UNIVERSIDAD TECNOLOGICA NACIONAL

Técnicas Digitales III

Ingeniería Electrónica

Proyecto anual:

“Sistema de acceso por reconocimiento facial”

Docentes: Ing. Mariano Mercado
Ing. Martín Marino

Alumnos: Borghi Orué, Emiliano Javier (LEG: *15-19855*)
Domecq, Brian Lucas (LEG: *15-19823*)
Morel, Gabriel (LEG: *15-19812*)

Grupo 4

5°2°

Año 2014

Índice

1 . Introducción teórica

El sistema de acceso tiene como basamento teórico el reconocimiento facial aplicando técnicas de visión por computadora. Estas técnicas aplicadas pueden dividirse en cuatro grande grupos:

- **Detección de caras:** consiste en localizar un rostro humano dentro de una imagen.
- **Preprocesamiento:** herramienta que sirve para mejorar el rendimiento de los algoritmos de reconocimiento.
- **Recolección y aprendizaje:** es una técnica de aprendizaje supervisado que toma muestras de caras preprocesadas y con ellas aprende a diferenciarlas.
- **Reconocimiento facial:** algoritmo que verifica cuál de las caras recolectadas se asemeja más a la que toma la webcam.

Principales desventajas

La principal desventaja que tiene el sistema de reconocimiento facial mediante OpenCV es que los métodos utilizados dependen de la iluminación del ambiente.
Los algoritmos implementados poseen una alta eficacia bajo condiciones estrictas de iluminación y éstas deben mantenerse para no disminuir la cantidad de “aciertos”. Por este motivo es de vital importancia la correcta recolección del set de entrenamiento.

Otro inconveniente que surge es la posición de la cara y de las particularidades que puedan llegar a tener los rostros detectados, por ejemplo, el uso de los anteojos u otros objetos distintivos.

Debido al primer problema mencionado, el preprocesado de las imágenes es muy importante, ya que minimiza las variaciones de iluminación u otros efectos no contemplados, causando un aumento en la confiabilidad del sistema.

Para la detección de rostros nuestros programas utilizan detectores XML previamente entrenados disponibles en OpenCV para su uso y se listan a continuación:
Para detección de la cara frontal:

- `lbpcascade_frontalface.xml` (Usa *Fast LBP - Fast Local Binary Pattern*)

Y para detectar los ojos (únicamente si están abiertos):

- `haarcascade_eye.xml`
- `haarcascade_eye_tree_eyeglasses.xml`

El proceso de recolección de caras consiste en colocar las caras preprocesadas en una matriz. A dicho arreglo se lo conoce como *set de entrenamiento*.
Este conjunto debe contemplar que las imágenes utilizadas contengan la información a utilizarse para la ocasión, o sea que si una persona debe ser detectada con expresiones

faciales cambiantes (alegre, triste, etc), estas fotos deben encontrarse en la matriz y así evitar reconocimientos erróneos.

A la hora de recolectar caras de personas es importante tener en cuenta que mejor que tener muchas imágenes de entrenamiento, es aún mejor tener una gran variedad de rostros.

En cuanto a la etapa de aprendizaje, OpenCV disponible de dos métodos para ello:

- **Eigenfaces** (más conocido como *PCA - Principal Component Analysis*)
- **Fisherfaces** (*LDA - Linear Discriminant Analysis*)

En nuestro sistema se emplea este último.
Ambos métodos son muy empleados en visión por computadora, aprendizaje de máquina y reconocimiento de patrones. Para dar una breve introducción, el método utilizado en este documento busca una combinación lineal entre las características de los datos a ser comparados, aplicando una técnica conocida como *Reducción de la dimensionalidad*.

2. Especificación de diseño

2.1. Descripción detallada del sistema completo

El *sistema de acceso por reconocimiento facial* consiste en una red *LAN* de cámaras conectadas de manera centralizada a un servidor que contendrá una base de datos para poder realizar el reconocimiento de rostros de personas. En términos generales consta de dos etapas:

- Carga de la base de datos facial
- Sistema de detección y acceso

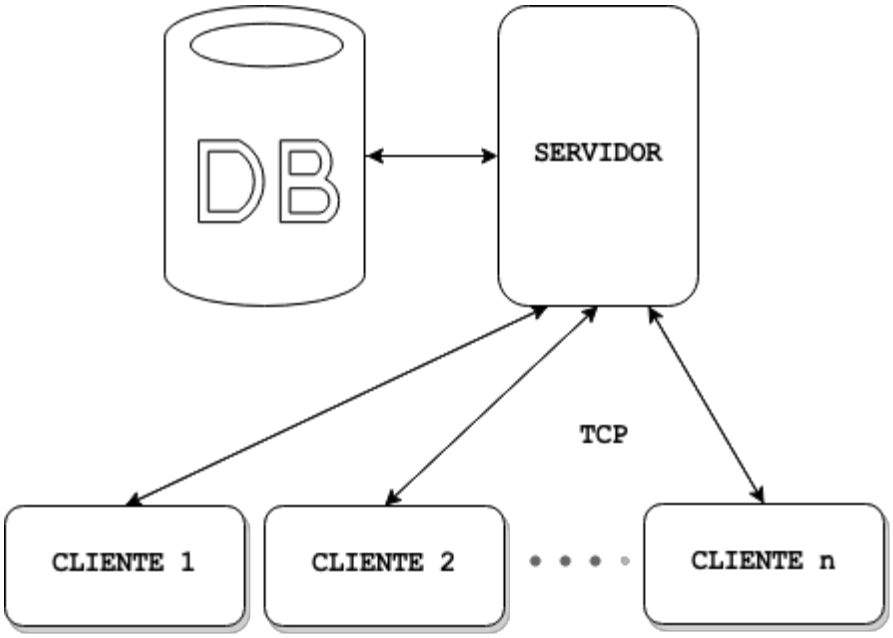

Fig.: 1 Sistema de acceso completo. Diagrama en bloques general

2.1.1. Carga de Base de datos (DB)

Antes de inicializar el sistema de detección es necesario que el servidor posea una base de datos de las caras de las personas que tienen permitido el acceso al sitio en donde será implementado. Para ello, se emplea un programa que ejecuta instrucciones de alto nivel de OpenCV para detectar rostros.
Para ellos se utilizan dos métodos de reconocimiento:

- Clasificadores Haar
- Patrones Binarios Locales (*LBP*, por sus siglas en inglés)

La base de datos es una aplicación utilizada por el servidor, la cual se encarga de detectar y guardar una cierta cantidad de rostros, para luego ser empleadas en la comparación en tiempo real con los frames tomados por las webcam clientes.

Una vez finalizado el procedimiento de carga, se procede con el sistema de detección y acceso.

2.1.2. Sistema de detección y acceso

En este momento las cámaras webs remotas comienzan a “interactuar” con el programa servidor como si fueran *Clientes* de una red.
En primera instancia, el servidor habilita las conexiones TCP entrantes de sus clientes, quienes comenzarán a tomar frames de la cámara web. Cuando esto suceda, el cliente le avisará al servidor que ha detectado una persona y le realiza un *pedido de reconocimiento*.

El servidor mientras tanto se comporta de manera *concurrente*, ya que las solicitudes pueden provenir de manera simultánea de distintos puntos de reconocimientos. Ante la

producción de este evento, se crea un proceso *Child* que se encargará durante todo el proceso de interactuar con el cliente en cuestión. En caso de fallos en la conexión Cliente-Servidor, por ejemplo, desconexión de la cámara, se empleó un sistema de mensajes *Keep Alive*, de esta manera la conexión defectuosa se terminará, liberando así los recursos usados por la misma.

El proceso hijo recibirá una *cierta cantidad* de frames provenientes del cliente con la cara detectada (no reconocida), para emplearlos posteriormente en la comparación con los guardados en la base de datos, cargada en el proceso anterior. En respuesta a esto pueden ocurrir dos sucesos:

> El resultado sea positivo, o sea, que el rostro haya sido encontrado, por lo cual la persona tiene acceso al lugar. El servidor responderá al cliente con el nombre de la persona detectada.
>
> El resultado sea negativo, negándose el sistema a dejar acceder a la persona al sitio. En este caso, el servidor responderá al cliente con “persona no encontrada”.

Entonces, cuando el cliente recibe esta información la muestra en pantalla y se cierran los procesos antes comunicados. Finalmente el cliente, antes de volver a detectar rostros, generará una *demora* que se ajustará de acuerdo a la actividad que se emplee. Esto quiere decir, por ejemplo, si la persona luego de ser detectada permanece un rato frente a la visión de la cámara, ésta no debe ser informada al servidor ya que esta misma fue previamente analizada.

2.2. Posibles usos y aplicaciones

Esta tecnología es aplicable a lugares en donde el acceso sea sólo de a una persona a la vez, esto es así por la naturaleza misma del funcionamiento de los algoritmos, ya que no permiten el reconocimiento facial simultáneo.

Un ejemplo típico de esta tecnología podría implementarse en una empresa donde se desee controlar el acceso del personal a un sector restringido, o para el uso de bloqueo/desbloqueo de computadoras u otros dispositivos que cuenten con una cámara.

2.3. Detalles de implementación

Para poder implementar este sistema es necesario contar con cámaras web conectadas a una misma red, a su vez los dispositivos que posean dichas cámaras deberán contar con un sistema operativo Linux (el mismo fue testeado en una distribucion Ubuntu) y deberán tener instaladas las librerías de OpenCV en su última versión. Además, se deberá contar con una computadora principal que cumplirá el papel de servidor.

Existen dos maneras (o medios) de conectar el sistema pero depende de la cantidad de cámaras a utilizar y la distancia a la que se encuentren del servidor.

Estos medios de comunicación LAN mencionados anteriormente son:

- Cable Ethernet (cableado) (*IEEE 802.3*)
- Wi - Fi (inalábrico) (*IEEE 802.11*)

A continuación se recomienda cuando usar cada medio dependiendo de las posibilidades antes comentadas.

		Cantidad de cámaras a utilizar	
		1	2 ó más
Distancia hasta el servidor	Menor a 100 m *	Ethernet	Ethernet con Router/ Switch
	Mayor a 100 m	Wi -Fi	Wi - Fi (con repetidora)

*Depende de la tecnología del cable empleado. Para este caso se tuvo en cuenta un cable par trenzado.

2.4. Plan de pruebas durante las etapas de desarrollo

Durante la etapa de desarrollo surgieron diversas alternativas de implementación para suplir las incidencias que pudieran ocurrir en la práctica. Es por esta razón que, como en el siguiente documento se presentarán dos alternativas para el sistema de acceso presentando sus ventajas y desventajas.

El funcionamiento general de ambos es similar y fue explicado al comienzo. A continuación se explicará cada uno con mayor detalle.

2.4.1. Primera versión

Como se pudo observar en el diagrama básico del funcionamiento general, el programa está formado por tres partes:

- Carga de la base de datos
- Programa Servidor
- Programa Cliente

A continuación se explicará, por separado, cómo funciona cada uno.

2.4.1.1. Carga de la base de datos

La carga de la base de datos sucede, para esta versión, dentro del programa servidor debido a que el "*model*" (vector del tipo FaceRecognizer resultante del entrenamiento de los datos tomados, nativo de OpenCV) obtenido aquí no se

guardará en el disco sino que será un vector con todas las características obtenidas de las personas a guardar.

La interfaz gráfica de carga posee dos botones: uno para tomar frames de la cámara adjunta y el otro para finalizar proceso de carga de datos.

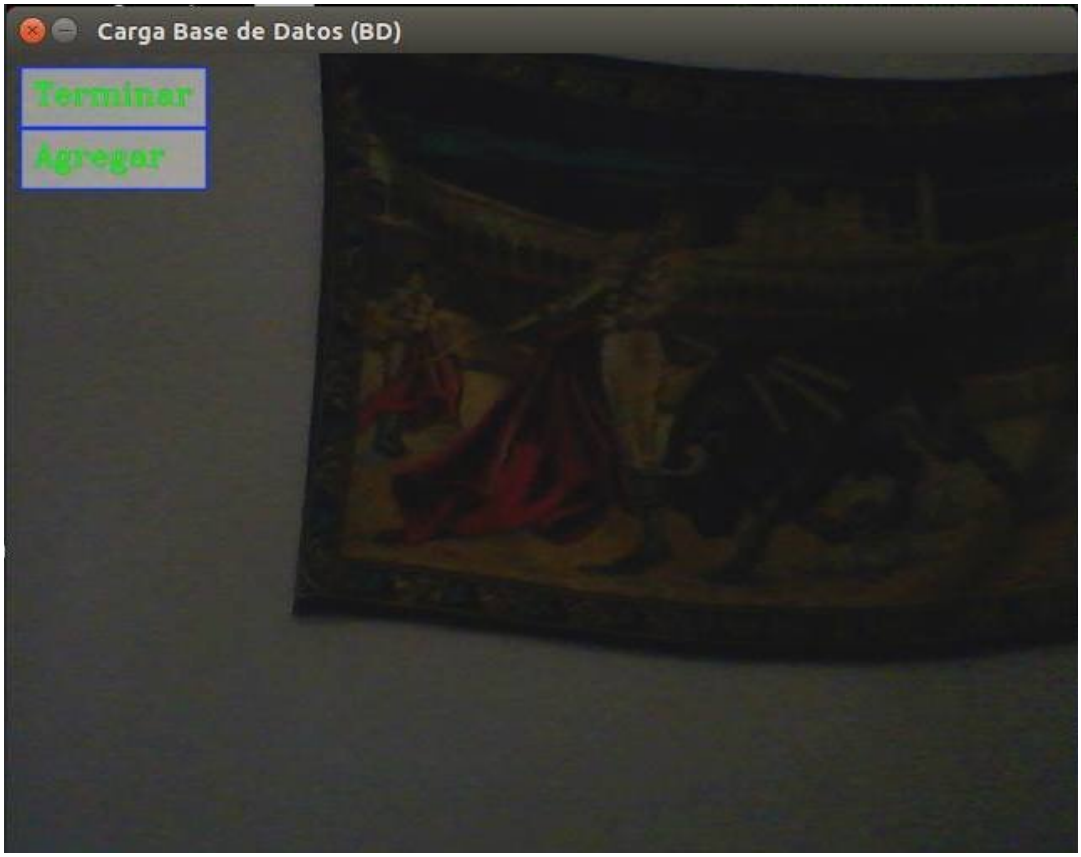

Fig. 2: GUI de la carga de la base de datos

Cuando se presiona el primero, por medio de TCP se da una orden al servidor para que se prepare a recibir los frames de los rostros detectados. La detección de un rostro será fácilmente identificable por medio de un rectángulo y círculos en los ojos de la persona. Cuando la detección sea factible, se iluminará dicho rectángulo dando la sensación de un flash que identifica al frame tomado y enviado al servidor. Estos frames se pre procesarán con OpenCV y se guardarán en el vector con los otros rostros procesados con anterioridad.
Los vectores empleados durante todo el sistema, en lo que respecta al tratamiento de imagenes, son del tipo Mat (matriz), tipo propio de OpenCV. Esto se hizo así, ya que dicho tipo nos proporciona una mayor gama de herramientas que facilitan y posibilitan un mejor tratamientos de las imagenes, como pueden ser el uso de matrices de transformación.

Todo lo anteriormente dicho se resume en el siguiente diagrama.

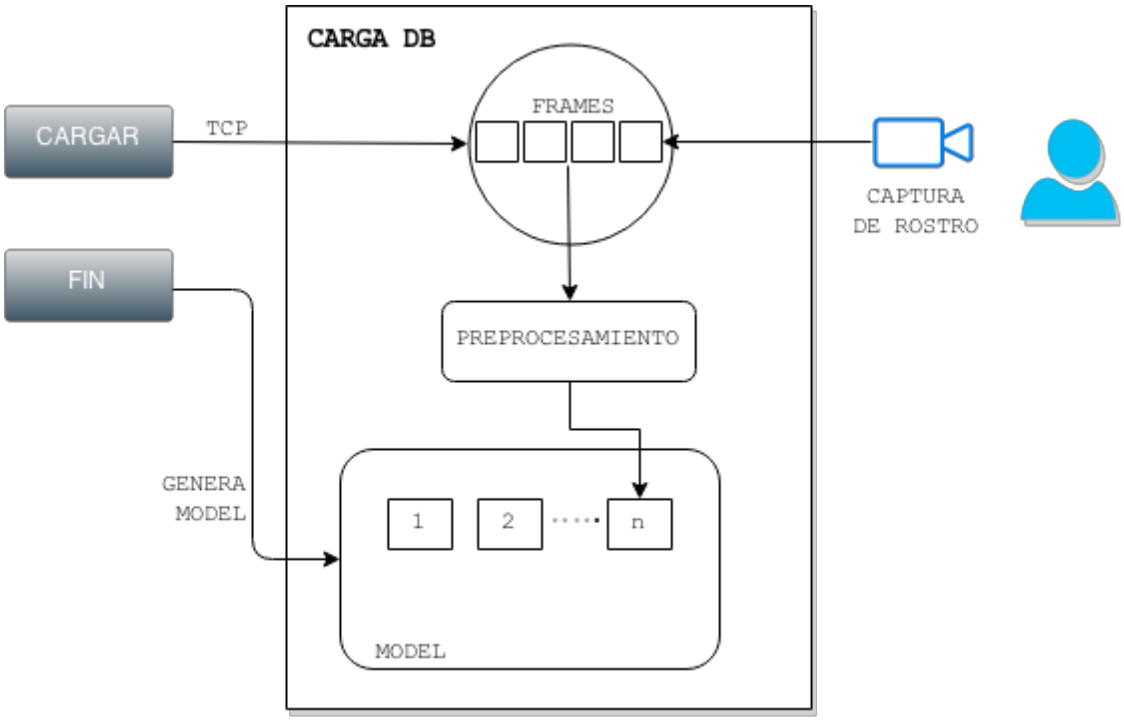

Fig.: 3 Diagrama de flujo de carga base de datos(DB)

Cuando se presione el botón para finalizar se genera el "*model*" que consiste en todos los rostros tomados durante el proceso de carga y que han sido modificados (pre procesados) especialmente para una mejor comparación luego, en el reconocimiento facial.

La principal desventaja de esta versión, es que el *"model"*, es decir, la base de datos, se realiza una única vez al ser ejecutado el programa servidor, imposibilitando la actualización de la misma durante el tiempo que el servidor se esté ejecutando. Además, al no ser guardado el *"model"* en memoria, si se finalizan los procesos o se cierran por algún evento inesperado, provoca la pérdida de estos y por consiguiente, habrá que volver a cargarla cada vez que se ejecute el servidor.

En base a los problemas antedichos, se avanzó sobre los mismo y se concluyó con la versión dos, que se explica en la sección *2.4.2*.

2.4.1.2. Programa Servidor

El servidor consta de un proceso padre que, esperará por pedidos de conexión entrantes provenientes de cualquiera de los clientes. Por cada uno generará un proceso hijo a quién le delegará las tareas del preprocesamiento y reconocimiento facial.
Una vez hecho esto, vuelve a esperar por nuevas conexiones comportándose como un *servidor concurrente*.

A continuación, se detalla el funcionamiento en un diagrama de flujo.

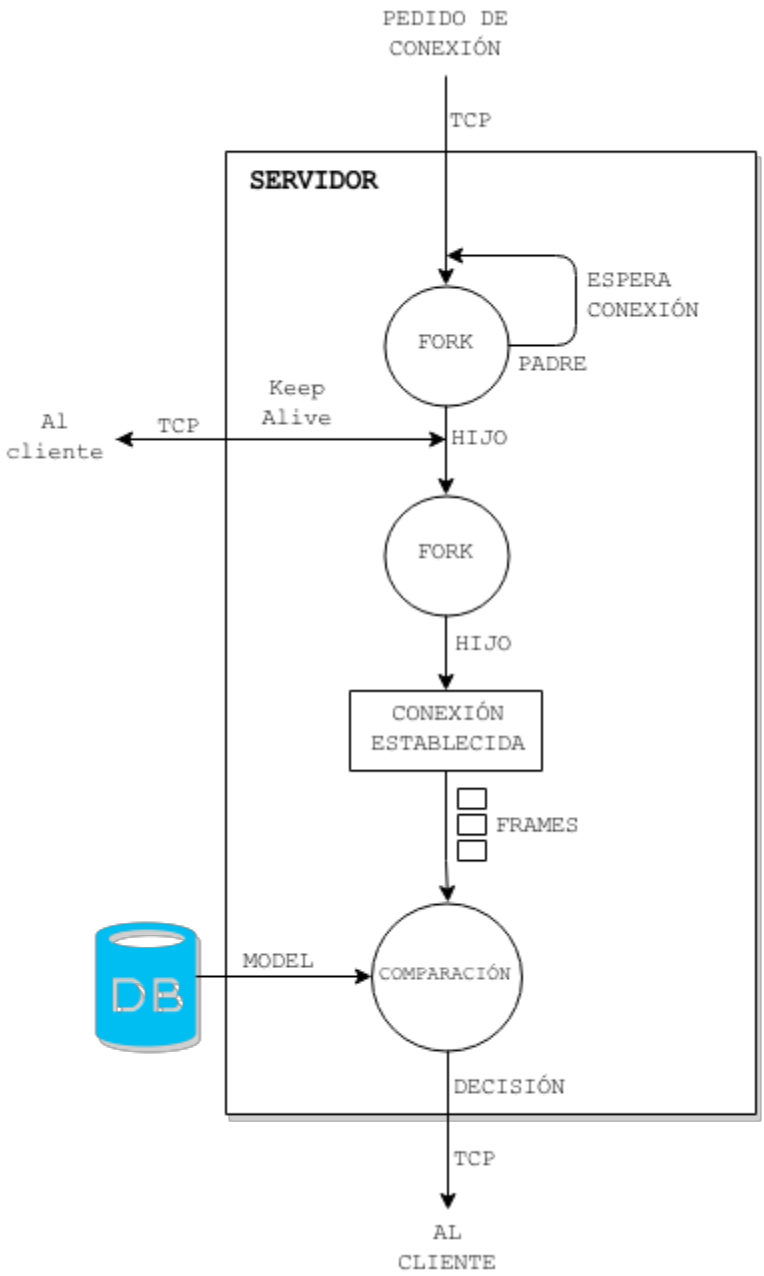

Fig.: 4 Diagrama de Flujo, Funcionamiento del Servidor (v1)

Una vez establecida la conexión, el Child toma los frames preprocesados provenientes del cliente y los compara con el "*model*" de la base de datos.
Este procedimiento genera una decisión que indicará si la persona se encuentra presente en la base de datos o no. En caso de ser afirmativo se devuelve junto con el nombre del mismo por TCP.

La base de datos es una zona de memoria del programa que, aunque sea utilizada por varios procesos, no es necesario que se controle su acceso debido a que solo será leída. No se la podrá escribir, por lo que se simplifica el algoritmo del sistema.

2.4.1.3. Programa Cliente

El programa cliente es el que tiene asociada la webcam que captura las imágenes (frames) de los rostros de las personas.

Antes de detectar si una persona se posa delante de la cámara, este proceso crea un *Child* que se encargará de ello, y al mismo tiempo crea otro *Child* que se ocupa de enviar al servidor el *Keep Alive*, de esta forma le hace saber al servidor que aunque no esté enviando frames por la no detección de rostros, el mismo sigue activo y que no eliminen sus recursos. Mientras que el padre escuchará la respuesta del servidor.

El proceso hijo toma un suma de frames de la webcam, solo cuando se detecta un rostro, y las preprocesa antes de enviarlas al servidor. Con esta técnica se obtiene, en cada frame, una “indicación” sobre los ojos y el rostro, la cual se emplea para que las personas sepan cuando el sistema los está detectando correctamente, en caso contrario podrá ajustar su postura hasta que lo detecte.

Luego, todos ellos, son enviados por TCP al servidor para que compare los resultados con la base de datos y le devuelva al proceso padre el resultado.

A continuación se ilustra el diagrama de funcionamiento completo del sistema en su primera versión.

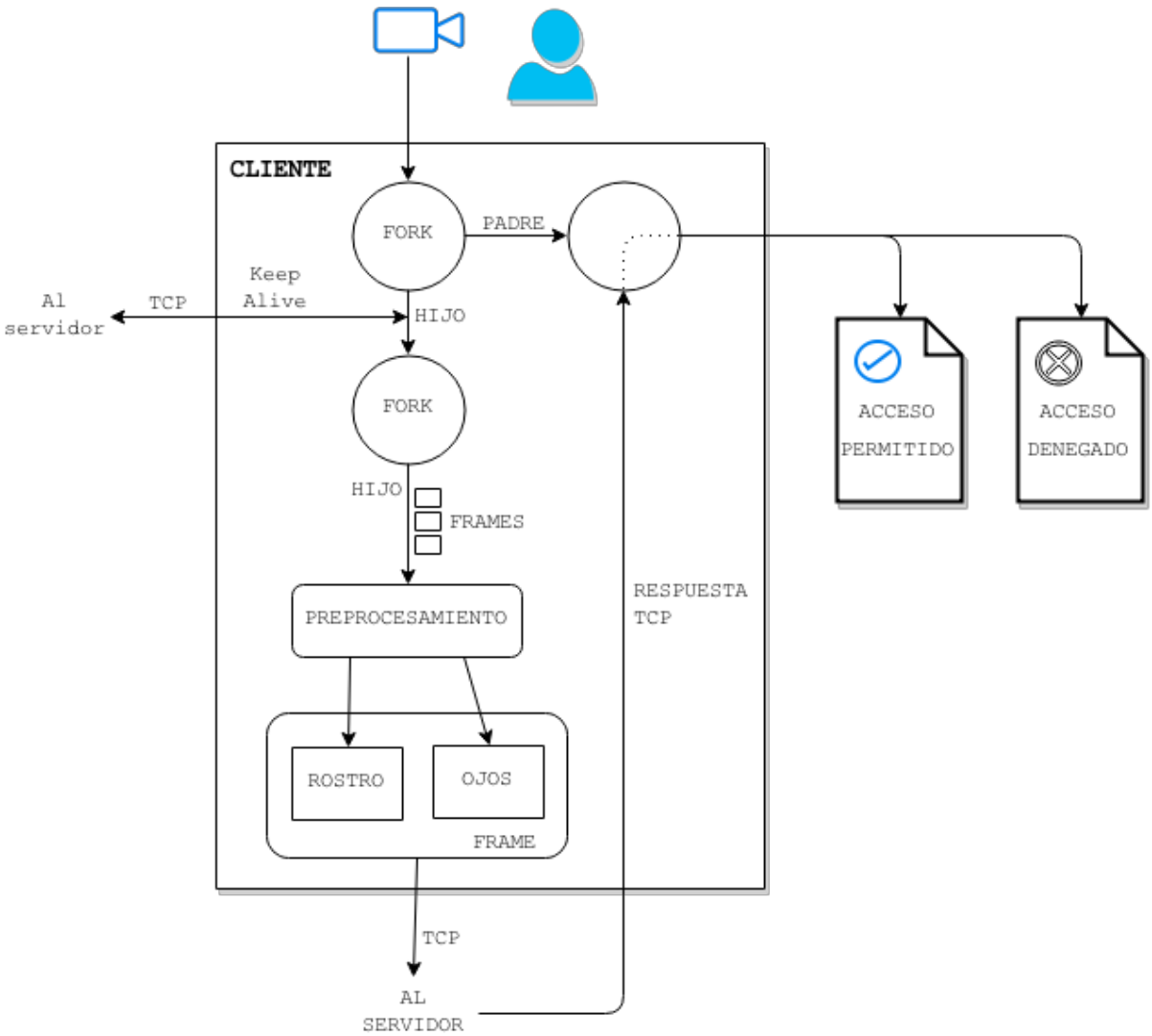

Fig.:5 Diagrama de flujo. Funcionamiento del Cliente

El padre, en caso de no haberse cortado la conexión, genera un mensaje en pantalla:

- *ACCESO PERMITIDO + NOMBRE*
- *ACCESO DENEGADO + NO IDENTIFICADO*

El primer caso sucede cuando la persona detectada se encuentra en la base de datos del servidor, en cambio el otro mensaje se da en caso de no encontrarlo.

Luego de introducir el mensaje deben esperarse por tiempos distintos para volver a verificar si hay una persona frente a la cámara.
Para el primer caso, se volverá a buscar luego de que la persona se retire del lugar. Dependiendo de la utilidad que se le dé al sistema de acceso, esta persona puede pasar mucho o poco tiempo frente a ella.

En cambio para el otro caso el sistema le da unos segundos a la persona para que se retire del campo de visión de la cámara y así no generar sucesivos procesamientos innecesarios y llamadas al servidor con identificaciones erróneas.

2.4.2. Segunda versión

2.4.2.1. Carga de la base de datos

Respecto a la versión primera, esta parte del sistema tiene dos diferencias importantes:

- Se ejecuta como un programa aparte, pudiendo estar en otra PC que no sea el servidor y luego copiarse el archivo generado a él.
- Este archivo generado se guarda en disco como un archivo YAML (*.YML*).

Además del archivo con las similitudes, se crea un vector donde figuran los nombres de las personas que fueron analizadas para luego ser enviadas al cliente.

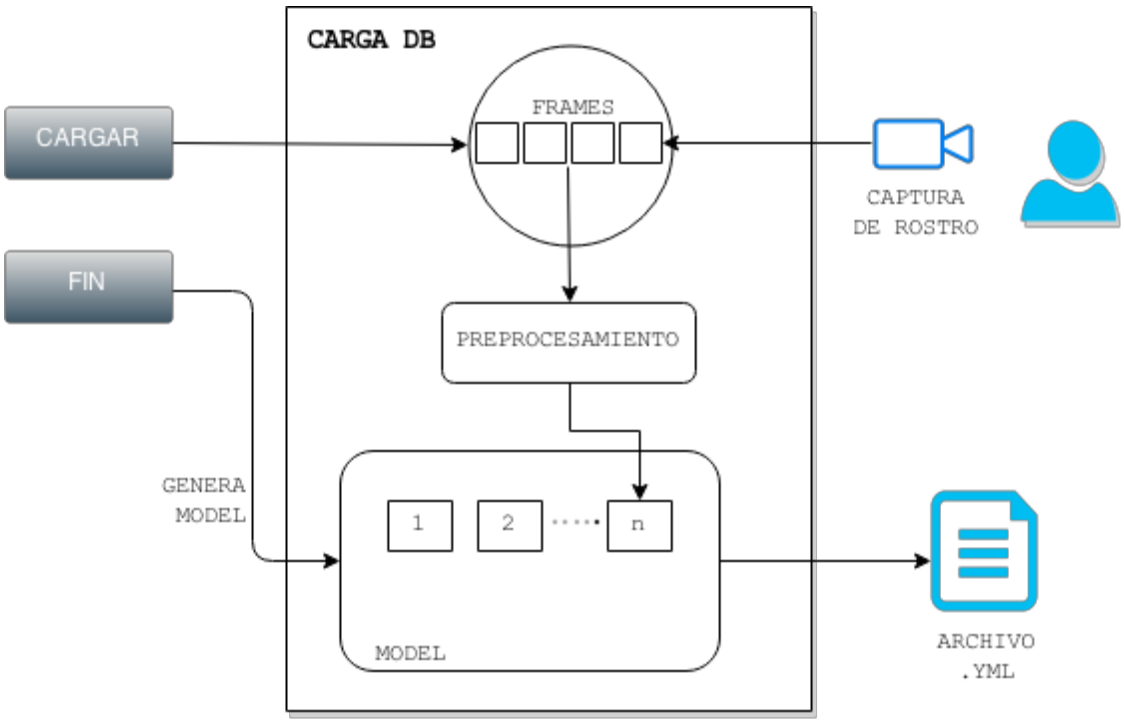

Fig.: 6 Diagrama de flujo. Carga de la base de datos (v2)

Este nuevo método de carga presenta dos ventajas a remarcar:

- Se puede volver a utilizar la base de datos o puede recuperarse en caso de terminación inesperada de los procesos.
- Es posible actualizar la base de datos mientras se ejecuta el programa servidor, cosa que no era posible en la primera versión una vez inicializado éste.

Otro aspecto a destacar es que, al ser ahora un programa aparte, la complejidad de programación del servidor es menor respecto a la primera versión ya que solo tiene que verificar los cambios realizados en el archivo YAML.

2.4.2.2. Programa Servidor

El programa Servidor presenta muchos cambios respecto a la versión anterior.

Lo primero que hace este programa es tomar el archivo .YML (contiene el *"model"*) de base de datos generado previamente y cargarlo a una variable de iguales características, esto es del tipo FaceRecognizer como se explicó en la versión 1.
Una vez cargada la base de datos en el servidor se cierra dicho archivo.

Una vez hecho esto, se "*forkea*" el proceso principal donde cada parte tendrá tareas específicas:

- El proceso padre se encarga de verificar si hubieron cambios en la base de datos. Para ello se utiliza un flag en la memoria compartida. Esto se hace debido a la funcionalidad agregada en esta nueva versión.
- El proceso hijo espera por pedidos de conexión del cliente a través de TCP. Si esto ocurre, crea un hijo que atenderá el llamado mientras que éste se quedará esperando nuevas conexiones comportándose como *servidor concurrente*.
- El proceso hijo que atiende al cliente funciona de manera similar al de la versión primera con la diferencia que la base de datos, al estar adjunta con un semáforo, solo podrá ser accedida cuando no esté siendo cargada.

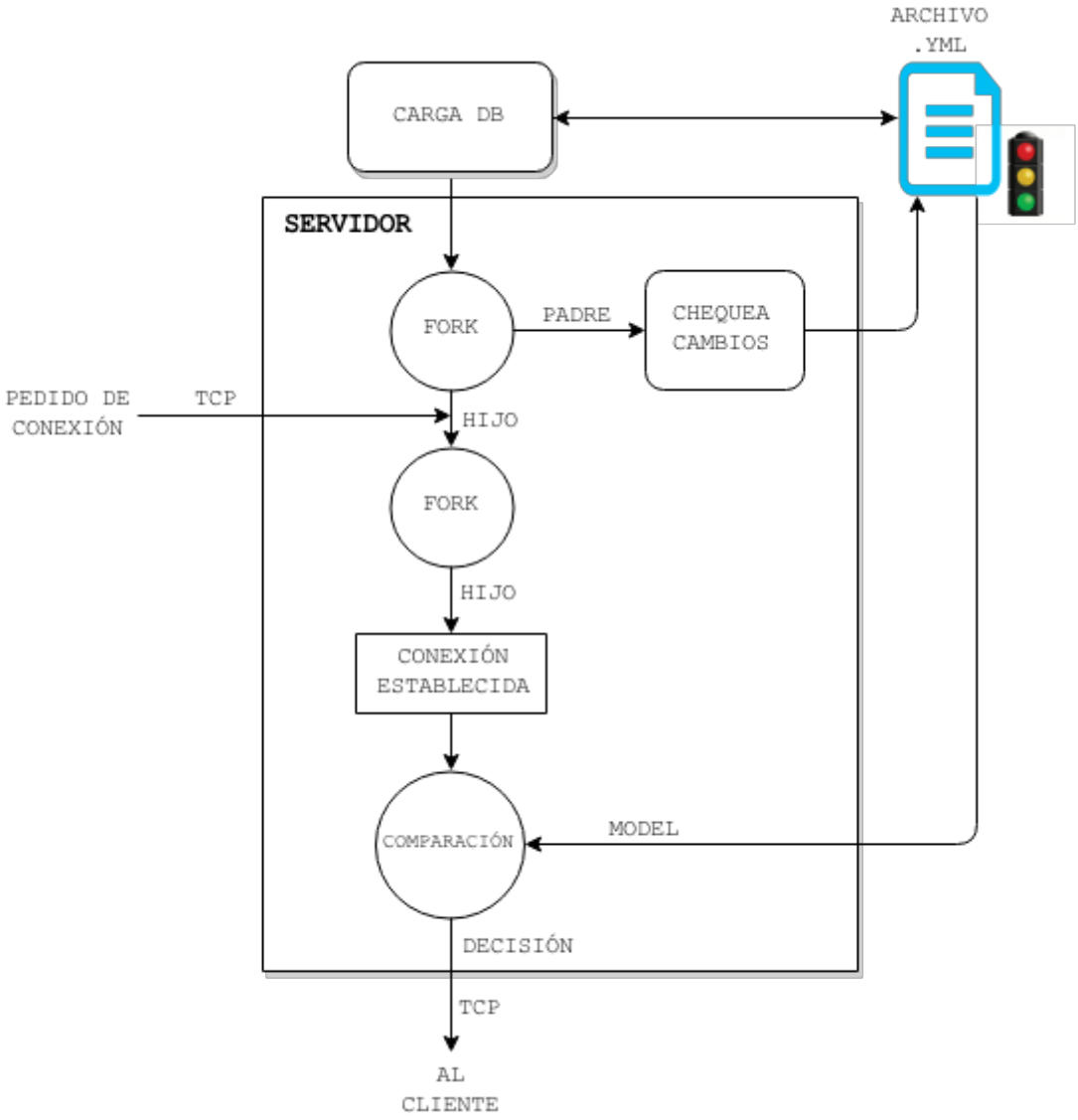

Fig.: 7 Diagrama de flujo. Funcionamiento del servidor (v2)

En esta versión se debió implementar el uso de semáforos para evitar el acceso simultáneo al archivo de lectores y escritores, esto es del servidor y del programa que genera la base de datos.
Básicamente la diferencia entre la versión 1 y la 2, radica en la forma de salvaguardar la base de datos.

La implementación de esta versión presenta una desventaja importante que está explicada en el anexo y se relaciona con la actualización de la base de datos.

2.4.2.3. Programa Cliente

El programa Cliente no presenta cambios sustanciales respecto a la primera versión.

3. Explicación del código

En todo programa lo primero que se realiza es la inicialización de los puertos, detectores, variables, etcétera.

3.1. Inicializaciones

La primera inicialización corresponde a sockets TCP para comunicar procesos de Linux a través de redes de datos. Aquí se usarán tres funciones elementales:

- `socket()`: Retorna el File Descriptor de un socket TCP.
- `bind()`: Le asigna un puerto y una dirección IP.
- `listen()`: Escucha conexiones entrantes del cliente.

Para ambas versiones se utilizaron dos sockets: uno para recibir los frames y para enviar la respuesta al reconocimiento, y el otro para el *Keep Alive*.

Para inicializar los detectores XML se usa la función `initDetectors()` que además implementa control de fallos. Básicamente carga con `load()` los archivos XML en variables del tipo `CascadeClassifier`. Estos archivos, son los que posteriormente se emplean para la detección de rostro.

3.2. Carga de base de datos

En la primera versión, a diferencia de la segunda, la carga de datos se realiza a través de una comunicación TCP. Para esto se implementan dos funciones conocidas:

- `accept()`: El servidor espera un aviso del programa de carga donde le avisa que comenzó a ejecutarse.
- `recv()`: En caso de apretar los botones de la GUI de carga, el servidor recibe el aviso con esta función.

En la segunda versión la diferencia sustancial es que utiliza una memoria compartida para del tipo `struct areacompartida` donde la variable a destacar es la matriz `Nombres` que almacena los nombres de las personas detectadas y que serán leídas de esta misma área de memoria por el servidor.

3.3. Procesamiento de caras y entrenamiento

Cuando se toman los frames, se les detecta el rostro y los ojos con la función `getPreprocessedFace()`, a la misma se le pasa como argumentos el frame tomado y los CascadeClassifier. Esto se hace para poder detectar rostros en una imagen. Además, en la misma se realiza una serie de tratamientos, como ser la aplicación de una ecualización por histograma , esto lo empleamos para mejorar el contraste de la imagen, hacerla más clara y de esta manera mejorar los aspectos de la influencia de la luz ambiente.

El resultado de esta función es una variable del tipo Mat, es decir, una imagen de la cara detectada en escala de grises, con tamaño, contraste y brillo estándar. La misma queda lista para emplearse en la búsqueda de parecidos. Esto se realiza en la función `Buscaparecidos()` que se explicará mas adelante.

La función `preprocessed()`, se emplea para guardar el rostro detectado y preprocesado con `getPreprocessedFace()`, esto se hace con el fin de posteriormente realizar el "*model* " con los mismos.
Cabe destacar que en esta función para no guardar los mismos rostros, se emplea un algoritmo para poder identificar cambios en la imagen y así llenar la base de datos con rostros de una misma persona que posean leves variaciones y así asegurarnos una mejor detección ante pequeños cambios en gestos de la persona.

La función `preprocessed()` se explica en la imagen siguiente con un diagrama de flujo.

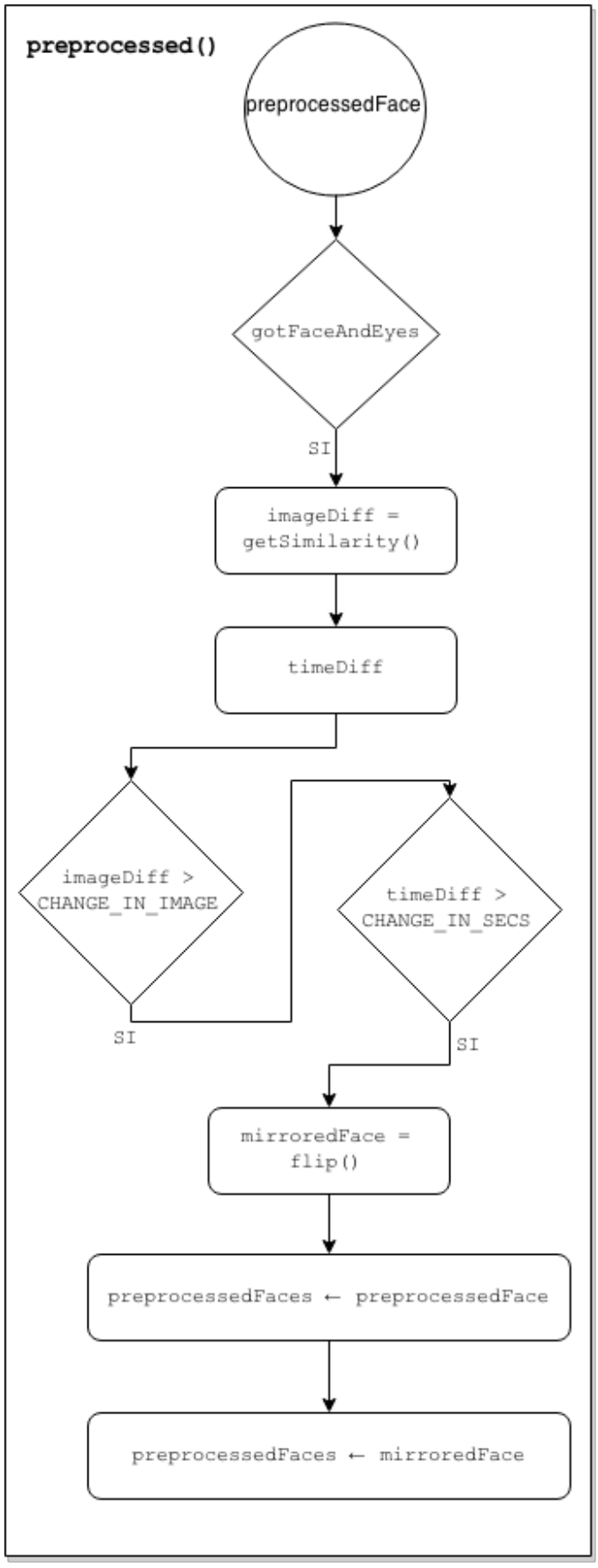

Fig.: 8 Función preprocessed()

El entrenamiento de la base de datos genera un "*model*" a través de la función `Trainning()`, el cual básicamente utiliza una función propia de OpenCV llamada `learnCollectedFaces()`, a la cual se le pasa como argumento el vector de rostros preprocesados obtenidos en la función anterior. A grandes rasgos, la función `Trainning()` se encarga de chequear que existan 2 o más personas en la base de datos antes de realizar el "*model"*, de lo contrario no tendría sentido realizarlo.

3.4. Recibir frames y responder

El proceso servidor utiliza la función `fork()` propia de Linux para duplicar procesos utilizando el método de *Light-Weight Process (LWP)* para atender las conexiones entrantes del cliente. Luego toma frames del mismo, los preprocesa y busca el parecido respecto al "*model*" usando `Buscaparecido()`.
La toma de los frames, la realiza la función `GetFrames()`, la cual merece atención, ya que la misma se encarga de rearmar el frame recibido, esto se hace debido a que se trabaja con variables tipo matrices.

`Buscaparecido()` a grandes rasgos lo que hace es, mediante el "*model"* y el rostro preprocesado actual, se reconstruye un modelo de face con los datos guardados. Luego con esta cara reconstruida y el frame actual se obtiene un valor de similitud y cuando dicho valor supera un valor de sensibilidad previamente establecido por nosotros, se obtiene mediante una función de OpenCV llamada `predict()`, la predicción de la persona, es decir, a quién más se parece el frame actual respecto a los guardados en la base de datos.

Lo dicho anteriormente se muestra a continuación en el diagrama de flujo de la figura.

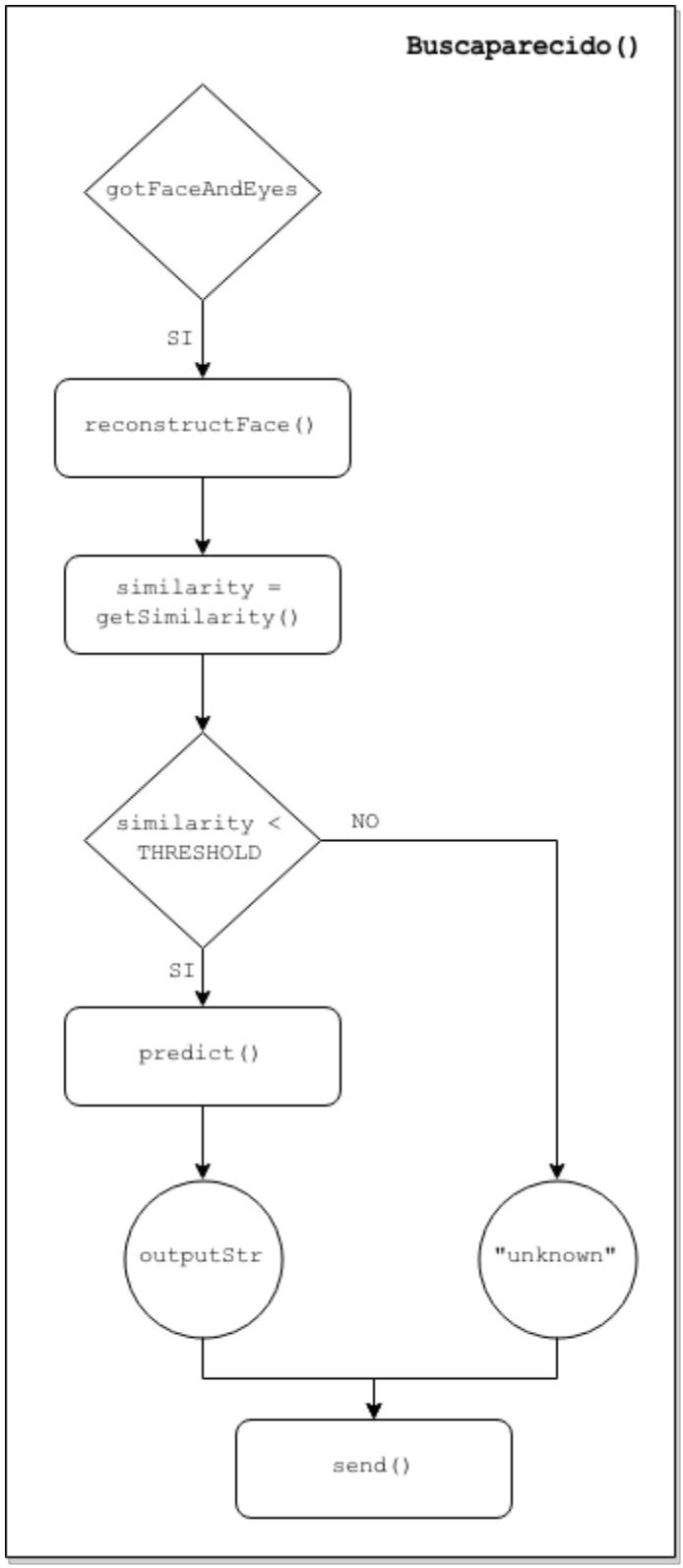

Fig.: 9 Función Buscaparecido()

Luego, y para finalizar, por TCP se envía el nombre de la persona usando una función propia de Linux, `send()`.

3.5. Cliente

Este proceso inicializa la webcam con `initWebcam()`, inicializa los detectores XML con `initDetectors()` al igual que el servidor y los sockets TCP con `init_TCP()`. Además usa la función `recognizeAndTrainUsingWebcam()` que toma los frames de la cámara y los envía al servidor por TCP con `send()`, en envío se realiza siempre y cuando se detecte rostros de personas. Esto lo hacemos así con el fin de no gastar recursos en frames que no lo merecen.

4. Anexos

4.1. Modo de uso

En esta sección se explicará brevemente cómo poner en funcionamiento el sistema, teniendo en consideración la versión que se esté empleando.

Puesta en ejecución versión uno:
Para asegurar un buen funcionamiento se deberá seguir el siguiente orden de ejecución:

1) Ejecutar el programa Servidor. El comando empleado para este caso sería:

```
$ ./Servidor
```

2) Una vez realizado el paso uno, ejecutar el programa Base de Datos, y realizar las cargas de personas que necesite empleando el botón “Agregar”. Finalice con “Terminar”. El comando empleado para este caso sería:

```
$ ./BD (IP servidor)
```

   En este punto tenemos cargada la base de datos, si desea modificarla deberá realizar nuevamente paso 1 y 2.

3) Active los clientes. Si todo marcha bien, se establecerá la conexión con el servidor, y comenzará a trabajar el cliente. El comando empleado para este caso sería:

```
$ ./Cliente (IP servidor)
```

Puesta en ejecución versión dos:
Para asegurar un buen funcionamiento se deberán seguir el siguiente orden de ejecución:

1) Ejecutar el programa Base de Datos, y realizar las cargas de personas que necesite empleando el botón “Agregar”. Finalize con “Terminar”. El comando empleado para este caso sería:

```
$ ./BDv2
```

2) Ejecutar el Servidor. El cual si no hubo problema en la carga de la base de datos dirá “*Base de datos actualizada”.* El comando empleado para este caso sería:

```
$ ./Servidorv2
```

3) Active los clientes. Si todo marcha bien, se establecerá la conexión con el servidor, y comenzará a trabajar el cliente. El comando empleado para este caso sería:

```
$ ./Cliente (IP servidorv2)
```

4) Si desea realizar una actualización de la Base de datos, ejecute paso uno. Si no hubo problema en la carga de la base de datos dirá “*Base de datos actualizada”.*

Obtención de la IP pública usando el terminal de Linux:

La dirección IP del servidor es necesaria para poder comunicarse con él. Para que esto sea posible es necesario que ejecute al comienzo lo siguiente:

```
$ ifconfig wlan0
```

Lo anterior se utiliza para el caso donde se encuentre conectado por Wi - Fi a internet. En cambio, si el servidor está conectado mediante Ethernet:

```
$ ifconfig eth0
```

4.2. Error al actualizar la base de datos

Cuando desarrollamos la segunda versión del sistema de acceso, nos topamos con un inconveniente cuya solución escapaba a nuestro conocimiento.

El proceso de actualización de la base de datos consiste en actualizar el archivo YAML tomando la información previa y los rostros procesados recientemente. Teniendo estos dos vectores lo que restaba utilizar las funciones específicas de OpenCV, pero aquí surge el error. Para entenderlo un poco mejor, comentaremos algo más sobre este tipo de archivos.

Los archivos de lenguaje de marcas como lo es éste y el XML poseen una estructura predefinida que debe mantenerse a la hora de actualizar el archivo. Según la documentación oficial de OpenCV, al utilizar la función creada para ello : `update()`, surgirá el siguiente error.

```
OpenCV Error: The function/feature is not implemented (This
FaceRecognizer does not support updating, you have to use
FaceRecognizer::train to update it.) in update, file ---,
terminate called after throwing an instance of 'cv::Exception'.
```

Además recomiendan cambiar de método de reconocimiento, como por ejemplo Eigenfaces, pero hemos comprobado que el error persiste.

Analizando el proceso de actualización un poco más, notamos que la función anterior no actualiza la lista previa sino que la agrega al final previamente separando la lista anterior con dos línea de comentarios. En sí lo que hace es anexar las dos listas como lo hace la función `append()` en *Python*.

Debido a este procedimiento, luego es que surge el error en la ejecución.
Buscando en foros una posible solución a este problema hemos encontrado lo siguiente:


#6 vincent....@gmail.com

Thanks, I'll report the bug on OpenCV.

#7 henrique...@gmail.com

Hi,
Do you fix the problem ?
I use the opencv248 and javacv0-7, And I'm facing with this problem:
http://answers.opencv.org/question/33363/how-to-createupdatedelete-an-face-stored-xml/

Could you help me please ?

Project Member #8 samuel.a...@gmail.com

Nope, it doesn't look like it was fixed in either OpenCV 2.4.8 or 2.4.9. Please ask about that on OpenCV's site. I am not responsible for what they do on their end.

Como puede verse en la imagen, y aunque no figure, el problema viene de versión anterior de OpenCV (el primer mensaje es del año 2012) y los siguientes mensajes son actuales (año 2014) y según verificó la persona del último mensaje el error no fue solucionado hasta la versión que nosotros utilizamos en este proyecto (*v2.4.9*).

5. Bibliografía

- “Mastering OpenCV with Practical Computer Vision Projects” - Packt Publishing (2012).

- “Learning OpenCV” - Bradski & Kaehler - Ed. O’Reilly (2008).

- “OpenCV API Reference” - http://docs.opencv.org/

- “Practical OpenCV” - Samarth Brahmbhatt

- OpenCV 2.4 Cheat Sheet (C++)