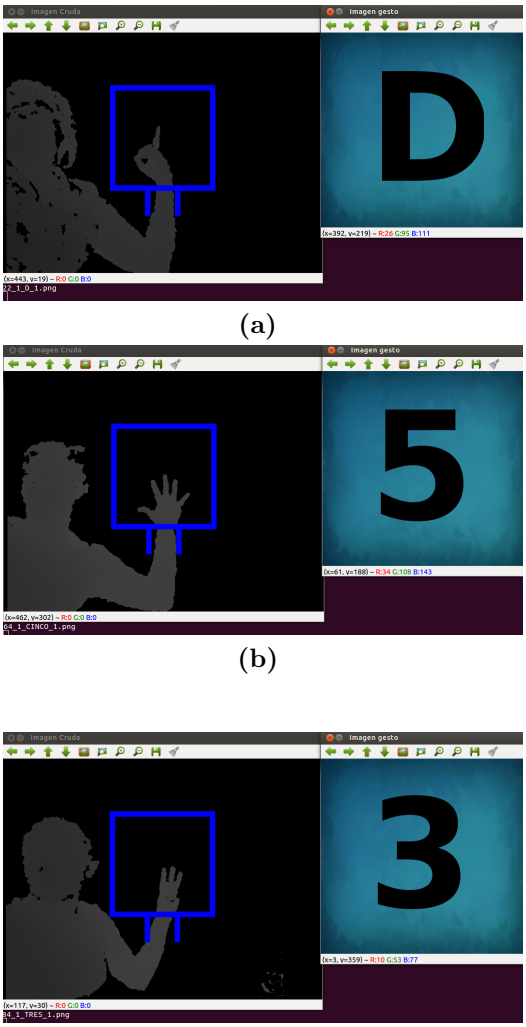


Figura 5.20: Recall, instancias relevantes recuperadas en 2D y 3D.

Y unas imágenes del programa reconociendo gestos en tiempo real como se observa en la figura 5.21.



(a)

(b)

(c)

Figura 5.21: Reconocimiento de imágenes en tiempo real, la imagen en blanco y negro es la imagen cruda, y la imagen en color verde es el significado alfanumérico del signo reconocido. a) Reconocimiento de la letra D en tiempo real, b) Reconocimiento del numero Cinco en tiempo real, c) Reconocimiento del numero Tres en tiempo real.

Si observamos bien las gráficas finales de la figura 5.22 de los resultados obtenidos en las tablas en este trabajo se ha demostrado con los resultados que las características Haar en 3D, tomadas a distintas profundidades a una tercia de imágenes son superiores a un conjunto de características Haar 2D, teniendo el sistema en conjunto de casi un 95 % de precisión al momento de reconocer un gesto. Por otra parte la versión paralelizable de AdaBoost que reduce enormemente el tiempo de computo al entrenar este algoritmo, lo cual

representa una gran simplificación del trabajo y depende únicamente de los núcleos que posea nuestro procesador.

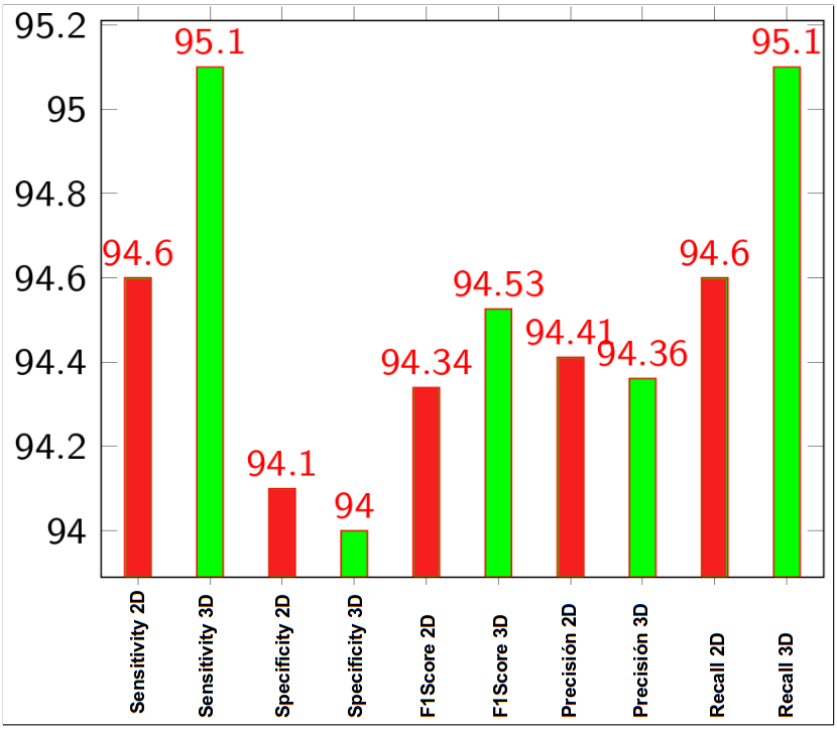


Figura 5.22: Graficas de resultados finales en 2D y 3D.

5.4. Discusión

Durante la etapa de entrenamiento, y posterior a este, es decir en la etapa de reconocimiento, en algunas ocasiones el porcentaje de identificación de las letras bajo por algunos momentos, y esto nos lleva a realizar la pregunta natural del por que en algunas letras el desempeño simplemente baja, y en algunas es muy superior al promedio. Observemos por ejemplo el caso del número tres con la letra B como se observa en la figura 5.23, en este caso ocurre un falso positivo favor de la letra B, en donde el gesto es incorrectamente catalogado como la letra B, cuando en realidad se trata del numero tres.

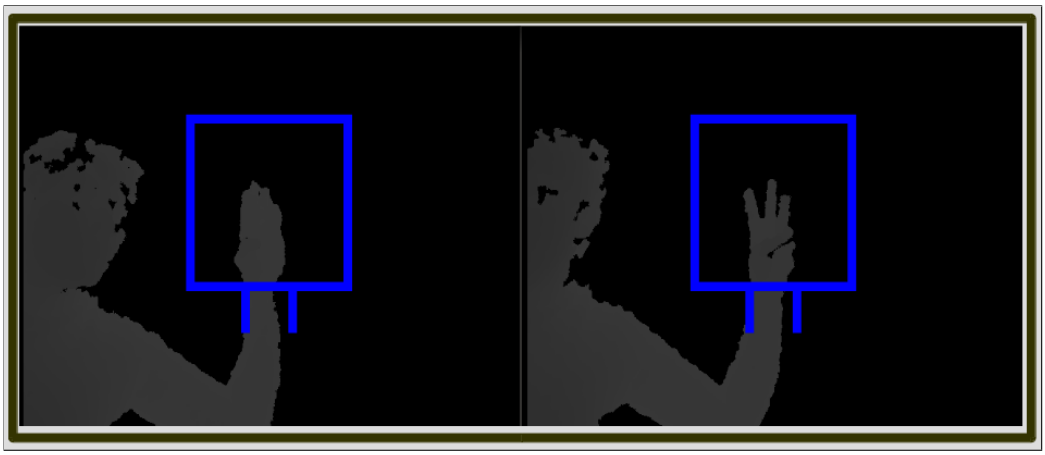


Figura 5.23: Error de etiquetado entre la letra B y el número Tres.

De igual forma, el número tres puede confundirse fácilmente con el número dos, y varios falsos positivos fueron precisamente los que indicaban al número tres con el número dos, y la razón que podemos encontrar es que la diferencia básica entre ambos gestos es solo un dedo, y al momento de reescalar las imágenes, la diferencia en pixeles suele ser muy poca como podemos observar en la figura 5.24.

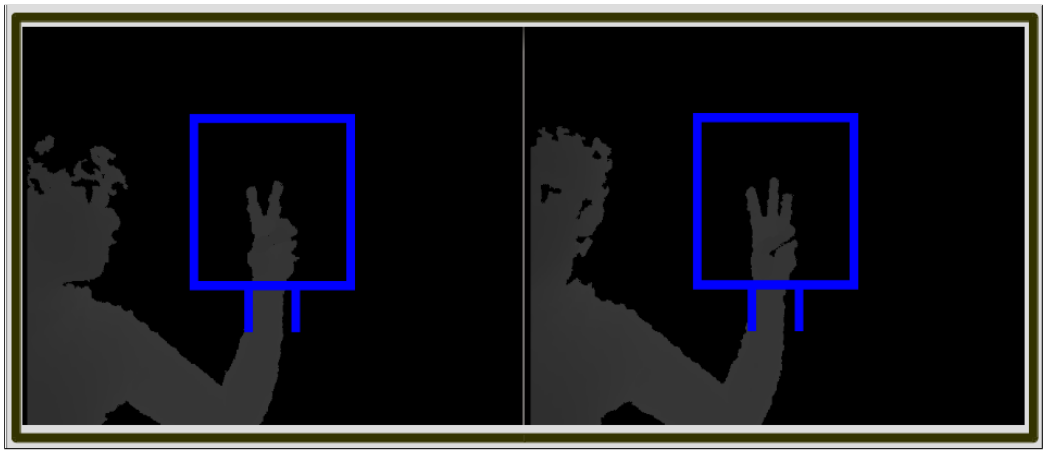


Figura 5.24: Error de etiquetado entre el número Dos y el número Tres.

Otro falso positivo que salta a la vista de manera mas evidente es la mal interpretación del número uno con la letra D, puesto que ambos, aunque se encuentran en diferentes ángulos al momento de hacer la captura, muchas veces parecen estar representando el mismo gesto como se observa en la figura 5.25:

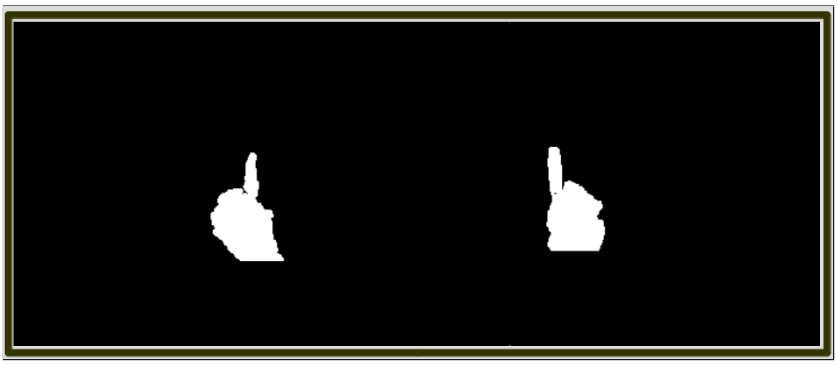


Figura 5.25: Error de etiquetado letra D y el número Uno.

Tampoco podemos pasar por alto que aunque cada seña tiene sus particularidades que la hacen única para diferenciarla del resto de letras del alfabeto del LSM, al momento de realizar algunas señas, la varianza para una misma seña se noto en una etapa posterior al de la captura. Este caso se dio especialmente en la letra C, y una razón probable de la no uniformidad al gesticular esta seña podría ser el la posición de la muñeca al realizar la seña, y la abertura entre los cuatro dedos superiores, y el dedo gordo de la mano como se observa en la figura 5.26:

Figura 5.26: Error varianza de la letra C.

Y finalmente los errores propios de una captura binaria como son por ejemplo pliegues en los dedos, que obviamente no existen en las manos, algunas zonas erosionadas antes de la morfología matemática, así como posiciones en los dedos fuera de lo normal por parte de los participantes, que hacen que la varianza en los signos aumente considerablemente como se observa en la figura 5.27:

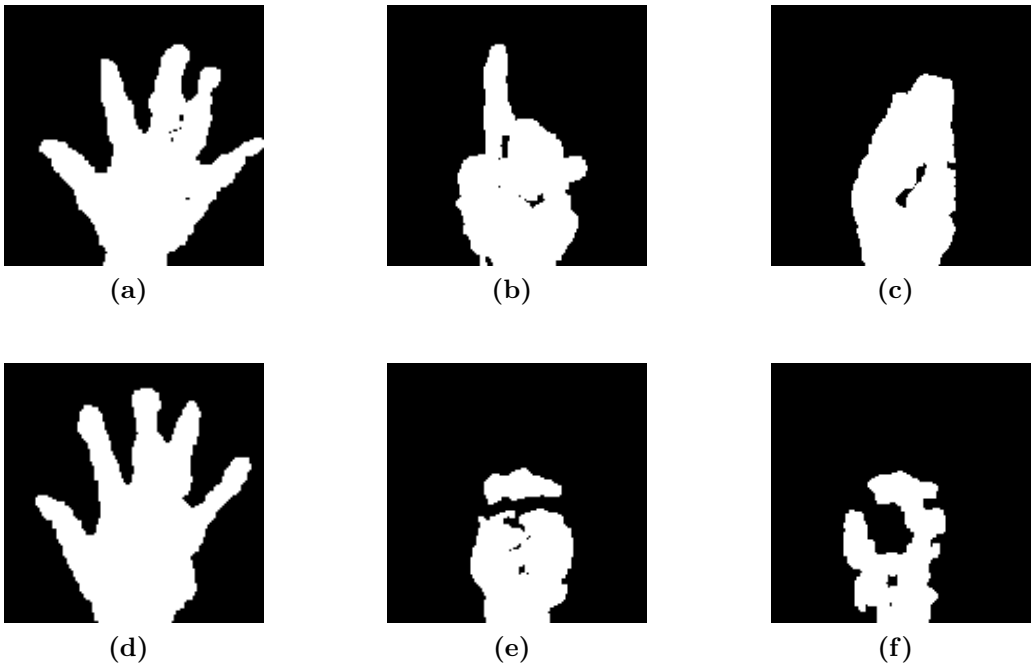


Figura 5.27: Errores comunes ocurridos durante el proceso de captura: a) Error de falanges sobrepuestas, b) Error de protuberancias inexistentes, c) Error de mala inclinación de la mano, d) Error de pliegues inexistentes, e) Error de áreas inconexas, e) Error de mala formación del signo.

Capítulo 6

Conclusiones

En el presente capítulo se resumen todas las experiencias y observaciones obtenidas a lo largo de todo este proyecto de tesis. Las ideas aportadas, así como los problemas encontrados al momento de realizar el proyecto y sus soluciones, esperando que resulte de utilidad la metodología planteada a lo largo de todo este camino para futuras investigaciones, o para ampliar este mismo proyecto bajo la licencia GPLv3.

La motivación de este proyecto, como en el primer capítulo introductorio de la tesis se planteó, fue mejorar la calidad de vida de las personas con discapacidades auditivas, mediante un sistema que de manera automática interprete un conjunto de señas del LSM para que una persona que desconoce el lenguaje de señas pueda interpretar dichos gestos observando su representación alfanumérica en una pantalla y entender el significado de un gesto. Este proyecto surge a raíz de investigaciones demográficas obtenidas de censos nacionales en donde casi 1.18 millones de habitantes de la población mexicana sufre algún tipo de trastorno degenerativo de la audición o del habla, y la carencia de sistemas que permitan a este segmento de la población comunicarse con personas no hablantes del lenguaje de señas es un problema real y que continua en aumento hoy por hoy.

Uno de los primeros problemas encontrados a lo largo de la investigación es la manera en la cual un gesto corporal puede ser medido, ya que si bien es sabido que todas las personas, o en su mayoría, poseemos dos manos, y cinco dedos por mano, las formas, así como los tamaños y las complexiones de las mismas varían entre persona y persona, además también de la diferencia entre géneros y edades puesto a pesar de que una mano femenina es igual en varios aspectos, difiere también de una mano masculina. El problema de la selección del conjunto óptimo de características para la correcta identificación

de un gesto nos llevo por un largo camino, tanto para poder completar una segunda tesis, acerca del conjunto valido de características. En las primeras pruebas se analizó la idea de identificar un gesto mediante redes neuronales multi-capa, con el perceptrón como unidad fundamental de cada capa, y para esta prueba consideramos la intensidad lumínica de cada pixel en un umbral de escala de grises comprendido entre cero y 255. Los resultados, aunque fueron positivos, venían con mucha varianza entre bloque y bloque de validación cruzada, siendo prácticamente los mismos signos gesticulados por los participantes. Esto se debía tal vez, a que la posición de la mano con respecto al área de interés que entraba como característica variaba entre participante y participante, así como las deformaciones propias de una binarización, las cuales en aquellas primeras etapas no se habían considerado. Otra idea que se puso en practica fue el identificar un gesto con métodos no basados en la forma, si no en la frecuencia usando el método de los descriptores de Fourier. Este método fue todavía más inexacto que las redes neuronales, ya que si bien a cada signo era posible extraerle una firma única basándonos en el contorno de la mano, esas firmas eran muy poco consistentes entre cada gesto realizado por varias personas, y por lo tanto era bastante difícil de lograr una buena clasificación. Igualmente se intento catalogar el conjunto de signos con las Máquinas de Soporte Vectorial, SVM, y esta técnica ofreció un nivel de identificación aceptable y con poca varianza, pero una vez más los resultados no fueron demasiado contundentes como para decantarnos por esta vía del reconocimiento. Finalmente los primeros resultados positivos y contundentes que estábamos buscando vinieron con la extracción de características Haar, combinadas con el método de identificación Adaboost, ya que la técnica de extracción era lo suficientemente rápida para procesar múltiples imágenes en muy poco tiempo y el Adaboost permitía una correcta identificación del signo en tiempo real.

Originalmente la propuesta de solución para esta tesis se planteo un sistema que de manera automática reconociera un conjunto de señas del LSM. En lo real, se creo un sistema, que permite la correcta identificación de un conjunto de 10 gestos del LSM lo suficientemente rápido para interpretar cada gesto y su significado en tiempo real. Comenzamos por mencionar que el método desarrollado es robusto con respecto a la variabilidad de escala y ruido lumínico dos de los principales problemas que en la mayoría de los trabajos mencionados en el estado del arte tienen. Por otra parte en esta tesis se aporto un novedoso método rápido y eficiente para la calibración, ajuste y centrado correcto de imágenes tomadas a diferentes profundidades, o imágenes tridimensionales, con el objetivo de mejorar la tasa de reconocimiento del sistema, que en resumidas cuentas ayudó mucho ubicando la mano en la

misma posición relativa, todas las veces. Otro aporte de esta proyecto fue una versión paralelizable del método de aprendizaje AdaBoost, que si bien la etapa de entrenamiento es rápida con un número pequeño de características a identificar, el tiempo crece de manera exponencial a medida que el número de características aumenta, que en este caso fueron más de 250,000, y cada ronda de entrenamiento por letra duraba aproximadamente 2.5 días por cada una de las 10 letras y tomaba aproximadamente 2 semanas entrenar todo el conjunto de letras corriendo en 2 estaciones de trabajo al mismo tiempo. La versión optimizada y paralelizable de este método presentada en este trabajo, realiza la etapa de entrenamiento de todo el conjunto valido de 10 signos en apenas 3 días, lo que representa menos de una cuarta parte del tiempo empleado para entrenar si utilizamos una versión lineal del AdaBoost.

6.1. Trabajo futuro

Aún queda mucho por hacer con respecto al reconocimiento del lenguaje de señas para cualquiera de sus variantes y un primer paso fue dado en este trabajo, sin embargo como igualmente se había planteado en el marco teórico, los gestos en los que se enfocó esta investigación fueron únicamente los estáticos, y un subconjunto de ellos. Parte del trabajo futuro es precisamente concluir todos los gestos estáticos posibles que conforman el alfabeto del LSM así como su simbología numérica esto último con el objetivo de lograr articular palabras completas al deletrearlas por una persona LSM-parlante. Por otra parte se planteo también durante este proyecto la idea de analizar los gestos móviles o Spotting mediante secuencias de video capturadas a una variedad de personas para obtener de nueva cuenta una base de datos lo suficientemente grande para iniciar la extracción de las nuevas características móviles. Afortunadamente el reciente abaratamiento de los sistemas de almacenamiento, así como las cámaras de profundidad permitirían en algún momento ampliar este proyecto y permitir entonces identificar lo que en años recientes ha recibido el nombre de *dynamic gesture recognition*, para robustecer aun más este sistema y ayudar a las personas con discapacidades auditivas o del habla a tener una mejor calidad de vida en una sociedad que en su mayoría es regida por el lenguaje hablado.

Bibliografía

[1] Microsoft, Human Interface Guidelines. Kinect for Windows v1.5.0. 2012. Retrieved from http://go.microsoft.com/fwlink/?LinkID=247735 (25 .12.2012). 2012.

[2] V Adithya and Usha Gopalakrishnan. Artificial Neural Network Based Method for Indian Sign Language Recognition. (Ict):1080–1085, 2013.

[3] Mahesh Chikkanna. Kinect Based Real-time Gesture Spotting Using HCRF. pages 925–928, 2013.

[4] Xinyi Cui Xinyi Cui, Yazhou Liu Yazhou Liu, Shiguang Shan Shiguang Shan, Xilin Chen Xilin Chen, and Wen Gao Wen Gao. 3D Haar-Like Features for Pedestrian Detection. *Multimedia and Expo, 2007 IEEE International Conference on*, pages 1263–1266, 2007.

[5] Michael Van den Bergh, Esther Koller-Meier, and Luc Van Gool. Fast body posture estimation using volumetric features. *Motion and Video Computing, IEEE Workshop on*, 0:1–8, 2008.

[6] Shishi Duan, Xiangyang Wang, and Wanggen Wan. The logitboost based on joint feature for face detection. *Image and Graphics, International Conference on*, 0:483–488, 2013.

[7] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

[8] Laetitia Gond, Patrick Sayd, Thierry Chateau, and Michel Dhome. A 3d shape descriptor for human pose recovery. 5098:370–379, 2008.

[9] Masayuki Hiromoto, Kentaro Nakahara, Hiroki Sugano, Yukihiro Nakamura, and Ryusuke Miyamoto. A specialized processor suitable for AdaBoost-based detection with haar-like features. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007.

[10] Masayuki Hiromoto, Hiroki Sugano, and Ryusuke Miyamoto. Partially parallel architecture for AdaBoost-based detection with Haar-like features. *IEEE*

Transactions on Circuits and Systems for Video Technology, 19(1):41–52, 2009.

[11] Weiming Hu, Wei Hu, and Steve Maybank. AdaBoost-based algorithm for network intrusion detection. *IEEE Transactions on Systems, Man, and Cybernetics*, 38(2):577–83, 2008.

[12] Chang Huang, Haizhou Ai, Yuan Li, and Shihong Lao. Learning sparse features in granular space for multi-view face detection. *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, 0:401–407, 2006.

[13] Lijin Huang, Jun Shi, Ruiling Wang, and Shicong Zhou. Shearlet-based ultrasound texture features for classification of breast tumor. *2013 Seventh International Conference on Internet Computing for Engineering and Science*, 0:116–121, 2013.

[14] Szu-Hao Huang and Shang-Hong Lai. Detecting faces from color video by using paired wavelet features. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 5:64, 2004.

[15] INEGI. http://cuentame.inegi.org.mx/poblacion/discapacidad.aspx?tema=P. 2012.

[16] Xiaofei Ji, Lu Zhou, and Qianqian Wu. A Novel Action Recognition Method Based on Improved Spatio- Temporal Features and AdaBoost-SVM Classifiers. 8(5), 2015.

[17] Motoki Kimura, Janarbek Matai, Matthew Jacobsen, and Ryan Kastner. A Low-Power AdaBoost-Based Object Detection Processor Using Haar-Like Features. (Cm):203–206, 2013.

[18] N. Larios, B. Soran, L.G. Shapiro, G. Martinez-Munoz, J. Lin, and T.G. Dietterich. Haar random forest features and svm spatial matching kernel for stonefly species identification. *Pattern Recognition, International Conference on*, 0:2624–2627, 2010.

[19] B Leibe, E Seemann, and B Schiele. Pedestrian detection in crowded scenes. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 1:878–885, 2005.

[20] Wei Li and QingXia Li. Using naive bayes with adaboost to enhance network anomaly intrusion detection. *Intelligent Networks and Intelligent Systems, International Workshop on*, 0:486–489, 2010.

[21] Yi Li. Hand Gesture Recognition Using Kinect. pages 196–199, 2010.

[22] Raul Gonzalez Perez Maria Esther Serafin de Fleischmann. *Manos con voz, diccionario de lengua de señas mexicana.* Conapred, 2011.

[23] Priyanka Mekala, Ying Gao, Jeffrey Fan, and Asad Davari. Real-time Sign Language Recognition based on Neural Network Architecture. pages 195–199, 2011.

[24] Takeshi Mita, Toshimitsu Kaneko, and Osamu Hori. Joint haar-like features for face detection. *Computer Vision, IEEE International Conference on*, 2:1619–1626, 2005.

[25] Sushmita Mitra and Tinku Acharya. Gesture Recognition: A Survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 37(3):311–324, May 2007.

[26] Constantine P. Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. pages 555–, 1998.

[27] Juan I. Pastore, Agustina Bouchet, Emilce Moler, and Virginia Ballarin. Segmentation of bone marrow biopsies by mathematical morphology in color spaces. *IEEE Latin America Transactions*, 11(1):329–333, 2013.

[28] Minh-Tri Pham, Yang Gao, Viet-Dung D. Hoang, and Tat-Jen Cham. Fast polygonal integration and its application in extending haar-like features to improve object detection. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 0:942–949, 2010.

[29] Ahmed Samirelons, Magdy Abull-ela, and Mohamed F Tolba. Pulse-coupled neural network feature generation model for Arabic sign language recognition. (February):829–836, 2013.

[30] Mohak Shah, Mario Marchand, and Jacques Corbeil. Feature selection with conjunctions of decision stumps and learning from microarray data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):174–186, 2012.

[31] Peter Sussner, Mike Nachtegael, and M Tom. L -Fuzzy Mathematical Morphology: An Extension of Interval-Valued and Intuitionistic Fuzzy Mathematical Morphology. *The 28th North American Fuzzy Information Processing Society Annual Conference (NAFIPS 2009)*, (28):6, 2009.

[32] Atsushi Takemura, Akinobu Shimizu, and Kazuhiko Hamamoto. Discrimination of breast tumors in ultrasonic images using an ensemble classifier based on the adaboost algorithm with feature selection. *IEEE Transactions on Medical Imaging*, 29(3):598–609, 2010.

[33] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 1, 2001.

[34] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 4:51–52, 2001.

[35] Dong Wei and Wang Tong. Research on Edge Detection Based on Mathematical Morphology Algorithm. *2010 International Conference on Optoelectronics and Image Processing*, 2:211–213, 2010.

[36] Anthony Williams. *C++ Concurrency in Action*. 2012.

[37] A Yamamoto, M Tezuka, T Shimizu, and R Ohbuchi. SHREC08 entry: Semi-supervised learning for semantic 3D model retrieval. *Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on*, (1):241–243, 2008.

[38] Liu Yucheng Liu Yucheng and Liu Yubin Liu Yubin. An Algorithm of Image Segmentation Based on Fuzzy Mathematical Morphology. *2009 International Forum on Information Technology and Applications*, 2(3):0–3, 2009.

[39] Zhao Yueai and Chen Junjie. Application of unbalanced data approach to network intrusion detection. *Database Technology and Applications, International Workshop on*, 0:140–143, 2009.