Universidad Autónoma del Estado de México

Facultad de Ingeniería

Ingeniería en Computación

DESARROLLO DE UN NUEVO ALGORITMO DE DETECCIÓN FACIAL Y SU APLICACIÓN A UN ATRIL AUTOMATIZADO

Tesis para obtener el título de Ingeniero en Computación

Saul David Fonseca Hernández

Asesor

Dr. Javier Salas García

Co-asesor

Dra. Rosa María Valdovinos Rosas

Toluca – México

Noviembre del 2022

RESUMEN

Las ponencias frente a un auditorio deben tener una calidad de sonido y de contenido durante su ejecución y muchas veces se ve que esta calidad es deficiente debido a problemas como una mala colocación del micrófono o las constantes interrupciones al orador para recibir mensajes de los coordinadores durante su ponencia además que la ergonomía de los atriles donde los oradores exponen no es cubierta debido a las distintas características físicas de las personas, por lo anterior dicho, en un proyecto se realizó un atril automatizado que aparte del hardware para elevar y bajar la paleta donde se colocan los documentos del ponente, se requiere el cálculo de altura para realizar dicho movimiento usando un algoritmo de detección facial. Con las variables necesarias para calcular la altura de la persona que está frente al atril con el fin de colocar los componentes como la paleta del atril y el micrófono a una altura ergonómica para el orador. Un algoritmo de detección facial *(Viola-Jones)* ejecutado en una *Raspberry Pi 3 B+* suele tener un elevado uso de CPU (por arriba del 70%), por lo tanto se busca desarrollar un nuevo algoritmo de detección facial *(Fonseca-Salas)* que pueda operar con un bajo uso de CPU. Dicho algoritmo reduce la cantidad de información a analizar mediante el uso de trazas y la detección de cambios de intensidad para detectar la región facial de interés. Los resultados obtenidos con el algoritmo *Fonseca-Salas* se ven reducidos en una tercera parte comparado con el algoritmo *Viola-Jones*, ambos implementados en *Python*. Los valores de uso de porcentaje de CPU obtenidos por el algoritmo *Fonseca-Salas* están alrededor de *26%*. Por el otro lado, el algoritmo *Viola-Jones* con *Adaboost* obtiene unos valores de *70%* a *80%* de uso de CPU, es importante mencionar que la detección facial para este algoritmo no siempre ocurre ya que se debe mirar de frente a la cámara, si la cara es levantada o agachada levemente no ocurre la detección facial. Por otro lado, una vez que la detección facial ocurre con el algoritmo *Fonseca-Salas* se realiza el cálculo de la altura, obteniendo unos errores porcentuales entre *0.32* y *1.96*. Con los valores de porcentaje de uso de CPU y errores porcentuales mencionados se ha logrado crear un algoritmo de detección facial capaz de realizar el cálculo de la altura a bajo coste computacional.

"Slowly is the fastest way to get to where you want to be."

- André de Shields

INDICE

1. INTRODUCCIÓN

1.1 Antecedentes de la detección facial

La detección facial consiste en analizar distintos tipos de fotografías e identificar dónde existen rostros, sin tomar importancia de a quién pertenece el rostro. Existen diversas técnicas que permiten la detección del rostro a través de distintos factores, como la distancia entre los ojos, distancia entre nariz y boca. Se han desarrollado distintos tipos de algoritmos que, en porcentajes elevados, reconocen satisfactoriamente los rostros en una fotografía o video y a partir de esto se realiza el reconocimiento facial para sistemas de seguridad en el hogar u oficinas.

Actualmente la detección facial forma parte del reconocimiento facial en dispositivos móviles y sistemas de seguridad en el hogar. El proceso de reconocimiento facial no consigue en todas las ocasiones un resultado satisfactorio porque depende de factores físicos de la obtención de la imagen como son: la intensidad de luz que recibe el rostro del individuo, el ángulo en que la fotografía fue tomada y las distintas resoluciones de la cámara empleada.

En el artículo *Comparative Testing of Face Detection Algorithms* se menciona que las tareas que requieren del uso de la detección facial son requeridas más frecuentemente en la vida diaria de las personas *(Degtyarev y Seredin, 2010)*. Por otro lado, en el trabajo de investigación *Sistema de reconocimiento facial y realidad aumentada para dispositivos móviles* se asegura que el *hardware* y *software* para sistemas de seguridad han tenido un gran impulso en los últimos años y el reconocimiento facial destaca entre estos porque el uso de cámaras de video también ha aumentado en nuestro hogar, así como en el trabajo y lugares de recreación *(Pérez y Pellicer, 2012)*. El mejoramiento de los algoritmos para la detección facial traería grandes beneficios a todo tipo de usuarios, desde el mejor posicionamiento de un filtro digital usado en las redes sociales, hasta la integración de sistemas de seguridad en hogares particulares.

1.2 Objetivos

Desarrollar un nuevo algoritmo de detección facial implementado en el lenguaje de programación *Python* que pueda operar a bajo consumo de CPU en una *Raspberry Pi 3 B+* para controlar la automatización de la altura de la paleta de documentos de un atril.

1.3 Hipótesis

El desarrollo de un algoritmo de detección facial es capaz de funcionar por debajo de un 30% de uso de CPU comparado con otro algoritmo incluido en bibliotecas de tratamiento de imágenes de uso frecuente.

1.4 Contenido

El capítulo 1 muestra los antecedentes de la detección facial, los usos más comunes actualmente y las ventajas de hacer más eficientes estos algoritmos. Por otro lado, se definen los objetivos, alcances y la hipótesis a desarrollar en este trabajo de investigación.

En el capítulo 2 se presentan los antecedentes sobre el procesamiento de imágenes digitales, definiendo matemáticamente qué es una imagen digital, el muestreo y transformación, así como la cuantificación de la información obtenida para ser representada digitalmente. Las etapas para el procesamiento de imágenes digitales son: mejora, restauración, análisis y compresión de una imagen. Una vez sentada las bases sobre imágenes digitales se explica el proceso de reconocimiento facial en imágenes digitales, explicando cada una de las cinco etapas de las que consiste: adquisición de la imagen, detección facial, acondicionamiento, normalización, extracción de características y reconocimiento. Por último, la definición de un algoritmo de detección facial en tiempo real y su funcionamiento, incluyendo el *hardware* requerido para su funcionamiento.

La metodología de esta tesis se muestra en el capítulo 3 y describe los elementos físicos que forman parte del atril además de sus conexiones físicas. También se describen los componentes eléctricos, mecánicos y electromecánicos para todo el funcionamiento del sistema físico. Además, se define el algoritmo de detección facial mencionando las etapas en las que consiste y las condiciones iniciales (de hardware y de software) que se deben tener en cuenta para su desarrollo y posteriormente la implementación.

El capítulo 4 contiene los resultados del desarrollo e implementación del algoritmo. Dichos resultados documentan el diseño y la implementación física de nuestro sistema de captura de imágenes que consisten en un soporte para videocámara que puede girar en distintos ángulos la videocámara con ayuda de un servomotor fijado a la paleta del atril con ayuda de *software*. Las pruebas del algoritmo de detección facial se realizaron de dos formas, a través de imágenes digitales y de video digital en tiempo real, en las primeras se evaluó el funcionamiento del algoritmo y que tan asertivo llega a ser, por otro lado, las pruebas de video digital en tiempo real evaluaron el rendimiento de ejecución de dicho algoritmo comparándolo con la implementación de otro algoritmo de detección facial en tiempo real.

Finalmente, las conclusiones de este trabajo se presentan en el capítulo 5. Los resultados obtenidos del capítulo 4 son explicados entre los dos algoritmos: el desarrollado en este trabajo nombrado algoritmo *Fonseca-Salas* y el algoritmo *Viola-Jones*; detallando el motivo de las diferencias de resultados, incluso, mencionando las ventajas de ambos algoritmos y que beneficios trae o no, que el consumo de recursos sea distinto.

2. ANTECEDENTES

2.1 Procesamiento de Imágenes Digitales

El procesamiento de imágenes digitales es un proceso que involucra distintas técnicas para interpretar claramente una imagen digital. Actualmente, este proceso es utilizado en infinidad de aplicaciones que van desde el ámbito de la medicina, pasando por el sector militar e incluso llegando hasta el usuario común, para tareas no necesarias, pero sí cotidianas. En los siguientes párrafos se describirán las bases de la detección facial y las distintas técnicas para su ejecución, también se mencionan las cinco etapas principales de este proceso.

Las imágenes digitales son usadas todos los días en grandes cantidades. De acuerdo con el artículo *Instagram, las cifras imprescindibles para el 2021* cada día en la red social de Instagram se cargan más de 100 millones de fotos y videos *(Morales, 2021)*. Ahora que se conoce el número de imágenes digitales utilizadas diariamente en redes sociales, se procede a definir qué es una imagen digital.

Una imagen digital puede ser definida como una función dimensión $f(x,y)$, donde x y y son coordenadas espaciales, y la amplitud de f a cualquier par coordenado x,y es llamado intensidad o nivel de gris de la imagen en ese punto *(Gonzalez y Woods, 2018)*.A cada uno de estos pares coordenados, se le da el nombre de pixel. Esta imagen por sí sola, es el insumo inicial para el procesamiento digital de imágenes.

De acuerdo con el libro *Digital Image Processing* para ser procesado digitalmente, se debe muestrear y transformar el insumo de entrada (imagen), en una matriz donde cada elemento debe ser cuantificado para ser representados digitalmente, es decir asignar un valor numérico entero *(da Silva & Mendonça, 2005)* . El procesamiento digital de imágenes se divide en varias etapas: *1)* mejora de imagen, *2)* restauración de imagen, *3)* análisis de imagen y *4)* compresión de imagen. A continuación, se describe brevemente cada etapa.

Dentro de la etapa *1) mejora de imagen*, ésta es manipulada generalmente mediante técnicas heurísticas para resaltar información útil, según el interés, así como sucede en las radiografías médicas. La etapa *2) restauración de imagen* pretende mejorar el insumo de entrada dañada con ayuda de matemáticas para restaurar la información dentro de la imagen. La etapa *3) análisis de la imagen* busca que la extracción de la información sea realizada en automático, es decir, obtener las características de interés en nuestra imagen, por ejemplo, si se requiere analizar la calidad de granos de arroz, la forma de cada arroz deberá ser detectada en la imagen automáticamente. Para llevar a cabo este análisis es necesario una gran cantidad de datos, por ejemplo, una imagen en escala de grises y de tamaño cuadrado de *512x512* pixeles y una cuantificación de 8 bits, es decir 256 valores de intensidad es necesario: $512*512*8 = 2,097,152 = 2 \cdot 10^6 bits$, con el resultado anterior, se necesita una forma óptima para poder almacenar y transmitir la información, esto se logra gracias a la etapa *4) compresión de imagen*, además que la redundancia de información en imágenes favorece esta etapa para poder presentar la imagen, existen distintos tipos de compresión para imágenes digitales como .jpg, .png. .tiff, por mencionar algunos.

El proceso del reconocimiento facial consta de 6 etapas: *1) adquisición de la imagen*, *2) detección del rostro*, *3) acondicionamiento*, *4) normalización*, *5) extracción de características* y *6) reconocimiento (Moreano, Pulloquinga, Lagla, Chisag y Pico, 2017)*. Cada una de estas etapas puede contener otras subetapas.

Se sabe que 100% de eficiencia no se logra obtener para ningún ámbito, de acuerdo con el *Center For Strategic & International Studies,* en el 2014 la tasa de error de los algoritmos de detección facial fue de 4.1% y para el 2020 este valor se había reducido a tan solo 0.08% *(Crumpler, 2020)*. Claro que para lograr una tasa de error tan baja se debe a que esas cifras aplican para algunos factores como condiciones iniciales durante su ejecución. Algunos factores que influyen para la detección facial son: *1)* la luz que recibe el rostro del individuo, *2)* el ángulo en que la fotografía fue tomada y *3)* las características de la cámara que captura la fotografía.

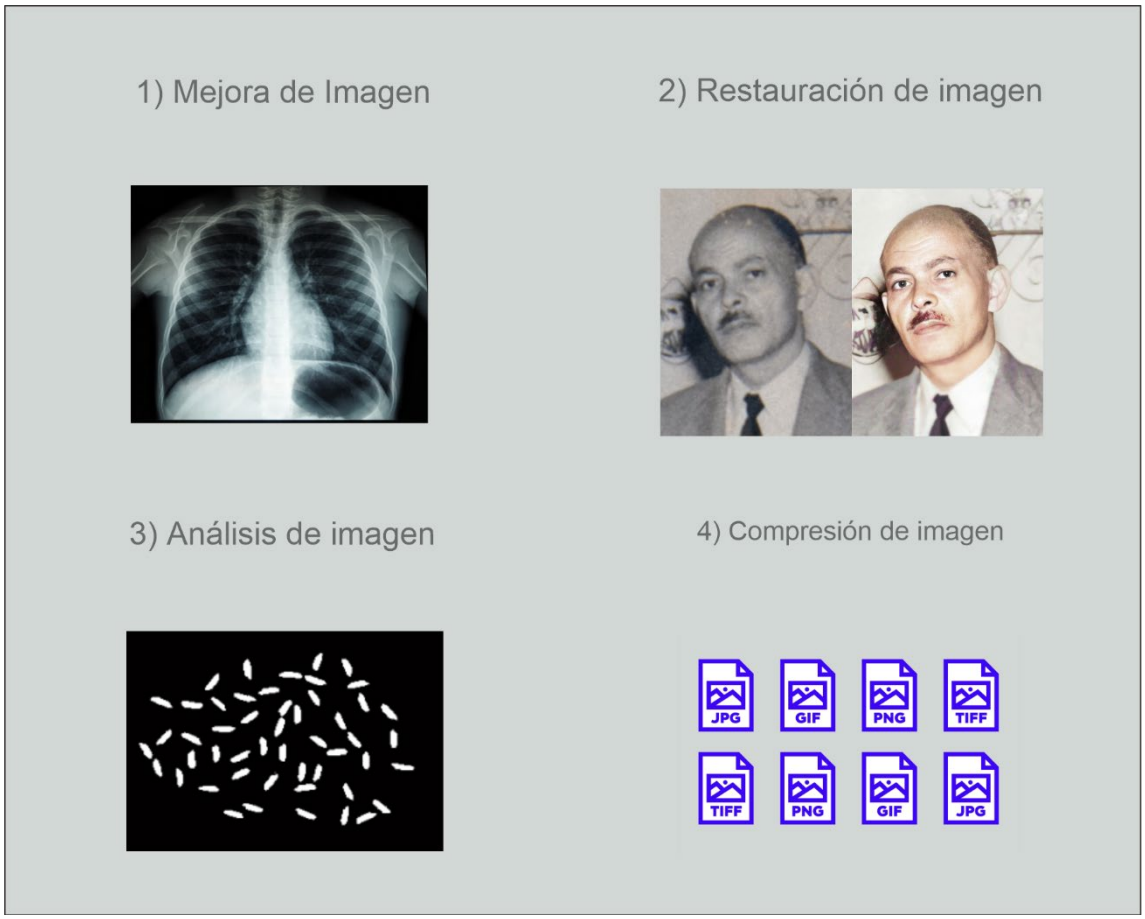

***Figura 1**: Etapas del procesamiento de imágenes digitales (elaboración propia).*

De la *Figura 2* se muestra un recuadro con línea punteada que encierra las etapas *de adquisición imagen* y *detección facial* (etapa 1 y 2, respectivamente), esto con la intención de no olvidar que este trabajo se centra solo en esos dos pasos mencionados.

2.2 Detección facial en tiempo real

Como se ha indicado antes, la detección facial precede al proceso del reconocimiento facial. A continuación, se explica en detalle esta primera etapa.

El algoritmo de *Viola-Jones* es un algoritmo para la detección de objetos en tiempo real, en donde dichos objetos tienen características del rostro humano. Garantiza ser extremadamente rápido, además, de ser altamente efectivo. Este proceso es realizado, primeramente, con una técnica para representar imágenes llamada *"Imagen integral"*, que

genera el conjunto de características faciales. A continuación, con el uso de redes neuronales y *AdaBoost*, estos conjuntos de características son los insumos para crear clasificadores de las características faciales base. Y por último, se ejecuta un algoritmo para clasificadores de cascada, que aumenta radicalmente el rendimiento (uso de CPU y tiempo de ejecución) a la hora de la detección facial *(Viola y Jones, 2004)*.

Lo anterior dio origen a un *Framework* sin nombre, para la detección facial capaz de procesar extremadamente rápido y con grandes porcentajes de éxito. Lograba detectar rostros hasta en 15 cuadros por segundos en imágenes con tamaño de 384 por 288 pixeles. Las características del *hardware* eran Intel Pentium III a 700 MHz. Es importante mencionar que este algoritmo no funciona con los valores de intensidad en los píxeles, pero sí con comparación de características *(Viola y Jones, 2004)*.

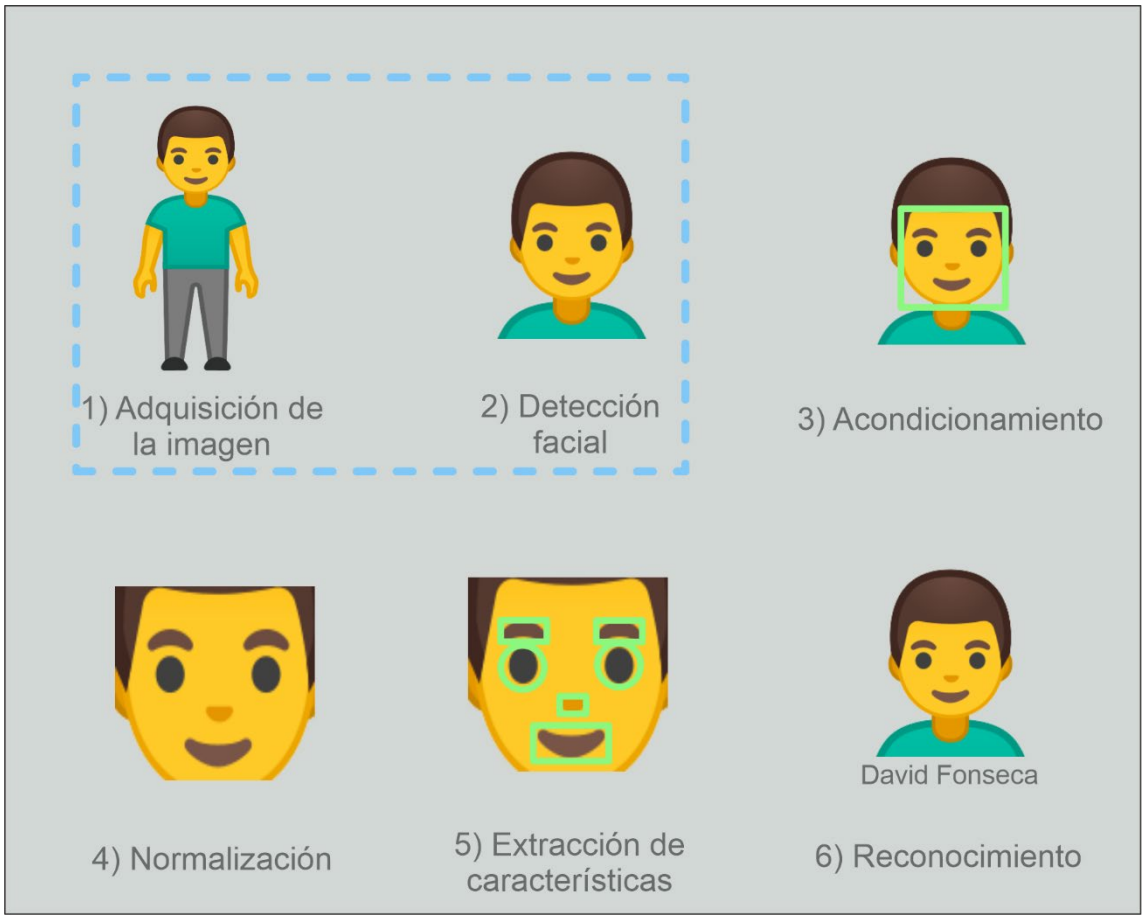

Figura 2: *Etapas de reconocimiento facial en imágenes digitales (elaboración propia).*

En el año 2001 se definió un tipo de clasificación, de acuerdo con los avances tecnológicos de *software* y *hardware*. Tiene dos grandes categorías: *características basadas en enfoques* e *imágenes basadas en enfoques.*

Las características basadas en enfoque parten de un problema principal: encontrar un rostro en una imagen con un fondo realista y así, surgen tres técnicas: *1*) análisis de bajo nivel, que una vez que se realiza la segmentación, trabaja con características de los pixeles, si está a color o en escala de grises, detección de bordes, movimiento y medidas generalizadas. *2) Análisis de características*, donde las características visuales (ojos, labios, nariz, etc.) son representados en conceptos más globales de la cara y rasgos faciales usando información geométrica del rostro. *3) Modelos de forma activa*, son modelos que intentan adaptarse a un objeto en una imagen, como el modelo de las serpientes, plantillas deformables y/o modelos distribuidos por puntos que se encargan de extraer características complejas y no rígidas como la pupila.

Por otra parte, las imágenes basadas en enfoque surgen cuando se desea encontrar más de un rostro en una imagen. Aquí, la detección facial es tratada como un problema de reconocimiento de patrones. Al tratarse de patrones, se deberá contar con un entrenamiento de imágenes con y sin rostros. Estos entrenamientos se basan en métodos de subespacio lineal, redes neuronales y enfoques estadísticos *(Hjelmås y Low, 2001)*.

Ahora que se ha descrito de manera general cómo funciona la detección facial en imágenes y en tiempo real, además de los dos grandes enfoques usados en estas tareas es necesario tener una descripción del *hardware* que se empleó en la implementación del algoritmo desarrollado en este trabajo de tesis.

2.3 Raspberry Pi para el control de dispositivos electrónicos

The Raspberry Pi Foundation es una organización benéfica que trabaja para que el poder de la computación y la creación digital llegue a manos de las personas en todo el mundo. Su objetivo es que más personas puedan aprovechar el poder de la computación y las tecnologías digitales para resolver problemas de su entorno y expresarse creativamente.

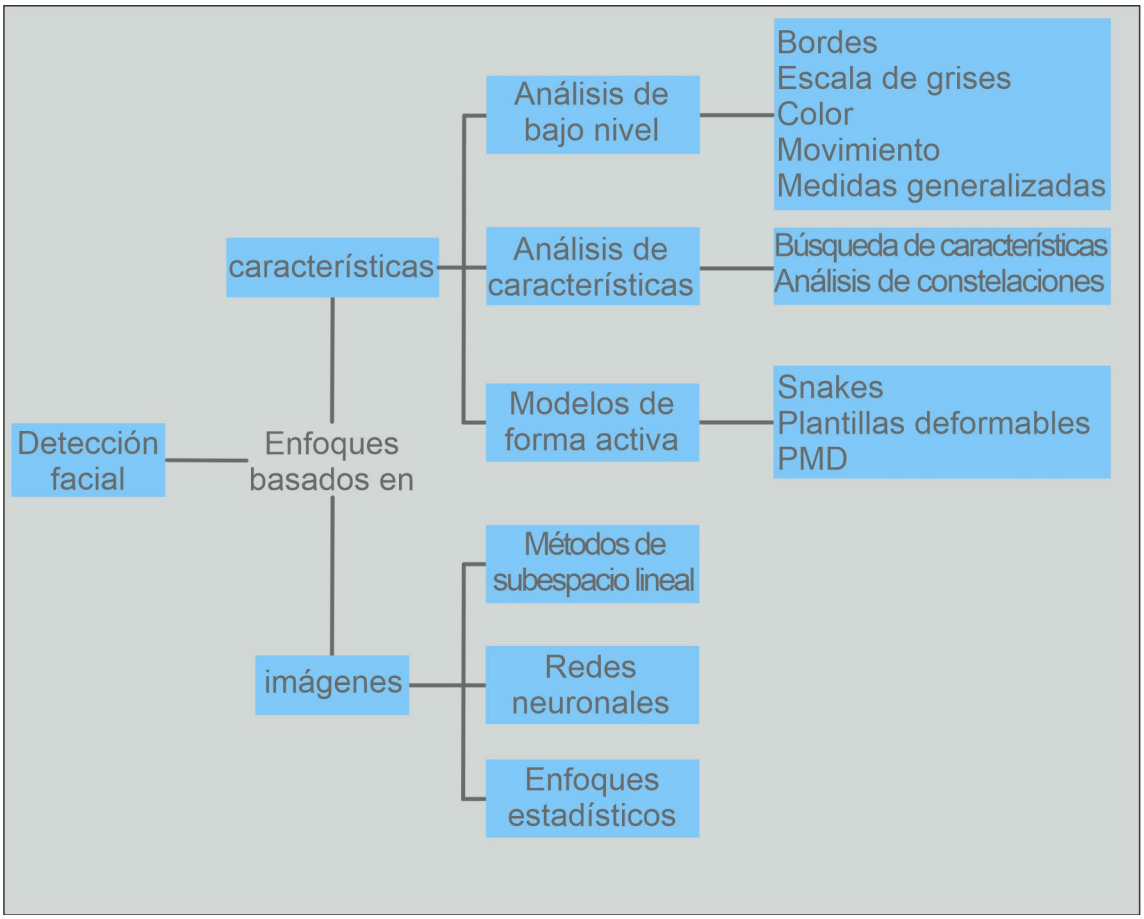

Figura 3: *Enfoques para métodos generales de la detección facial, de acuerdo con Hjelmås y Low (2001) (elaboración propia).*

La *Raspberry Pi* es una minicomputadora de bajo costo desarrollada en el Reino Unido por *The Raspberry Pi Foundation*. Actualmente consta de 8 versiones del producto. En cada versión ha traído mejoras de hardware. Nos centraremos en mostrar las características principales del producto *Raspberry Pi 3 B+* en la siguiente tabla.

Gracias a *The Raspberry Pi Foundation* se cuenta con material educativo para personas de cualquier edad y tutoriales básicos de programación. Por otro lado, también cuenta con un *blog*, donde la comunidad participa creando entradas para el uso de *Raspberry Pi* en proyectos electrónicos de todo tipo, desde la creación de pequeños robots hasta temas relacionados a domótica.

***Tabla 1**: Características de la Raspberry Pi 3 B+ empleada en el desarrollo de este trabajo.*

Especificaciones	Raspberry Pi 3 modelo B+
SoC*	Broadcom BCM2837
CPU	1.4 GHz 64-bit quad-core ARMv8
Set de instrucciones	RISC de 64 bits
GPU	VC-1 (con licencia), 1080p30 H.264/MPEG-g AVC
Memoria	1GB (compartidos con la GPU)

**SoC proviene del acrónimo en inglés System-on-a-Chip y que hace referencia a la tendencia de implementar en un chip tantas funciones como fuera posible.*

2.4 Problemática: ajuste de un atril en presentaciones

Se tiene un atril con un sistema electrónico que cuenta con una pantalla en la cual se puede enviar mensajes al orador comunicándole cuanto tiempo le queda de exposición o que el nivel de voz no es el adecuado para el auditorio y, por ende, no se escucha correctamente. También cuenta con un mecanismo capaz de subir o bajar la paleta que soporta los documentos del expositor, con la finalidad de que dicha paleta y sus componentes, se sitúe a una altura correcta, de acuerdo con la estatura del orador. Todo este sistema es controlado por una *Raspberry Pi 3* y no puede ejecutar algún algoritmo actual de detección facial en tiempo real, ya que el consumo de recursos es elevado para este tipo de minicomputadora. Por lo que se busca desarrollar un algoritmo que obtenga un desempeño eficiente en dicho dispositivo y que además funcione correctamente. El uso de un atril con las mejoras mencionadas anteriormente, reducirían el tiempo que tarda una persona en adecuar el *pódium*, de una manera ergonómica. También se le podrían enviar notas al orador sobre el tiempo restante que le queda, sin interrumpirlo y ofreciendo una exposición más ordenada para el auditorio presente, ya que no se verán personas que ayudan al orador con la adecuación mencionada.

En este trabajo se parte del desarrollo de un atril automatizado, capaz de ajustarse a la altura de la persona que lo use, haciendo que la interacción con el atril sea totalmente ergonómica. Para lograr esta ergonomía se utiliza una cámara para obtener imágenes de la persona parada frente al atril y con el producto de dicho proceso, los resultados son evaluados para comenzar el proceso de imágenes digitales y detección facial. Una vez terminado, el atril, a través de su sistema electromecánico, debe situar el micrófono y otros componentes de la sección móvil a la altura que mejor se adecue a la persona que está enfrente.

Cabe destacar que el proyecto general es la construcción de un atril automatizado y este trabajo únicamente se centra en la creación del algoritmo de detección facial. En el subcapítulo *2.5.1 Requisitos del automatizado de un atril* se describen los componentes físicos mecánicos, eléctricos y electromecánicos que pertenecen a la configuración del atril.

2.4.1 Requisitos del automatizado de un atril

Como se mencionó al comienzo de este capítulo, el atril automatizado está formado de varios elementos, que en conjunto dotan de características para la comodidad de los oradores que hacen uso del aparato. A continuación, se describen los elementos y las características del atril.

La *Figura 4* muestra que el atril en la parte superior de la base de sujeción de libro cuenta con una pantalla LCD para transmitir mensajes al orador, haciendo que la comunicación sea más discreta y evitando las interrupciones abruptas. A un lado, se encuentran un semáforo, de tres indicadores LED para mostrar el estado actual del atril, si está encendido, en uso o apagado. Una videocámara para la captura de fotografías, con el fin de calcular la altura de la persona que se encuentra frente al atril. Tanto el semáforo como la cámara están contenidas en una misma base, dotando de movilidad a la cámara. Del lado izquierdo a la base móvil de la videocámara y semáforo, se sujeta el micrófono de cuello de ganso. En la parte lateral inferior de la paleta, se cuenta con un imán que actúa con un sensor para indicar cuando debe parar de bajar la sección móvil, este movimiento se realiza a través de un motor de corriente alterna con su respectiva transmisión que puede deslizar en dirección hacia arriba o hacia abajo de una barra dentada (cremallera). Es importante mencionar que no solo la paleta del atril es la parte que se mueve (de arriba y/o abajo), sino

que también la superficie denominada pasa manos, creando una sección móvil, haciendo que sea una diferencia de otros atriles, que solo varían la altura del micrófono o de la base. Cuenta con dos controles manuales para manejar la altura, de lado izquierdo es para bajar y lado derecho para subir. En la parte frontal se ha fijado un sensor ultrasónico, que indica la presencia de una persona frente al atril, y a partir de aquí, comienza el funcionamiento de desplazamiento. El *driver* encargado del desplazamiento por la barra dentada es el de una ventanilla de carro. Todo lo anterior es controlado desde una *Raspberry Pi 3 B+*, colocada por la parte trasera de la base de sujeción.

En la *Figura 5*, se describen los elementos electrónicos necesarios para el atril automatizado. La fuente de voltaje V1 de 5V que se encarga de alimentar toda la parte de la circuitería de control. La pantalla LCD consta de varios pines que se deben conectar para su funcionamiento, pero para reducir este número de conexiones, se agregó un circuito integrado (CI) I^2 C, conectando solamente los pines SDA y SCL directamente a los puertos de la Raspberry, transmitiendo toda la información necesaria al módulo I^2 C y posteriormente a la pantalla. El semáfor es conectado a tres pines de la Raspberry. La videocámara fue conectada directamente al puerto MIPI CSI de la Raspberry. Dos conexiones más para el sensor ultrasónico. Los interruptores de control de la altura de la paleta del atril, no se conectan directamente al *driver* del motor de la ventanilla de coche, sino tiene que pasar primero a la *Raspberry*, debido a que el funcionamiento de dicho driver pasados más de veinte segundos con alimentación deja de funcionar, debido a las características del fabricante. El banco de relevadores consta de cuatro, dos son para señal de control del motor y los otros dos para control de encendido o apagado de la fuente de voltaje. Con la conexión intermediaria entre la Raspberry, el banco de relevadores, la fuente y el *driver*, se evita la limitación del fabricante que se mencionó. El funcionamiento de esta parte es el siguiente. Al presionar el botón subir o bajar, una señal es enviada a la Raspberry indicando la acción correspondiente, la Raspberry se comunica con el banco de relevadores, activando primero la fuente de voltaje que envía la señal de 120V CA, pasando por el banco de relevadores y después alimentando a la fuente, esta fuente convierte la entrada de 120V CA a 12V CA para el funcionamiento del motor, todo esto realizado en un segundo, a partir que se presionó uno de los botones de control. Pasado ese segundo, la Raspberry envía una nueva señal al banco de relevadores, ahora sí, indicando si es subida o bajada. En caso de que el sensor magnético esté activado (es decir, esté en el tope de la distancia de bajada), una señal es enviada al relevador indicando que debe detenerse, evitando daños físicos en la barra dentada o en el piñón del motor.

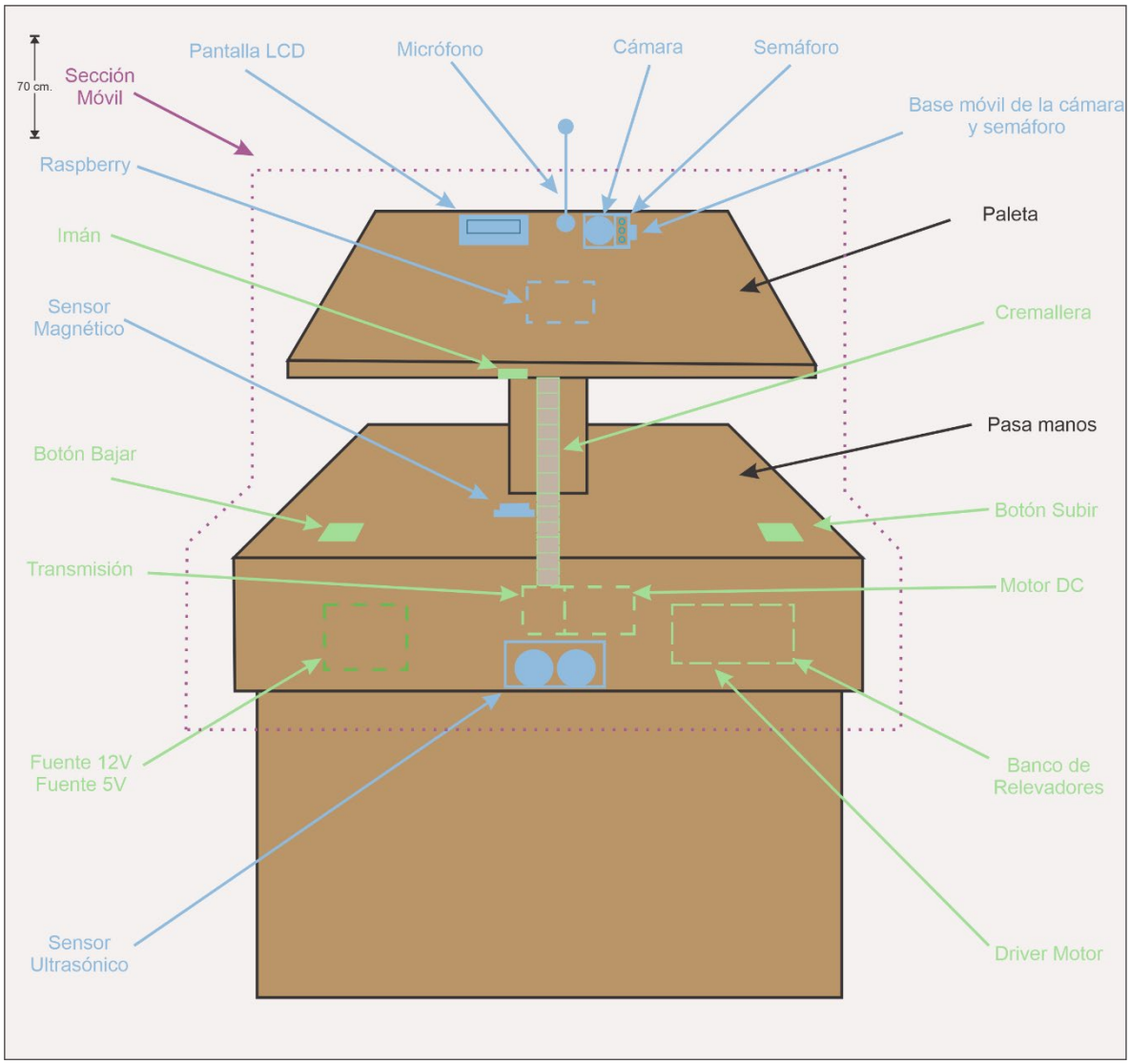

Figura 4: *Diagrama de las partes del atril automatizado donde se indica la separación móvil en color rojo, en color verde los componentes electromecánicos, y por último en color azul los componentes electrónicos (elaboración propia).*

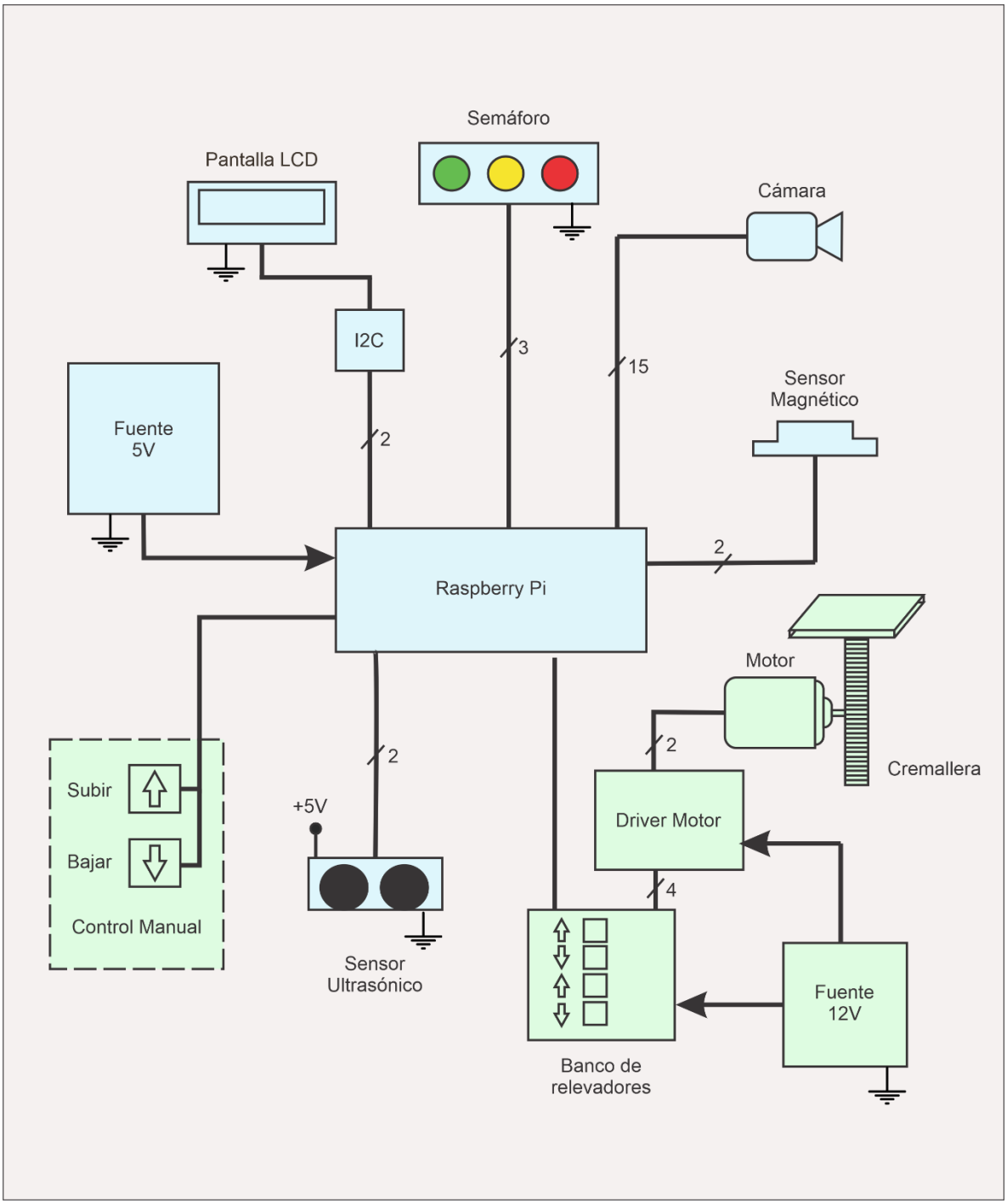

Figura 5: *Diagrama electrónico a bloques del atril automatizado. Los elementos de color azul corresponden a componentes electrónicos y los de color verde a componentes electromecánicos (elaboración propia).*

Ahora que se ha definido el objetivo para el desarrollo de esta tesis, el siguiente capítulo se centra en la metodología empleada para el desarrollo del algoritmo y su implementación. Es importante mencionar que esta tesis se centrará únicamente en realizar la detección facial con el *hardware* y las limitaciones que éste conlleva.

3. METODOLOGÍA

Como se ha mostrado en el capítulo anterior, la automatización de la altura del atril mencionado en este proyecto consiste en acomodar la paleta del atril en una posición acorde a la altura del expositor. En este capítulo se muestra la forma en que se hace la identificación de la región facial una vez capturadas las fotografías de la persona en distintos ángulos frente al atril y posteriormente obtener el valor de la coordenada más baja donde se detecte un cambió de intensidad para realizar el cálculo de la altura de la persona y con ello ajustar la altura de la paleta del atril.

3.1 Diseño del sistema de captura de imágenes

La captura de una imagen digital requiere de un soporte para sujetar la videocámara controlada por una *Raspberry Pi 3 B+*, las dimensiones de la videocámara son de 2592*x1944* pixeles y tiene una resolución de *5* megapíxeles, lo cual (*a priori*) resulta ser más de lo requerido para el algoritmo *Fonseca-Salas*. Dicho soporte, también cuenta con los orificios para el semáforo (LEDs indicadores de estado) como se muestra en la *Figura 6*.

3.2 Descripción del algoritmo para la detección facial

Ahora que se conoce el sistema físico que se encarga de realizar la captura de las imágenes, el algoritmo para el cumplimiento del objetivo principal: la detección facial, se divide en tres partes: *1) adquisición de la imagen*, *2) pre-procesado de la imagen* y *3) localización de la región facial* cada una de estas partes se describirán a detalle a lo largo de este capítulo.

3.2.1 Adquisición de la imagen

Para la adquisición de la imagen se desarrolló un *software* capaz de controlar el servomotor desde la *Raspberry Pi* con el fin de mover la videocámara en distintos ángulos una vez es fijada al soporte en la paleta del atril. El lenguaje de programación utilizado es *python* y consta de dos funciones: la primera llamada *moveServo* y sus parámetros de entrada son *1) número de grados* y *2) variable que controla al servomotor*.

Por otro lado, la segunda función se llama *getMiliseconds* se encarga de convertir el número de grados a mover el servomotor en milisegundos para posteriormente, generar un tren de pulsos que indica el ángulo que el servomotor debe girar para llegar a posición deseada, siendo así el único dato de entrada que recibe esta función.

Otra parte importante es la captura de fotografías en los distintos ángulos. Se desarrolló otro script en el lenguaje de programación *python* y usando la biblioteca de *OpenCV*, con la finalidad de controlar la cámara conectada a la Raspberry. La función llamada *takePhoto* se encarga de leer la información de la cámara, posteriormente la función *savePhoto* convierte la información en un archivo de fotografía digital, los parámetros de entrada para la función *takePhoto* son: *1) el nombre de la foto* y *2) la información obtenida de la cámara*.

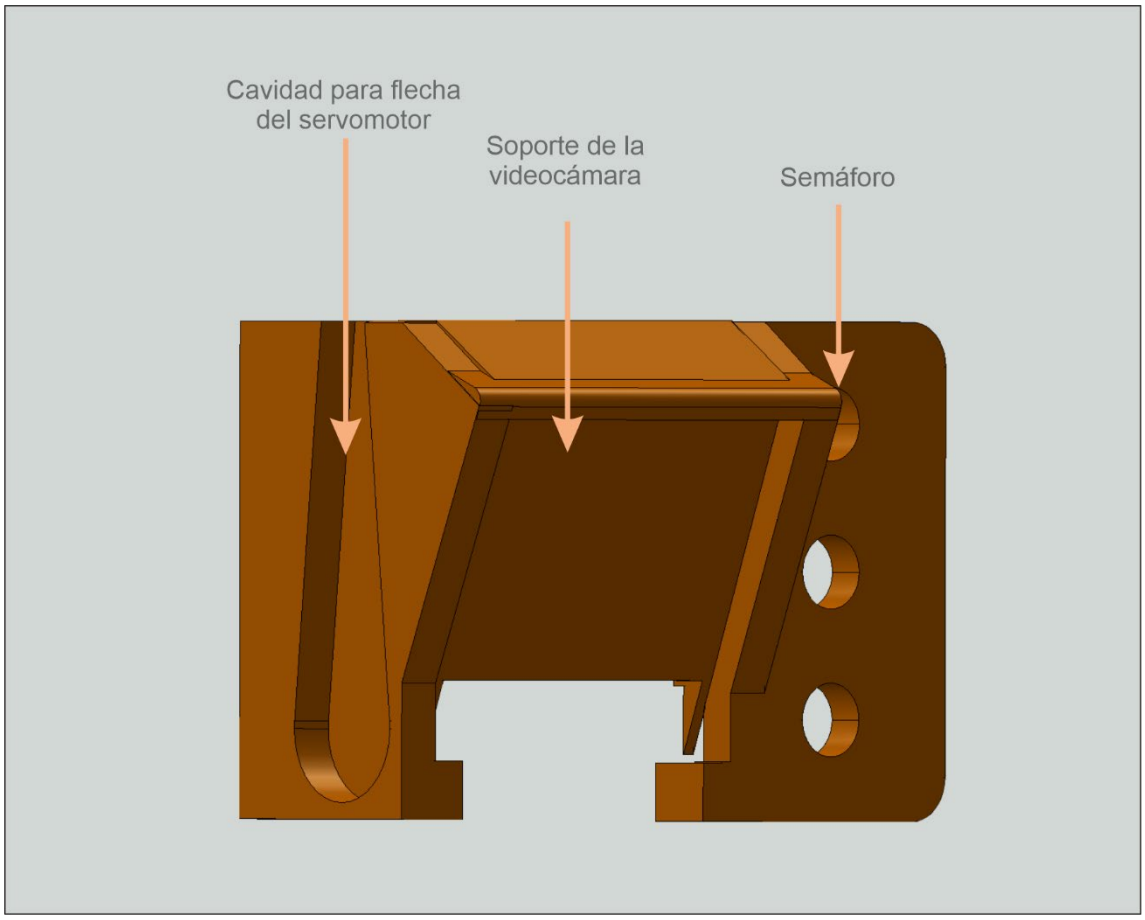

Figura 6: *Soporte para la videocámara y el semáforo. Del lado izquierdo se muestra la cavidad para flecha del servomotor, mientras que en la parte de en medio se muestra el soporte de la videocámara y por último se muestran las cavidades para los LEDs del semáforo (elaboración propia).*

3.2.2 Pre-procesado de la imagen

El pre-procesado de la imagen consiste en diferentes tipos de modificaciones por los que pasa la imagen, como parte previa a la localización de la región facial con la finalidad de reducir la cantidad de ciclos de procesador empleados durante su análisis. Algunas de estas modificaciones y las que se aplican a este trabajo son: *1) conversión a escala de grises*, *2) escalación de la imagen* y por último *3) la reducción de ruido*. Es importante recordar que, en esta etapa, no existe una serie de reglas que dicten qué se debe hacer primero y/o cuáles modificaciones se deben realizar, dependiendo así de la necesidad que se busca suplir y del desarrollador del proyecto.

Escalas de grises

Partiendo del proceso de captura de una imagen y una vez representada en tres matrices bidimensionales (por tratarse de una imagen de color) cada una con valores de intensidad distintos y en conjunto, forman una imagen a color. Al tratarse de tres matrices, cada una pertenece a un rango de color del *RGB*. Convirtiendo la imagen de color a una imagen en escala de grises, solo se obtiene una matriz con las intensidades entre *0* y *255*, así, en términos de procesamiento, se reduce el trabajo en *2/3* partes, además, el almacenamiento también es reducido, siendo menor para las imágenes en escala de grises.

La forma de convertir la imagen a color a una en escala de grises consiste en convertir la intensidad de un mismo pixel en cada una de las matrices que componen la imagen a color, multiplicado por un coeficiente de 0.299 para la matriz roja (R), 0.597 para la matriz verde (G) y 0.114 para la matriz azul (B), el resultado nos daría la intensidad en escala de grises equivalente al color de la imagen en cada uno de los pixeles *(UIT-R, 2011)*. La *Ecuación 3.1* representa la descripción anterior:

$$Y = \mathrm{R} \cdot 0.299 + \mathrm{G} \cdot 0.597 + \mathrm{B} \cdot 0.114 \tag{3.1}$$

donde el valor de R, G y B es la intensidad en cada una de la matriz correspondiente al sistema de color RGB. Afortunadamente, este tipo de cambio de color a escala de grises, en diferentes lenguajes de programación existe ya como una función.

Para realizar la conversión de la imagen a escala de grises, dentro la función *takePhoto* que se mencionó anteriormente, una vez que obtiene la información de la videocámara, hace uso de la función *cvtColor* de la biblioteca *OpenCV*, que recibe como parámetros la información del sensor de la videocámara (la imagen) y la variable *COLOR_RGB2GRAY*. La función nos devuelve la nueva imagen en escala de grises, posteriormente es guardada con formato *.jpg* así como se muestra en la *Figura 7*.

Escalación

Es una función que cambia el tamaño de la imagen y puede ser mayor o menor que la imagen de entrada, esto, regularmente se realiza con el objetivo de reducir el procesamiento de cómputo al tratar imágenes. Además, la pérdida de calidad está presente en el escalamiento de imágenes y depende del objetivo a cumplir si es importante o no dicha pérdida de calidad.

Una vez que la imagen es leída y convertida a escalas de grises, se realiza el escalamiento de la imagen. Primero se define el valor del largo y ancho para la nueva imagen. En este trabajo se eligen los valores *1280x720* pixeles (largo por ancho) y se asigna a un arreglo dimensional con dos posiciones, por último, se usa la función *resize* que recibe como entrada la imagen a escalar, el arreglo dimensional con los nuevos valores de ancho y largo de la imagen y el tipo de interpolación para la imagen definida como *cv2.INTER_CUBIC*, definida previamente en la biblioteca de *OpenCV*. La *Figura 8* ejemplifica la entrada y salida de esta función.

Figura 7: *Salida de la función takePhoto. a) muestra una imagen a color que se convierte a escala de grises y b) muestra la imagen convertida a escala de grises (elaboración propia).*

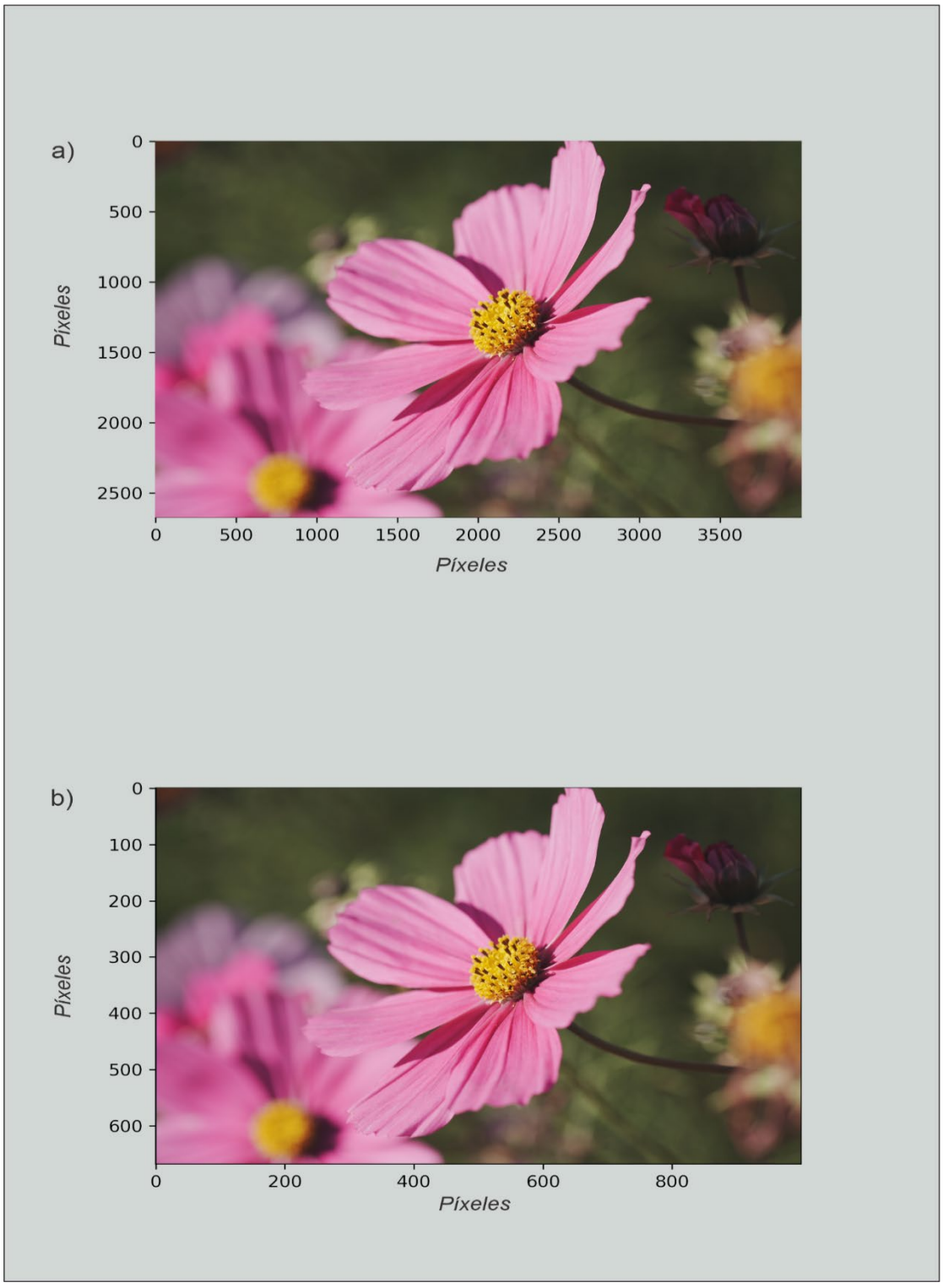

Figura. 8: *Escalación de la imagen dentro de la función takePhoto. a) posee una resolución de 10.6 megapíxeles, su tamaño es de 4000x2672 pixeles. Por otro lado, b) muestra la misma imagen, pero en resolución de 2.6 megapíxeles con tamaño de 1000x668 pixeles. Fue reducida en un 75% con respecto a la original (elaboración propia).*

Reducción de ruido

El ruido se refiere a cualquier señal no deseada, por lo tanto, afectan el producto final, quitando calidad a la imagen *(Villa y Yáñez, 2017)*. Esta calidad reducida se visualiza como puntos blancos en ciertas secciones de la imagen (o en toda la imagen), produciendo un efecto borroso. El ruido no puede ser eliminado, solo atenuado.

Algunos filtros para la reducción de ruido consisten en definir una máscara (una región de *NxM pixeles*) para operar en toda la imagen y obtener un resultado mejor que la imagen de entrada. Los valores para operar son extraídos de los pixeles vecinos en relación con otro píxel, el tamaño de la máscara puede ser variable. Las operaciones más comunes que se pueden realizar en los filtros son:*1) media* y *2) gaussiano (Gonzalez y Woods, 2018)*.

Se implementó una función nombrada *noiseReduction* que consiste en aplicar las 3 operaciones a filtros para evaluar cuál de ellas es la que mejor resultado entrega. Retomando la salida de la función *takePhoto*, es decir una vez convertida a escala de grises y escalada a un tamaño menor se le inyecta ruido *Gaussiano* a la imagen con la función de *OpenCV blur*, recibiendo como parámetros la imagen de entrada, el modo *gaussian, seed=None* y *clip=True* (esta etapa solo es ejecutada para la elección de la operación de filtro, no se incluirá en la versión final del algoritmo). El retorno de la función *blur* es la imagen con ruido Gaussiano y sirve como entrada para la función *blur* nuevamente pero solo agregando los parámetros imagen y tamaño del filtro que será de 3x3. La función para el filtro Gaussiano es Guassianblur recibiendo como parámetros *la imagen, el tamaño del filtro y un número 1* indicando que el filtro es Gaussiano. La *Figura 9* muestra el resultado de los dos filtros propuestos.

Figura 9: *Comparación del filtro de la media y gaussiano. a) muestra la imagen original y b) muestra la misma imagen, pero con ruido gaussiano, para simular que fue tomada a baja luz, creando los puntos blancos. Tanto c) y d) muestran los dos tipos de filtros usados: gaussiano y media respectivamente y ambos con una máscara de 3x3 pixeles. El filtro de la media muestra el mejor resultado para la reducción del ruido (elaboración propia).*

3.2.3 Resta de imágenes

Este proceso consiste en realizar una resta de imágenes entre el fondo y el sujeto, con el fin de obtener solamente la silueta del sujeto con ayuda de la función *substract*. Anteriormente, se mencionó que las imágenes son representadas como matrices numéricas, donde cada valor representa una intensidad. Matemáticamente, si se realizan las restas entre ambas imágenes, los valores idénticos se convierten en *0s*, así la imagen resultante es una matriz, con la mayoría de sus intensidades cercanas a *0* ya que las intensidades no son exactas en ambas fotos, no siempre se logra este valor exacto, excepto la zona de la persona en la imagen, donde sus intensidades varían unos cuantos valores menos, pero sin modificar la imagen original. El resultado gráfico es la silueta de la persona, con un fondo casi totalmente negro como se muestra en la *Figura 10*.

3.2.4 Definición de las trazas en la imagen

Ahora que se tiene la resta de las imágenes de la persona, reducida en tamaño y filtrada de ruido, se define un número de líneas a lo ancho de la imagen llamadas trazas. Dichas trazas sirven como referencia para ubicar la región facial. Este número varía entre 3, 5, 7 y 9 con el fin de reducir el tiempo de ejecución y tener un umbral de la posición de la persona en la fotografía. Antes de definir dicho número, ayuda a que el algoritmo reduzca los ciclos repetitivos que deben ejecutarse para encontrar la primera variación de intensidad, esta variación debe encontrarse en cada una de las trazas definidas, a mayor número de trazas mayor rango para detectar la región facial pero mayor número de ciclos de ejecución. La distancia en pixeles entre cada una de las trazas es de 50, la *Figura 11* muestra gráficamente las trazas en una imagen. Así se asegura que por lo menos tres puntos de interés de la región facial son detectados y se consideran suficientes para lograr el objetivo, que es detectar la ubicación de la parte superior de la cabeza.

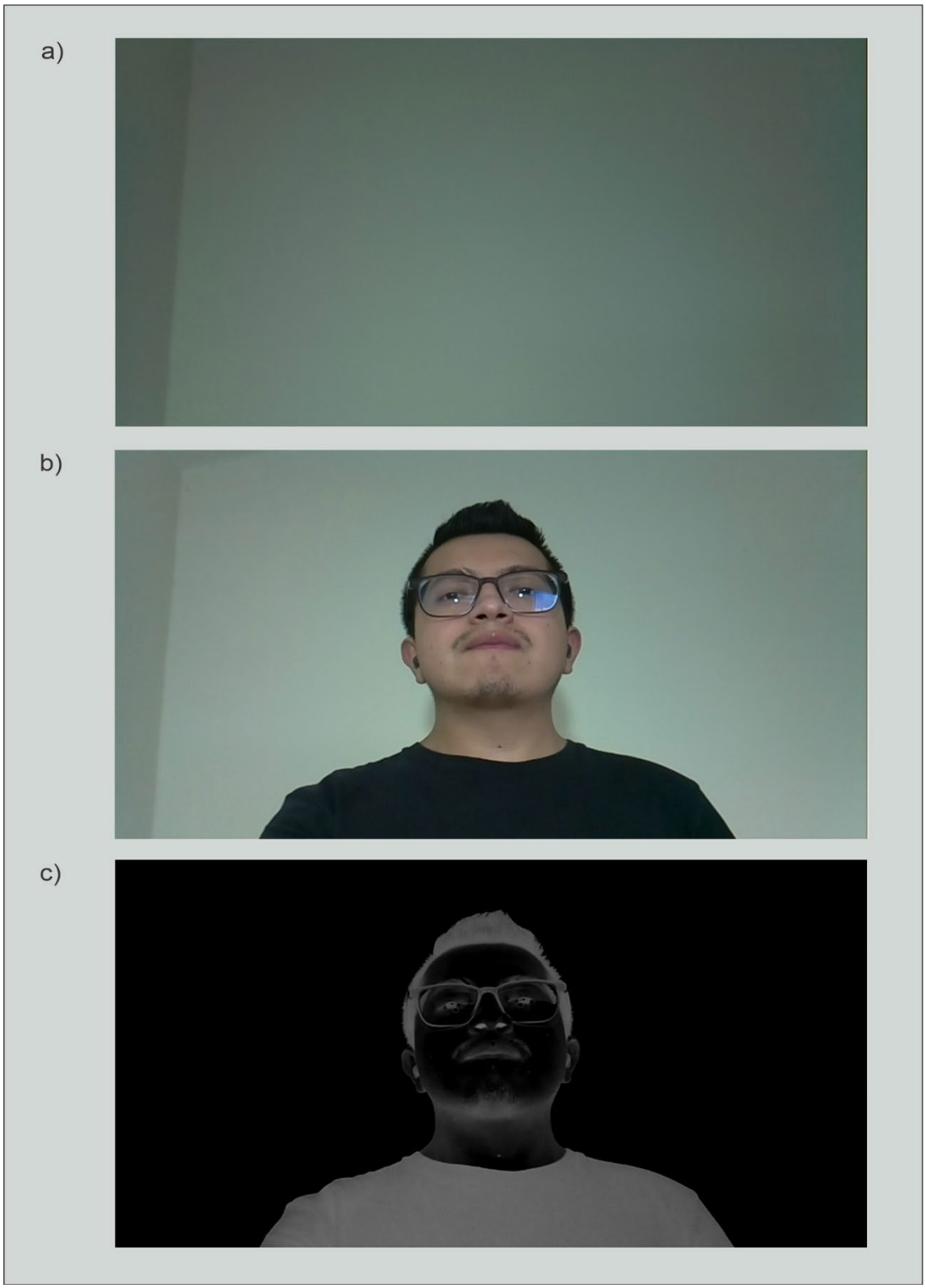

Figura 10: *Imágenes de entrada para la función substract. a) es la imagen de fondo mientras que b) es la imagen de la persona y c) es el resultado de la resta de las dos imágenes a través de la función mencionada (elaboración propia).*

Para la definición de las trazas en la fotografía se parte de la mitad de su tamaño. Se sabe que el rostro no es simétrico, al menos para la mayoría de las personas, quizá un ojo es minúsculamente más pequeño que el otro, o alguna de las fosas nasales a pesar de ser redondas, quizá su excentricidad (desde un punto de vista geométrico) es mayor que el de la otra. Una simetría como la del rostro, a grandes rasgos es suficiente para saber que lo que hay de un lado debe estar en el otro. A partir de esta traza a la mitad de la foto, se añaden una traza a cada lado. Dado que el ancho de la foto es de *1280* píxeles, su mitad es *640* píxeles. Antes y después de esta traza definida, se agregarán las nuevas trazas con la distancia de *50* píxeles (definido en el párrafo anterior), por lo tanto, para que algoritmo funcione con *3* trazas, los valores son: *590*, *640* y *690* pixeles, con *5* trazas los valores: *540*, *590*, *640*, *690* y *740* pixeles, para *7* trazas los valores son: *490*, *540*, *590*, *640*, *690*, *740* y *790* pixeles, y así sucesivamente como se muestra en la *Figura 11.*

3.2.5 Localización de la región facial

Entre la literatura, destacan algunos métodos y algoritmos que son muy conocidos, por ejemplo, el algoritmo *Viola-Jones* que consiste en unos clasificadores de cascada que identifican distintas características del rostro en una serie de iteraciones hasta agotar dicho número de características *(Viola y Jones, 2004)*. De primer vistazo, es un algoritmo funcional pero no se puede decir que eficiente en términos de procesamiento, ya que hacerlo en tiempo real demanda una gran cantidad de los recursos de una *Raspberry*.

El algoritmo *Viola-Jones* se basa principalmente en la comparación de características de los rostros de personas. Este trabajo no tiene interés en todas las características que el algoritmo *Viola-Jones* puede comparar, ya que es de interés la parte superior de la cabeza. *OpenCV* es una biblioteca de código abierto para visión de computadora y *Machine Learning.* Cuenta con una serie de funciones para el uso del algoritmo *Viola-Jones* incluidos los clasificadores de cascada y el también reconocido algoritmo *AdaBoost* para elección de características. Al tratarse de un algoritmo repetitivo y que depende de características, si se reduce el número de características las repeticiones también se reducen, así como el número de operaciones del procesador.

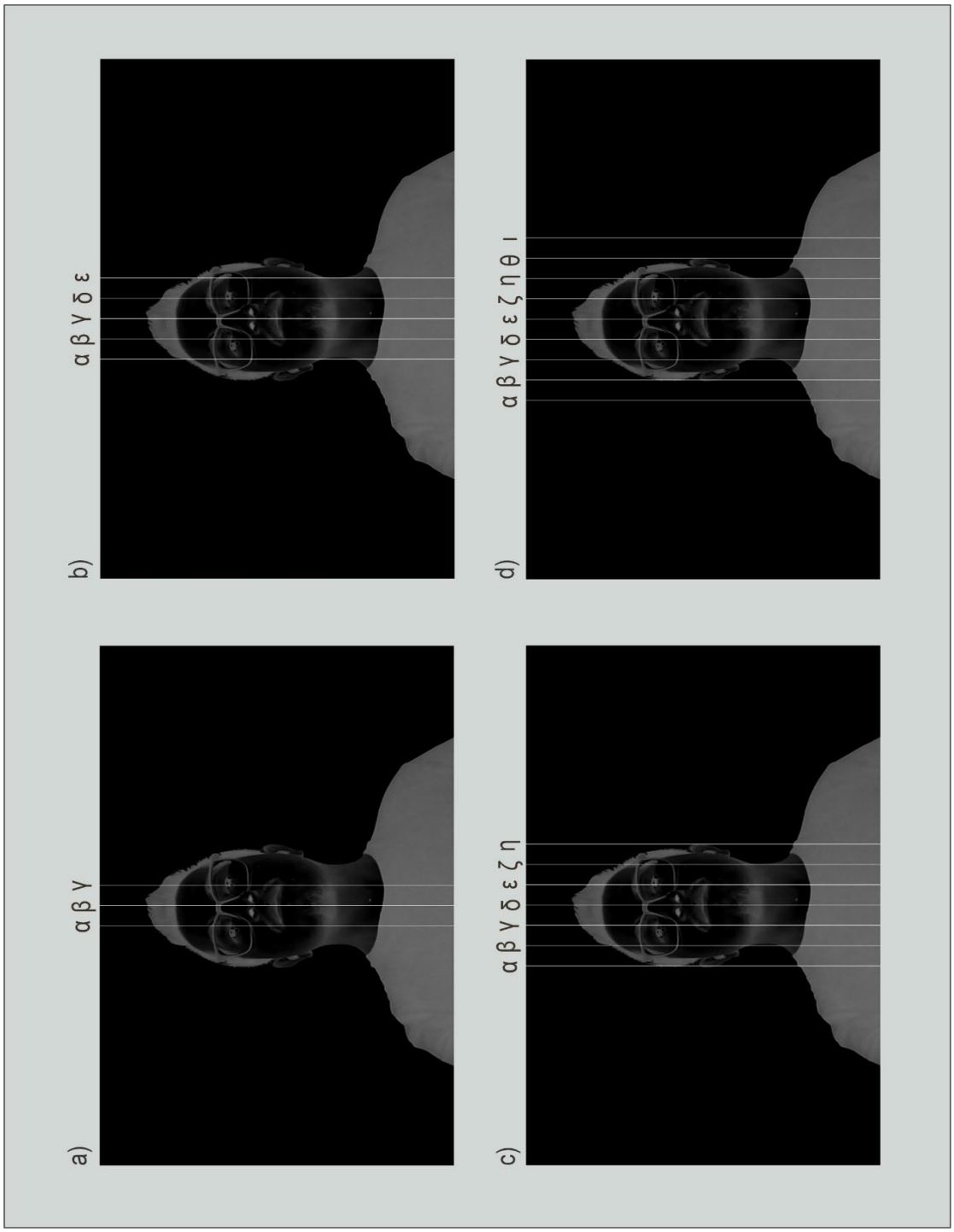

***Figura 11**: Una imagen de 1280x720 píxeles mostrando las trazas en color blanco. a) muestra 3 trazas representadas por α, β y γ. b) 5 trazas representadas por α, β, γ, δ, y ε. c) muestra 7 trazas representadas por α, β, γ, δ, ε, ζ y η. d) muestra 9 trazas representadas por α, β, γ, δ, ε, ζ, η, θ, y ι. En todas las trazas se definió la nomenclatura del alfabeto griego para una fácil identificación de dichas trazas, además esta nomenclatura se hizo de izquierda a derecha (elaboración propia).*

La solución que se quiere implementar consiste en generar un par de fotografías que son tomadas en cuanto la persona se posicione frente al atril, además de una serie de fotografías previas con lo que sea que haya de fondo detrás de la persona. Una vez que las imágenes son obtenidas, escaladas y convertidas a escala de grises, se realiza una resta de entre las imágenes, es decir el fondo menos la persona, esto se logra con la función *subtract* de *OpenCV* que recibe como parámetro la imagen de la persona y la de fondo, regresando como resultado la persona en un fondo negro y sin objetos como se muestra en la *Figura 11*. Ahora se procede a realizar una comparativa de cambios de intensidades para ubicar el borde de la figura de la cabeza.

La detección del cambio de intensidades se realiza a través de la operación de la *Ecuación 3.2*:

$$\frac{imagen[i]}{i} \tag{3.2}$$

donde *i* es una posición dentro de la imagen e *imagen[i]* es el valor de la intensidad de la imagen en dicha posición, el resultado de esta operación deberá ser mayor al seno de *7°,* lo anterior mencionado cumple con un cambio drástico de las tonalidades para cada uno de los pixeles además que ese valor es obtenido cuando los valores de intensidades en escala de grises permanecen por debajo de *40* y que pertenecen a nuestra área de interés de la región facial.

Ahora, definido el uso de las trazas para el algoritmo *Fonseca-Salas*, la detección de cambios de intensidad en cada una de las trazas se realiza a través de un ciclo repetitivo *for*, y una función llamada *getCoordinates* que recibe como parámetros de entrada la traza completa de la foto (como si fuera un vector unidimensional) y el número de la traza en píxeles. La traza recibida es recorrida para evaluar las intensidades que se consideran apropiadas como variaciones, esta explicación se explica en la *Figura 12*.

3.3 Cálculo de la altura de la persona

Una vez que se obtienen las fotografías y terminado el procesamiento, el siguiente paso es calcular la altura de la persona frente al atril. Primero se necesita calibrar los valores en centímetros que cada imagen puede capturar, es decir, una fotografía en la posición inicial del soporte de la cámara en *X* grados captura una altura de *Y* centímetros. Dicho valor *Y*, cambiará a medida que los ángulos incrementan o se reducen. Se conoce que la distancia de la videocámara a la persona frente al atril es de *60* centímetros como se muestra en la *Figura 13*.

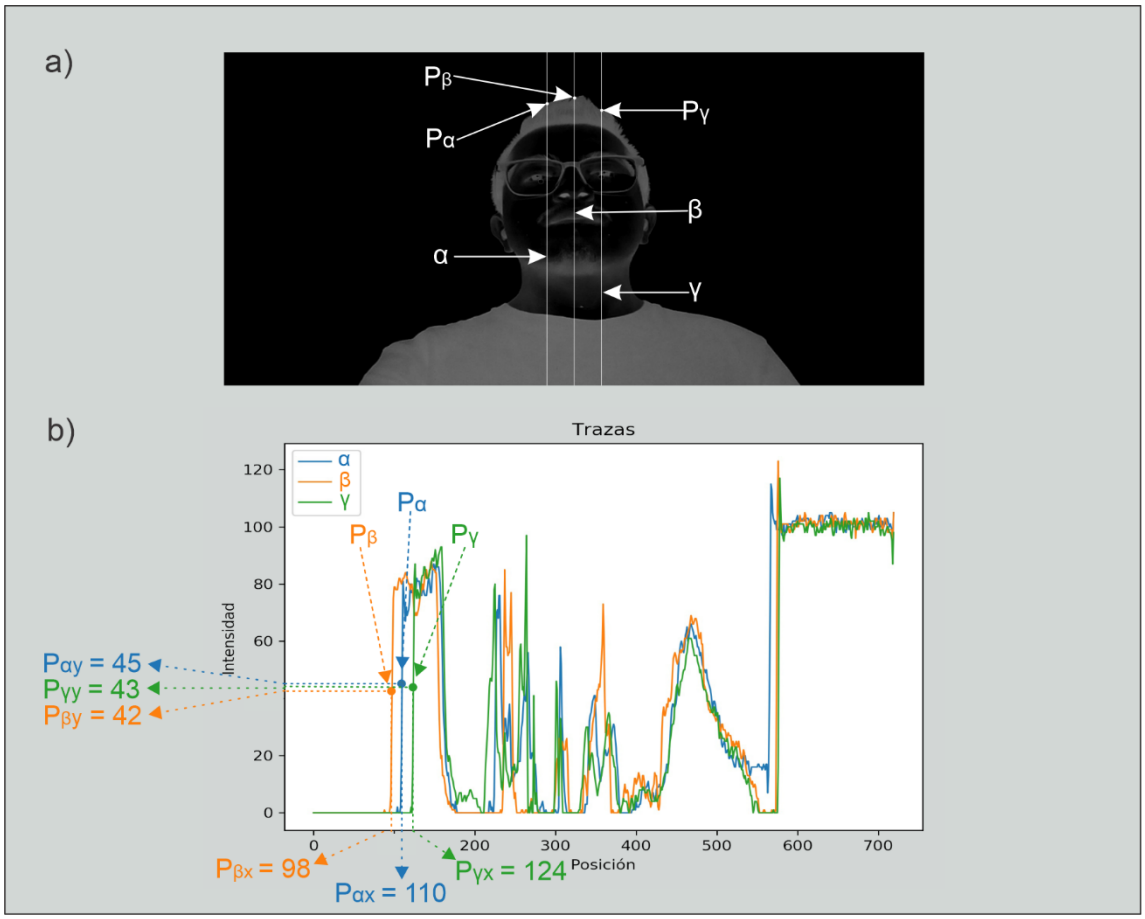

***Figura 12**: Representación gráfica del cambio de intensidades para el algoritmo de detección facial. a) muestra las trazas α, β y γ y los puntos de interés del algoritmo Pα, Pβ y Pγ respectivamente. Por otro lado, b) representa la gráfica de intensidades de cada una de las trazas α, β y γ así como los puntos de interés Pα, Pβ y Pγ (elaboración propia).*

Como seguimiento al proceso de calibración, se realiza también una serie de equivalencias entre las medidas obtenidas de las imágenes, dichas relaciones serán entre pixeles y el número de centímetros que se obtuvieron para cada fotografía. Una vez dada la coordenada donde el cambio de intensidad sucede primero, se conocerá la altura de la persona. Para realizar este cálculo se tienen las siguientes variables: *1)* valor inferior en la imagen tomada en los distintos grados (*90°, 97.5°* y *105°*) representada como $val_{\inf}$, *2)* la distancia del suelo hasta el valor inferior de la imagen tomada representada con una d , *3)* valor de la relación *píxel-altura* representada como rel_{pa} (se explica en el siguiente párrafo) y *4)* valor de la componente y de la coordenada más baja, ahora poniendo todo en la *Ecuación 3.3* , nos queda:

$$(d + val_{\inf}) + (rel_{pa})(720 - y) \tag{3.3}$$

La resta de $(720 - y)$, es realizada ya que el algoritmo comienza a contar los pixeles de arriba hacia abajo, siendo *1* el valor más alto y *720* el menor. Con la *ecuación 3.3* se obtiene la altura de la persona.

Se concluye que para esta configuración entre ángulos de enfoque de la cámara (es decir, posición) y resolución de las imágenes tomadas es de *1280x720* pixeles. Siendo más explícitos, para cada *x* centímetros corresponde a *y* píxeles en las imágenes. Por ejemplo, si se tiene una imagen de una persona con una resolución de *15* megapíxeles (*4475x3351 pixeles*), situada a *60* centímetros y que la fotografía captura de su barbilla a su pecho y que esta distancia es *55* centímetros. Se asume que para saber el número de píxeles que corresponde a cada centímetro, se obtiene a través de una simple división $\frac{no.pixeles}{centimetros}$ esta relación se nombra *pixel-altura* y que es un dato de entrada para obtener la altura de una persona que cumpla con las condiciones iniciales, es decir, una persona que se sitúe frente a la cámara del atril a una distancia de 60 centímetros del atril.

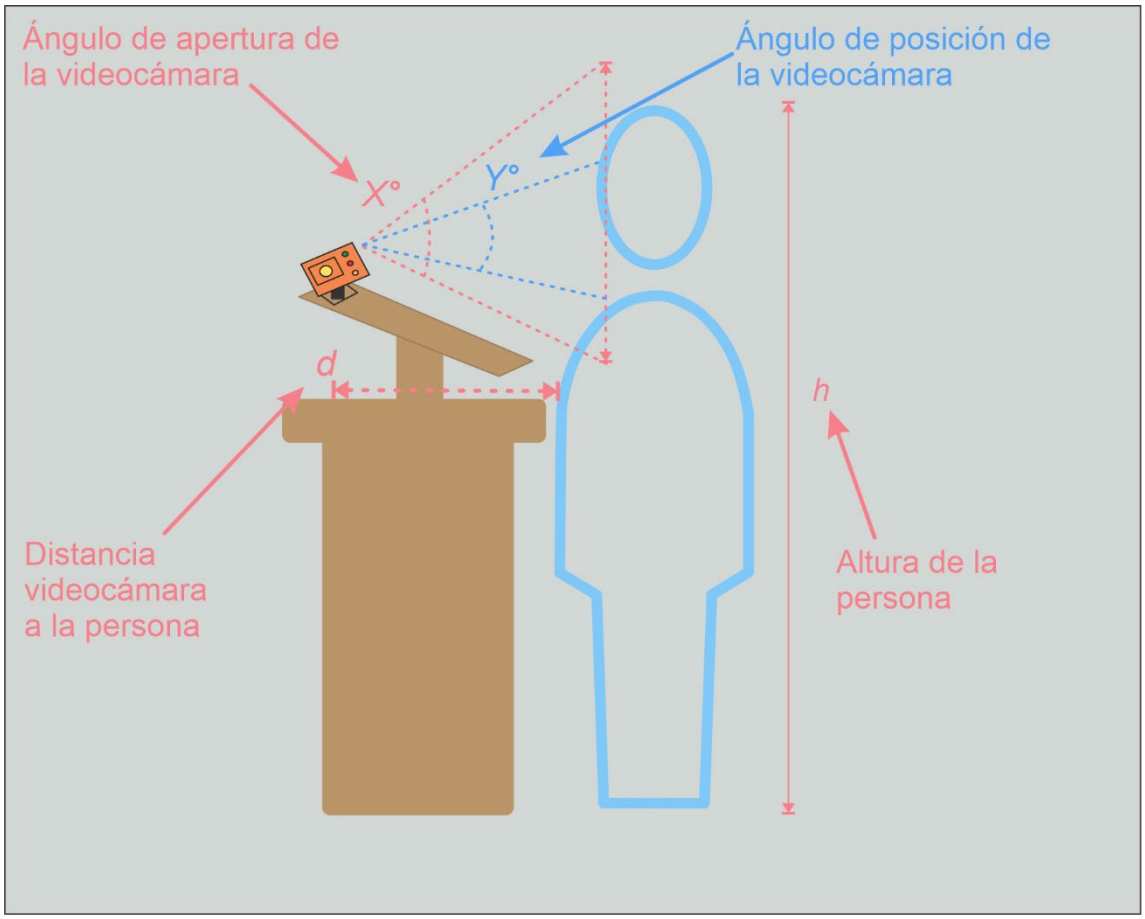

Figura 13: *Representación de la calibración de la videocámara para el cálculo de la altura de la persona. Dicha calibración debe hacerse en 3 diferentes ángulos (elaboración propia).*

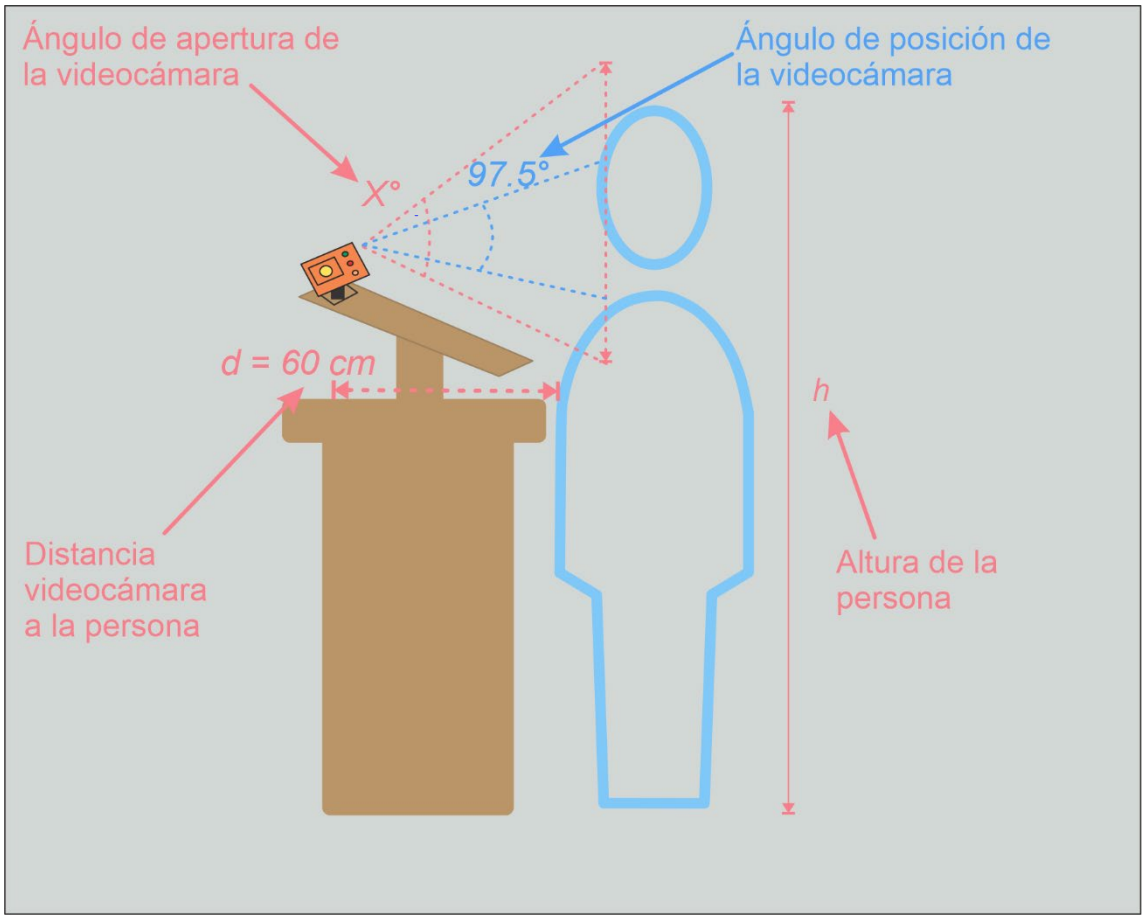

Figura 14: *La figura muestra las condiciones iniciales a las que la fotografía es tomada: distancia de la cámara al sujeto y la cantidad de centímetros que la videocámara puede captar a esa distancia y el ángulo de la posición de la videocámara (elaboración propia).*

Una vez obtenida la foto será procesada por el algoritmo *Fonseca-Salas* que se describe en la siguiente sección, regresa la coordenada del pixel donde hay un cambio drástico de la intensidad, la componente y del pixel será multiplicada por la relación *píxel-altura* y así se obtiene la altura final de la persona de la fotografía.

3.4 Desplazamiento de la sección móvil

El último paso consiste en desplazar la sección móvil del atril, con el fin de lograr una ergonomía para la persona que haga uso de este. Además de dichas mejoras para el expositor, cuenta con funciones extras que se otorgan en este dispositivo como la pantalla de LCD para recibir mensajes sin interrumpir sus exposiciones.

Una vez obtenida la altura de la persona, se debe activar el motor para realizar el desplazamiento a través de la cremallera vertical. El funcionamiento del motor es recibir una señal durante *N* segundos para comenzar a trabajar, es decir, para subir o bajar la sección móvil. Dado que se busca alcanzar una posición ergonómica para el ponente, este valor varía en función de la altura del ponente. Para calcularlo solo se realiza una simple sustitución en la *Ecuación 3.4:*

$$v = \frac{d}{t} \tag{3.4}$$

donde el tiempo es la incógnita, una vez obtenido, se usa como parámetro de entrada para activar el controlador del motor y obtener la altura requerida.

Aunque en este trabajo no implementa la función que realiza el desplazamiento de la sección móvil, es importante comentarla ya que el insumo para una función que realice lo anterior dicho es generada en este trabajo. El siguiente capítulo muestra las pruebas realizadas del algoritmo mostrado en este capítulo, también se comparará el consumo de CPU y el tiempo durante su ejecución comparado con un algoritmo común (*OpenCV*).

4. RESULTADOS

En este capítulo se presentan los resultados de la creación del sistema de captura de imágenes para la detección facial y su instalación en el atril automatizado que se abarca en la sección 4.1, también se documentan las pruebas que se realizaron con el algoritmo *Fonseca-Salas* una vez completado el diseño y la fase final de desarrollo (sección 4.3). Posteriormente, en la sección 4.4. se detallan las pruebas de la implementación de dicho algoritmo, es decir, ya ejecutándose en la Raspberry del atril automatizado y su correspondiente videocámara. Otra pieza importante y final para lograr el objetivo es la implementación de la función para cálculo de la altura de la persona, que se abarca en la sección 4.4. Las pruebas de funcionamiento del atril automatizado con la movilidad de la sección móvil también son mostradas aquí y, por último, las pruebas de rendimiento del algoritmo implementado que se muestran en la sección 4.6.

4.1 Instalación del sistema de captura de imágenes

Como se vio en el subcapítulo *3.1 diseño del sistema de captura de imágenes*, se ha diseñado un dispositivo capaz de sujetar una videocámara de una *Raspberry Pi* y que a su vez la mueva en distintos ángulos horizontales. Para conseguir este movimiento se usa un servomotor *Tower Pro SG5010,* sujetado a la paleta del atril mediante una base de PLA, propiamente diseñada para que no obstruya la movilidad de la videocámara.

Para conseguir el movimiento, a su lateral izquierdo se le agregó una perforación para encajar una flecha de servomotor, dicho servomotor debe estar sujeto a la superficie de la paleta del atril y así realizar el movimiento del soporte completo, es decir, soporte de la videocámara y semáforo.

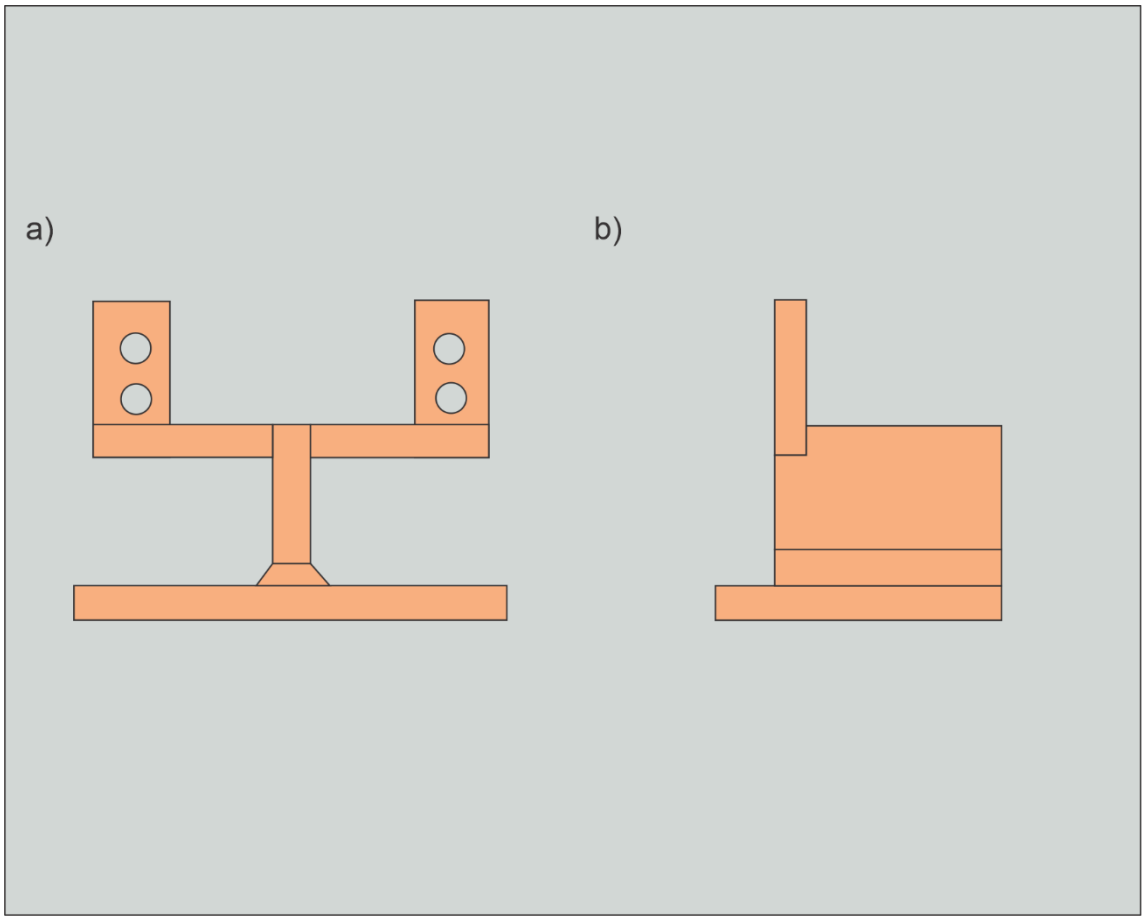

Figura 15: *Los planos de la base de sujeción para el servomotor. a) Parte frontal de la base de sujeción donde el servomotor es fijado. b) Vista lateral de la base de sujeción (elaboración propia).*

Para colocar la base de sujeción del servomotor, el soporte de la videocámara y la videocámara fueron sujetados con tornillos en la parte superior a la madera de la paleta del atril. El servomotor también fue fijado con tornillos a su base.

Figura 16: *Fotografía de la base de sujeción del servomotor y soporte de la cámara con cavidades para semáforo una vez colocado en la paleta del atril automatizado (elaboración propia).*

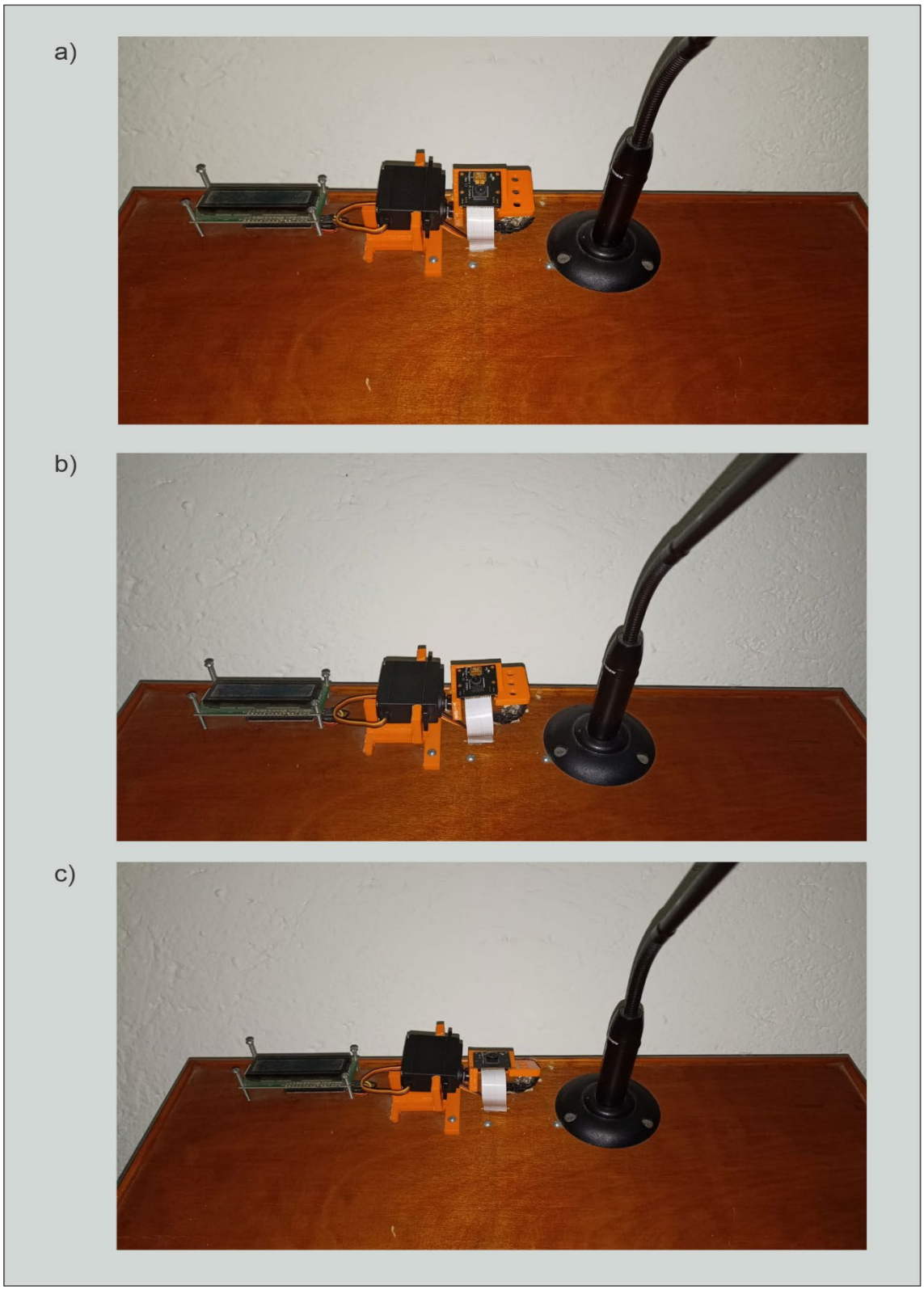

Figura 17: *Fotografía que muestra la base de sujeción y base de la cámara con semáforo, donde la cámara está posicionada en diferentes ángulos θ. Cámara posicionada a a) 90° b) 97.5° y 105° (elaboración propia).*

Fig. 18: *Fotografía tomada con el software creado para captura de imágenes con las Raspberry y la videocámara con resolución 5 Megapixeles (2592x1944) (elaboración propia).*

Con los componentes de *hardware* y de *software* para la captura de imágenes, se procede a realizar las pruebas del algoritmo de detección facial.

4.2 Algoritmo de detección facial

Como se ha escrito en los capítulos anteriores, la necesidad de crear un algoritmo de detección facial radica en la reducción del consumo de CPU del algoritmo de *Viola-Jones* y del tiempo de ejecución, además de que las características evaluadas por dicho algoritmo no son necesarias para la tarea del cálculo de la altura de una persona frente al atril.

Es importante recodar los pasos que se describen en la sección *3.2.2 Pre-procesado de la imagen* se encarga básicamente de tratar la imagen, a modo que se reduzca las señales no deseadas en las fotografías, así como escalar la imagen y convertirla a escala de grises. El siguiente paso.

De la *Figura 11* se comprueba que, a mayor número de trazas, el rango para detectar la región facial también es mayor. Por lo tanto, se considera que siete trazas son ideales en esta aplicación ya que encajan con el ancho de la cabeza de una persona. Recordando que la detección facial del algoritmo *Fonseca-Salas* para este proyecto debe considerar el tiempo y el uso de CPU durante la ejecución.

En la sección *3.2.5 Localización de la región facial* se menciona la ejecución de un ciclo para cada una de las trazas definidas en la fotografía. Si se grafican las trazas de la imagen que se utilizan para el análisis del algoritmo de detección facial *Fonseca-Salas*. Un ejemplo de este paso se muestra en la *Figura 19*, donde se aprecia una gráfica con distintas líneas de colores, cada color pertenece a un número de traza. El primer cambio de intensidad tiende a estar por arriba del valor *40* de intensidad, próximo a un pico en la gráfica en cada una de las líneas de colores. También se deduce que los puntos cercanos a dichos picos en las gráficas representan el comienzo de la región facial a localizar, además estos puntos de interés representados como coordenadas dentro de la imagen, son el resultado que se pretende encontrar como se muestra en la *Figura 20*.

Las trazas se definieron a partir de una comparativa de distintas fotografías tomadas, y se concluye que aquellas intensidades mayores a *40* son las más cercanas a la región facial de una persona. Posteriormente, se obtiene el valor del cambio de intensidad, evaluando la intensidad de la imagen en cierta posición dividida entre su número de posición (*ecuación 3.2*), el cual debe ser mayor al *seno de 7°* para considerarlo acorde a nuestras necesidades, una vez obtenido, el valor de la posición en que se encuentra este cambio de intensidad es regresado a través de la función, logrando así la obtención de la coordenada. Se asegura que el primer cambio de intensidad es el resultado, ya que cuando se realiza la sustracción de las imágenes, la mayoría de los valores de intensidad son cercanos a *0*, nuestras intensidades a evaluar solo son mayores a *40*, dada la lejanía entre intensidades, el valor esperado representa un punto de la región facial buscada.

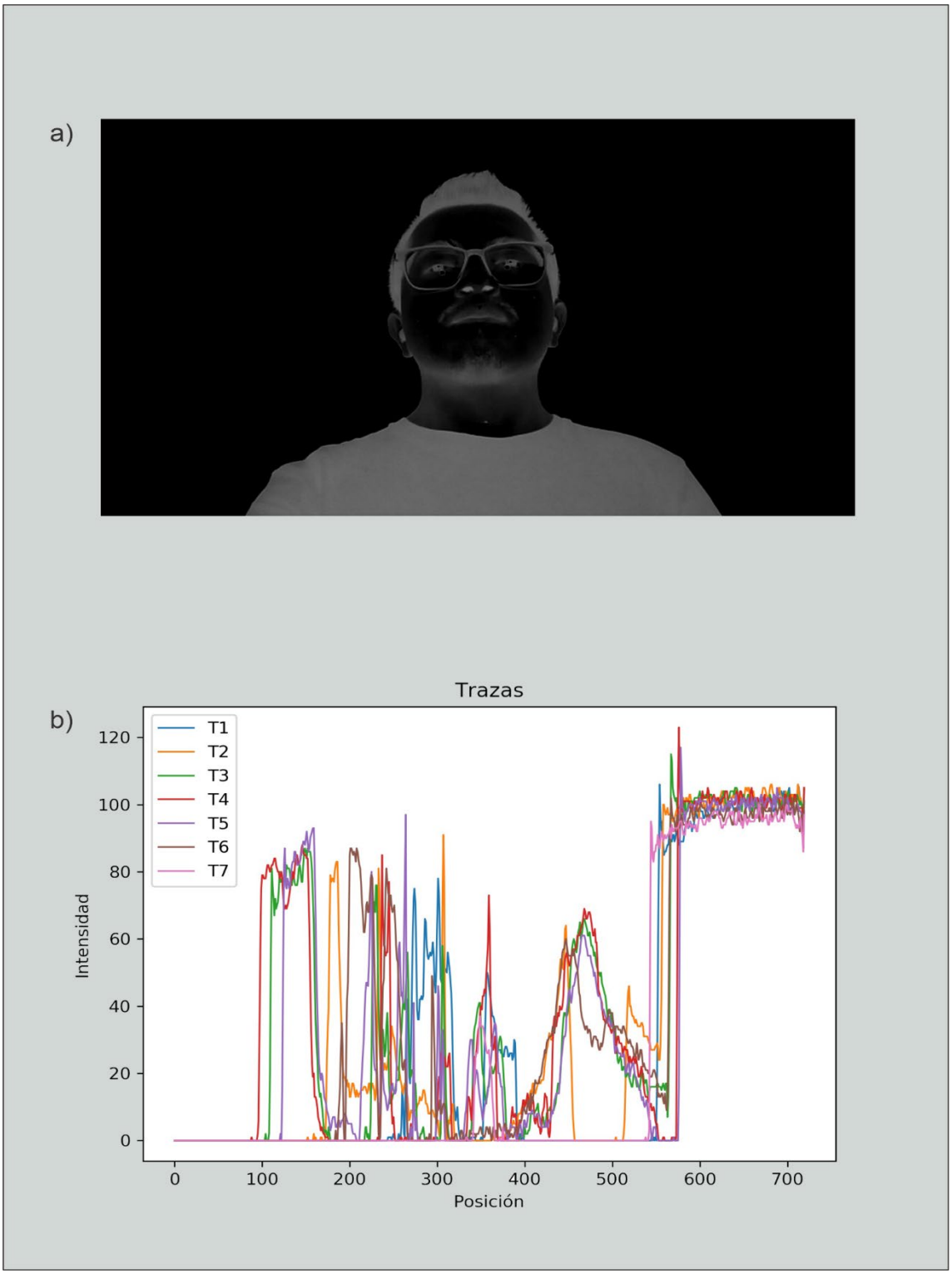

Figura 19: *a) muestra la imagen que se evaluó con 7 trazas, mientras que b) es una gráfica que muestra los valores de intensidad de las trazas que se analizan en el algoritmo de detección facial (elaboración propia).*

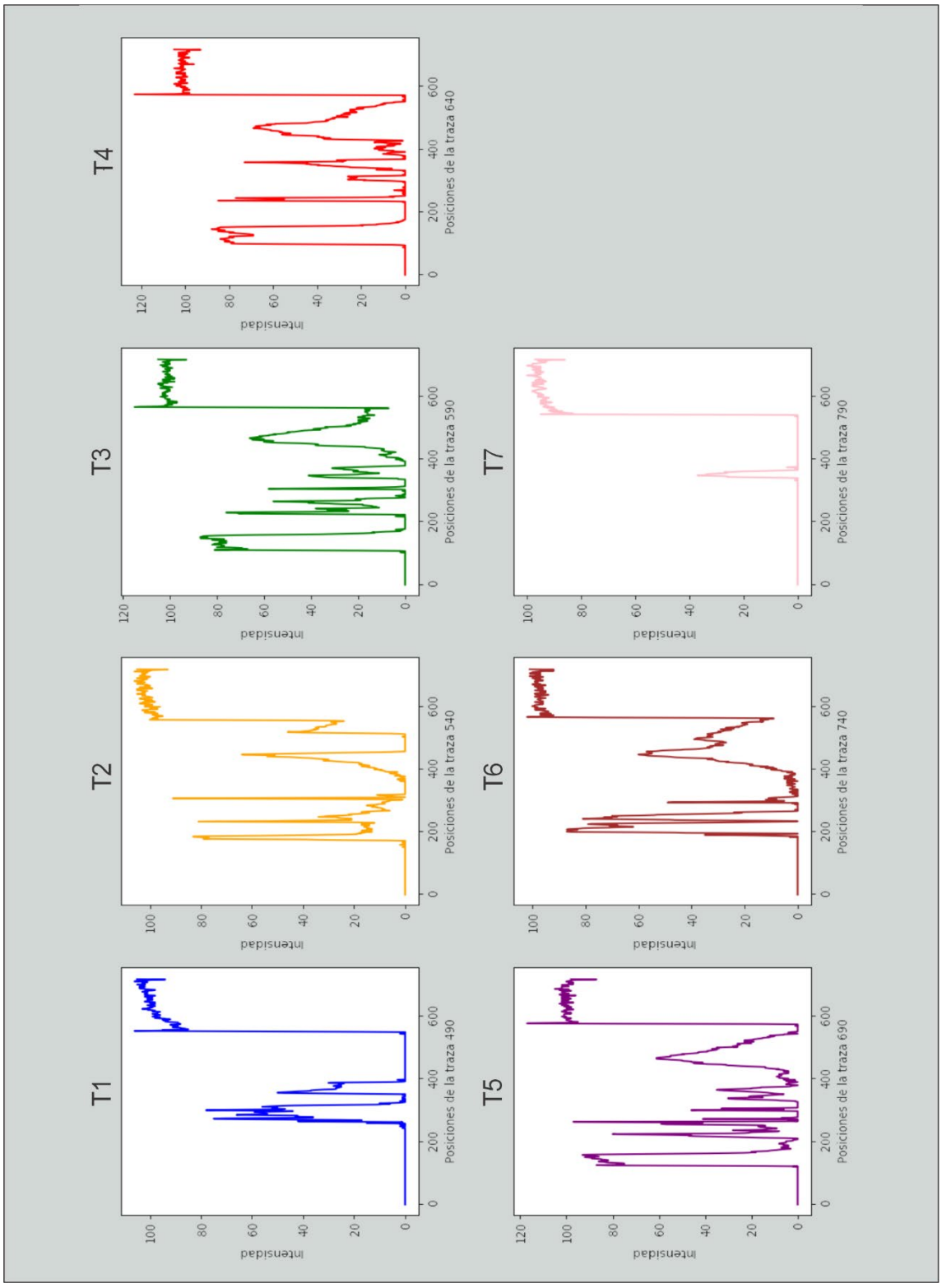

Figura 20: *Las trazas graficadas de forma individual, representando las intensidades para cada traza. Las intensidades que tienden a ser más oscuros son representadas por un 0, por otro lado, las intensidades claras son representadas con un valor cercano a 100 (elaboración propia).*

Con fines representativos, se genera una nueva imagen (*Figura 21*) que muestra las coordenadas donde hay cambios de intensidad y pertenecen a la región facial que se quiere ubicar en la imagen analizada. Al ser un conjunto de coordenadas, se agregan unos marcadores a la imagen resultante para representar gráficamente donde fueron ubicados los cambios de intensidad. Es importante mencionar que el algoritmo también genera un archivo de texto con los valores de las coordenadas además de otros datos de interés para esta investigación. Por otro lado, la *Figura 22* nos muestra una gráfica para cada traza con un marcador donde la intensidad fue encontrada.

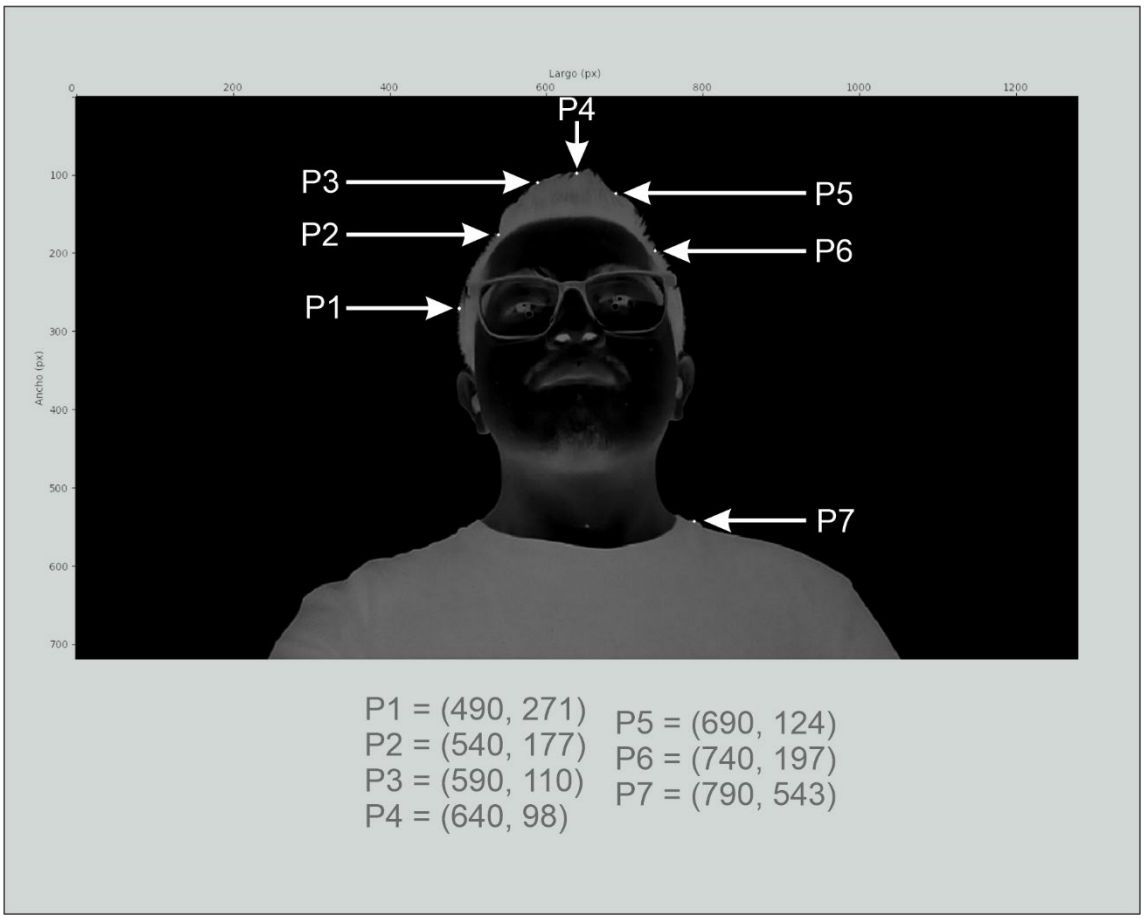

Figura 21: *Imagen resultante después de ser procesada por el algoritmo resaltando en color blanco la posición donde se señalan las coordenadas con los cambios de intensidad. Cada punto pertenece a una de las trazas, donde P representa una coordenada y el número corresponde a el número de traza a la que pertenece (elaboración propia).*

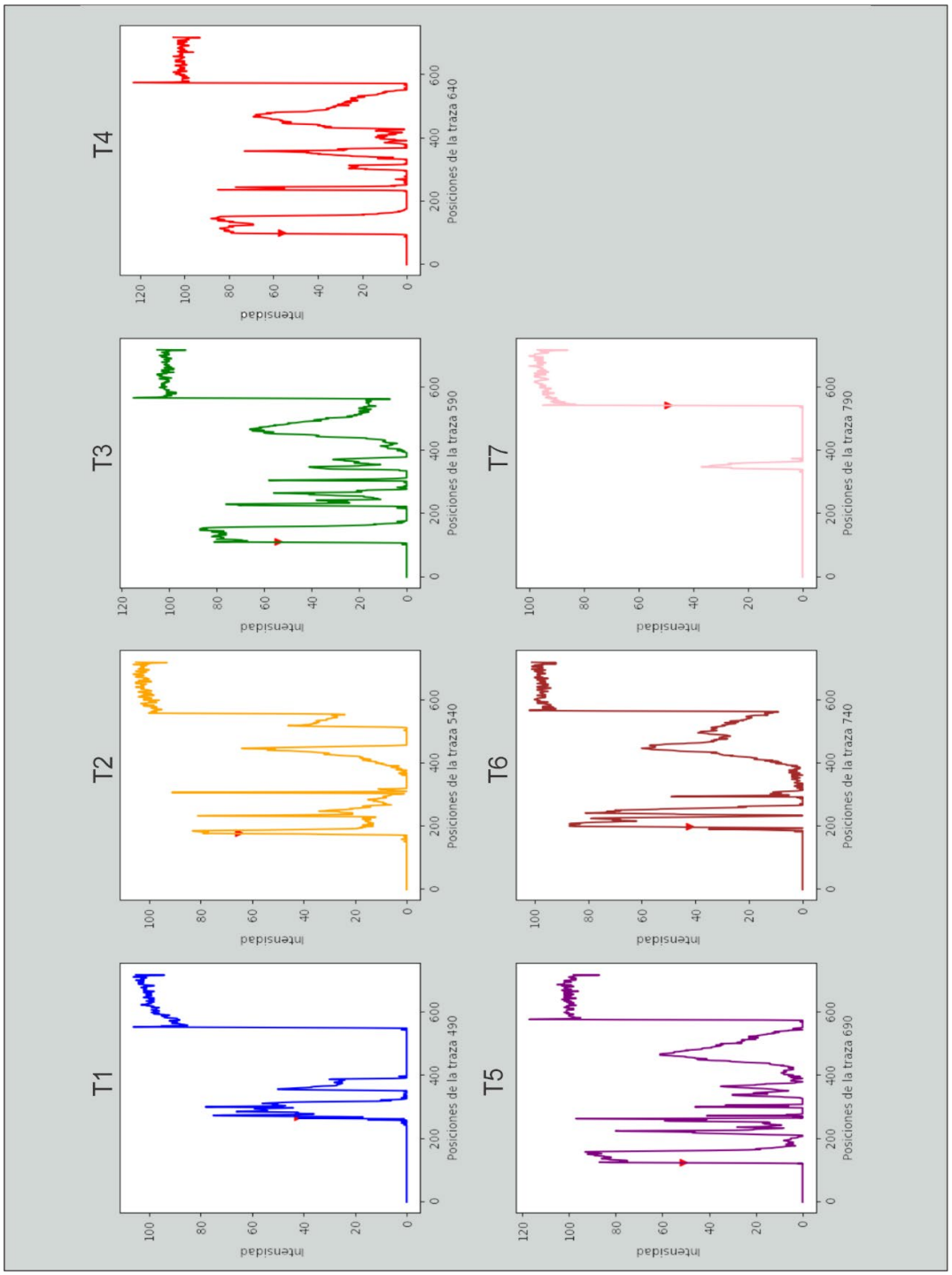

Figura 22: *Gráfica correspondiente a las trazas evaluadas de la Figura 20. Los marcadores en forma de triángulo en color rojo representan el punto donde la intensidad en la posición i cambió drásticamente respecto con la de la posición i-1 (elaboración propia).*

Ahora que el algoritmo de detección facial *Fonseca-Salas* se ha desarrollado, se procede a hacer el despliegue en la *Raspberry* para operar en el atril automatizado y hacer uso de su videocámara junto con la base giratoria para realizar nuevas pruebas.

4.3 Implementación del algoritmo de detección facial en el atril automatizado

La implementación de código de software en un nuevo dispositivo puede ser una tarea complicada y tediosa, los pasos por lo que se pasa son la elección e instalación del sistema operativo, configuración básica de red, memoria, particiones, etc. Y una vez ya funcionando el dispositivo se procede a hacer la instalación de las dependencias necesarias para el funcionamiento de una aplicación, en este caso, el algoritmo de detección facial. Afortunadamente no se necesitan de tantas dependencias y la mayoría de ellas son instaladas a través del gestor de paquetes de *python* llamado *pip*. Dichas dependencias, facilitan el desarrollo del algoritmo ya que nos permiten usar funciones existentes para tareas específicas, por decir, para controlar la videocámara en el sentido de tomar fotografías, seguramente habría que definir una serie de instrucciones que se encarguen de enviar y recibir los datos del sensor fotográfico al microprocesador, la conversión de números de flotantes a enteros, la señales para indicar que el sensor debe comenzar a capturar luz y otra para indicar que debe parar y así infinidad de funciones previas a la captura de una fotografía.

Las dependencias con las que se trabajan son *numpy* que, entre tantas funcionalidades, se usan aquellas que nos ayudan a realizar operaciones para vectores y matrices multidimensionales. *Matplotlib* ayuda a realizar las gráficas de los datos en las trazas de las imágenes. Y, por último, la biblioteca de *OpenCV* para la captura y tratamiento de imágenes. Es importante mencionar que se hace uso de otras bibliotecas de uso común para Python, pero no se mencionan porque el algoritmo de detección facial no depende directamente de éstas.

El sistema operativo es *Raspbian* ya que es el más usado con proyectos involucrados con la *Raspberry* además de proveer módulos que hacen facilitan el control de componentes electrónicos conectados a la *Raspberry*, un ejemplo de esos módulos es GPIO para el control de los pines de la *Raspberry*. Una vez completada la instalación, se procede a la configuración de conexión del servidor.

La R*aspberry* cuenta con salida a internet para acceder a ella desde cualquier lugar que se tenga acceso a una conexión *ssh*. Esto facilita el desarrollo de este trabajo ya que no era necesario estar presente junto al atril automatizado para su operación y para las pruebas se hicieron presencialmente. El dominio para realizar la conexión fue *jsg.dynathome.net* que se enlaza a la dirección pública que el proveedor de servicio de internet (*ISP*) asigna, el puerto *86* y el usuario *pi*. El código del algoritmo de detección facial fue almacenado en un repositorio de código abierto y gratuito, pero de manera privada y así realizar más fácilmente el desarrollo y despliegue de éste. La plataforma usada para control de versiones del algoritmo fue *Gitlab* porque ya se tenía experiencia previa trabajando en ella. Para obtener el código fuente del repositorio de *Gitlab*, se clona por primera vez y después todas las solicitudes de actualización o de sincronización del contenido, se hacen a través de la consola o de un cliente *git* con interfaz gráfica. Una vez clonado y sincronizado el repositorio del algoritmo, se ejecuta con el comando `Python3 reconocimiento.py`, dado que ya se tienen instaladas las dependencias.

Una vez que se ejecuta el programa principal, un menú es mostrado indicando si se quiere comenzar a ejecutar el algoritmo, una vez con la persona ubicada frente al atril en un ángulo inicial de *90°*, si se inicia el algoritmo, deberá encender un pequeño led con el que cuenta la videocámara, indicando que una fotografía está siendo tomada, después, la cámara gira *7.5°*, toma una segunda foto, y, por último, gira otros *7.5°* más y toma la última foto. Estas tres fotografías son procesadas por el algoritmo *Fonseca-Salas* y regresa las coordenadas donde la región facial es detectada.

Figura 23: *Fotografías de entrada para el algoritmo de detección facial. a) muestra la primera fotografía tomada a 90° b) muestra la segunda fotografía tomada a 97.5° y c) muestra la fotografía tomada a 105° (elaboración propia).*

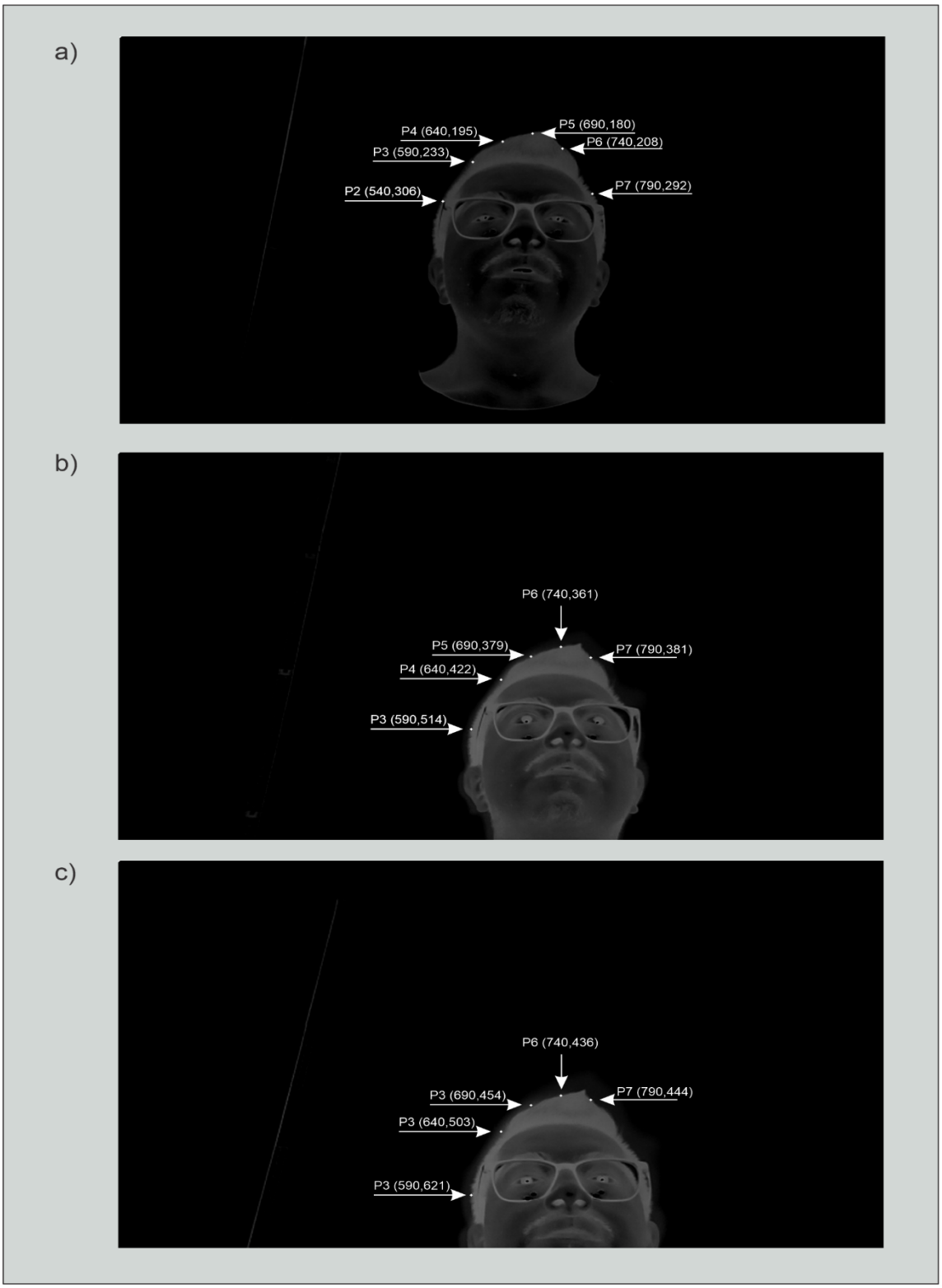

Figura 24: *Resultados del algoritmo de detección facial para las fotos de entrada de la Figura 23 señalando las coordenadas encontradas para cada imagen entrada. a) primera fotografía tomada a 90° b) segunda fotografía tomada a 97.5° c) fotografía tomada a 105°(elaboración propia).*

Estas coordenadas son los datos de entrada para nuestra última fase: el cálculo de la altura de la persona, dicho proceso consistirá en obtener una relación entre los pixeles y la distancia que la videocámara puede capturar para que dada una componente de la coordenada más baja se calcule la altura.

4.4 Cálculo de la altura de la persona

Ahora se retoma la sección *3.3 Cálculo de la altura de la persona* donde se explica la relación entre pixeles y centímetros para obtener la altura de una persona a través de una imagen. Recordando que la distancia entre la cámara y la persona frente a la cámara está situada a una distancia de *60* cm y la altura de la cámara está a *130* cm fijado a la paleta del atril, así como lo describe la *Figura 13* del capítulo 3.

Se retoma la captura de la fotografía y las condiciones iniciales comentadas anteriormente. En el fondo se ha colocado una cinta métrica para obtener los centímetros que dicha fotografía va a capturar. Para apreciar mejor los números en la cinta métrica la imagen es capturada con una resolución de *1920x1080* pixeles, la captura de más pixeles ayudará a identificar mejor las áreas de interés en la fotografía, para este caso los números, aunque las operaciones siguientes se realizarán con los valores de *1280x720 pixeles*.

Es importante mencionar que la cinta métrica se colocó a *1* m de distancia sobre el suelo, y que a los números que se aprecian en las *Figura 25* se le debe sumar dichos centímetros para obtener la altura real que la videocámara está capturando. Ahora para la obtención de la relación pixel-altura se le resta el valor menor al valor mayor, obteniendo así *45* cm como rango de amplitud en centímetros de captura de nuestra videocámara.

Haciendo una operación simple, si la resolución de las imágenes capturadas es de *1280x720* pixeles *LargoxAncho* respectivamente, se procede a dividir el ancho entre el rango de amplitud de captura de la videocámara entre los *720* pixeles; $\frac{R.Amplitud}{Ancho}$ dando así un valor de $0.063195\,\text{cm}$ por cada pixel en la imagen, este valor deberá ser calculado tres veces para cada uno de los ángulos en los que se tomarán las imágenes *90°, 97.5°* y *105°* previamente definidos.

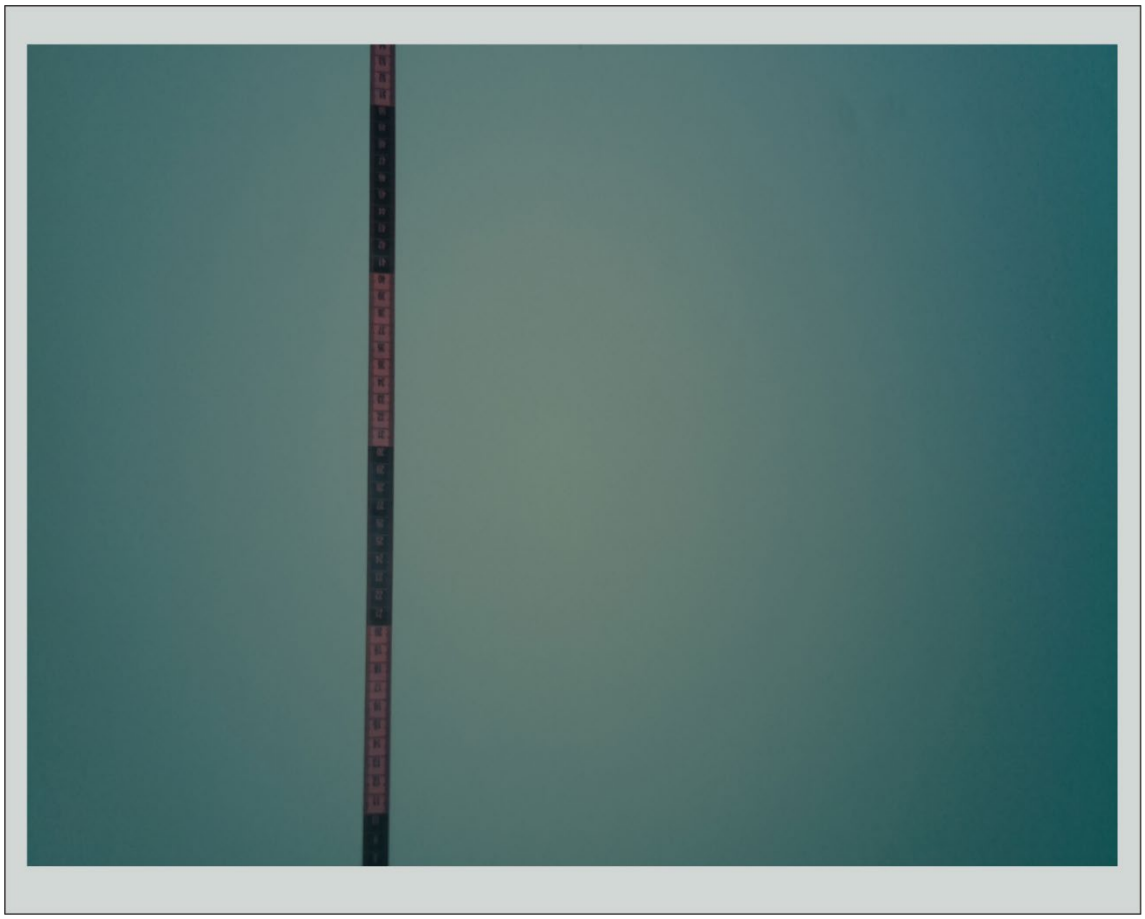

Figura 25: *Imagen del fondo con la cinta métrica en la pared que muestra los centímetros que la videocámara puede capturar, se puede observar que el primer número que captura (de abajo hacia arriba) es 8, mientras que el último número que se aprecia (hasta la parte superior de la fotografía) es 53.5 aproximadamente (elaboración propia).*

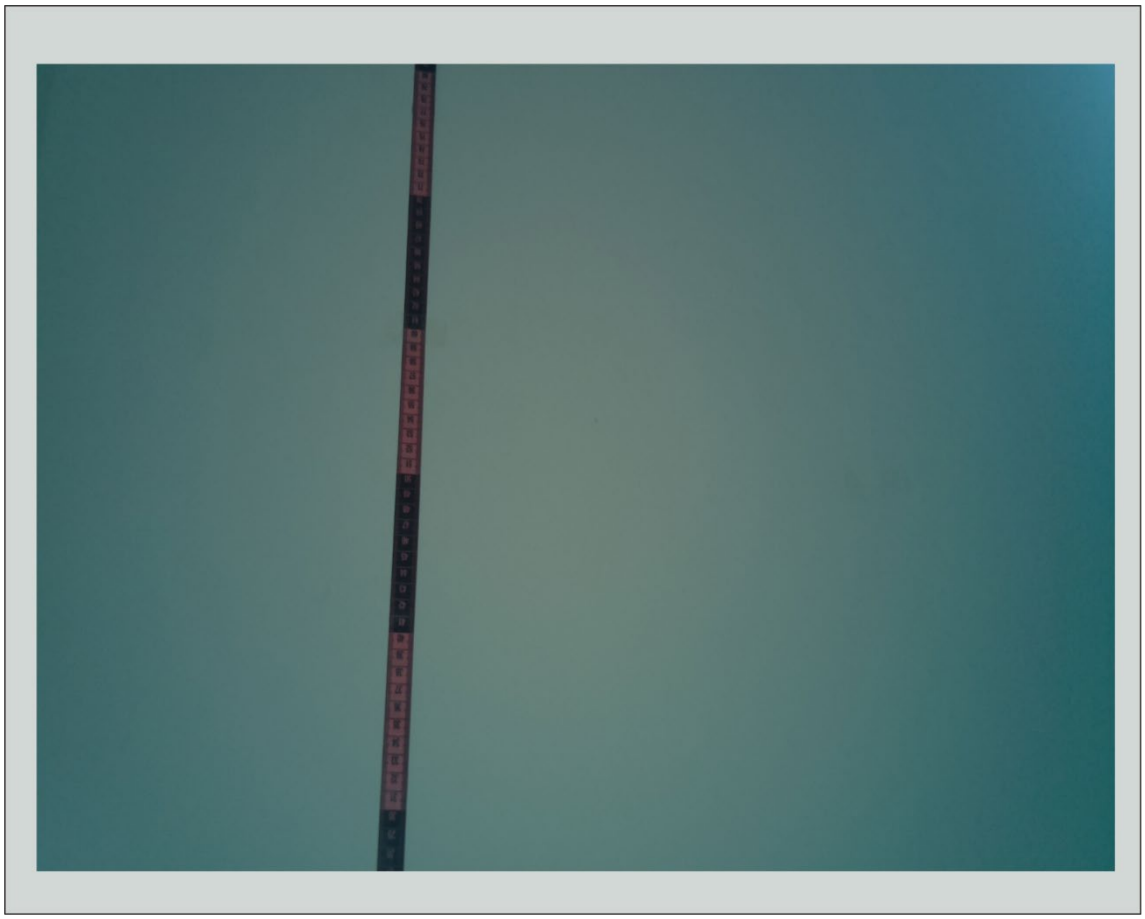

Figura 26: *Imagen capturada con un ángulo de 90°, se aprecia que el valor inferior en la cinta métrica es de 28 cm, mientras que el valor superior es de 80 cm (elaboración propia).*

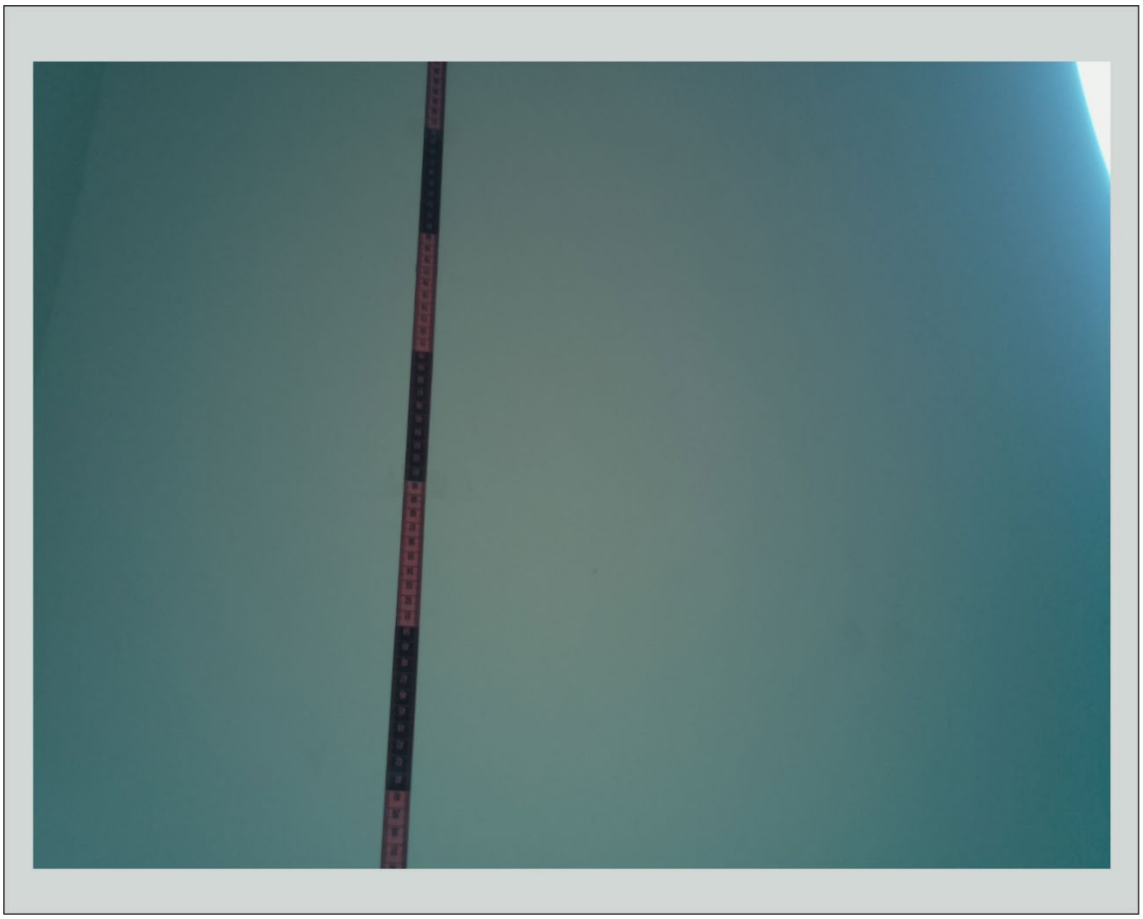

Figura 27: *Imagen capturada con un ángulo de 97.5°, se aprecia que el valor inferior en la cinta métrica es de 37 cm, mientras que el valor superior es de 96.5 cm (elaboración propia).*

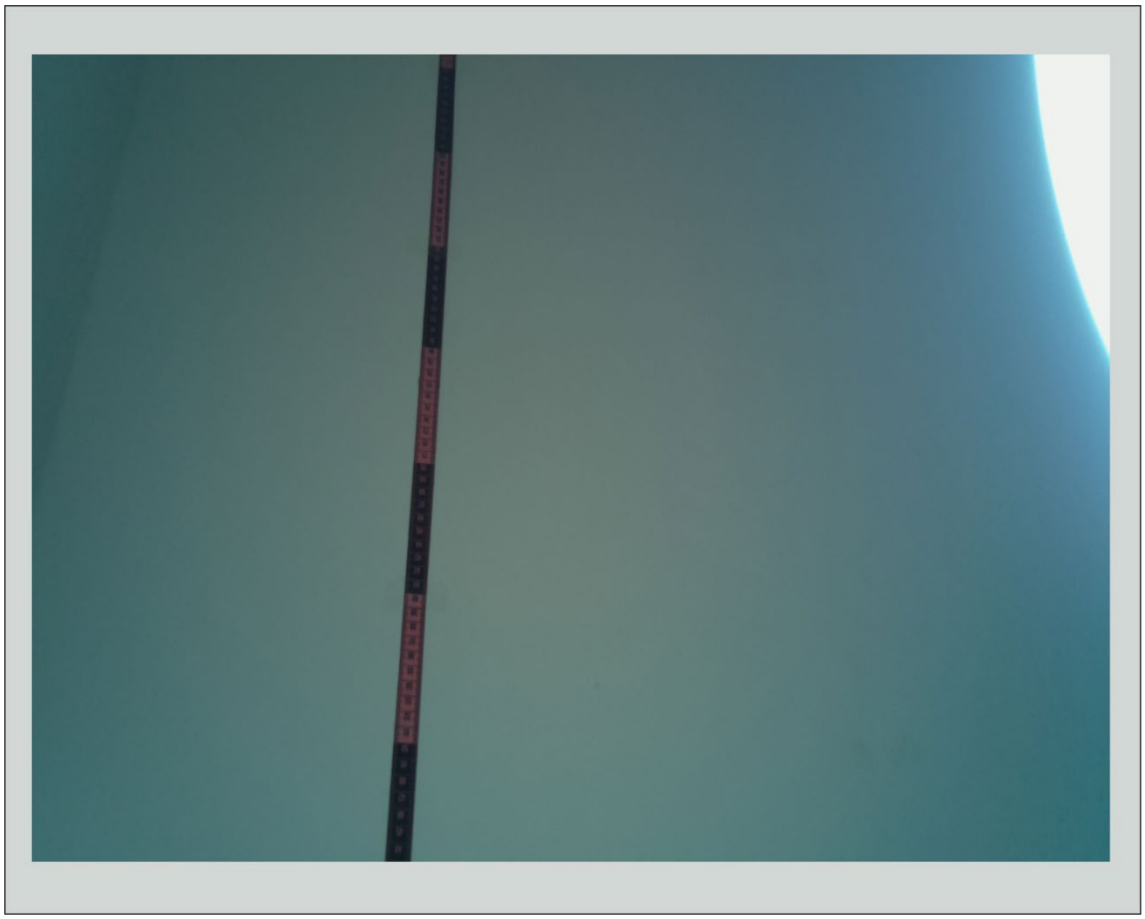

Figura 28: *Imagen capturada con un ángulo de 105°, se aprecia que el valor inferior en la cinta métrica es de 44 cm, mientras que el valor superior es de 111.15 cm (elaboración propia).*

De los valores obtenidos en las *figuras 26, 27 y 28* se representan en una tabla que nos calcula los valores de la relación *pixel-altura* a partir de los datos valor inferior, valor superior, $altura_c$ y relación *pixel-altura* calculada a partir de la diferencia dividida entre $altura_c$ y ancho de la foto. Una vez realizados estos cálculos se obtiene la altura de una persona que su fotografía haya sido procesada por el algoritmo (cumpliendo las condiciones iniciales) y con el valor de la componente *y* también la más baja de las coordenadas obtenidas.

***Tabla 2**: Cálculo de relación pixel-altura a partir de la información de las figuras 26, 27 y 28, donde el valor inferior es la primera marca de la cinta métrica que aparece en la fotografía, por otro lado, el valor superior marca el último valor de la cinta métrica en la imagen.*

Grados	V. Inferior	V. Superior	Diferencia	Relación pixel-altura
90°	28	80	52	0.072222222
97.5°	37	96.5	59.5	0.082638889
105°	44	111.5	67.5	0.09375

Para el cálculo de la altura de la persona se tienen las siguientes variables: *1)* valor inferior en la imagen tomada en los distintos ángulos de la videocámara (*90°, 97.5°* y *105°*) representada como $val_{\inf}$, *2)* la distancia del suelo hasta el valor inferior de la imagen tomada representada con una d que además dicho valor es de *100* cm, *3)* valor de la relación pixel-altura representada como rel_{pa} y *4)* valor de la componente y de la coordenada más baja, ahora poniendo todo en una formula, nos queda:

$$(d + val_{\inf}) + (rel_{pa})(720 - y) \tag{4.1}$$

en las pruebas realizadas la parte frente al atril tiene un desnivel de 8.5 cm más, los cuáles se deberán restar para obtener el valor de *d*. Además, la resta de $(720 - y)$, es realizada ya que el algoritmo comienza a contar los pixeles de arriba hacia abajo, siendo *1* el valor más alto y *720* el menor. Ahora es posible obtener la altura aproximada de la persona.

De la *Figura 24* se obtiene los valores de las coordenadas cuya componente y es la más baja, para las tres imágenes tomadas en los diferentes ángulos. Los valores para imagen tomada a un ángulo de *90°*: $d = 100; val_{\inf} = 28; rel_{pa} = \frac{13}{180}; y = 180$, y sustituyendo en la fórmula *(4.1)* se llega al resultado:

$$(100 + 28 - 8.5) + (\tfrac{13}{180})(720 - 180) = 158.49$$

Ahora, los valores para la imagen tomada a un ángulo de *97.5°* son:
$d = 100; val_{\inf} = 37; rel_{pa} = \frac{119}{1440}; y = 361$, y sustituyendo en la fórmula (4.1) se llega al resultado:

$$(100 + 37 - 8.5) + (\frac{119}{1440})(720 - 361) = 158.16$$

Y por último, los valores para la imagen tomada a un ángulo de *105°* son:
$d = 100; val_{\inf} = 44; rel_{pa} = \frac{3}{32}; y = 436$, y sustituyendo en la fórmula (4.1) se llega al resultado:

$$(100 + 44 - 8.5) + (\frac{3}{32})(720 - 436) = 162.125$$

Estas pruebas fueron realizadas personalmente, y teniendo como dato que mi altura es de 159 cm y comparando con los resultados obtenidos se deduce que son cercanos a lo esperado, con los siguientes márgenes de error *0.32, 0.52* y *1.96.* Con todo lo anterior descrito, el algoritmo de detección facial genera los datos necesarios para el cálculo de la altura, pero ahora se analizará el rendimiento de éste, medido a través del consumo de CPU y el tiempo de ejecución.

4.5 Pruebas de rendimiento del algoritmo de detección facial

En la sección anterior se describe el proceso de implementación y pruebas del algoritmo *Fonseca-Salas* además del cálculo de la altura de una persona con la información obtenida de dicho algoritmo. Como se ha comentado, aunque existen algoritmos de detección facial funcionales, estos requieren de un gran consumo de procesamiento para lograr su objetivo, y si la tarea debe ser hecha en tiempo real, dicho consumo aumenta considerablemente.

La serie de pruebas que se realizan consiste en evaluar el tiempo de ejecución del algoritmo *Fonseca-Salas* que consiste en cronometrar el tiempo que un dispositivo tarda en completar la ejecución de una o más instrucciones, guardando la hora de inicio antes de las instrucciones a cronometrar y, por ende, guardando la hora de fin de las instrucciones. Después tan solo se realiza una simple resta del tiempo final menos el tiempo inicial, obteniendo así el tiempo de ejecución. Es importante saber qué instrucciones se evalúan en el programa para definir correctamente donde se guardarán las marcas de tiempo de inicio y de fin. Por ejemplo, estas marcas pueden estar en la primera línea de código y en la última respectivamente, esto sería correcto si las líneas de código de interés fuera el código entre estas dos marcas de tiempo. El programa creado para el algoritmo *Fonseca-Salas* consiste en una serie de funciones y la medición del tiempo será iniciado antes de comenzar el ciclo *for* y terminando al final de dicho ciclo que únicamente ejecuta la función *getCoordinates*, calculando así el tiempo que el algoritmo tarda en ejecutarse. Además, se ejecuta una función que guardará el valor actual de uso del CPU para conocer su valor durante la ejecución.

Además, para las pruebas se modifica el número de trazas para el algoritmo, ya que en la sección *4.2 Algoritmo de detección facial* se definió que siete trazas son ideales y con las condiciones iniciales marcadas, son capaces de cubrir el área de interés para el algoritmo. Haciendo esta variación se verifica si dicho número cumple los dos criterios principales del algoritmo, detección de la región facial y la eficiencia.

Otro aspecto para considerar es el número de pruebas, se realizan tres pruebas para las cuatro distintas series de trazas. Estas imágenes son tomadas en tres posiciones distintas cada una, la primera con el sujeto a la mitad de la imagen, la segunda con el sujeto desplazado a la derecha y la última imagen con el sujeto desplazado a la izquierda.

4.5.1 Prueba de rendimiento a 3 trazas

Esta primera prueba consiste en *281* iteraciones del algoritmo *Fonseca-Salas* y se toman *10* imágenes con el sujeto desplazado entre derecha, centro e izquierda y solo se muestran tres imágenes que representen lo anterior mencionado.

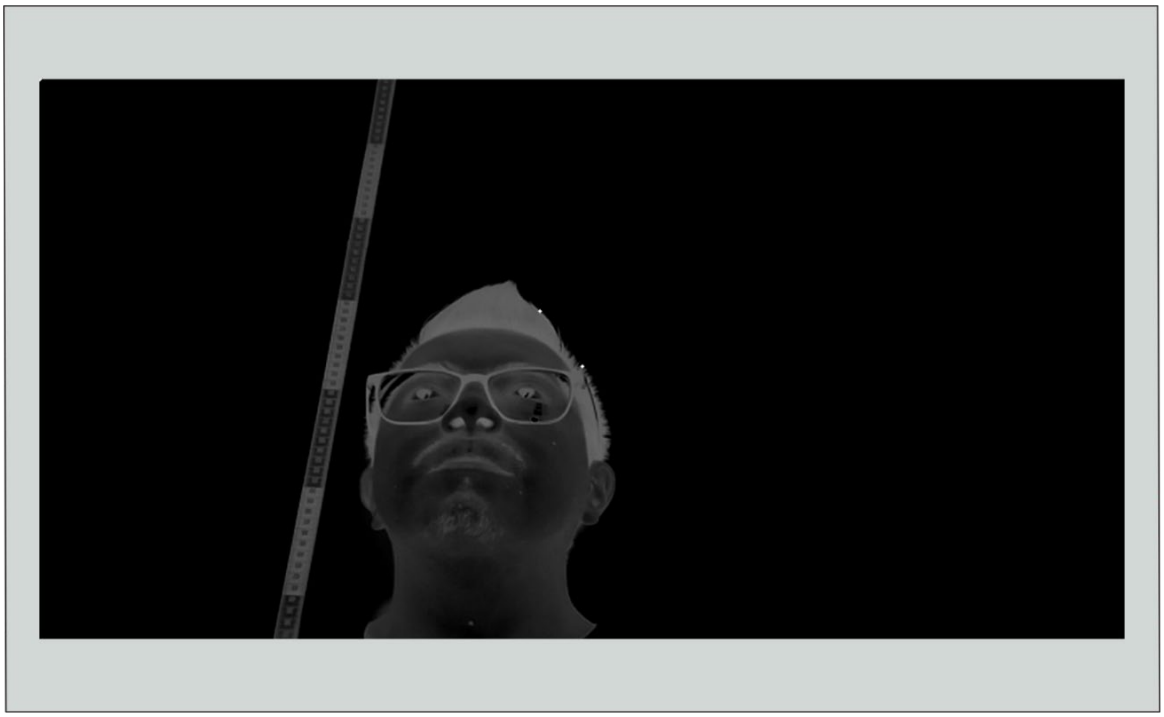

Figura 29: *Imagen de salida del algoritmo de detección facial con tres trazas analizadas y el sujeto situado a la izquierda (elaboración propia).*

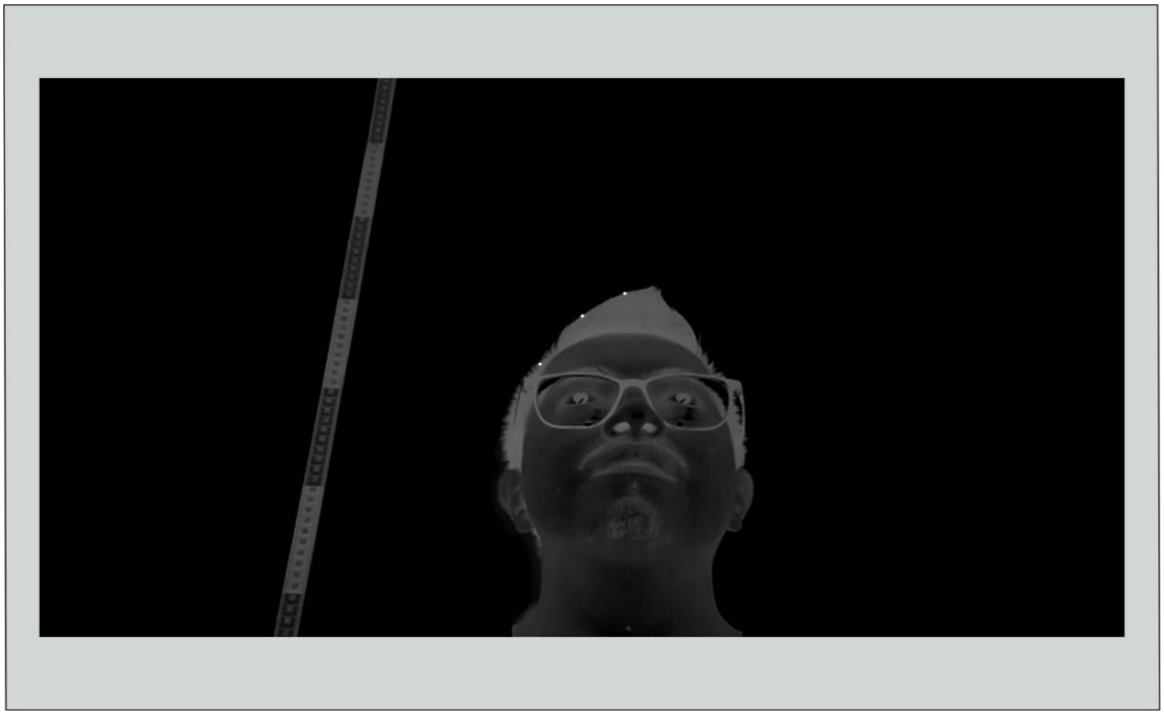

Figura 30: *Imagen de salida del algoritmo de detección facial con tres trazas analizadas y el sujeto situado al centro (elaboración propia).*

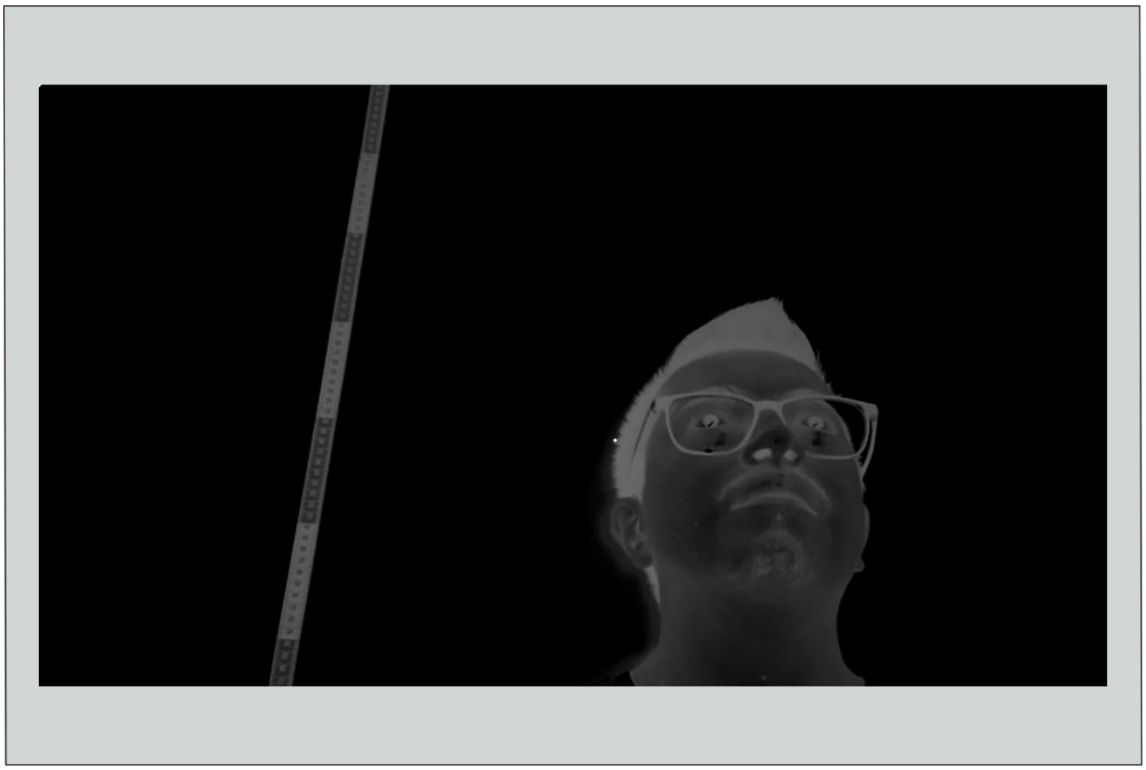

Figura 31: *Imagen de salida del algoritmo de detección facial con tres trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

La *Figura 32* es la gráfica que representa el consumo del CPU durante las *281* iteraciones de esta prueba, se observa que el consumo mayor es de *29.4%* y el menor de 4.7%, teniendo así un promedio de *26.41%* de CPU para tres trazas evaluadas. Por otro lado, la *Figura 33* es la gráfica que representa el tiempo de ejecución del algoritmo para cada interacción, se observa que el valor máximo es *0.06662* segundos, el mínimo *0.03212* segundos y el promedio es *0.04366* segundos.

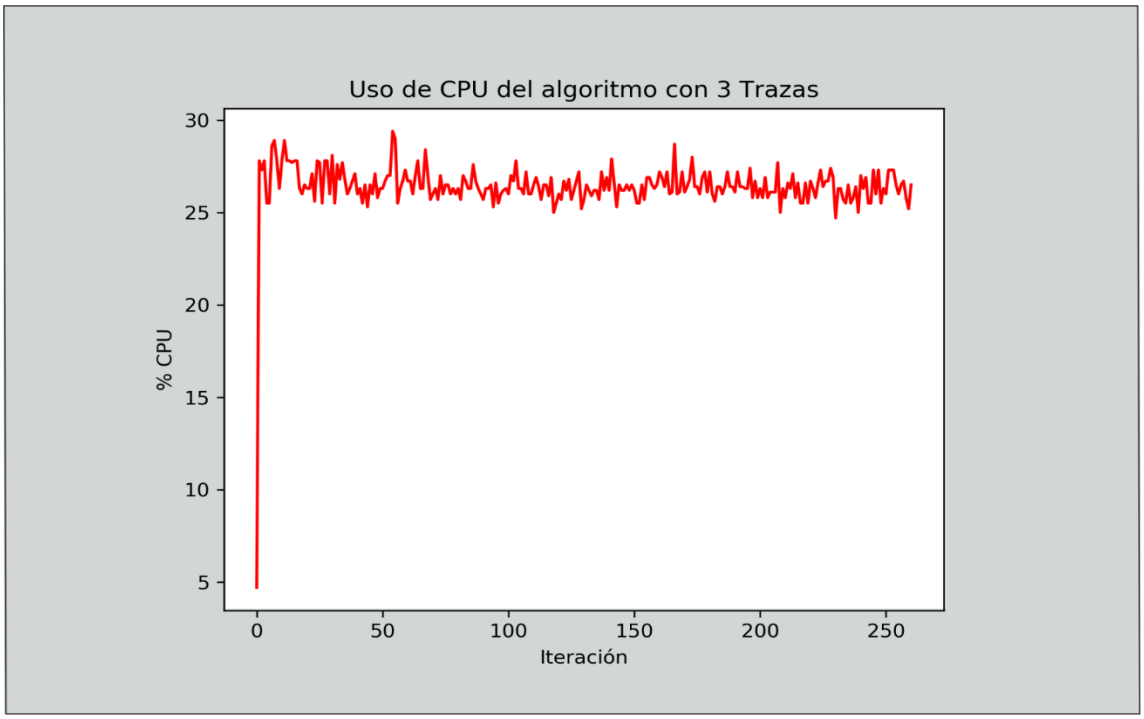

Figura 32: *Gráfica de uso del CPU durante la ejecución del algoritmo de detección facial evaluando tres trazas (elaboración propia).*

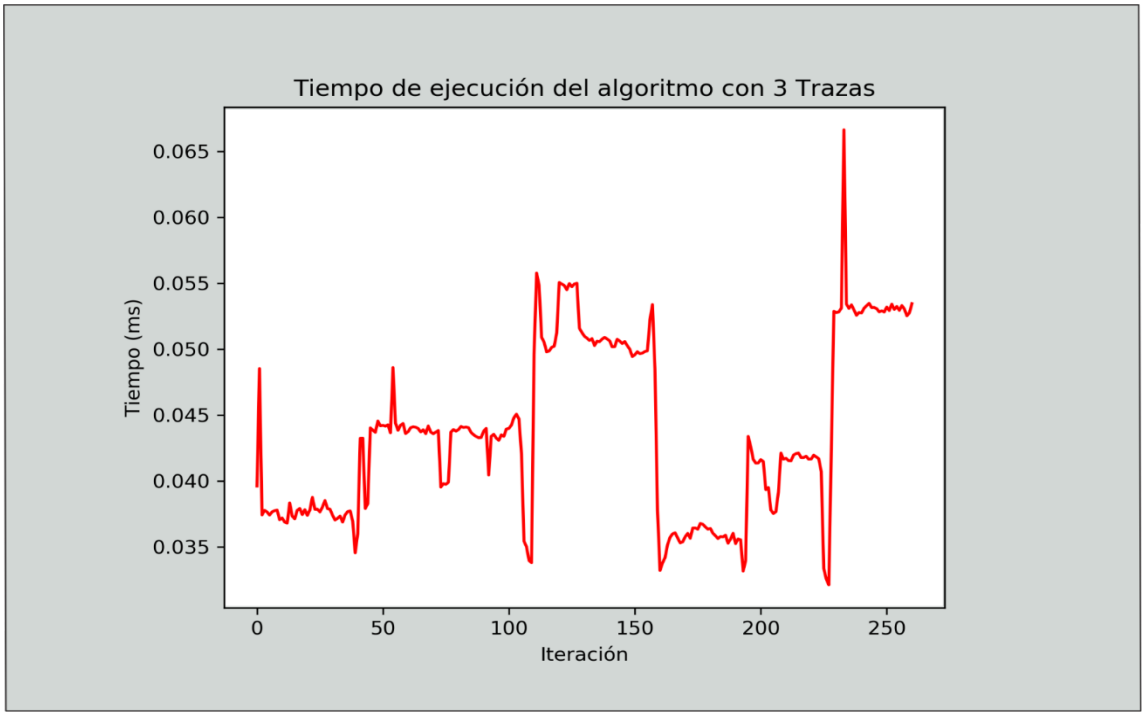

Figura 33: *Gráfica del tiempo de ejecución del algoritmo de detección facial evaluando tres trazas (elaboración propia).*

4.5.2 Prueba de rendimiento a 5 trazas

La segunda prueba consiste en *234* iteraciones del algoritmo *Fonseca-Salas* y se toman 10 imágenes con el sujeto desplazado entre derecha, centro e izquierda y solo se muestran tres imágenes que representen lo anterior mencionado.

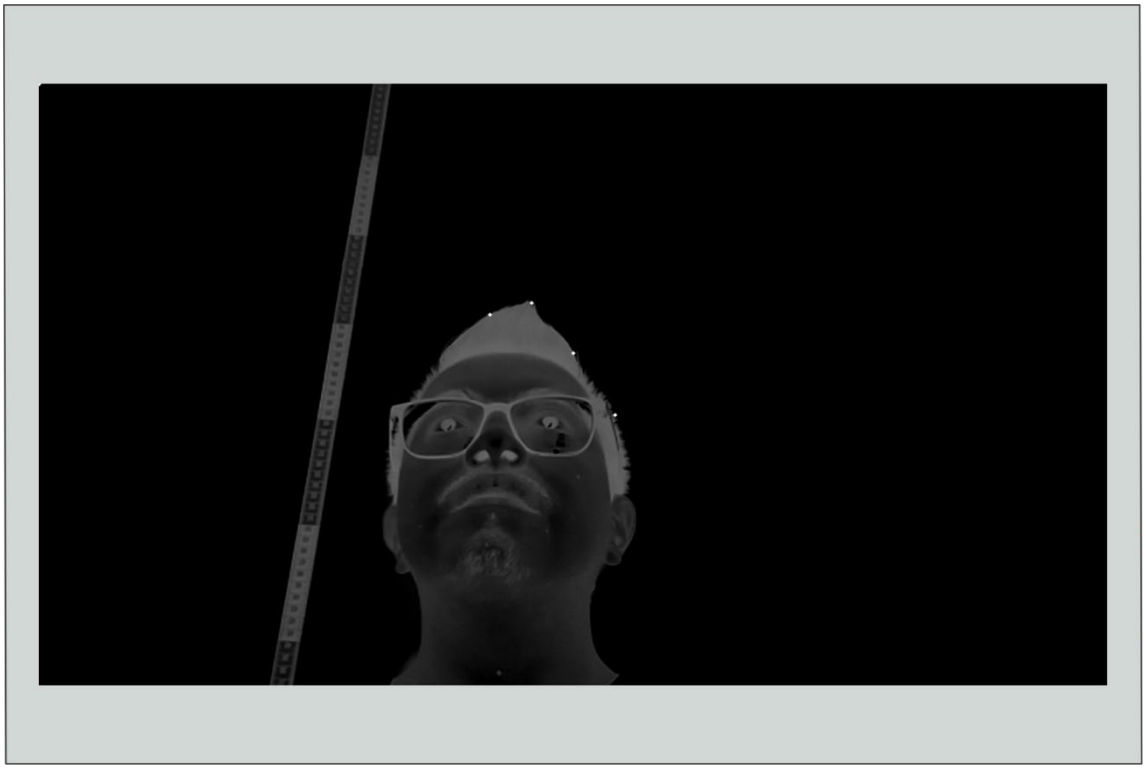

Figura 34: *Imagen de salida del algoritmo de detección facial con cinco trazas analizadas y el sujeto situado a la izquierda (elaboración propia).*

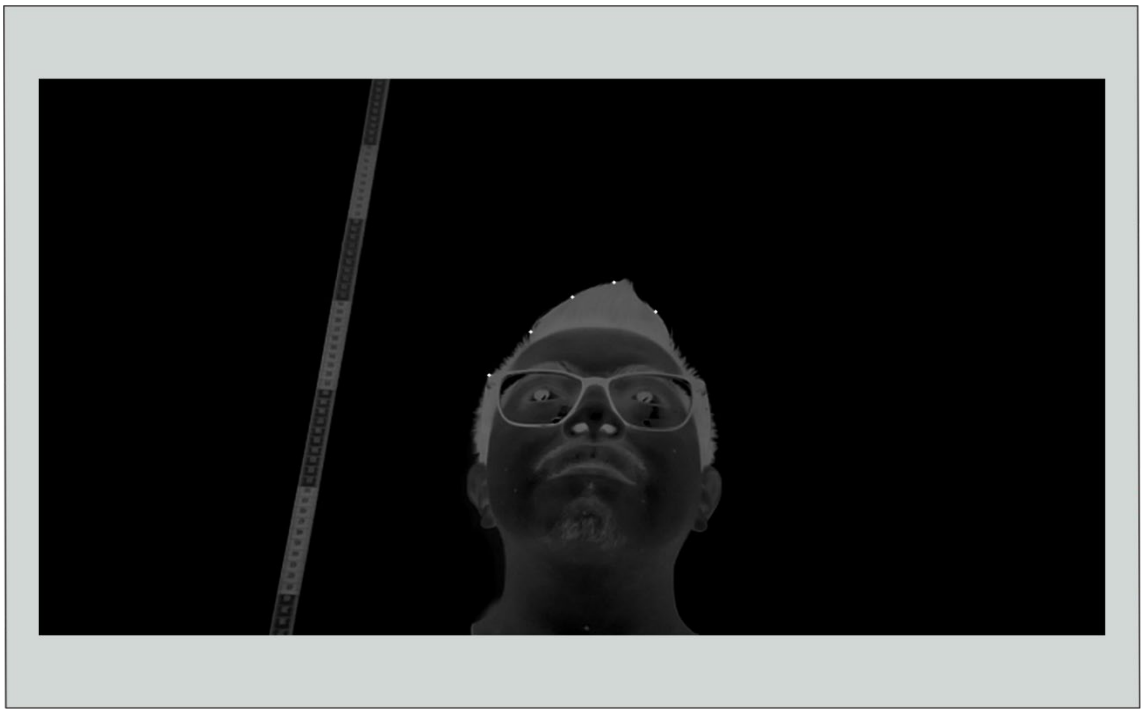

Figura 35: *Imagen de salida del algoritmo de detección facial con cinco trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

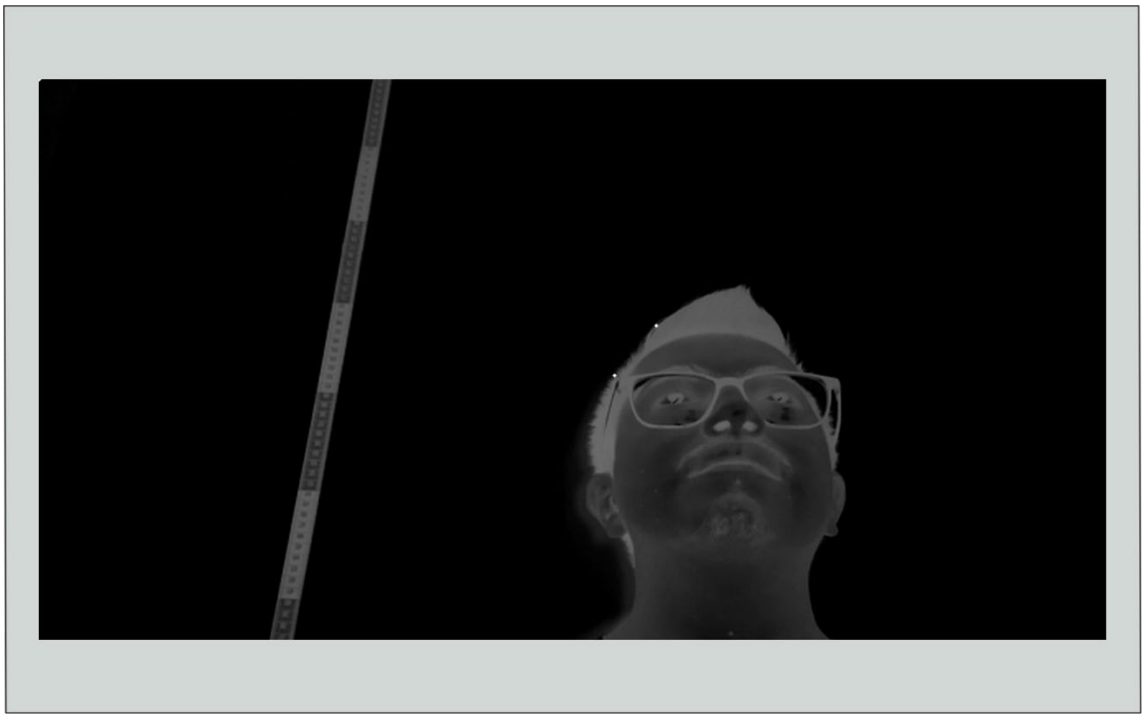

Figura 36: *Imagen de salida del algoritmo de detección facial con cinco trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

La *Figura 37* es la gráfica que representa el consumo del CPU durante las *234* iteraciones de esta prueba. El consumo mayor es de *29.5%* y el menor de 0.6%, teniendo así un promedio de *26.22%* de CPU para cinco trazas evaluadas. Por otro lado, la *Figura 38* es la gráfica que representa el tiempo de ejecución del algoritmo para cada interacción y se observa que el valor máximo es *0.09670* segundos, el mínimo *0.05708* segundos y el promedio es *0.07358* segundos.

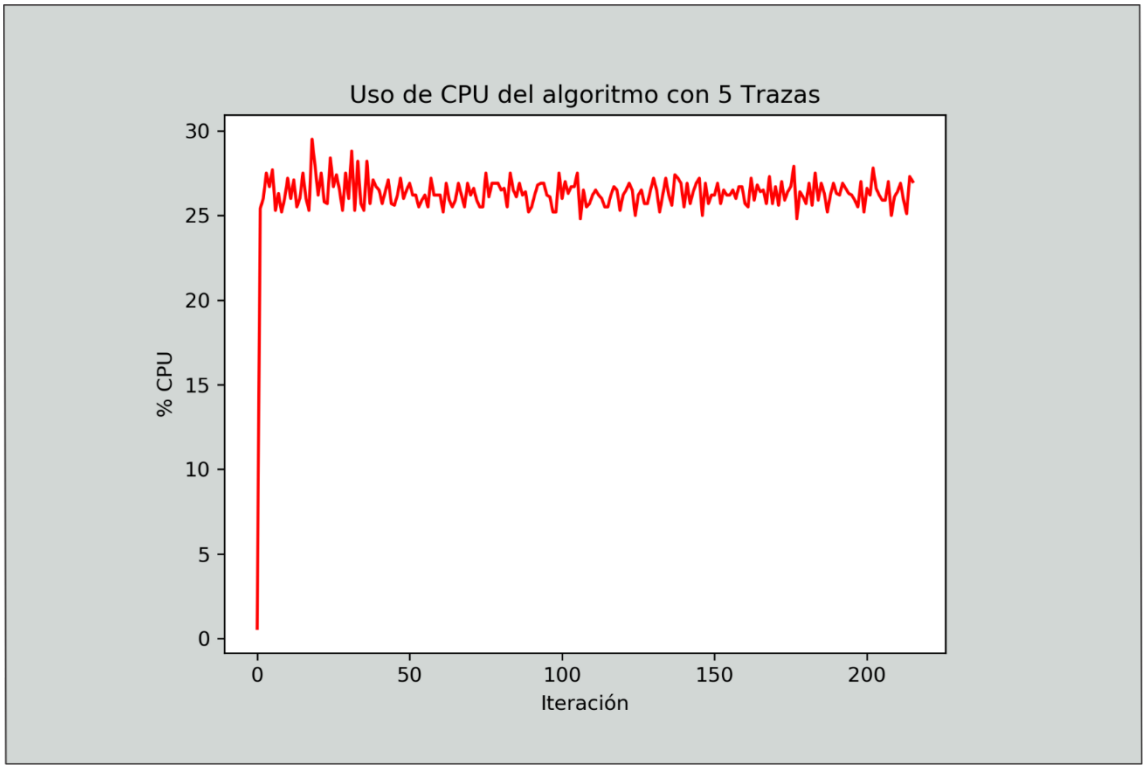

Figura 37: *Imagen de salida del algoritmo de detección facial con cinco trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

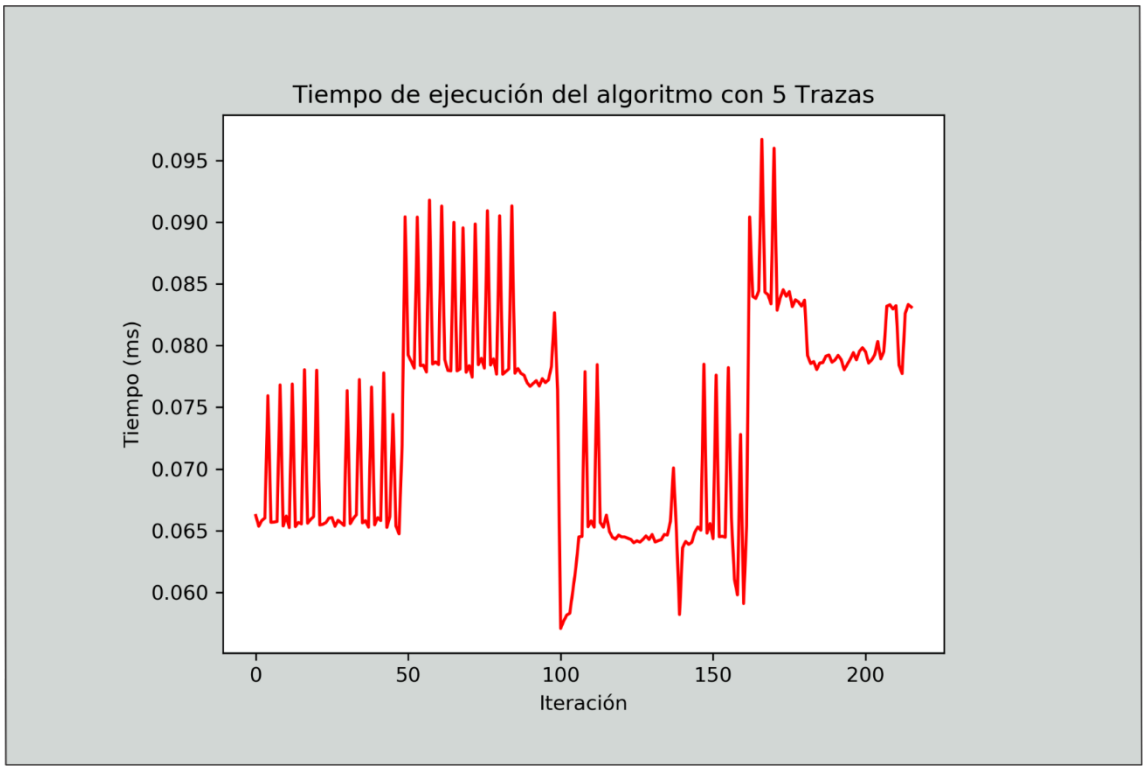

Figura 38: *Imagen de salida del algoritmo de detección facial con cinco trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

4.5.3 Prueba de rendimiento a 7 trazas

La tercera prueba consiste en *228* iteraciones del algoritmo *Fonseca-Salas* y se toman *10* imágenes con el sujeto desplazado entre derecha, centro e izquierda y solo se muestran tres imágenes que representen lo anterior mencionado.

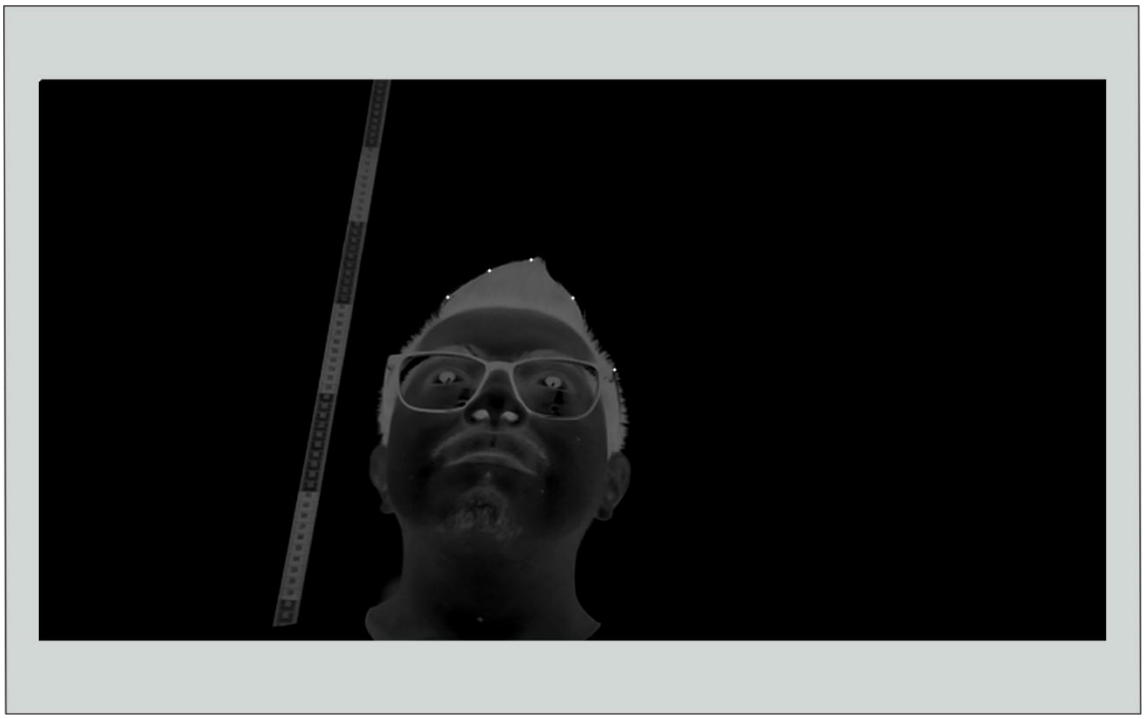

Figura 39: *Imagen de salida del algoritmo de detección facial con siete trazas analizadas y el sujeto situado a la izquierda (elaboración propia).*

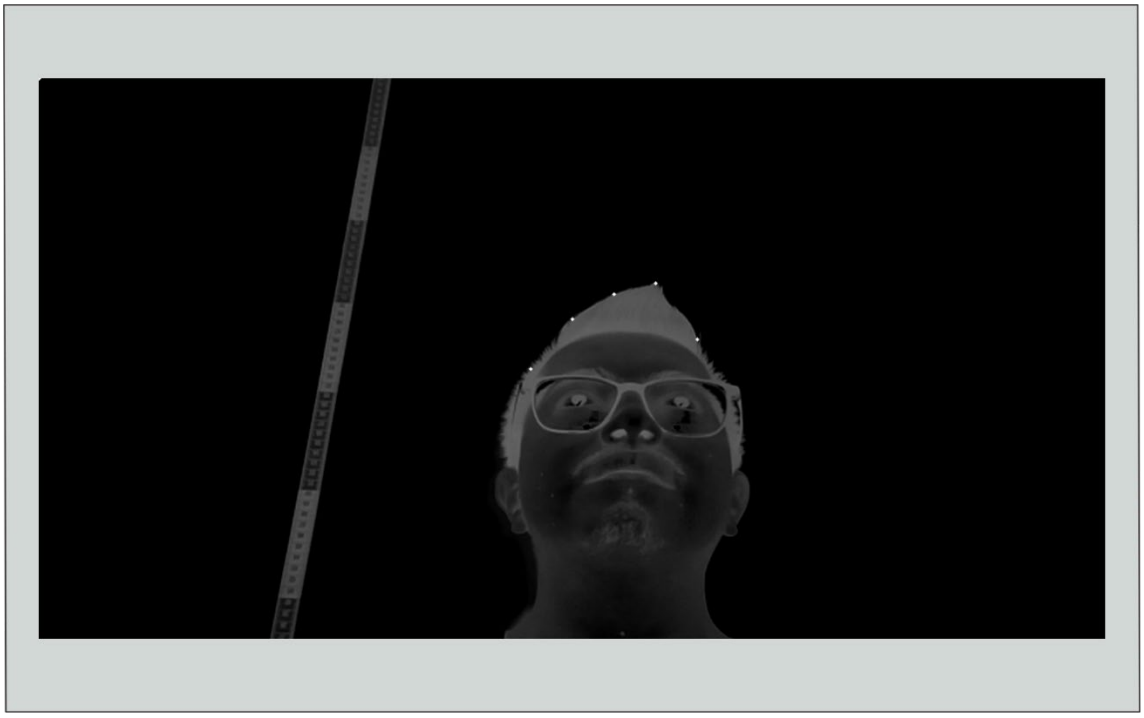

Figura 40: *Imagen de salida del algoritmo de detección facial con siete trazas analizadas y el sujeto situado al centro (elaboración propia).*

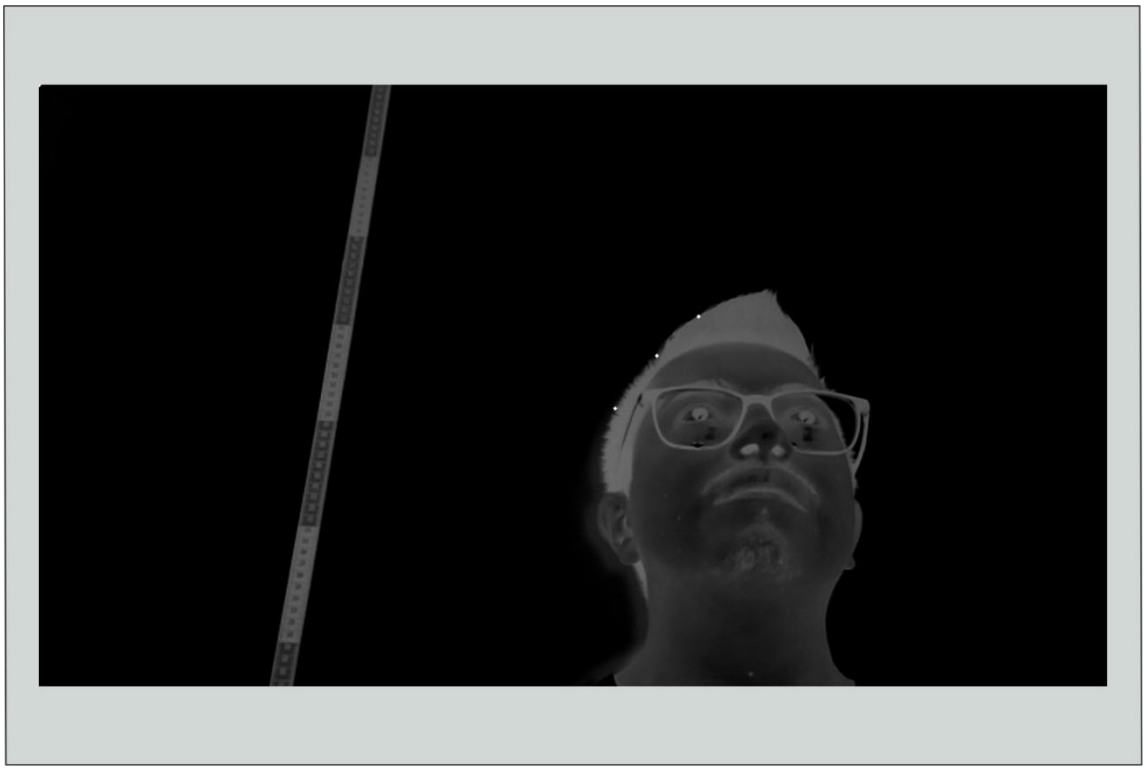

Figura 41: *Imagen de salida del algoritmo de detección facial con siete trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

La *Figura 42* es la gráfica que representa el consumo del CPU durante las *228* iteraciones de esta prueba, los valores obtenidos representan que el consumo mayor es de *28.2%* y el menor de *0.7%*, teniendo así un promedio de *26.005%* de CPU para siete trazas evaluadas. Por otro lado, la *Figura 41* es la gráfica que representa el tiempo de ejecución del algoritmo para cada interacción, y se observa que el valor máximo es *0.27237* segundos, el mínimo *0.08692* segundos y el promedio es *0.10133* segundos.

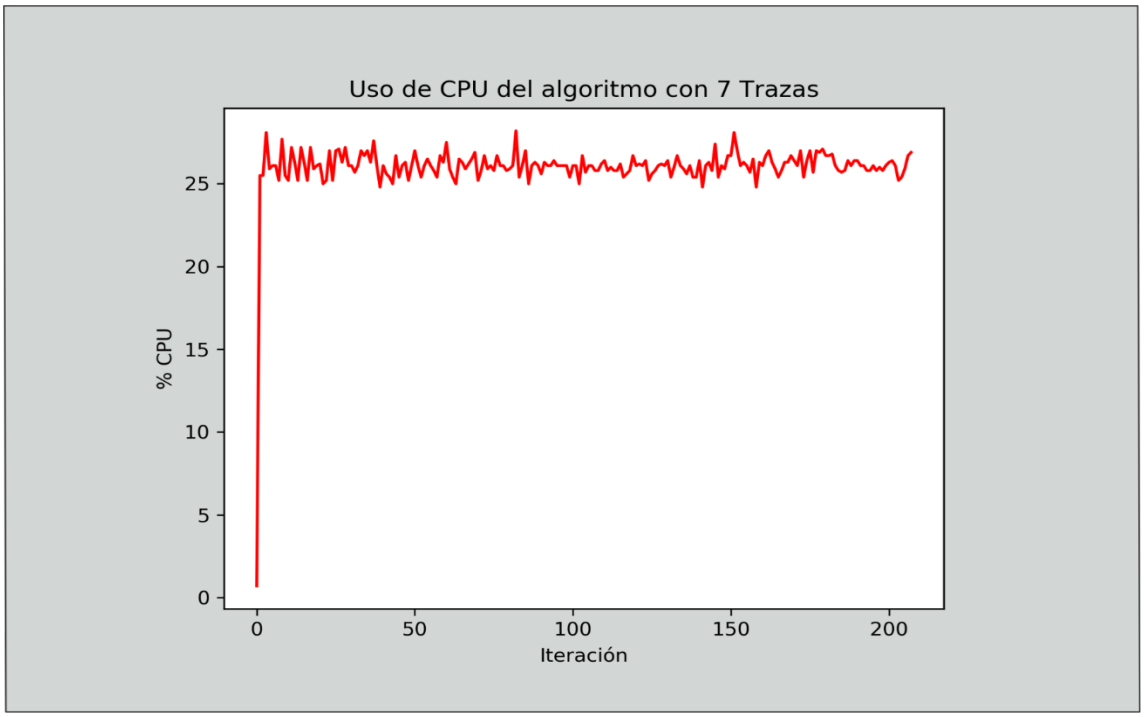

Figura 42: *Imagen de salida del algoritmo de detección facial con siete trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

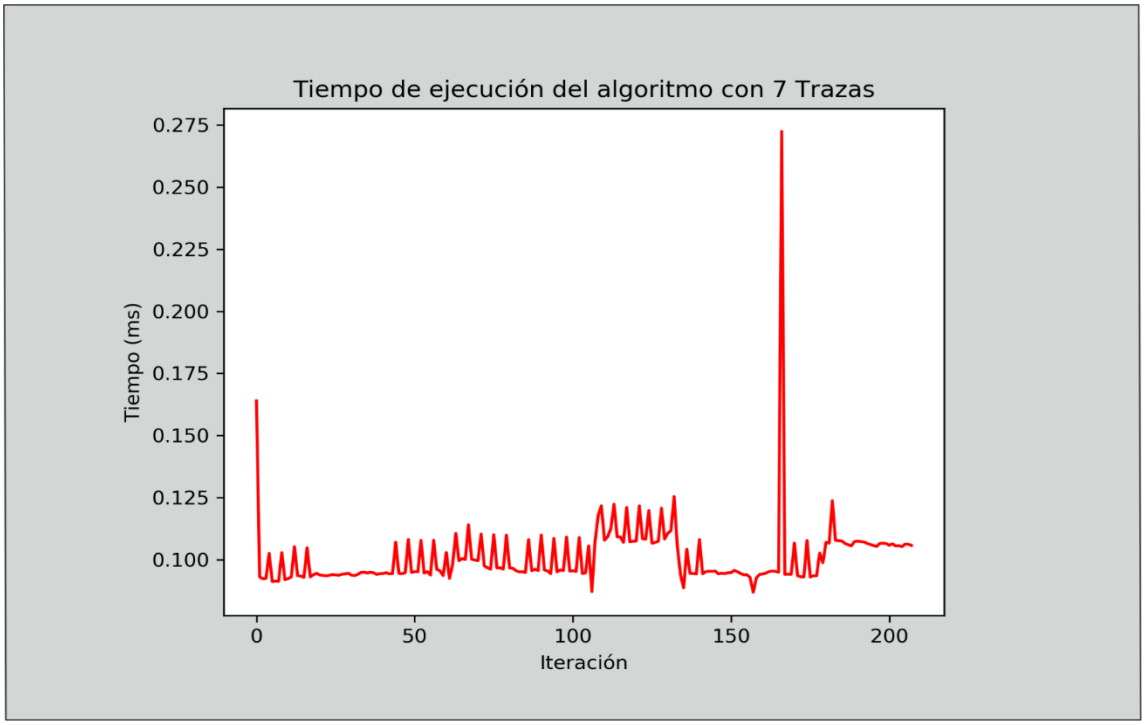

Figura 43: *Imagen de salida del algoritmo de detección facial con siete trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

4.5.4 Prueba de rendimiento a 9 trazas

La cuarta prueba consiste en *160* iteraciones del algoritmo *Fonseca-Salas* y se toman ocho imágenes con el sujeto desplazado entre derecha, centro e izquierda y solo se muestran tres imágenes que representen lo anterior mencionado.

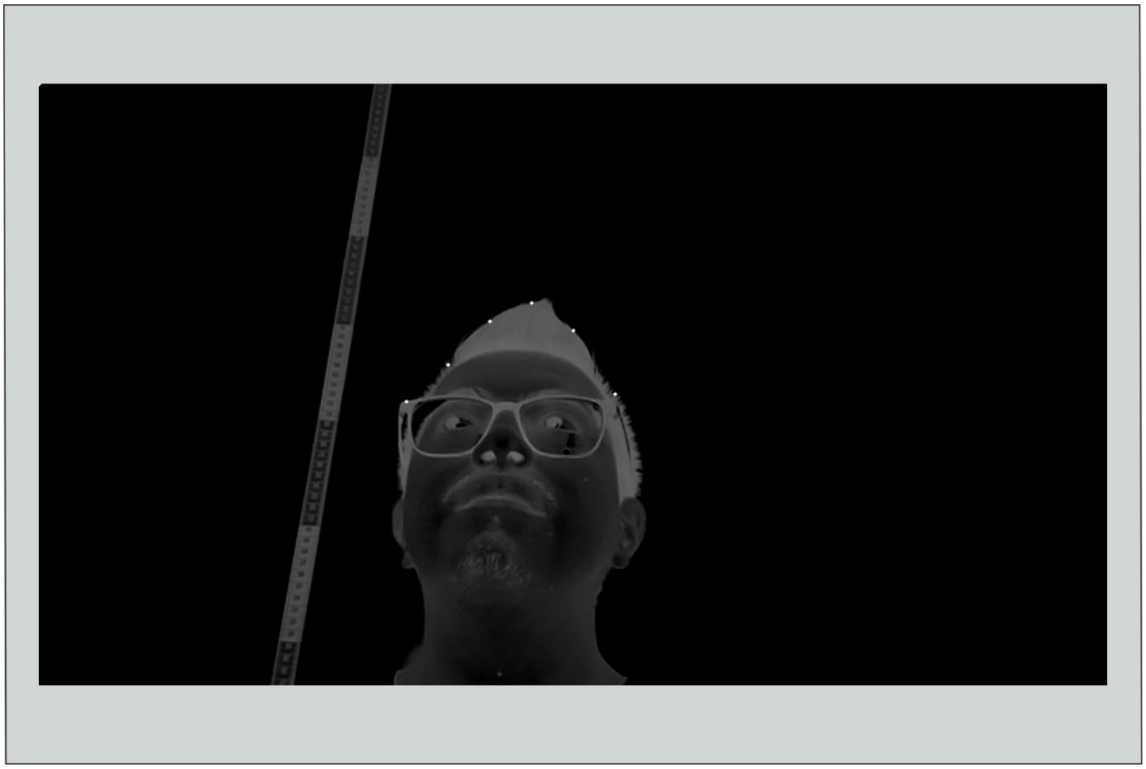

Figura 44: *Imagen de salida del algoritmo de detección facial con nueve trazas analizadas y el sujeto situado a la izquierda (elaboración propia).*

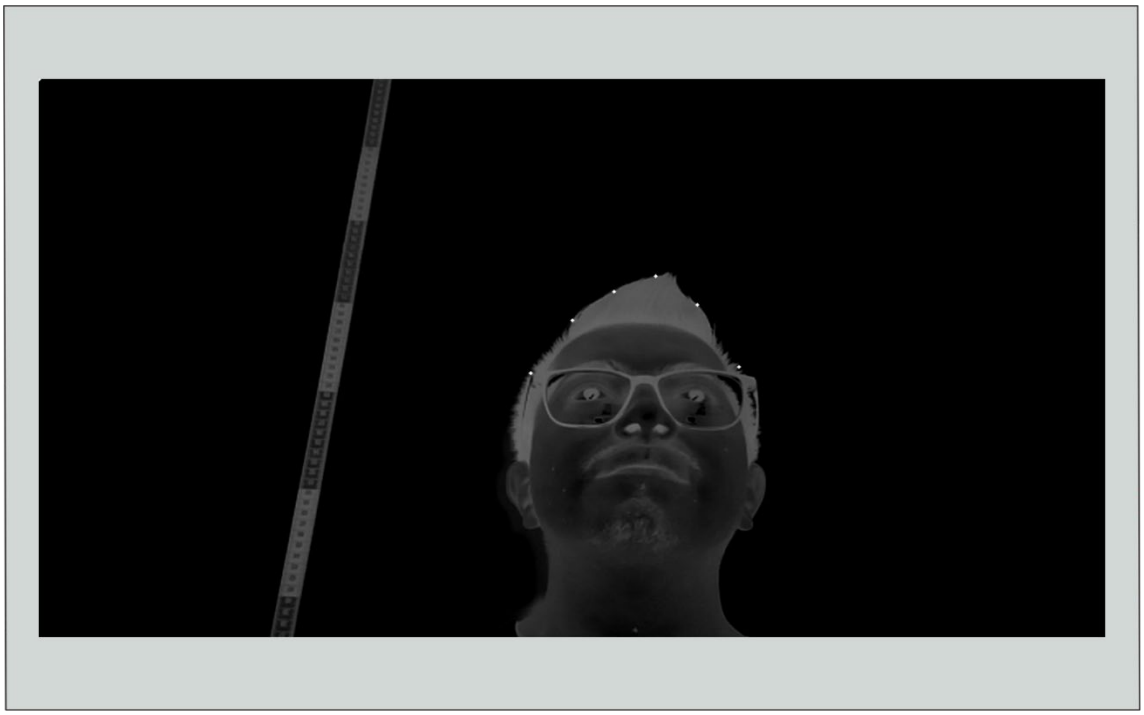

Figura 45: *Imagen de salida del algoritmo de detección facial con nueve trazas analizadas y el sujeto situado al centro (elaboración propia).*

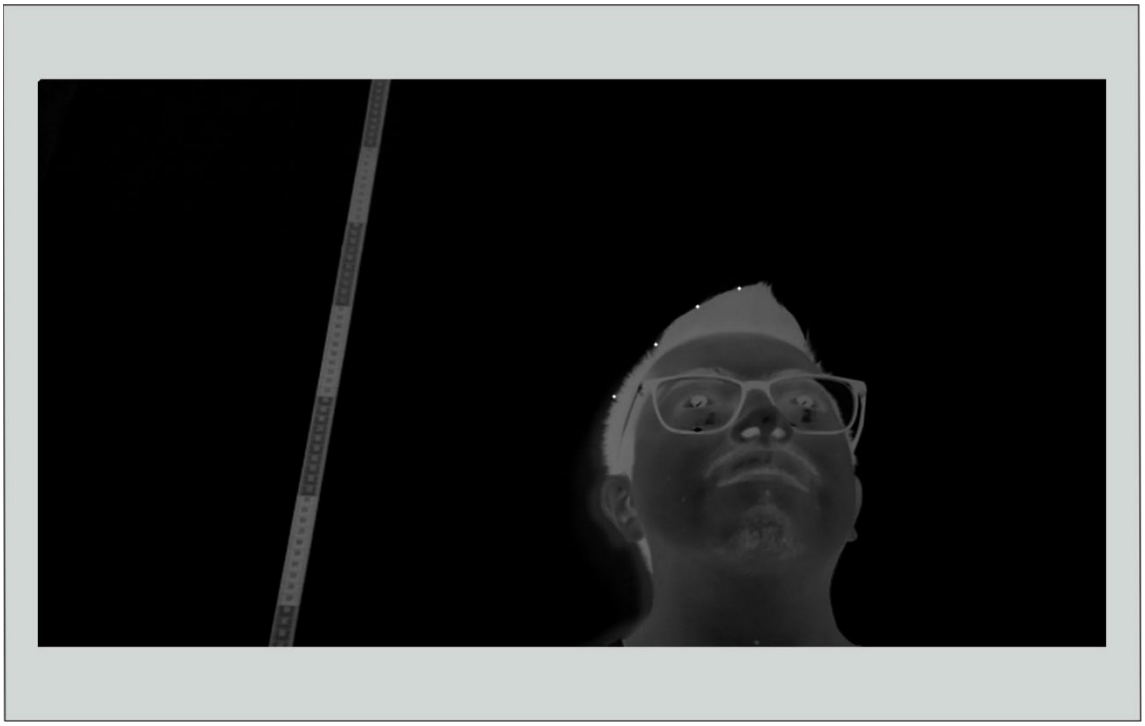

Figura 46: *Imagen de salida del algoritmo de detección facial con nueve trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

La *Figura 47* es la gráfica que representa el consumo del CPU durante las *160* iteraciones de esta prueba, los valores muestran que el consumo mayor es de *27.8%* y el menor de 0.5%, teniendo así un promedio de *25.97%* de CPU para nueve trazas evaluadas. Por otro lado, la *Figura 48* es la gráfica que representa el tiempo de ejecución del algoritmo para cada interacción y se observa que el valor máximo es *0.18743* segundos, el mínimo *0.11633* segundos y el promedio es *0.13086* segundos.

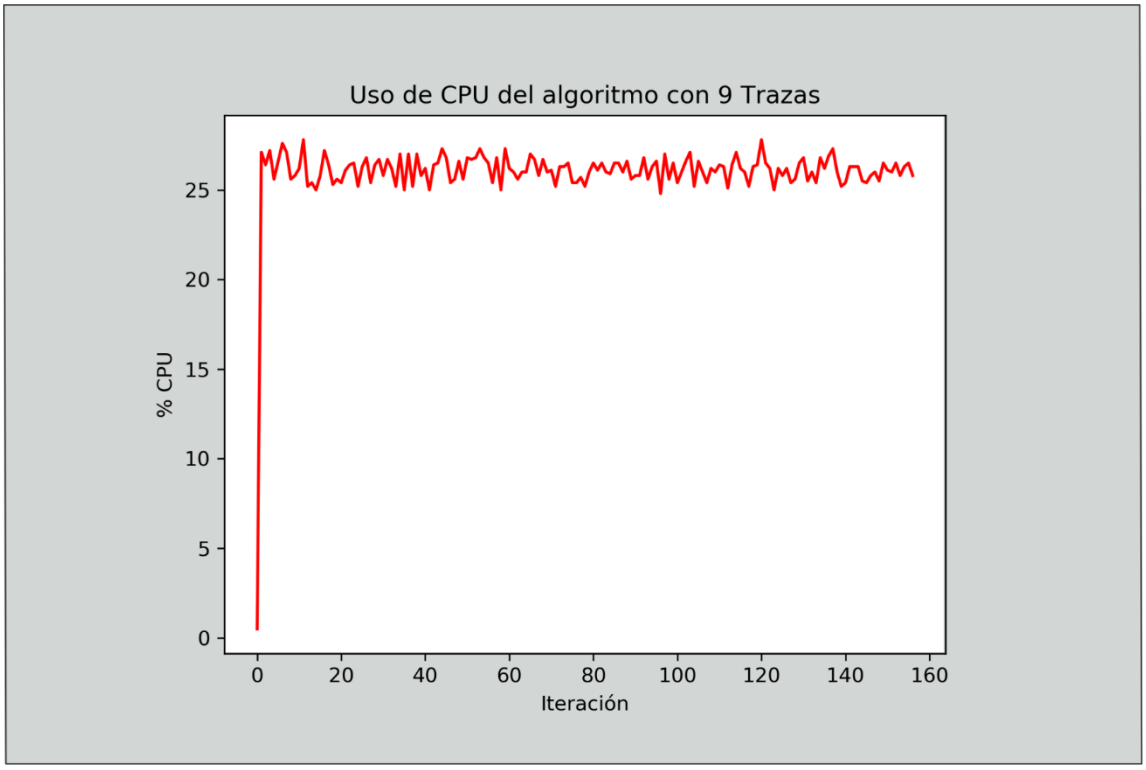

Figura 47: *Imagen de salida del algoritmo de detección facial con nueve trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

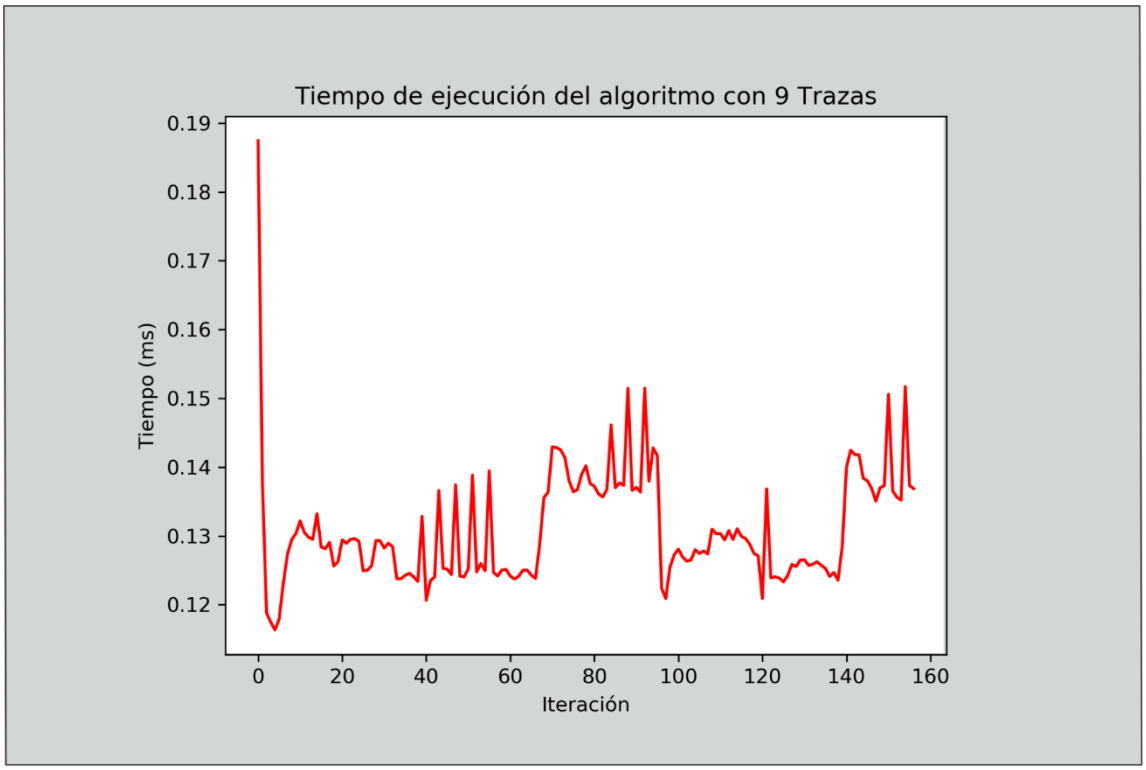

Figura 48: *Imagen de salida del algoritmo de detección facial con nueve trazas analizadas y el sujeto situado a la derecha (elaboración propia).*

4.6 Comparación de funcionamiento y rendimiento del algoritmo

En la sección anterior se ha realizado una serie de pruebas del algoritmo donde se mide el porcentaje de uso de CPU y el tiempo de ejecución, en esta sección se compara con el algoritmo de *Viola-Jones* implementado en *python*, cumpliendo las mismas condiciones iniciales que el algoritmo de detección facial desarrollado para esta tesis. Para lograr la comparativa, primero se muestra una serie de pruebas del algoritmo ejecutado en la *Raspberry Pi* y guardando los valores del uso de CPU y de tiempo de ejecución en diferentes gráficos además de mostrar un recuadro de color verde que corresponde a la región facial detectada por el algoritmo *Viola-Jones*.

4.6.1 Primera prueba Viola-Jones

Esta primera prueba se realizó como en las pruebas de la sección anterior, es decir que el sujeto de pruebas aparece en tres posiciones distintas frente a la cámara. El número de iteraciones fue de *187* y se capturaron cinco imágenes, de las cuales ninguna se muestra un cuadro de color verde encerrando el rostro. Se muestran tres imágenes con el sujeto de prueba desplazado a la izquierda, centro y derecha.

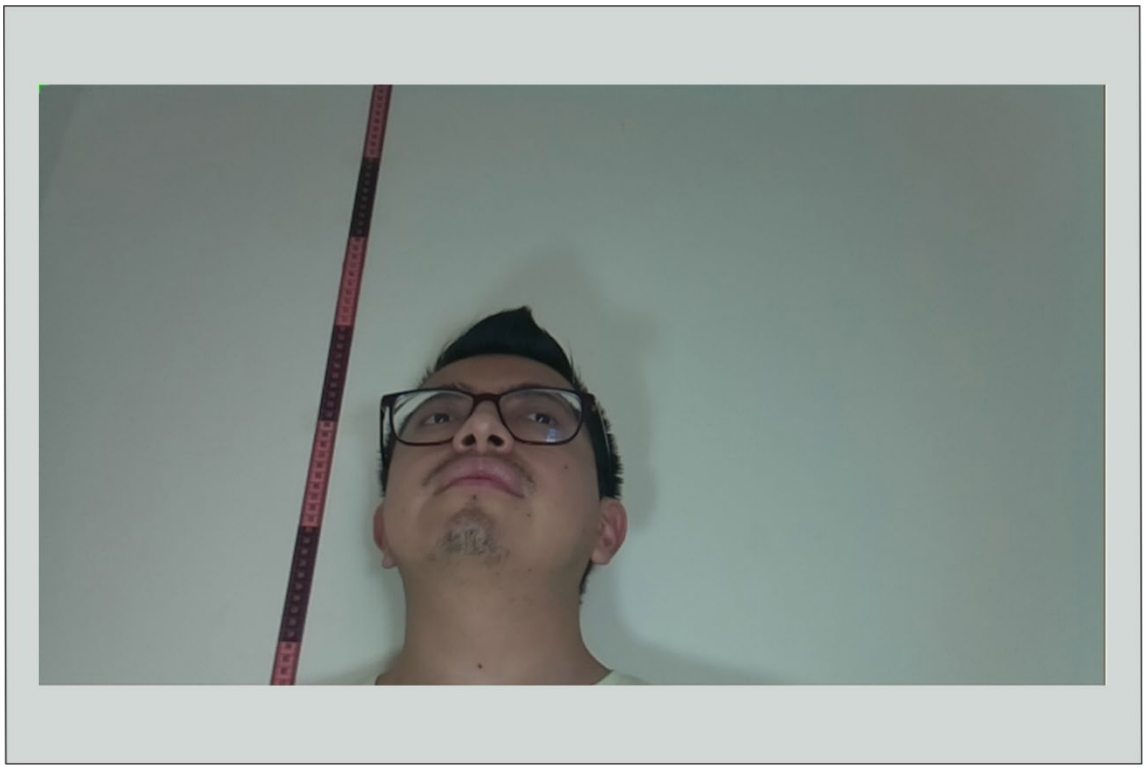

Figura 49: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas a la derecha (elaboración propia).*

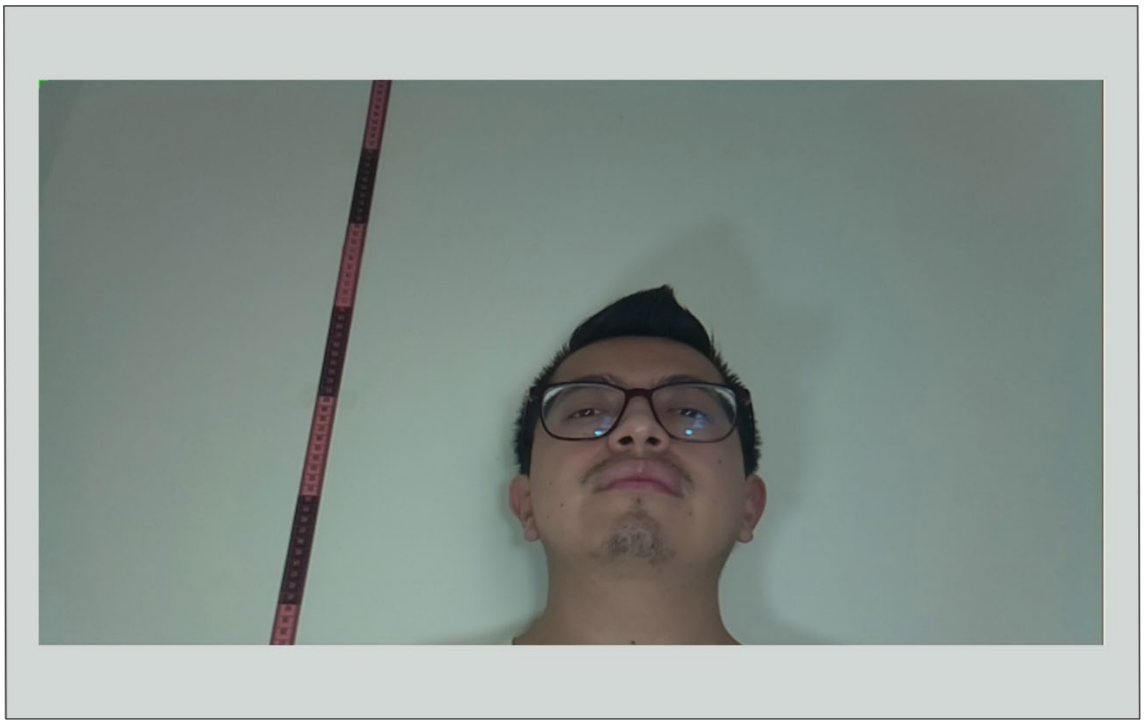

Figura 50: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas al centro (elaboración propia).*

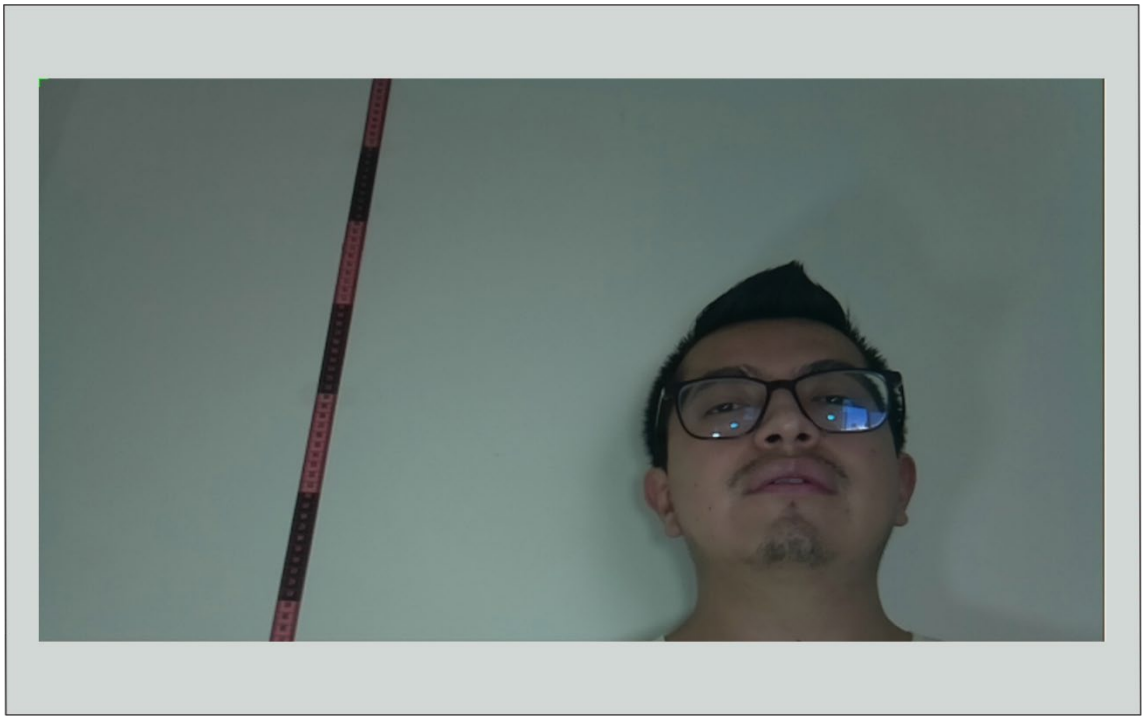

Figura 51: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas a la izquierda (elaboración propia).*

En nuestra primera prueba no se obtuvieron los resultados buscados en cuanto a la detección facial se refiere, pero sí la información necesaria para conocer el porcentaje de uso del CPU y el tiempo de ejecución de este algoritmo. Para el porcentaje de uso se obtuvo un valor máximo de *79.7%*, un valor mínimo de *5.8%* con un promedio de *71.56%,* por otro lado, el tiempo de ejecución máximo y mínimo fue de *0.66210* y *0.07542* segundos respectivamente y con un promedio de *0.16535* segundos.

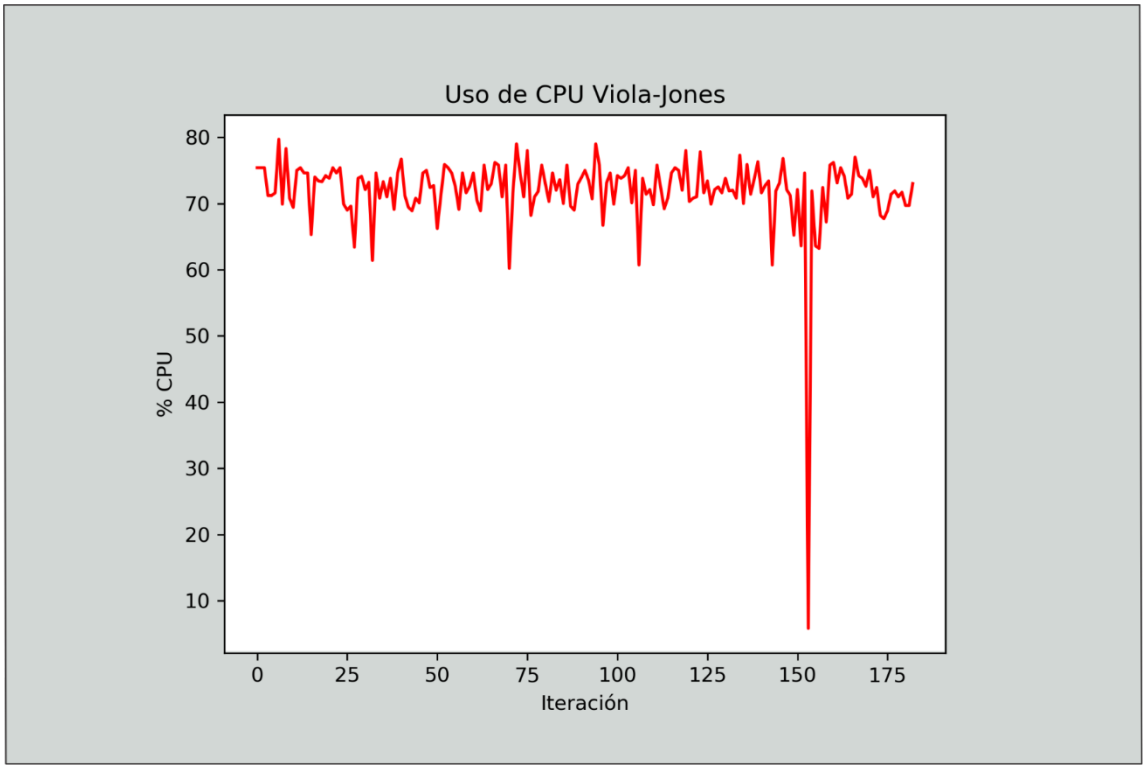

Figura 52: *Gráfica de porcentaje de uso del CPU durante la ejecución del algoritmo Viola-Jones (elaboración propia).*

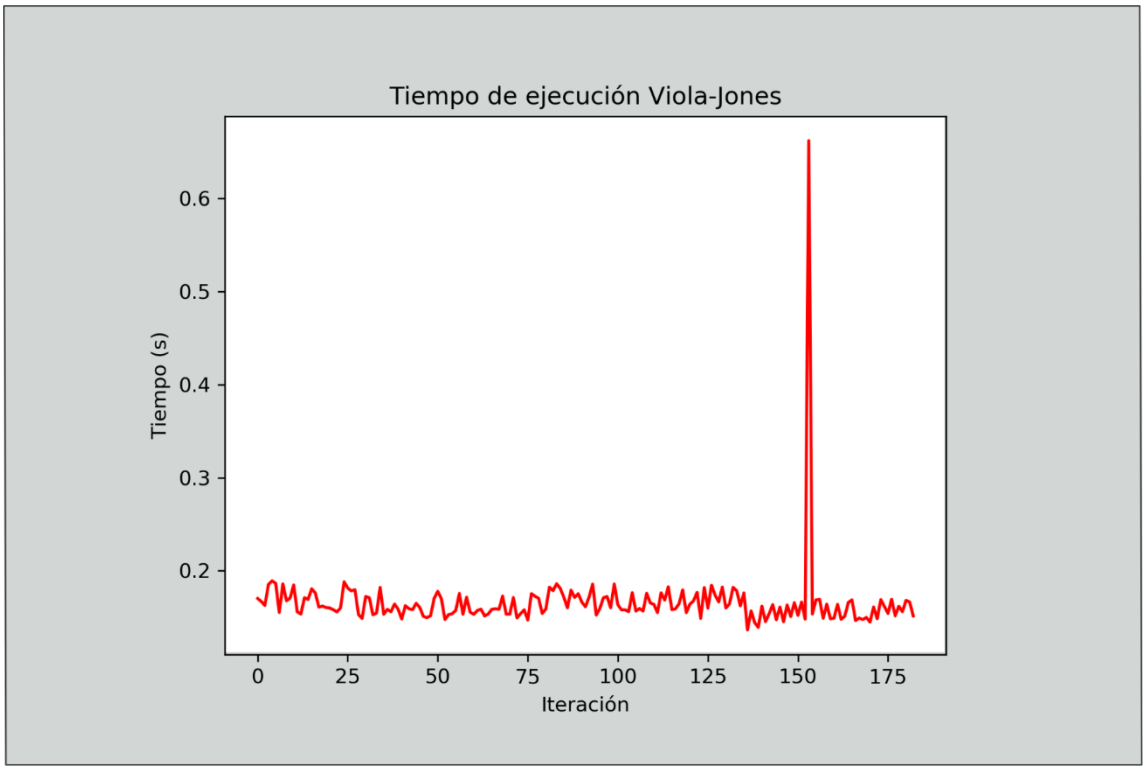

Figura 53: *Gráfica de tiempo de ejecución del algoritmo Viola-Jones (elaboración propia).*

4.6.2 Segunda prueba Viola-Jones

En la segunda prueba se analizaron *199* iteraciones y se capturaron *5* imágenes, de las cuáles solo en *1* aparece el rectángulo de color verde encerrando el rostro. Comparando entre las imágenes de ambas pruebas, en la única que contiene el rectángulo se mira directamente a la cámara, como se puede apreciar en las siguientes imágenes.

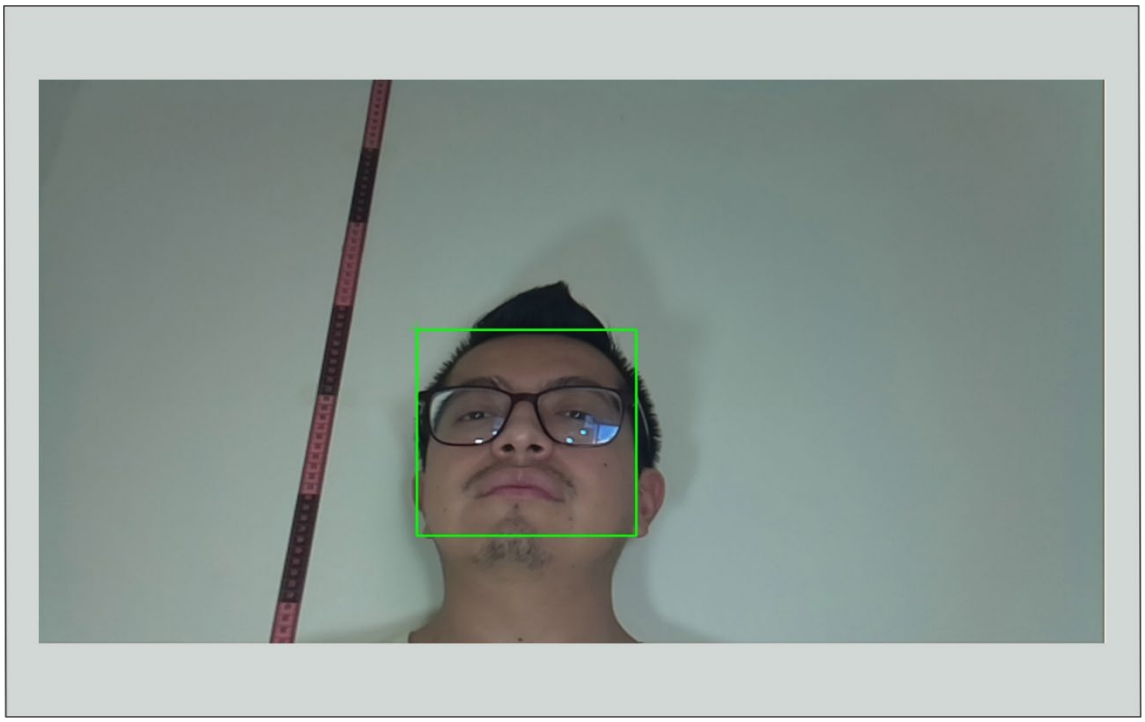

Figura 54: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas a la derecha (elaboración propia).*

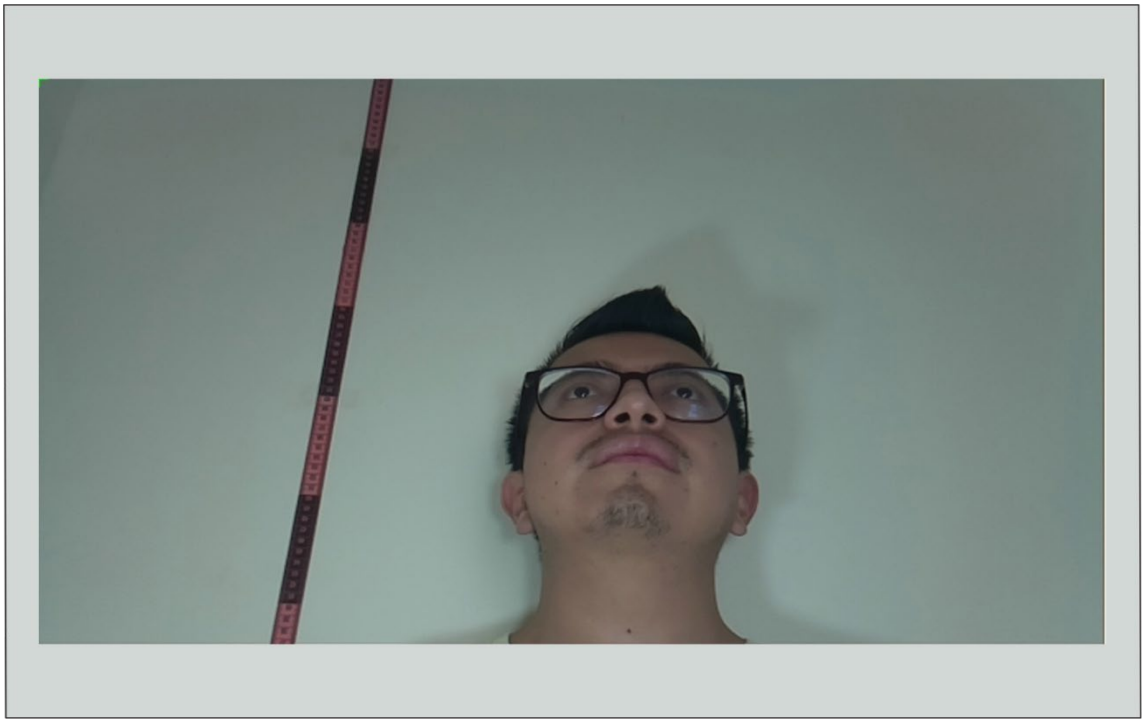

Figura 55: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas al centro (elaboración propia).*

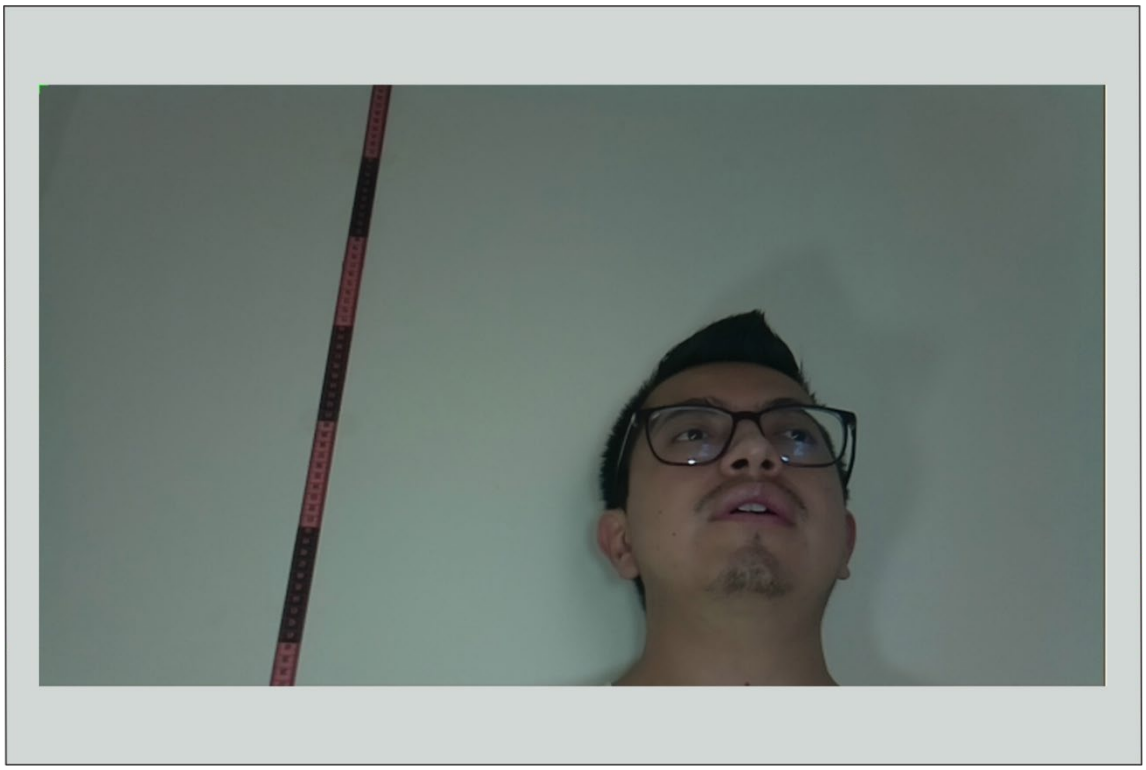

Figura 56: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas a la derecha (elaboración propia).*

Los resultados obtenidos para el porcentaje de uso del CPU y el tiempo de ejecución de esta prueba fue un valor máximo de *81.0 %*, un valor mínimo de *4.7 %* con un promedio de *72.79%,* por otro lado, el tiempo de ejecución máximo y mínimo fue de *0.66199* y *0.14374* segundos respectivamente y con un promedio de *0.16511* segundos.

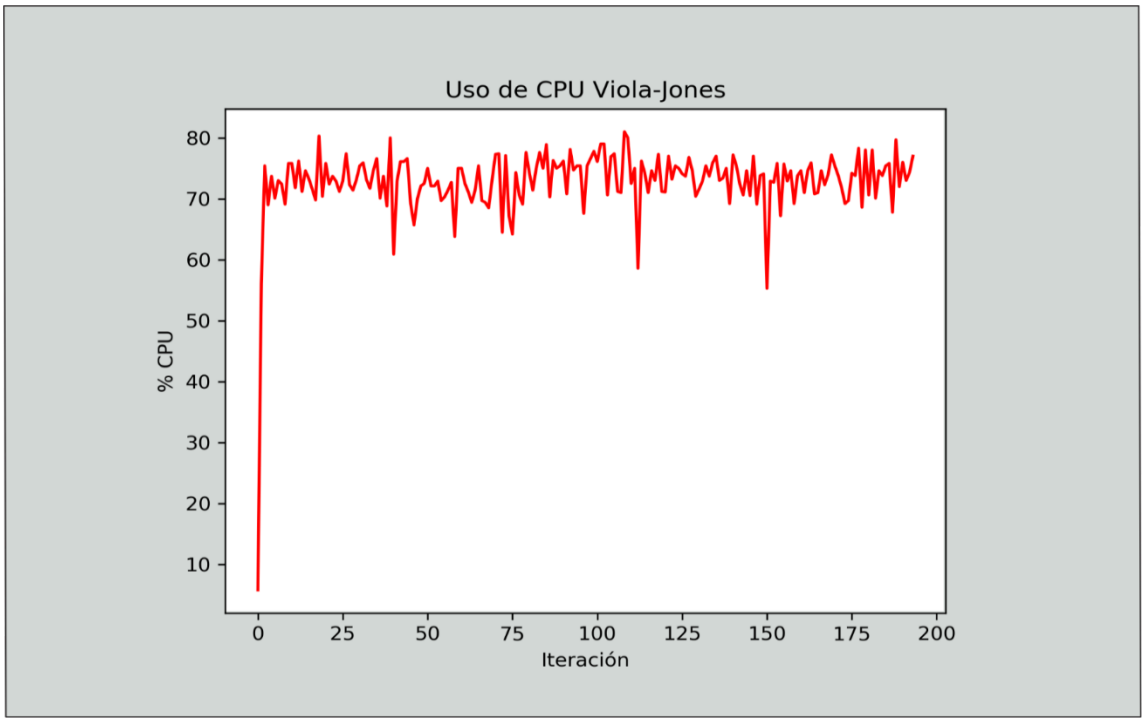

Figura 57: *Gráfica de porcentaje de uso del CPU durante la ejecución del algoritmo Viola-Jones (elaboración propia).*

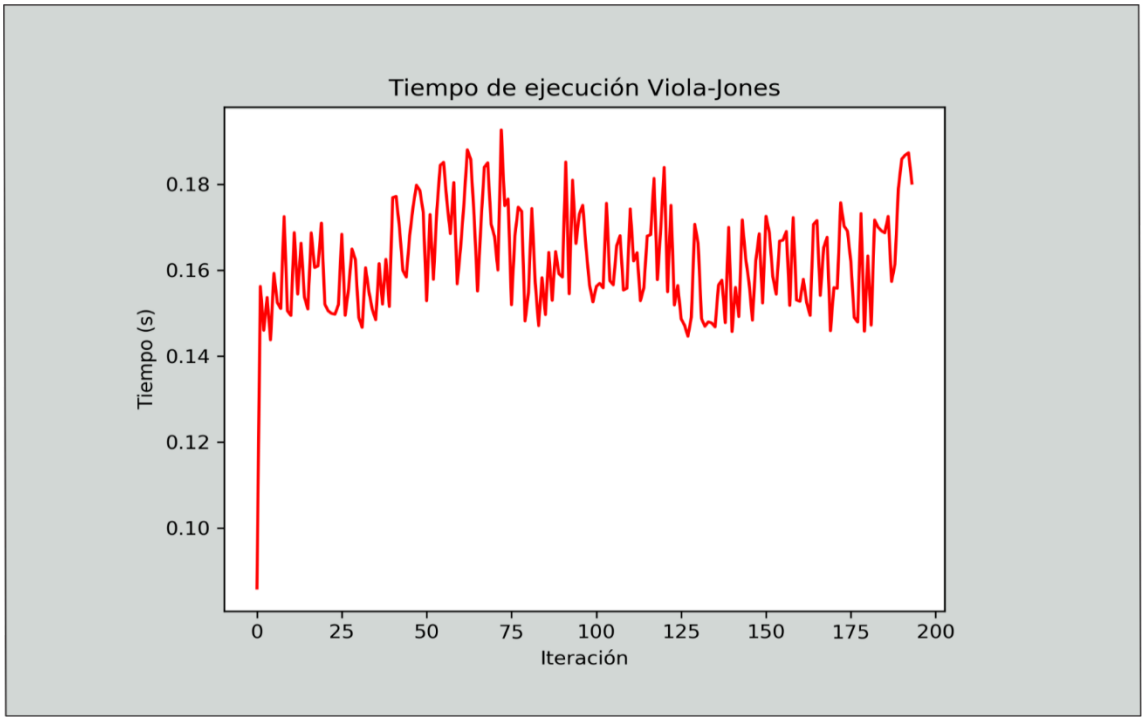

Figura 58: *Gráfica de tiempo de ejecución del algoritmo Viola-Jones (elaboración propia).*

4.6.3 Tercera prueba Viola-Jones

Para la última prueba, teniendo en cuenta que había que mirar a la cámara para que nuestra detección de rostro fuera aplicada a las imágenes y tras analizar *282* iteraciones del algoritmo y sacado *7* imágenes se obtuvo unos resultados esperados.

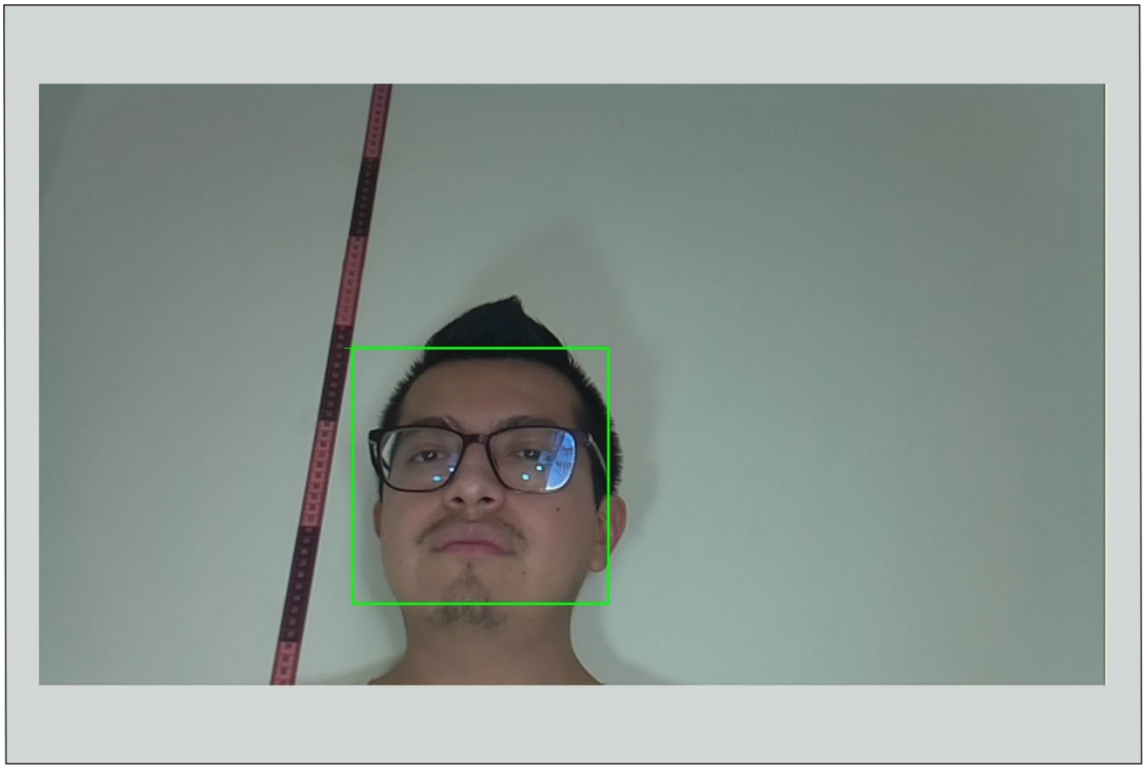

Figura 59: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas a la derecha (elaboración propia).*

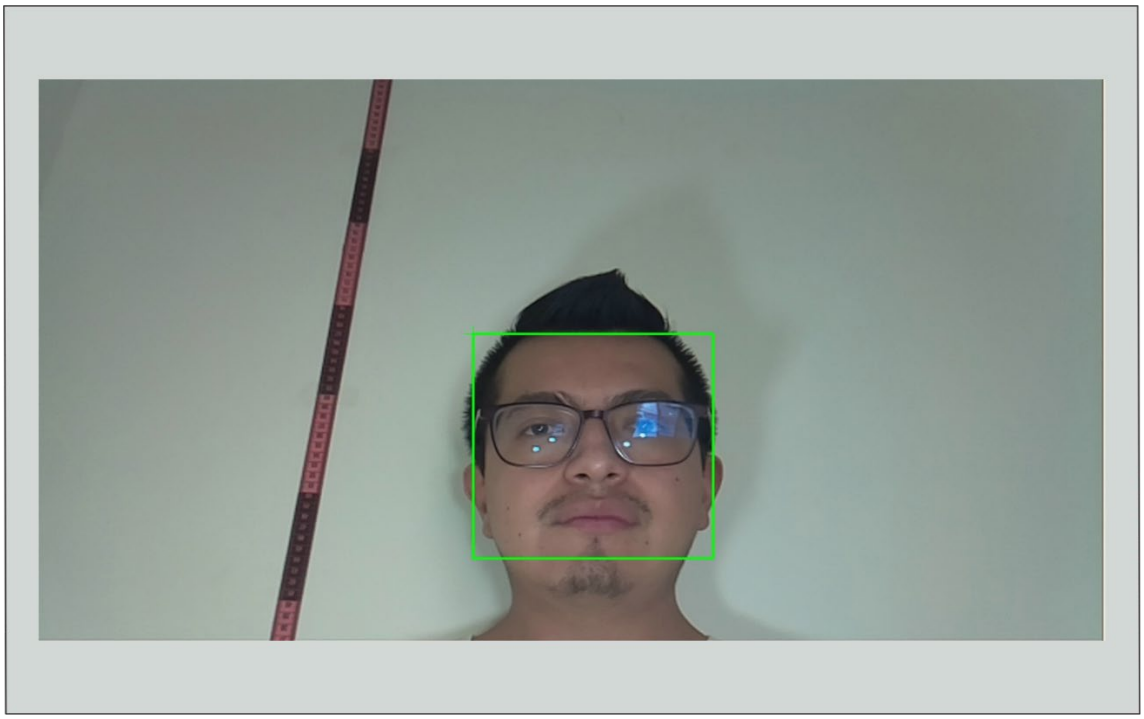

Figura 60: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas al centro (elaboración propia).*

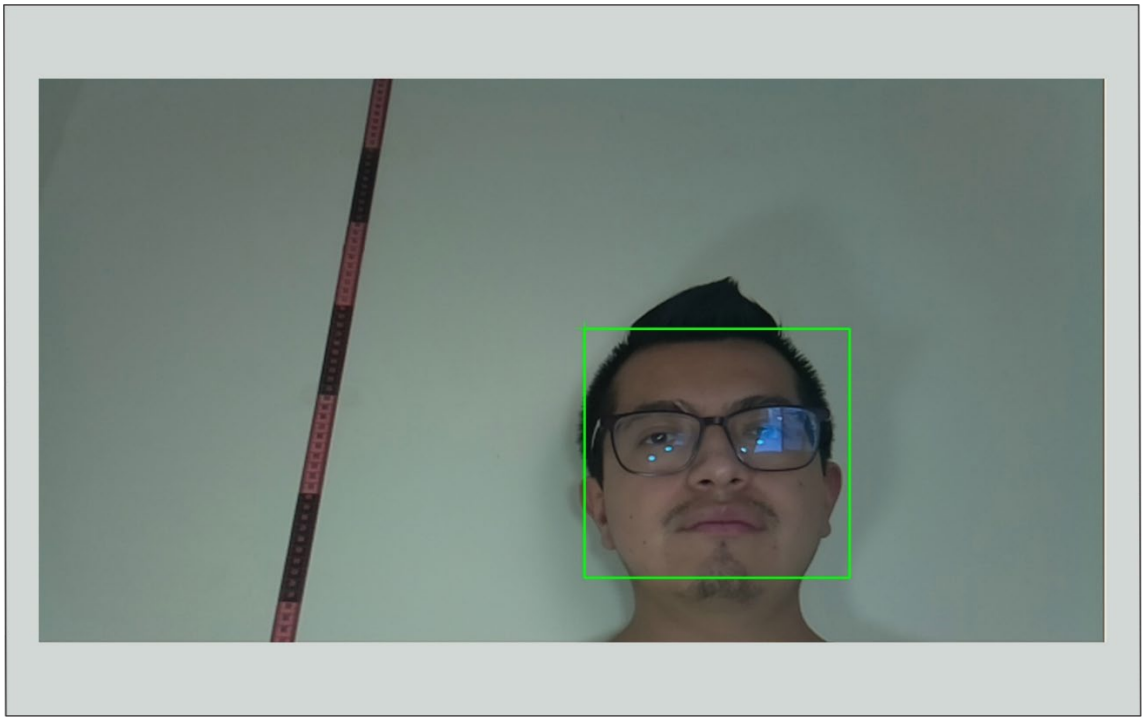

Figura 61: *Imagen de salida del algoritmo Viola-Jones con sujeto de pruebas a la derecha (elaboración propia).*

Los resultados obtenidos para el porcentaje de uso del CPU y el tiempo de ejecución de esta prueba fue un valor máximo de *81.0 %*, un valor mínimo de *9.8 %* con un promedio de *72.81%,* por otro lado, el tiempo de ejecución máximo y mínimo fue de *0.19832* y *0.10623* segundos respectivamente y con un promedio de *0.16679* segundos.

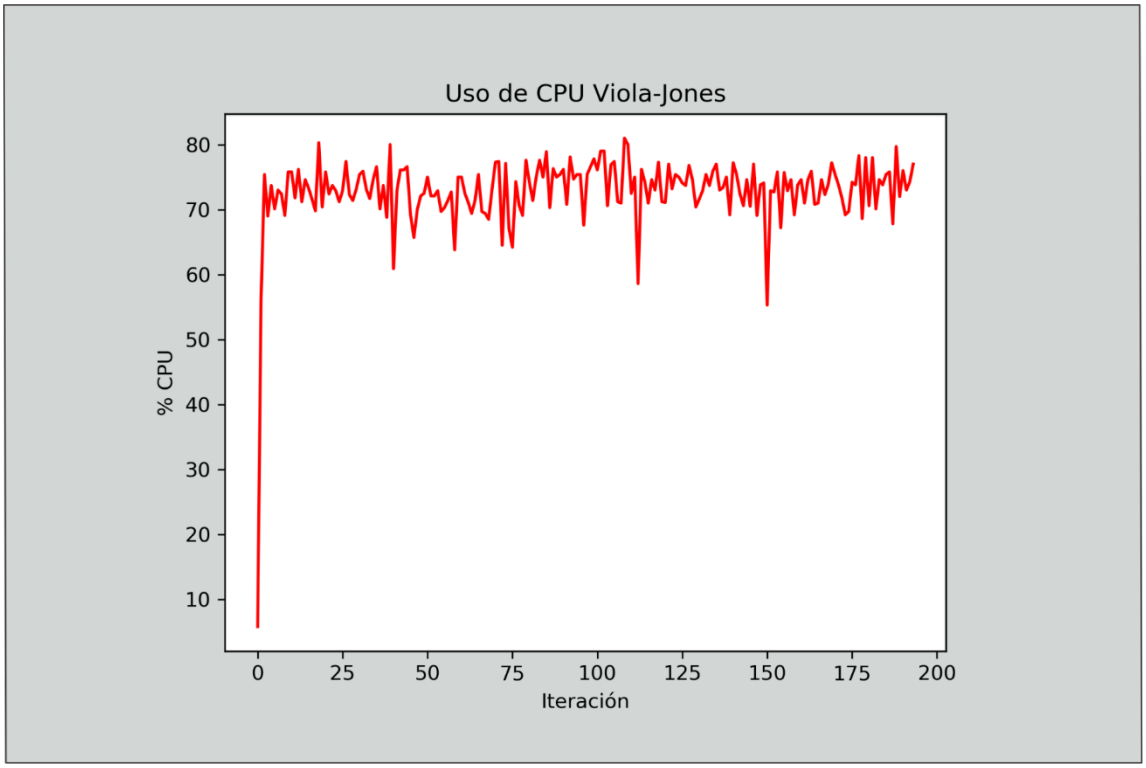

Figura 62: *Gráfica de porcentaje de uso del CPU durante la ejecución del algoritmo Viola-Jones (elaboración propia).*

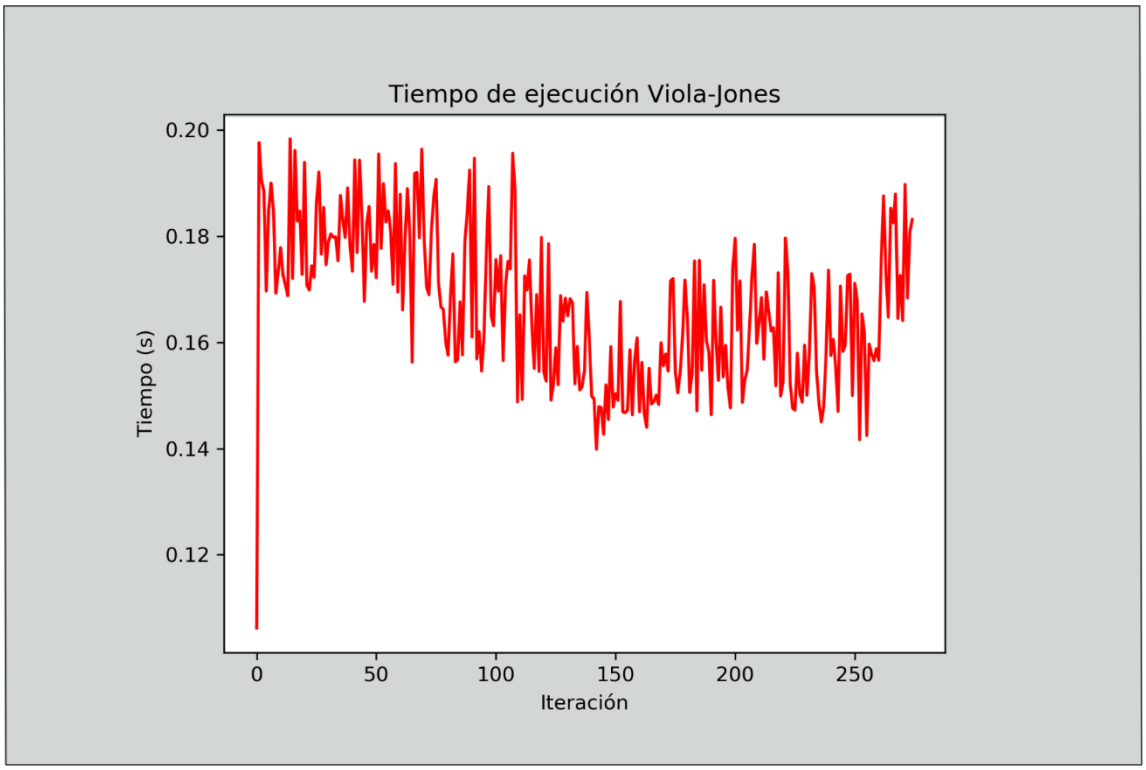

Figura 63: *Gráfica de tiempo de ejecución del algoritmo Viola-Jones (elaboración propia).*

Ahora que se han obtenido los resultados de las pruebas y graficado algunos resultados, en el siguiente capítulo se comparan y se concluye el trabajo de investigación.

La *tabla 3* representa una comparación más legible entre las cuatro pruebas hechas con el algoritmo de detección facial *Fonseca-Salas* y las tres pruebas hechas con el algoritmo de *Viola-Jones*. La información que contiene son los valores máximos, mínimos y promedios del porcentaje de uso del CPU y del tiempo, además de un campo nombrado *detección* para indicar si en la prueba se logró la detección facial. En la columna *Prueba* se definió la nomenclatura *TX* para las cuatro pruebas realizadas con el algoritmo desarrollado en este trabajo (donde *X* varía dependiendo el número de la prueba), por otro lado, para las pruebas del algoritmo *Viola-Jones* se definió la nomenclatura *VJX* (donde *X* varía dependiendo el número de la prueba del algoritmo)

***Tabla 3**: Representación de las pruebas de rendimientos de ambos algoritmos.*

Prueba	% CPU Max	% CPU Min	% CPU Prom.	T. Max	T. Min.	T. Prom.	Detección
T3	29.4	4.7	26.41	0.06662	0.03212	0.04366	Sí
T5	29.5	0.6	26.22	0.09670	0.05708	0.07358	Sí
T7	28.5	0.7	26.005	0.27237	0.08692	0.10133	Sí
T9	27.8	0.5	25.97	0.18743	0.11633	0.13086	Sí
VJ1	79.7	5.8	71.56	0.66210	0.07542	0.16535	No
VJ2	81	4.7	72.79	0.66199	0.14374	0.16511	No
VJ3	81	9.8	72.81	0.19832	0.10623	0.16679	Sí

Por último, las *figuras 64 y 65* representan la información de la *tabla 3*, donde se aprecia que el porcentaje promedio del uso de CPU y del tiempo para ambos algoritmos. El algoritmo *Viola-Jones* sin duda es el que más recursos usa (entre un *70% y 80%*) impactando directamente en el consumo del CPU del dispositivo, por otro lado, el factor tiempo está alrededor de 0.165 segundos, además se encontró que este algoritmo puede estar siendo ejecutado y no realizar la detección facial correctamente (incluso puede ser nula).

El algoritmo de detección facial *Fonseca-Salas* obtiene valores menores de rendimiento para el uso de CPU, por debajo de un *30%* a pesar de que se incrementan el número de trazas que se evalúan y el tiempo de ejecución tiene en el peor de los casos un 23% de mejora (cuando son 9 trazas evaluadas) tendiendo una mejora de 300% pero sin ser asertivo con la detección facial. La tendencia es que, a mayor número de trazas a evaluar, el porcentaje de uso de CPU baja y el tiempo de ejecución incrementa. Y, por último, en la gráfica se observa que cuando se evalúan 7 trazas, los valores de uso de CPU y tiempo de ejecución son eficientes respecto al algoritmo *Viola-Jones* y además logra la detección facial.

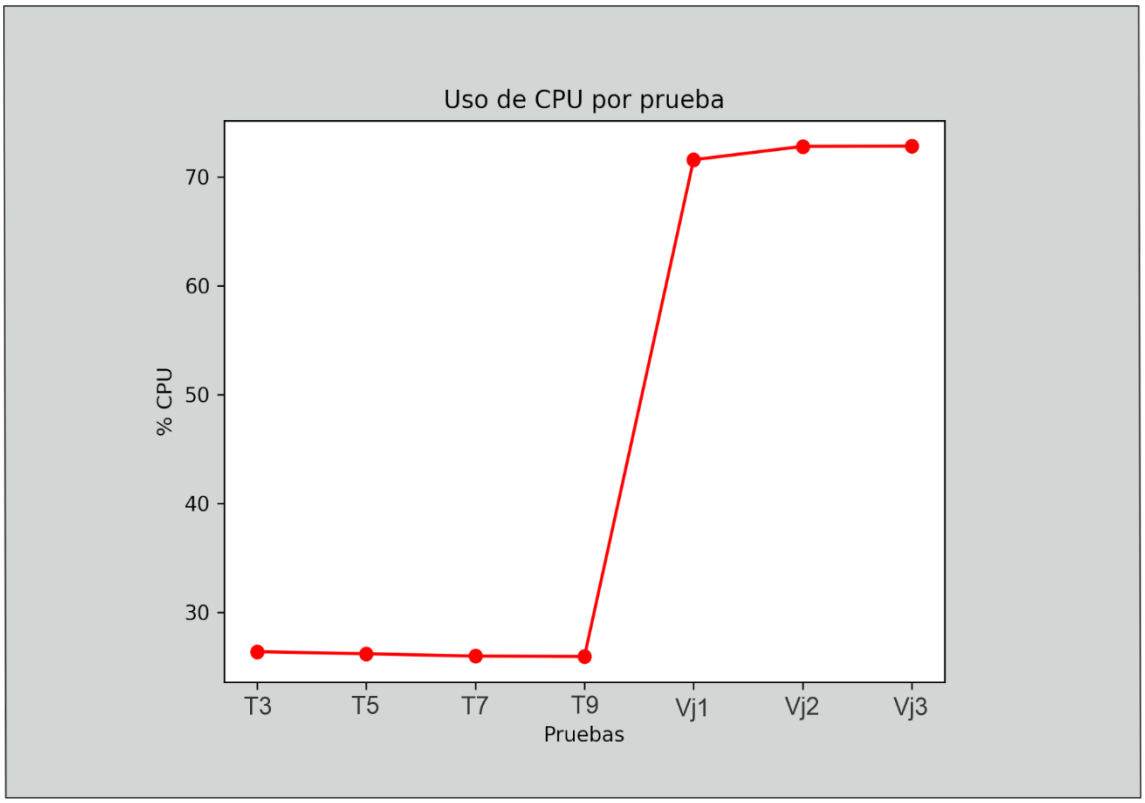

***Figura 64**: Gráfica de valores promedio del porcentaje de uso del CPU para cada una de las pruebas de ambos algoritmos (elaboración propia).*

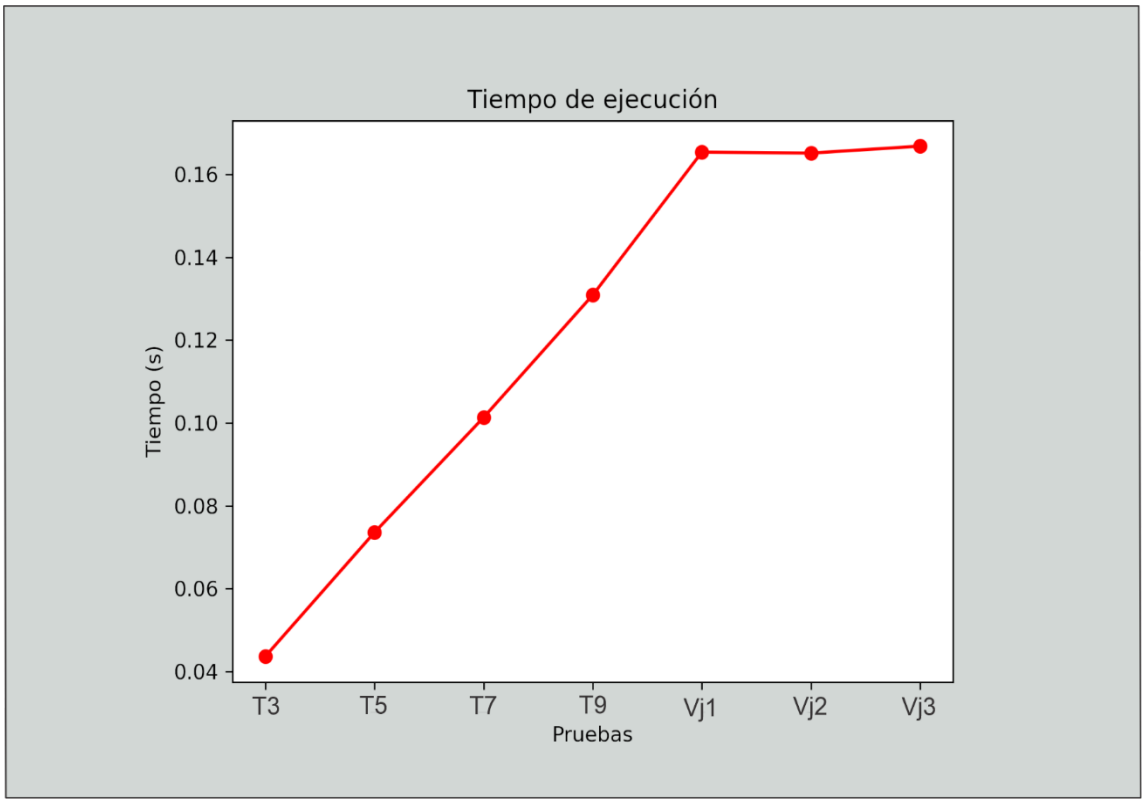

***Figura 65**: Gráfica de valores promedio del tiempo de ejecución para cada una de las pruebas de ambos algoritmos (elaboración propia).*

5. CONCLUSIONES

En el desarrollo de esta tesis se planteó el objetivo de desarrollar un algoritmo de detección facial y controlar la automatización de un atril a bajo consumo de recursos de cómputo. La metodología empleada en este trabajo se explica en el capítulo 3 y los resultados son documentados en el capítulo 4. En virtud de lo anterior se tienen las siguientes conclusiones.

El objetivo de desarrollar un nuevo algoritmo de detección facial se ha cumplido. El análisis de las trazas definidas en una imagen genera como resultado un arreglo de n coordenadas que indican el contorno superior de la región facial de interés.

La hipótesis se verifica verdadera dado que los porcentajes promedio de uso de CPU durante la ejecución de algoritmo desarrollado fueron de entre *25%* y *26%*. Por otro lado, los porcentajes máximos alcanzados en las pruebas de rendimiento estuvieron por debajo del *30%*.

La metodología empleada contribuyó a lograr los valores obtenidos, ya que al trabajar con imágenes en escala de grises (una capa) la cantidad de información procesada fue reducida en una tercera parte. Además, el uso de las trazas en la imagen también reduce aún más la cantidad de información a procesar.

Los resultados del *capítulo 4* muestran que el consumo de CPU del algoritmo de detección facial alcanza un valor arriba de *25%* y menor al *30%.* Por otro lado, los valores obtenidos con la ejecución de un algoritmo de detección facial de uso común e implementado en el mismo lenguaje de programación a través de la biblioteca *OpenCV* se encuentran entre el *70%* y *80%* de consumo de CPU. Ambos algoritmos ejecutados en una *Raspberry Pi 3 B+*.

Al realizar la implementación de este algoritmo, el uso del CPU bajará considerablemente y otras instrucciones serán ejecutadas en paralelo. Por ejemplo, con la implementación del algoritmo de uso común: el atril se encuentra ejecutando la detección facial en tiempo real (porcentaje de CPU arriba del *70%)* si se le mandara una indicación en la pantalla al ponente habrá un retraso y puede que no sea mostrado, debido al cuello de botella para procesar la información, Con respecto al algoritmo desarrollado, este cuello de botella en el procesador no sucedería por su porcentaje de CPU utilizado.

Este nuevo algoritmo de detección facial puede ser aplicado para ubicación de rostros en imágenes y videos, además para la extracción de características del rostro con menos iteraciones, es decir realizar una segmentación de los rostros, extraerlos y posteriormente aplicar un proceso de reconocimiento facial a menor costo.

BIBLIOGRAFÍA

Crumpler, W. (2020). How Accurate are Facial Recognition Systems – and Why Does It Matter? Retrieved from https://www.csis.org/blogs/technology-policy-blog/how-accurate-are-facial-recognition-systems-%E2%80%93-and-why-does-it-matter

Degtyarev, N., & Seredin, O. (2010, 2010//). *Comparative Testing of Face Detection Algorithms.* Paper presented at the Image and Signal Processing, Berlin, Heidelberg.

Eduardo A. D. da Silva, & Mendonça, G. V. (2005). 4 - Digital Image Processing. *The Electrical Engineering Handbook*, 891-910.

Hjelmås, E., & Low, B. K. (2001). Face Detection: A Survey. *Computer Vision and Image Understanding, 83*(3), 236-274. doi:https://doi.org/10.1006/cviu.2001.0921

Martínez Pérez Juan Vicente, & Joan, L. P. J. (2012). Sistema de reconocimiento facial y realidad aumentada para dispositivos móviles. *3c Tic*, 7-16.

Morales, P. (2021). Instagram, las cifras imprescindibles para el 2021. Retrieved from https://blog.digimind.com/es/insight-driven-marketing/instagram-cifras-imprescindibles-2021

Moreano, J. A. C., Pulloquinga, R. H. M., Lagla, G. A. F., Chisag, J. C. C., & Pico, O. A. G. (2017). Reconocimiento facial con base en imágenes.

Rafael C. Gonzalez, & Woods, R. E. (2018). *Digital Image Processing* (4 ed.): Pearson.

UIT-R. (2011). Recomendación UIT-R BT.601-7. 20.

Villa, M. M., & Yáñez, R. E. S. (2017). Fundamento de la reducción de ruido en imágenes. *Jóvenes en la ciencia, 3*(2).

Viola, P., & Jones, M. J. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision, 57*(2), 137-154. doi:10.1023/B:VISI.0000013087.49260.fb

GLOSARIO

Algoritmo: conjunto de instrucciones definidas para resolver un problema y/o realizar un cálculo.

Atril: soporte para sostener documentos, libros entre otros objetos.

Automatizado: tecnología aplicada a un proceso que reduce la intervención humana.

Calibración: definir un estándar de medición para determinar la relación entre el valor mostrado por un instrumento de medición y el valor verdadero.

CPU: Unidad Central de Procesamiento.

Cuantificar: expresar numéricamente una magnitud.

Despliegue: actividades necesarias para que un software esté disponible para su uso.

Detección facial: ubicar un rostro en una imagen o video.

Filtro: operaciones matemáticas aplicada a imágenes para resaltar o suprimir una característica.

Hardware: término utilizado para referirse a componentes físicos que conforman un equipo de computación y/o accesorios.

Imagen digital: fotografías digitalizadas tomadas de una escena, paisaje, documento, persona, etc.

Intensidad: valor numérico que representa un color en una imagen.

Pixel: la menor unidad homogénea en color que forma parte de una imagen.

LCD: es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora.

LED: fuente de luz constituida por un material semiconductor dotado de dos terminales.

Muestrear: representación de una imagen mediante un conjunto de muestras de pixeles igualmente espaciadas, representas en forma matricial con dimensiones *NxM*.

OpenCV: biblioteca libre de visión artificial

PLA: polímero o bioplástico utilizado para la creación de envases.

Python: lenguaje de alto nivel de programación interpretado.

Raspberry Pi 3 B+: minicomputadora de placa única de bajo costo.

Raspbian: sistema operativo oficial de los sistemas Raspberry Pi.

Reconocimiento facial: ubicar y reconocer a quien pertenece un rostro dentro de una imagen o video.

Resolución: número de pixeles por pulgada que contiene una imagen.

RGB: composición del color en términos de la intensidad de los colores primarios de la luz.

SoC: proviene del acrónimo en inglés System-on-a-Chip y que hace referencia a la tendencia de implementar en un chip tantas funciones como fuera posible.

Software: programas informáticos que hacen posible la ejecución de tareas específicas dentro de un computador.

Técnicas heurísticas: procedimientos encargados de encontrar o construir algoritmos con buena velocidad para ser ejecutados.

Tratamiento de imágenes: técnica que permite modificar una imagen con el objetivo de lograr una mayor calidad y/o resaltar o remover características.

Tren de pulsos: serie de pulsos de corriente directa continuos por un intervalo de tiempo.