Asignatura:

Programación lll

Facilitador:

Willis Ezequiel Polanco Caraballo

Asignación:

Documento de Arquitectura IA

Sustentantes:

Felipe Mejía	2018-7235
Hansel Cabrera Espinal	2018-6499

INTRODUCCIÓN

El estudio de los mecanismos de la visión humana es el objetivo de la Visión Artificial o también llamada Visión por Computador. La visión computacional busca crear sistemas que sean capaces de reconocer un objeto determinado en una imagen. Para los humanos esta tarea es natural y es realizada constantemente sin preocuparse por el cómo; pero cuando trasladamos esta tarea a una máquina, nos encontramos con varias dificultades que resolver

La principal dificultad en el proceso de reconocimiento de rostros y reconocimiento de gestos, radica en las enormes variaciones que existen entre las distintas imágenes del rostro o gesto de un mismo individuo. Estas diferencias son dadas por los gestos, la intensidad o ausencia de luz, diferentes peinados, maquillaje y accesorios, por los cuales una misma persona puede parecer diferente en dos imágenes

Este proyecto se enfoca en estudiar los diferentes métodos y algoritmos que permiten detectar un rostro y reconocimiento de señas en una imagen, y desarrollar una aplicación que pueda ser entrenada para realizar el reconocimiento de una persona especifica dentro de la imagen al igual que permita reconocer gestos de auxilio.

OBJETIVOS

Objetivo general Crear un prototipo de aplicación que permita entrenar una red neuronal, utilizando técnicas de visión artificial y la biblioteca OpenCV, para reconocer el rostro de una persona en una imagen y gesto de auxilio realizado por la mano

Objetivos específicos

- Identificar, analizar y evaluar los diferentes algoritmos de la visión artificial, que permitan realizar el reconocimiento de un objeto en una imagen.
- Analizar y evaluar las técnicas de visión artificial y de biometría que permitan la identificación de una persona mediante sus características faciales, para utilizarlas en el proceso de identificación de una persona mediante su rostro.
- Implementar una red neuronal que permitan reconocer el rostro de una persona y gestos en una imagen, mediante técnicas y algoritmos de entrenamiento implementados por la biblioteca de visión artificial - OpenCV.

La visión artificial provee conceptos y técnicas que, aplicadas mediante algoritmos y redes neuronales, permiten detectar un objeto especifico presente en una imagen. Una de las bibliotecas de visión artificial que implementa y nos permite realizar este trabajo es OpenCV, esta biblioteca implementa algoritmos para la detección de objetos y también para la detección de rostros en imágenes y video.

OpenCV nos facilita la terea de detectar rostros y gestos, ya que cuenta con clasificadores entrenados para esta tarea, además tiene implementado el algoritmo de Viola-Jones (Viola & Jones, Rapid object detection using a boosted cascade of simple features, 2001), el cual permite realizar este proceso de detección de rostros de forma sencilla y a muy bajo costo computacional.

En caso de que lo necesitemos podemos crear nuestros propios clasificadores, OpenCv nos permite entrenar nuestros propios clasificadores utilizando AdaBoost (OpenCv).

La biblioteca OpenCV puede ser integrada con diferentes lenguajes de programación como: C++, Java, Python y los lenguajes de la plataforma .Net; por lo cual, es posible utilizarla para crear una aplicación para entornos web o de escritorio, que permita la detección de rostros dentro de una imagen tomada mediante una cámara.

MARCO TEÓRICO

La visión artificial o visión por computador, pretende reproducir artificialmente el sentido de la vista humana mediante el procesamiento e interpretación de imágenes captadas desde diferentes dispositivos, como cámaras y utilizando computadores para su procesamiento. El sistema de visión artificial requiere de dos elementos fundamentales, el primero es el hardware encargado de la percepción de las imágenes y el segundo el software encargado del procesamiento de la información (Pajares & De La Cruz, 2004). 12 debido a la complejidad del proceso, la visión artificial se ha dividido en varias etapas o procesos. En cada una de ellas, las imágenes y la cantidad de información se va refinando hasta lograr el reconocimiento del objeto buscado. Generalmente se consideran cuatro procesos (Sánchez, 2002):

- Captura: Consiste en la captura de las imágenes por medio de un sensor.

- Análisis (Tratamiento digital): Mediante la aplicación de filtros se descartan partes de la imagen y se enfoca en las partes con mayor probabilidad de coincidencia con lo buscado.

- Segmentación: Consiste en aislar los elementos de la imagen que se desean analizar, La segmentación permite comprender la imagen individualizando sus elementos.

- Reconocimiento (Clasificación): En ella se distinguen los objetos segmentados, mediante al análisis de ciertas características, que se establecen previamente para diferenciarlos. Siguiendo estos procesos e implementándolos mediante algoritmos de procesamiento y apoyándose en redes neuronales, es posible identificar el objeto buscado dentro de una imagen.

MARCO CONCEPTUAL

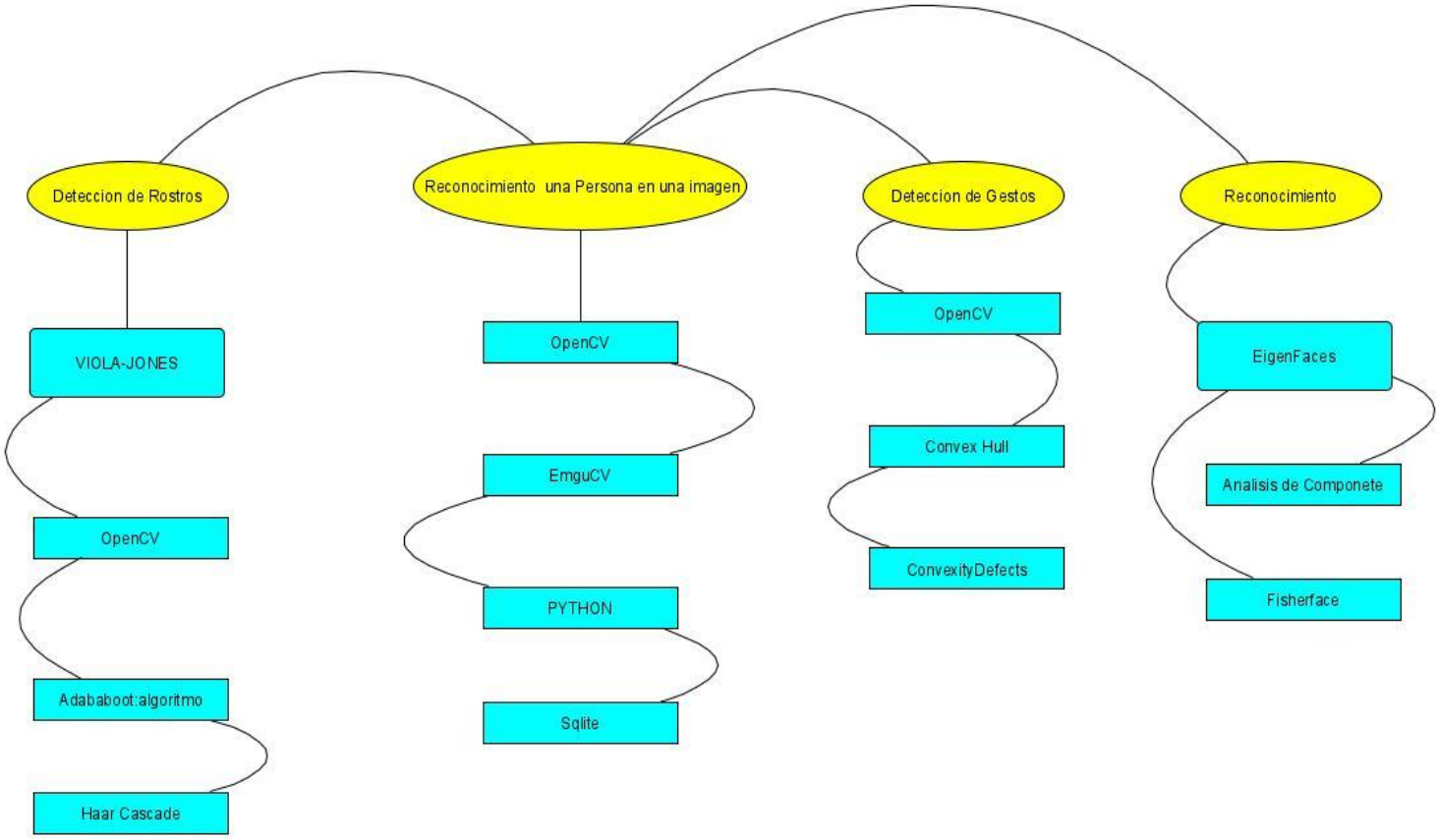
El reconocimiento automático de rostro busca extraer los rasgos significativos de un rostro presente en una imagen, ponerlos en una representación útil, realizar algún tipo de clasificación en ellos y lograr, mediante algoritmos de detección y patrones, la ubicación de un rostro en la imagen.

El reconocimiento facial basado en características fisiológicas y geométricas, es probablemente el método más intuitivo y utilizado para el reconocimiento de un rostro. Uno de los primeros sistemas automáticos de reconocimiento facial utilizó posición de los ojos, oídos y nariz, para construir un vector de características (distancia entre los puntos, el ángulo entre ellos) (PHILLIPS & et al., 2005).Para lograr el reconocimiento de un rostro en una imagen se realizan dos etapas de procesamiento con sus respectivas técnicas y algoritmos.

En la primera etapa se utilizan clasificadores en cascada y el algoritmo de Viola-Jones para lograr detectar un rostro dentro de la imagen (Viola & Jones, Robust real-time face detection). Viola-Jones utiliza clasificadores en cascada los cuales se entrenan utilizando el algoritmo de AdaBoost (Mayhua-Lopez, Gómez-Verdejo, & Figueiras-Vidal, 2012), con el cual se logra crear clasificadores fuertes a partir de clasificadores débiles.

En la segunda etapa se realiza el reconocimiento del rostro de una persona, comparando el rostro detectado en la imagen con una seria de imágenes guardadas llamadas imágenes de muestra. Para este reconocimiento, se utiliza el algoritmo de Eigenfaces (Turk & Pentland, Eigenfaces for recognition, 1991), el cual realiza la comparación de las imágenes de muestra con la del rostro detectado, e indica si existe concordancia. Para observar la relación de estos conceptos con las etapas realizadas para la detección y reconocimiento, se elaboró el siguiente mapa conceptual. En él se observan los conceptos más relevantes en el proceso y su relación con el proceso general que permite el reconocimiento de rostros utilizando la librería de visión artificial OpenCV (OpenCv, 2017).

Mapa de Conceptos para detección de gestos y reconocimiento de Rostros en Imágenes



Sistema de reconocimiento facial

Es una aplicación informática que se utiliza para identificar personas a partir de una imagen o un video. Se utiliza principalmente con fines de seguridad para rastrear quién ingresa a una instalación determinada o para buscar a alguien en un lugar determinado. es tan apropiado como un escáner biométrico o de iris, pero es mucho más fácil de implementar. Este sistema de reconocimiento de rostros funciona con una base de datos donde se guardan las imágenes de los rostros de las personas. Este sistema ejecuta un par de algoritmos para determinar si el rostro de la persona en la cámara o el metraje coincide con cualquier imagen de la base de datos.

En **grayScale** formato. Otro algoritmo busca una cara. Otro algoritmo compara si un área, tamaño o forma de la cara coincide con la base de datos. El efecto combinado de todos estos algoritmos es este sistema. Para hacer **Face Recognising System** uno básico, la mejor manera es use **openCV.**

Reconocimiento facial con Python y OpenCV


Face Recognizer
Bienvenido HANSEL!!!
HANSEL

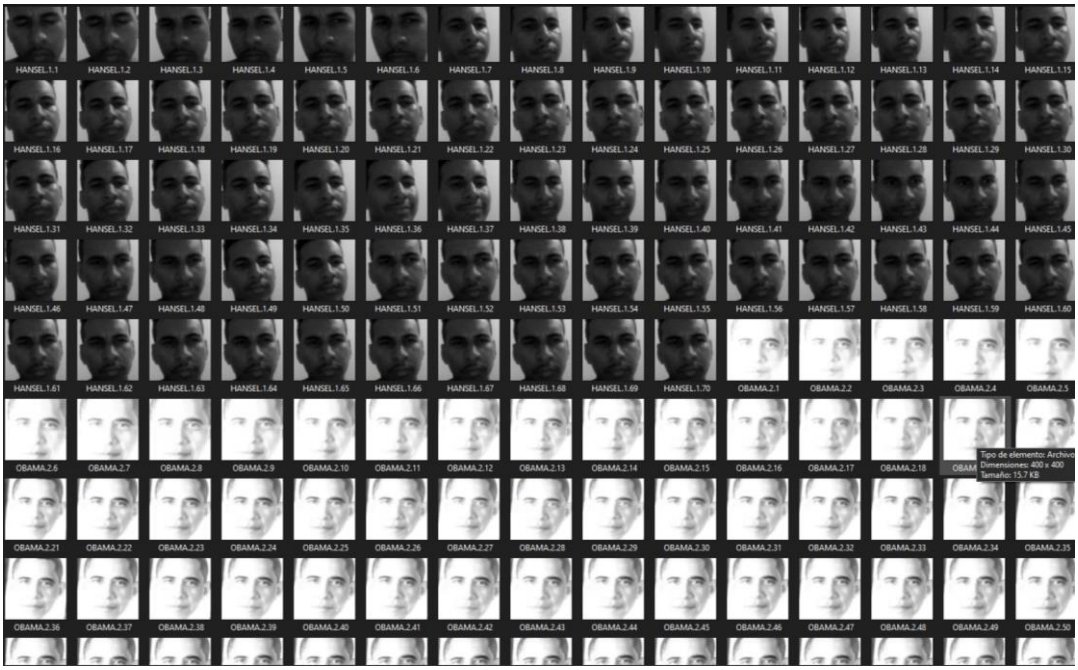

Face Recognizer
Bienvenido ELONK MUSK!!!

Lógica del programa

Este programa solicita a una persona que proporcione una identificación única que solicite:

- ✓ Nombre
- ✓ Carrera
- ✓ Matricula

Luego recopila 70 imágenes rostros de la persona y pasa por un proceso de conversión en grayScale



Donde se guardan todos los datos de esa persona en la base de datos o las actualizaciones si existente.

Las imágenes se codifican como matrices. En particular, las imágenes de intensidad o escala de grises se codifican como una matriz de dos dimensiones, donde cada número representa la intensidad de un pixel. Pero eso significa que cualquiera de estas matrices que generamos se puede visualizar como una matriz. Para visualizar imágenes, usamos el módulo **pyplot** de la librería **matplotlib**.

Esta parte toma fotos de caras, las convierte a grayScaleformato, luego compara las fotos con las fotos de la base de datos. Cuando se encuentra el fósforo, muestra toda la información en la pantalla. Sin el entrenador, el reconocimiento facial adecuado no es posible.

Guarda todos los datos de esa persona en la base de datos o las actualizaciones si existente. cv2Aquí está la biblioteca openCV y el uso de la base de datos es sqlite3.

El **clasificador en cascada** haarcascade_frontalface_default.xml se utiliza para el sistema de reconocimiento facial.

FisherFaces

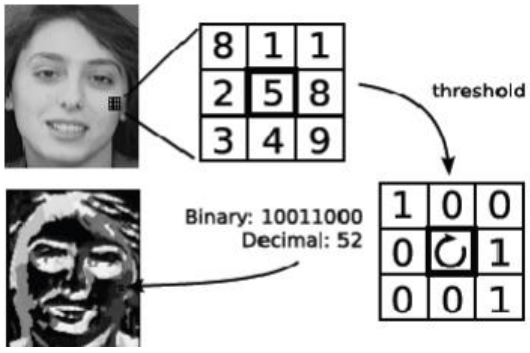
Esta técnica considera las imágenes de entrenamiento de un mismo individuo como clases, por lo tanto, existen el mismo número de clases que personas. Una vez definida las clases se procede a calcular dos matrices: la matriz de dispersión entre clases y la matriz de dispersión dentro de clases. Una vez calculada estas matrices se obtiene una matriz de proyección donde cada columna será la base del nuevo sub-espacio, denominada Fisherfaces.

Para realizar el cálculo de la matriz de proyección, requiere que las matrices de dispersión sean no-singulares (que posean inversa), esto no siempre es posible debido a que el número de imágenes casi siempre es menos al número de pixeles de cada imagen. Para solucionar esto, se utilizan Componentes Principales en las matrices de dispersión para reducir su dimensión. Las FisherFaces también permiten una reconstrucción de la imagen, por lo tanto, también se utiliza la comparación de imágenes mediante la distancia euclidiana.

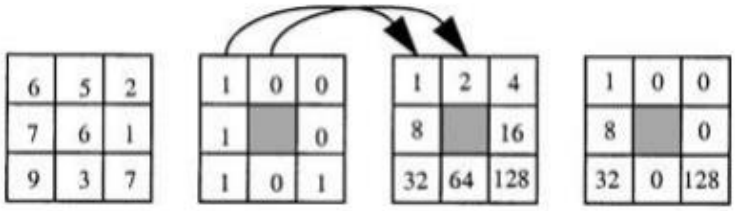
Local Binary Pattern

Local BinaryPattern o Patrón Binario Local es un operador de textura que etiqueta los pixeles de una imagen por thresholding o umbral, método de segmentación de imagen que a partir de una en escala de grises se crea una imagen binaria, cada vecino del pixel y el resultado de la operación se considera como un numero binario. Una de las propiedades más importantes es su indiscriminación frente a los cambios en la escala de grises como por ejemplo la luminosidad.

En este algoritmo se dice que para cada pixel de la imagen a procesar se examina su 8- vecindad alrededor del pixel central como se muestra en la figura X-X de dimensiones 3x3. Para cada pixel en la vecindad se determinara la intensidad de ella por medio de dos valores: 1 si el valor es mayor al del pixel central o 0 si el valor es menor al del pixel, como se puede ver en la figura 2.8 el valor del píxel central es 5 por lo tanto 8 es mayor que él y en la matriz resultante se agregara un 1 en ese mismo lugar y los valores se codificaran en una cadena binaria formada los 8 números, a veces se transforma a número decimal para el mejor funcionamiento del algoritmo.



Al ya tener calculada la matriz de número con valores 1 y 0 se tienen 2 elevado a 8 posibles patrones, como podemos ver en la figura 2.9 cada valor del pixel se multiplica con la matriz para así calcular la distribución de textura de forma similar a los histogramas de escala de grises.



LBP=1+8+32+128=169

Para comparar dos imágenes se deben comparar los histogramas de cada una de ellas a través de la siguiente formula:

$$f(G,H) = \frac{\sum_{i=1}^{256} min(G_i, H_i)}{n}$$

En donde G, H son los histogramas de dos imágenes creadas por la técnica de la vecindad, Gi, Hi es el número de valores iguales en los histogramas respectivamente y n es el número de pixeles

Como se puede ver en la figura la fotografía del rostro de la persona se ha dividido en bloques de igual tamaño, aunque no es necesario que sean iguales, y a cada uno de ellos se le ha calculado un histograma único el cual se ira complementando con los demás para así crear uno para toda la imagen.

Cabe destacar que cada histograma se ve referenciado por los tonos grises de la imagen y es por esto que posee varios niveles de altura en su gráfico.

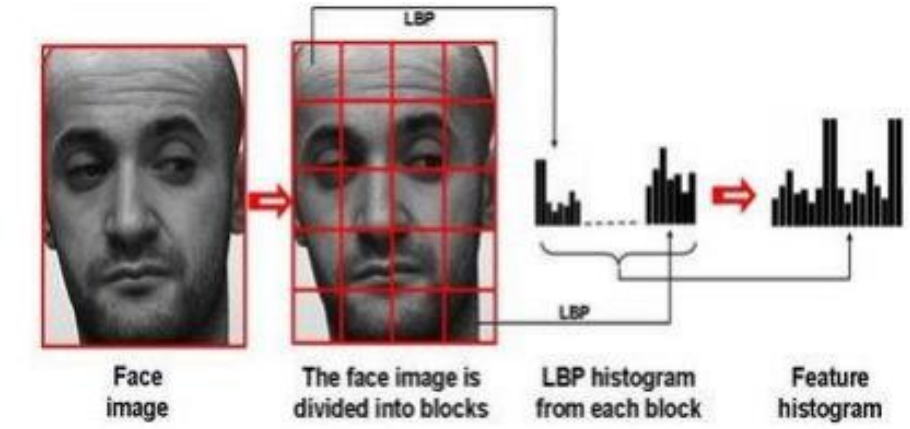
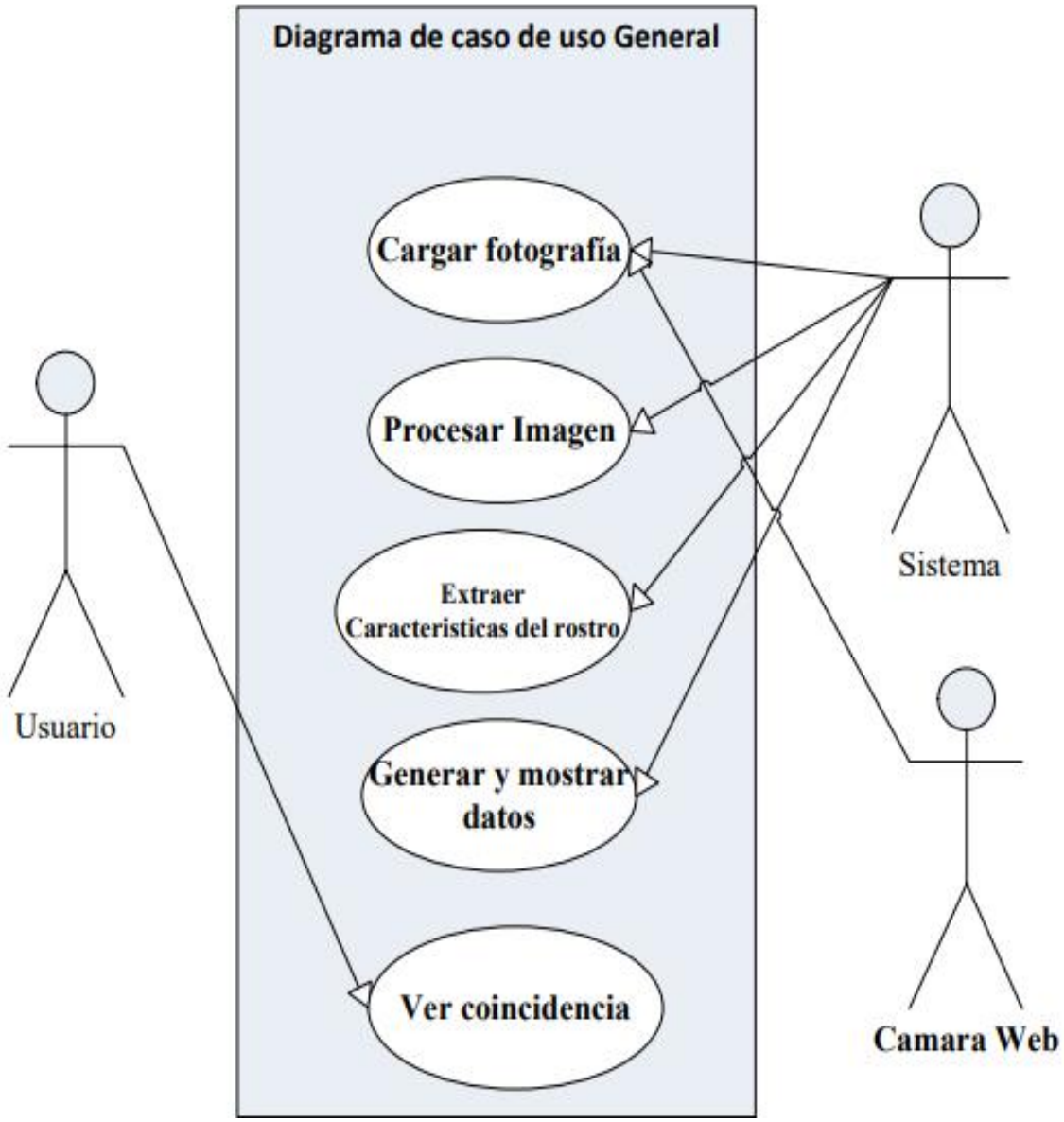


Diagrama de Caso de uso



Caso de uso Cargar fotografía

El siguiente caso de uso permitirá ingresar un usuario al sistema por medio de una imagen tomada por la cámara web.

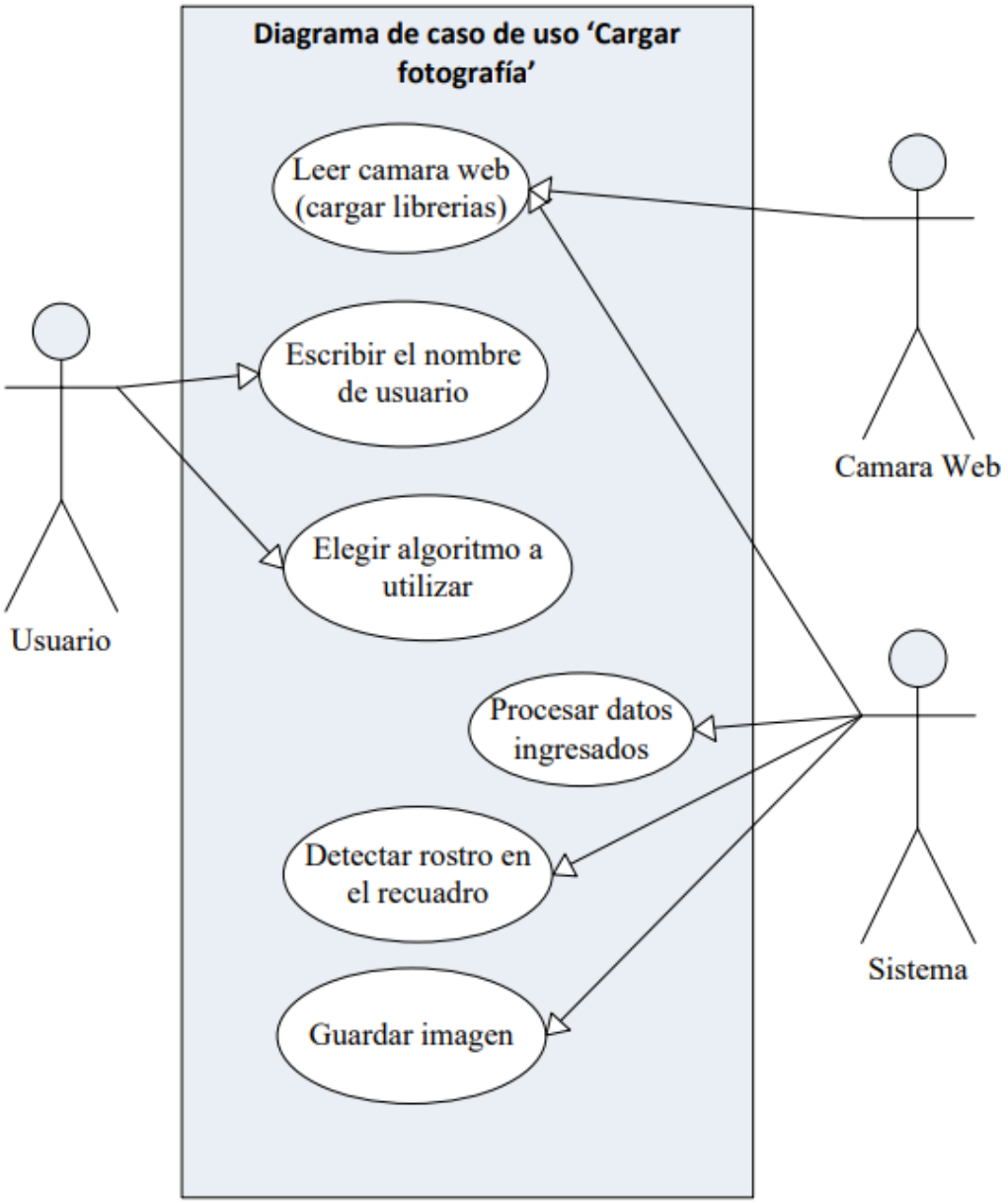


Diagrama de secuencia Detectar Rostro

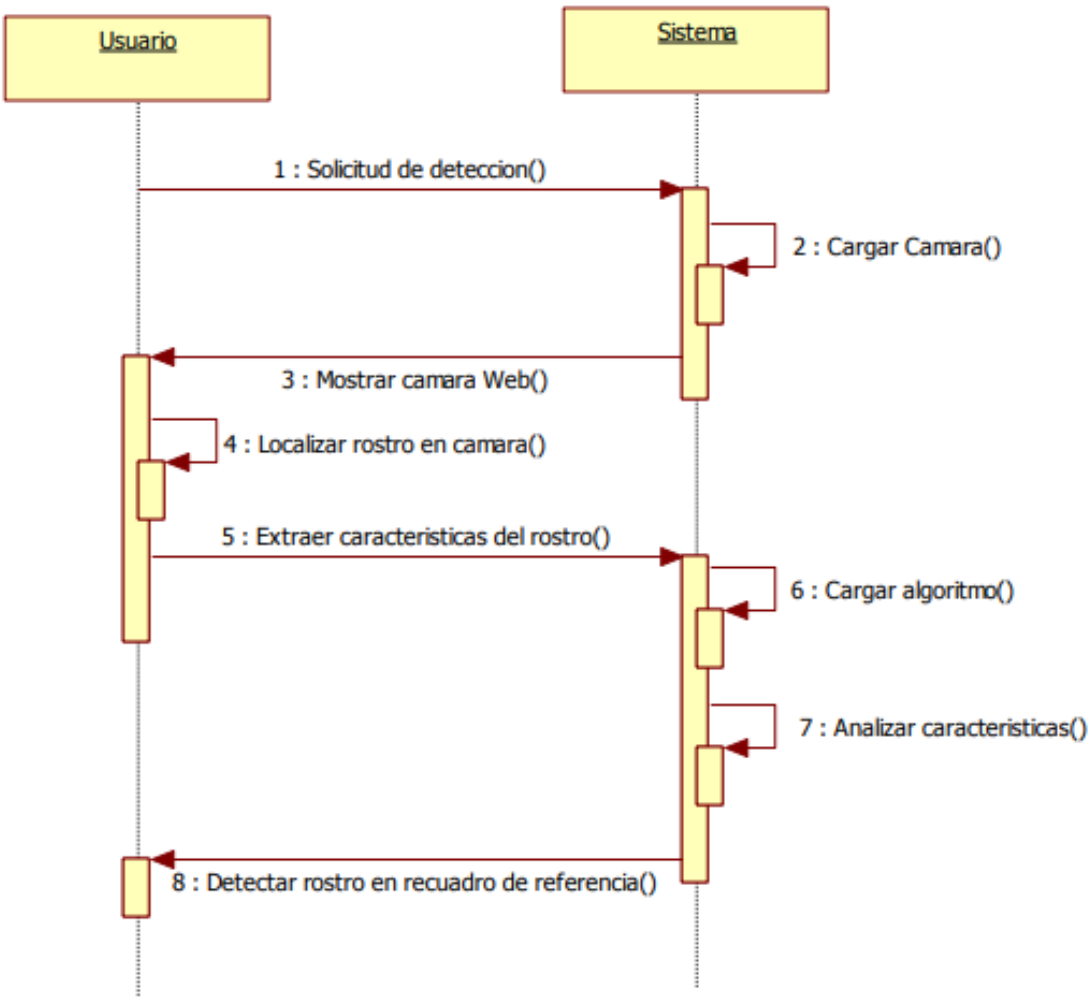


Diagrama de secuencia Reconocer Rostro

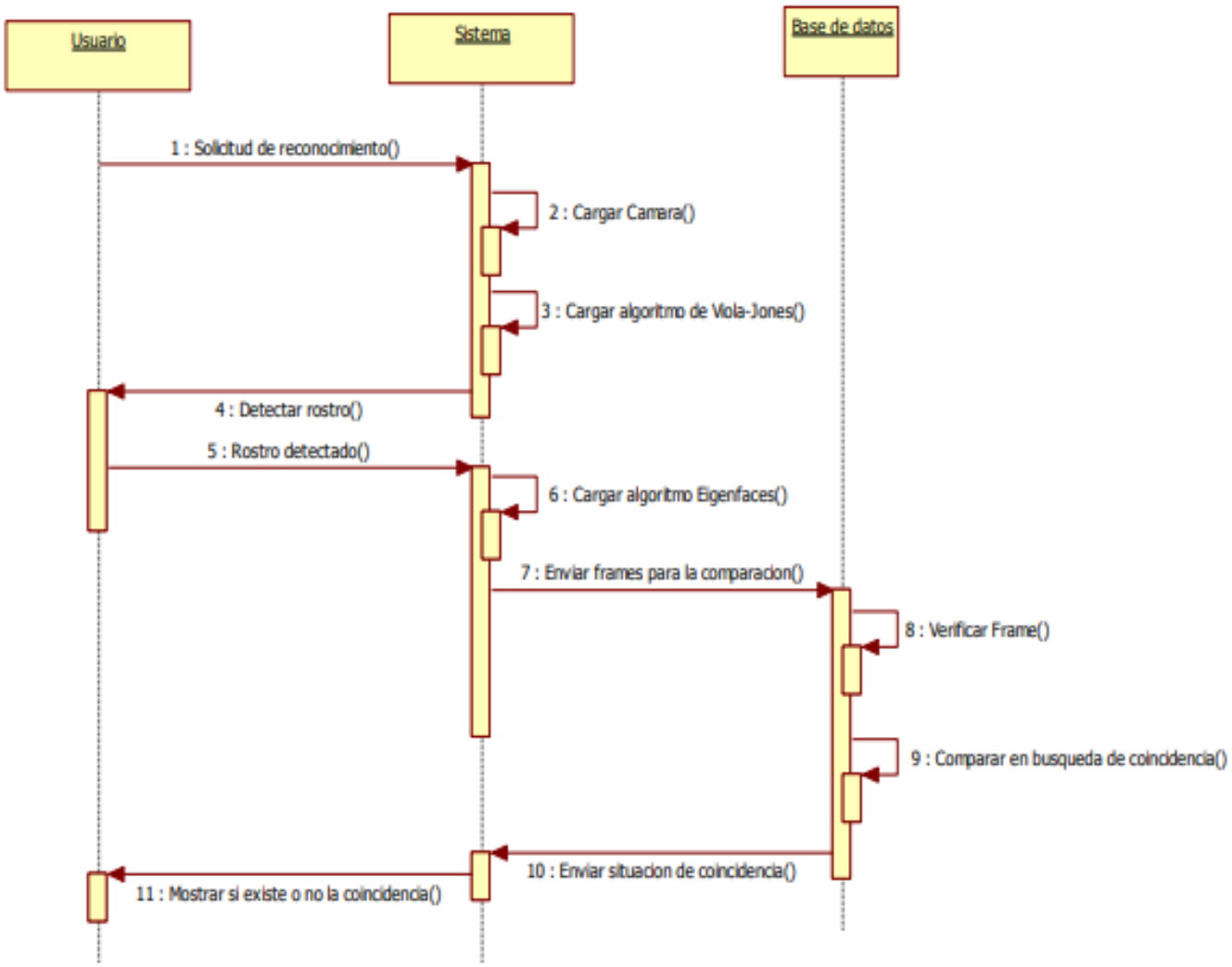
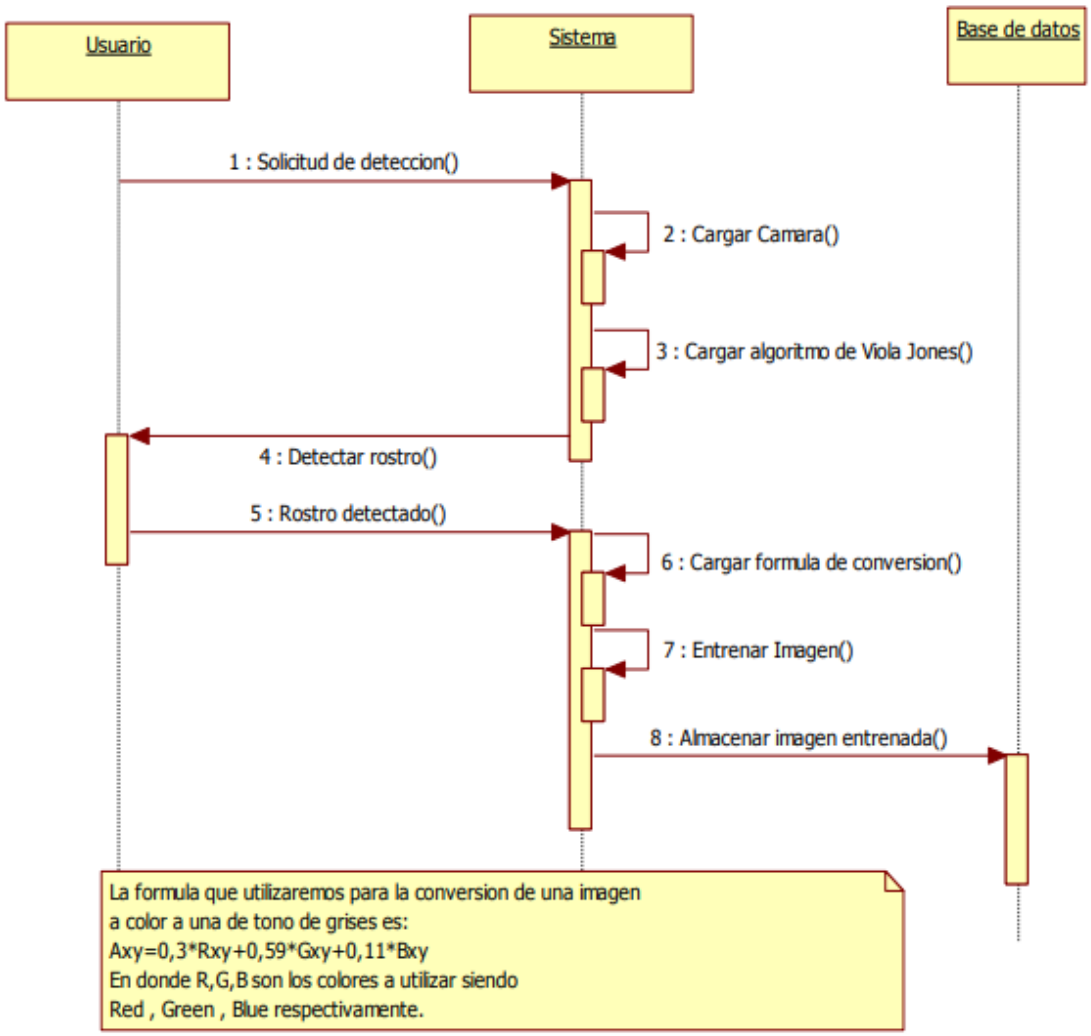


Diagrama de secuencia Convertir imagen a grises



Diagramas BPMN

El principal objetivo de BPMN es proporcionar una notación estándar que sea fácilmente legible y entendible por parte de todos los involucrados e interesados del negocio. Entre estos interesados están los analistas de negocio, los desarrolladores técnicos y los gerentes y administradores del negocio. En síntesis, BPMN tiene la finalidad de servir como lenguaje común para cerrar la brecha de comunicación que frecuentemente se presenta entre el diseño de los procesos de negocio y su implementación.

Diagrama de arquitectura enfocado en la interoperabilidad

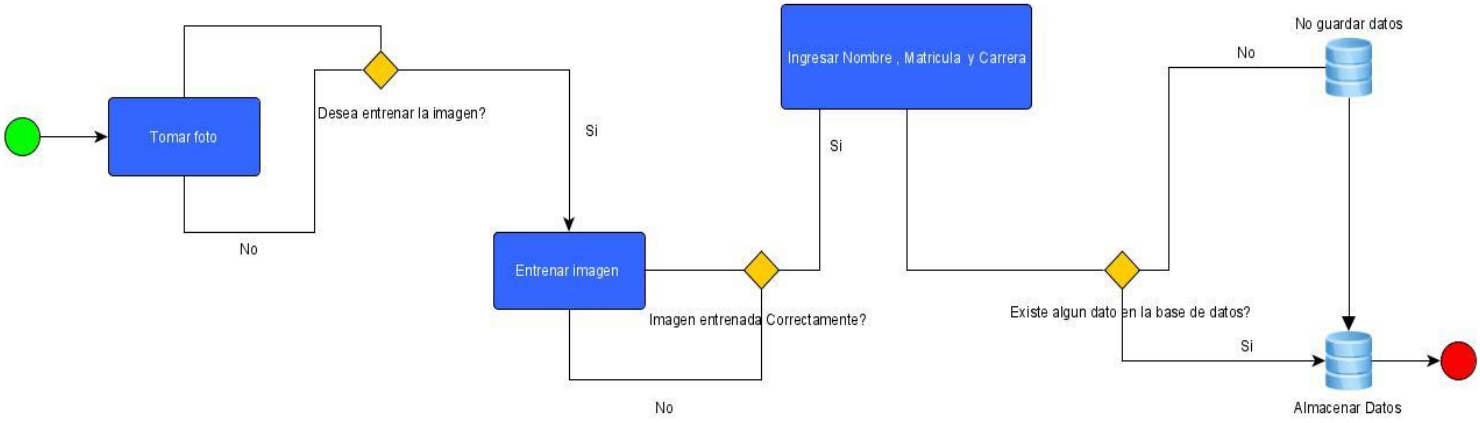
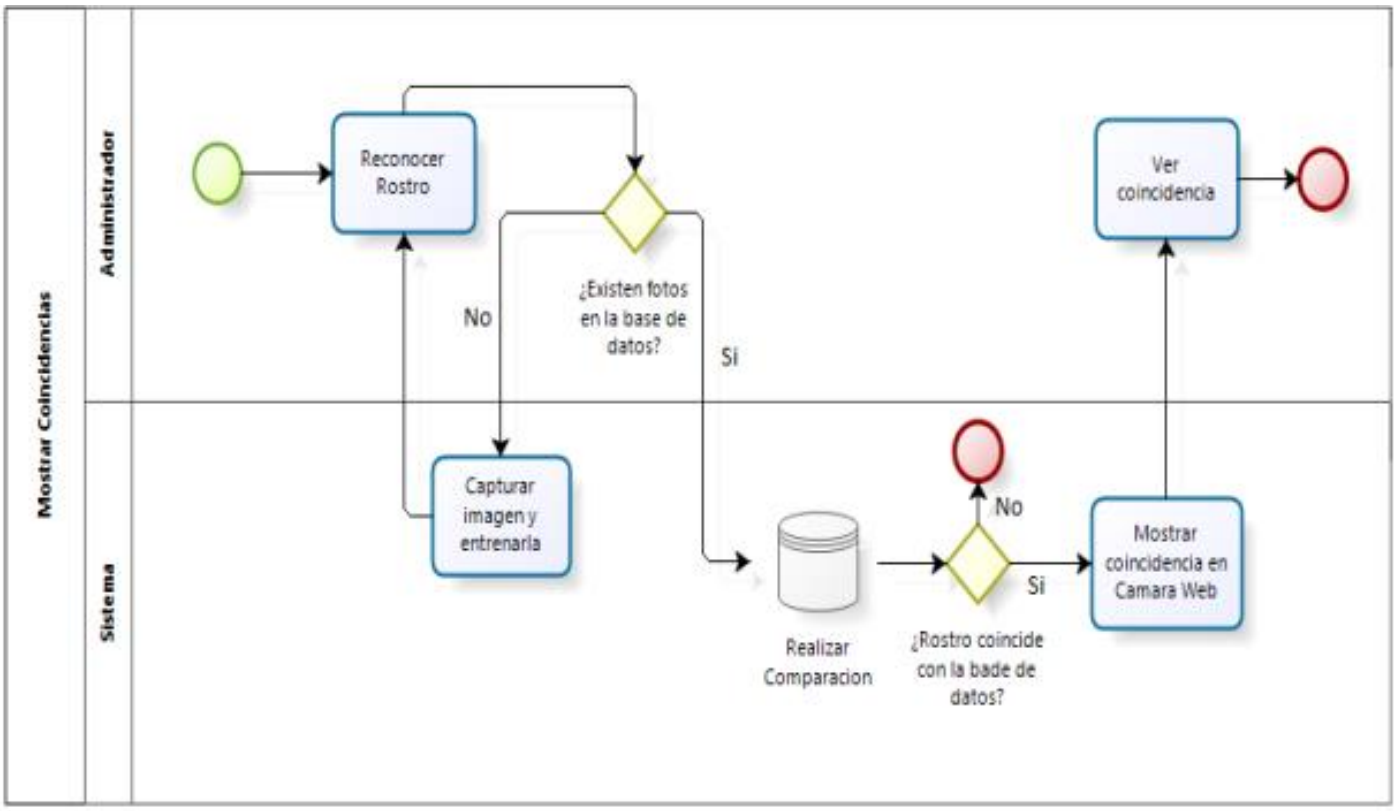


Diagrama BPMN Mostrar Coincidencia



Entrenamiento de rostros

Estas imágenes permiten ahorrarnos el paso del cambio de color a gris por medio de un algoritmo en el software y a su vez poder entrenarlas inmediatamente.

Como se muestra en la figura a cada imagen original se le ha aplicado un algoritmo en particular ya sea:

- Eigenfaces.
- FisherFaces.
- Laplacianfaces.

Esto permite saber qué imágenes son aptas para poder reconocerlas y cuales se deben rechazar por no ser aptas para entrenarlas y así se podrán realizar estudios sobre qué algoritmo posee menor taza de error y cuál es el que permite mayor tolerancia a la luz o a los movimientos de la cabeza.

Especificación de caso de uso Detectar Rostro

Caso de uso	Detectar Rostro
Actor Principal	Sistema
Participantes e Intereses	Sistema: El sistema debe generar la detección del rostro de la persona situada en la cámara web.
Precondiciones	Algoritmo Viola Jones cargado y Cámara web encendida
Post condiciones	Detección de rostro por medio de un recuadro de referencia
Escenario Principal	1.- El Administrador da inicio al programa 2.- El Sistema carga la cámara web 3.- El Sistema ejecuta el algoritmo de Viola-Jones 4.- Se detectan las características faciales de la persona Detecta el rostro de la persona
Extensiones	3.1.- Elegir color de recuadro de referencia
Requisitos Especiales	----------
Frecuencia de Ocurrencia	Muy Alta

Especificación de caso de uso Reconocer Rostro

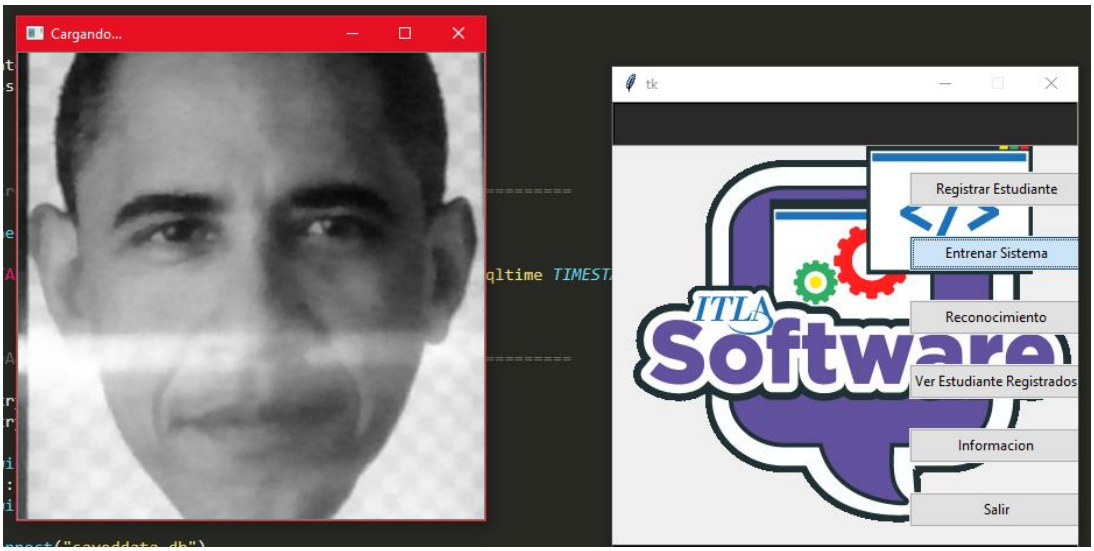
Caso de uso	Reconocer Rostro
Actor Principal	Sistema
Participantes e Intereses	El sistema luego de detectar y ejecutar la comparación del rostro de la persona con la Base de Datos, reconoce ésta y muestra si coincide o no.
Precondiciones	Rostro de la persona detectado e imágenes en la Base de Datos.
Post condiciones	Reconocimiento facial de la persona con nombre en el recuadro de referencia.
Escenario Principal	1.- El administrador da inicio al programa. 2.-Sistema detecta el rostro. 3.- Sistema carga el algoritmo de Eigenfaces. 4.- Sistema compara los frames del rostro de la imagen en la cámara web con las imágenes de la Base de Datos. 5.- Muestra la coincidencia de la persona en un nombre encima del recuadro de detección.
Extensiones	5.1.- Elegir el color del nombre de la coincidencia.
Requisitos Especiales	----------
Frecuencia de Ocurrencia	Muy alta.

Especificación de caso de uso Convertir Imagen a Grises

Caso de uso	Convertir imagen a grises.
Actor Principal	Sistema.
Participantes e Intereses	El Sistema debe ser capaz de convertir una imagen a color a una en tono de grises.
Precondiciones	Rostro detectado.
Postcondiciones	Imagen entrenada en todo de grises.
Escenario Principal	1.- El administrador da inicio al programa. 2.- Sistema detecta el rostro de la persona. 3.- El administrador debe seleccionar la opción de entrenar imagen. 4.- El sistema multiplica los pixeles de la imagen a color por una fórmula Matemática. 5.- Genera la imagen entrenada.
Extensiones	5.1.- Imagen entrenada se almacena en una Base de datos local.
Requisitos Especiales	----------
Frecuencia de Ocurrencia	Muy alta.

Interfaz de entrenamiento Interfaz de entrenamiento

En la interfaz de entrenamiento se producen todos los pasos para entrenar el rostro de la persona. Como se muestra en la figura es una interfaz muy sencilla y fácil para la manipulación del usuario.



Esta interfaz como la desarrollada anteriormente posee una ventana en donde se cargara la cámara web para poder entrenar la imagen, el método para realizar este proceso primero que todo es detectar el rostro de la persona, una vez detectado clickear el botón Registrar el cual agregara la imagen entrenada a la base de datos local anterior a esto se debe escribir el nombre de la persona en el label Nombre, ya teniendo estos pasos realizados el sistema está en óptimas condiciones para poder realizar el reconocimiento facial. Algunas opciones extras integradas en esta interfaz son, primero que todo, el poder entrenar hasta 70 imágenes en un solo clic lo que ayudara a desarrollar el entrenamiento de forma más rápida y el Delete Data el cual eliminara todos los registros de la base de datos.

Diagrama UML de clases

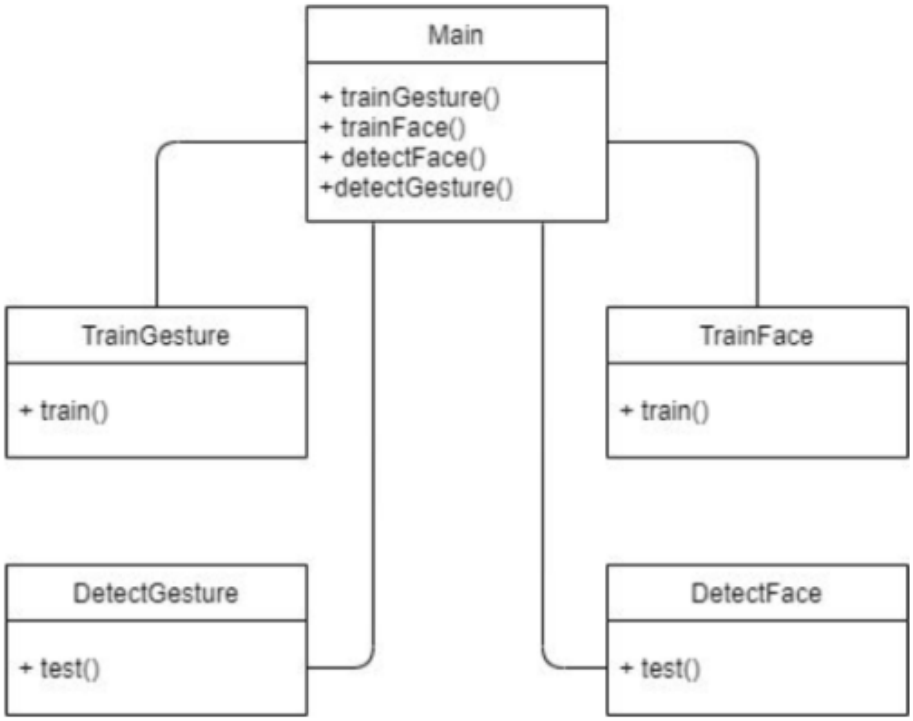
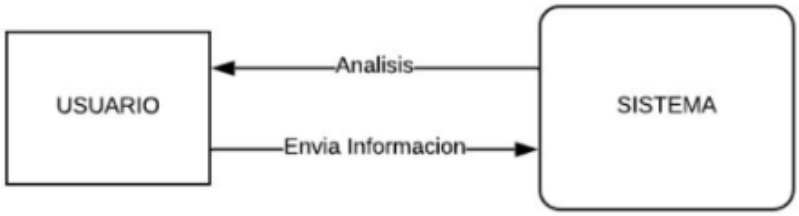
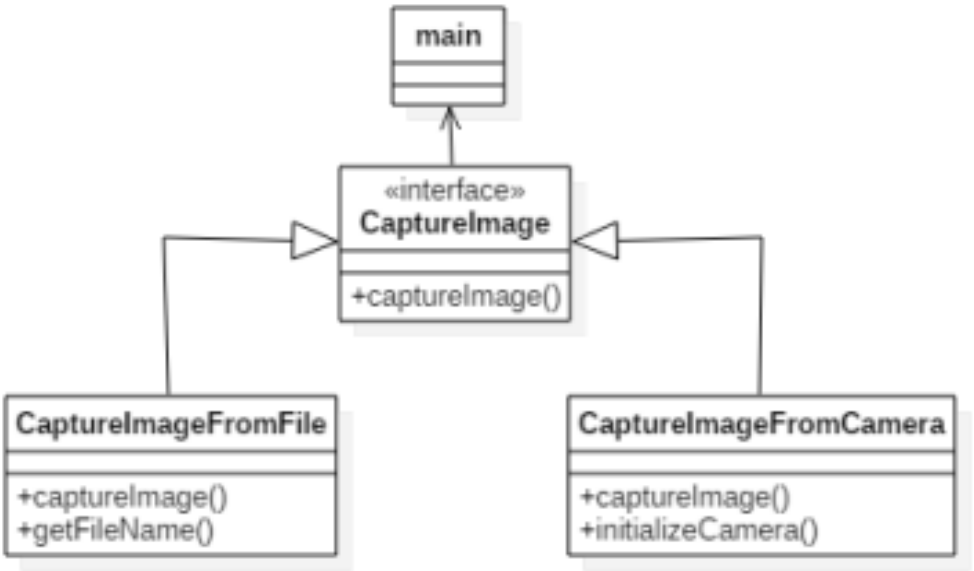
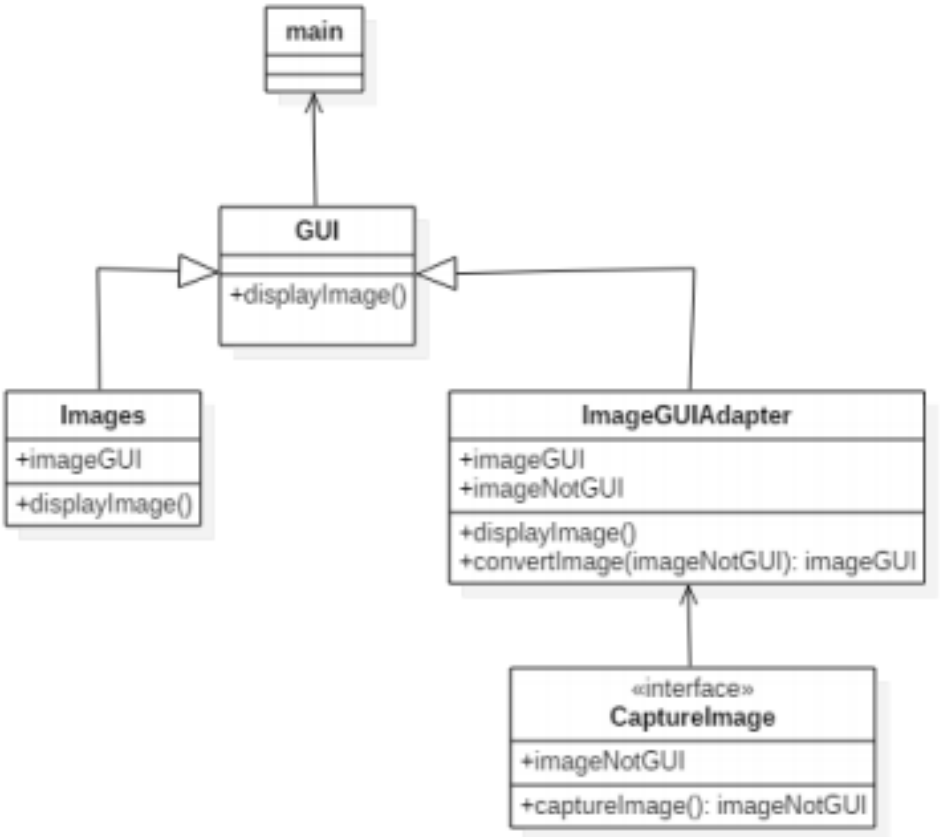
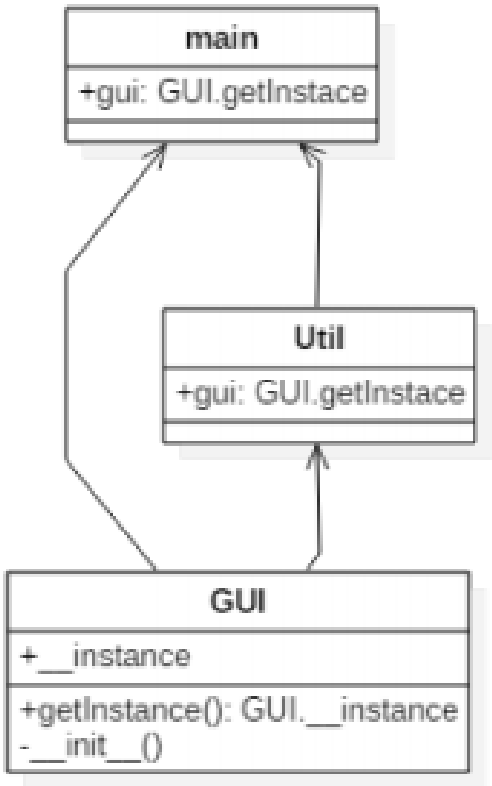


Diagrama Conceptual del sistema






main
GUI
+displayImage()
Images
+imageGUI
+displayImage()
ImageGUIAdapter
+imageGUI
+imageNotGUI
+displayImage()
+convertImage(imageNotGUI): imageGUI
«interface»
CaptureImage
+imageNotGUI
+captureImage(): imageNotGUI

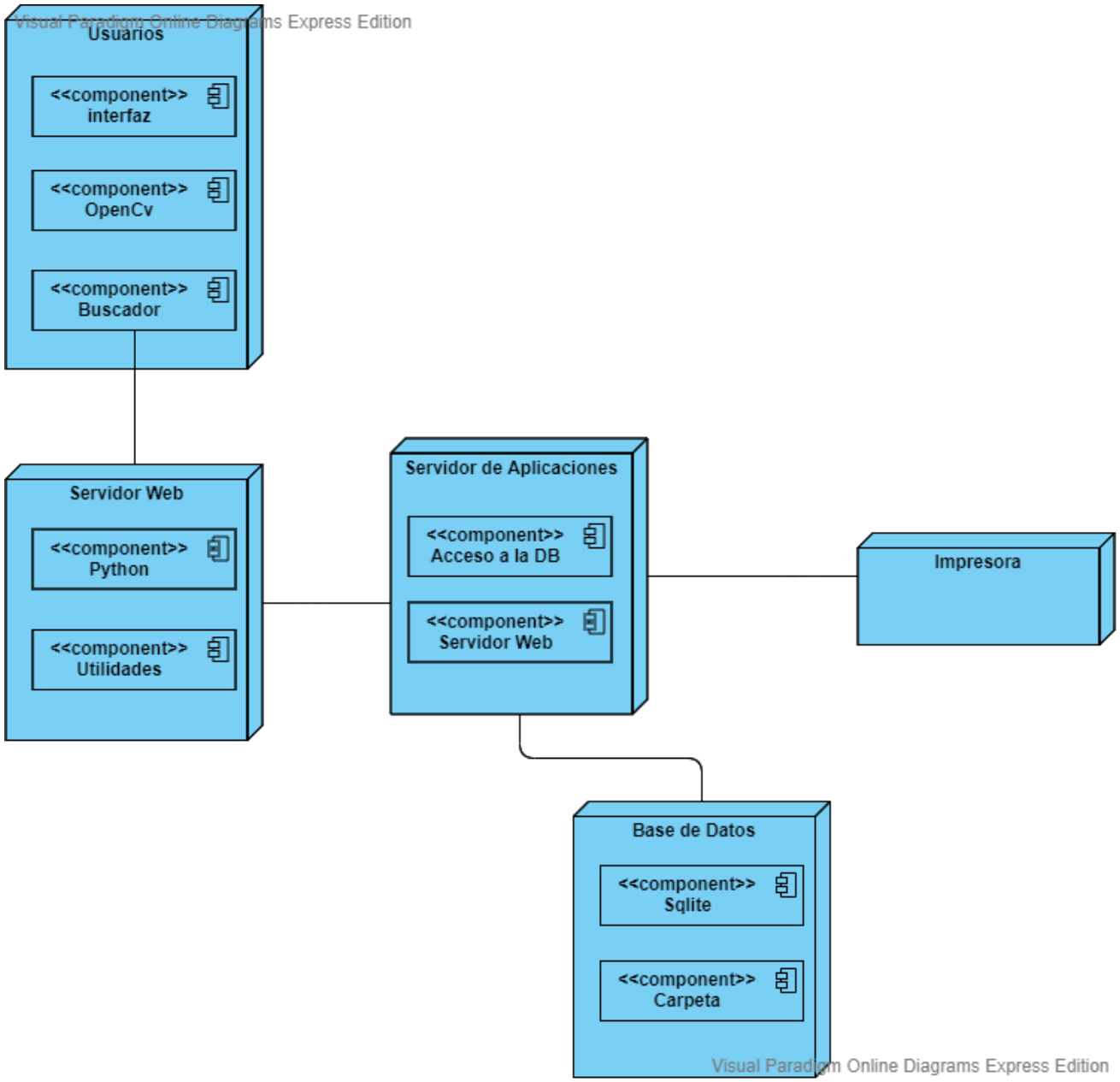

main
+gui: GUI.getInstace
Util
+gui: GUI.getInstace
GUI
+__instance
+getInstance(): GUI.__instance
-__init__()

Base de datos

Al principio la idea era diseñar una base de datos que, aunque sencilla, tuviera algún tipo de persistencia para poder asignar los datos a cada imagen. Se hicieron pruebas con SQLite, también se decidió utilizar una carpeta predefinida como sistema de base de datos provisional.

Se elige la opción de SQLite. Dado que no requiere de soporte de un servidor, no ejecuta un proceso para administrar la información si no que implementa un conjunto de librerías encargadas de la gestión. Tampoco necesita configuración, liberando al programador de todo tipo de configuraciones. Además, SQLite crea un archivo para el esquema completo de una base de datos, de esta manera los datos de las aplicaciones no pueden ser accedidos por contextos externos. Por todas estar razones y además por ser de código abierto se ha elegido la opción de SQLite.

Diagrama de Componentes



Herramientas Utilizadas

Algunas de las herramientas que hemos utilizado para el desarrollo del proyecto han sido los siguientes:

Herramientas de desarrollo

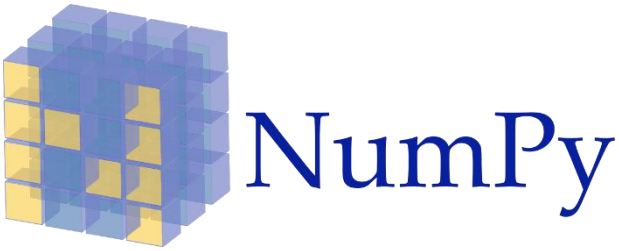
- El lenguaje de programación ha sido Python

- Para realizar la instalación del entorno virtual utilizado durante el desarrollo del trabajo se ha utilizado SublimeText

- Se ha utilizado la herramienta OpenCV para realizar el reconocimiento facial con computer-vision.

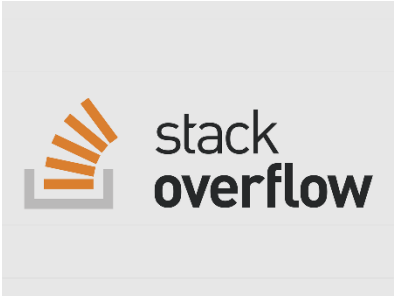
- Para tratar y modificar imágenes se ha utilizado Pillow (PIL para Python).
- Pandas es una biblioteca de software escrita como extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python.

- NumPy es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas



- Para el desarrollo de la interfaz se ha utilizado tKinter para Python

- Se han obtenido respuestas y soluciones a varios de los problemas encontrados durante la realización del proyecto en stackoverflow



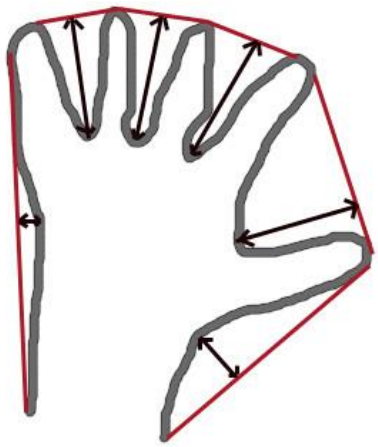
Herramientas de planificación y diseño

Para las tareas semanales/mensuales y llevar un control de las versiones del proyecto se ha utilizado Github y su cliente nativo para el escritorio de Windows Github Desktop

Sistema de Reconocimiento de Gestos

Es una aplicación informática que se utiliza para identificar seña de auxilio a partir de una imagen o un video. Se utiliza principalmente reconocer señas de auxilio, es tan apropiado como un escáner biométrico o de iris, pero es mucho más fácil de implementar. Este sistema de reconocimiento de señas con una base de datos donde se comparan las imágenes de las manos de las personas. Este sistema ejecuta un par de algoritmos para determinar si esta señar de auxilio de la persona en la cámara o el metraje coincide con cualquier imagen de la base de datos.

Convex Hull y defectos de convexidad



Tenemos el dibujo de una mano, aquel contorno rojo que rodea la mano es llamado *casco convexo o convex hull,* el cual da un efecto de banda elástica que está rodeando el objeto.

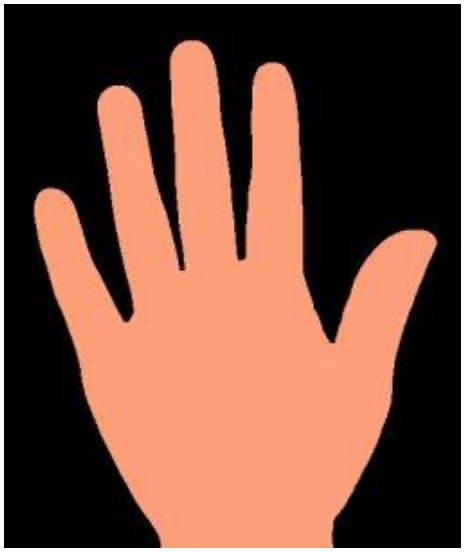
Si prestamos atención a este contorno de color rojo dibujado, vamos a darnos cuenta existen áreas en donde no está topando a la mano (en la figura están representadas con flechas bidireccionales negras), todas estas cavidades son consideradas *defectos de convexidad* que son desviaciones máximas locales, y en ellas nos apoyaremos para construir la aplicación de hoy, el conteo de dedos levantados, debido a que existirán defectos de convexidad.

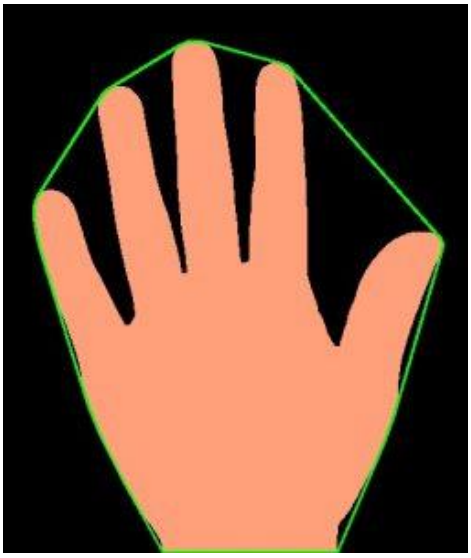
cv2.convexHull

Esta es la función que emplea OpenCV para construir un casco convexo/convex hull que rodee un objeto, por lo tanto, nos permitirá conseguir la línea roja que podemos ver en la figura, además nos ayudará a determinar los defectos de convexidad de un objeto.

Parámetros que usamos:

- *Points*, que es el contorno encontrado.
- *returnPoints*, por defecto es True que nos devolverá las coordenadas del casco convexo. Mientas que si es False devolverá los índices de los puntos del casco convexo, lo cual permitirá encontrar los defectos de convexidad.





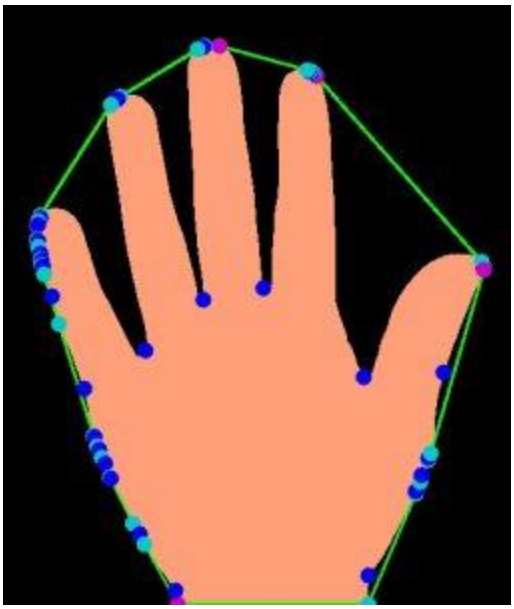
Cv2.convexityDefects

Una vez que se ha calculado el casco convexo y tenemos False en returnPoints, vamos a aplicar la función cv2.convexityDefects para obtener los defectos de convexidad.

Parámetros que Usamos:

- Contorno de entrada.
- Casco convexo obtenido a través de cv2.convexHull y cuyo parámetro returnPoints debió estar en False. Este debe contener los índices de los puntos que forman el casco convexo.

De esta función obtendremos un array en donde cada columna tendrá estos valores: **[punto inicial, punto final, punto más alejado, distancia aproximada al punto más alejado]**.



Aquí hemos etiquetado el punto inicial, final y más alejado del defecto de convexidad de entre el pulgar y el índice, mientras que en la figura está la totalidad de puntos que corresponden a los defectos de convexidad encontrados (como podemos ver son muchísimos defectos encontrados y algunos parecen no aportar mucha información).

Cabe destacar que también necesitaremos de otras funciones, aquellas que ya hemos visto en otros posts, por lo tanto, cuando las usemos te dejaré el link a cada una de ellas para que aclares algún dato en caso de ser necesario.

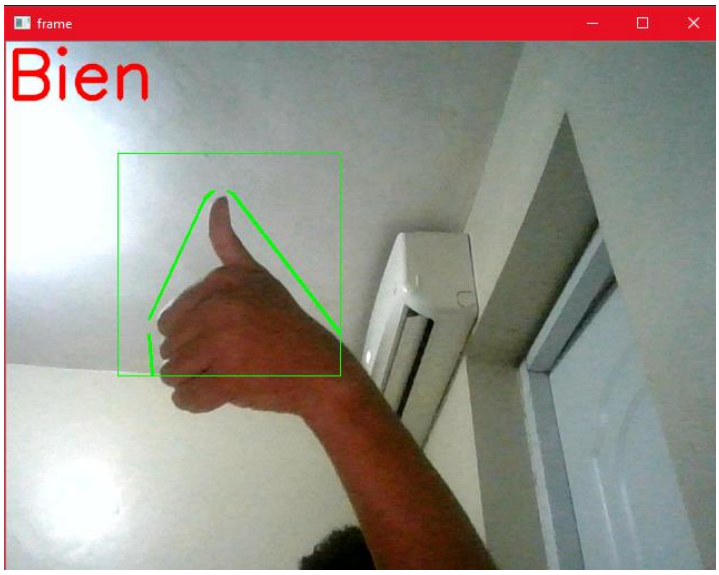
¿Como reconocer una seña de auxilio?

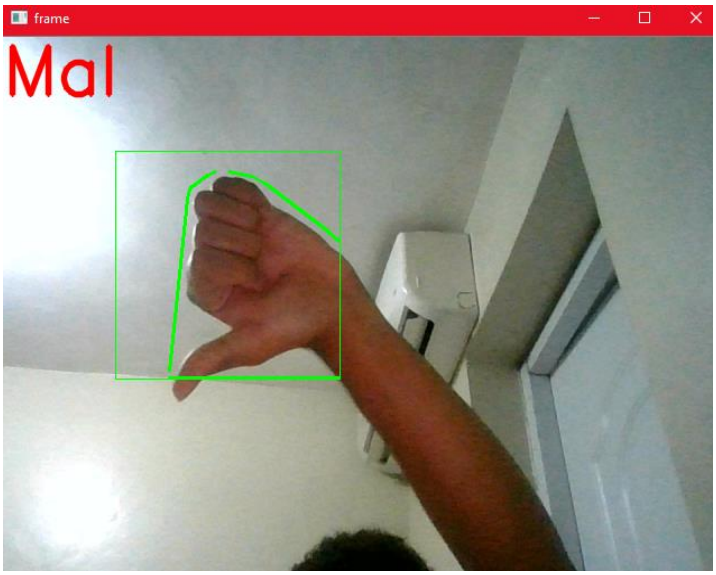
Para realizar esta aplicación tendremos que seguir algunos pasos tales como, obtención del fondo de la imagen, sustracción de imágenes (primer plano y fondo), encontrar el contorno de la mano en la imagen, extraer el casco convexo y los defectos de convexidad, diferenciar los casos en que estén levantados.

A la hora iniciar el sistema se coloca una de la mano mediante una seña se podría pedir ayuda será inmediatamente reconocido siempre y cuando este en el sistema.

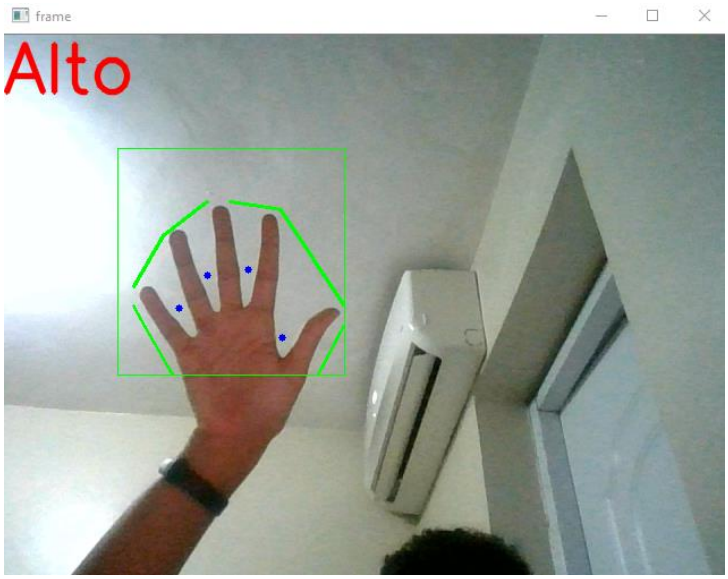
Este sistema cuenta con 6 señales de auxilio:

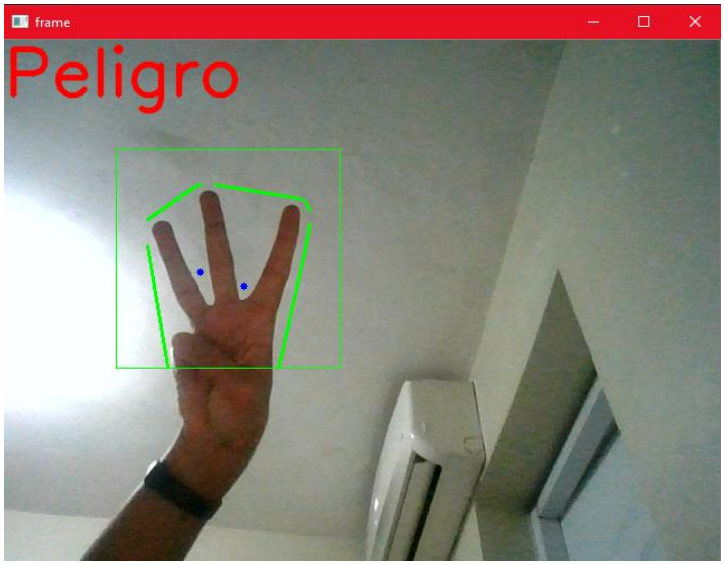
- ✓ Bien
- ✓ Mal
- ✓ Ayuda
- ✓ Peligro
- ✓ Alto
- ✓ Herido






frame
Herido


frame
Alto


frame
Peligro

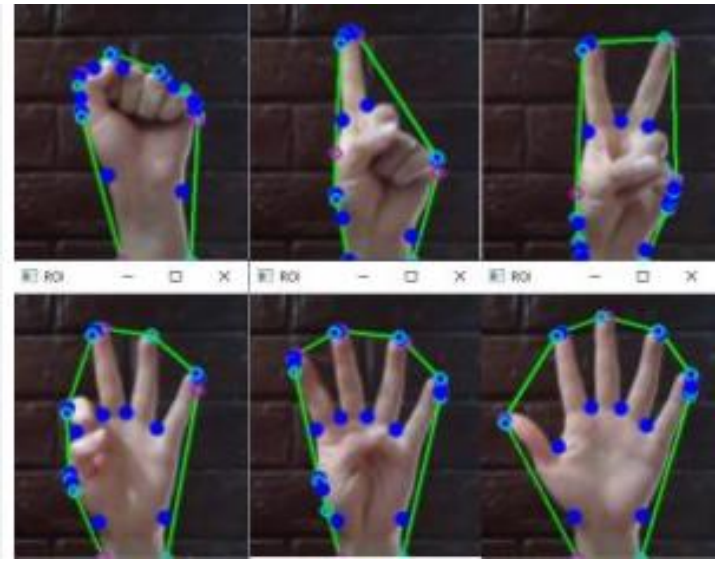
Importamos OpenCV, numpy e imutils, este último lo emplearemos para redimensionar cada fotograma.

Iniciamos con

cv2.VideoCapture para realizar un video streaming. He usado cv2.CAP_DSHOW en Windows para que la visualización del video aparezca completamente en la ventana de visualización. Podrías omitir esto en caso de no ser necesario.

Para poder tomar puntos que nos ayuden a solucionar esta aplicación hemos tomado 3 criterio ríos:

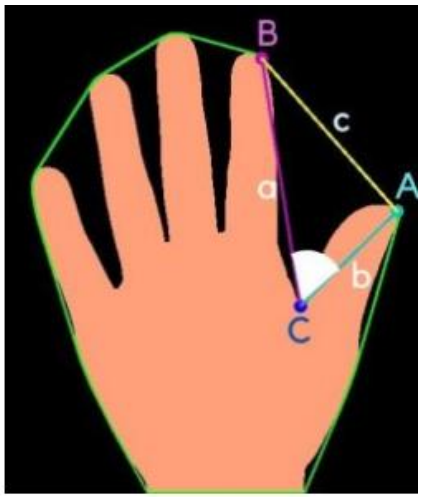
- Ángulo que forma el punto inicial al punto más alejado, y del punto más alejado al punto final.
- Distancia mínima entre el punto inicial y final.
- Distancia aproximada al punto más alejado.



Ángulo que forma el punto inicial al punto más alejado, y del punto más alejado al punto final.

Voy a empezar explicando el procedimiento que necesita un poco más de líneas de código. Si tomamos en cuenta los ángulos que pueden formar los dedos consecutivamente, tenemos que estos podrán formar hasta unos 90 grados aproximadamente (Por el ángulo que forma el pulgar y el índice).

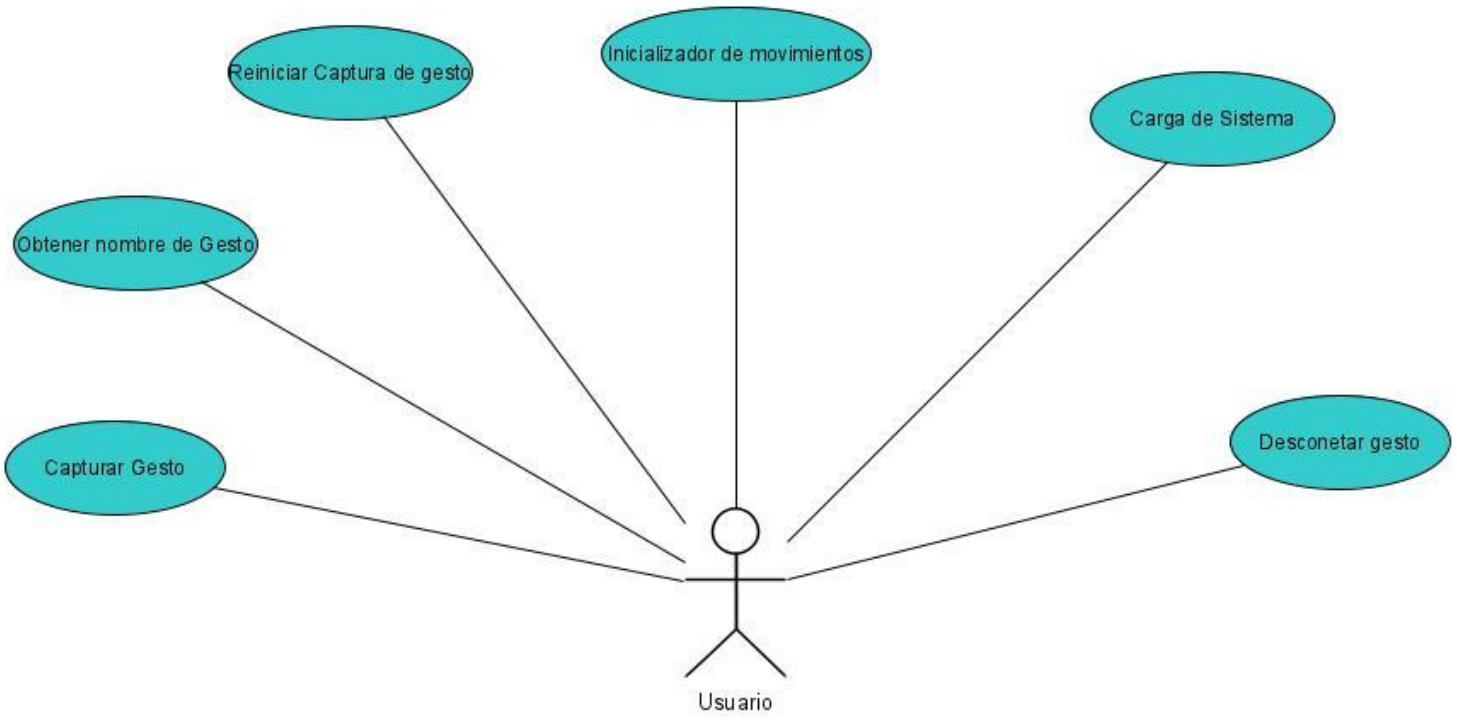
Para poder calcular los ángulos que forme cada par de dedos, podremos aplicar trigonometría, de tal modo que, dados los lados de un triángulo formados por los segmentos del punto inicial al punto más alejado, del punto más alejado al punto final y del punto final al punto inicial, tendríamos lo siguiente:



Para obtener el ángulo entonces tendríamos lo siguiente:

$$C = arcCos\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

Caso de uso


Reiniciar Captura de gesto
Inicializador de movimientos
Carga de Sistema
Obtener nombre de Gesto
Capturar Gesto
Desconetar gesto
Usuario

Distancia mínima entre el punto inicial – final y distancia aproximada al punto más alejado

Otros aspectos que probé es la distancia entre dos puntos, el punto inicial y final ya que si están demasiado cerca puede que no nos den buenos resultados al identificar los defectos convexos en esta aplicación, por ello deben cumplir con ser mayores a una distancia de 20.

Otra condición sería la distancia al punto más alejado, puesto que pueden aparecer identificaciones en los dedos que doblamos cuando extendemos otros, con esto nos aseguramos que haya una distancia mínima a cumplir.

Vamos a encontrar los defectos de convexidad. Los datos que nos devolverá defects serán escenciales ya que con ellos vamos a calcular el número de dedos levantados.

Si la distancia minY es mayor o igual a 110 (valor que he determinado después de pruebas, tu podrías modificarlo), quiere decir que un dedo ha sido levantado, por lo tanto, en la línea 121 se visualizará 1 en el la coordenada del punto más alto del contorno.

Diagrama de clases

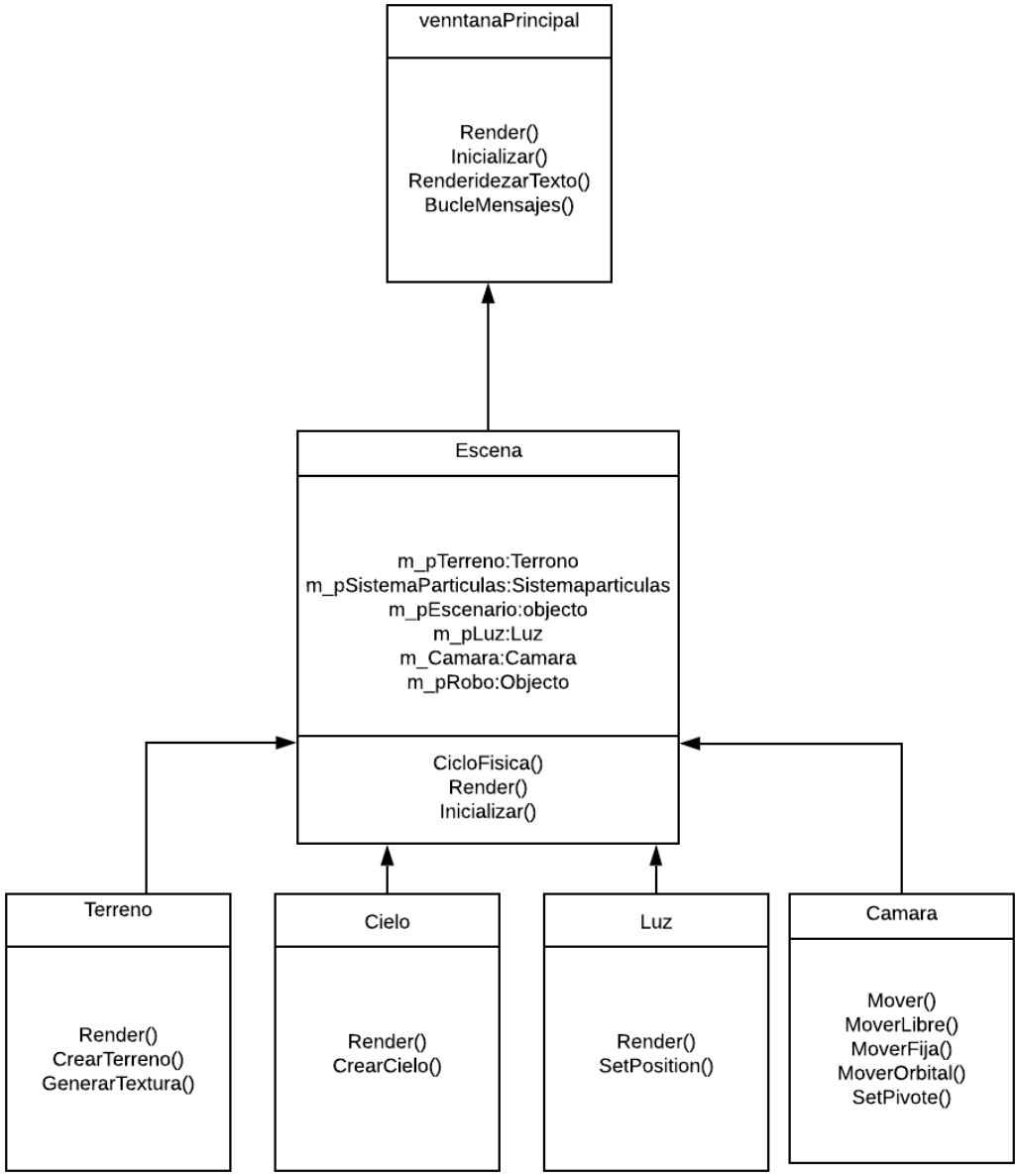


Diagrama de arquitectura enfocado en la interoperabilidad

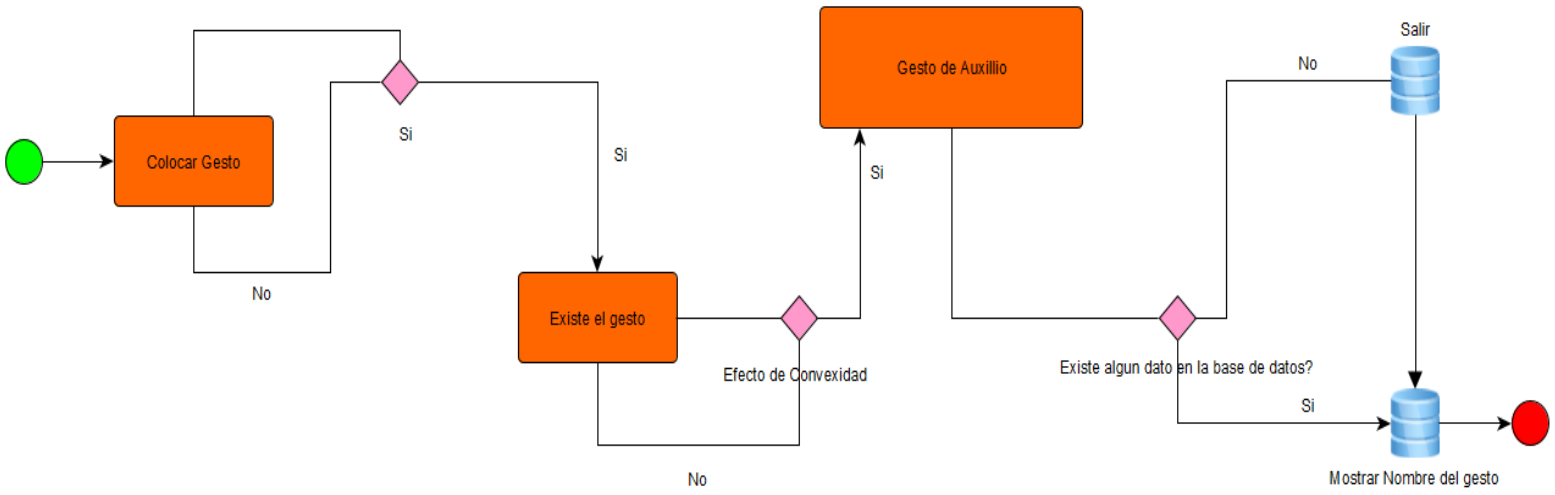
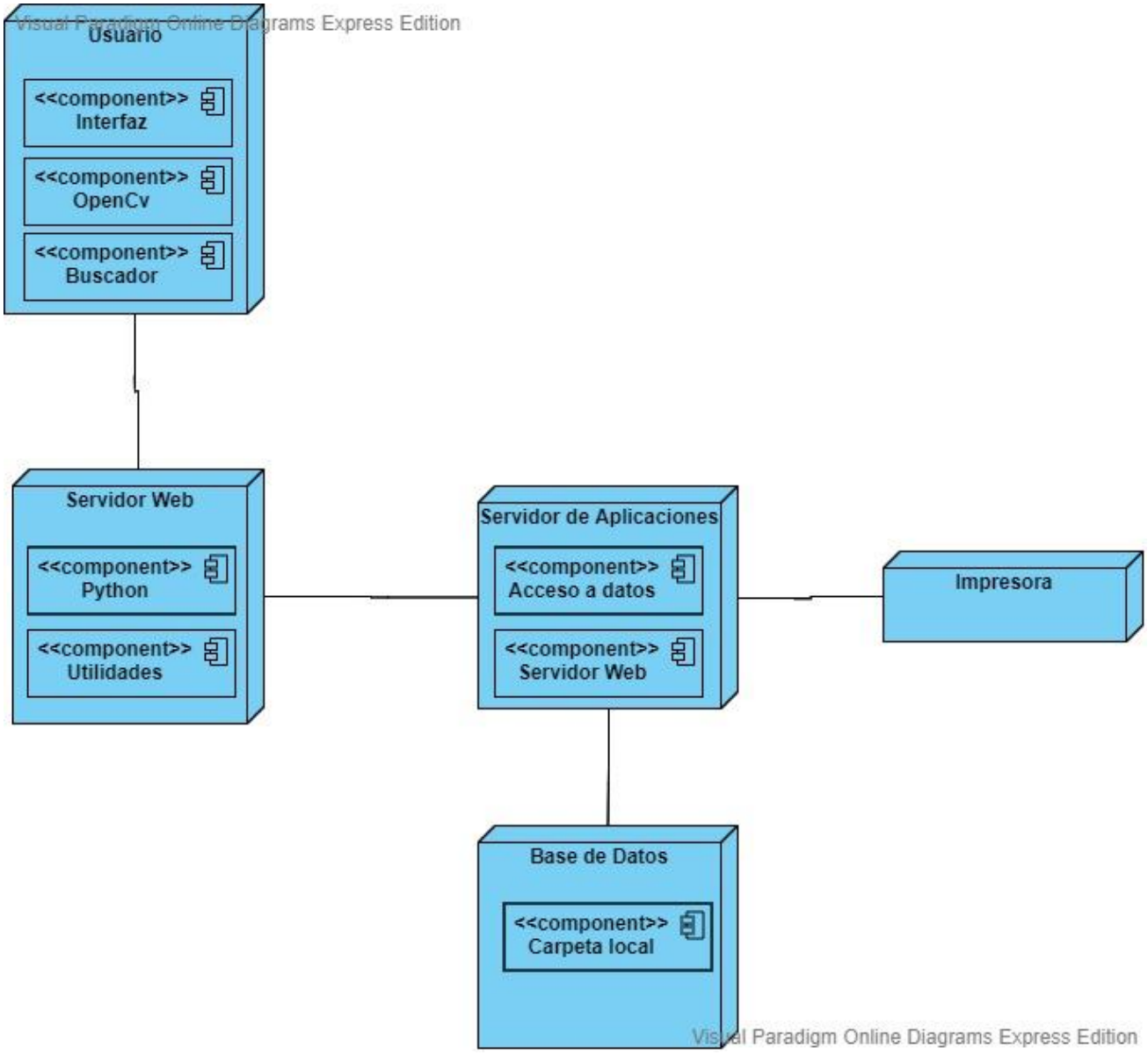


Diagrama de Componentes



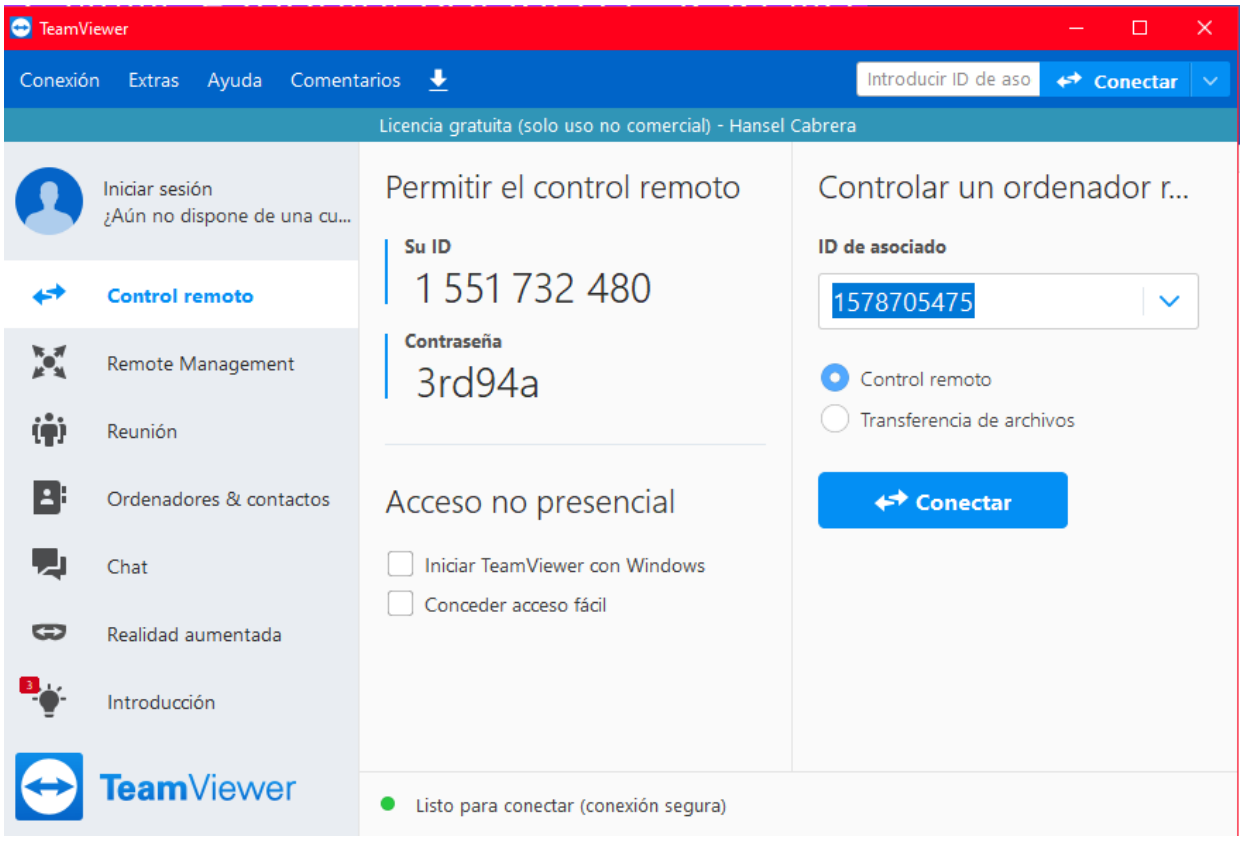
Presupuesto

ID	Entregable	Duracion(Dias)	Recurso
1	**Analisis/ Diseño del software**		
1.1	Diseño de pantallas	0.5	ANL
1.2	Diagramas de Base de Datos/Clases	3	DVS
2	**Desarrollo Software**		
2.1	Desarrollo del FrontEnd	8	DVJ
2.2	Desarrollo del BackEnd	7	DVS
3	**Capacitacion**		
3.1	Documentacion de Usuario	1	ANL,GDI
3.2	Entrenamiento de Usuario	0.5	ANL

Recursos Humanos		
DESCRIPCION	**TARIFA X HORA**	**CODIGO**
DEV JR	1,000	DVJ
DEV SENIOR	2,500	DVS
GRAPHIC DESIGNER	500	GDI
ANALISTA	700	ANL

COSTO DIRECTO	
MANO DE OBRA	**Costos**
DVS	600,000
DVJ	192,000
GDI	12,000
ANL	33,600
Total	837,600

Evidencia del uso de las herramientas de comunicación implementadas



Descripción: Como este proyecto lo realizamos solo dos personas decidimos que la mejor opción era utilizar “**Teamviewer**” la cual podemos ir trabajando simultáneamente, cada uno puede ir aportando ideas, ya que tenemos acceso audio visual.