La versión digital de esta tesis está protegida por la Ley de Derechos de Autor del Ecuador.

Los derechos de autor han sido entregados a la "ESCUELA POLITÉCNICA NACIONAL" bajo el libre consentimiento del (los) autor(es).

Al consultar esta tesis deberá acatar con las disposiciones de la Ley y las siguientes condiciones de uso:

- Cualquier uso que haga de estos documentos o imágenes deben ser sólo para efectos de investigación o estudio académico, y usted no puede ponerlos a disposición de otra persona.

- Usted deberá reconocer el derecho del autor a ser identificado y citado como el autor de esta tesis.

- No se podrá obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.

El Libre Acceso a la información, promueve el reconocimiento de la originalidad de las ideas de los demás, respetando las normas de presentación y de citación de autores con el fin de no incurrir en actos ilegítimos de copiar y hacer pasar como propias las creaciones de terceras personas.

Respeto hacia sí mismo y hacia los demás.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA PROTOTIPO DE CONTROL DE ACCESO AL LABORATORIO DE COMUNICACIONES UNIFICADAS DE LA FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA (FIEE) DE LA EPN EMPLEANDO RECONOCIMIENTO FACIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

BYRON GONZALO ARIAS MELENDRES
byron.arias@epn.edu.ec

DIRECTOR: Ing. ANDRÉS FERNANDO REYES CASTRO, MSc.
andres.reyes@epn.edu.ec

CODIRECTOR: Ing. XAVIER ALEXANDER CALDERÓN HINOJOSA, MSc.
xavier.calderon@epn.edu.ec

Quito, Enero 2020

AVAL

Certificamos que el presente trabajo fue desarrollado por Byron Gonzalo Arias Melendres, bajo nuestra supervisión.

Ing. ANDRÉS REYES, MSc
DIRECTOR DEL TRABAJO DE TITULACIÓN

Ing. XAVIER CALDERÓN, MSc.
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUDITORIA

Yo Byron Gonzalo Arias Melendres, declaro bajo juramento que el trabajo aquí descrito es de autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

BYRON GONZALO ARIAS MELENDRES

DEDICATORIA

Dedico este Trabajo a mi familia Gonzalo, Flor, Miriam, Orlando, Marcela y Monserrat, han sido un gran apoyo durante toda mi carrera estudiantil, han sido el motivo principal para continuar por este camino, han sido siempre mi ejemplo e inspiración y espero retribuir una parte de todo el esfuerzo, paciencia y cariño que han tenido hacia mí.

Se lo dedico a ustedes que han sido mis padres, madres, hermanos, hermanas y amigos durante toda mi vida.

AGRADECIMIENTO

Agradezco a mi familia, por siempre en todos los buenos y malos momentos, por siempre apoyarme a pesar de las circunstancias, por siempre confiar en mí capacidad y creer en mí, por los consejos y por toda la experiencia que me han trasmitido a lo largo de estos años, todo esto me ha ayudado a lograr mis objetivos.

Agradezco a mis amigos, que ha sido parte de este proceso, con los que he compartido varias experiencias ya que también saben las diferentes dificultades que se presentan durante nuestra carrera estudiantil, gracias, David, Alexander, Alejandra y Andrés, amigos de toda la vida.

Agradezco al Ing. Andrés Reyes por ayudarme en el desarrollo de este trabajo, su guía y recomendaciones, al igual que al Ing. Xavier Calderón, que no solo fueron mis directores de titulación, sino mis profesores durante la carrera.

Finalmente, agradezco a la Escuela Politécnica Nacional y todos los profesores que contribuyeron para que pueda formarme como un buen profesional, y sobre todo una buena persona.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

RESUMEN

El presente Trabajo de Titulación tiene como objetivo el desarrollo de un sistema prototipo de control de acceso, utilizando reconocimiento facial y orientado a mejorar la seguridad del Laboratorio de Comunicaciones Unificadas de la Escuela Politécnica Nacional. El proceso de reconocimiento facial se logra con el empleo de la librería OpenCV, el lenguaje de programación Python y otros módulos asociados a estos. Además, se emplea una PC y una Raspberry Pi que se encargan de ejecutar los diferentes programas que ayudan con el proceso de control de acceso.

El prototipo dispone de una aplicación que permite la administración de los usuarios que tienen autorización de ingresar al laboratorio, esta aplicación se encarga de permitir el registro de nuevos usuarios, eliminación de usuarios registrados y también la visualización de la bitácora de accesos al laboratorio a través de reportes de acceso.

El Trabajo de Titulación está compuesto por cuatro capítulos, los mismo que describen los fundamentos teóricos necesarios para la realización del prototipo, así como su diseño, codificación, implementación y pruebas.

En el Capítulo 1, se realiza un breve estudio de los diferentes conceptos necesarios para el desarrollo del sistema prototipo, como: lenguaje Python, OpenCV, sistemas biométricos, reconocimiento facial, Raspberry Pi y la metodología de desarrollo de prototipos.

En el Capítulo 2, se detalla el proceso de diseño del prototipo y la aplicación, definiendo los subsistemas y módulos que lo componen. Además, se describe el proceso de implementación de cada uno de los módulos, su codificación, despliegue y ejecución, así como también la conexión física da cada dispositivo utilizado.

En el Capítulo 3, se elabora un protocolo de pruebas, mismo que es empleado para comprobar el funcionamiento del prototipo. Finalmente, se realiza el proceso de tabulación de resultados y análisis de estos.

En el Capítulo 4, se determinan las conclusiones y recomendaciones.

PALABRAS CLAVE: Prototipo, Reconocimiento Facial, OpenCV, Python, Control de Acceso.

ABSTRACT

The objective of this Degree Work is to develop a prototype access control system, using facial recognition. It is aimed at improving the security of the Unified Communications Laboratory of the National Polytechnic School. The facial recognition process is achieved with the use of the OpenCV library, the Python programming language and other modules associated with them. In addition, a PC and a Raspberry Pi are used, which are responsible for executing the different programs that help with the access control process.

The prototype has an application that allows the administration of users who are authorized to enter the laboratory, this application is responsible for allowing the registration of new users, elimination of registered users and also the visualization of the access log to the laboratory through of access reports.

The Degree Work consists of four chapters, which describe the theoretical foundations necessary for the realization of the prototype, as well as its design, coding, implementation and testing.

In Chapter 1, a brief study of the different concepts necessary for the development of the prototype system is carried out, such as: Python language, OpenCV, biometric systems, facial recognition, Raspberry Pi and the prototype development methodology.

In Chapter 2, the prototype design and application process are detailed, defining the subsystems and modules that compose it. In addition, it describes the process of implementation of each of the modules, their coding, deployment and execution, as well as the physical connection of each device used.

In Chapter 3, a test protocol is developed, which is used to check the operation of the prototype. Finally, the process of tabulation of results and analysis of these is carried out.

In Chapter 4, the conclusions and recommendations are determined.

KEYWORDS: Prototype, Facial Recognition, OpenCV, Python, Access Control.

1. INTRODUCCIÓN

Actualmente, el reconocimiento facial ha tomado mucha relevancia dentro del campo de la seguridad, a pesar de ser un tema que se ha venido desarrollando hace varios años atrás. Los sistemas biométricos, así como los de control de acceso en general implementan este método de seguridad dentro de sus dispositivos y aplicaciones, gracias a que los avances tecnológicos han hecho posible implementar este método de una forma menos compleja.

Los dispositivos actuales, como celulares inteligentes, ordenadores y dispositivos biométricos, son capaces de soportar este tipo de algoritmos y procesos necesarios para la detección y reconocimiento de rostros, que, a pesar de presentar ciertas limitaciones, cumplen con el objetivo principal de este método de seguridad.

El presente Trabajo de Titulación se enfoca en la implementación de un sistema prototipo de control de acceso, empleando reconocimiento facial como un mecanismo de seguridad. A partir de la utilización de dispositivos que están al alcance de la mayoría de las personas, se pretende demostrar que es posible brindar una solución de control y seguridad de acceso para un determinado sitio, específicamente en este caso al Laboratorio de Comunicaciones Unificadas de la Escuela Politécnica Nacional.

1.1. OBJETIVOS

El objetivo general de este Trabajo de Titulación es Desarrollar un sistema prototipo de control de acceso al Laboratorio de Comunicaciones Unificadas de la Facultad de Ingeniería Eléctrica y Electrónica (FIEE) de la Escuela Politécnica Nacional (EPN) empleando reconocimiento facial.

Los objetivos específicos de este Trabajo de titulación son:

- Analizar los conceptos necesarios para el desarrollo del proyecto.
- Diseñar los módulos del software del sistema prototipo.
- Implementar el sistema prototipo de seguridad para el control de acceso en base al diseño realizado.
- Analizar los resultados obtenidos de acuerdo con las pruebas de funcionalidad.

1.2. ALCANCE

El sistema prototipo constará del módulo de control de acceso (reconocimiento de rostros), el sistema de interconexión a través del cual se conectará los diferentes dispositivos del sistema y una PC que contendrá la aplicación encargada de controlar el sistema y alojará el repositorio de imágenes.

El sistema para controlar el acceso será definido como un modelo Cliente-Servidor, el mismo que se especifica en la Figura 1.1, donde la PC actuará de servidor y contendrá el software desarrollado, además de repositorios de rostros y archivo de registro de acceso. Se hará uso de una Raspberry Pi, la misma que contendrá uno de los módulos de la aplicación encargado de bridar o no acceso al laboratorio.

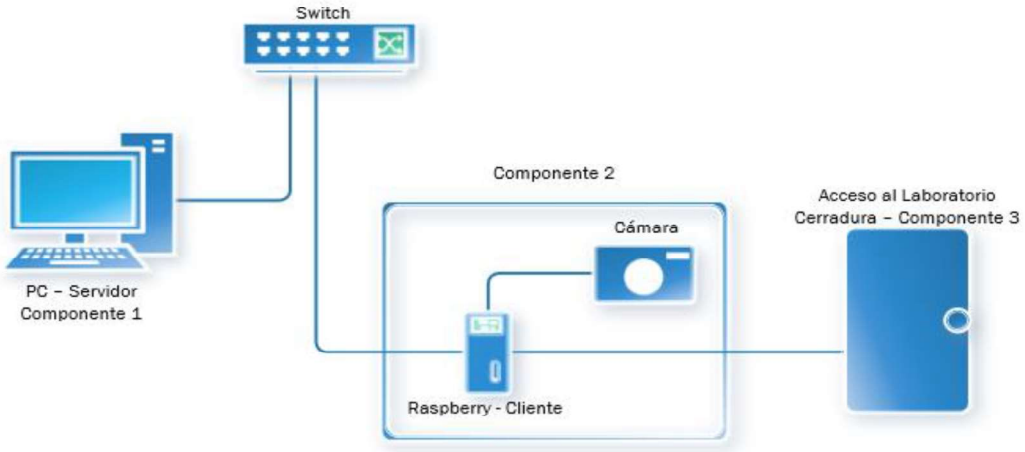


Figura 1.1. Diagrama del Sistema de Control de Acceso – Componentes de Hardware

Principalmente el sistema prototipo constará de tres componentes que son:

- **Hardware:** contiene todos los dispositivos empleados para el funcionamiento e implementación del sistema, principalmente una PC que actuará como servidor y contendrá la aplicación encargada de la administración de rostros, aprendizaje de rostros y registro de acceso. Además de la cerradura.
- **Software:** es la aplicación encargada de controlar el sistema, está compuesta por diferentes módulos detallados en la Tabla 1.1. Esta aplicación permitirá la comunicación entre el cliente y el servidor para obtener los datos necesarios para el correcto funcionamiento del sistema, así como la interacción con el repositorio de rostros.

- **Conectividad:** es el dispositivo (switch) encargado de comunicar tanto al cliente como al servidor, componentes del sistema de control de acceso.

Para el desarrollo del software que controlará el sistema prototipo se empleará el lenguaje de programación Python. La Tabla 1.1 detalla los módulos propuestos para el software del sistema prototipo y las funciones básicas que tendrá cada uno.

Tabla 1.1. Módulos del software del sistema prototipo de control de acceso.

Módulos	Funciones	Dispositivo
WebCam	▪ Reconocimiento-Video	Raspberry-Cliente
Buffer	▪ Almacenamiento-Rostros	PC-Servidor
Rostros	▪ Codificación-Rostros	PC-Servidor
Registro	▪ Reportes de acceso	PC-Servidor
Acceso	▪ Control de dispositivo de ingreso	Raspberry-Cliente

El PC-servidor contendrá los módulos de: Buffer, que es el encargado de almacenar nuevos rostros dentro del repositorio de imágenes; Rostros, que se encarga de la codificación de los rostros contenidos dentro del repositorio para la posterior comparación y el módulo de Registro que permitirá mostrar reportes del personal que haya ingresado al laboratorio.

La Raspberry-cliente contendrá dos módulos, que son: WebCam, que es el encargado de realizar la detección y reconocimiento de rostros y el de Acceso que se encargará de accionar o no la cerradura.

Es importante mencionar que para la administración de rostros (Buffer y Rostros) se diseñará y codificará una interfaz gráfica para facilitar la utilización del software. Además, se seleccionará una metodología de desarrollo de software adecuada para la codificación e implementación de la aplicación del sistema prototipo.

El sistema prototipo permitirá capturar video en tiempo real para de esta forma detectar el rostro de la persona que este frente a la cámara, la Raspberry-cliente será la encargada de procesar y autorizar o no el acceso al laboratorio, además de registrar a la persona que ingrese y enviar esta información al PC-servidor para actualizar el registro de personas autorizadas que hayan ingresado. Dentro del software se

desarrollará un módulo que permita el ingreso de nuevos rostros de personas autorizadas.

El dispositivo que controlará la Raspberry-cliente será una cerradura electrónica, la misma que se accionará dependiendo de la respuesta que brinde el módulo Acceso del software de control del sistema prototipo.

En la fase final, se propone realizar pruebas de funcionamiento del prototipo. Para esto se establece un protocolo de pruebas en el que se definen los parámetros a considerar como son: nivel de iluminación, distancia del rostro con referencia al lente de la cámara y casos excepcionales. La Tabla 1.2 describe más en detalle cada uno de los parámetros a tomar en cuenta.

Finalmente, se realizarán las pruebas de funcionalidad del prototipo mostrado en la Figura 1.1. En esta etapa se comprobará conectividad entre cliente y servidor, también se verificará el correcto funcionamiento de cada componente y que se otorgue o no el acceso al laboratorio.

Se procederá a realizar también un análisis de los resultados obtenidos.

Tabla 1.2. Descripción de parámetros de pruebas.

Parámetro	Descripción	Niveles
Iluminación	Variar el nivel de iluminación durante el proceso de reconocimiento facial.	Alta, media y baja.
Distancia	Para la detección en tiempo real, variar la distancia entre el rostro y el lente de la cámara.	Entre: 0 - 1 [m]
Excepciones	Probar el porcentaje acierto en el reconocimiento en casos especiales como: uso de gafas, anteojos, gorras y la presencia o no de vello facial.	Ninguno

1.3. MARCO TEÓRICO

1.3.1. Lenguaje de programación: Python

Python es un lenguaje de programación interpretado de alto nivel, dispone de eficientes estructuras de datos y de un enfoque simple para la programación orientada a objetos. Es ideal para el desarrollo de scripts y aplicaciones de una manera sencilla y rápida en varias plataformas. Además, es compatible con varios paradigmas de programación como son: Orientado a Objetos, Imperativa o de Procedimientos y Funcional [1].

Python dispone de una librería estándar muy completa, su interprete presenta la característica de ser compatible con varios sistemas operativos (Windows, Linux), además, se puede encontrar tanto el intérprete como la librería en la página web oficial de Python (www.python.org).

Es importante mencionar que este lenguaje es software libre, lo que significa que está al alcance de todas las personas, ya que no es necesario el pago de una licencia para el uso de todas sus características.

1.3.1.1. Historia

Originalmente la idea de crear este lenguaje de programación fue concebida a finales de los años 1980, por Guido Van Rossum en el CWI (Centrum Wiskunde & Informatica) un centro de investigación holandés. Su implementación inició a finales del año 1989. La idea principal era que este lenguaje sea fácil de usar y aprender, aun manteniendo su robustez. Van Rossum sigue tomando las decisiones respecto a las variaciones y nuevas versiones de este leguaje [1].

Existen varias versiones de Python, estas son: Python 2.0 lanzada en el año 2000 y Python 3.0 lanzada en el año 2008. La última versión de Python y la más estable es la 3.7, lanzada el 28 de junio de 2018.

1.3.1.2. Sintaxis

La sintaxis de Python es muy sencilla, ya que no es necesario el uso de símbolos para la delimitación de métodos y funciones. Al contrario, la delimitación se la hace a partir de las llamadas identaciones, que no es nada más que el empleo de espacios o

tabulaciones. La Figura 1.2 muestra un ejemplo de identaciones en un fragmento de código.

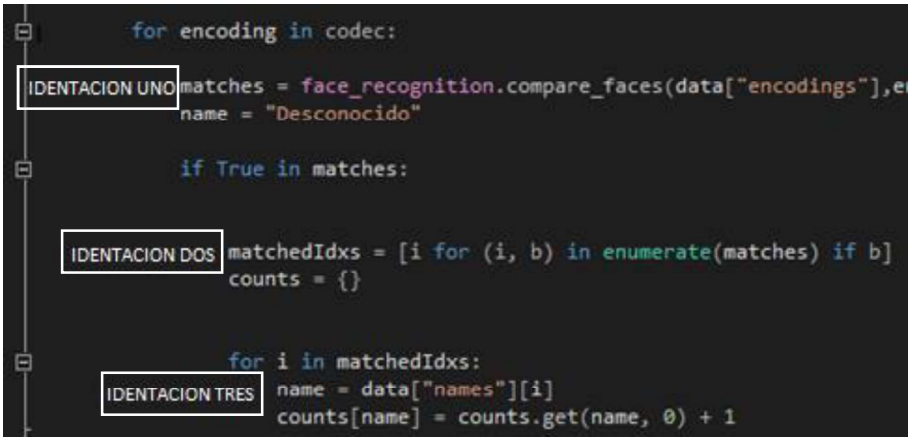


Figura 1.2. Ejemplo de Identación - Python

Además, Python está orientado a facilitar la programación, ya que este lenguaje fue desarrollado para que sea escrito de la forma en la que normalmente una persona se comunica o habla (idioma inglés) [1].

1.3.1.3. Librerías, paquetes y módulos

El lenguaje Python está compuesto por una infinidad de librerías y módulos, que permiten realizar diferentes procesos de una forma más sencilla. Es importante mencionar que debido a que este lenguaje es software libre, otras personas pueden proponer y aportar sus propios módulos e incluso librerías, con la condición de cumplir con lo que se rige dentro de la filosofía del lenguaje.

Inicialmente, al instalar Python este viene con una librería estándar, compuesta por varios módulos que pueden ser utilizados de forma directa para desarrollar scripts o programas. Dentro de los módulos que más se emplean de la librería estándar podemos encontrar los siguientes [1]:

- **os:** este módulo permite utilizar diferentes funciones, dependiendo del sistema operativo. Algunas de las funcione de este módulo son: open() – permite leer o edita run archivo, os.path – permite manipular rutas hacia archivos o directorios, fileinput – permite leer todas las líneas de un archivo, entre otras funciones que permiten verificar el estado del sistema operativo dentro del cual se esté ejecutando Python.

- **sys:** permite acceder a variables contenidas por el intérprete, y a ciertas funciones que interactúan con este.
- **random:** contiene generadores pseudoaleatorios de números.
- **math:** este módulo provee acceso a diferentes funciones matemáticas, no es capaz de trabajar con números complejos, para esto es necesario el uso de otro módulo (cmath), contiene funciones como: ceil, floor, factorial, gcd, entre otras.
- **Tkinter:** más que un módulo, es considerado un paquete que contiene un kit de herramientas que se emplean para el desarrollo de Interfaces Graficas de Usuario (GUI), viene instalado por defecto en todas las distribuciones de Python. Este paquete contiene varias clases y funciones que permiten el desarrollo de una interfaz de usuario simple y de fácil implementación.

La ventaja de la librería estándar es que ofrece varios módulos que incluyen diferentes funciones, las mismas que permiten realizar desde procesos tan simples como una suma de dos números, hasta un análisis numérico avanzado.

1.3.1.4. Desarrollo

La facilidad de Python al momento de programar hace que sea uno de los lenguajes más usados por las personas que quieren dar sus primeros pasos dentro de la programación, la versatilidad de este hace que sea muy útil en este sentido [1]. Python permite una programación estructurada debido a su cualidad de lenguaje interpretado[1], pero también permite emplear el paradigma orientado a objetos[2].

Desde sus inicios Python ha sido un lenguaje muy completo, y de alto nivel, ha sido empleado para el desarrollo de infinidad de aplicaciones, entre las más destacadas podemos mencionar DropBox, Instagram e incluso en varios aplicativos de la NASA.

Uno de los factores que es importante mencionar, son las diferentes versiones existentes de Python, este factor se lo puede tomar como una desventaja debido a que las versiones no son completamente compatibles, pero también como una ventaja debido a las nuevas funcionalidades que se han venido incorporando en cada versión. Entre las versiones más importantes y conocidas se tienen las que se observa en la Tabla 1.3.

[1] Es el lenguaje en el cual su código no necesita ser compilado, lo que significa que el computador ejecuta línea por línea las secciones de código sin la necesidad de la traducción previa de todas las secciones de este [48].
[2] Se basa en la creación de objetos, en lugar de variables, los mismos que poseen características propias y están interrelacionados entre sí para de esta forma resolver problemas [45].

Tabla 1.3. Versiones de Python

VERSION	AÑO DE LANZAMIENTO
1.0	1994
2.0	2000
2.7.X	2010 – 2019
3.0.X	2008 – Actualmente en desarrollo

1.3.2. Librería OpenCV

La librería OpenCV, o más conocida como OpenSource Computer Vision Libary [2] está definida bajo licencia BSD (Berkeley Software Distribution), la cual permite la modificación y uso del código fuente de esta para diferentes proyectos comerciales y académicos. Esta librería está compuesta por cientos de algoritmos de visión computarizada, los mismos que han sido desarrollados en C/C++.

Es importante mencionar la versatilidad que tiene esta librería ya que puede funcionar en sistemas operativos como Windows, Linux y Mac OS sin ningún problema, además presenta una característica que permite emplear esta librería conjuntamente con diferentes lenguajes de programación como son: C, C++, Java, Python, entre otros; debido a esta ventaja OpenCV ha sido muy utilizada.

Las principales funciones de OpenCV se enfocan en el procesamiento de imágenes en tiempo real, permitiendo capturar video e imágenes, procesarlos e importar un nuevo archivo de video o imagen con nuevas características. Los principales usos que se le ha dado a esta librería son enfocados en el área de seguridad, debido a módulos utilizados para la detección de objetos y personas.

OpenCV tiene una estructura modular, la cual presenta diferentes módulos que cumplen con diferentes funcionalidades. Entre los más importantes:

- **Módulo Core:** contiene o define la estructura básica de datos, arreglos mat y funciones básicas.
- **Módulo Features 2D:** permite usar detectores, comparadores y descriptores para imágenes.
- **Módulo Highgui:** contiene funciones como imshow, imread, imwrite, empleadas para visualizar, leer y guardar imágenes. Este módulo posee varias funciones que permiten el trabajo con imágenes en tiempo real.

- **Módulo Calib3d:** posee funciones para definir vistas, calibrar la cámara, elementos de captura 3D, entre otros.
- **Módulo Imgproc:** contiene funciones para el procesamiento de imágenes.

1.3.3. Reconocimiento Facial

El reconocimiento facial o de rostros (Face Recognition (FR)) es uno de los temas donde se ha desarrollado investigaciones importantes durante los últimos años, debido al gran interés por este tema dentro de diferentes áreas de la ingeniería. Alrededor del mundo, varios investigadores se han centrado en establecer diferentes métodos para lograr el reconocimiento de rostros, además de proponer varios algoritmos para mejorar los procesos tanto de detección, como de reconocimiento de rostros [3].

Dentro del desarrollo de sistemas biométricos es donde más trabajo e investigación se ha realizado sobre reconocimiento facial, ya que en la actualidad son empleados en la investigación e implementación del concepto de ciudades inteligentes. Pero, no solo dentro de esta área, también en la de seguridad en general, ya que se han planteado varios proyectos para la identificación de personas perdidas a través de capturas de video en tiempo real [4], además de otras áreas como se especifica en la Tabla 1.4.

Tabla 1.4. Aplicaciones y áreas de desarrollo de sistemas de reconocimiento facial [3].

ÁREAS	APLICACIONES
Entretenimiento	▪ Realidad Virtual ▪ Videojuegos ▪ Redes sociales
Tarjetas Inteligentes	▪ Licencias de conducir ▪ Identificaciones ▪ Pasaportes
Seguridad	▪ Desbloqueo de dispositivos ▪ Controles de acceso ▪ Identificación de personas ▪ Bases de datos ▪ Video-vigilancia avanzada

El proceso de reconocimiento facial se divide en dos partes esenciales que son:

- la detección de rostros, y
- la identificación o reconocimiento como tal.

La detección de un rostro se refiere a la forma en que el algoritmo emplea los diferentes métodos o mecanismos para determinar que la imagen que se está procesando, contiene o no un rostro humano. Por otro lado, el proceso de identificación o reconocimiento depende de la base de datos de rostros que se tenga almacenada, el algoritmo utiliza dicha base de datos para comparar el rostro detectado y así dar una respuesta. Para estos procesos se han desarrollado varias técnicas que permiten realizar cada uno de estos, queda a elección del programador el emplear la técnica que mejor resultado le ofrezca.

Como se había mencionado anteriormente, el proceso consta de dos partes principales, sin embargo, se definen un par de módulos y elementos necesarios para completar el proceso, como se muestra en la Figura 1.3.

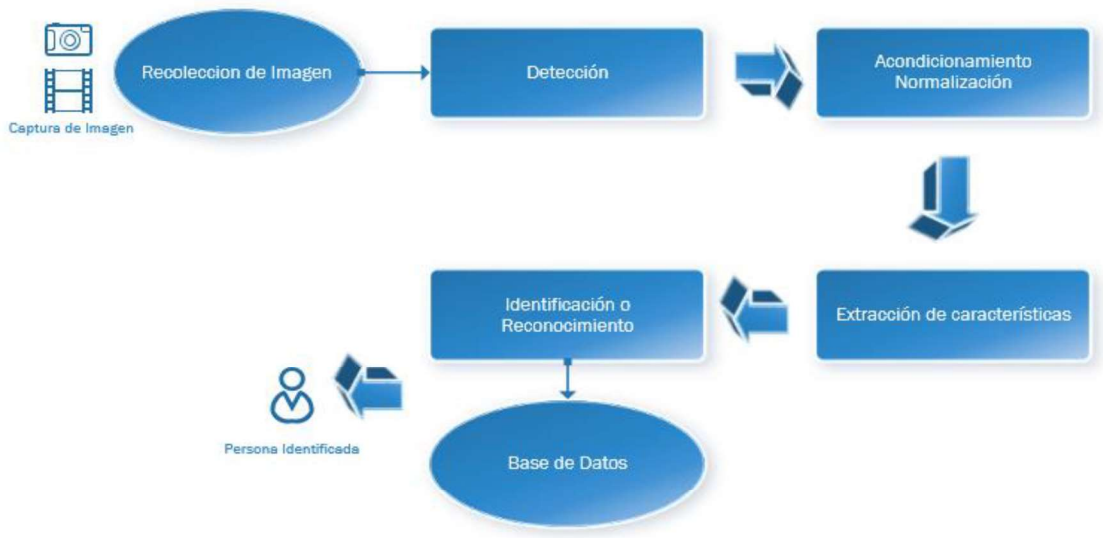


Figura 1.3. Proceso de reconocimiento facial [5]

1.3.3.1. Técnicas y algoritmos de reconocimiento facial

a. Técnica PCA (Principal Component Analysis)

El algoritmo PCA de reducción dimensional, se enfoca en encontrar vectores que representen la distribución de un grupo de imágenes, proyectando estas sobre un espacio en el que se puede observar las variaciones existentes entre imágenes de rostros conocidos, a estas proyecciones se las conoce como eigenfaces (Ver Figura 1.4). Esta técnica depende mucho de la iluminación con la que se hayan adquirido las imágenes.

Figura 1.4. Ejemplos de Eigenfaces [6]

b. LDA (Linear Discriminant Analysis)

El algoritmo LDA se encarga de crear un conjunto de vectores de características, a partir de la información extraída de imágenes del rostro de una misma persona (fisherfaces), donde es evidente las diferencias en las variaciones de los rostros, no se toman en cuenta los niveles de iluminación y otros factores. La idea principal de este algoritmo es aumentar la varianza entre muestras de distintas imágenes, y reducirla en muestras de las mismas imágenes. La Figura 1.5 presenta un ejemplo de fisherfaces.

Esta técnica depende mucho de los datos de entrada que se recopilen, ya que si las muestras se han tomado en condiciones no muy favorables (nivel de iluminación, resolución de las imágenes, etc) presentará errores al momento de ejecutarla.

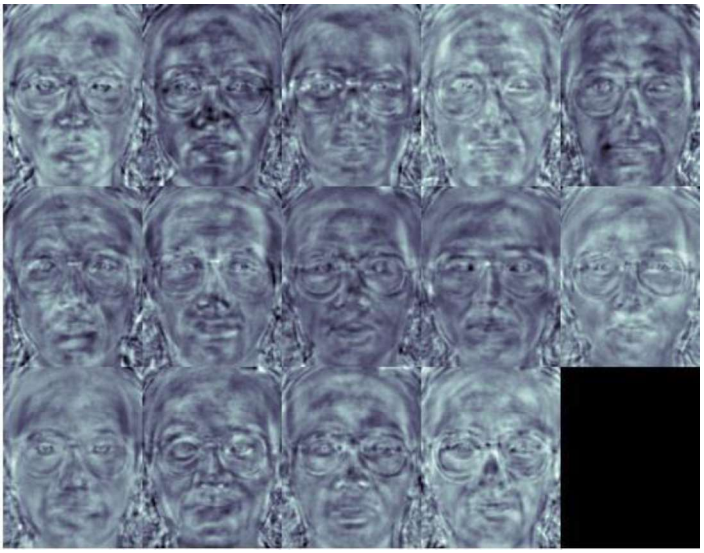


Figura 1.5. Ejemplos de Fisherfaces [7]

c. LBP (Local Binary Pattern)

La técnica LBP consiste en realizar un análisis de los rasgos de un rostro a partir de la información de los pixeles de la imagen que contiene dicho rostro. Para cada segmento se obtiene un píxel central y otros alrededor de este, se procede a realizar una comparación de pixeles para obtener un numero decimal, el mismo que se utiliza para formar un vector o histograma correspondiente a cada rostro.

Esta técnica proporciona una gran robustez con respecto a las variaciones de iluminación, además debido a la simple implementación computacional se lo puede emplear para analizar imágenes en tiempo real.

Gracias a su característica de alta eficiencia computacional, este algoritmo es muy empleado para proyectos en los que no se dispone de altos recursos para el procesamiento de imágenes. Un ejemplo de una imagen extraída utilizando LBP lo podemos observar en la Figura 1.6.

Figura 1.6. Ejemplo de LBP [8]

d. Algoritmo de Visión Artificial

La visión artificial tiene como finalidad obtener modelos matemáticos de los procesos de percepción visual para de esta forma simular estos procesos con la ayuda de un computador, a través de algoritmos o programas generados empleando diferentes técnicas de procesamiento de imágenes [9].

El proceso de visión artificial se lo puede dividir así:

- **Sensado:** se refiere a obtener la imagen.
- **Preprocesamiento:** se realiza técnicas para mejorar los detalles de la imagen.
- **Segmentación:** divide la imagen de acuerdo con la importancia en el procedimiento que se vaya a realizar.
- **Descripción:** se realiza la selección de características útiles para diferenciar entre objetos.
- **Reconocimiento:** se identifica los objetos.
- **Interpretación:** se establece el significado de los objetos identificados.

Se pueden mencionar diferentes áreas donde la visión artificial o visión por computadora es empleada:

- Industrias (automotriz, medica, financiera)
- Robótica
- Comunicación persona-maquina
- Seguridad, vigilancia, accesos biométricos
- Control de tráfico

Es importante mencionar que estos procesos de visión artificial serán empleados en este trabajo, para esto se hará uso de la librería OpenCV que contiene varias funciones basadas en visión por computadora.

e. Algoritmo de Viola-Jones

Este algoritmo fue desarrollado por Paul Viola y Michael Jones, se enfocaron principalmente en resolver el problema de la detección de rostros en tiempo real, empleando la menor cantidad de recursos computacionales. En “Robust real-time Object Detection” [10], presentaron un sistema capaz de detectar objetos, procesando imágenes de una manera muy rápida, logrando altas tasas de detección.
Es importante mencionar que este algoritmo solo se enfoca en la detección de rostros, no permita la identificación de estos.

Las características que hacen de este algoritmo uno de los mejores para la detección de rostros, son:

- Robustez, ofrece una tasa de detección muy alta, y un bajo índice de falsos positivos.
- Detección en tiempo real, permite procesar al menos dos frames por segundo.
- Permite solo detección, no se enfoca en el reconocimiento de rostros, gracias a esto su alta eficiencia, es importante recalcar que la fase previa al proceso de identificación de rostros es la detección.

Componentes del sistema

En la Tabla 1.5 se listan los componentes de este algoritmo.

Tabla 1.5. Componentes del Algoritmo de Viola-Jones

N°	COMPONENTE
1	Selección de Características Haar
2	Imagen Integral
3	Entrenamiento con AdaBoost
4	Clasificadores en cascada

Las características de Haar, ayudan al momento de definir si se trata o no de un rostro humano, ya que se definen ciertas propiedades comunes en todos los rostros de personas, estos pueden ser: ojos, nariz, boca. La imagen de entrada pasa por los llamados filtros de Haar (Ver Figura 1.7) para determinar si hay la posibilidad de encontrar o no un rostro humano en dicha imagen.

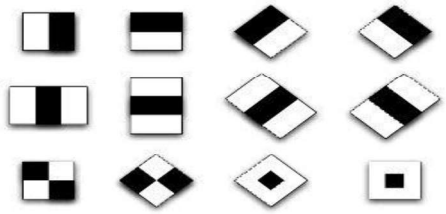


Figura 1.7. Ejemplos de filtros de Haar

Imagen Integral, permite la extracción de características de la imagen de entrada, se define como la imagen de la cual cada punto contiene el resultado de la suma de los valores de todos los puntos, situados por encima y a su izquierda en la imagen original [5], como se observa en el ejemplo de la Figura 1.8.

Figura 1.8. Ejemplo de imagen integral [11]

Entrenamiento con AdaBoost, este proceso se basa en un algoritmo de machine learning, este permite adaptarse a otros algoritmos para mejorar su respuesta. Es empleado dentro del sistema Viola-Jones ya que permite el aprendizaje de características a partir de filtros de Haar, para mejorar el proceso de detección de rostros cada que se emplea dicho algoritmo.

Clasificadores en cascada, realizan una combinación de clasificadores con un número mayor de características, agrupados en etapas de una forma ascendente en el número de clasificadores por etapa, para así obtener como resultado la imagen con la probabilidad más alta de contener un rostro. La Figura 1.9 muestra un ejemplo de clasificador en cascada con cuatro etapas, la idea es ir descartando en cada etapa los frames procesados que no contengan un rostro y el frame procesado que logre atravesar todas las etapas será el que tenga mayor probabilidad de contener un rostro.

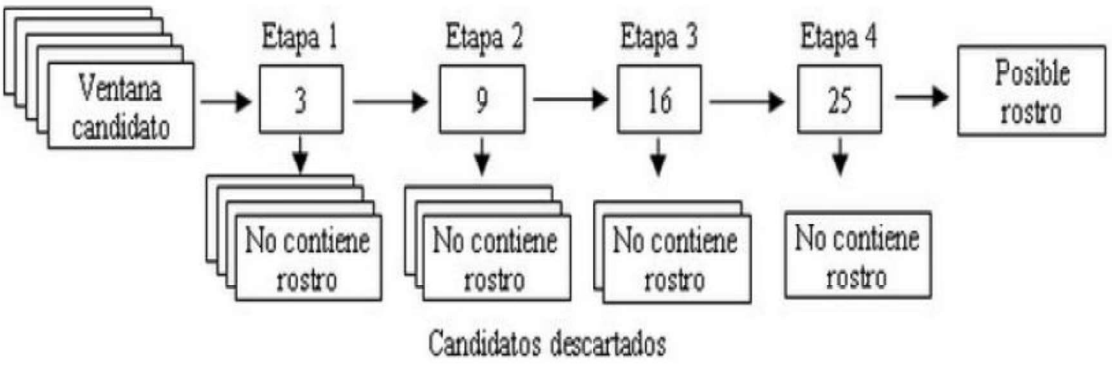


Figura 1.9. Ejemplo de clasificador en cascada de cuatro etapas [5]

Este procedimiento se encuentra implementado dentro de la librería OpenCV, además, está ya tiene incorporada por defecto varios archivos o plantillas empleadas para la detección de ciertos rasgos de un rostro.

1.3.4. Sistemas biométricos con Reconocimiento de rostros

La biometría se define como la ciencia que se encarga de reconocer a un individuo, basado en sus características fisiológicas. Los sistemas biométricos han sido empleados dentro de varios campos, específicamente en el de la seguridad. Históricamente los sistemas biométricos que emplean características fisiológicas son:

- De huella dactilar
- Escaneo de retina
- Escaneo de la geometría de la mano
- Reconocimiento Facial
- Reconocimiento de voz

Y una combinación de los antes mencionados.

Durante los últimos años, gracias a la inclusión del procesamiento por computador, estos sistemas han ido evolucionando, los procesos se han ido automatizando y de esta manera pueden ofrecer mejores prestaciones para estos sistemas.

1.3.4.1. Biometría con reconocimiento facial

El principal problema enfrentado por la biometría para el reconocimiento de rostros fue al momento de identificar a quien pertenecía dichos rostros, sin embargo, con el desarrollo de la visión por computadora, los diferentes algoritmos, las diferentes técnicas y herramientas que se han investigado durante los últimos años se ha podido solventar este problema. Actualmente, gracias a los diferentes procesos mencionados en la sección 1.3.3, los sistemas biométricos ofrecen una alta eficiencia al momento de reconocer rostros.

Existen varias aplicaciones en las que se puede emplear la biometría de rostros, estas son:

- **Vigilancia:** detección de intrusos a un establecimiento.
- **Videoconferencia:** localizar a un individuo durante una secuencia de video.
- **Detección de expresiones faciales:** empleado en aplicaciones médicas o también para seguridad en la conducción de un vehículo.
- **Control de acceso:** permitir el ingreso a un área determinada.

a. Funcionamiento

Un dispositivo biométrico debe ser capaz de medir, codificar, almacenar y transmitir o reconocer una característica única y propia de la persona. Además, debe brindar una alta confiabilidad en la respuesta a dicho proceso. La idea de emplear un sistema biométrico es para mejorar la identificación de personas a través de reconocer características que las hacen únicas y así evitar el uso de contraseñas, pines u otros mecanismos que pueden ser inseguros [12].

Para el proceso de identificación de una persona, se lo debe hacer de una forma digital, midiendo sus características y comparándolas con las almacenadas en un repositorio o base de datos antes generado, que sea correspondiente a dicha persona. Para esto se emplean ciertas técnicas computacionales de acuerdo con la característica física que emplee dicho sistema biométrico. Algunos de estos procedimientos o técnicas computacionales pueden ser:

- Reconocimiento de formas
- Inteligencia artificial
- Algoritmos de aprendizaje

Es importante mencionar, que los sistemas biométricos más utilizados son los de lectura de huella digital y el de escáner de la geometría de la mano, sin embargo, el desarrollo acelerado de las técnicas de reconocimiento facial ha hecho que estos sean más utilizados durante los últimos años [12].

1.3.5. Raspberry Pi

Es una placa de computadora de tamaño reducido y de bajo costo, fue desarrollado en el Reino Unido con la finalidad de incentivar la enseñanza de la informática en las escuelas, con la idea de facilitar el aprendizaje brindando una visión no tan formal de estos temas.

La Raspberry Pi está formada por una placa que soporta la conexión de varios componentes (teclado, ratón, webcam, etc) permitiendo el funcionamiento de esta como un computador pequeño. A pesar de su tamaño, tiene un alto poder de procesamiento, por esta razón que es muy utilizada dentro de la ingeniería para el desarrollo de proyectos.

El proyecto Raspberry Pi, fue desarrollad en la Universidad de Cambridge y Eben Upton en el año 2006, a partir del año 2009 se crea la fundación Raspberry Pi, para el año 2011 se fabricaron alrededor de 100 placas, pero a partir del año 2012 se empezó la comercialización de estas.

1.3.5.1. Componentes

A pesar de su reducido tamaño, la Raspberry Pi tiene varios componentes (Ver Figura 1.10) que son:

- **Procesador:** Broadcom BCM2835/2836/2837, incluyen CPU de tipo ARM de 700/900/1200 MHz. Permiten la ejecución de tareas empleando el mínimo consumo de energía, operando a 5V – 1A.
- **GPU:** es un procesador gráfico VideoCore IV.
- **Memoria:** la RAM es la memoria principal del sistema, se inició con una capacidad de 256 MB, aunque en la actualidad puede llegar hasta los 4GB.
- **Disco Duro:** o unidad de almacenamiento, puede tener la capacidad de 8 a 128 GB.
- **Entradas y Salidas:** contiene puertos USB 2.0 (Actualmente 3.0), HDMI, salida analógica de audio y puertos GPIO (General Purpose Input/Output).

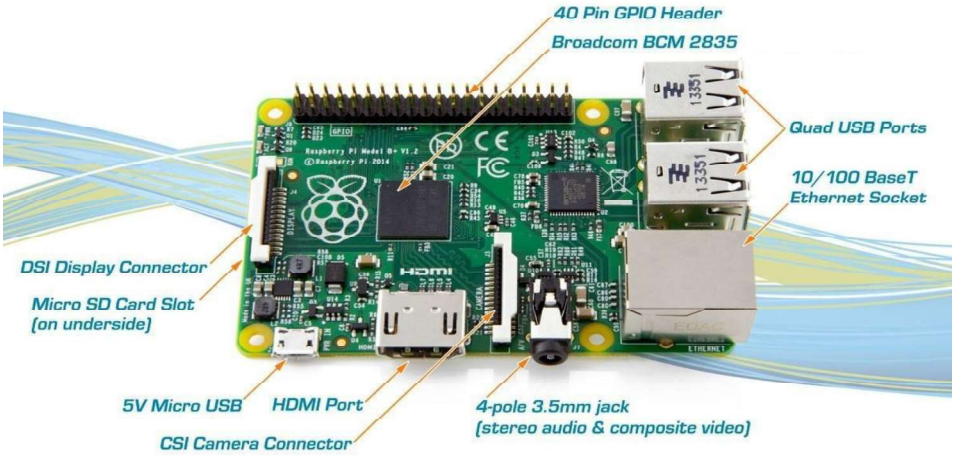


Figura 1.10. Elementos de la Raspberry Pi

Actualmente, se han lanzado varios modelos de Raspberry Pi, los mismos que cuentan con ciertas variantes, pero conservan su diseño base. En la Tabla 1.6 se listan los modelos principales que han sido desarrollados.

Tabla 1.6. Modelos de Raspberry Pi

MODELO	AÑO DE LANZAMIENTO
A	2012
A+	2014
B	2012
B+	2014
2B	2015
3B	2016
3B+	2018
4B	2019

1.3.5.2. Software

El software de la Raspberry Pi está basado en una distribución de Linux (Debian) y se lo conoce como Raspbian, este es un sistema operativo libre y gratuito, fue dado a conocer en el año 2012, al mismo tiempo de la comercialización de las primeras tarjetas Raspberry Pi. Raspbian es el software recomendado para estas minicomputadoras.

Existen otros sistemas operativos que pueden ser compatibles para la Raspberry, estos son:

- Kano OS
- Flint OS
- Windows 10 IoT Core
- Ubuntu Mate

1.3.5.3. Python y Raspberry Pi

El lenguaje de programación recomendado por los fundadores de la Raspberry es Python, debido a que es un lenguaje de alto nivel, interpretado y presenta una sintaxis muy entendible. Python permite el desarrollo de un sinnúmero de aplicaciones y al ser multiplataforma es compatible para el sistema operativo que maneja a la Raspberry, además que cuenta con varias librerías que permiten realizar varios proyectos y sin la necesidad de un alto poder de procesamiento.

Python es el lenguaje de programación que viene instalado por defecto en la distribución Raspbian.

1.3.6. Metodología de desarrollo de Prototipos

Las metodologías son una serie de actividades que se llevan a cabo para el desarrollo de sistemas o aplicaciones de software y su puesta en funcionamiento. Existen diferentes metodologías propuestas para el desarrollo de estos sistemas, cada una de ellas define que actividades realizar en cada fase del diseño. En primera instancia para el diseño de una aplicación se definen cinco fases principales, que son (Tabla 1.7):

Tabla 1.7. Fases del Diseño de Software

Fase	Definición
1	Análisis de Requerimientos
2	Diseño
3	Implementación
4	Pruebas
5	Documentación

Algunos autores definen más fases para el proceso de diseño, al ser un proceso que no está normalizado, queda a elección de la persona encargada de este proceso la selección o no de un mayor número fases dentro del proceso.

Dentro de la ingeniería de software se pueden definir principalmente dos tipos de metodologías de desarrollo: las Estructuradas y las Orientadas a Objetos. La principal diferencia reside en el formato de programación seleccionado (estructurada y orientada a objetos), debido a esto la clasificación de estas metodologías.

La metodología de prototipos puede formar parte de ambas, ya que esta se enfoca en brindar una primera versión del sistema final, independientemente del tipo de programación empleado y del paradigma que se seleccione para el desarrollo de la aplicación [13].

1.3.6.1. Metodología de Prototipos

Se la conoce también como modelo de prototipos o modelo de desarrollo evolutivo, debido a que se inicia definiendo los objetivos generales del software e identificando los requisitos más esenciales que debería cumplir en primera instancia. Este modelo es empleado para dar al usuario una versión rápida y preliminar del software o de una parte

de este, de esta manera se puede ir refinando los diferentes requerimientos o a su vez ir descubriendo nuevas necesidades que podría solventar el software [14].

Este modelo es muy útil para el desarrollo de aplicaciones que necesiten una solución rápida, ya que en primera instancia permite al usuario observar un posible resultado de visión del producto final, de esta forma facilita que este se dé cuenta de los posibles errores que el software puede llegar a tener y a su vez puede sugerir o solicitar cambios, de esta forma facilita la recopilación de los requerimientos para el desarrollador.

El diseño debe ser rápido, debido a que se enfoca en brindar una visión al usuario del producto final, esto conduce a la construcción de un prototipo que será evaluado por el usuario final, este debe brindar sus observaciones para que el desarrollador realice los cambios para la nueva versión del prototipo.

El modelo de prototipos presenta las siguientes etapas:

- Recolección de requisitos o Comunicación
- Modelado, diseño rápido
- Construcción del prototipo
- Desarrollo, evaluación y retroalimentación
- Refinamiento de requisitos
- Entrega del desarrollo final (Versión final)

Cada etapa, está relacionada con las correspondientes al desarrollo de software, sin embargo, para el modelo de prototipos tenemos una etapa extra para la refinación de los requisitos del software.

a. Ventajas

- Permite que el cliente se involucre mucho más, para conseguir el producto final.
- Es muy útil, ya que a partir del objetivo general que debe cumplir, el software se puede detallar de mejor manera los diferentes requisitos, que en muchos de los casos no es muy obvio para el cliente y el desarrollador ya que no se tiene idea de cuál será el resultado final.
- Ofrece al desarrollador obtener un mejor enfoque, para el desarrollo de los diferentes algoritmos, verificar o no su eficiencia dentro del software.

- Es aceptable reutilizar el código.
- Reduce costos, y mejora la posibilidad de obtener un resultado aceptable.
- Debido a que el usuario está en permanente contacto con el desarrollo del software, permite que el producto final satisfaga casi en su totalidad las necesidades del cliente.

b. Desventajas

- El usuario confunde el prototipo, con el producto ya terminado, debido a que ya es funcional.
- El usuario puede llegar a crearse varias expectativas de acuerdo con lo que va observando, y puede exigir que en base al prototipo entregado se construya el software final, sin tomar en cuenta otros factores como son la calidad y el mantenimiento a largo plazo de este.
- Debido al rápido desarrollo del software, se puede llegar a tomar decisiones que a posteriori pueden resultar poco convenientes.

c. Tipos de Modelo de Prototipos

Antes de definir los tipos de modelos, debemos definir los tipos de prototipos que existen, estos son (Ver Tabla 1.8):

Tabla 1.8. Tipos de prototipos [13]

PROTOTIPO	DESCRIPCIÓN
Desechable	▪ Enfocado al desarrollo de la interfaz. ▪ Es empleado para ayudar al usuario con dudas que tenga, sobre ciertos requisitos.
Evolutivo	▪ Es un modelo que ofrece una versión muy cercana al producto (software) final.

Se pueden definir varios tipos de modelos, entre los que se puede mencionar los siguientes, Tabla 1.9:

Tabla 1.9. Tipos de Modelo de Prototipos [14]

MODELO	DESCRIPCIÓN
Rápido	▪ Desarrollo rápido de nuevos diseños. ▪ Desecha el prototipo previamente desarrollado, después de realizar un nuevo diseño.
Reutilizable	▪ Empleado mayormente en el desarrollo de software. ▪ Conocido también como “Prototipado Evolutivo”. ▪ Se puede reutilizar partes del prototipo para el desarrollo del producto final.
Modular	▪ Conocido como Prototipado Incremental. ▪ Se va sumando elementos al prototipo, con cada avance durante el ciclo de desarrollo.
Horizontal	▪ Cubre varios aspectos funcionales, pero no operativos. ▪ Muy utilizado para evaluar el alcance del producto.
Vertical	▪ Se enfoca en funciones operativas. ▪ Empleado para evaluar el uso real del producto.
Baja-fidelidad	▪ Se establece un diseño en papel, para emular el funcionamiento real del producto.
Alta-fidelidad	▪ Se implementa el prototipo de una forma muy cercana al producto final.

d. Observaciones

Dentro de esta metodología o modelo, hay que tomar en cuenta ciertos factores o características que debe tener el sistema para poder emplear este modelo, esto son:

- El sistema debe permitir al desarrollador experimentar con el mismo.
- No debe ser de un costo elevado.
- Permitir un desarrollo rápido.
- No se debería requerir un grupo de desarrolladores muy grande.
- Debe contener a menos una interfaz gráfica.
- Versatilidad para emplear diferentes lenguajes de programación.

Además, este modelo puede llegar a ser muy efectivo, para esto la comunicación entre usuario y desarrollador debe ser muy estrecha ya que deben llegar a ciertas convenciones, como:

- Definir que el prototipo brindará una versión muy aproximada al producto final, pero que se entienda que no lo es.

- Las diferentes versiones de prototipos serán utilizadas para el refinamiento y/o definición de requisitos.
- El prototipo puede ser descartable.
- En base a las recomendaciones, y a la versión final del prototipo se pueda desarrollar un software final enfocado a la calidad y mantenimiento de este.

2. METODOLOGÍA

El presente capitulo se divide en dos secciones: Diseño e Implementación. En la primera sección se definirá los requerimientos necesarios para el desarrollo del prototipo, también se definirá las diferentes funciones y módulos del software y finalmente se detallará las características de los elementos de hardware necesarios para la implementación del prototipo. En la sección de implementación se documentará el proceso de codificación, definido en la fase previa, así como los procesos de instalación del prototipo.

Es importante mencionar que para el desarrollo de este proyecto se ha tomado como referencia la metodología de prototipos, específicamente el modelo evolutivo. De esta forma se han obtenido dos prototipos, el primer prototipo diseñado de forma rápida fue empleado para la refinación de requerimientos y la corrección de errores para la posterior implementación del segundo prototipo, que es el resultado final de este Trabajo de Titulación.

2.1. DISEÑO

En esta sección se realiza un breve análisis del estado del Laboratorio de Comunicaciones Unificadas y se define los requerimientos funcionales y no funcionales del prototipo, los módulos y funciones principales de cada uno de estos, así como la elección de los diferentes elementos de hardware que serán necesarios para la implementación del prototipo.

2.1.1. Análisis del estado del Laboratorio de Comunicaciones Unificadas

El análisis fue realizado a partir de una visita al laboratorio, para observar el estado de este. En la Figura 2.1 se detalla la distribución del Laboratorio de Comunicaciones Unificadas.

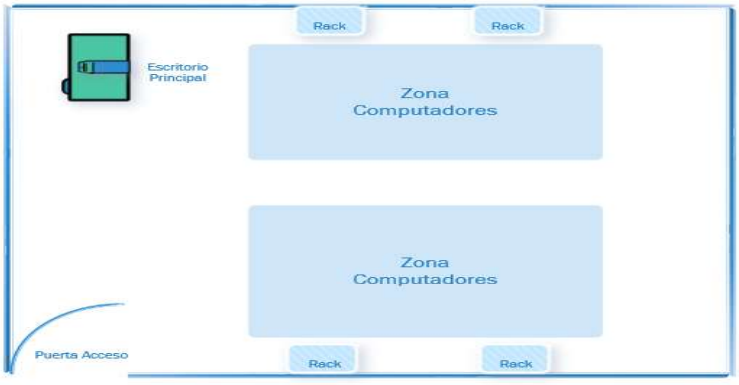


Figura 2.1. Diagrama del Laboratorio de Comunicaciones Unificadas

Los principales elementos que se puede encontrar dentro del laboratorio son computadores y equipos de telefonía, además de otros objetos necesarios para las prácticas que ahí se desarrollan. La falta de un servidor dentro del laboratorio hace que para el diseño del sistema se considere este detalle, ya que las características para un óptimo funcionamiento del software controlador del sistema serán determinadas por el hardware del computador que sea tomado como contenedor del aplicativo para administrar el control de acceso.

Actualmente, el Laboratorio de Comunicaciones Unificadas no dispone de un sistema de control de acceso, además, no se tiene un registro del personal que se encuentra utilizando las instalaciones. Es evidente la falta de un sistema más seguro para controlar el acceso al laboratorio. De acuerdo con la visita en sitio, es viable la instalación de un sistema de control de acceso, ya que se cuenta con el espacio necesario para la instalación y funcionamiento del mismo.

2.1.2. Análisis de Requerimientos

Para el análisis de requerimientos se definieron dos actores, en este caso el Administrador del Laboratorio y el Diseñador/Programador que serán los encargados de recopilar los requisitos necesarios del sistema prototipo. Se definen y detallan los requerimientos funcionales y no funcionales [13], en las Tablas 2.1, 2.2, 2.3, 2.4.

Debido a la metodología empleada se ha tomado en cuenta el desarrollo de dos prototipos: **M1** y **M2**; la descripción de los requerimientos se la ha dividido de acuerdo con cada prototipo para evidenciar la característica de la metodología en referencia a la refinación de requerimientos. Sin embargo, cabe mencionar que al ser un modelo evolutivo solo se consideró dos prototipos: el M1 como una versión inicial y el M2 como la versión final del prototipo.

2.1.2.1. Requerimientos Prototipo M1

En la Tabla 2.1 se definen los requerimientos funcionales y en la Tabla 2.2 los requerimientos no funcionales correspondientes al prototipo M1.

Tabla 2.1. Requerimientos Funcionales M1

CÓDIGO	DESCRIPCIÓN
RF1	Controlar acceso al laboratorio.
RF2	Permitir acceso solo al personal registrado.
RF3	Permitir el ingreso de nuevos rostros.
RF4	Guardar un registro de las personas que hayan accedido al laboratorio.
RF5	Verificación del personal que haya ingresado, mostrar reportes.
RF6	La aplicación debe detectar y reconocer los rostros registrados.
RF7	La aplicación debe presentar una interfaz gráfica para su administración.
RF8	La cerradura debe ser controlada por las funciones del software.

Tabla 2.2. Requerimientos No Funcionales M1

PROPIEDAD	CÓDIGO	DESCRIPCIÓN
Fiabilidad	**RNF1**	La aplicación debe ser funcional independientemente del sistema operativo donde esta alojada.
Disponibilidad	**RNF2**	El sistema debe estar disponible al menos en horario laboral.
Procesamiento	**RNF3**	El sistema debe estar alojado en un servidor con una capacidad media-alta para el procesamiento de rostros.
Facilidad de uso	**RNF4**	El manejo de la aplicación debe ser sencillo e intuitivo.
Facilidad de uso	**RNF5**	Se debe generar un manual de uso del sistema.
Capacidad	**RNF6**	Debe permitir el ingreso de al menos 5 imágenes de rostros, por cada persona.

2.1.2.2. Requerimientos Prototipo M2

Para este prototipo, se recurre a la refinación de los requerimientos descritos para el prototipo previo.

Tabla 2.3. Requerimientos Funcionales M2

CÓDIGO	DESCRIPCIÓN
RF1	Controlar y permitir acceso al laboratorio al personal registrado.
RF2	Permitir registrar nuevos rostros.
RF3	Permitir la eliminación de rostros de personas registradas.
RF4	Guardar un registro de las personas que hayan accedido al laboratorio.
RF5	Verificación del personal que haya ingresado, mostrar reportes.
RF6	La aplicación debe detectar y reconocer los rostros registrados.
RF7	La aplicación debe presentar una interfaz gráfica para su administración.
RF8	La cerradura debe ser controlada por las funciones del software.

Es importante mencionar que, en esta etapa de refinamiento, se ha añadido el requerimiento RF3 y se ha eliminado el RF2 de la Tabla 2.1 ya que se considera como un solo requerimiento en la Tabla 2.3 (RF1). Además, se tienen ciertas variaciones en la descripción de algunos de los requerimientos.

Tabla 2.4. Requerimientos No Funcionales M2

PROPIEDAD	CÓDIGO	DESCRIPCIÓN
Fiabilidad	**RNF1**	La aplicación debe ser funcional independientemente del sistema operativo donde este alojada.
Disponibilidad	**RNF2**	El sistema debe estar disponible al menos en horario laboral.
Procesamiento	**RNF3**	El sistema debe estar alojado en un servidor con una capacidad media-alta para el procesamiento de rostros.
Facilidad de uso	**RNF4**	El manejo de la aplicación debe ser sencillo e intuitivo.
Facilidad de uso	**RNF5**	Se debe generar un manual de uso del sistema.
Capacidad	**RNF6**	Debe permitir el ingreso del número de imágenes de rostros que el usuario considere, por cada persona.
Robustez	**RNF7**	El porcentaje de acierto en la detección debe ser alto.

Para el caso de los requerimientos no funcionales, se añadió el RNF7 y no se consideró necesario más cambios.

2.1.3. Componentes del sistema prototipo

Luego de determinar los requerimientos que debe cumplir el sistema, específicamente su software controlador, es importante definir los componentes principales del prototipo, como se observa en la Tabla 2.5:

Tabla 2.5. Componentes del sistema prototipo

N°	ELEMENTOS	COMPONENTE
1	PC – Servidor	Hardware
	Raspberry	Hardware
	Cerradura	Hardware
	Cámara	Hardware
2	Dispositivo red (switch)	Conectividad
3	Aplicación	Software

Principalmente se identifican tres componentes que son: hardware, software y conectividad, los mismos que se componen de diferentes elementos, que al combinarlos se obtienen los subsistemas que forman el prototipo. La Tabla 2.6 especifica cada subsistema.

Tabla 2.6. Subsistemas del prototipo

SUBSISTEMA	ELEMENTOS	COMPONENTE
1	PC – Servidor	Hardware
	Aplicación	Software
2	Raspberry	Hardware
	Cámara	Hardware
	Aplicación	Software
3	Cerradura	Hardware
4	Switch	Conectividad

- **Subsistema 1:** El primer subsistema consta de dos elementos, el PC – Servidor que conjuntamente con la aplicación es el encargado de registrar y procesar las imágenes de los rostros contenidos en el repositorio ubicado en este, además de hospedar los archivos que el cliente empleará para el proceso de reconocimiento de rostros y posterior activación de la cerradura.

- **Subsistema 2:** El segundo subsistema se compone por tres elementos, la Raspberry que es el dispositivo encargado de controlar la cerradura y realizar parte del proceso de reconocimiento facial. Este subsistema estará relacionado con el subsistema 1 ya que ciertas funciones de este son necesarias para complementar el funcionamiento del subsistema 2.

- **Subsistema 3:** Este subsistema es muy simple ya que consta de la cerradura y los elementos necesarios para su correcto funcionamiento.

- **Subsistema 4:** El cuarto subsistema se compone del switch encargado de comunicar tanto al PC – Servidor con la Raspberry cliente, para fines de organización se lo ha definido como un subsistema individual, debido a que, por las características de los demás elementos, este podría variar ya que se podría considerar un sistema de conectividad diferente al propuesto.

Luego de definir los subsistemas que compondrán el prototipo, se observa que la aplicación encargada de controlar el sistema está incluida en los subsistemas 1 y 2. A continuación, se procede a definir los módulos que compondrán la aplicación y los correspondientes subsistemas a los que pertenecen.

2.1.4. Definición de módulos componentes del software

Una parte importante del diseño es la definición de los módulos que el software va a tener, en base a los requerimientos definidos se determinan los módulos principales. Para esta sección se tomará en cuenta los requerimientos de la Tabla 2.3 que corresponden a los requerimientos del prototipo M2 ya que es el prototipo final. Primero formamos grupos de requerimientos como se muestra en la Tabla 2.7 y a partir de estos, se definirá las funciones que realizará cada uno de los módulos. Además de especificar a que subsistema corresponde cada módulo.

Tabla 2.7. Módulos del software

MÓDULO	REQUERIMIENTO	DESCRIPCIÓN
Acceso	RF1	Controlar y permitir acceso al laboratorio al personal registrado.
	RF8	La cerradura debe ser controlada por las funciones del software.
Buffer	RF2	Permitir registrar nuevos rostros, de acuerdo con el nuevo personal responsable del laboratorio.
	RF3	Permitir la eliminación de rostros de personas registradas.
Registro	RF4	Guardar un registro de las personas que hayan accedido al laboratorio.
	RF5	Verificación del personal que haya ingresado, mostrar reportes.
WebCam	RF6	La aplicación debe detectar y reconocer los rostros registrados.
Principal	RF7	La aplicación debe presentar una interfaz gráfica para su administración.

Se define un módulo más, llamado **Rostros**, este se relaciona con los requerimientos RF2, RF3 y RF6 ya que es un complemento de los módulos que corresponden a dichos requerimientos. El módulo denominado **Principal** no es estrictamente un módulo en sí, sino que se lo ha definido de esta forma ya que es el llamado "***Programa Principal***" de la aplicación.

De acuerdo con cada requerimiento asociado a los módulos definidos, se determinan las diferentes funciones que estos realizarán.

2.1.4.1. Programa Principal

Es el encargado de articular todos los módulos dentro de un solo segmento de código para la ejecución de las diferentes funciones que realiza el prototipo. Es importante mencionar que tanto para el subsistema 1 como para el 2, el programa principal es diferente debido a las funciones que realiza cada uno de estos subsistemas.

Para el subsistema 1, este se compone de:

- Interfaz gráfica para administrar ingreso de rostros y verificación de reportes y los módulos que se pueden observar en la Figura 2.2.

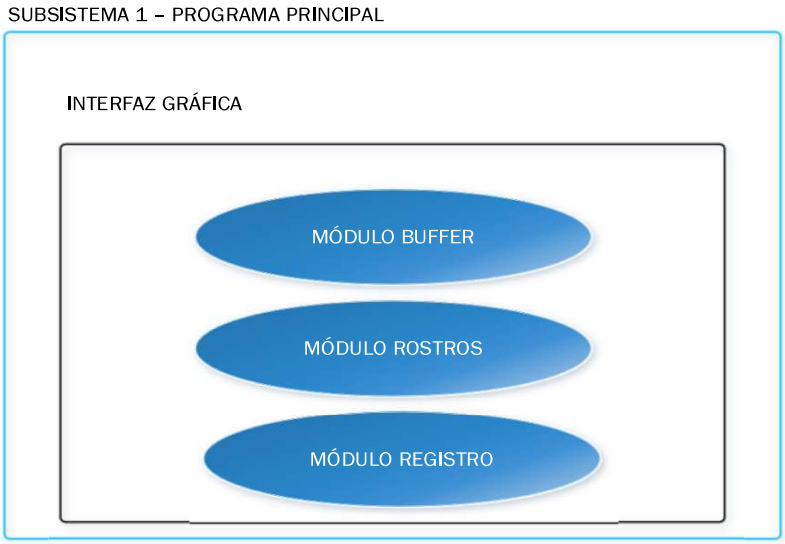


Figura 2.2. Módulos componentes del Subsistema 1

Para el subsistema 2, se define un script para la ejecución de los módulos correspondientes a este subsistema como se observa en la Figura 2.3.

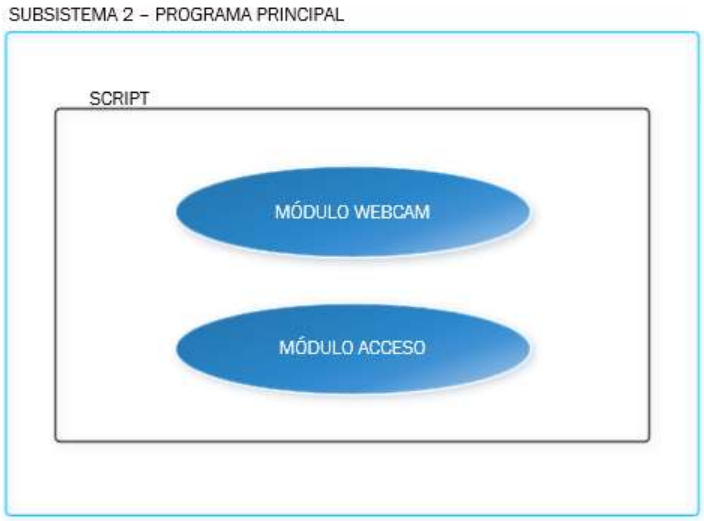


Figura 2.3. Módulos componentes del Subsistema 2

2.1.4.2. Módulos y funciones

Conociendo a que subsistema pertenece cada uno de los módulos, es importante también determinar las funciones que cumplirán cada uno de estos, las mismas que se relacionan directamente con los requerimientos antes definidos. La Tabla 2.8 muestra las funciones principales que contendrán cada uno de los módulos.

Tabla 2.8. Funciones principales de cada módulo

MÓDULOS	FUNCIONES	SUBSISTEMA
Buffer	Almacenamiento de rostros Eliminación de rostros	1
Rostros	Codificación de rostros	
Registro	Reportes de acceso	
WebCam	Reconocimiento de rostros en video	2
Acceso	Control de la cerradura (acceso)	

Se han definido seis funciones principales que componen el software para cumplir con los requisitos antes definidos, sin embargo, al momento de implementar el prototipo se podrían necesitar funciones secundarias para conseguir los objetivos trazados.

2.1.5. Diagramado de funciones principales

En esta sección, se realizará los diagramas de flujo correspondientes a las funciones principales definidas en la Tabla 2.8, como se sabe un diagrama de flujo describe un proceso, sistema o algoritmo informático además que son muy utilizados para documentar, planificar y comunicar procesos complejos ya que estos diagramas son muy simples y fáciles de comprender [15].

Un diagrama de flujo es muy útil al momento de escribir un programa o algoritmo, ya que permite explicar la lógica de este de una manera muy sencilla y detallada, además de brindar una visión general del proceso que desarrolla cierto algoritmo [15]. Debido a esto, se considera el uso de estos diagramas para determinar los procesos que sigue cada una de las funciones principales del software controlador del prototipo.

2.1.5.1. Módulo Buffer

a. Almacenamiento de rostros

La Figura 2.4 muestra cómo se realiza el almacenamiento de nuevos rostros dentro del repositorio, para esto es necesario ingresar inicialmente el nombre de la persona a la que le corresponde el rostro, además de definir el modelo que se empleará para la detección del rostro a través de la cámara, para así asegurar que es un rostro humano la imagen que se ingresará. Es necesario presionar una tecla, para controlar el ingreso

del número de imágenes que se deseen tomar de un rostro en particular, finalmente se guardarán estas y finalizará el proceso.

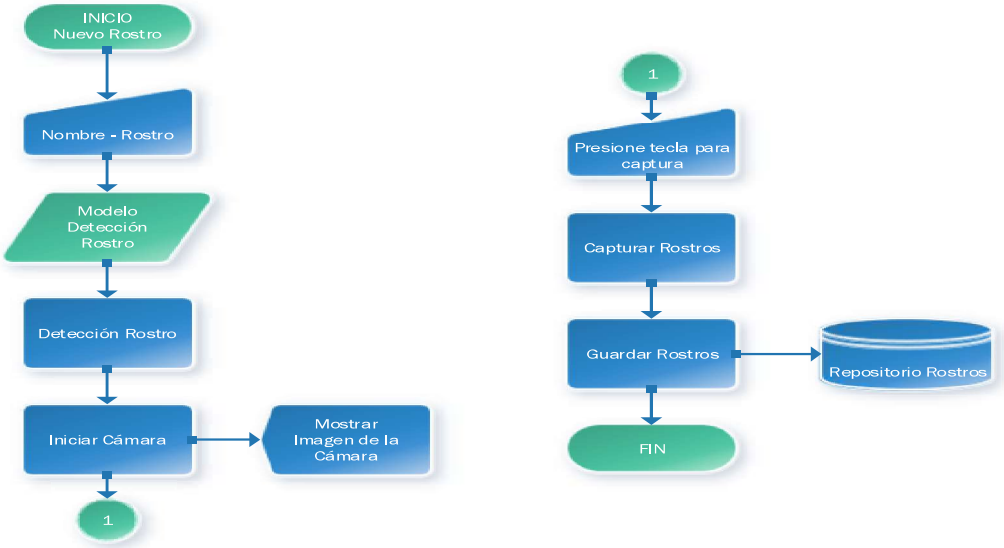


Figura 2.4. Diagrama de flujo - Función de almacenamiento de rostros

b. Eliminación de rostros

La Figura 2.5 muestra el proceso para la eliminación de un determinado directorio de rostros, para esto es necesario brindarle al administrador el listado de personas ingresadas, y que especifique el nombre de la persona que desea eliminar del repositorio.

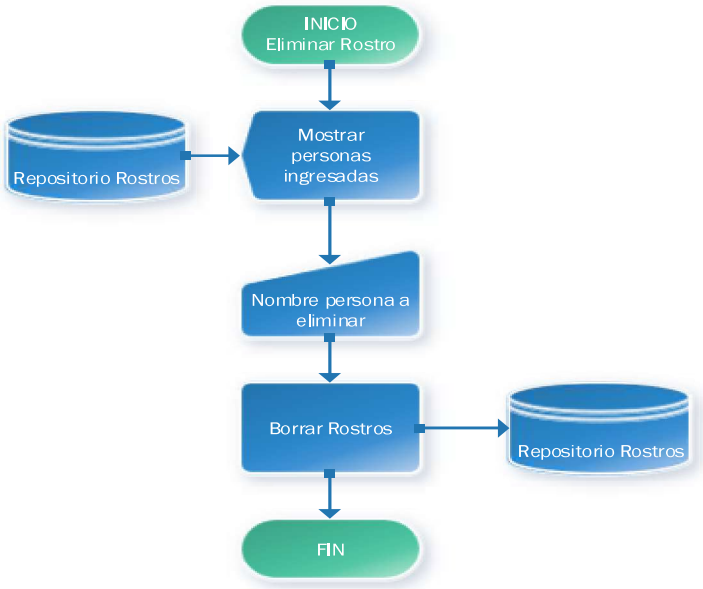


Figura 2.5. Diagrama de flujo - Función eliminar rostros

2.1.5.2. Módulo Rostros

a. Codificación de rostros

Esta función es un tanto particular, ya que se relaciona con el módulo Buffer, además se la ha definido dentro de un módulo independiente debido al proceso que realiza, al codificar los rostros que se encuentran dentro del repositorio ayudará a mejorar el proceso de reconocimiento y así reducir los posibles errores que se podrían suscitar.

La Figura 2.6 indica el proceso de codificación de los rostros almacenados dentro del repositorio, es necesario definir el nombre que llevara el archivo que este algoritmo va a generar, de la misma forma es necesario definir el método de detección de rostros que necesita este algoritmo para brindar una solución óptima. Finalmente, este archivo generado será almacenado en la PC – Servidor ya que será empleado por la Raspberry para el proceso de reconocimiento de rostros.

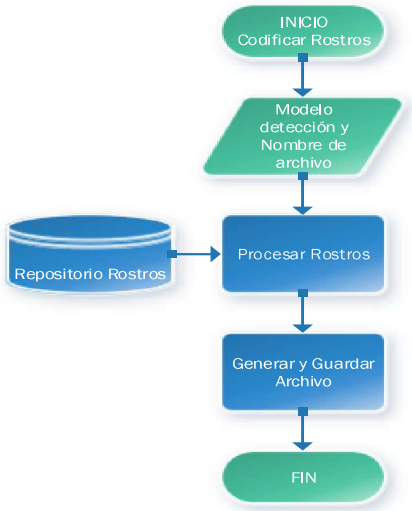


Figura 2.6. Diagrama de flujo - Función codificar rostros

2.1.5.3. Módulo Registro

a. Reportes de acceso

Esta función se encarga de mostrar un reporte de las personas que hayan ingresado al laboratorio, además permite observar un reporte ya sea por fecha o por nombre. La Figura 2.7 muestra el proceso para generar un reporte de acceso, para esto se hace uso de un archivo el cual se irá actualizando cada vez que una persona ingrese al laboratorio.

Es importante mencionar que este archivo está alojado dentro del PC-Servidor.

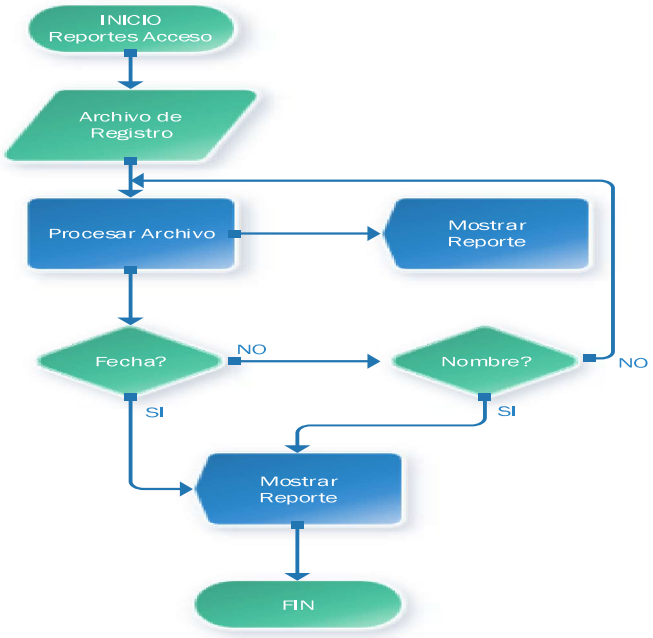


Figura 2.7. Diagrama de flujo - Función reporte acceso

2.1.5.4. Módulo WebCam

a. Reconocimiento de rostros en video

Este módulo es el principal dentro de todo el sistema prototipo, debido a que es el encargado de realizar el reconocimiento de rostros a través de la captura de video en tiempo real a través de una cámara web, además también actualiza el archivo de registro de acceso que es necesario para la generación de los reportes y finalmente su respuesta determina si se otorga o no acceso al laboratorio.

La Figura 2.8 detalla el proceso que realiza el algoritmo de la función de reconocimiento de rostros, para esto es necesario precisar el archivo de rostros codificados el cual se empleará para reconocer un rostro y también definir el método de detección de rostros. Al iniciar la captura de video, tomará un frame del rostro a través de la cámara para la posterior comparación y reconocimiento, de encontrar dicho rostro dentro del archivo se dará una respuesta positiva, se actualizará el archivo de registro de acceso, y se otorgará acceso al laboratorio, en caso de no encontrar dicho rostro el acceso no será permitido.

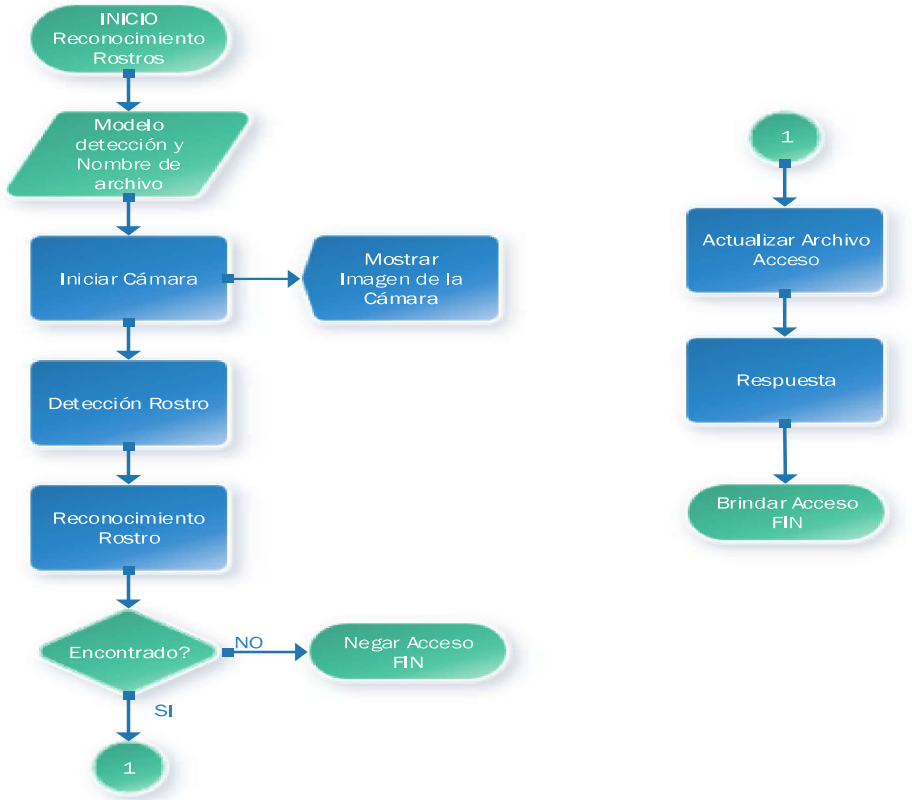


Figura 2.8. Diagrama de flujo - Función reconocimiento rostros

2.1.5.5. Módulo Acceso

a. Control de cerradura

La Figura 2.9 muestra el proceso para controlar la cerradura, a partir de la respuesta que brinde la función de reconocimiento está se activará o no dependiendo del caso. Esta función lo único que hará es controlar o no la cerradura a través de la Raspberry.

Figura 2.9. Diagrama de flujo - Función control de cerradura

2.1.6. Diseño de la interfaz gráfica

El subsistema 1, donde se encuentra la PC – Servidor, es el encargado de la administración de rostros, es decir el ingreso de nuevos rostros, eliminación de rostros y mostrar reportes de acceso, debido a esto se considera importante el desarrollo de una interfaz gráfica para que la interacción entre la aplicación y el usuario sea lo más intuitiva y amigable.

En la sección 2.1.2 se tomó en cuenta los dos prototipos que se desarrollaron, sin embargo, el segundo prototipo es el de interés ya que es el resultado final de este proyecto, para esta sección se vuelve a mencionar ambos prototipos debido a que la metodología ayuda en la parte del desarrollo de interfaces gráficas. A pesar de que las diferencias son mínimas en la presentación de cada interfaz, es necesario verificar los cambios realizados, los mismo que se presentan a continuación.

2.1.6.1. Diseño Interfaz Prototipo M1

El diseño de la interfaz está ligado a las funciones que desempeña la PC – Servidor, por esto se plantea un modelo de interfaz como se observa en la Figura 2.10. El diseño presenta dos secciones principales: la primera se encarga del ingreso de nuevos rostros y la codificación de estos, y la segunda se encarga de mostrar los reportes de acceso, en este caso se considera importante el incluir filtros para determinar el reporte ya sea por el nombre de la persona o por la fecha.

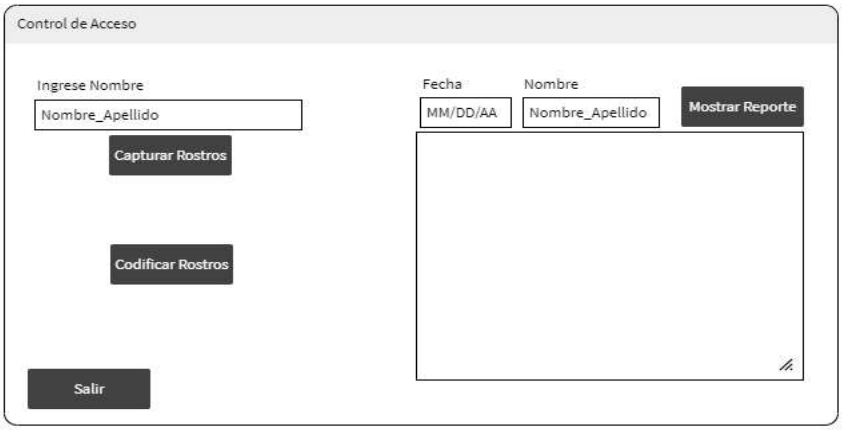


Figura 2.10. Modelo de interfaz gráfica para M1

2.1.6.2. Diseño Interfaz Prototipo M2

El diseño de la interfaz para M1 se lo considera el inicial, para el caso de M2 se han realizado unas pequeñas variaciones como se muestra en la Figura 2.11. Es relevante mencionar que el diseño de cada interfaz está ligado a los requerimientos del prototipo.

Al igual que en el diseño inicial, este se compone de dos secciones con la diferencia de que en la primera sección se ha añadido la funcionalidad de poder eliminar el directorio que contiene los rostros de alguna de las personas que ya no se necesite que estén registradas.

Figura 2.11. Modelo para interfaz gráfica para M2 - Final

Como se puede notar, los diseños son simples, ya que hay que considerar las limitaciones de Python para el desarrollo de interfaces gráficas, debido a esto que no se considera necesario la inclusión de más elementos, sin embargo, la interfaz gráfica cumple con los requerimientos definidos. Debido a la metodología empleada, el diseño final de la interfaz gráfica se definirá en la etapa de implementación ya que la metodología permite realizar cambios que mejoren la funcionalidad de esta.

2.1.6.3. Diagrama de clases

Para el desarrollo de la interfaz gráfica se define esta como una clase, debido a recomendaciones de que la mejor forma de desarrollar una interfaz gráfica con Python es a partir de emplear el paradigma orientado a objetos [16] [17] [18] . Debido a esto se ha definido una clase llamada Principal que es la encargada de contener todos los widgets[3] y las diferentes funciones que esta empleará.

La Figura 2.12 muestra la única clase definida para el desarrollo de la interfaz gráfica.

Las funciones definidas dentro de la clase se relacionan con los módulos que se encuentran en el subsistema 1 – Servidor. Estas funciones son las que permitirán la

[3] En Python se conoce como widget a los elementos gráficos que componen una interfaz, sean estos botones, etiquetas, contenedores, entre otros [46].

interacción entre todos los módulos definidos, tanto para el Servidor como para el Cliente.

La función init() es la principal dentro de la interfaz ya que esta será la que contendrá la creación y definición de los distintos widgets que se emplearan para el diseño de la interfaz.

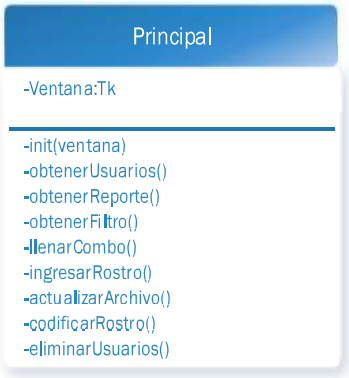


Figura 2.12. Clase Principal - Interfaz Gráfica

2.1.7. Diseño del hardware del sistema prototipo

2.1.7.1. Requerimientos físicos del sistema prototipo

El sistema prototipo ha sido definido como un modelo Cliente-Servidor para facilitar el cumplimiento de ciertas funcionalidades, es necesario explicar la necesidad de emplear este modelo. Primero se define como servidor al dispositivo encargado de:

- Almacenar las imágenes de los rostros
- Realizar el proceso de codificación de rostros
- Almacenar el archivo de control de acceso

Estas funciones son las que debe cumplir el servidor, para esto se requiere de una buena capacidad de almacenamiento y procesamiento, ya que el proceso de codificación consume varios recursos del dispositivo.

En el caso del cliente, se determina que debe encargarse de los procesos de:

- Reconocimiento de rostros
- Control de acceso (cerradura)

Es importante dar a conocer que este dispositivo debe ser capaz de controlar la cerradura que bridará o no acceso al laboratorio.

Debido a estas razones se consideró necesario emplear el modelo Cliente-Servidor para el funcionamiento del sistema de control de acceso, ya que cada dispositivo debe cumplir con una determinada función de forma independiente.

En la Tabla 2.9 se describen los requerimientos físicos del sistema prototipo de control de acceso.

Tabla 2.9. Requerimientos físicos del sistema prototipo

N°	Requerimiento
1	Computador de capacidad media-alta para que actúe de servidor.
2	Dispositivo controlador del acceso al laboratorio.
3	Dispositivo de conectividad
4	Cerradura
5	Dispositivos de seguridad eléctrica
6	Cableado eléctrico y de datos.

2.1.7.2. Selección de elementos físicos

En base a los requerimientos de la Tabla 2.9 se realizará la selección de los dispositivos que pueden cumplir con estos, además de determinar las capacidades mínimas que debería tener cada uno de ellos.

Para esto, la Tabla 2.10 describe cada uno de los dispositivos necesarios con sus características, asociándolo con su respectivo requerimiento.

Tabla 2.10. Características mínimas de dispositivos para el sistema prototipo

Requerimiento	Dispositivo	Características mínimas
1	Computador (Laptop o Escritorio)	Procesador: 2.5 GHz RAM: 4GB Disco Duro: 256 GB Cámara integrada (Laptop) Cámara web (Escritorio)
2	Raspberry Pi	Procesador: 1.4 GHz GPU Wi-Fi integrado o Adaptador Puerto Ethernet Puertos GPIO RAM: 1GB Almacenamiento: 32GB Puertos USB
2	Cámara	WebCam o PiCam
3	Switch	8 puertos 100Mbps
4	Cerradura	Tipo: electromagnética Capacidad: 200 Lbs
5	Fuentes	Correspondiente a cada dispositivo
6	Cables eléctricos Cable UTP	18 AWG Categoría 5E

Luego de determinar los dispositivos necesarios para la implementación del prototipo, se procede en base a las características definidas a precisar los dispositivos que serán utilizados en la implementación del sistema. A continuación, se detalla cada dispositivo con sus respectivas características.

a. Computador o laptop

Se ha seleccionado una laptop con las siguientes características (ver Figura 2.13), cumpliendo con los requisitos mínimos.

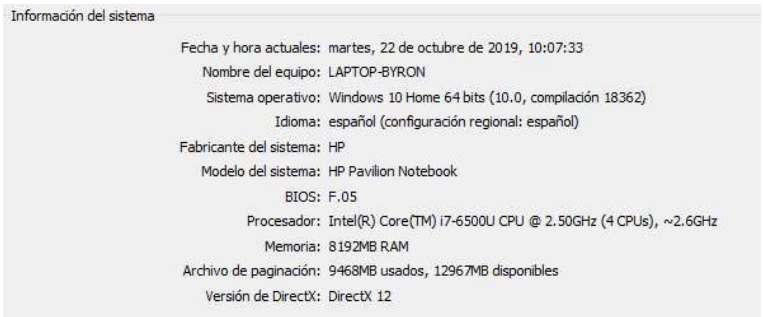


Figura 2.13. Características del computador (laptop) seleccionado.

El disco duro tiene una capacidad de 1 Tb.

b. Raspberry Pi

Debido a la variedad de versiones que se pueden encontrar de Raspberry, y debido a que durante el desarrollo se lanzó la última versión de esta con nuevas características se seleccionaron dos dispositivos que corresponden al prototipo M1 y M2.

- Para M1 se seleccionó el modelo 2B (ver Figura 2.14) con sus características [19]:

  Procesador: Broadcom BCM2836 de 900 MHz ARM Cortex-A7 de cuatro núcleos con GPU.

  GPU: VideoCore IV de doble núcleo, proporciona una tecnología Open GL ES 2.0, hardware acelerado OpenVG y admite imágenes de alta resolución 1080p.

  RAM: 1 GB

  Salida de vídeo: HD 1080p

  Conector hembra Ethernet: RJ45 10/100 BaseT

  Conector hembra de vídeo/audio: HDMI

  USB: 4 conectores hembra USB 2.0

  Conector: MPI CSI-2 de 15 vías para cámara de vídeo HD Raspberry Pi.

  Conector: para tarjeta MicroSD

  GPIO: Conector macho de 40 pines

Figura 2.14. Raspberry Pi 2 modelo B [19].

- Para M2, se seleccionó la versión 4B (ver Figura 2.15) lanzada en junio del presente año debido principalmente a la mejora en su capacidad de procesamiento [20].

  Procesador: ARM Cortex-A72

  GPU: VideoCore VI (con soporte para OpenGL ES 3.x)

  Memoria RAM: 2 GB

Conectividad: Bluetooth 5.0, Wi-Fi 802.11ac, Gigabit Ethernet

Puertos: GPIO 40 pines

2 micro HDMI

4 USB 2-2.0 y 2-USB 3.0

CSI (cámara Raspberry Pi)

DSI (pantalla táctil)

Micro SD

Conector de audio Jack

USB-C (alimentación)

Figura 2.15. Raspberry Pi 4 modelo B.

Ambos modelos disponen de un puerto para memoria microSD, la misma actúa como disco duro de las Raspberry y contiene el sistema operativo, es recomendable emplear una memoria de al menos 32 Gb para no tener ningún problema al momento de instalar las diferentes dependencias y paquetes necesarios para el funcionamiento del prototipo.

c. Cámara

Se seleccionó la WebCam Logitec c270 (ver Figura 2.16) que presenta las siguientes características [21]:

Resolución máx.: 720p/30 fps

Tipo de enfoque: foco fijo

Tecnología de lente: estándar

Micrófono integrado: mono

Campo visual: 60°

Gracias a que la Raspberry dispone de puertos USB, es posible emplear una cámara web para la implementación del prototipo. Además, por la facilidad al momento de adquirir una cámara web en el mercado.

Figura 2.16. WebCam Logitech c270

d. Dispositivos de conectividad

Se ha definido un tipo de comunicación cableada, por esta razón se ha seleccionado un conmutador o switch (Ver Figura 2.17) para la conexión entre cliente y servidor, con las siguientes características:

Modelo: TP-link TL-SF1008D
Puertos: 8 – 10/100 Mbps
Velocidad de conmutación: 1.6 Gbps

Figura 2.17. Switch TP-link TL-SF1008D

e. Cerradura

La selección de la cerradura se la ha hecho en base a la funcionalidad de los puertos GPIO de la Raspberry, ya que esta será la encargada de controlar el funcionamiento de la cerradura, por esta razón se determina que es factible emplear una cerradura de tipo electromagnética (ver Figura 2.18) con las siguientes características:

Capacidad: 600 Lb
Alimentación: 12 – 24V

Figura 2.18. Cerradura Electromagnética

f. Fuentes y elementos de seguridad eléctrica

Es importante para cualquier tipo de conexión eléctrica utilizar los diferentes elementos de seguridad, para que los dispositivos funcionen correctamente y de una forma óptima, de esta forma se puede garantizar que el tiempo útil de cada dispositivo será más duradero.

Los dispositivos críticos para el sistema prototipo son: raspberry, cerradura y switch, debido a que son los que más tiempo estarán en funcionamiento, otro factor importante es que estos dispositivos se accionan con diferentes voltajes en corriente continua. Al trabajar con este tipo de voltajes hay que tener especial cuidado para no provocar averías en los equipos.

De acuerdo con las especificaciones de cada dispositivo de ha seleccionado las fuentes de voltaje adecuadas, estas son:

- **Fuente para Raspberry**

La Tabla 2.11 define sus especificaciones.

Tabla 2.11. Características de fuentes para Raspberry

MODELO	CARACTERISTICAS
2B	Entrada: 100 – 240V / 50 – 60Hz / 0.5A (AC) Salida: 5V / 2.5A
4B	Entrada: 100 – 240V / 50 – 60Hz / 0.8A (AC) Salida: 5.1V / 3.5A Conector: USB tipo C

La principal diferencia se evidencia en el tipo de conector, ya que desde el modelo 4B Raspberry ha decidido utilizar un USB tipo C.

- **Fuente para Cerradura Electromagnética**

La fuente seleccionada para alimentar la cerradura tiene una característica que permite modular el voltaje que esta pueda proporcionar, la Figura 2.19 muestra la fuente de voltaje, que a su vez cumple la función de transformador ya que energizamos este a través de un voltaje de corriente alterna, para esto es necesario proporcionar el voltaje AC a través de otro dispositivo (ver Figura 2.20).

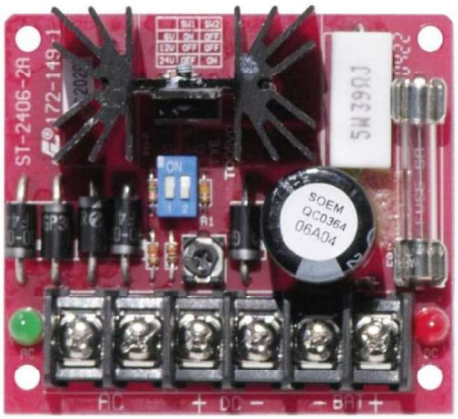


Figura 2.19. Fuente/Transformador 6 - 12 - 24V

Figura 2.20. Transformador 120V AC – 16.5V AC

Además de estos elementos, es importante mencionar que para accionar la cerradura desde la Raspberry es necesario emplear un módulo de relé, para la interacción entre el circuito que alimenta a la cerradura con los puertos GPIO de la Raspberry. El uso de este módulo tiene como finalidad no solo permitir el control de la cerradura, sino también proteger a la Raspberry de voltajes residuales que puedan afectar a esta.

- **Módulo Relé (Ver Figura 2.21)**

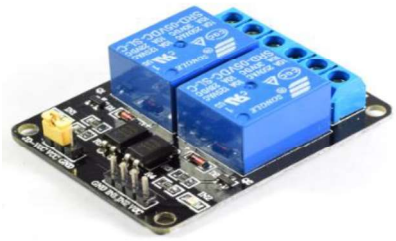


Figura 2.21. Módulo relé doble - 5V

Con sus especificaciones técnicas definidas en la Tabla 2.12.

Tabla 2.12. Especificaciones técnicas - Módulo Relé [22] [23]

N°	Especificación
1	Voltaje de Operación: 5V DC
2	Señal de Control: TTL (3.3V o 5V)
3	Nº de Relés (canales): 2 CH
4	Modelo Relé: SRD-05VDC-SL-C
5	Capacidad máx: 10A/250VAC, 10A/30VDC
6	Corriente máx: 10A (NO), 5A (NC)
7	Tiempo de acción: 10 ms / 5 ms
8	Entrads Optoacopladas
9	Indicadores LED de activación

2.2. IMPLEMENTACIÓN

En esta sección se detalla el proceso de codificación de los diferentes módulos definidos en la sección anterior, así como también la forma en la que los diferentes subsistemas se van a acoplar para formar el sistema prototipo completo. Debido a que se definió dos prototipos (M1 y M2), se resaltará más la implementación del prototipo final (M2), sin embargo, se hará referencia en las partes donde se haya realizado un cambio o mejora con respecto al primer prototipo (M1).

2.2.1. Sistema Operativo

Uno de los principales criterios a tomar en cuenta para el proceso de implementación del prototipo es la selección de los sistemas operativos que se van a emplear tanto en el PC como en la Raspberry.

2.2.1.1. PC – Servidor

Se selecciona el sistema operativo Windows debido a que es el más conocido y por su facilidad de uso. Para la implementación se trabajará específicamente con la versión Windows 10 Home.

2.2.1.2. Raspberry – Cliente

Para Raspberry existen varios sistemas operativos que son compatibles con esta tarjeta, sin embargo, el fabricante recomienda el uso de Raspbian [24] a través del uso de NOOBS que es un asistente de instalación de sistemas operativos para Raspberry [25].

Además, Raspbian es un sistema con base Linux, diseñado específicamente para administrar los diferentes modelos de Raspberry Pi [26].

Por esta razón el sistema seleccionado ha sido: Raspbian Buster 10.

2.2.2. Instalación de Python

El lenguaje de programación Python viene por defecto instalado en varios de los sistemas operativos conocidos, para este caso en Windows y Raspbian, lo importante es verificar la versión que cada sistema tiene instalada. Para verificar la versión instalada realizamos lo siguiente:

2.2.2.1. Windows 10 Home – Proceso Instalación Python

a. Ingresar a la ventana de comandos presionando **Inicio+R**, y escribir las letras **cmd** como se observa en la Figura 2.22.

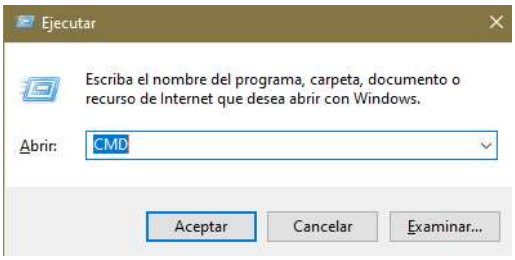


Figura 2.22. Ingreso al shell o ventana de comandos – Windows

b. En la ventana de comandos tipear la palabra "*python*" para verificar la versión instalada (Ver Figura 2.23)

```
Microsoft Windows [Versión 10.0.18362.418]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Byron Arias>python
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura 2.23. Versión de Python en Windows 10

Como se observa en la Figura 2.23 la versión de Python en la PC es la 3.7.0 siendo esta una de las últimas versiones estables [1]. Para este prototipo se trabajó con la versión 3.4.0 o superior, así que la PC cumple con este requisito.

El proceso de instalación completo se lo detalla en el ANEXO A.

2.2.2.2. Raspbian Buster – Proceso Instalación Python

a. Para verificar la versión instalada en este sistema operativo, una forma es ingresar al terminal de comandos, dando *click* en el icono de la parte superior izquierda de la barra de tareas del escritorio de Raspbian, como se muestra en la Figura 2.24.

Figura 2.24. Icono de la ventana de terminal – Raspbian

b. Escribir el comando "*python*" en el terminal para verificar la versión que dispone este sistema (Ver Figura 2.25).

```
pi@raspberrypi:~ $ python
Python 2.7.16 (default, Apr  6 2019, 01:42:57)
[GCC 8.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura 2.25. Versión de Python – Raspbian

En este caso la versión es la 2.7.16 que es la versión que por defecto viene instalada en las distribuciones de Linux. Como para el prototipo en necesario una versión superior es necesario instalarla en el sistema operativo.

El proceso de instalación es sencillo, para esto es necesario abrir el terminal de comandos y escribir la siguiente instrucción:

sudo apt-get install python3-dev

Esto permitirá instalar la última versión estable de Python en el sistema Raspbian.
Es importante mencionar que tanto para Windows como para Raspbian se debe emplear la versión 3 de Python, no se recomienda utilizar la versión 2 debido a que hay ciertas diferencias en sus sintaxis, así se evitaría cualquier error en el desarrollo del prototipo.

En el ANEXO A se detalla de mejor manera el proceso de instalación de Python en Raspbian.

2.2.3. Instalación de OpenCV

Luego de instalar Python, es necesario instalar la librería OpenCV en cada uno de los sistemas operativos, ya que esta será empleada en los procesos de reconocimiento de rostros. El proceso de instalación de esta librería es un tanto más complejo, a continuación, se explica lo más importante de los procesos que se deben seguir para la correcta instalación tanto en Windows como en Raspbian, en el ANEXO B se realiza una explicación detallada de cada procedimiento.

Antes de realizar la instalación de OpenCV, definimos el entorno de desarrollo integrado (IDE[4]) que se va a utilizar para la codificación del software controlador del prototipo. Para Windows se empleará Visual Studio 2017, gracias a las diferentes características y herramientas [27] que posee se ha seleccionado este IDE, sin embargo, la principal razón para emplear Visual Studio es que en esta versión permite el desarrollo de aplicaciones con Python. La Figura 2.26 muestra la ventana principal del entorno de desarrollo Visual Studio 2017.

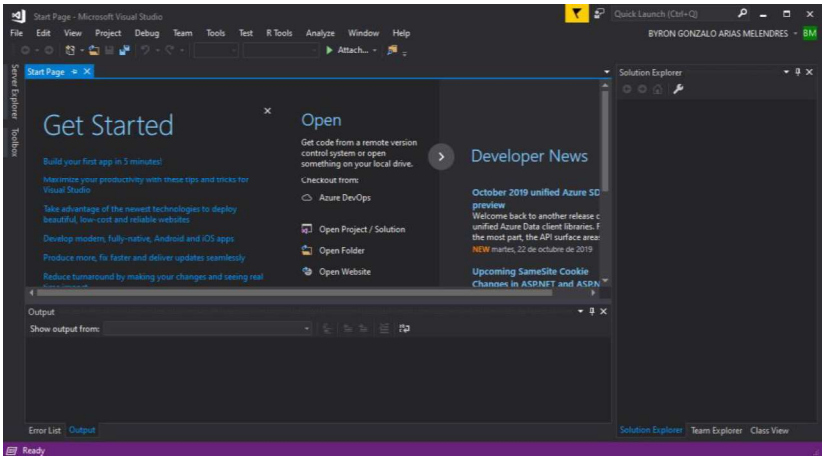


Figura 2.26. Pantalla principal - Visual Studio 2017

[4] *IDE – Integrated Development Environment,* siglas en inglés para Entorno de Desarrollo Integrado.

Además, este IDE permite instalar las diferentes librerías desarrolladas para Python de una forma directa, desde el mismo entorno de desarrollo.
En el caso de Raspbian, para visualizar o editar el código desarrollado en Python se lo puede hacer a través de cualquier editor de texto como: gedit, nano, sublimeText, entre otros. Sin embargo, este sistema operativo ofrece un IDE denominado Thonny Python IDE, que viene instalado por defecto (Ver Figura 2.27).

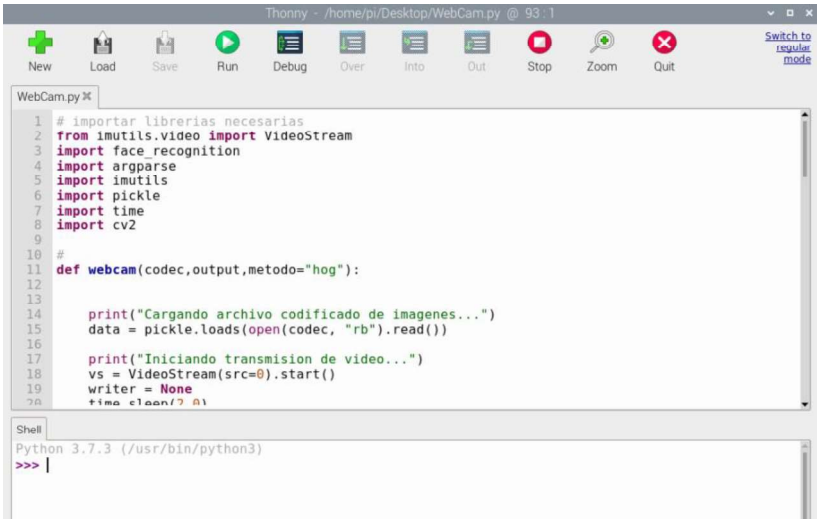


Figura 2.27. Pantalla principal - Thonny Python IDE

Es importante mencionar los IDEs que van a ser empleados, ya que, sobre todo para Windows será de gran utilidad al momento de instalar determinados paquetes o librerías necesarias para complementar el funcionamiento del prototipo.

2.2.3.1. Windows 10 Home – Instalación OpenCV

La instalación de OpenCV en este sistema operativo no es tan compleja, para esto es necesario tener instalado Python, a partir de esto dentro del terminal se procede a escribir los siguientes comandos:

python -m pip install numpy
python -m pip install opencv-python

La instalación tarda unos minutos, pero luego de instalado podemos verificar que se ha completado correctamente el procedimiento desde Visual Studio, en el explorador de soluciones donde aparecerá la librería ya instalada como se puede ver en la Figura 2.28.

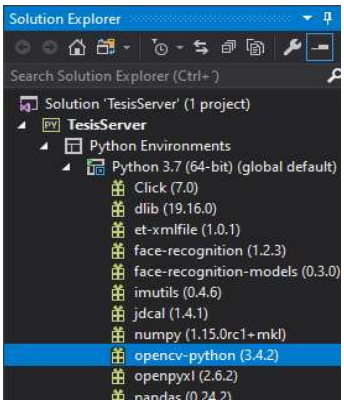


Figura 2.28. Explorador de soluciones, librería OpenCV - Visual Studio

En este caso la versión de OpenCV que se ha instalado ha sido la 3.4.2.

Antes de continuar, en esta parte es necesario explicar un nuevo concepto denominado *Virtual Environment o Python Environment,* que es una herramienta que permite crear un entorno virtual para la interpretación de un script o aplicación desarrollada en Python, la particularidad de esta herramienta es que permite definir la versión del intérprete de Python que será utilizado dentro del *Virtual Environment* definido y así no interferir con otras aplicaciones que funcionen con otra versión de Python [28].

Visual Studio crea un *Python Environment* por defecto con la versión de Python que este instalada por defecto en Windows, al iniciar con un proyecto es necesario definir estos parámetros. Para esto se selecciona la pestaña **Tools**, la opción **Python**, y en el submenú la opción **Python Environments** como se muestra en la Figura 2.29.

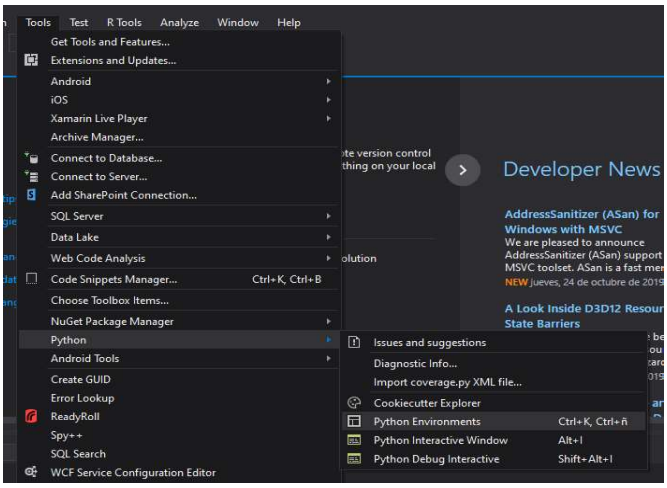


Figura 2.29. Visual Studio - Python Environments

Esta opción muestra los diferentes entornos virtuales que están disponibles, y por defecto estará seleccionado el correspondiente a la versión de Python instalada previamente en el sistema operativo (Ver Figura 2.30).

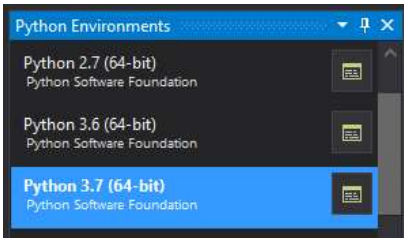


Figura 2.30. Python Environment por defecto

Una vez definido el entorno, se debe completar la instalación de las diferentes librerías y paquetes que complementaran el proceso de implementación del prototipo. Es importante mencionar que, antes de instalar la librería OpenCV como tal, hubo la necesidad de instalar Numpy que es una librería de operaciones y funciones matemáticas con matrices y vectores, necesaria para varios procedimientos que realizan las diferentes funciones de OpenCV.

Al igual que esta librería, se deben instalar otras para las diferentes funcionalidades del prototipo, estas son (Tabla 2.13):

Tabla 2.13. Librerías complementarias - Windows

N°	Nombre	Descripción
1	Dlib	Contiene algoritmos de machine learning y análisis de datos.
2	face-recognition	Permite manipular y reconocer rostros con Python.
3	face-recognition-models	Contiene los modelos que se utilizan en la librería face-recognition.
4	Imutils	Contiene funciones para procesar imágenes en conjunto con OpenCV.
5	Openpyxl	Librería que permite el manejo de archivos en formato de tablas (bases de datos).
6	Pandas	Permite la manipulación de archivos.
7	Pillow	Es la librería de imágenes para Python. Permite la manipulación de imágenes.

Para instalar estas librerías se lo realiza de forma sencilla a través de Visual Studio con la herramienta para instalar paquetes de Python. La Figura 2.31 muestra la forma en la que se puede acceder a esta herramienta, ubicando el puntero del mouse sobre el entorno virtual, presionar el botón derecho para desplegar el menú y seleccionar la opción **Install Python Package.**

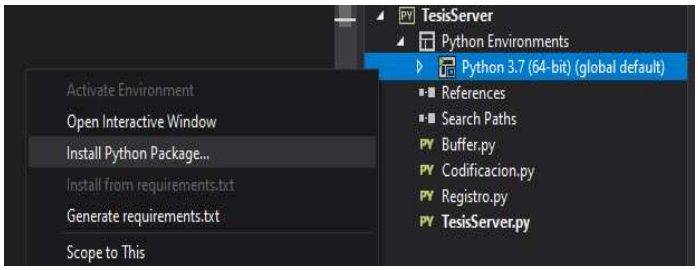


Figura 2.31. Selección de herramienta de instalación - Paquetes Python

Luego de dar *click*, se desplegará una ventana donde se podrá realizar la búsqueda del paquete que se vaya a instalar, y simplemente seleccionando dicho paquete el IDE lo instalará (Ver Figura 2.32).

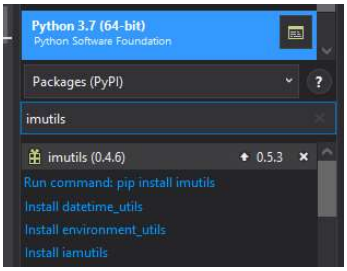


Figura 2.32. Instalación de paquetes - Python en Visual Studio

Este proceso se repite para cada una de las librerías listadas en la Tabla 2.13.

Luego de culminada la instalación, la PC estará lista tanto para el proceso de codificación de cada módulo, como para su respectivo funcionamiento.

2.2.3.2. Raspbian Buster – Instalación OpenCV

El proceso de instalación de OpenCV en Raspbian difiere un poco con respecto al de Windows, ya que no es tan sencillo debido a que se debe instalar una serie de elementos extras para una instalación completamente funcional de esta librería.

Previo a la instalación, es necesario definir la versión de OpenCV que se desea instalar, en el caso de Linux es recomendable instalar las versiones estables más actuales para evitar cualquier tipo de error. La versión Buster es la última de Raspbian, para efectos de compatibilidad y reducción de errores se selecciona la versión 4.1.1 de OpenCV a ser instalada.

Existen varias formas de realizar el proceso de instalación de OpenCV para Raspbian, sin embargo, para este trabajo se ha realizado el siguiente procedimiento, desde el terminal a través de utilizar el comando *sudo apt-get install* [29]:

a. Instalar las dependencias necesarias, esto evitara errores al momento de la compilación e instalación de la librería. En la Tabla 2.14 se listan las diferentes dependencias que deben ser instaladas, las mismas que son necesarias para el funcionamiento completo de la librería OpenCV y sus complementos.

Tabla 2.14. Dependencias para OpenCV - Raspbian

N°	Dependencia
1	build-essential
2	Cmake
3	pkg-config
4	Libjpeg-dev
5	Libtiff5-dev
6	Libjasper-dev
7	Libpng-dev
8	Libavcodec-dev
9	Libavformat-dev
10	Libswscale-dev
11	Libv4l-dev
12	Libxvidcore-dev
13	Libx264-dev
14	Libfontconfig1-dev
15	Libcairo2-dev
16	Libgdk-pixbuf2.0-dev
17	Libpango1.0-dev
18	Libgtk2.0-dev
19	Libgtk-3-dev
20	Libatlas-base-dev
21	Gfortran
22	Libhdf5-dev
23	Libhdf5-serial-dev
24	Libhdf5-103
25	Libqugui4
26	Libqtwebkit4
27	Libqt4-test
28	Python3-pyqt5

b. Al igual que en Windows es necesario definir un *Virtual Environment* para trabajar sin problemas con la versión seleccionada de Python. Para esto en necesario instalar el administrador de paquetes de Python, que no es más que el comando *pip*, ejecutando lo siguiente:

wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py

c. Una vez instalado *pip*, se procede a instalar los comandos para la creación del entorno virtual, así:

sudo pip install virtualenv virtualenvwrapper

d. Es necesario modificar el archivo *~/.bashrc* para poder utilizar estos comandos, la Figura 2.33 muestra el archivo modificado.

```
pi@raspberrypi:~ $ sudo su
root@raspberrypi:/home/pi# cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.

# Note: PS1 and umask are already set in /etc/profile. You should not
# need this unless you want different defaults for root.
# PS1='${debian_chroot:+($debian_chroot)}\h:\w\$ '
# umask 022

# You may uncomment the following lines if you want `ls' to be colorized:
# export LS_OPTIONS='--color=auto'
# eval "`dircolors`"
# alias ls='ls $LS_OPTIONS'
# alias ll='ls $LS_OPTIONS -l'
# alias l='ls $LS_OPTIONS -lA'
#
# Some more alias to avoid making mistakes:
# alias rm='rm -i'
# alias cp='cp -i'
# alias mv='mv -i'

#virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

Figura 2.33. Archivo ~/.bashrc modificado

e. Luego, crear el entorno virtual, con el siguiente comando:

mkvirtualenv cv -p python3

Donde, *cv* es el nombre del entorno virtual asociado a la versión de Python instalada, como se puede muestra la Figura 2.34.

```
root@raspberrypi:/home/pi# workon cv
(cv) root@raspberrypi:/home/pi# python
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figura 2.34. Entorno Virtual *cv* asociado a la versión 3.7.3 de Python

f. Descargar OpenCV empleando los comandos, como se observa en la Figura 2.35.

```
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.1.1.zip
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib/archive/4.1.1.zip
$ unzip opencv.zip
$ unzip opencv_contrib.zip
$ mv opencv-4.1.1 opencv
$ mv opencv_contrib-4.1.1 opencv_contrib
```

Figura 2.35. Descarga de OpenCV – Raspbian [29]

g. Al igual que en Windows, es necesario instalar el paquete Numpy.

h. Para Raspbian, y específicamente con este proceso de instalación es necesario la creación de un directorio de construcción o *build* (ver Figura 2.36), para la posterior construcción del archivo que será instalado.

```
$ cd ~/opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D ENABLE_NEON=ON \
    -D ENABLE_VFPV3=ON \
    -D BUILD_TESTS=OFF \
    -D INSTALL_PYTHON_EXAMPLES=OFF \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D CMAKE_SHARED_LINKER_FLAGS=-latomic \
    -D BUILD_EXAMPLES=OFF ..
```

Figura 2.36. Creación del directorio *build* – Raspbian [29]

i. Para la construcción del archivo a instalar, se debe ejecutar: *make -j4.* El resultado es el siguiente (Figura 2.37) si la compilación no genero errores.

```
[100%] Building CXX object modules/stereo/CMakeFiles/opencv_perf_stereo.dir/perf
/perf_main.cpp.o
[100%] Linking CXX executable ../../bin/opencv_perf_stereo
[100%] Built target opencv_perf_stereo
[100%] Linking CXX shared module ../../lib/python3/cv2.cpython-37m-arm-linux-gnu
eabihf.so
[100%] Linking CXX shared module ../../lib/cv2.so
[100%] Built target opencv_python2
[100%] Built target opencv_python3
(cv) pi@raspberrypi:~/opencv/build $
```

Figura 2.37. Resultado de la construcción del directorio *build* [29]

j. Finalmente, para instalar OpenCV, escribir en el terminal: *sudo make install.* Para comprobar que se haya instalado correctamente se emplea el comando Python (Ver Figura 2.38).

```
(cv) root@raspberrypi:/home/pi# python
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.1.1'
```

Figura 2.38. Comprobación de la instalación de OpenCV - Raspbian

En el ANEXO B se encuentra la instalación más en detalle de esta librería para Raspbian.

2.2.4. Codificación de los módulos del sistema

En esta sección se presenta la codificación de cada uno de los módulos definidos y sus respectivas funciones, como se determinó en la sección 2.1. Se analizan los fragmentos de código más relevantes y se hará énfasis en la funcionalidad de cada módulo, con el fin de justificar el cumplimiento de cada requerimiento.

2.2.4.1. Módulo Buffer

El primer módulo es el denominado Buffer, este módulo corresponde al subsistema 1 y estará alojado en el PC – Servidor, la función principal de este es la de almacenar nuevos rostros dentro del repositorio de imágenes. En primer lugar, se deberá crear el repositorio que contendrá todas las imágenes y un directorio correspondiente a cada usuario del que se desea guardar un nuevo rostro. En la línea 9 del Código 2.1 se puede observar que define dos argumentos de entrada para la función, uno de ellos *output* es el path del directorio donde se va a guardar las imágenes capturadas y el otro *cascade* se refiere al modelo de detección de rostros que se va a emplear, para este caso se utiliza el clasificador en cascada y es necesario cargar una plantilla definida con un filtro de Haar [30].

```
8   # Definimos la funcion principal del módulo
9   def buffer(cascade,output):
10
11  # Cargamos el archivo "Haar cascade" de OpenCV para la deteccion de rostros
12      detector = cv2.CascadeClassifier(cascade)
13
14  # Iniciamos la captura de video
15      print("Iniciando video")
16      vs = VideoStream(src=0).start()
17      time.sleep(2.0)
18      total = 0 #Inicializamos un contador para el numero de imagenes capturadas
```

Código 2.1. Fragmento 1 - Función *buffer*

En la línea 12, a través de la función *CascadeClassifier* que pertenece a una clase con el mismo nombre, permite cargar el clasificador que se va a emplear [31]. Para iniciar la cámara, se instancia un objeto para la clase *VideoStream* [32], como se observa en la línea 16.

En el Código 2.2, la línea 23 indica la forma en la que se va a capturar los frames a través de la cámara con la función *read* [32] y que serán procesados. Para a detección de rostros es necesario emplear la función *detectMultiScale* [31] como se observa en la línea 27.

La función *buffer* es bastante simple ya que gracias a las diferentes funciones definidas para Python se puede realizar la mayoría de los procedimientos de una forma rápida y sencilla.

```
20  # Realizamos un bucle sobre los frames de la captura de video
21      while True:
22          #Capturamos el frame, y lo redimensionamos
23          frame = vs.read()
24          orig = frame.copy()
25          frame = imutils.resize(frame, width=400)
26          #Procesamos el frame, para realizar la deteccion del rostro frente a la camara
27          rects = detector.detectMultiScale(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
28
```

Código 2.2. Fragmento 2 - Función *buffer*

La siguiente función que contiene este módulo, es la de eliminar rostros del repositorio, para esto se emplea la función *rmtree* que forma parte de la librería *shutil*, esta función se encarga de remover o borrar un directorio que contenga información, en este caso las imágenes de los rostros que deseamos eliminar [33].

```
shutil.rmtree("repImagenes\\" + nombreSeleccionado.strip('\n'))
```

Código 2.3. Función para eliminar rostros

Es interesante mencionar que la función de eliminar rostros no se consideró para el primer prototipo (M1), y este es uno de los cambios con respecto al prototipo final (M2).

2.2.4.2. Módulo Rostros

El segundo módulo tiene una particularidad, y es que se relaciona estrechamente con el módulo Buffer debido a que este se encarga de procesar el repositorio de imágenes y codificarlo, de esta forma se genera un archivo de tipo binario para realizar el reconocimiento del rostro en la Raspberry, a través del módulo WebCam. Es decir, este módulo se relaciona tanto con el módulo Buffer, como con el módulo WebCam y es parte esencial para el funcionamiento del subsistema 2.

El módulo Rostros, se alojará en el PC – Servidor, sin embargo, el archivo que este genera se lo empleará en la Raspberry.

La función principal de este módulo es la de codificar los rostros almacenados en el repositorio de imágenes, para esto se define la función *codRostros* la cual necesita de

tres parámetros como se puede ver en el Código 2.4. Para el desarrollo de este código se ha tomado como referencia [34] y [35].

```
9   # Definimos la funcion principal
10  def codRostros(repImagenes, archCodif, deteccion="cnn"):
11
12  # Leemos el path del repositorio deonde se encuentran las imagenes
13      print("Cargando imagenes...")
14      pathImagenes = list(paths.list_images(repImagenes))
```

Código 2.4. Fragmento 1 - Función *codRostros*

El parámetro *pathImagenes*, no es más que el path o ruta del repositorio de imágenes, *archCodif* es el nombre que se le da al archivo que esta función va a generar y *deteccion* es el modo de detección de rostros que se va a emplear, este modo puede ser *hog*[5] o *cnn*[6]. Como este módulo se ejecuta dentro del PC – Servidor se selecciona la opción *cnn*, debido a que este método de detección necesita de un nivel alto de procesamiento y la PC puede soportar la ejecución de este proceso.

Esta función también utiliza la librería *face_recognition* [36], para los procesos de reconocimiento y codificación de rostros. El Código 2.5 presenta en la línea 29 y 32 las formas en las que se emplean dos funciones de esta librería. Estas funciones son muy importantes ya que también se las utiliza el módulo WebCam.

```
25          #Procesaomo cada imagen, en base al modelo seleccionado "hog" o "cnn"
26          image = cv2.imread(path)
27          rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
28
29          boxes = face_recognition.face_locations(rgb, model=deteccion)
30
31          #Generamos una lista con las imagenes codificadas
32          encodings_r = face_recognition.face_encodings(rgb, boxes)
```

Código 2.5. Fragmento 2 - Función *codRostros*

Como se puede observar en la línea 29, la función *face_locations* permite detectar o localizar un rostro debido a esto uno de sus parámetros es el modo de detección ingresado en *deteccion*, para la acción de codificar los rostros detectados se emplea la

[5] Histogram of Oriented Gradients, por sus siglas en inglés, no es más que un descriptor de características utilizado para la detección de objetos en imágenes.
[6] Convolutional Neural Network, es un clasificador empleado par detección de rostros.

función *face_encodings* (ver línea 32), gracias a estas funciones se puede realizar el proceso de codificación de rostros.

Finalmente, una parte fundamental de la función *codRostros* es la generación del archivo codificado de imágenes, para esto se utiliza la librería *pickle* [37] que permite la serialización de información en forma binaria. En el Código 2.6 se observa la forma en la que ha sido codificada la generación del archivo binario.

```
39  # Generamos archivo binario de imagenes aprendidas
40      print("Codificando imagenes...")
41      data = {"codigos": listaImagenes, "nombres": listaNombres}
42      f = open(archCodif, "wb")
43      f.write(pickle.dumps(data))
44      f.close()
```

Código 2.6. Fragmento 3 - Función *codRostros*

2.2.4.3. Módulo WebCam

El módulo WebCam corresponde al subsistema 2, y estará alojado en la Raspberry; es el encargado de realizar el reconocimiento de los rostros que estarán frente a la cámara. Se puede decir que es el módulo más importante del sistema prototipo, su función principal genera dos resultados: el primero un indicador que determina si se identificó o no el rostro y el segundo, el nombre de la persona identificada.

El indicador se relaciona directamente con el módulo Acceso, ya que de esta respuesta sea positiva o negativa se otorgará o no el acceso al Laboratorio de Comunicaciones Unificadas. El segundo resultado, se relaciona con el módulo Registro, ya que este es el encargado de manipular el archivo de acceso y mostrar los reportes al administrador.

La función principal de este módulo se denomina *webcam* y acepta tres parámetros de entrada como se puede ver en la línea 11 del Código 2.7. Para el desarrollo de este código se ha tomado como referencia [35] y [38].

```
10  # Definimos el metodo pricnipal
11  def webcam(codec,output,metodo="hog"):
12
13      #Leemos el archivo binario de la codificacion de rostros
14      print("Cargando archivo codificado de imagenes...")
15      data = pickle.loads(open(codec, "rb").read())
16      #Iniciamos la camara
17      print("Iniciando transmision de video...")
18      vs = VideoStream(src=0).start()
```

Código 2.7. Fragmento 1 - Función *webcam*

Al igual que en la función *codRostros* del módulo Rostros, se tiene el parámetro método y posee la misma funcionalidad en comparación a la función mencionada, con la diferencia que para este caso se utilizará el modo *hog*, ya que la Raspberry no dispone de una alta capacidad de procesamiento. *Output* es opcional, en el caso que se desee guardar el último frame capturado por la cámara, se debe especificar el path donde deseamos guardar dicha imagen. El principal parámetro de esta función es *codec*, ya que es el archivo binario creado con anterioridad y que se utiliza para el proceso de identificación de rostros.

El proceso de identificación de rostros se lo realiza a través de una función de la librería *face_recognition* [36]. Como se pude observar en la línea 36 del Código 2.8 la función definida es *compare_faces*, esta ayuda a verificar las coincidencias entre el rostro capturado por la cámara y los datos guardados en el archivo binario.

```
34          for encoding in codec:
35              # Comparamos el frame capturado, con los que tenemos codificados
36              matches = face_recognition.compare_faces(data["codigos"],encoding)
37              nombre = "Desconocido"
38
39              if True in matches:
40
41                  #Se realiza un porceso para determinar las coincidencias en la comparacion
42                  #Y dar el resultado del rostro reconocido
43                  matchedIdxs = [i for (i, b) in enumerate(matches) if b]
44                  counts = {}
45
46
47                  for i in matchedIdxs:
48                      nombre = data["nombres"][i]
49                      counts[nombre] = counts.get(nombre, 0) + 1
50
51                  nombre = max(counts, key=counts.get)
52
53              nombres.append(nombre)
```

Código 2.8. Fragmento 2 - Función *webcam*

La codificación del proceso de comparación de rostros se realiza a través de los índices correspondientes al resultado de la función *compare_faces*, como se puede observar

entre las líneas 43 y 49, lo que hace esta función es determinar un índice por cada acierto o coincidencia, se realiza un conteo de estos y se extrae el valor más alto, para asociarlo al nombre y determinar a quién pertenece dicho rostro [35].

En caso de no existir una coincidencia, la respuesta que entrega es: "*Desconocido*", caso contrario retornará el nombre de la persona a la que le corresponde dicho rostro.

Este método debe proporcionar dos respuestas que serán utilizadas por otros módulos, este proceso se lo realiza de una forma sencilla a través de un condicional *if*, como muestra el Código 2.9.

```
81      #El metodo devuelve un valor si se ha detectado el rostro o no para permitir o no el ingreso
82      if name == "Desconocido":
83          print("No hay acceso")
84          for x in names:
85              print(x)
86          return 0,x
87      else:
88          print("Acceso concedido")
89          for x in names:
90              print(x)
91          return 1,x
```

Código 2.9. Fragmento 3 - Función *webcam*

Una ventaja de programar en Python se evidencia en esta función, ya que no hace falta definir el tipo de dato que la función retornará, ni definir la función o método de inicio para que sea capaz de retornar un tipo en particular de dato, simplemente hay que emplear la sentencia *return* como se observa en las líneas 86 y 91.

2.2.4.4. Módulo Registro

El penúltimo módulo es el de Registro, este es el encargado a través de las funciones *registro* y *filtro* de guardar la bitácora de ingreso, cada que se produzca un acceso exitoso al laboratorio y también de filtrar la información almacenada. El Código 2.10 indica la forma en la que se recopila la información del archivo de Registro.

Es importante mencionar, que para almacenar la información del personal que accede al laboratorio se emplea un archivo en formato "*.xlsx*". Este tipo de archivo cuenta con la particularidad de ser administrable a partir de la librería *pandas* orientada al uso a través del lenguaje Python.

```
6   def registro():
7
8       #Cargo el archivo de registro de acceso
9       path='C:\\Users\\Byron Arias\\Desktop\\Compartida\\Registro.xlsx'
10      doc=pd.read_excel(path)
11      tupla = [tuple(x) for x in doc.values]
12
13      for row in tupla:
14          print(row)
15
16
17      return tupla
```

Código 2.10. Función *registro*

La función *read_excel*, es la encargada de leer la información del archivo, esta función retorna una tupla[7] con la información recopilada. En el caso de la función *filtro*, lo que pretende es trabajar con la información del archivo como se observa en las líneas 24 hasta la 42 del Código 2.11, al igual que en la función anterior el parámetro retornado es una tupla con la información de interés para el usuario.

```
20  def filtro(nombre, fecha):
21      path='C:\\Users\\Byron Arias\\Desktop\\Compartida\\Registro.xlsx'
22      doc=pd.read_excel(path)
23      #Filtrado por nombre
24      if nombre and fecha=='':
25          docs=doc[doc['NOMBRE']==nombre]
26          tupla = [tuple(x) for x in docs.values]
27          print(nombre)
28      #Filtrado por fecha
29      elif fecha and nombre=='':
30          docs=doc[doc['FECHA']==fecha]
31          tupla = [tuple(x) for x in docs.values]
32          print(fecha)
33      #Filtrado por nombre y fecha
34      elif nombre=='' and fecha=='':
35          doc=pd.read_excel(path)
36          tupla = [tuple(x) for x in doc.values]
37      else:
38          docs=doc[(doc.FECHA==fecha) & (doc.NOMBRE==nombre)]
39          tupla = [tuple(x) for x in docs.values]
40          print(tupla)
41      #Retorno los valores filtrados
42      return tupla
```

Código 2.11. Función *filtro*

2.2.4.5. Módulo Acceso

El módulo Acceso es el último módulo del software del sistema, este estará alojado en la Raspberry en el subsistema 2. Para el funcionamiento de este módulo es necesario instalar la librería *RPi.GPIO* la cual permite controlar los puertos GPIO de la Raspberry.

El control de la cerradura electromagnética se lo realizará a través de dichos puertos, para desarrollar el código que controle la cerradura es necesario conocer la distribución de estos puertos, la Figura 2.39 indica como se distribuye cada pin.

[7] Se define como un conjunto de valores de cualquier tipo separados por una coma.

Figura 2.39. Puertos GPIO - Raspberry Pi 4 modelo B [39]

Los pines de interés son los de color verde, como se ve en la Figura 2.39, ya que estos pines funcionan como entradas o salidas. Se utilizará uno de estos pines con la finalidad de enviar una señal para controlar la apertura o no de la cerradura. El código 2.12 es el que se emplea para activar el pin 16.

```
6   def acceso():
7       puerto=16
8       GPIO.setmode(GPIO.BOARD)
9       GPIO.setup(puerto,GPIO.OUT)
10      GPIO.output(puerto,False)
11      time.sleep(1)
12      GPIO.output(puerto,True)
```

Código 2.12. Función *acceso*

La función *acceso* es la definida para el módulo que lleva el mismo nombre, primero se define la numeración con la que se va a trabajar, en este caso con la función *setmode* (línea 8), después es necesario indicar si el puerto trabajara como entrada o salida a través de la función *setup* (línea 9), finalmente con *output* (líneas 10 y 12) se cambia el estado de la salida del puerto utilizado, en este caso con *False* el pin queda en estado bajo[8]. Sin embargo, es recomendable devolver al pin a su estado inicial [40] para evitar errores al momento de activar nuevamente este pin.

Es importante mencionar que la cerradura electrónica se la controlará a través de otro dispositivo, un módulo de relé y en realidad la función *acceso* se encarga de activar o desactivar este dispositivo, a partir del funcionamiento del relé la cerradura se accionará.

2.2.5. Codificación de la interfaz gráfica

La interfaz gráfica ayuda con la administración de rostros y también con la verificación de reportes, estará alojada en el PC – Servidor y es necesaria la creación de esta interfaz ya que el PC es el encargado de realizar la interacción entre el usuario y la aplicación, permitiendo ingresar nuevos rostros, verificar los existentes y eliminarlos, además de generar un reporte de las personas que hayan ingresado al laboratorio.

Como se desarrolló en la sección 2.1.6, se definieron dos modelos de interfaces gráficas, en el que la principal diferencia residía en la opción de eliminar rostros. Sin embargo, debido a la característica de la metodología de prototipos, se mejoró la disposición y funcionalidad de la interfaz gráfica con dos cambios importantes que son: el proceso de eliminación de rostros y la forma de filtrar la información para generar los reportes de acceso.

[8] Los pines GPIO Entrada/Salida de la Raspberry trabajan o generan dos niveles de voltaje, nivel alto – 3.3V y nivel bajo – 0V.

La Figura 2.40 muestra el diseño final correspondiente al prototipo M2, descartando así la que se determinó en la fase de diseño. El cambio más significativo es el emplear un nuevo elemento gráfico para realizar el filtrado de los registros de acceso.

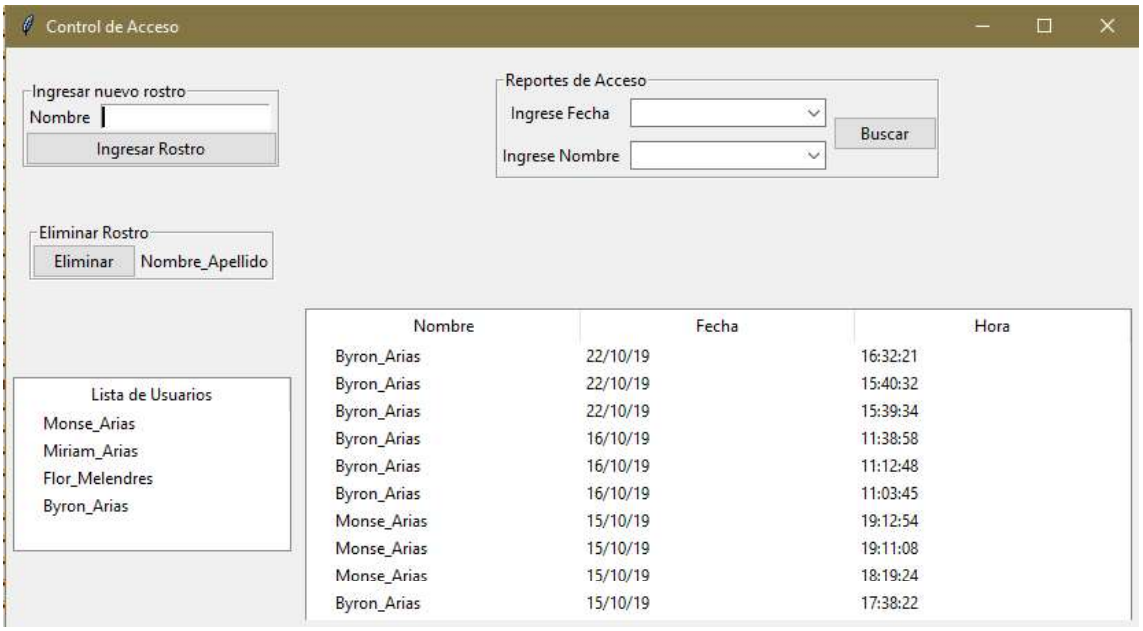


Figura 2.40. Interfaz gráfica del Sistema Prototipo de Control de Acceso

Para codificar la interfaz gráfica se emplea una librería que viene por defecto en todas las versiones de Python llamada *tkinter*; es una librería básica para el desarrollo de interfaces con este lenguaje y cuenta con los elementos gráficos más importantes y utilizados para poder diseñar y desarrollar una buena interfaz.

Como se observa en la Figura 2.40 existen tres partes importantes de la interfaz, a la izquierda en la parte superior están agrupados los componentes que permitirán el ingreso de nuevos rostros de una persona (*Ingresar nuevo rostro*), en la parte media el componente para eliminar los rostros de una persona que se seleccione de *Lista de Usuarios*, y toda la parte derecha corresponde al *Reporte de Acceso*.

A continuación, se indica el proceso para la creación de cada componente, empezando por la ventana principal, para esto el Código 2.13 indica la forma de hacerlo; cabe recordar que para la creación de esta interfaz se la ha definido como una clase.

```
14  class Principal:
15
16      def __init__(self, ventana):...
77
78      def obtenerUsuarios(self):...def obtenerReporte(self):...
101
102
103     def obtenerFiltro(self):...
114
115     # Funcion para llenar el combobox de los elementos para el fltrado
116     def llenarCombo(self):...
126
127     # Funcion que permite el ingreso de nuevos rostros
128     def ingresarRostro(self):...
151
152     def actualizarArchivo(self):...
158
159     # Codificacion de rostros almacenados
160     def codificarRostro(self):...
165
166     # Eliminacion de usuarios registrados
167     def eliminarUsuarios(self):...
181
182 # Inicio de la App
183 if __name__=='__main__':
184     ventana = Tk()
185     app = Principal(ventana)
186     ventana.mainloop()
```

Código 2.13. Clase *Principal* - Inicio de la interfaz gráfica

Las líneas entre la 14 y 167 corresponden a la clase *Principal*, contiene el código necesario para cada elemento gráfico o widget y los métodos que ayudan para que la interfaz funcione de forma correcta. Ya que se ha definido como una clase a la interfaz, en las líneas 183 a la 186 se instancia esta clase y se crea la ventana que contendrá a todos los elementos de la interfaz.

La libería *tkinter* está formada por clases, que corresponden a la definición de cada elemento gráfico con sus correspondientes métodos asociados [17]. En Python para trabajar con elementos gráficos, primero es necesario definir un contenedor para estos, denominado *frame,* el Código 2.14 indica la forma de crear los frames o contenedores para la interfaz. Para esto se utiliza la clase *LabelFame*, de *tkinter* (Ver Figura 2.41).

```
20      # Frames o contenedores
21      # Frame ingreso de rostros
22      frameIngreso = LabelFrame(self.wind, text = 'Ingresar nuevo rostro')
23      frameIngreso.grid(row = 0, column = 0, columnspan = 3, pady = 15, padx=5)
24      # Frame de reportes de acceso
25      frameRegistro = LabelFrame(self.wind, text = 'Reportes de Acceso')
26      frameRegistro.grid(row = 0, column = 5, columnspan = 4, pady = 15, padx=5)
27      # Frame eliminar rostros
28      frameEliminar = LabelFrame(self.wind, text = 'Eliminar Rostro')
29      frameEliminar.grid(row = 2, column = 0, columnspan = 3, pady = 15, padx=5)
```

Código 2.14. Fragmento 1 - Clase *Principal* - Codificación *LabelFrame*

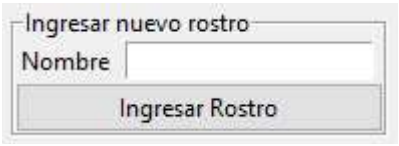


Figura 2.41. Ejemplo *frame*

La función *grid* se utiliza para ubicar un elemento gráfico o widget dentro de la ventana principal. Para la creación de los demás elementos gráficos las clases a utilizar son (Tabla 2.15):

Tabla 2.15. Clases utilizadas de la librería *tkinter*

Clase	Descripción
Label	Etiqueta: permite crear una etiqueta, esta puede almacenar un texto o una imagen.
Entry	Entrada: Crea una caja de texto, permite que el usuario pueda ingresar texto en dicho widget.
Button	Botón: crea los conocidos botones, para realizar una determinada acción.
Treeview	Árbol de vistas: crea una tabla.
Combobox	Esta clase crea un elemento que permite desplegar una lista de opciones previamente definidas.
messagebox	Caja de mensajes: son elementos que ayudan para mostrar advertencias al usuario de posibles errores o de información en general.

Luego de definir las clases que se utilizarán para crear los elementos gráficos, se procede a codificar cada widget, los Códigos 2.15, 2.16, 2.18 y 2.19 definidos a continuación muestran un ejemplo para crear cada uno de estos elementos.
También, en las Figuras 2.42, 2.43, 2.44, 2.45 y 2.46 se puede ver la forma gráfica de cada elemento o widget.

2.2.5.1. Codificación de etiquetas y cajas de texto

El Código 2.15 muestra la codificación de una etiqueta y una caja de texto. La Figura 2.42 es un ejemplo de los componentes gráficos.

```
31          # Igreso nombre
32          Label(frameIngreso, text = 'Nombre').grid(row = 1, column = 0)
33          self.nombre=Entry(frameIngreso)
34          self.nombre.focus()
35          self.nombre.grid(row=1,column=1,padx=5)
```

Código 2.15. Fragmento 2 - Clase *Principal* - Codificación *Label* y *Entry*

Figura 2.42. Ejemplo *Label* y *Entry*

2.2.5.2. Codificación de botones

La Figura 2.43 es un ejemplo de widget correspondiente a un botón y su codificación se puede observar en el Código 2.16.

```
47        # Boton de ingreso de rostros
48        ttk.Button(frameIngreso, text='Ingresar Rostro',command=self.ingresarRostro).grid(row=3,columnspan=2,sticky=W+E)
49        # Boton de filtrado
50        ttk.Button(frameRegistro, text='Buscar', command=self.obtenerFiltro).grid(row=0,column=9,rowspan=2,sticky=W+E)
51        # Boton de eliminar rostro
52        ttk.Button(frameEliminar, text='Eliminar', command=self.eliminarUsuarios).grid(row=2,column=0,sticky=W+E)
```

Código 2.16. Fragmento 3 - Clase *Principal* - Codificación *Button*

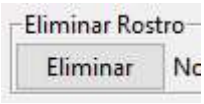


Figura 2.43. Ejemplo *Button*

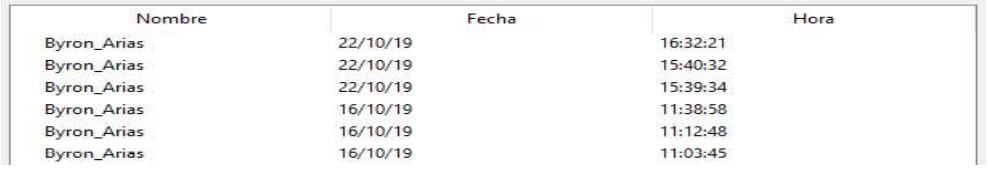
Los botones deben cumplir ciertas funciones al ser presionados, para esto se debe asociar cada botón con la función que se desea este realice, la propiedad *command*, que se observa en las líneas 48, 50 y 52 del Código 2.16, ayuda a asociar una función definida con el evento *click* del botón.

2.2.5.3. Codificación de tablas visuales

El Código 2.17 es un ejemplo de codificación de una tabla visual o treeview.

```
59        # Tabla Reporte
60        self.treeReporte = ttk.Treeview(height=10,columns=('fecha','hora'))
61        self.treeReporte.grid(row=3,column=5,padx=5,pady=5)
62        self.treeReporte.heading('#0',text='Nombre',anchor=CENTER)
63        self.treeReporte.heading('fecha',text='Fecha',anchor=CENTER)
64        self.treeReporte.heading('hora',text='Hora',anchor=CENTER)
```

Código 2.17. Fragmento 4 - Clase *Principal* - Codificación *Treeview*



Nombre	Fecha	Hora
Byron_Arias	22/10/19	16:32:21
Byron_Arias	22/10/19	15:40:32
Byron_Arias	22/10/19	15:39:34
Byron_Arias	16/10/19	11:38:58
Byron_Arias	16/10/19	11:12:48
Byron_Arias	16/10/19	11:03:45

Figura 2.44. Ejemplo *Treeview*

2.2.5.4. Codificación de cajas de texto desplegables

Las cajas desplegables o combobox se codifican de la forma como indica el Código 2.18 en las líneas 67 y 70. Un ejemplo de estos componentes gráficos se puede observar en la Figura 2.45.

```
66          # Combobox valores de filtros
67          self.comboNombre = ttk.Combobox(frameRegistro, state='readonly')
68          self.comboNombre.grid(row=1,column=6,padx=5,pady=5)
69
70          self.comboFecha = ttk.Combobox(frameRegistro, state='readonly')
71          self.comboFecha.grid(row=0,column=6,padx=5,pady=5)
```

Código 2.18. Fragmento 5 - Clase *Principal* - Codificación *Combobox*

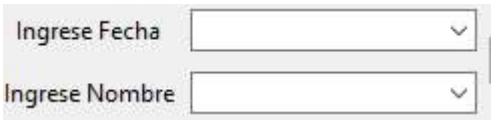


Figura 2.45. Ejemplo *Combobox*

2.2.5.5. Codificación de cajas de mensaje

La codificación de cajas de mensaje sirve para interactuar con el usuario e informarle sobre un posible error, el Código 2.19 define la forma de codificación de este widget, en la Figura 2.46 observamos un ejemplo de este componente gráfico.

```
146             else:
147                 messagebox.showinfo(message='Ingrese Nombre sin espacios en blanco', title='Aviso')
148         else:
149             print('Ingrese el nombre')
150             messagebox.showinfo(message='Debe Ingresar Nombre', title='Aviso')
```

Código 2.19. Fragmento 6 - Clase *Principal* - Codificación *messagebox*

Figura 2.46. Ejemplo *messagebox*

2.2.5.6. Codificación de funciones

Otro aspecto importante es la definición de las funciones que ayudan para complementar la funcionalidad de la interfaz gráfica, como se pudo observar en el Código 2.13 donde se definió a la clase *Principal*, se muestra también las funciones que esta contiene. A continuación, se analizan cada una de estas y el papel que desempeñan.

a. Función *__init__(self, ventana)*

Es la función principal de la interfaz, ya que aquí se codifica todos los widgets que formaran la interfaz. Esta función acepta dos parámetros que son: *self* y *ventana*. El parámetro *self* hace referencia para apuntar o referenciar a elementos que forman parte de la clase, todos los métodos o funciones que se definan en la clase *Principal* deberán especificar este como un parámetro. Por otro lado, *ventana* se refiera al argumento que recibe esta función y no es más que el contenedor principal, que es la ventana que se mostrara cada que se ejecute la interfaz, la Figura 2.40 es el ejemplo de el atributo *ventana*.

b. Función *obtenerUsuario(self)*

Esta función se asocia a una de las tablas definidas con la clase *Treeview*, llamada *Lista de Usuarios*, como se ve en la Figura 2.47. Lo único que realiza esta función es llenar con la información requerida esta tabla. El Código 2.20 corresponde al de esta función.

Figura 2.47. Tabla 1 - *Lista de Usuarios*

```
78      def obtenerUsuarios(self):
79          # Borrar tabla Eliminar
80          listaU = self.treeEliminar.get_children()
81          for elemento in listaU:
82              self.treeEliminar.delete(elemento)
83          self.actualizarArchivo()
84          with open('Usuarios.txt','r') as usuarios:
85              # llenar datos en la lista de usuarios
86              for line in usuarios:
87                  self.treeEliminar.insert('', 0, text=line)
88                  print(line)
```

Código 2.20. Fragmento 7 - Clase *Principal* - Función *obtenerUsuarios*

c. Función *obtenerReporte(self)*

Esta funciona es similar a la anterior, ya que se asocia a la segunda tabla correspondiente a *Reportes de Acceso*, esta tabla está conformada por tres columnas que son: *Nombre, Fecha y Hora*, como se puede observar en la Figura 2.48. Con esta función lo que se hace es llenar la información correspondiente a la bitácora de acceso.

El Código 2.21 corresponde a esta función, una particularidad de esta es la asociación con el módulo Registro, como se observa en la línea 97, con la ayuda de la función *registro* se obtiene los datos desde el archivo de bitácoras.

Nombre	Fecha	Hora
Byron_Arias	22/10/19	16:32:21
Byron_Arias	22/10/19	15:40:32
Byron_Arias	22/10/19	15:39:34
Byron_Arias	16/10/19	11:38:58
Byron_Arias	16/10/19	11:12:48
Byron_Arias	16/10/19	11:03:45
Monse_Arias	15/10/19	19:12:54
Monse_Arias	15/10/19	19:11:08
Monse_Arias	15/10/19	18:19:24
Byron_Arias	15/10/19	17:38:22

Figura 2.48. Tabla 2 - *Reporte de Acceso*

```
91      def obtenerReporte(self):
92          # Borrar tabla Reporte
93          listaR = self.treeReporte.get_children()
94          for elemento in listaR:
95              self.treeReporte.delete(elemento)
96
97          datos=Registro.registro()
98
99          for line in datos:
100             self.treeReporte.insert('', 0, text=line[0],values=(line[1],line[2]))
```

Código 2.21. Fragmento 8 - Clase *Principal* - Función *obtenerReporte*

d. Función *obtenerFiltro(self)*

Esta función se relaciona estrechamente con la anterior, ya que permite manipular la información que se muestra en la tabla de *Reporte de Acceso*, al aplicar un filtrado a la información que se presenta. En esta función también se involucra el módulo Registro, la información se obtiene a partir de las cajas de texto desplegables, y se envían como parámetros de la función *filtro* como se puede observar en las líneas 109, 110 y 110 del Código 2.22.

La Figura 2.49 es un ejemplo del funcionamiento de este método, empleando los componentes gráficos previamente definidos en la sección 2.2.5.4.

```
103     def obtenerFiltro(self):
104         # Borrar tabla Reporte
105         listaR = self.treeReporte.get_children()
106         for elemento in listaR:
107             self.treeReporte.delete(elemento)
108         #Cargar registro con fecha definida
109         nombreF=self.comboNombre.get()
110         fechaF=self.comboFecha.get()
111         listaFiltro=Registro.filtro(nombreF,fechaF)
112         for line in listaFiltro:
113             self.treeReporte.insert('', 0, text=line[0],values=(line[1],line[2]))
```

Código 2.22. Fragmento 9 - Clase *Principal* - Función *obtenerFiltro*

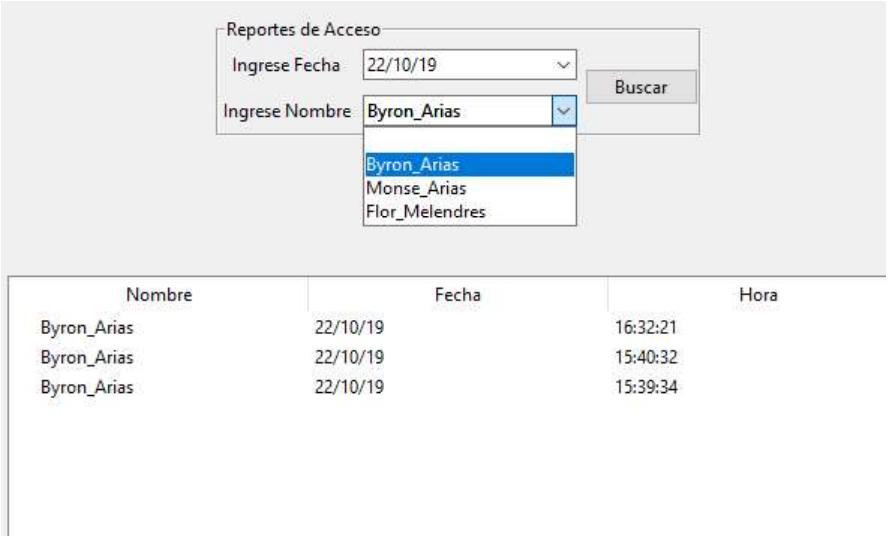


Figura 2.49. Ejemplo Función *obtenerFiltro*

La función *obtenerFiltro*, se relaciona con el botón *Buscar*, como se puede ver en la Figura 2.49, ya que, el evento *click* del componente está asociado a la función.

e. Función *llenarCombo(self)*

Esta función se encarga de cargar la información necesaria en las cajas de texto desplegables, también conocidas como *combobox*, al igual que las anteriores se relaciona con el módulo Registro, específicamente con la función *registro*, ya que es necesario obtener esta información para que el despliegue de información de los componentes corresponda a los datos que se encuentran en el archivo de registro de acceso. Este proceso se define en las líneas 119 a la 122 del Código 2.23.

```
115     # Funcion para llenar el combobox de los elementos para el fltrado
116     def llenarCombo(self):
117         listaN = ['']
118         listaF = ['']
119         datosC = Registro.registro()
120         for a in datosC:
121             listaF.append(a[1])
122             listaN.append(a[0])
123 
124         self.comboFecha['values']=list(set(listaF))
125         self.comboNombre['values']=list(set(listaN))
```

Código 2.23. Fragmento 10 - Clase *Principal* - Función *llenarCombo*

La información que se despliega en los *combobox* se verifica en la Figura 2.50.

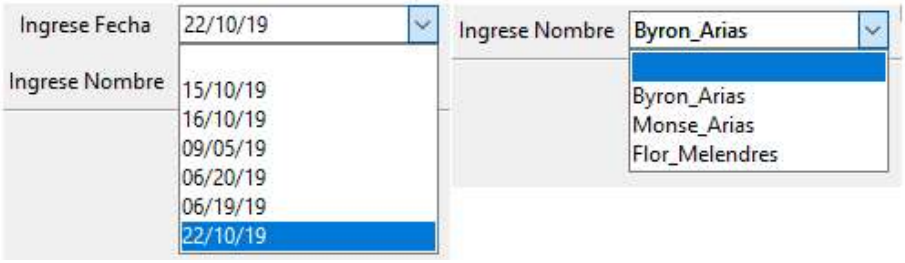


Figura 2.50. Ejemplo Función *llenarCombo*

f. Función *codificarRostro(self)*

Esta es una función auxiliar ya que se la emplea como mecanismo para utilizar la función *codRostros* del módulo Rostros. El Código 2.24 corresponde a esta función.

```
159     # Codificacion de rostros almacenados
160     def codificarRostro(self):
161         nombreRepositorio='repImagenes'
162         nombreArchivo='C:\\Users\\Byron Arias\\Desktop\\Compartida\\codImagen.pickle'
163 
164         Rostros.codRostros(nombreRepositorio,nombreArchivo)
```

Código 2.24. Fragmento 11 - Clase *Principal* - Función *codificarRostro*

g. Función *actualizarArchivo(self)*

Esta función se enfoca en la manipulación de un archivo de texto, que se emplea para obtener una lista de los usuarios que se han registrado sus respectivos rostros, como se explicó en la sección 2.1.4 estos se almacenan en un repositorio. La función extrae el nombre de todos los directorios que han sido creados, de acuerdo con los usuarios que se hayan registrado (Ver Código 2.25).

```
152     def actualizarArchivo(self):
153         listaRostros=os.listdir("repImagenes")
154         archivo = open('Usuarios.txt','w')
155         for nombre in listaRostros:
156             archivo.write(nombre+'\n')
157         archivo.close()
```

Código 2.25. Fragmento 12 - Clase Principal - Función *actualizarArchivo*

h. Función *ingresarRostro(self)*

Luego de codificar las dos funciones anteriores, se pude continuar con la función *ingresarRostros*, ya que son parte esencial de esta. Otra función que interviene es la de *buffer* del módulo Buffer, siendo la encargada de realizar el almacenamiento de los rostros de cada usuario. El Código 2.26 corresponde a la codificación del actual método. Las líneas 135 y 141 indican la forma en la que se activará la función *buffer* para almacenar nuevos rostros.

Esta función está asociada al botón *Ingresar Rostro* (ver Figura 2.51), además que la información del nombre asociado a los rostros que se van a almacenar se la extrae de la caja de texto que esta sobre el botón.

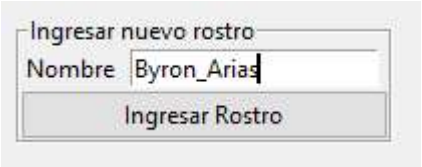


Figura 2.51. Wigets asociados a las Función *ingresarRostros*

```
127     # Funcion que permite el ingreso de nuevos rostros
128     def ingresarRostro(self):
129         detector = 'haarcascade_frontalface_default.xml'
130         nombreNuevo = self.nombre.get()
131         listaRostros=os.listdir("repImagenes")
132         if len(nombreNuevo)!=0:
133             if nombreNuevo.find(' ')==-1:
134                 if nombreNuevo in listaRostros:
135                     Buffer.buffer(detector, "repImagenes\\" + nombreNuevo)
136                     self.actualizarArchivo()
137                     self.obtenerUsuarios()
138                     self.codificarRostro()
139                 else:
140                     os.mkdir("repImagenes\\" + nombreNuevo)
141                     Buffer.buffer(detector, "repImagenes\\" + nombreNuevo)
142 
143                     self.actualizarArchivo()
144                     self.obtenerUsuarios()
145                     self.codificarRostro()
146             else:
147                 messagebox.showinfo(message='Ingrese Nombre sin espacios en blanco', title='Aviso')
148         else:
149             print('Ingrese el nombre')
150             messagebox.showinfo(message='Debe Ingresar Nombre', title='Aviso')
```

Código 2.26. Fragmento 13 - Clase *Principal* - Función *ingresarRostro*

i. Función *eliminarUsuarios(self)*

Esta función es la encargada de eliminar los rostros de usuarios que ya no tengan permitido el acceso al laboratorio, el Código 2.27 corresponde a esta función.

```
166     # Eliminacion de usuarios registrados
167     def eliminarUsuarios(self):
168         try:
169             self.treeEliminar.item(self.treeEliminar.selection())['text'][0]
170 
171         except IndexError as e:
172             messagebox.showinfo(message='Seleccione un nombre de la lista', title='Aviso')
173             return
174         nombreSeleccionado=self.treeEliminar.item(self.treeEliminar.selection())['text']
175         self.personaEliminar['text']=nombreSeleccionado
176         print(nombreSeleccionado.strip('\n'))
177         shutil.rmtree("repImagenes\\" + nombreSeleccionado.strip('\n'))
178         self.codificarRostro()
179         self.actualizarArchivo()
180         self.obtenerUsuarios()
```

Código 2.27. Fragmento 14 - Clase *Principal* - Función *eliminarUsuarios*

2.2.6. Despliegue de módulos, conexión física de dispositivos y conectividad

La parte final de la implementación del Sistema Prototipo está dividida en tres fases:

2.2.6.1. Fase de despliegue

Esta fase consiste en ubicar cada módulo en su correspondiente subsistema y dispositivo, tal como se determinó en la sección 2.1. Lo primero que se debe hacer es preparar todos los complementos necesarios para cada módulo.

a. Subsistema 1 – Despliegue de módulos

En este subsistema se encuentra el PC – Servidor y los módulos: Buffer, Registro y Rostros; también la interfaz gráfica para el programa principal en el servidor. La Figura 2.52 corresponde al directorio principal, donde se encuentra la aplicación, el script principal se lo ha llamado *TesisServer* y es el que contiene la codificación del programa principal.

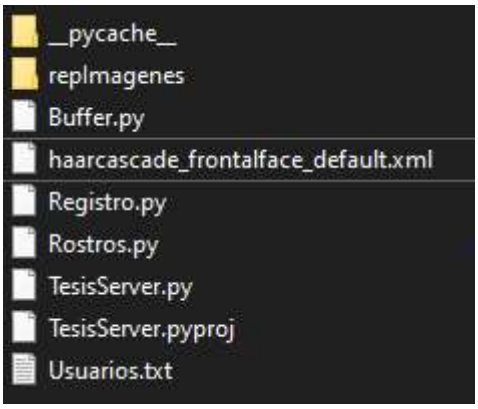


Figura 2.52. Directorio Principal de la aplicación

Lo elementos complementarios son:

- **Usuarios.txt:** es un archivo auxiliar que se emplea para almacenar los nombres de los usuarios registrados. (Ver Figura 2.53)
- **haarcascade_frontalface_default.xml:** es una plantilla que se emplea para el reconocimiento de rostros a partir de las funciones de Haar en Python.
- **replmagenes:** Es el repositorio de almacenamiento de rostros. (Ver Figura 2.53)

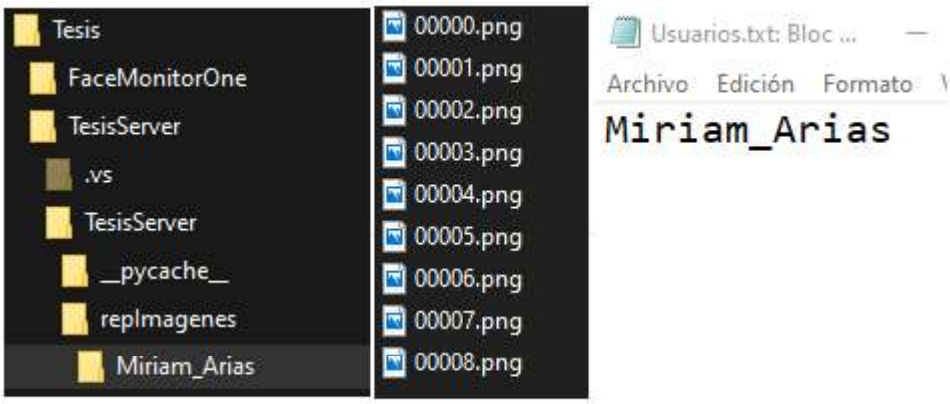


Figura 2.53. Repositorio de Rostros *replmagenes* - Archivo *Usuarios.txt*

Adicionalmente en el servidor se crea una carpeta compartida con la Raspberry, esta contendrá dos archivos: *codlamagen.pickle* y *Registro.xlsx*. Estos archivos son necesarios para ciertas funciones de los módulos correspondientes a la Raspberry, como se había explicado en la sección 2.2.4 y también para funciones de los módulos del servidor.

b. Subsistema 2 – Despliegue de módulos

La Raspberry forma parte de este subsistema y debe contener los módulos: Acceso y WebCam, para relacionar ambos módulos es necesario crear un script que controle la cerradura y actualice el archivo de bitácoras de acceso.

Se crea un script denominado *ControlAcceso* a través de este se integran los módulos antes mencionados, y su función es la de permitir a los usuarios interactuar con la Raspberry para acceder al laboratorio (Ver Código 2.28). Además, en este script se define un código extra de acceso, para casos en los que aún no existe ningún usuario registrado o en caso que se produzca una falla en el reconocimiento facial, para activar la cerradura.

ControlAcceso.py *

```
documento=openpyxl.load_workbook('Registro.xlsx')
while True:
    print("Ingrese el password:")
    password=input()
    if password == "1234":
        a, nombre = WebCam.webcam("codImagen.pickle","Server")

        if a==1:
#Modulo de control de cerradura
                print('Bienvenido')
                fecha=time.strftime("%x")
                hora=time.strftime("%X")
                print("El sujeto es: "+nombre)
                print(documento.sheetnames)
                hoja=documento.active
                hoja.append([nombre,fecha,hora])
                documento.save('Registro.xlsx')
                Acceso.acceso()

        else:
                print('Rostro no registrado')
    else:
        print("Password Incorrecto")
```

Código 2.28. Script *ControlAcceso.py*

Además, como en el PC se compartió una carpeta es necesario verla reflejada en la Raspberry, para esto se utiliza el protocolo CIFS (Common Internet File System) [41], este protocolo permite montar la carpeta compartida de Windows sobre una carpeta propia de otro sistema, en este caso Raspbian. Para esto se emplea el siguiente comando [42]:

sudo mount -t cifs //IP_PC-Servidor/NombreCarpetaCompartida /CarpetaRaspbian

En la Raspberry se crea la carpeta llamada *Server* y esta será la carpeta que reflejara la información del directorio compartido desde el PC – Servidor. Luego de ejecutar dicho

comando aparece una segunda carpeta en la ubicación de *Server*, como se observa en la Figura 2.54, y su contenido.

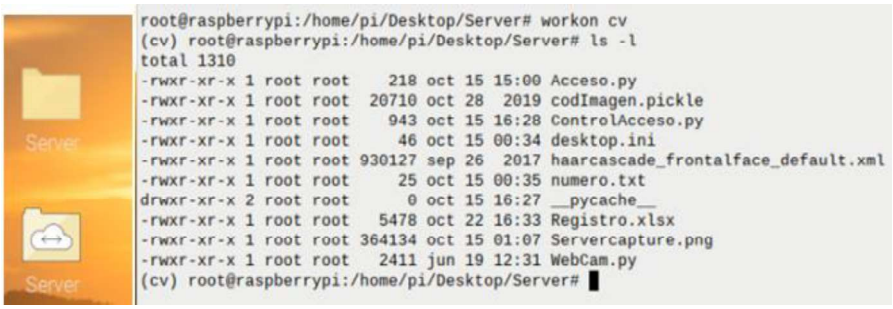


Figura 2.54. Carpeta montada desde Windows **–** Contenido *Server*

En la carpeta se encuentran tanto los módulos correspondientes a la Raspberry, el script antes creado y los archivos que se encontraban en la carpeta compartida del servidor, de esta forma se podrá tener la funcionalidad completa de este subsistema.

2.2.6.2. Fase de conexión

Esta fase se refiere a la integración de todos los elementos de hardware del prototipo, como se especifica en el diagrama de conexiones (Ver Figura 2.55). Primero se realiza la conexión de la cerradura electromagnética y sus elementos de protección (Subsistema 3), es recomendable realizar todas las conexiones antes de energizar los dispositivos.

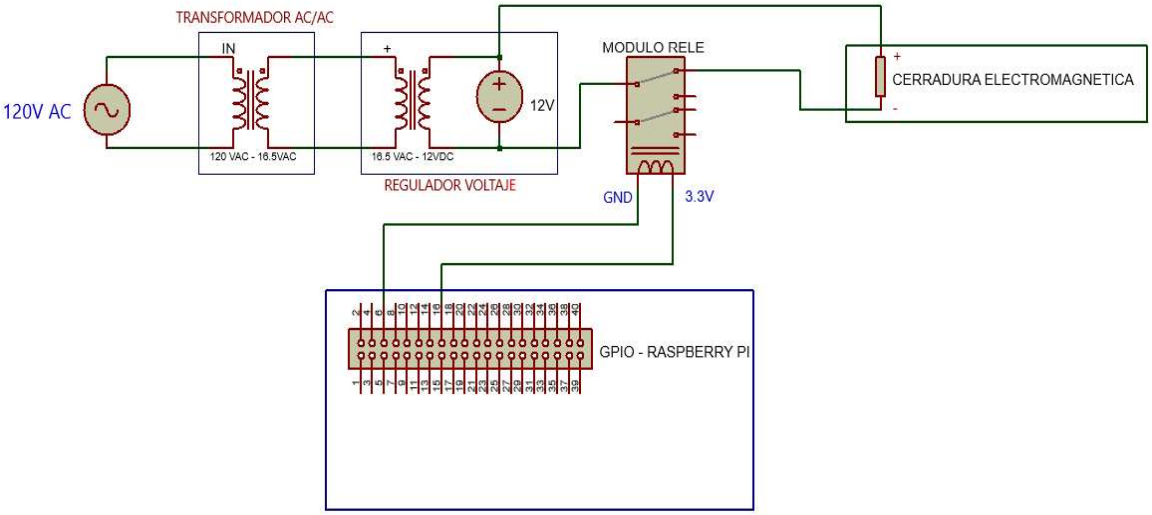


Figura 2.55. Diagrama de conexiones del prototipo

La Figura 2.56 concierne a la conexión del transformador AC/AC con la fuente/transformador AC/DC.

Figura 2.56. Conexión transformador - fuente de la cerradura

Luego se realiza la conexión de la Raspberry con el módulo de relé a través de los puertos GPIO (Subsistema 2), la distribución de pines se revisó en la sección 2.2.4.5, se utilizarán el pin 2 que genera un voltaje de 5V, necesario para energizar el relé, el pin 6 que es la conexión a tierra y el pin 16 de entrada/salida enviara la señal de control para activar el relé. Los pines de entrada del módulo relé son 4, dos corresponden a *Vcc* y *GND*, y los otros son los pines de entrada para la señal de control. La Figura 2.57 indica la conexión entre estos dispositivos.

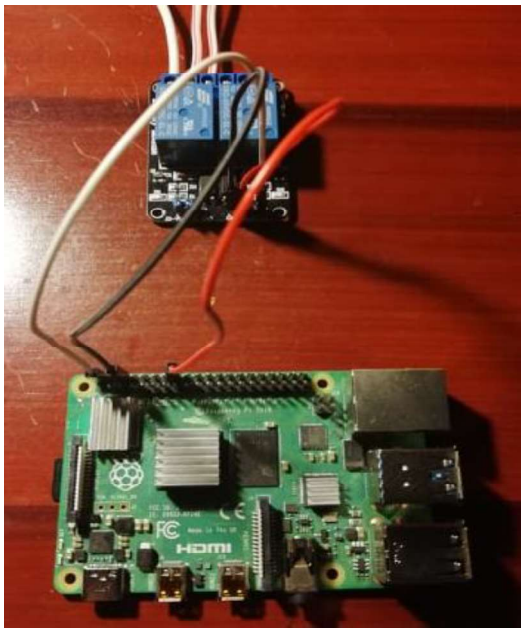


Figura 2.57. Conexión Raspberry - Modulo relé

Finalmente, se integra ambos subsistemas (2 y 3) su conexión es a través del relé. Un relé no es más que un dispositivo encargado de acoplar dos circuitos diferentes ya que actúa como una especie de interruptor, en este caso la señal emitida por la Raspberry accionará el relé y este conectará o desconectará el circuito de la cerradura.

Para el prototipo y debido a que la cerradura debe estar siempre activa, se conecta el relé de la forma normalmente cerrado, al accionar este dispositivo desde la Raspberry lo que provocara es cortar el paso de energía hacia la cerradura, por lo tanto, esta se abrirá. La Figura 2.58 indica la conexión de los subsistemas 2 y 3.

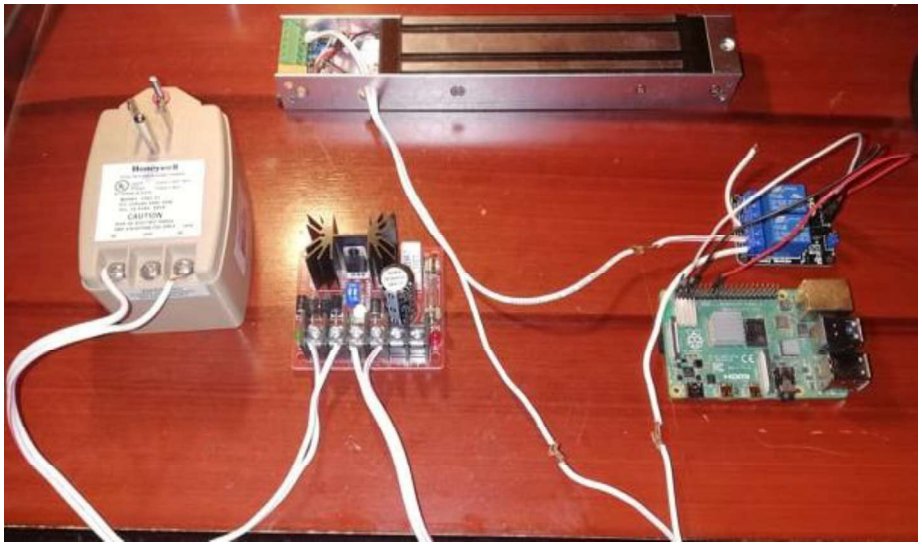


Figura 2.58. Conexión Subsistemas 2 y 3

2.2.6.3. Fase de conectividad

Esta fase consiste en establecer la comunicación entre PC y Raspberry, a través de un dispositivo de conectividad. Lo que se consigue en esta fase es la integración de todos los subsistemas que conforman el prototipo.

Es importante aclarar que se ha descrito cada fase, pero esto no significa que se haya seguido este orden, es recomendable, en primer lugar, establecer la comunicación entre el servidor y el cliente, debido a ciertos procesos que necesitan de la interacción de ambos.

Para establecer la comunicación se emplea un switch como se detalló en la fase 2.1.7, la característica *Plug and Play*[9] de este dispositivo evita el proceso de realizar una configuración específica, solo es necesario conectar el PC y la Raspberry. Sin embargo, es necesario especificar las direcciones IP de los dispositivos que se conecten al switch (Ver Figura 2.59).

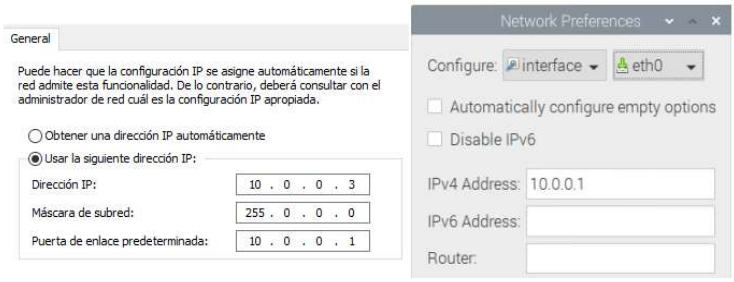


Figura 2.59. Configuración de IPs - PC y Raspberry

Para comprobar que se estableció la conexión, se prueba a través del comando *ping.* Como se observa en las Figuras 2.60 y 2.61.

```
C:\Users\Byron Arias>ping 10.0.0.1

Haciendo ping a 10.0.0.1 con 32 bytes de datos:
Respuesta desde 10.0.0.1: bytes=32 tiempo=2ms TTL=64
Respuesta desde 10.0.0.1: bytes=32 tiempo<1m TTL=64
Respuesta desde 10.0.0.1: bytes=32 tiempo=1ms TTL=64
Respuesta desde 10.0.0.1: bytes=32 tiempo<1m TTL=64

Estadísticas de ping para 10.0.0.1:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 0ms, Máximo = 2ms, Media = 0ms
```

Figura 2.60. Prueba conectividad - Windows a Raspbian

```
pi@raspberrypi:~ $ ping 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=128 time=0.344 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=128 time=0.301 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=128 time=0.289 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=128 time=0.304 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=128 time=0.227 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 167ms
rtt min/avg/max/mdev = 0.227/0.293/0.344/0.037 ms
```

Figura 2.61. Prueba conectividad - Raspbian a Windows

[9] Se refiere a una característica que permite conectarse a un equipo sin necesidad de una configuración previa, para empezar a utilizarlo.

3. RESULTADOS Y DISCUSIÓN

El presente capítulo se enfoca en el procedimiento de pruebas de funcionamiento del Sistema Prototipo y como se determinó en el plan previo al desarrollo de este Trabajo de Titulación, las pruebas se basan en verificar si a través del proceso de reconocimiento facial, se otorga o no acceso al Laboratorio de Comunicaciones Unificadas, para esto, en este capítulo se determina un protocolo de pruebas que permita comprobar el funcionamiento de este prototipo.

Es importante también verificar las demás funcionalidades del Sistema Prototipo, para esto, en la sección 3.1 previo a la determinación del protocolo de pruebas, se detalla el proceso que efectúan las demás funciones, para así corroborar el cumplimiento de los requerimientos definidos en la sección 2.1.

3.1. FUNCIONALIDAD Y PROTOCOLO DE PRUEBAS

3.1.1. Funcionalidad del Sistema Prototipo

En esta sección se verificarán las diferentes funciones del sistema prototipo, en base a los requerimientos determinados en la fase de diseño.

3.1.1.1. Subsistema 1 – Funcionalidad

El requerimiento RF7, descrito en la Tabla 3.1, verifica la funcionalidad de la interfaz gráfica desarrollada, como muestra la Figura 3.1. La interfaz permite el ingreso de rostros, eliminación de usuarios y la verificación de los diferentes accesos al laboratorio y que persona ingresó.

Tabla 3.1. Descripción RF7

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF7	1	La aplicación debe presentar una interfaz gráfica para su administración.

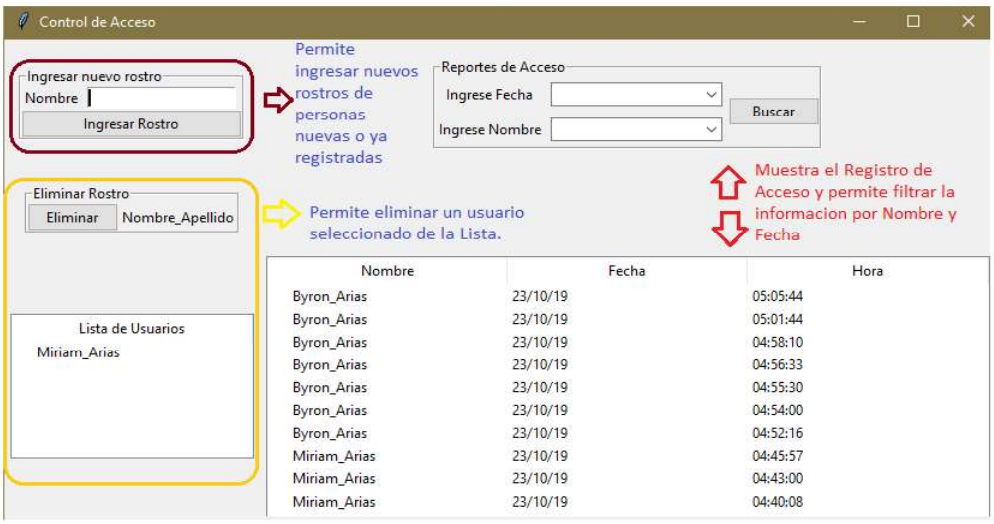


Figura 3.1. Descripción de las funcionalidades de la Interfaz Gráfica

Además, la Figura 3.2 indica los mensajes que se generan, para validar el ingreso de los campos requeridos.

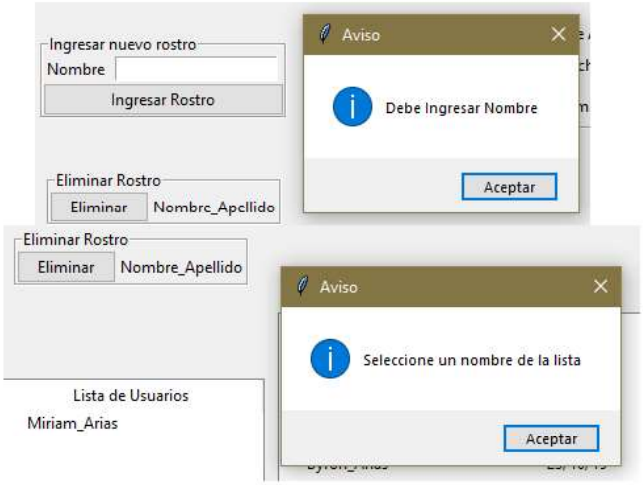


Figura 3.2. Validación de los campos: Nombre y Eliminar

El requerimiento RF2, descrito en la Tabla 3.2. La sección *Ingresar nuevo rostro* de la interfaz gráfica, se puede cumplir con este requerimiento, la Figura 3.3 muestra el desarrollo del proceso de ingreso de rostros a través de la aplicación.

Tabla 3.2. Descripción RF2

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF2	1	Permitir registrar nuevos rostros, de acuerdo con el nuevo personal responsable del laboratorio.

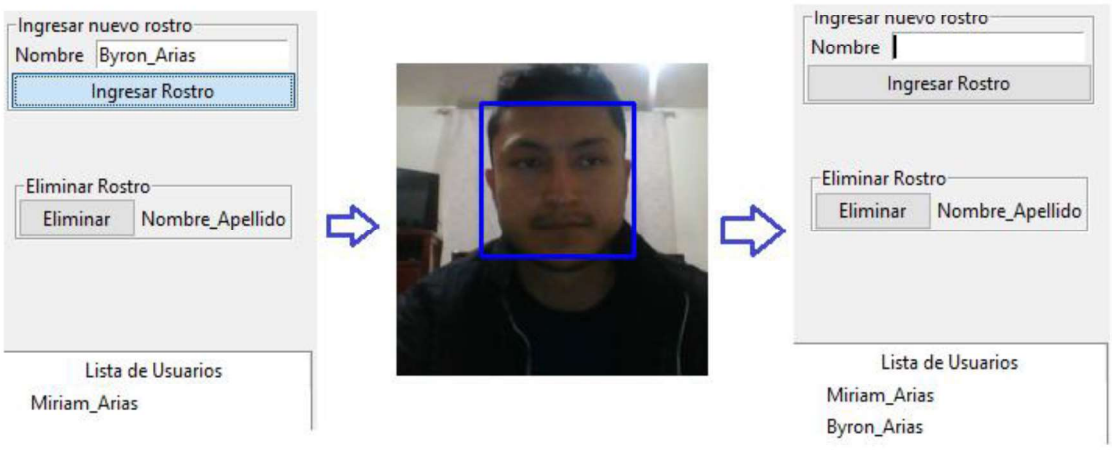


Figura 3.3. Proceso de ingreso de Rostros

El número de rostros que se desee guardar dependerá del administrador del laboratorio. Se recomienda un mínimo de 6 imágenes por persona ingresada, para mejorar la eficiencia de reconocimiento del algoritmo de reconocimiento facial.

El requerimiento RF3, descrito en la Tabla 3.3. En la sección *Eliminar Rostro*, de la interfaz gráfica se realiza el proceso de borrar rostros de las personas que ya no tenga autorización de ingresar al laboratorio, la Figura 3.4 muestra el proceso.

Tabla 3.3. Descripción RF3

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF3	1	Permitir la eliminación de rostros de personas registradas.

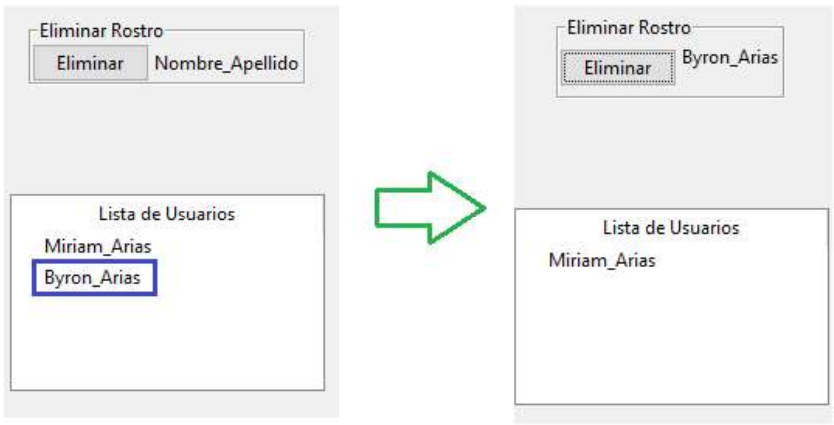


Figura 3.4. Proceso de Eliminación de Rostros

El requerimiento RF5, descrito en la Tabla 3.4. La interfaz gráfica permite verificar la hora de acceso, fecha y a la persona que haya ingresado, además de aplicar filtros para mejorar el proceso de búsqueda. La Figura 3.5 muestra un ejemplo de uso de los filtros para mostrar reportes.

Tabla 3.4. Descripción RF5

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF5	1	Verificación del personal que haya ingresado, mostrar reportes.

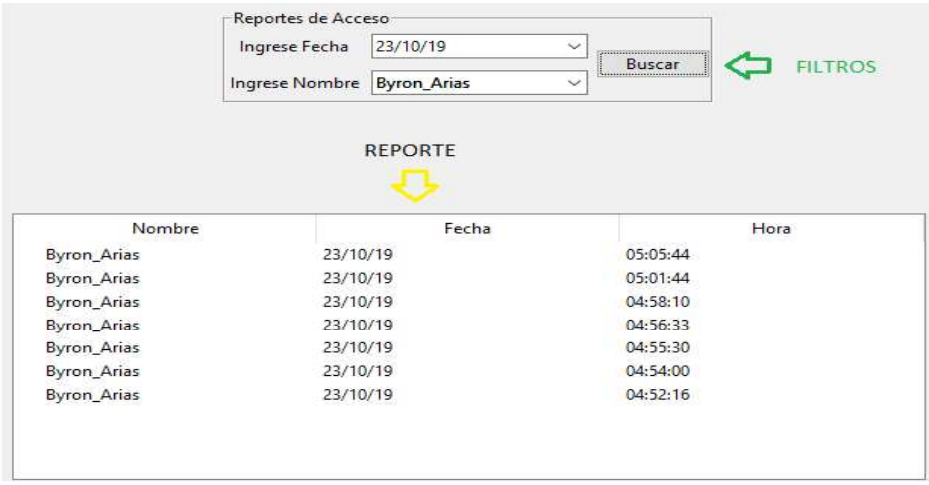


Figura 3.5. Ejemplo de filtrado - Reportes de Acceso

3.1.1.2. Subsistema 2 – Funcionalidad

El requerimiento RF4, descrito en la Tabla 3.5. Este proceso se lo realiza en la Raspberry, luego de que se haya generado un acceso exitoso al laboratorio, se guarda *Nombre, fecha y hora* en el archivo de *Registro.*

La Figura 3.6 muestra un acceso exitoso, y como se refleja en la interfaz en la tabla de *Reportes de Acceso* el nuevo ingreso que se ha dado.

Tabla 3.5. Descripción RF4

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF4	2	Guardar un registro de las personas que hayan accedido al laboratorio.

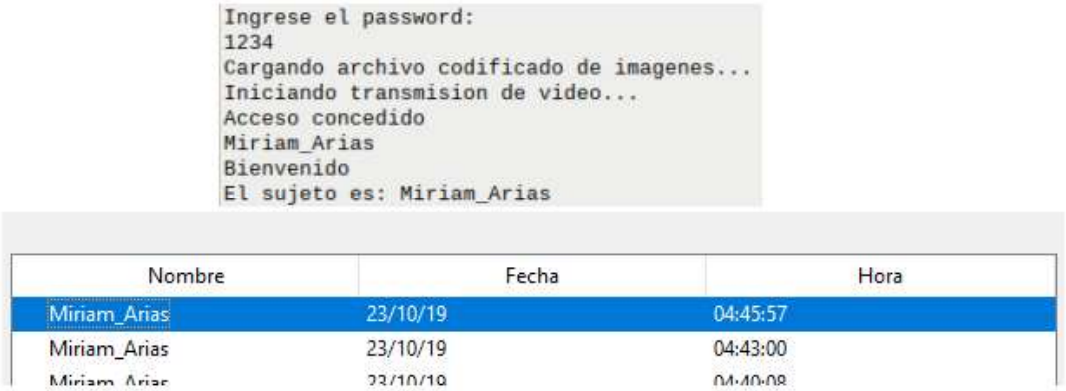


Figura 3.6. Ejemplo de ingreso exitoso al laboratorio

El requerimiento RF6, es uno de los principales requerimientos (Ver Tabla 3.6), ya que el sistema prototipo basa su funcionamiento en la detección y reconocimiento facial, para comprobar su funcionalidad, la Figura 3.7 muestra la imagen que se captura desde la cámara y se procesa en la Raspberry para otorgar o no acceso, el recuadro alrededor del rostro indica que ha sido detectado y reconocido.

Tabla 3.6. Descripción RF6

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF6	2	La aplicación debe detectar y reconocer los rostros registrados.

Figura 3.7. Ejemplo de Detección y Reconocimiento de rostros

Los requerimientos RF1 y RF8, son de los más importantes que debe cumplir el prototipo ya que, son los encargados de realizar el proceso de reconocimiento y activación de la cerradura para así cumplir con el objetivo principal del prototipo. Para esto se definirá el protocolo de pruebas en la sección 3.1.2 y se comprobará su funcionamiento. Estos requerimientos son descritos en la Tabla 3.7.

Tabla 3.7. Descripción RF1 y RF8

CÓDIGO	SUBSISTEMA	DESCRIPCIÓN
RF1	2	Controlar y permitir acceso al laboratorio al personal registrado.
RF8	2	La cerradura debe ser controlada por las funciones del software.

3.1.2. Protocolo de Pruebas – Control de Acceso

La definición de este protocolo depende de los parámetros planteados en la Tabla 1.2, que son: *Iluminación, Distancia y Excepciones*.

Iluminación, se refiere a la variación de la intensidad de luz al momento de probar la detección y reconocimiento de rostros, ya que, en condiciones reales, los niveles de iluminación de un lugar son variables.

Distancia, es necesario variar la distancia desde donde se ubica el lente de la cámara hasta el rostro de la persona para determinar una distancia efectiva, donde el porcentaje de reconocimiento sea alto, para las pruebas se estima una distancia máxima de 1 metro, ya que se desea enfocar un solo rostro para el control de acceso.

Excepciones, es un parámetro arbitrario, se considera factores que dependen de la persona que se pondrá frente a la cámara y si su aspecto cambia por el uso de prendas de vestir o accesorios que dificulten el reconocimiento, se consideró tres casos:

- Uso de anteojos
- Uso de gafas de sol
- Uso de gorras

Una vez descritos estos parámetros, se plantea el siguiente protocolo de pruebas para el Sistema Prototipo de Control de Acceso.

El protocolo va a constar de dos partes, correlacionadas, la primera tomará en cuenta los parámetros de *Iluminación y Distancia*; la segunda el de *Excepciones*, con la particularidad de que a partir de los mejores resultados de la primera parte se realizaran las pruebas para este.

3.1.2.1. Protocolo – Parte Uno

La Tabla 3.8 define parámetros, niveles y especificaciones para la primera parte del protocolo de pruebas. En este caso se evaluará el nivel de reconocimiento tomando en cuenta los parámetros de iluminación y distancia. Además, se definen otras especificaciones para el protocolo, como son: número de personas, imágenes por persona, capturas que se realizarán por persona. Los valores asociados han sido determinados en base al estudio de diferentes proyectos realizados en los que intervienen procesos de reconocimiento facial [5], [11], [43], [44].

Tabla 3.8. Protocolo - Parte 1

Parámetro 1	Iluminación
Niveles	▪ Alto ▪ Medio ▪ Bajo
Parámetro 2	Distancia
Niveles	▪ 0.25 m ▪ 0.50 m ▪ 1.0 m
Número de Personas	Mínimo: 2
Géneros	▪ Masculino (1) ▪ Femenino (1)
Imágenes/persona:	Mínimo: 5 Máximo: 15
Capturas/persona:	Mínimo: 2 Máximo: 10

Directrices:

- Ambos parámetros se relacionan, las pruebas consisten que con cada nivel de iluminación se realice un determinado número de capturas de acuerdo con cada nivel de distancia.
- Solo existen dos valores de respuesta: *check (✓)* o *fail (X)*.
- A partir de esto se determinará la mejor distancia y el mejor nivel de iluminación en que se obtuvo el mejor reconocimiento de rostros.
- La Tabla 3.9 especifica la plantilla de pruebas, con 5 capturas por persona.

Tabla 3.9. Plantilla de Pruebas - Protocolo Parte 1

Rostro ##	**Distancia**														
Iluminación	**0,25 m**					**0,50 m**					**1,0 m**				
Alto															
Medio															
Bajo															

3.1.2.2. Protocolo – Parte Dos

La Tabla 3.10 define la segunda parte del protocolo, donde se evaluará el nivel de reconocimiento a partir de casos excepcionales, para este caso es necesario que se defina en la primera parte del protocolo los mejores niveles de iluminación y distancia.

Tabla 3.10. Protocolo - Parte 2

Parámetro	Excepciones
Casos	▪ Anteojos ▪ Gafas de sol ▪ Gorra
Nivel de Iluminación	Por definir (Parte Uno)
Nivel de Distancia	Por definir (Parte Uno)
Capturas/persona:	Mínimo: 2 Máximo: 10

Directrices:

- Se realiza esta prueba para determinar casos especiales, y determinar limitaciones al momento de reconocer un rostro.
- Es necesario determinar en la Parte Uno, el mejor nivel de iluminación y distancia de reconocimiento, para emplearlo en este caso.
- Solo existen dos valores de respuesta: *check (✓)* o *fail (X)*.
- La Tabla 3.11 especifica la platilla de pruebas, para 5 capturas por persona.

Tabla 3.11. Plantilla de Pruebas - Protocolo Parte 2

Casos	**Rostro 1**					**Rostro 2**				
Anteojos										
Gafas										
Gorra										

3.2. PRUEBAS DE FUNCIONAMIENTO

Luego de definir el protocolo, es necesario verificar los usuarios registrados y la cantidad de rostros que se han almacenado por cada usuario. Desde el PC – Servidor, verificamos los usuarios que existen, la Figura 3.8 indica la lista de usuarios registrados.

Figura 3.8. Lista de Usuarios

Como determina el protocolo, son dos usuarios de diferente género. Cumpliendo esa condición, también es necesario verificar cuantos rostros existen por cada uno de ellos, para hacerlo, ingresamos al repositorio de imágenes *repImagenes*. Como se observa en la Figura 3.9 por cada directorio, existen almacenadas 9 imágenes.

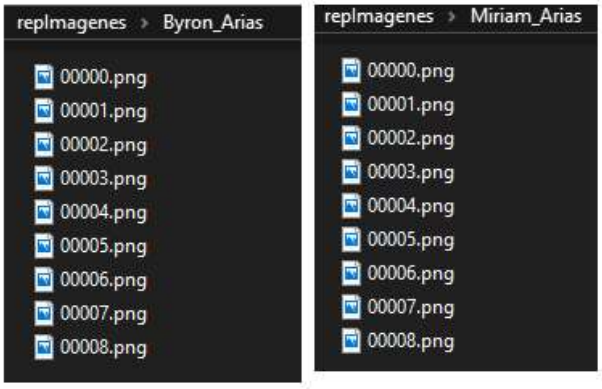


Figura 3.9. Repositorios de Imágenes

Después de verificar que se cumple con las condiciones necesarias para aplicar el protocolo de pruebas, ingresamos a la Raspberry y ejecutamos el script *ControlAcceso*, para empezar con las pruebas de acuerdo lo establecido en el protocolo (Ver Figura 3.10).

```
(cv) root@raspberrypi:/home/pi/Desktop/Server# python ControlAcceso.py
Ingrese el password:
1234
Cargando archivo codificado de imagenes...
Iniciando transmision de video...
```

Figura 3.10. Ejecución de *ControlAcceso.py*

Para activar el proceso de reconocimiento facial es necesario ingresar una contraseña, esto activará la cámara para capturar el rostro de la persona que desea ingresar. Este proceso se repetirá para cada una de las pruebas que se realicen.

3.2.1. Prueba Uno

Los datos del sujeto de prueba se presentan en la Tabla 3.12.

Tabla 3.12. Datos Sujeto 1

Usuario	Miriam_Arias
Género	Femenino
Imágenes/Usuario	9
Capturas por realizar	5
Denominación	Rostro 1

La primera ronda de capturas se las realiza con un nivel de iluminación Alto. La Figura 3.11 muestra algunas de las capturas realizadas.

Figura 3.11. Capturas Iluminación Alta - (A) 0.25 m - (B) 0.50 m - (C) 1.0 m

La segunda ronda se realiza con un nivel de iluminación Medio. Un ejemplo de capturas realizadas se puede observar en la Figura 3.12.

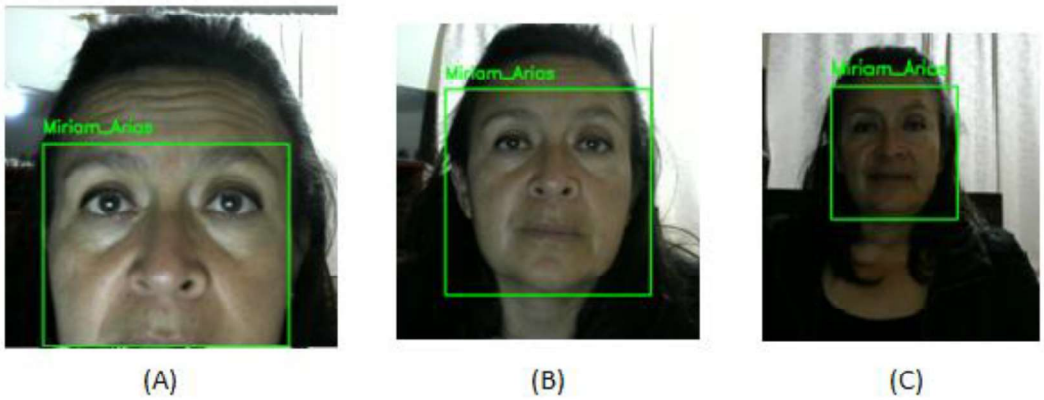


Figura 3.12. Capturas Iluminación Media – (A) 0.25 m – (B) 0.50 m – (C) 1.0 m

La ronda final para el Rostro 1, se realiza con un nivel Bajo de iluminación. Algunas capturas se muestran en la Figura 3.13.

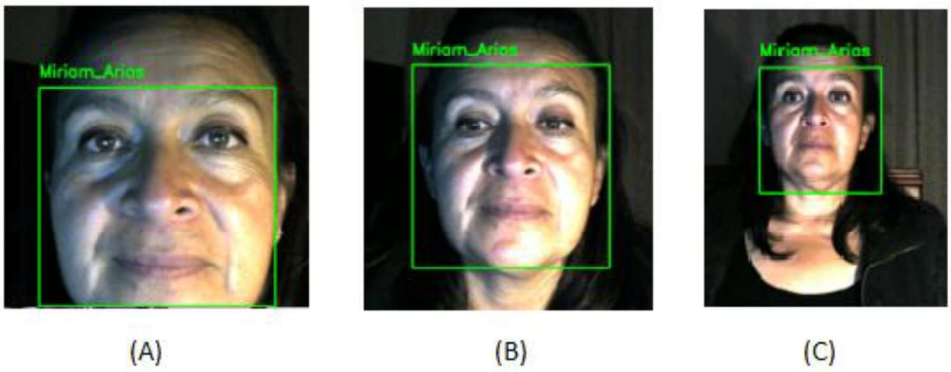


Figura 3.13. Capturas Iluminación Baja – (A) 0.25 m – (B) 0.50 m – (C) 1.0 m

Una vez realizadas todas las capturas, la plantilla de pruebas correspondiente a la Parte Uno del protocolo, para el Rostro 1, queda de la siguiente manera (Tabla 3.13):

Tabla 3.13. Parte Uno - Resultados Rostro 1

Rostro 1	**Distancia**														
Iluminación	**0,25 m**					**0,50 m**					**1,0 m**				
Alto	✓	✓	X	✓	✓	✓	✓	✓	✓	✓	X	✓	✓	✓	✓
Medio	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	X
Bajo	✓	✓	✓	✓	X	✓	✓	✓	✓	✓	✓	✓	X	✓	✓

La Tabla 3.12 muestra los resultados obtenidos, en los que se puede inferir que la mejor distancia es la de 0.50 m con un nivel de iluminación Medio.

3.2.2. Prueba Dos

Los datos del sujeto de prueba se presentan en la Tabla 3.14

Tabla 3.14. Datos Sujeto 2

Usuario	Byron_Arias
Género	Masculino
Imágenes/Usuario	9
Capturas por realizar	5
Denominación	Rostro 2

El proceso es el mismo que en la prueba anterior, se presentan los resultados obtenidos en la Tabla 3.15 y en la Figura 3.14 ejemplos de las capturas realizadas.

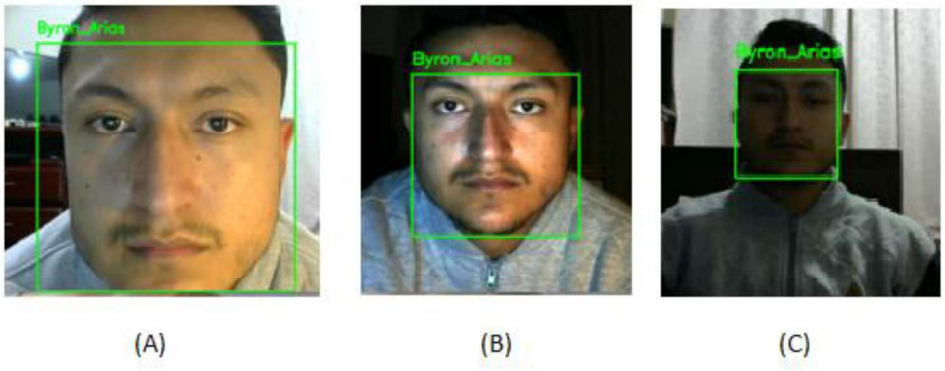


Figura 3.14. Capturas - (A) Ilum: A / Dist: 0.25 m - (B) Ilum: B / Dist: 0.50 m / (C) Ilum: M / Dist: 1.0 m

Tabla 3.15. Parte Uno - Resultados Rostro 2

Rostro 2	**Distancia**														
Iluminación	**0,25 m**					**0,50 m**					**1,0 m**				
Alto	X	✓	✓	X	✓	✓	✓	✓	✓	✓	✓	X	✓	✓	✓
Medio	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	X
Bajo	X	✓	X	✓	✓	✓	✓	✓	X	✓	✓	✓	✓	✓	✓

La Tabla 3.15 muestra los resultados obtenidos, en los que se puede inferir que la mejor distancia es la de 0.50 m y el mejor nivel de iluminación Medio ya que presenta el menor fallo.

3.2.3. Prueba Tres

Esta prueba consiste en aplicar la segunda parte del protocolo de pruebas, con las pruebas anteriores se determinó que la mejor distancia es la de 0.50 m, y el mejor nivel de iluminación es el Medio.

La Figura 3.15 presenta ejemplos de capturas realizadas que no presentaron errores en los casos de uso de anteojos y gorras, sin embargo, la Figura 3.16 muestra las capturas con el índice más alto de error en el caso de usar gafas de sol. Las capturas corresponden al Rostro 1.

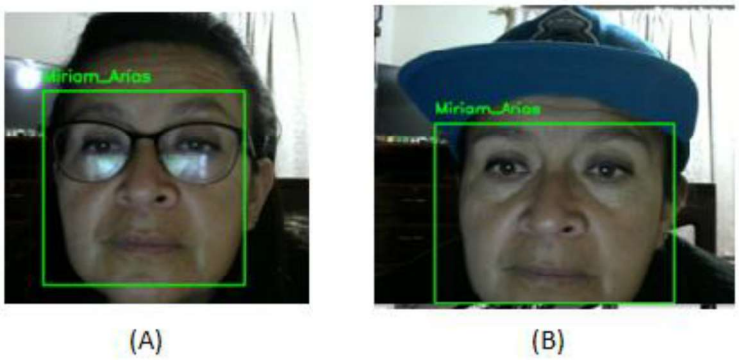


Figura 3.15. Capturas Rostro 1 - (A) Anteojos - (B) Gorra

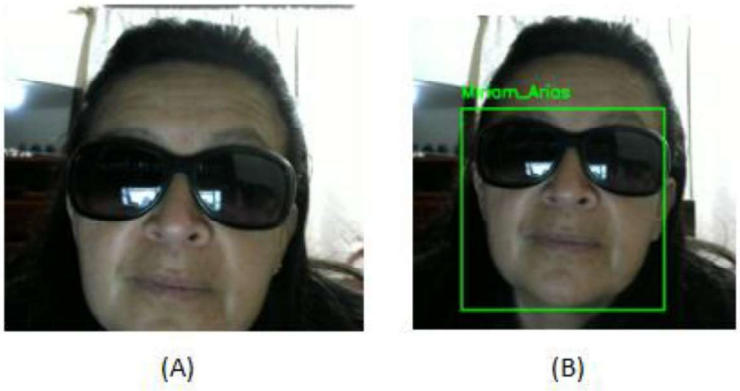


Figura 3.16. Capturas Rostro 1 - (A) Gafas Error - (B) Gafas Acierto

Para el Rostro 2 se realizaron las mismas capturas, en este caso fue diferente ya que el mayor índice de error se presentó en el caso de uso de gorras, la Figura 3.17 presenta un ejemplo de las capturas realizadas. La Figura 3.18 por otro lado, muestra las capturas de mayor acierto en los casos de uso de anteojos y gafas de sol.

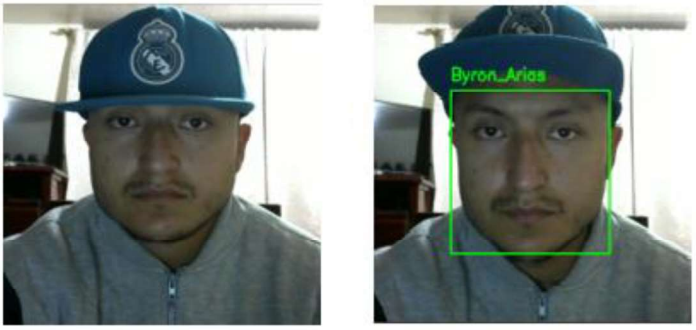


Figura 3.17. Capturas Rostro 2 - (A) Gorra Error - (B) Gorra Acierto

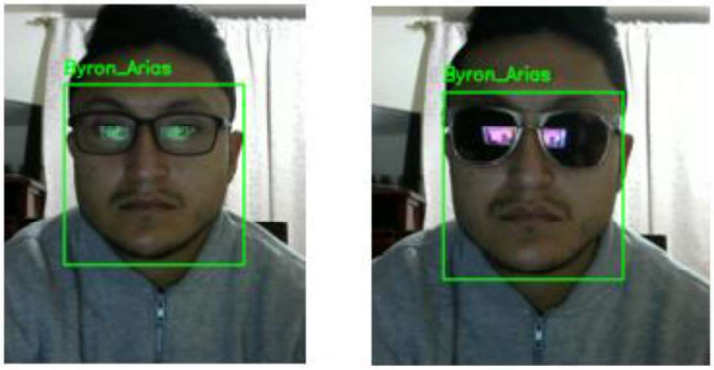


Figura 3.18. Capturas Rostro 2 - (A) Anteojos - (B) Gafas

La Tabla 3.16 muestra los resultados obtenidos para los casos especiales analizados, en este caso se pudo evidenciar que los errores dependen de distintos factores, los mismos que serán analizados en la siguiente sección.

Tabla 3.16. Parte Dos - Resultados Prueba 3

Casos	**Rostro 1**					**Rostro 2**				
Anteojos	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Gafas	X	✓	X	X	✓	✓	✓	✓	✓	X
Gorra	✓	✓	✓	X	✓	✓	✓	✓	X	X

3.3. RESULTADOS Y ANÁLISIS

Las Tablas 3.13, 3.15 y 3.16 muestran los resultados después de haber realizado las pruebas de acuerdo con lo establecido en el protocolo definido en la sección 3.1.2. En esta sección se realizará una tabulación de los resultados, para proceder a analizarlos. Las Tablas 3.17, 3.18 y 3.19 muestran los resultados tabulados, de las pruebas realizadas de acuerdo con la Parte Uno del protocolo de pruebas.

Tabla 3.17. Resultados Generales Tabulados - Pruebas Uno y Dos

Rostro	# Capturas	Aciertos	Errores	% Acierto
1	45	40	5	89 %
2	45	38	7	84 %

De acuerdo con estos resultados, el porcentaje de acierto en promedio es de un 87% lo que es realmente alto, esto quiere decir que el prototipo tiene un alto nivel de reconocimiento de rostros. Sin embargo, si existe variación en el porcentaje de acierto al momento de variar el nivel de iluminación y la distancia.

Tabla 3.18. Resultados Tabulados- Nivel Iluminación - Pruebas Uno y Dos

Nivel Iluminación	# Capturas	Aciertos	Errores	% Acierto
Alto	30	25	5	83 %
Medio	30	28	2	94 %
Bajo	30	25	5	83 %

Tabla 3.19. Resultados Tabulados - Distancia - Pruebas Uno y Dos

Distancia	# Capturas	Aciertos	Errores	% Acierto
0.25 m	30	24	6	80 %
0.50 m	30	29	1	97 %
1.0 m	30	25	5	83 %

El nivel de iluminación óptimo para mejorar el reconocimiento de rostros debe ser el Medio, de acuerdo con los resultados de la Tabla 3.18. También la mejor distancia a la que se puede reconocer un rostro con una alta efectividad debería estar a 0.5 m.

Considerando los casos especiales, se obtuvieron los siguientes datos (Tabla 3.20):

Tabla 3.20. Tabulación Resultados - Prueba Tres

Casos	# Capturas	Aciertos	Errores	% Acierto
Anteojos	10	10	0	100 %
Gafas	10	7	3	70 %
Gorras	10	7	3	70 %

De acuerdo con los resultados obtenidos, se puede decir que no afecta en nada la efectividad del reconocimiento de rostros cuando la persona utiliza anteojos. Sin embargo, en el caso de que la persona utilice gafas de sol, el porcentaje de acierto es muy bajo, esto es entendible ya que, dependiendo de la forma y tamaño de las gafas estas pueden llegar a cubrir tanto los ojos, cejas y una buena parte del rostro lo que dificultaría la detección.

En el caso de que la persona utilice gorras, también afecta en gran porcentaje la efectividad del reconocimiento, es decir en ciertos casos no será posible reconocer el rostro de las personas que utilicen gorras, además que, también depende del tipo de gorra y la forma en la que la persona la utilice.

3.4. OBSERVACIONES

En esta sección se explica una particularidad que se presenta al momento de realizar el proceso de reconocimiento facial en el prototipo, ya que se tomó como base para codificar los módulos la librería *face_recognition*. La referencia [36] define que la efectividad y confiabilidad de las funciones de esta librería depende del modo de detección que se utilice, como se analizó en la sección 2.2.4 se definen dos métodos: *hog* y *cnn*.

Las diferencias entre estos métodos son varias, pero específicamente las principales diferencias de acuerdo con [36] son:

- *hog* no es menos preciso en el proceso de reconocimiento, pero más rápido al momento de procesar una imagen mientras que, *cnn* es más preciso en el proceso de reconocimiento, pero mucho más lento.
- *hog* no requiere mucho procesamiento y recursos del dispositivo donde se esté ejecutando, en cambio *cnn* requiere de mucho procesamiento y recursos por lo que se recomienda ejecutarlo desde un servidor con buenas características de procesamiento y gráficas.

Lo que se quiere demostrar o evidenciar es que, para este prototipo se ha trabajado con el método de detección *hog*, por las razones ya definidas en las secciones 2.1 y 2.2. Es decir, al momento de reconocer un rostro será menos preciso y confiable, no en todos los casos, sino en casos en los que los rasgos de las personas sean muy similares.

Se realizó una prueba adicional para demostrar este caso especial, a partir de los rostros utilizados en la sección 3.2, se escogió imágenes de seis personas que tienen rasgos similares a los denominados: *Rostro 1 y Rostro 2*; como se observa en la Figura 3.19, donde se puede observar los casos exitosos y erróneos.

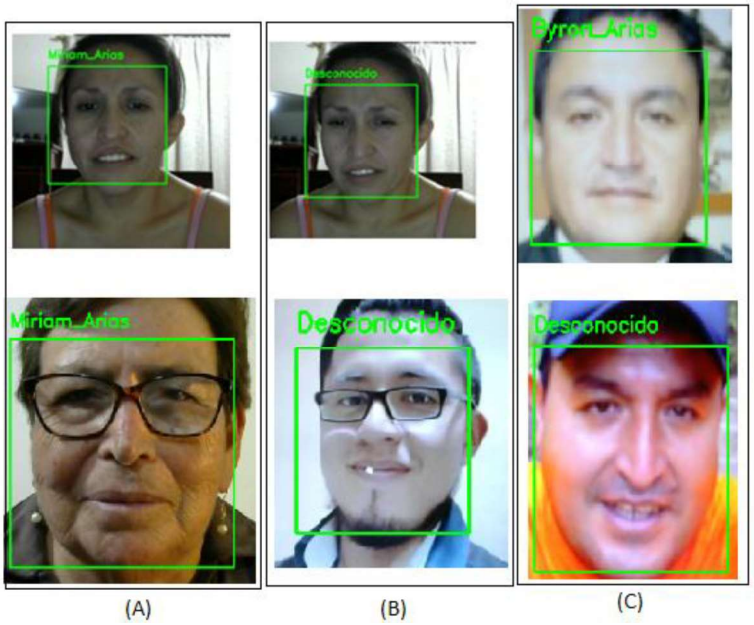


Figura 3.19. (A) Error - (B) Acierto – (C) Error/Acierto

Se realizaron pruebas con estos nuevos rostros, tomando en cuenta los resultados obtenidos previamente, capturando imágenes a una distancia entre 0.50 – 0.60 metros con un nivel de iluminación Medio. Se produjo un error recurrente, en ciertas capturas no reconocía el rostro y en otros sí, pero en otros casos mejoro la confiabilidad de reconocimiento, como se puede ver en la Figura 3.20.

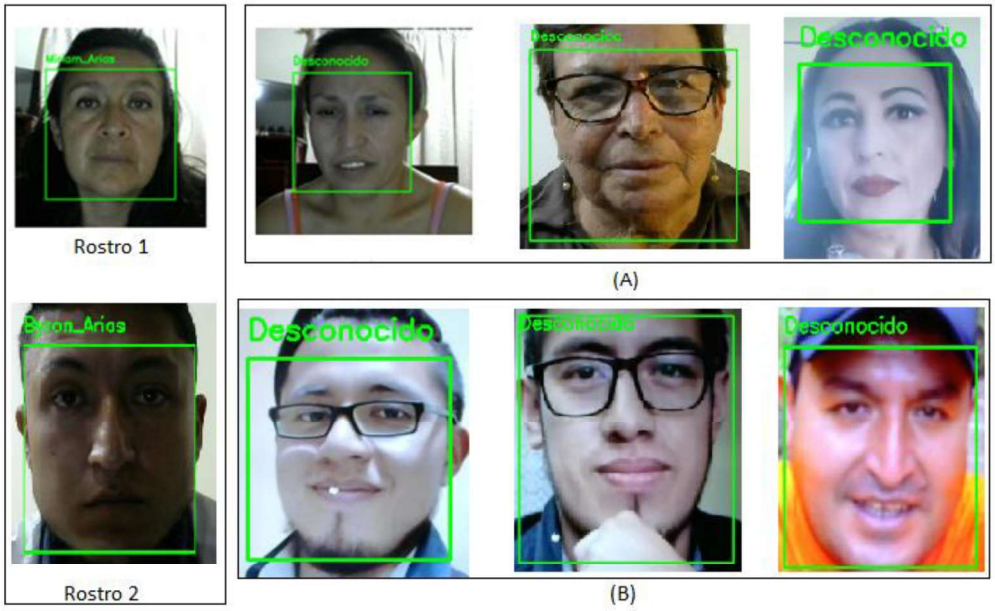


Figura 3.20. (A) Aciertos/Rostro 1 – (B) Aciertos/Rostro2

Una forma de corregir este error es, al momento de crear el repositorio de imágenes, se debe ingresar un numero alto de rostros por usuario alrededor de 25 a 30 imágenes por cada uno, este fue el factor que permitió mejorar la confiabilidad del prototipo durante esta prueba, ya que se recopilaron 30 imágenes por cada rostro (*Rostro 1 y Rostro 2*). La otra y más efectiva es emplear el método *cnn* [35], [36].

En el caso del Sistema Prototipo de Control de Acceso, es imposible utilizar el método *cnn* ya que la Raspeberry no dispone de los recursos necesarios para poder soportar este método. Entonces, es recomendable que se ingrese un número mayor de rostros por usuario para disminuir la ocurrencia de este error. La Tabla 3.21 muestra los resultados de esta prueba adicional, en el que se mide el nivel de confiabilidad del prototipo.

Tabla 3.21. Resultados Prueba Adicional - Nivel de Confiabilidad

Rostro	Referencia	# Capturas	Aciertos	Errores	% Acierto
A	Rostro 1	10	8	2	80 %
B		10	9	1	90 %
C		10	7	3	70 %
D	Rostro 2	10	6	4	60 %
E		10	9	1	90 %
F		10	8	2	80 %

En base a las pruebas realizadas, se puede determinar que el nivel de confiabilidad es alto, dando un promedio de un 78% de acierto al no asociar los rostros de estas personas con los de las ya registradas. Sin embargo, es evidente que existe un alto porcentaje de que el sistema pueda errar al momento de reconocer un rostro con características similares a la de una persona que este registrada en el sistema.

Se consideró importante realizar esta observación ya que forma parte de las limitaciones que presenta este prototipo. Sin embargo, para el caso puntual para personas con rasgos similares se considera necesario mencionar esta limitación debido a que influye directamente en el nivel confiabilidad del reconocimiento facial que emplea el prototipo. A pesar de la alta confiabilidad este factor contribuye a que se puedan generar errores al momento de la utilización del prototipo.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- En el presente Proyecto de Titulación se desarrolló un sistema prototipo de control de acceso, que emplea reconocimiento facial para controlar el ingreso de personal no autorizado a un sitio específico, para este caso, al Laboratorio de Comunicaciones Unificadas de la Escuela Politécnica Nacional. El prototipo permite el ingreso y eliminación de rostros a través de una interfaz gráfica, también brinda la posibilidad de verificar los accesos registrados en determinada fecha y hora.

- La metodología empleada para desarrollar este Proyecto de Titulación ha sido la de Desarrollo de Prototipos, una metodología no muy conocida y no tan empleada para proyectos de software, sin embargo, para este trabajo resultó muy útil ya que, el sistema prototipo está compuesto por componentes de software y una parte muy importante de hardware, por esto el empleo de esta metodología ya que puede ser utilizada para proyectos que integren ambos campos. Además, esta metodología no solo permite el desarrollo rápido del proyecto, sino que permite a través de cada prototipo ir mejorando el mismo.

- El proceso de reconocimiento de rostros desarrollado en el Proyecto de Titulación resulto tener una alta efectividad gracias a desarrollarlo en Python, utilizando diferentes librerías compatibles con este lenguaje, como OpenCV, que permite realizar un procesamiento de imágenes muy sencillo gracias a las funciones que esta librería dispone y también disponen de varias funciones asociadas a procesos de detección, procesamiento y reconocimiento de objetos.

- Las pruebas realizadas al sistema prototipo permitieron comprobar su correcta funcionalidad, sin embargo, es importante mencionar que debido a las diferentes limitaciones de procesamiento por parte de los dispositivos utilizados (Raspberry) no fue posible emplear el mejor método de reconociendo facial (cnn), influyendo directamente en el nivel de confiabilidad del sistema prototipo.

- El porcentaje de eficiencia de reconocimiento del sistema prototipo depende tanto del nivel de iluminación, como de la distancia a la que se capture la imagen del rostro del sujeto a ser reconocido, además de la cantidad de imágenes recopiladas para cada usuario. Las pruebas determinaron un porcentaje de acierto del 94% con un

nivel de iluminación Medio y 97% a una distancia de 0.50m. Por esta razón, se concluye que las capturas deben realizarse a 0.50m con un nivel Medio de iluminación; cabe mencionar que el nivel de confiabilidad del prototipo depende de estos factores.

- En base a las pruebas de confiabilidad realizadas, se concluye que, a pesar de las limitaciones, se obtuvo un 78% de acierto, es decir el prototipo casi siempre brinda una respuesta correcta al momento de capturar un rostro similar a uno ya registrado, por esto se puede determinar que tiene un alto nivel de confiabilidad, sin embargo, al ser un dispositivo de seguridad este porcentaje no es lo ideal.

4.2. RECOMENDACIONES

- Si se desea mejorar la eficiencia de reconocimiento facial para futuros prototipos se recomienda emplear un servidor con una alta capacidad de procesamiento donde se pueda alojar el módulo WebCam, para de esta forma utilizar el método *"cnn"* de detección de rostros. Además, de utilizar la Raspberry únicamente para el control de la cerradura y actualización del archivo de reportes de acceso.

- Es recomendable utilizar Python para desarrollar programas o scripts que se ejecutarán en diferentes sistemas operativos ya que su característica multiplataforma lo permite; sin embargo, hay que tomar en cuenta que la versión del interprete Python que se vaya a utilizar ya que puede presentar errores de compatibilidad al momento de ejecutar algún programa.

- Para el desarrollo de la interfaz gráfica, en el caso de mejorarla, se podría utilizar otra librería que disponga de más elementos gráficos a disposición y que permita darle más funcionalidad, ya que con Tkinter, que fue la utilizada en el sistema prototipo, se tiene algunas limitaciones.

- Al momento de utilizar los puertos GPIO de la Raspberry Pi, tener mucho cuidado al conectar los pines a otros dispositivos, asegurarse de emplear todas las protecciones, verificar voltajes y corrientes de entrada y salida, verificar las funcionalidades de cada pin y no energizar la Raspberry sin comprobar que todo este correcto, ya que un sobrevoltaje podría dañar la placa de la misma.

- Para mejorar en nivel de confiabilidad del proceso de reconocimiento facial del prototipo se recomienda: recopilar al menos 25 imágenes por cada usuario registrado, realizar las capturas por lo menos con un nivel medio de iluminación, además de procurar que el usuario se ubique en un rango de distancia de 0.45 a 0.55 metros desde el lente de la cámara hacia su rostro.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Python Community, «Python,» 2001. [En línea]. Available: https://www.python.org/. [Último acceso: 2 Diciembre 2018].

[2] OpenCV Team, «OpenCV,» 2019. [En línea]. Available: https://opencv.org/about/. [Último acceso: 2 Diciembre 2018].

[3] M. A. Garduño Santana, L. E. Díaz-Sánchez, I. Tabarez Paz y M. Romero Huertas, «Estado del arte en reconocimiento facial,» *Research in Computer Science,* nº 140, pp. 19-27, 2017.

[4] Xataka, «Xataka - Reconocimiento Facial en la India,» Xataka, Mayo 2018. [En línea]. Available: https://www.xataka.com/robotica-e-ia/en-la-india-han-usado-reconocimiento-facial-para-localizar-ninos-extraviados-y-en-solo-cuatro-dias-han-encontrado-casi-3-000. [Último acceso: 09 Marzo 2019].

[5] H. P. Espinosa Peralta, *DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE SEGURIDAD Y ALERTA PARA VEHICULOS, BASADO EN RECONOCIMIENTO FACIAL Y LOCALIZACIÓN GPS, EN UNA RASPBERRY PI B PLUS.,* Quito: Tesis Ingeniería, 2016.

[6] N. Acar, «Towards Data Science,» 21 Agosto 2018. [En línea]. Available: https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184. [Último acceso: 5 Junio 2019].

[7] F. Emiliano Aguayo, E. A. Rivas-Araiza y J. C. Pedraza Ortega, «Research Gate,» Octubre 2017. [En línea]. Available: https://www.researchgate.net/publication/320269791_Desarrollo_de_un_proceso_de_autenticacion_facial_en_un_sistema_Android_utilizando_el_algoritmo_LDA_Analisis_de_Discriminacion_Lineal. [Último acceso: 08 Junio 2019].

[8] A. Rosebrock, «pyimagesearch - Local Binary Patterns with OpenCV,» 7 Diciembre 2015. [En línea]. Available: https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/. [Último acceso: 1 Julio 2019].

[9] U. González Benítez, *Detección de presonas a partir de Visión Artificial,* Madrid: Tesis Ingeniería - Universidad Carlos III Madrid, 2010.

[10] P. Viola y J. Michael , «Robust Real-Time Object Detection,» *International Journal of Computer,* vol. II, nº 57, pp. 137-154, 2004.

[11] M. Guevara, J. Echeverry y W. Ureña, «Detección de Rostros en imagenes digitales usando Clasificadores en Casacada,» *Scientia et Technica,* nº 38, pp. 1-6, 2008.

[12] C. Tolosa Borja y Á. Giz Bueno, «UCLM - Trabajo de Biometría,» [En línea]. Available: https://www.dsi.uclm.es/personal/MiguelFGraciani/mikicurri/Docencia/Bioinformatica/web_BIO/Documentacion/Trabajos/Biometria/Trabajo%20Biometria.pdf. [Último acceso: 1 Julio 2019].

[13] I. Sommerville, Ingenieria de Software, México: Pearson Education, 2002, pp. 95-190.

[14] Varios, «EcuRed - Modelo de Prototipos,» 20 Junio 2018. [En línea]. Available: https://www.ecured.cu/Modelo_de_prototipos. [Último acceso: 28 Mayo 2019].

[15] Varios, «Lucidchart - Qué es un diagrama de flujo,» Lucid Software Inc, 2019. [En línea]. Available: https://www.lucidchart.com/pages/es/que-es-un-diagrama-de-flujo. [Último acceso: 30 Agosto 2019].

[16] H. Costa Guzmán, «Medium - Fudamentos para Interfaces Graficas con Python,» Medium, 21 Mayo 2019. [En línea]. Available:

https://medium.com/@hektorprofe/fundamentos-para-interfaces-gr%C3%A1ficas-con-python-c7503e30de58. [Último acceso: 15 Septiembre 2019].

[17] Varios, «Wikibooks - Gestión de Ventanas Tkinter,» WikiMedia, 4 Agosto 2019. [En línea]. Available: https://es.wikibooks.org/wiki/Python/Interfaz_gr%C3%A1fica_con_Tkinter/Gesti%C3%B3n_de_ventanas. [Último acceso: 15 Septiembre 2019].

[18] Comunidad Python Agentina, «PyAr - Interfaces Gráficas,» PyAr, Septiembre 2004. [En línea]. Available: http://www.python.org.ar/wiki/InterfacesGraficas. [Último acceso: 15 Septiembre 2019].

[19] Raspberry Pi Fundation, «Teach, learn and make with Raspberry Pi - Raspberry Pi 2 Model B,» 2019. [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/. [Último acceso: 17 Septiembre 2019].

[20] Varios, «Xataka - Raspberry Pi 4 Caracteristicas, precio y ficha técnica,» Xataka, 24 Junio 2019. [En línea]. Available: https://www.xataka.com/ordenadores/raspberry-pi-4-caracteristicas-precio-ficha-tecnica. [Último acceso: 17 Semptiembre 2019].

[21] Logitech, «Logitech - WebCam C270 Specifications,» 2019. [En línea]. Available: https://www.logitech.com/es-roam/product/hd-webcam-c270#specification-tabular. [Último acceso: 17 Septiembre 2019].

[22] Varios, «Naylamp Mechatronics - Modulo de Relay 2 canales,» 2019. [En línea]. Available: https://naylampmechatronics.com/drivers/31-modulo-relay-2-canales-5vdc.html. [Último acceso: 18 Setiembre 2019].

[23] SONGLE, *SONGLE RELAY - DataSheet.*

[24] Raspberry Pi, «Raspberry Pi - Raspberry Pi 4 Model B,» 2019. [En línea]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. [Último acceso: 18 Septiembre 2019].

[25] Raspberry Pi, «Raspberry Pi - Documentation Noobs,» Septiembre 2014. [En línea]. Available: https://www.raspberrypi.org/documentation/installation/noobs.md. [Último acceso: 18 Septiembre 2019].

[26] Raspbian Team, «Raspbian,» Junio 2012. [En línea]. Available: https://www.raspbian.org/FrontPage. [Último acceso: 18 Septiembre 2019].

[27] Microsoft, «Microsoft - Documentos de Visual Studio,» Microsoft , 18 Marzo 2019. [En línea]. Available: https://docs.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2017. [Último acceso: 19 Septiembre 2019].

[28] Python Community, «Python Documentation,» Python, 2001. [En línea]. Available: https://docs.python.org/3/tutorial/venv.html. [Último acceso: 19 Septiembre 2019].

[29] A. Rosebrock, «pyimagesearch - Install OpenCV 4 on Raspberry Pi 4 and Raspbian Buster,» pyimagesearch, 16 Septiembre 2019. [En línea]. Available: https://www.pyimagesearch.com/2019/09/16/install-opencv-4-on-raspberry-pi-4-and-raspbian-buster/. [Último acceso: 20 Septiembre 2019].

[30] A. Rosebrock, «pyimagesearch - How to build a custom face recognition dataset,» pyimagesearch, 11 Junio 2018. [En línea]. Available: https://www.pyimagesearch.com/2018/06/11/how-to-build-a-custom-face-recognition-dataset/. [Último acceso: 12 Diciembre 2018].

[31] OpenCV, «OpenCV Documentation - Cascade Classification,» OpenCV, 2014. [En línea]. Available:

https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html. [Último acceso: 19 Septimebre 2019].

[32] GitHub Contributors, «GitHub - imutils,» 1 Julio 2019. [En línea]. Available: https://github.com/jrosebr1/imutils/blob/master/imutils/video/videostream.py. [Último acceso: 19 Setiembre 2019].

[33] Python, «Python Documentation - shutil - High-level file operations,» Python, 2001. [En línea]. Available: https://docs.python.org/3/library/shutil.html. [Último acceso: 20 Septiembre 2019].

[34] A. Rosebrock, «pyimagesearch - Face Detection with OpenCV and deep learning,» pyimageserach, 26 Febrero 2018. [En línea]. Available: https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/. [Último acceso: 10 Diciembre 2018].

[35] A. Rosebrock, «pyimagesearch - Face recognition with OpenCV, Python and deep learning,» pyimagesearch, 18 Junio 2018. [En línea]. Available: https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/. [Último acceso: 5 Enero 2019].

[36] A. Geitgey, «GutHub - face_recognition,» GitHub, 2017. [En línea]. Available: https://github.com/ageitgey/face_recognition. [Último acceso: 25 Septiembre 2019].

[37] Python , «Python Documentation - pickle - Python object serialization,» Python, 2001. [En línea]. Available: https://docs.python.org/3/library/pickle.html. [Último acceso: 25 Septiembre 2019].

[38] A. Rosebrock, «pyimagesearch - Raspberry Pi Face Recognition,» pyimagesearch, 25 Junio 2018. [En línea]. Available: https://www.pyimagesearch.com/2018/06/25/raspberry-pi-face-recognition/. [Último acceso: 20 Febrero 2019].

[39] L. Pounder, «tom´sHARDWARE - Raspberry Pi GPIO Pinout: What Each Pin Does on Pi 4, Earlier Models,» 26 Junio 2019. [En línea]. Available: https://www.tomshardware.com/reviews/raspberry-pi-gpio-pinout,6122.html. [Último acceso: 28 Septiembre 2019].

[40] Anónimo, «SOURCEFORGE - raspberry - gpio - python,» SOURCEFORGE, 1 Enero 2016. [En línea]. Available: https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/. [Último acceso: 28 Septiembre 2019].

[41] Visuality Systems, «CIFS - ALL ABOUT CIFS,» Visuality Systems, 2016. [En línea]. Available: https://cifs.com/. [Último acceso: 30 Septiembre 2019].

[42] Diverteka, «DIVERTEKA - Windows comparte con Raspberry,» Diverteka, 13 Enero 2013. [En línea]. Available: http://www.diverteka.com/?p=867. [Último acceso: 30 Septiembre 2019].

[43] J. Santamaría Formoso, *Tesis - Módulo biométrico de imágenes para reconocimiento facial de usuarios,* Logroño - España: Universidad de La Rioja, 2017.

[44] C. Niola y W. Sarango, *Tesis - DESARROLLO DE UN SOFTWARE DE SEGURIDAD PARA DETECCIÓN Y RECONOCIMIENTO FACIAL BASADO EN LOS ALGORITMOS DE VIOLA-JONES Y PCA EIGENFACE,* Cuenca - Ecuador: Universidad Politécnica Salesiana, 2019.

[45] Varios, «Programación Orientada a Objetos,» UNAL, 2017. [En línea]. Available: https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/index.html. [Último acceso: 10 Julio 2019].

[46] A. Alvarez, «Guia Tkinter,» Git Hub, 2015. [En línea]. Available: https://guia-tkinter.readthedocs.io/es/develop/chapters/6-widgets/6.1-Intro.html. [Último acceso: 16 Septiembre 2019].

[47] TP-LINK Technologies Co., «TP-Link - TL-SF1008D Datasheet,» TP-LINK Technologies Co., 2019. [En línea]. Available: https://static.tp-link.com/res/down/doc/TL-SF1008D_V8_Datasheet.pdf. [Último acceso: 2019 Septiembre 18 2019].

[48] Varios, «EcuRed - Lenguaje Intepretado,» EcuRed, 16 Septiembre 2011. [En línea]. Available: https://www.ecured.cu/Lenguaje_interpretado. [Último acceso: 29 Junio 2019].

[49] A. Rosebrock, «pyimagesearch - Accessign RPi.GPIO and GPIO Zero with OpenCV + Python,» pyimagesearch, 2 Mayo 2016. [En línea]. Available: https://www.pyimagesearch.com/2016/05/02/accessing-rpi-gpio-and-gpio-zero-with-opencv-python/. [Último acceso: 15 Marzo 2019].

[50] A. Rosebrock, «pyimagesearch - Python, argparse and command line arguments,» pyimageserach, 12 Marzo 2018. [En línea]. Available: https://www.pyimagesearch.com/2018/03/12/python-argparse-command-line-arguments/. [Último acceso: 10 Diciembre 2018].

ANEXOS

ANEXO A. Instalación de Python

ANEXO B. Instalación de OpenCV

ANEXO C. Manual de Usuario

ANEXO D. Códigos Sistema Prototipo

NOTA: Los anexos se encuentran adjuntos en el CD.

ORDEN DE EMPASTADO