UNIVERSIDAD NACIONAL PEDRO RUIZ GALLO

ESCUELA PROFESIONAL DE INGENIERIA ELECTRONICA

"SISTEMA DE MONITOREO PARA DETECCION DE SOMNOLENCIA EN CHOFERES DE RUTAS INTERPROVINCIALES DE EMPRESAS DE TRANSPORTE DE CHICLAYO"

PARA OBTENER EL TITULO PROFESIONAL DE INGENIERO ELECTRONICO

- Bach. Augusto Joel Custodio Effio
- Bach. Leonardo Tarrillo Nuntón

ASESORA:

Mg. Ing. Lucía Isabel Chamán Cabrera

Lambayeque, 2021

UNIVERSIDAD NACIONAL PEDRO RUIZ GALLO

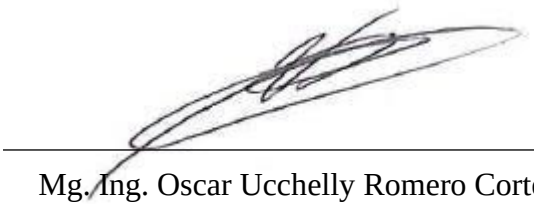
FACULTAD DE CIENCIAS FISICAS, MATEMATICAS Y COMPUTACION

TESIS

"SISTEMA DE MONITOREO PARA DETECCION DE SOMNOLENCIA EN CHOFERES DE RUTAS INTERPROVINCIALES DE EMPRESAS DE TRANSPORTE DE CHICLAYO"

Aceptada por la Escuela Profesional de Ingeniería Electrónica.

Ing. Segundo Francisco Segura Altamirano
PRESIDENTE



Mg. Ing. Oscar Ucchelly Romero Cortez
SECRETARIO

Mg. Ing. Martín Augusto Nombera Lossio
VOCAL

Lambayeque, 2021

ACTA DE SUSTENTACIÓN VIRTUAL Nº009-2022-D/FACFyM

Siendo las 10:10 am del día 25 de febrero del 2022, se reunieron vía plataforma virtual, meet.google.com/vep-ftnf-unh los miembros del jurado evaluador de la Tesis titulada:

"Sistema de Monitoreo para Detección de Somnolencia en Choferes de Rutas Interprovinciales de Empresas de Transporte de Chiclayo. "

Designados por Resolución N°792-2019-D/FACFyM de fecha 12 de Junio del 2019

Con la finalidad de evaluar y calificar la sustentación de la tesis antes mencionada, conformada por los siguientes docentes:

Ing. Segundo Francisco Segura Altamirano Presidente
Mg. Ing. Oscar Ucchelly romero Cortez Secretario
Mg. Ing. Martín Augusto Nombera Lossio Vocal

La tesis fue asesorada por la Dra. Lucia Isabel Chaman Cabrera, nombrado por Resolución N° 500-2019-D/FACFyM de fecha 12 de abril del 2019.

El Acto de Sustentación fue autorizado por Resolución N°157-2022-VIRTUAL-D/FACFyM de fecha 14 de febrero del 2022.

La Tesis fue presentada y sustentada por los Bachilleres: Custodio Effio Augusto Joel y Tarrillo Nuntón Leonardo y tuvo una duración de 30 minutos.

Después de la sustentación, y absueltas las preguntas y observaciones de los miembros del jurado se procedió a la calificación respectiva, otorgándole el Calificativo de 17 (Diecisiete) en la escala vigesimal, mención Bueno.

Por lo que quedan aptos para obtener el Título Profesional de **Ingeniero Electrónico** de acuerdo con la Ley Universitaria 30220 y la normatividad vigente de la Facultad de Ciencias Físicas y Matemáticas y la Universidad Nacional Pedro Ruiz Gallo.

Siendo las 11:40 am se dio por concluido el presente acto académico, dándose conformidad al presente acto con la firma de los miembros del jurado.

Ing. Segundo Francisco Segura Altamirano
Presidente

Mg. Ing. Oscar Ucchelly Romero Cortez
Secretario

Mg. Ing. Martín Augusto Nombera Lossio
Vocal

Mg. Ing. Lucía Isabel Chamán Cabrera
Asesor

DECLARACIÓN JURADA DE ORIGINALIDAD

Yo, Custodio Effio Augusto Joel, junto a mi compañero, Tarrillo Nuntón Leonardo, investigadores principales, y la Mg. Ing. Chamán Cabrera Lucía Isabel, asesora de nuestro trabajo de investigación denominado "Sistema de monitoreo para detección de somnolencia en choferes de rutas interprovinciales de empresas de transporte de Chiclayo", declaramos bajo juramento que este trabajo no ha sido plagiado, ni contiene datos falsos. En caso se demostrara lo contrario, asumimos responsablemente la anulación de este informe y por ende el proceso administrativo a que hubiera lugar. Que puede conducir a la anulación del titulo o grado emitido como consecuencia de este informe.

Lambayeque, Diciembre del 2021

Investigadores:

Bach. Custodio Effio Augusto Joel

Bach. Tarrillo Nuntón Leonardo

Asesora:

Mg. Ing. Chamán Cabrera Lucía Isabel

DEDICATORIA

Dedico esta tesis a Dios quien desde el cielo cuida de mi día a día y ha guiado mi camino hasta la actualidad.

A mis padres José Presbítero Tarrillo Vásquez y Rosario Francisca Nuntón Flores, por haber sido las personas que me cuidaron, formaron con mucha disciplina y motivaron a siempre estudiar y salir adelante hasta llegar a ser quien soy actualmente. A mis hermanos José Eduardo Tarrillo Nuntón y Joel Tarrillo Nuntón por acompañarme en mis años de formación como persona y darme calidez de hogar. A María Fatima Del Milagro Monteza Núñez, por siempre estar a mi lado incondicionalmente, sobre todo en los momentos más críticos de mi vida y siempre apoyarme a salir adelante y preocuparse por mi bienestar.

Leonardo Tarrillo Nuntón

Dedico esta tesis a Dios por ser mi guía y acompañarme en cada momento de mi vida y brindarme sabiduría para lograr mis objetivos.

A mi madre Guillermina Mercedes Effio Gonzales por haberme enseñado el valor del sacrificio y acompañarme en cada paso. A mi mamita Eugenia que desde niño me cuidó y lo sigue haciendo desde el cielo. A Teresa de Jesús Gutierrez Carlos, por ser mi compañera, por su apoyo, y estar incondicionalmente en los buenos y malos momentos.

Augusto Joel Custodio Effio

AGRADECIMIENTOS

Agradezco en primer lugar a Dios por todo lo que ha sucedido en mi vida y me ha ayudado guiándome siempre para no desviarme del buen camino.

A mis padres por criarme con mucho respeto, honestidad, rectitud y amor. A ellos mis más sinceros agradecimientos, ya que gracias a ellos tuve una educación de calidad y con valores que permitieron crecer tanto personalmente como profesionalmente.

A Fatima Monteza, la persona a quien amo con todo mi corazón, ella que siempre hace que salga lo mejor de mi día a día, me apoya y aconseja en las situaciones más adversas, me brinda su amor incondicional y me permite crecer y compartir logros y metas siempre juntos.

A la ingeniera Lucía Chamán, nuestra asesora, quien desde que empezamos este proyecto siempre estuvo dispuesta a apoyarnos en todo lo que pueda estar a su alcance hasta estos momentos que ya la hemos finalizado, nuestros más sinceros agradecimientos.

Muchas gracias al ingeniero Carlos Pardo, quien desde Colombia nos guio para desarrollar este proyecto, aun cuando la pandemia afectó su salud, siempre estuvo predispuesto a darnos su consejo.

Leonardo Tarrillo Nuntón

A Dios porque sin él nada habría sido posible.

A mi madre por ser mi ejemplo de esfuerzo y superación constante, por su amor y apoyo incondicional, y ser el principal cimiento para la construcción en mi vida profesional.

A Teresa Gutierrez y Shamir Adrián, las personas que amo y que son el motor y motivo para ser mejor cada día.

A la familia por parte de mi querida madre, por cada uno de los consejos brindados y por estar pendientes de mi en cada momento

Al Señor Alfredo Gutierrez y la Señora Flor Carlos por el apoyo que me brindan desde el momento que me conocieron.

A la ingeniera Lucía Chamán, nuestra asesora, por su tiempo brindado desde que comenzamos con este proyecto, por sus consejos y enseñanzas desde mi etapa universitaria hasta el día de hoy, nuestros más sinceros y amplios agradecimientos.

Así también al ingeniero Carlos Pardo, quien a pesar de la distancia nos orientó para desarrollar este proyecto.

Augusto Joel Custodio Effio

INDICE DE CONTENIDO

INDICE DE TABLAS

INDICE DE FIGURAS

Resumen

La gran cantidad de accidentes de tránsito causados por efectos de somnolencia ocurrida por diversas causas, así como la poca oferta de sistemas de monitoreo de somnolencia en el Perú fueron los motivos para desarrollar el presente trabajo. Para el cual se definió diseñar un sistema de monitoreo, el cual mediante signos de fatiga alertará oportunamente a potenciales conductores y permitirá detectar somnolencia en los choferes de rutas interprovinciales de Chiclayo. En primera instancia se escogió el lenguaje de programación para su desarrollo, el cual se eligió Python. Posteriormente se eligió usar el entorno virtual Anaconda, en el cual se editó, desarrolló y compiló el sistema desarrollado. El sistema fue basado en OpenCv para la aplicación en visión artificial, herramientas como Dlib, TensorFlow y Keras (para el apoyo en el uso de una red neuronal pre entrenada), así como algunas librerías usadas. Mediante el uso de una cámara se analizó el estado del rostro (de frente e inclinación de este) haciendo un procesamiento digital de imágenes, así como el comportamiento ocular de cada uno de los 30 voluntarios que participaron. El estado del rostro fue analizado mediante algoritmos matemáticos tomando en consideración 68 puntos faciales y se pudo tanto detectar la presencia de un rostro, así como alinearlo si es que este se encontraba inclinado. La región que comprende los ojos fueron material de análisis para la definición de “ojos abiertos” y “ojos cerrados”, mediante el EAR (Ear Aspect Ratio). Estos definidos de tal forma una vez realizada la inferencia de la red neuronal pre entrenada. Se implementó una alarma sonora para cuando hubo presencia de somnolencia (“ojos cerrados”), la cual sirvió para alertar oportunamente al conductor. Luego de realizar todas las pruebas correspondientes al sistema de monitoreo de somnolencia se concluyó que tuvo una eficacia del 97.78%.

Palabras Claves: Somnolencia, visión artificial, procesamiento digital de imágenes, conductor.

Abstract

The large number of traffic accidents caused by effects of drowsiness occurred for various causes, as well as the limited supply of sleep monitoring systems in Peru were the reasons for developing this study. For which it was defined to design a monitoring system, which by means of signs of fatigue will promptly alert potential drivers and will allow to detect drowsiness in the drivers of interprovincial routes of Chiclayo. In the first instance, the programming language was chosen for its development, which Python was chosen. Subsequently, it was chosen to use the Anaconda virtual environment, in which the developed system was edited, developed and compiled. The system was based on OpenCv for the application in artificial vision, tools such as Dlib, TensorFlow and Keras (to support the use of a pre-trained neural network), as well as some used libraries. Through the use of a camera, the state of the face (front and tilt) was analyzed by digitally processing images, as well as the ocular behavior of each of the 30 volunteers who participated. The state of the face was analyzed using mathematical algorithms taking into account 68 facial points and the presence of a face could both be detected, as well as aligning it if it was tilted. The region that comprises the eyes were analysis material for the definition of "open eyes" and "closed eyes", using the EAR (Ear Aspect Ratio). These defined in such a way once the inference of the pre-trained neural network has been carried out. An audible alarm was implemented for when there was the presence of drowsiness (“closed eyes”), which served to promptly alert the driver. After performing all the tests corresponding to the sleepiness monitoring system, it was concluded that it had an efficacy of 97.78%.

Key Words: Drowsiness, machine vision, digital image processing, driver.

I. INTRODUCCION

Los accidentes de tránsito día a día suceden en todo el mundo, en algunos casos las pérdidas pueden ser remediables cuando se trata de un accidente no fatal, pero cuando involucra la pérdida de una vida, esta es irremediable. En tal sentido la sensibilización ante esta problemática lleva a investigar de qué manera mitigar estos eventos lamentables.

Es así que a lo largo del tiempo se ha ido desarrollando tecnologías que ayuden ante esta problemática, es el caso del software OpenCV, para poder desarrollar proyectos de visión artificial, el cual mediante el uso de diferentes clasificadores y modelos matemáticos pueden dar resultado a aplicaciones útiles en la vida diaria.

Una de sus aplicaciones más interesantes es la detección de somnolencia, partiendo desde el reconocimiento del rostro, para posteriormente monitorear el estado de los ojos, los cuales nos dan las características esenciales para determinar si una persona se encuentra bajo signos de fatiga causadas por diversas causas como puede ser cansancio, efectos del alcohol, consumo de drogas, etc.

Actualmente no se lleva un monitoreo eficiente en cuanto a la detección de somnolencia a los conductores a nivel regional y nacional, esto debido a la poca oferta de autos con un sistema ya integrado para esta función, y si los tuviera estos sólo son accesibles a una sección de mercado demasiado reducida debido a su alto costo, por lo que incluso su demanda es bastante baja. En tal sentido la cultura de los conductores acerca del tema es muy escasa, debido a la inaccesibilidad a estos.

Se sabe que, a nivel nacional los accidentes de tránsito son causados por: un 29.37% por la imprudencia del conductor, 28.85% por el exceso de velocidad y un 7.67% por el consumo de alcohol. De las cuales en la gran mayoría de accidentes se registran por somnolencia. Si bien es cierto el MINSA recomienda descansar por lo menos 6 horas antes de conducir y realizar una parada cada 2 o 3 horas si el viaje será largo, sin embargo, son varios los conductores de la región que hacen caso omiso a estas recomendaciones y ocasionan en varias oportunidades accidentes (MINSA, 2017).

Como formulación del problema tenemos: ¿Cómo diseñar el sistema de monitoreo, para detectar somnolencia en los choferes de rutas interprovinciales de Chiclayo?

Ante esto justificamos el presente trabajo en el aspecto teórico teniendo en cuenta que las variables están fundamentadas epistemológicamente por las teorías, garantizando tanto la

variable dependiente la cual es detección de somnolencia y la variable dependiente la cual es sistema de monitoreo. En el aspecto práctico la detección de somnolencia fue lograda haciendo uso de sistema basado en visión artificial OpenCv y usando algoritmos matemáticos se analizó tanto el estado de rostro como de los ojos, determinando signos de fatiga en los conductores. En el aspecto social el presente trabajo permitirá la reducción de accidentes de tránsito mediante su implementación y podrá culturizar al conductor promedio sobre sistemas de detección de somnolencia haciendo mayor el uso de estos. En el aspecto metodológico el desarrollo de un sistema de monitoreo permite que la manera de controlar en tiempo real los pequeños momentos de distracción debido a la somnolencia sea más eficiente, debido a la advertencia oportuna por parte de esta a los conductores, evitando de esta manera potenciales accidentes de tránsito.

Es así que el objetivo general de esta investigación es diseñar un sistema de monitoreo, el cual mediante signos de fatiga alertará oportunamente a potenciales conductores y permitirá detectar somnolencia en los choferes de rutas interprovinciales de Chiclayo. Y como objetivos específicos tenemos: Hacer un estudio entre los diversos métodos de detección de rostros, usando diferentes tipos de herramientas matemáticas y clasificadores, estudiar las tecnologías más actuales sobre detección de somnolencia, a fin de poder mencionarlos y sirvan de guía para futuros proyectos de mejora al presentado, diseñar un sistema detector de somnolencia, el cual advierta al conductor y ayude a reducir la probabilidad de que esta sufra un accidente debido a algún síntoma de fatiga, realizar pruebas simuladas para confirmar el correcto funcionamiento del sistema, fomentar el desarrollo de nuevas tecnologías en diversas aplicaciones relacionadas con la somnolencia en la vida diaria.

La hipótesis desarrollada fue que, si desarrollamos el sistema de monitoreo, entonces este permitirá detectar somnolencia en los choferes de rutas interprovinciales de Chiclayo, con la finalidad de tomar las medidas de seguridad necesarias.

II. DISEÑO TEORICO

A nivel internacional, España & Oña (2018) en su investigación realizada en la Universidad Politécnica Salesiana, utilizando el método Haar-Cascade, estudiaron los parámetros y operación del rostro expuesto con respecto a visión artificial, a través de la apertura y cierre del ojo se mostró el nivel de cansancio. Mediante el uso de una mini computadora Rasperry pi3 y una cámara de visión nocturna pudieron verificar el estado de fatiga del conductor. Obtuvieron como resultados del 82.5% de funcionalidad en las pruebas que realizaron a 10 conductores, finalizando su trabajo con una encuesta a estas personas.

Asimismo, Egas (2017) desarrolló una aplicación móvil Android usando un smartphone. Este aplicativo lo desarrollaron usando el lenguaje Java y la librería Google Mobile API visión. Usaron patrones para la detección de somnolencia tales como el asentimiento frontal y lateral de la cabeza, el estado de los ojos y bostezo. En las pruebas que realizaron pudieron demostrar que su sistema tiene buena funcionalidad al estar sometido a luz natural, incluso cuando los conductores llevaran accesorios tales como lentes, gorras o auriculares. Obteniendo una precisión total del sistema del 92.5%.

De igual manera Follonier & Peroni (2018) realizaron este proyecto buscando reducir errores de conducción causados por el cansancio usando herramientas tecnológicas para poder brindar una asistencia de conducción durante el manejo. El sistema en tiempo real realiza un procesamiento digital de imágenes en el monitoreo de ojos y cabeza, apoyándose sobre la herramienta OpenCV para detectar el comportamiento del ojo humano, para poder detectar somnolencia en base al tiempo que estos estuvieron cerrados. El prototipo fue realizado haciendo uso de una placa Raspberry Pi3 y una cámara la cual era la encargada de recibir la información para realizar el procesamiento digital de imágenes, además de contar con una alarma sonora con la cual el conductor iba a interactuar y ser alertado en caso se presente somnolencia. Lograron obtener como resultado de sus pruebas realizadas una predicción del 90 % en el estado del conductor.

En el ámbito nacional, Crespin & Julián (2019) realizaron una investigación con respecto a la construcción de la arquitectura de un sistema de visión artificial basada en Python (OpenCV y Dlib), acertando más del 90%.

También en Huancavelica Mayon & Limaquispe (2019) desarrollaron el trabajo, el cual tuvo como finalidad realizar la detección de somnolencia y distracción mientras se conduce, ya que estas son las causantes de accidentes de tránsito con altas probabilidades de siniestrabilidad. Para

su desarrollo hicieron uso de la oximetría del pulso cardiaco y visión artificial, Usaron EmguCV para detectar el estado en el que se encuentran los ojos, ya sea cerrados o abiertos, la distracción y orientación, además del lenguaje C#, con el uso de redes neuronales entrenaron dichos eventos del ojo, adicionalmente instalaron un zumbador de 12V para una alarma sonora. Luego de realizaron las pruebas del sistema obtuvieron una precisión del 98.48%.

Además, Becerra (2019) realizó su trabajo con el objetivo de poder realizar la identificación de un rostro, para de esta manera poder brindar en tiempo real, mayor seguridad al reconocimiento facial. Para lo cual, mediante las técnicas de un histograma de degradados, así como el reconocimiento facial filtros de Haar, usando un clasificador SVM realizaron la detección del rostro. Realizando la identificación de los 68 puntos faciales de referencia, pudieron evaluar las diferentes características de la cara (sonrisa, parpadeo y boca abierta o cerrada), todos evaluados con la finalidad de ofrecer adicional seguridad al sistema de reconocimiento facial.

Como definiciones conceptuales, según Wedro (2016), la fatiga se define física y mentalmente como falta de energía y presencia de sueño. Es un síntoma que puede usar términos como letargo, cansancio y agotamiento. Algunos de los síntomas que identifican la fatiga según Rondon y Nuñez (2013), en un ser humano son: alteraciones sensoriales, tiempo lento de reacción, parpadeos constantes y prolongados, adormecimiento y ardor de los ojos, concentración menor. Asimismo, hay varias razones por las que las personas se quedan dormidos mientras conducen, las cuales serían: conducir mucho tiempo, consumo de pastillas para dormir y consumo de drogas.

Con respecto a la somnolencia Rosales y De Castro (2010), la definen como una persona puede tender a conciliar el sueño, también conocida como la tendencia a quedarse dormido o capacidad de pasar del estar atento a dormirse. Dicha necesidad está presente y su intensidad se infiere de la velocidad con la que comienza el sueño, las facilidades con la que se interrumpe y el tiempo de sueño.

La clasificación de los métodos para detectar somnolencia se detalla a continuación**:**

Según Flores (2011), los modelos basados en patrones de conducción se crean sobre la base de variables medibles las cuales son: aceleración, velocidad, los cambios, velocidad, presión manual en el timón, frenado y trayectoria vehicular en el carril experimentalmente. Este método tiene la desventaja en base a las prestaciones de un vehículo en específico y de las prácticas durante el manejo de algún conductor, el modelado es dependiente.

También encontramos otro modelo según Fuletra y Bosamiya (2013), que está basado en cambios físicos de las expresiones faciales y oculares; donde para verificar el estado del conductor se puede hacer uso del procesamiento digital de imágenes. Donde el rostro en imagen indica si el chofer está despierto o dormido. Cuando el conductor duerme, sus ojos se cerrarán, lo que le permitirá identificar su somnolencia. La ventaja de este método es ser discreto y posee tres técnicas, las cuales son:

La primera técnica es la de emparejamiento de plantillas, el cual identifica la muestra del chofer (ojos abiertos y ojos cerrados), y la compara con la actual imagen del chofer para determinar la somnolencia.

Según el estudio de Walter Wierwille (1994), nos muestra la técnica del comportamiento ocular, el cual mediante la frecuencia de los párpados y el lapso del cierre de los ojos estos son calculados para determinar los indicadores de somnolencia. Aquí se hace uso de PERCLOS (PERcentaje of the time Eyelids are CLOSed), el cual mide el tiempo (en porcentaje) durante un periodo determinado en el cual un individuo tiene los ojos cerrados.

Por último tenemos la técnica basada en el bostezo que según Fuletra y Bosamiya (2013), el cual está basado en la frecuencia de bostezos que da un conductor. Aquí se comparó el nivel de apertura bucal y un punto de referencia adquirido mediante pruebas experimentales, calcula el tiempo que bosteza el conductor y da una indicación de somnolencia. Ellos también mostraron un modelo basado en la variación de las medidas fisiológicas, el cual se encuentra basado en el análisis del estado de somnolencia mediante la medición de parámetros como son: actividad cerebral, respiración, temperatura externa de la piel, frecuencia cardiaca y variación de esta y la presión arterial haciendo uso de sensores.

Para Castañeda y Hernández (2004), el ojo humano es uno de los sentidos más valiosas porque nos permite recopilar e interactuar con imágenes del entorno. El sistema ocular es fisiológica y ópticamente complejo y atractivo. Es muy importante que el conductor esté en buenas condiciones visuales para poder responder adecuadamente a cualquier evento.

Según Quevedo (2012), el parpadeo se puede definir como un comportamiento de reflejo que puede ocurrir de forma espontánea o involuntaria. Intermitente tiene dos funciones principales: lubrica los ojos para evitar que se sequen y elimina los cuerpos extraños y protege el ojo humano de factores externos como el polvo y la luz. También define a la frecuencia de parpadeo como el número de parpadeos durante un lapso. La actividad y el estado mental de una persona se

encuentran ampliamente relacionadas con la frecuencia del parpadeo. Según el libro de Boeglin (2015), durante una conversación, el interlocutor parpadea un promedio de 22 veces por minuto. Cuando alguien lo lee, esta frecuencia de parpadeo cae a 10 veces por minuto. Pero cuando estás sentado frente a tu computadora, tus ojos solo se cierran 7 veces cada 60 segundos.

Según Wikinson (2013), los siguientes parámetros se determinan en tiempo de parpadeo: normal o despierto (menos de 200 ms) y con signos de fatiga o somnolencia (más de 500 ms).

Según García y Caranqui (2015), la visión por computadora o la visión artificial intenta imitar la capacidad de los seres vivos para ver, comprender y actuar lo que sucede en su alrededor. El número y los tipos de aplicaciones industriales que requieren el uso de tecnologías de visión artificial se han incrementado con el tiempo. Entre sus aplicaciones tenemos: en la agricultura que se utiliza para la ubicación y control automático de malezas y plagas, en biología que permite identificar una característica particular de una planta o de un animal, en geología para la detección de movimiento geológico, en medicina se puede manejar imágenes de ultrasonido, radiografías y mamografías, en robótica un sistema de visión que permite que las máquinas se conduzcan automáticamente y en seguridad donde se puede realizar un seguimiento para detectar la presencia y el movimiento de objetos extraños.

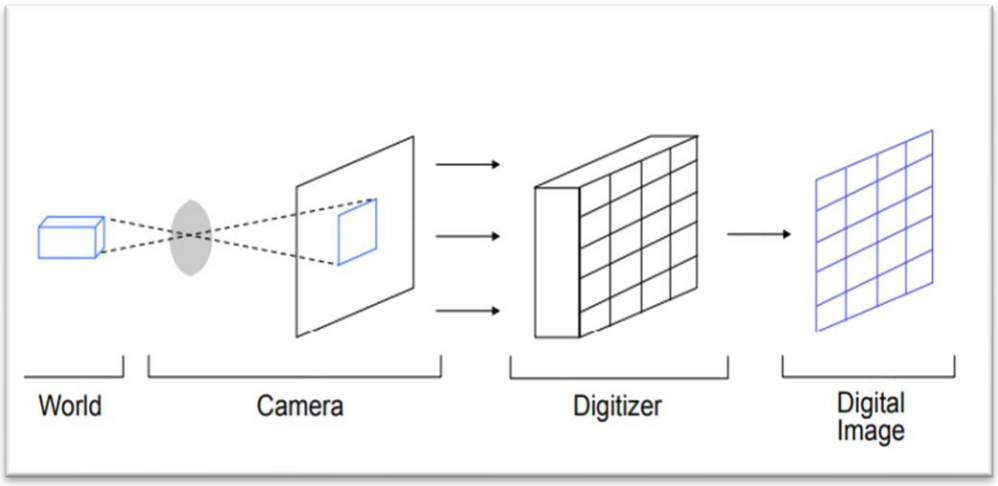
Además, estos definen a las etapas como: la adquisición de imágenes el cual es el proceso de obtener una imagen usando una cámara; pre procesamiento el cual es donde se reduce las características de imagen no deseadas, el ruido, por ejemplo, y resalta las cualidades de importancia que permiten que la fase de segmentación funcione mejor; como segunda etapa tenemos la segmentación que consiste en dividir una imagen en objetos o áreas (grupos de píxeles) a estudiar. Esta etapa es una parte muy importante del éxito o fracaso de su aplicación; la descripción es donde se recibe propiedades relevantes adaptadas para distinguir un tipo de objeto de otro. Estas propiedades pueden ser externas como el perímetro, eje, un rectángulo mínimo que contiene el área, concentración, enfoque, modelo de textura (liso, áspero, regular), color (niveles promedio y de intensidad media, valores máximos y al menos valores de intensidad); el reconocimiento es la etapa que clasifica los objetos en categorías. Los objetos reconocieron que tienen descriptores similares se agrupan automáticamente en la misma clase y por último la etapa de interpretación que trata de imitar la visión humana, y utiliza técnicas cognitivas para el proceso de tomar decisiones. Esta fase depende de cada campo de aplicación.

Según García (2018), “una imagen digital se puede definir como una función bidimensional F (x, y), donde se definen las coordenadas espaciales XY". Dichas imágenes son un arreglo de puntos llamados pixeles. Rojo, verde y azul son los canales por los cuales están conformados una imagen a color, con cada píxel que representa tres valores en un intervalo de 0 a 255.

Y la frecuencia imagen (velocidad de cuadro) se refiere a la cantidad de imágenes por segundo. Esta es definida como la frecuencia medible donde un lector de imágenes muestra distintos fotogramas. Estos están formados por una serie de píxeles distribuidos en la longitud de una estructura de ajuste de textura. La frecuencia de los fotogramas es proporcional a la cantidad de píxeles. La frecuencia con la que se actualizan las imágenes en un entorno digital está entre 15 y 60fps (cuadros por segundo). El intervalo entre 25 y 30 fps es el más común.

Figura *1*

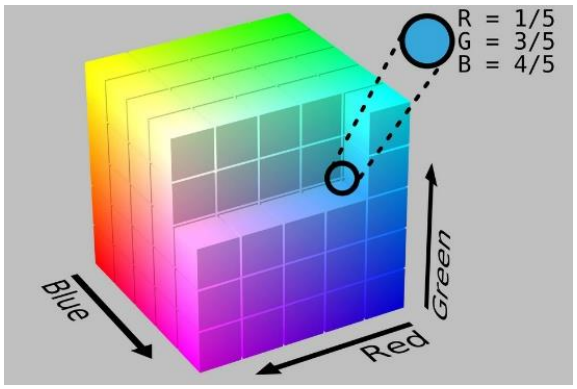
Etapas de formación de una imagen digital



De acuerdo a Vezhnevets (2003), el espacio de color RGB es el más usado en imágenes digitalizadas. No obstante, el espacio RGB se correlaciona ampliamente entre los canales, la mezcla de cromancias y datos de luminancia, los cuales no son una óptima alternativa para el analizar el color.

Figura *2*

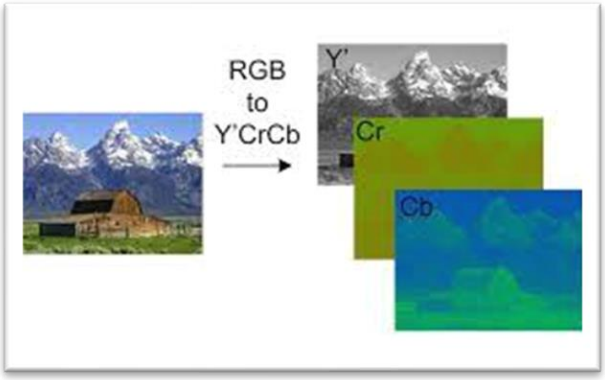
Representación de espacio RGB



Para este estudio también se debe considerar el espacio de color YCBCR que según García (2018), es una señal del tipo RGB codificada no lineal. Este espacio se define como una sumatoria ponderada de valor RGB y 2 valores que difieren del color CB y CR formados por la configuración de componentes azul y rojo. La simplicidad de la separación y transformación explícitamente de brillo con el color hacen que este llamativo aquel lugar para el modelado de la tonalidad de la piel.

Figura 3

Espacio de color YCbCr



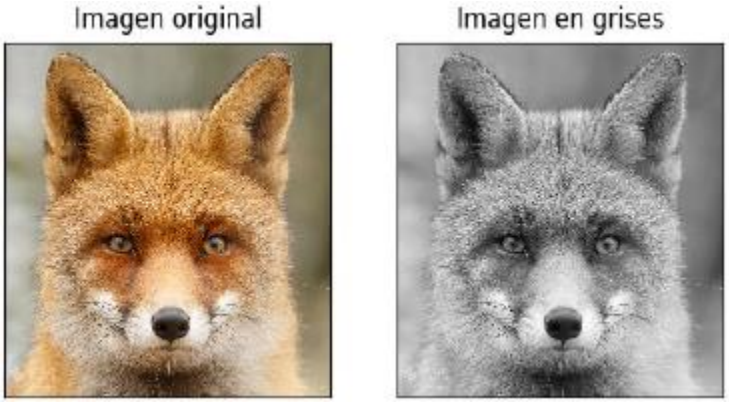
El tratamiento de imágenes nos sirve considerablemente para reconocer las propiedades de los ojos, la cara, los labios, etc. Usado si la información del color de una imagen para solucionar un problema en particular no es de utilidad.

Para obtener una imagen en escala de grises según Vezhnevets (2003), o en casos particulares denominados luminancia de la imagen, se aplicará una ecuación "y" a cada píxel definido, donde

cada componente RGB mostrará la sensibilidad del ojo humano en relación con las frecuencias de colores rojos, verdes y azules.

Figura *4*

Representación de imagen en escala de gris



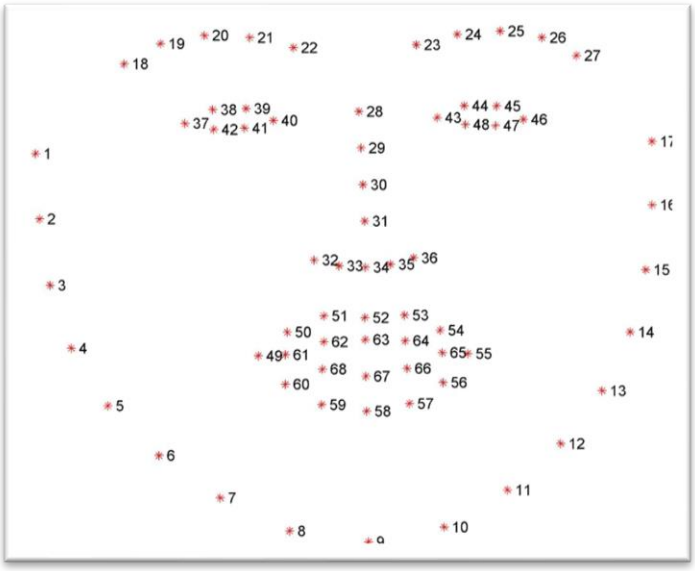
Según García (2018), el Region of Interest (ROI) es comúnmente usado como un término en la visión por computadora en la cual a la parte de una imagen se le aplican varios filtros u operaciones para usarlas en una zona en particular para evitar la cobertura de las zonas no deseadas.

Como técnicas de reconocimiento, el método de reconocimiento de Viola Jones según Gonzalez y Velásquez (2019), es un método para reconocer los rostros, aunque actualmente otorga muchos objetos con gran eficiencia, y una pequeña solicitud de cálculo hace la oportunidad de reconocer objetos en tiempo real. La operación se basa en la extracción de cierta funcionalidad de las imágenes de entrada utilizando conceptos como propiedades del Haar, Adaboost y Clasificador en cascada.

Otra de las técnicas de reconocimiento es el predictor de las marcas faciales, el cual es un clasificador de objetos con el cual es posible ubicar los 68 puntos que son característicos de un rostro. Este, el cual ya ha sido entrenado con el uso de códigos de aprendizaje autónomo que actualmente utiliza 2000 imágenes de rostros.

Figura *5*

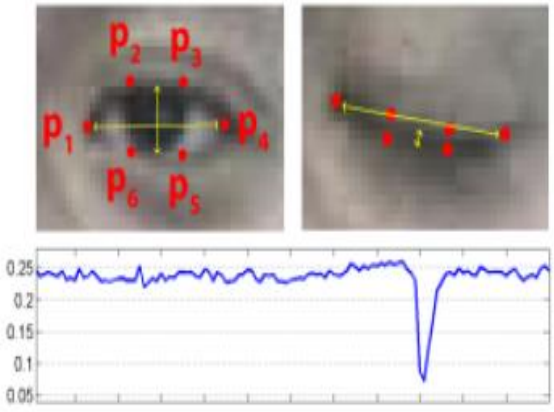
Ubicación de 68 puntos en rostro



Por otro lado tenemos el método EAR (Eye Aspect Ratio) según Soukupova y Cech (2016), es un método en el cual se evalúa el contorno y la apariencia de los ojos de una persona según el predictor de marcas faciales, un ojo posee 6 puntos que forman un contorno ocular, este método suma las distancias verticales entre puntos 2 a 6 y puntos de 3 a 5, y es dividido entre los dos y multiplicados por los puntos horizontales de 1 a 4, mostrada en la Figura 6.

Figura *6*

Representación gráfica de EAR.



$$EAR = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

De acuerdo con los resultados obtenidos por el autor de este método, este concluye que el punto de equilibrio es de 0.25, es decir, el valor igual o mayor que el que se usa para predecir que el ojo está abierto y a un valor más bajo, se puede afirmar que el ojo está cerrado.

Dentro de los software utilizado tenemos OpenCv que según su página web (OpenCv, 2016) se trata de una biblioteca de código abierto de la visión por computadora. Desarrollado por Intel en 1991, y es distribuido bajo la licencia BSD, lo cual permite a los usuarios la edición, modificación y distribución del algoritmo teniendo o no cambios realizados. La biblioteca tiene más de 2.500 algoritmos, los cuales detectan y reconocen caras, identifican objetos, reclasifican, seguimiento de objetos, buscar imágenes con similitudes a una data de imágenes, eliminación de los ojos rojos en las imágenes paperwitting con un rayo, siguen el movimiento ocular, etc. OpenCV posee más de 7.000 usuarios y descarga en más de 7 millones. Esta biblioteca tiene versiones funcionales en Linux, Windows, Mac OS y C y el código C se ha optimizado. A partir de OpenCv 2.3.0 tiene un OpenCV SDK para Android, con aplicaciones desarrolladas para aplicaciones que permite una visión artificial de los dispositivos móviles.

Por su parte el lenguaje de programación Python es de nivel alto y se enfoca principalmente en el aprendizaje y la legibilidad. Python es un lenguaje que puede ser usado en diferentes tipos de sistemas. Python es un software de la categoría libre y está distribuido bajo la licencia "Python Software Foundation License". La gran ventaja de usar este lenguaje es la gran interaccion de su comunidad de desarrolladores, hay una infinidad de librerías para realizar cálculo de matrices y procesamiento de imágenes usando OpenCv. Dentro del lenguaje de Python encontramos librerías que nos serán útiles en el análisis tal es el caso de DLIB que según (*Dlib*, n.d.) es un conjunto de herramientas modernas que contienen algoritmos y herramientas automáticas de aprendizaje para crear un software Complejo para resolver problemas reales. Se utiliza en una variedad de áreas, así como en el área académica, incluidas dispositivos integrados, robótica, entornos computacionales de alta gama y teléfonos móviles. DLIB (sus licencias de código libre) le facilita usarla gratuitamente en todo tipo de aplicación.

III. METODOS Y MATERIALES

3.1. Tipo y Diseño de la investigación

Esta investigación es del tipo aplicada con un enfoque tecnológico

3.2. Definición y Operacionalización de las variables

La variable independiente tomada en consideración es: Sistema de monitoreo.

Definición conceptual:

Para Lauriac (2016), es aquel que permite analizar, recolectar, difundir y tratar la información de un proyecto haciendo uso de un conjunto de herramientas, con el fin de realizar la toma de decisiones e informando lo que sucede en este.

Definición operacional:

El sistema de monitoreo permite realizar en tiempo real un seguimiento al estado de somnolencia en el cual se encuentre un conductor, con la finalidad de tomar las medidas de corrección del caso en cuanto se confirme signos de esta.

La variable independiente tomada en consideración es: Detección de somnolencia

Definición conceptual:

Según Monclús y Ortega (2018), Esta hace uso del sistema para que durante el inicio de la conducción pueda analizar el comportamiento al volante del conductor. De comprobarse que se difiere dicho comportamiento con respecto al iniciado teniendo en cuenta las maniobras realizadas durante la conducción, se habrá detectado somnolencia. La cual puede ser advertida mediante una alerta visual, acústica o sensorial (vibraciones en asiento o al volante).

Definición operacional:

Para detectar la somnolencia el sistema sigue un proceso lógico el cual va desde la detección del rostro, indicación de inclinación de este mediante un predictor de puntos de referencia faciales, si lo estuviera mediante un algoritmo matemático se logra alinearlo, posteriormente en la región comprendida de ambos ojos se analiza el EAR el cual mediante un algoritmo se define si estos están abiertos o cerrados. Este análisis es logrado mediante la inferencia de la red neuronal pre entrenada usada.

3.3. Población y muestra

Para el presente trabajo se consideró como población a las 30 empresas de transporte interprovincial formales que trabajan actualmente en la ciudad de Chiclayo (Fuente: MTC).

La muestra para este trabajo viene siendo tomada por conveniencia, ya que por la situación de pandemia no se pudo conseguir acceso a ninguna de ellas. Esta será considerada teniendo en cuenta un promedio de 30 conductores que en este caso serán nuestros voluntarios sometidos a pruebas desde un área de laboratorio.

3.4. Análisis de los algoritmos usados.

3.4.1. Detección de rostro usando Dlib

Figura 7

Código para importación de librerías e inicialización de cámara.

```
3   from imutils import face_utils
4   import numpy as np
5   import cv2
6   import dlib
7   import imutils
8
9
10
11  detector = dlib.get_frontal_face_detector()
12  color_rectangle = (0, 255, 0)
13  #Iniciamos la camara
14  captura = cv2.VideoCapture(3)
15
16
17  while True:
18
19          #Capturamos una imagen y la convertimos de RGB -> HSV
20          #_, imagen = captura.read()
21          (grabbed, image) = captura.read()
22
23          # Si hemos llegado al final del vídeo salimos
24          if not grabbed:
25              break
26          gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
27
28          rects = detector(gray, 1)
29
30          # Buscamos contorno en la imagen
31          for (i, rect) in enumerate(rects):
```

El programa se inicia llamando a las librerías *imutils, numpy, cv2, dlib* e *imutils.* Hacemos uso del detector por defecto para detección frontal de rostros que brinda *Dlib. color_rectangle = (0, 255, 0)* nos indica que usaremos un rectángulo de color “verde” para hacer un recuadro del rostro detectado. Dicho rostro será de uso mediante *captura = cv2.VideoCapture(3)* el cual viene a ser un objeto de video, el índice (3) es el puerto asignado por el ordenador para la cámara que se está usando. Para poder realizar lectura mediante la cámara se hace uso de *(grabbed, image) = captura.read()* en donde *grabbed* es una variable booleana e *image* viene a ser la imagen a

procesar. Se realizará la lectura del video mediante el condicional *if*. Posteriormente las imágenes adquiridas serán transformadas a escala de grises, puesto que la librería Dlib hace uso de rostros frontales en este formato, esta transformación se realiza usando *gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY),* las ubicaciones de los rostros son retornadas en *rects = detector(gray, 1)*, el índice "1" es el que nos permite redimensionar la imagen, para un uso más práctico se dejó en dicho valor el cual no afecta en el dimensionamiento ya que es por defecto. Para dibujar el limitador del rostro hacemos uso de un ciclo *for*.

Figura 8

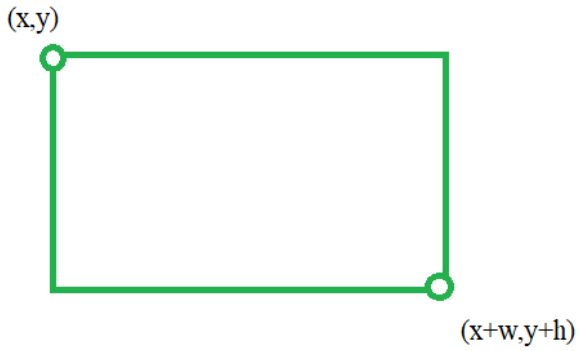
Código para dibujar rectángulo con puntos de referencia

```
                (x, y, w, h) = face_utils.rect_to_bb(rect)
                cv2.rectangle(image, (x, y), (x + w, y + h), color_rectangle, 2)
                #cv2.rectangle(image, (x,y+h- 35), (x+w, y+h), (0, 0, 255), cv2.FILLED)
                #cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
                label = "Face:#" + str(i+1)+" "
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
                xLeftBottom = x
                yLeftBottom = y
                #print(baseLine)


                cv2.rectangle(image, (xLeftBottom-1, yLeftBottom - labelSize[1]),
                                     (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                                     color_rectangle, cv2.FILLED)
                cv2.putText(image, label, (xLeftBottom, yLeftBottom),
                             cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))




        #Mostramos la imagen original


        cv2.imshow('FACE DETECTOR - Frontal Face DLIB', image)
        tecla = cv2.waitKey(1) & 0xFF

        if tecla == 27:
                break

captura.release()
cv2.destroyAllWindows()
```

Para dibujar el rectángulo se toman los puntos de referencia (x, y, w, h). Para poder hacer uso de estos se hace uso de *(x, y, w, h) = face_utils.rect_to_bb(rect)* aquí se realiza la transformación del formato rects a dichas coordenadas. Su distribución se especifica a continuación:

Figura 9

Ejemplo de distribución de coordenadas en rectángulo



Donde w y h son el ancho del recuadro delimitador y el alto del recuadro delimitador respectivamente.

Para darle características al rectángulo realiza mediante *cv2.rectangle(image, (x, y), (x + w, y + h), color_rectangle, 2).*

En donde:

- ***Image:*** Es la imagen que dibujará
- ***(x,y):*** Son las coordenadas de origen
- ***(x+w,y+h):*** Son las coordenadas de los 2 puntos opuestos
- ***color_rectangle*:** Es el color asignado al rectángulo
- *2:* Es el grosor del trazo

Mediante *label* colocamos el texto que se desee. En nuestro caso será "Face:" y se le asigna características usando adicionalmente un tipo de fuente en específico la cual se describe en *labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)* además el texto se está redimensionando a modo que quede encuadrado en la imagen, posteriormente se definen los valores de *(x,y).* Mediante *cv2.rectangle(image, (xLeftBottom-1, yLeftBottom - labelSize[1]), (xLeftBottom + labelSize[0], yLeftBottom + baseLine, color_rectangle, cv2.FILLED)* se dibuja un rectángulo dinámico, el cual se moverá de acuerdop al movimiento que tenga el rostro detectado, mientras que el texto viene definido a continuación: *cv2.putText(image, label, (xLeftBottom, yLeftBottom), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))*

Para mostrar la imagen como tal hacemos uso de *cv2.imshow('FACE DETECTOR - Frontal Face DLIB', image)* obteniendo un resultado como se muestra:

Figura 10

Dibujo de rectángulo en área de rostro

Para poder cerrar la ejecución del código se le asignó *tecla = cv2.waitKey(1) & 0xFF*, el tiempo mínimo de ejecución viene asignado por el índice (1). Si se presiona la tecla "Esc" este finalizará, se le colocó su equivalente ASCII en el código definido con el número 27.

3.4.2. Detección de rostro mediante un entorno Dlib – CNN

Figura 11

Código de programación usando Dlib y CNN

```
from imutils import face_utils
import numpy as np
import cv2
import dlib
import imutils
import time




cnn_face_detector = dlib.cnn_face_detection_model_v1('mmod_human_face_detector.dat')
color_rectangle = (0, 255, 0)

#Iniciamos la camara
captura = cv2.VideoCapture(3)


while True:

        #Capturamos una imagen y la convertimos de RGB -> HSV
        #_, imagen = captura.read()
        (grabbed, image) = captura.read()

        # Si hemos llegado al final del vídeo salimos
        if not grabbed:
            break

        h_image, w_image,channel_image = image.shape
        start = time.time()

        dets = cnn_face_detector(image, 1)

        end = time.time()
```

El programa se inicia llamando a las librerías imutils, numpy, cv2, dlib, time. Hacemos uso del clasificador convolucional cnn_face_detector = dlib.cnn_face_detection_model_v1 (´mmod_human_face_detector.dat´) posteriormente daremos color al rectángulo delimitador del rostro, el cual será de color “verde” e iniciamos la cámara, haciendo uso del puerto (3) asignado por el ordenador.

Ejecutamos un bucle While en él capturamos la imagen y la transformamos a escala de grises mediante (grabbed, image) = captura.read() finalizando el video si no hay imágenes por procesar.

Para dar características a las imágenes se hace uso de h_image, w_image,channel_image = image.shape.

Donde:

- ***h_image****:* Es la cantidad de filas
- ***w_image:*** Es la cantidad de columnas
- ***channel_image:*** Es la cantidad de canales de la imagen

Para empezar a detectar el tiempo el tiempo de procesamiento de la imagen se usa *start = time.time()*. Mediante *dets = cnn_face_detector(image, 1)* se entrega la imagen a color, ya que la CNN fue entrenada de ese modo (inferencia de la detección).

Para calcular el tiempo que se demoró para procesar la imagen se usa *end = time.time()*

Figura 12

Código para dibujar rectángulo en rostro

```
        print("Number of faces detected: {}".format(len(dets)))
        if len(dets) >0:

            cv2.rectangle(image,(5,h_image-40),(240,h_image-10),(255,255,255),-1)
            cv2.putText(image, "Execute Time (s): {:.5f}".format(end - start), (10,h_image-20),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))

        for i, d in enumerate(dets):
            print("Detection {}: Left: {} Top: {} Right: {} Bottom: {} Confidence: {}".format(
                i, d.rect.left(), d.rect.top(), d.rect.right(), d.rect.bottom(), d.confidence))
            left =  d.rect.left()
            top = d.rect.top()
            right = d.rect.right()
            bottom = d.rect.bottom()
            label = "Face:#" + str(i+1)+" "
            labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
            xLeftBottom = left
            yLeftBottom = top



            cv2.rectangle(image,(left,top),(right,bottom),(0,255,0),2)
            cv2.rectangle(image, (xLeftBottom-1, yLeftBottom - labelSize[1]),
                                  (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                                  color_rectangle, cv2.FILLED)
            cv2.putText(image, label, (xLeftBottom, yLeftBottom),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))


        #Mostramos la imagen original con la marca del centro y


        cv2.imshow('FAC DETECTOR - CNN', image)
        tecla = cv2.waitKey(1) & 0xFF
```

Mediante *if len(dets) >0:* se condiciona a sólo dibujar el rectángulo sólo si el número de detecciones es mayor a 0.

Posteriormente se procede a darle características al rectángulo pequeño ubicado al lado inferior izquierdo debajo del rectángulo del rostro mediante *cv2.rectangle(image,(5,h_image-40),(240,h_image-10),(255,255,255),-1)* el índice "-1" quiere decir que el rectángulo esta totalmente relleno o FILLED.

Para dar formato al texto que estará dentro del rectángulo relleno usamos:

- *cv2.putText(image, "Execute Time (s): {:.5f}".format(end - start), (10,h_image-20),*
- ***cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0)),***aquí se mostrará el tiempo de ejecución y el valor tendrá 5 cifras decimales.

Hacemos uso de un ciclo *for i, d in enumerate(dets):*, el cual recorrerá todas las detecciones, en el parámetro "d" están guardadas las coordenadas x e y que definen los puntos delimitantes del rostro.

Mediante *print("Detection {}: Left: {} Top: {} Right: {} Bottom: {} Confidence: {}".format(i, d.rect.left(), d.rect.top(), d.rect.right(), d.rect.bottom(), d.confidence))* se imprimirán las posiciones izquierda, arriba, derecha y abajo, además de la confidencia el cual es la precisión de detección con la que se está realizando.

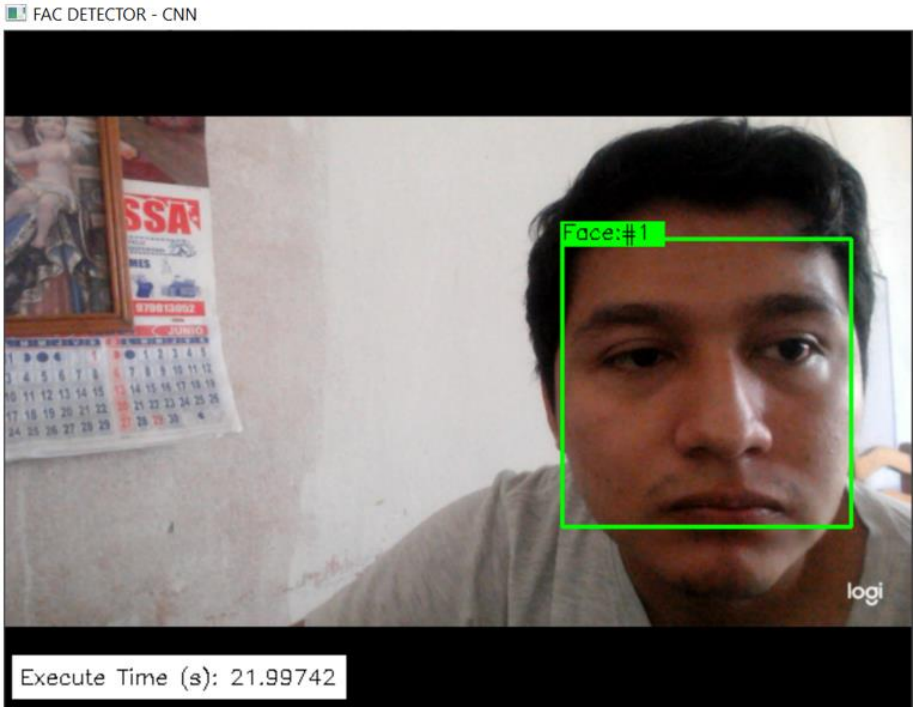
Para poder dibujar el rectángulo del rostro hacemos uso de *cv2.rectangle(image,(left,top),(right,bottom),(0,255,0),2)* y para poder dibujar un rectángulo más pequeño en la parte superior izquierda del rectángulo del rostro hacemos uso de *cv2.rectangle(image, (xLeftBottom-1, yLeftBottom - labelSize[1]), (xLeftBottom + labelSize[0], yLeftBottom + baseLine), color_rectangle, cv2.FILLED)* el cual será un rectángulo relleno.

Para dar formato al texto que estará dentro del rectángulo relleno hacemos uso de *cv2.putText(image, label, (xLeftBottom, yLeftBottom), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))*

Al ejecutar el código se aprecia de la siguiente manera:

Figura 13

Detección de rostro usando CNN



Se nombra al ejecutable mediante *cv2.imshow('FAC DETECTOR - CNN', image)* y finalmente se le asigna la tecla “Esc” o “27” en ASCII para cerrar el ejecutable.Se pudo notar bastantes retrasos en la ejecución de este código, dando la impresión de movimientos desfasados del rostro.

3.4.3. Detección de rostros mediante ResNet + Framework Caffe

Figura 14

Código de programación usando ResNet y Framework Caffe

```
import numpy as np
import cv2
import time
import imutils


color = (0,255,0)
color1 = (0,255,0)
#captura = cv2.VideoCapture("video.mp4")
captura = cv2.VideoCapture(3)
time.sleep(2.0)


net = cv2.dnn.readNetFromCaffe('deploy.prototxt.txt', 'res10_300x300_ssd_iter_140000.caffemodel')
while True:

        #Capturamos frame a frame
        (grabbed, image) = captura.read()

        # Si hemos llegado al final del vídeo salimos
        if not grabbed:
            break
        #frame = imutils.resize(image, width=400)
        frame = cv2.resize(image,(300,300))
        # grab the frame dimensions and convert it to a blob
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,(300, 300), (104.0, 177.0, 123.0))

        # pass the blob through the network and obtain the detections and
        # predictions
        net.setInput(blob)
        detections = net.forward()
        count = 0
```

El programa se inicia llamando a las librerías *numpy, cv2, time, imutils.* Iniciamos con el objeto de video haciendo uso del puerto (3) definido por el ordenador.

Hacemos uso del módulo de OpenCV para leer una red neuronal del Framework Caffe mediante *net = cv2.dnn.readNetFromCaffe(´deploy.prototxt.txt´,´res10_300x300_ssd_iter_140000.caffemo model´)*

Donde:

- *'deploy.prototxt.txt':* Es el modelo de la red neuronal
- *'res10_300x300_ssd_iter_140000.caffemodel':* son los pesos de la red neuronal

Hacemos uso de un bucle *While* en el cual se hace la captura de imágenes, finalizando el proceso si es que se llegó al final del video.

Se realiza un redimensionamiento de las imágenes a un formato de 300x300p, puesto que fue con ese formato en el cual fue entrenada la red neuronal mediante frame = *cv2.resize(image,(300,300))*.

Previo a realizar la imagen para su inferencia en la red neuronal se crea una versión nueva de la imagen mediante *blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,(300, 300), (104.0, 177.0, 123.0))*

Para definir la entrada a la red neuronal se usa *net.setInput(blob)*

Definiendo lo que se pudo detectar se realiza la inferencia mediante *detections = net.forward()*

Figura 15

Código para dibujar rectángulo si la detección es mayor al 50%

```
        for i in range(0, detections.shape[2]):

            confidence = detections[0, 0, i, 2]

            if confidence < 0.5:
                continue
            count += 1

            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])

            (startX, startY, endX, endY) = box.astype("int")
            heightFactor = image.shape[0]/300.0
            widthFactor = image.shape[1]/300.0

            startX = int(widthFactor * startX)
            startY = int(heightFactor * startY)
            endX   = int(widthFactor * endX)
            endY   = int(heightFactor * endY)


            text = "{:.2f}%".format(confidence * 100) #+ ", Count " + str(count)
            y = startY - 10 if startY - 10 > 10 else startY + 10

            #EFECTO DE TRASLUCIDO ------------------------------
            image_copy = image.copy()
            roi_1 = image_copy[startY:endY,startX:endX]
            cv2.rectangle(image, (startX, startY), (endX, endY),(184, 169, 165), -1,1)
            roi = image[startY:endY,startX:endX]
            dst = cv2.addWeighted(roi,0.2,roi_1,0.8,0)
            image[startY:endY,startX:endX] = dst
            #cv2.imshow('ROI', roi)
            cv2.putText(image, text, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 1)
            cv2.putText(image, "confidence", (startX, y-15), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 1)
```

Mediante el ciclo *for i in range(0, detections.shape[2]):* realizamos todas las interacciones sobre los rostros, para posteriormente usar el condicional *if confidence < 0.5:* en el cual sólo se dibujará el rectángulo si es que la confidencia es mayor al 50%.

Se calculará las coordenadas (x,y) del cuadro delimitador del objeto mediante *box = detections[0, 0, i, 3:7] * np.array([w, h, w, h]).*Posteriormente se escalará la detección de objetos al frame mediante

- **heightFactor = image.shape[0]/300.0**
- **widthFactor = image.shape[1]/300.0**
- **startX = int(widthFactor * startX)**

- **startY = int(heightFactor * startY)**
- **endX = int(widthFactor * endX)**
- **endY = int(heightFactor * endY)**

Luego se tendrá que dibujar el cuadro delimitador del rostro junto con el asociado, se empezará con el efecto traslúcido, el cual consiste en superponer en las mismas posiciones los recuadros con diferente tonalidad de color haciendo un efecto de "mezcla" de estos.

Figura 16

Delimitación al momento de dibujar rectángulo

```
            # MARCO-----------------------------------------------

            ancho = int((endX - startX)/6)
            alto = int((endY - startY)/6)
            #--lineas horizontales
            g = 2
            cv2.line(image,(startX,startY),(startX+ancho,startY),color1,g) # arriba
            cv2.line(image,(endX-ancho,startY),(endX,startY),color1,g) # arriba
            cv2.line(image,(startX,endY),(startX+ancho,endY),color1,g) # abajo
            cv2.line(image,(endX-ancho,endY),(endX,endY),color1,g) # abajo
            #--lineas verticales
            cv2.line(image,(startX,startY),(startX,startY+alto),color1,g) # arriba
            cv2.line(image,(endX,startY),(endX,startY+alto),color1,g) # arriba
            cv2.line(image,(startX,endY),(startX,endY-alto),color1,g) # abajo
            cv2.line(image,(endX,endY),(endX,endY-alto),color1,g) # abajo

        #Mostramos imagen
        cv2.imshow('Video', image)
        #Capturamos teclado
        tecla = cv2.waitKey(1) & 0xFF
        #Salimos si la tecla presionada es ESC
        if tecla == 27:
                break
#Liberamos objeto
captura.release()
#Destruimos ventanas
cv2.destroyAllWindows()
```

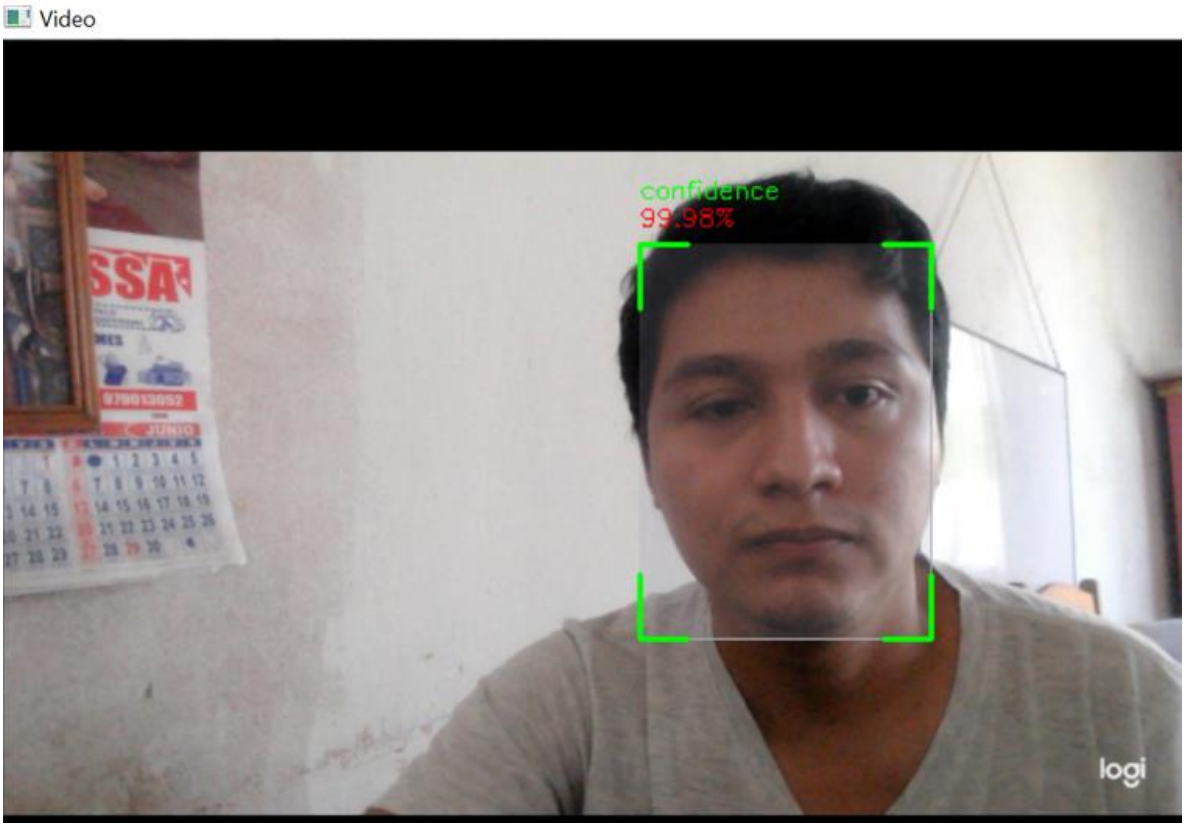
Para realizar el marco lo primero será delimitar tanto el ancho como el alto del recuadro, para lo cual haremos uso de las variables *endX, startX, endY, startY.*

Posteriormente se dibujan las líneas horizontales superior tanto izquierdo y derecho, lado inferior izquierdo y derecho así como sus correspondientes verticales superior izquierda y derecha, así como inferior izquierda y derecha.

Mostramos la imagen resultante mediante *cv2.imshow('Video', image),* obteniendo como resultado:

Figura 17

Ejecución de código



Para posteriormente tener la opción de cerrar la ejecución del programa mediante la tecla “Esc” o su equivalente “27” en ASCII.

Los resultados en pruebas realizados en este código fueron muy buenos, obteniendo un tiempo de respuesta bastante optimizado.

3.4.4. Reconocimiento facial usando 68 puntos de referencia (Face Landmarks)

Figura 18

Código de programación usando el predictor de 68 puntos de referencia

```
from imutils import face_utils
import numpy as np
import cv2
import dlib
import imutils

predictor_path= "shape_predictor_68_face_landmarks.dat"

JAWLINE_POINTS = list(range(0, 17))
RIGHT_EYEBROW_POINTS = list(range(17, 22))
LEFT_EYEBROW_POINTS = list(range(22, 27))
NOSE_POINTS = list(range(27, 36))
RIGHT_EYE_POINTS = list(range(36, 42))
LEFT_EYE_POINTS = list(range(42, 48))
MOUTH_OUTLINE_POINTS = list(range(48, 61))
MOUTH_INNER_POINTS = list(range(61, 68))


 # create the landmark predictor
predictor = dlib.shape_predictor(predictor_path)
detector = dlib.get_frontal_face_detector()


#Iniciamos la camara
captura = cv2.VideoCapture(3)


while True:

        #Capturamos una imagen y la convertimos de RGB -> HSV
        #_, imagen = captura.read()
        (grabbed, image) = captura.read()

        # Si hemos llegado al final del vídeo salimos
        if not grabbed:
```

El programa se inicia llamando a las librerías *numpy, cv2, dlib, imutils.* Llamamos al predictor de interés *predictor_path= "shape_predictor_68_face_landmarks.dat"*, lo siguiente será definir los puntos faciales a tomar en cuenta, los cuales se detallan como:

- ***JAWLINE_POINTS*** = *list(range(0, 17)):* Son los puntos de la barbilla
- ***RIGHT_EYEBROW_POINTS*** = *list(range(17, 22)):* Son los puntos de la ceja derecha
- ***LEFT_EYEBROW_POINTS*** = *list(range(22, 27)):* Son los puntos de la ceja izquierda
- **NOSE_POINTS** = list(range(27, 36)): Son los puntos de la nariz
- **RIGHT_EYE_POINTS** = list(range(36, 42)): Son los puntos del ojo derecho
- **LEFT_EYE_POINTS** = list(range(42, 48)): Son los puntos del ojo izquierdo

- **MOUTH_OUTLINE_POINTS =** list(range(48, 61)): Son los puntos del labio superior
- **MOUTH_INNER_POINTS =** list(range(61, 68)): Son los puntos del labio inferior

Se crea el predictor mediante predictor = dlib.shape_predictor(predictor_path)

Lo primero que se realiza es la detección del rostro y este es procesada como imagen al predictor, este proceso está definido por detector = dlib.get_frontal_face_detector()

Se inicia la cámara haciendo uso del puerto (3) definido por el ordenador, posteriormente se inicia un bucle While, en el cual capturamos una imagen y se transforma a escala de grises mediante (grabbed, image) = captura.read(), si se llega al final del video, salimos.

Figura 19

Código para definir el área a detectar

```
        if not grabbed:
            break
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        rects = detector(gray, 1)

        # Buscamos contorno en la imagen
        for (i, rect) in enumerate(rects):

              shape = predictor(gray, rect)
              shape = face_utils.shape_to_np(shape)

              (x, y, w, h) = face_utils.rect_to_bb(rect)
              cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

              cv2.putText(image, "Rostro #{}".format(i + 1), (x - 10, y - 10),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)

              for (x, y) in shape:
                      cv2.circle(image, (x, y), 2, (0, 0, 255), -1)

        cv2.imshow('TESIS_DETECCION_SOMNOLENCIA', image)
        tecla = cv2.waitKey(1) & 0xFF

        if tecla == 27:
                break

captura.release()
cv2.destroyAllWindows()
```

Mediante *for (i, rect) in enumerate(rects):* recorremos todas las detecciones

Se hace un barrido de todas las regiones del rostro usando *shape = predictor(gray, rect)*

Posteriormente todas las coordenadas de todos los puntos de referencia retornan mediante *shape = face_utils.shape_to_np(shape)*

Debido a que se necesitan las coordenadas en valores (x,y,w,h) se realiza la transformación del formato de Dlib usando *(x, y, w, h) = face_utils.rect_to_bb(rect)*

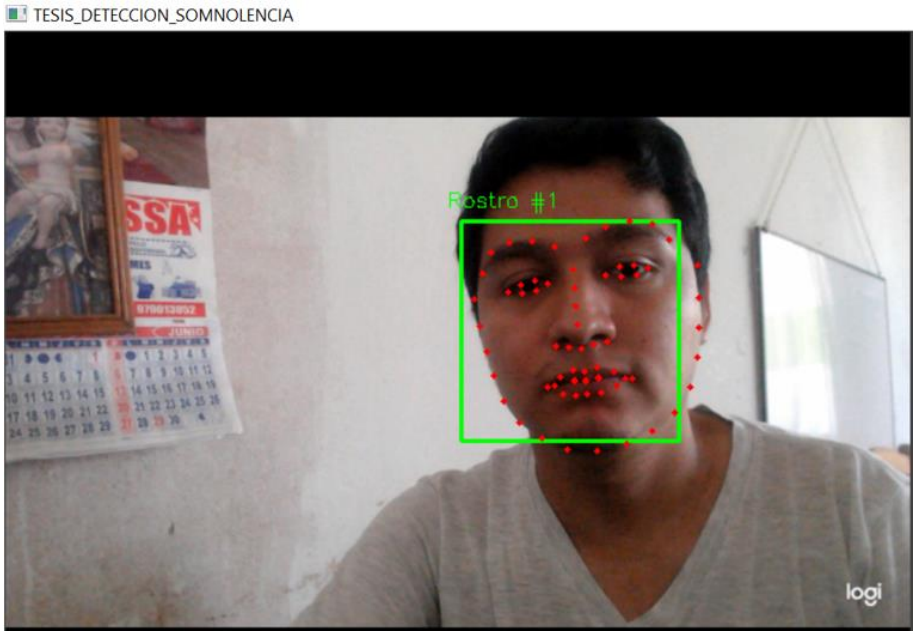
Dibujamos el rectángulo mediante *cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)*

Para escribir el texto “Rostro#” hacemos uso de *cv2.putText(image, "Rostro #{}".format(i + 1), (x - 10, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)*

Haciendo uso de un ciclo *for (x, y) in shape:* se recorre las coordenadas (x,y) para los puntos de referencia faciales y se dibujan en la imagen mediante *cv2.circle(image, (x, y), 2, (0, 0, 255), -1)* De esta manera se dibujarán círculos de radio 2 color rojo todos rellenos totalmente. (x,y) es el centro de cada círculo.Para mostrar la imagen con los puntos faciales en ella se usa *cv2.imshow('TESIS_DETECCION_SOMNOLENCIA', image)* obteniendo el siguiente resultado:

Figura 20

Ejecución de código mostrando 69 puntos faciales



Para posteriormente tener la opción de cerrar la ejecución del programa mediante la tecla “Esc” o su equivalente “27” en ASCII.

3.4.5. Detección de mayor rostro

Figura 21

Código de programación para la detección de una mayor área

```
from imutils import face_utils
import numpy as np
import cv2
import dlib
import imutils

image = cv2.imread("1.jpg")
detector = dlib.get_frontal_face_detector()
predictor_path= "shape_predictor_68_face_landmarks.dat"
predictor = dlib.shape_predictor(predictor_path)

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
rects = detector(gray, 1)
print(rects[0])

area = []
for (j,rect) in enumerate(rects):
    (x, y, w, h) = face_utils.rect_to_bb(rect)
    cv2.rectangle(image, (x, y), (x + w, y + h), (0,0,255), 2)
    area_faces = w*h
    area.append(area_faces)

print("AREAS",area)
MAX_face = max(area)
```

El programa se inicia llamando a las librerías *numpy, cv2, dlib, imutils.* Cargamos la imagen que se analizará mediante *image = cv2.imread("1.jpg"),* dicha imagen será detectada por *detector = dlib.get_frontal_face_detector()* para poder pasarla por el predictor *predictor_path= "shape_predictor_68_face_landmarks.dat",* este será creado por *predictor = dlib.shape_predictor(predictor_path),* la imagen la transformamos a escala de grises mediante *gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)* y se pasa al formato de *Dlib* mediante *rects = detector(gray, 1).*

Se crea una lista donde se guardarán las áreas de los rostros usando *area = [],* posteriormente mediante un ciclo *for (j,rect) in enumerate(rects):* se estará haciendo un barrido de las detecciones, al necesitar las coordenadas (x,y,w,h) se hará la transformación a este formato desde el formato Dlib mediante *(x, y, w, h) = face_utils.rect_to_bb(rect),* luego se dibujará el rectángulo delimitador usando *cv2.rectangle(image, (x, y), (x + w, y + h), (0,0,255), 2)* siendo este un rectángulo de contorno color rojo. Se define el área de los rostros como *area_faces = w*h* el cual sería el ancho * el alto del rectángulo.

Haciendo uso de *area.append(area_faces)* a través del *append* es que se le adicionará a la lista *"area"* el área que se calculó en *"area_faces"*. El máximo rostro se vendrá calculando mediante *MAX_face = max(area).*

Figura 22

Código de programación para que se pueda mostrar una mayor área

```
indice_max_face = area.index(MAX_face)
print("Indice de Maxima area: ",indice_max_face)
rect_max_face = rects[indice_max_face]
shape = predictor(gray, rect_max_face)

shape = face_utils.shape_to_np(shape)
(x, y, w, h) = face_utils.rect_to_bb(rect)
cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 3)

cv2.imshow("MAYOR ROSTRO", image)


cv2.waitKey(0)
cv2.destroyAllWindows()
```

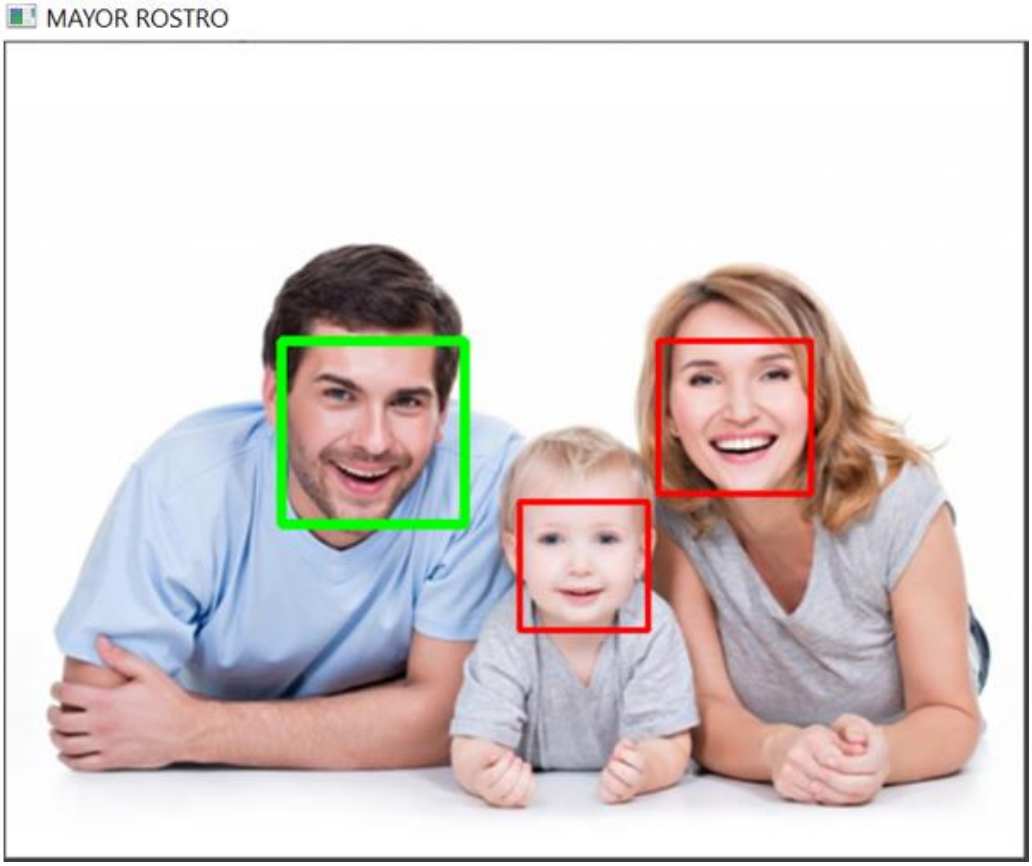
Mediante *indice_max_face = area.index(MAX_face)* nos indica cuál es el índice de la lista haciendo uso del valor obtenido en *MAX_face,* se imprime cuál es el índice del 0 al 9 con mayor área usando *print("Indice de Maxima area: ",indice_max_face).*

Guardamos las coordenadas del rectángulo delimitador en el formato Dlib frontal fase correspondiente a *indice_max_face* mediante *rect_max_face = rects[indice_max_face],* posteriormente se calcula los puntos de referencia facial del máximo rostro usando *shape = predictor(gray, rect_max_face),* estos puntos de referencia facial serán de utilidad para retornar las coordenadas mediante *shape = face_utils.shape_to_np(shape)* ahora estas coordenadas en formato Dlib frontal face se transformarán en formato (x,y,w,h) usando *(x, y, w, h) = face_utils.rect_to_bb(rect).*

Se culmina dibujando el rectángulo del rostro de mayor área en *cv2.rectangle(image, (x, y), (x + w, y + h), (0,255,0), 3)* el cual viene siendo un rectángulo de contorno color verde y grosor ligeramente mayor al de los demás rostros que hayan sido detectados. Pudiéndose apreciar en la siguiente imagen en la cual fue probado este código:

Figura 23

Ejemplo de detección de mayor área



3.4.6. Alineación de un rostro utilizando puntos de referencia facial

Dado un conjunto de puntos de referencia faciales (las coordenadas de entrada) nuestro objetivo es deformar y transformar la imagen en un espacio de coordenadas de salida.

En este espacio de coordenadas de salida, todas las caras deben:

- Estar centrada en la imagen.
- Girarse de tal manera que los ojos se encuentran en una línea horizontal (es decir, la cara se gira de tal manera que los ojos se encuentran a lo largo de las mismas coordenadas y).
- Se escala de forma que el tamaño de las caras sea aproximadamente idéntico.

El proceso de alineación de rostro considera los siguientes pasos:

1. Determinación de ángulo de inclinación.
2. Definir matriz de transformación 2D a través de cv2.getRotationMatrix2D().
3. Aplicar transformación afín utilizando la función cv2.warpAffine().

Figura 24

Código de programación para proceso de puntos faciales

```
from scipy.spatial import distance as dist
from imutils.video import FileVideoStream
from imutils.video import VideoStream
from imutils import face_utils
import numpy as np
import cv2
import dlib
import imutils

predictor_path= "shape_predictor_68_face_landmarks.dat"

JAWLINE_POINTS = list(range(0, 17))
RIGHT_EYEBROW_POINTS = list(range(17, 22))
LEFT_EYEBROW_POINTS = list(range(22, 27))
NOSE_POINTS = list(range(27, 36))
RIGHT_EYE_POINTS = list(range(36, 42))
LEFT_EYE_POINTS = list(range(42, 48))
MOUTH_OUTLINE_POINTS = list(range(48, 61))
MOUTH_INNER_POINTS = list(range(61, 68))

(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

El programa se inicia llamando a las librerías *numpy, cv2, dlib, imutils.* Se carga el predictor *predictor_path= "shape_predictor_68_face_landmarks.dat"* y se identifican los 68 puntos de referencias faciales, los cuales se detallan a continuación:

- **JAWLINE_POINTS =** list(range(0, 17)): Son los puntos de la barbilla
- **RIGHT_EYEBROW_POINTS =** list(range(17, 22)): Son los puntos de la ceja derecha
- **LEFT_EYEBROW_POINTS =** list(range(22, 27)): Son los puntos de la ceja izquierda
- **NOSE_POINTS =** list(range(27, 36)): Son los puntos de la nariz
- **RIGHT_EYE_POINTS =** list(range(36, 42)): Son los puntos del ojo derecho
- **LEFT_EYE_POINTS =** list(range(42, 48)): Son los puntos del ojo izquierdo
- **MOUTH_OUTLINE_POINTS =** list(range(48, 61)): Son los puntos del labio superior
- **MOUTH_INNER_POINTS =** list(range(61, 68)): Son los puntos del labio inferior

Obtendremos los índices de inicio y fin tanto para el ojo izquierdo y derecho en:

- ***(lStart, lEnd)** = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]*
- ***(rStart, rEnd)** = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]*

Figura 25

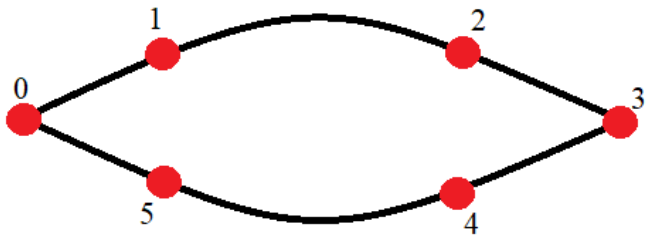
Código de programación para definir aspecto de radio del ojo EAR

```
def eye_aspect_ratio(eye):
    # calcula la distancia euclidea entre coordenadas (x,y) puntos verticales
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])

    # calcula la distancia euclidea entre coordenadas (x,y) puntos horizontales
    C = dist.euclidean(eye[0], eye[3])

    # calcula la relacion de aspecto
    ear = (A + B) / (2.0 * C)

    return ear


#-----------------------------------------------------------------------

predictor = dlib.shape_predictor(predictor_path)
detector = dlib.get_frontal_face_detector()

#----------------------  Parametros de alineacion ----------------------
#desiredLeftEye=(0.35, 0.35)
desiredLeftEye=(0.20, 0.20)
desiredFaceWidth= 256
desiredFaceHeight= 256
#-----------------------------------------------------------------------

#Iniciamos la camara
captura = cv2.VideoCapture(3)
```

Para establecer el aspecto de radio del ojo (EAR) hacemos uso de *def eye_aspect_ratio(eye):*

Figura 26

Distribución de los puntos considerados en los ojos



Para calcular la distancia euclideana entre coordenadas (x,y) puntos verticales usamos:

- *A = dist.euclidean(eye[1], eye[5])*
- *B = dist.euclidean(eye[2], eye[4])*

Para calcular la distancia euclideana entre coordenadas (x,y) puntos horizontales usamos:

- *C = dist.euclidean(eye[0], eye[3])*

La relación de aspecto (EAR) se calcula mediante:

- *ear = (A + B) / (2.0 * C)*

Cargamos tanto el predictor como el detector mediante:

- *predictor = dlib.shape_predictor(predictor_path)*
- *detector = dlib.get_frontal_face_detector()*

Por ensayo y error se definió el valor de 0.20, esto para que no esté muy cerca el ojo, y está dado como

- *desiredLeftEye=(0.20, 0.20).*

Para nuestra imagen de salida se consideró una imagen de 256x256, siendo *desiredFaceWidth= 256* el ancho de la imagen de salida y *desiredFaceHeight= 256* el alto de la imagen de salida.

Figura 27

Detección de rostros en imagen a escala de grises

```
while True:


        #_, imagen = captura.read()
        (grabbed, image) = captura.read()
        image_copy = image.copy()

        # Si hemos llegado al final del vídeo salimos
        if not grabbed:
            break
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        rects = detector(gray, 1)

        print('numero de rostros', len(rects))


        if len(rects) == 1:
            for (i, rect) in enumerate(rects):

                # Determinacion de puntos de referencia facial

                shape = predictor(gray, rect)

                # Conversion de coordenadas (x,y) de facial landmarks a arreglo tipo numpy
                shape = face_utils.shape_to_np(shape)
                (x, y, w, h) = face_utils.rect_to_bb(rect)

                faceOrig = image[y:y + h, x:x + w]

                faceOrig = cv2.resize(faceOrig,(256,256), interpolation = cv2.INTER_LINEAR)
                cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Se realiza la lectura de la imagen y si se ha llegado al final del video salimos.

Pasamos la imagen a grises mediante *gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)* y se realiza la detección de rostros en imagen a escala de grises usando *rects = detector(gray, 1).*

Se condiciona de tal manera que siga con el ciclo *for* siempre y cuando haya detectado 1 solo rostro.

Se determinan los punto de referencia facial mediante *shape = predictor(gray, rect)*

Se realiza la conversión de coordenadas (x,y) de facial landmarks a un arreglo Numpy (x,y,w,h) usando:

- shape = face_utils.shape_to_np(shape)
- (x, y, w, h) = face_utils.rect_to_bb(rect)

Posteriormente se redimensiona la imagen inicial a un formato de 256x256 y se dibuja el rectángulo delimitador haciendo uso de:

- faceOrig = image[y:y + h, x:x + w]
- faceOrig = cv2.resize(faceOrig,(256,256), interpolation = cv2.INTER_LINEAR)
- cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

Figura 28

Código de programación para definir centro de cada ojo

```
88
89              # ----- Calculo de angulo entre ojos -------------------------
90
91              leftEyePts = shape[lStart:lEnd]
92              rightEyePts = shape[rStart:rEnd]
93              # Determinacion de centro de masa de cada ojo
94              leftEyeCenter = leftEyePts.mean(axis=0).astype("int")
95              rightEyeCenter = rightEyePts.mean(axis=0).astype("int")
96
97              # Determina el angulo entre los centroides
98              dY = rightEyeCenter[1] - leftEyeCenter[1]
99              dX = rightEyeCenter[0] - leftEyeCenter[0]
100             angle = np.degrees(np.arctan2(dY, dX)) - 180
101             print("Angle",angle)
102             print("leftEYEcenter",leftEyeCenter)
103             print("rightEYEcenter",rightEyeCenter)
104
105             Rx = rightEyeCenter[0]
106             Ry = rightEyeCenter[1]
107             Lx = leftEyeCenter[0]
108             Ly = leftEyeCenter[1]
109
110             cv2.line(image, (Rx,Ry),(Lx,Ly),(0, 255, 255), 3, cv2.LINE_AA)
111             cv2.circle(image, (Rx, Ry), 3, (255, 0,0), -1)
112             cv2.circle(image, (Lx, Ly), 3, (0, 0, 255), -1)
```

Para determinar el ángulo de inclinación calculamos el centroide, también conocido como el centro de masa, de cada ojo promediando todos los puntos (x, y) de cada ojo, respectivamente.

- leftEyePts = shape[lStart:lEnd]
- rightEyePts = shape[rStart:rEnd]

- leftEyeCenter = leftEyePts.mean(axis=0).astype("int")
- rightEyeCenter = rightEyePts.mean(axis=0).astype("int")

Dados los centroides, podemos calcular las diferencias entre coordenadas (x, y), es decir, el delta en la dirección de cada una de las coordenadas:

- *dY = rightEyeCenter[1] - leftEyeCenter[1]*
- *dX = rightEyeCenter[0] - leftEyeCenter[0]*

Finalmente, calculamos el ángulo de rotación de la cara utilizando la función *arctan2* con argumentos *dY* y *dX*, seguido de la conversión a grados mientras se resta 180.

- *angle = np.degrees(np.arctan2(dY, dX)) – 180*

Figura 29

Argumento para calcular ángulo de rotación

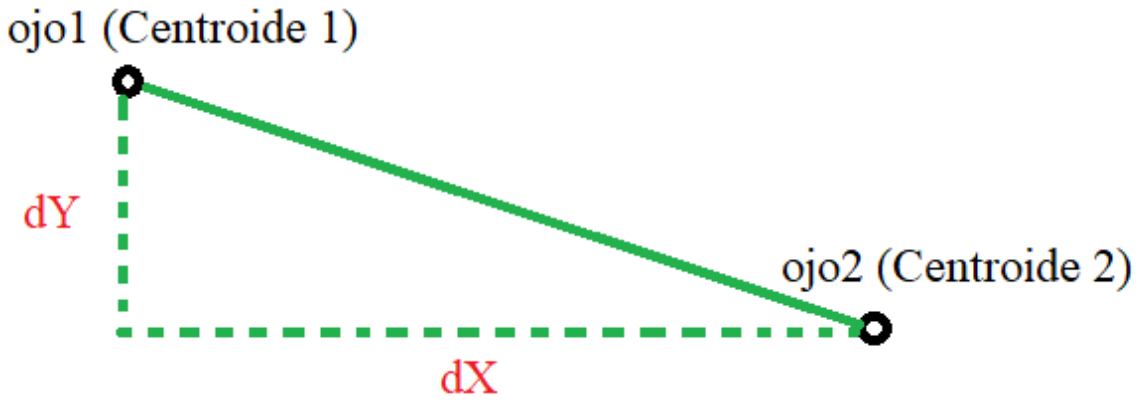


Figura 30

Código de programación para hallar coordenadas en ojos

```
            desiredRightEyeX = 1.0 - desiredLeftEye[0] # desiredLeftEye=(0.20, 0.20)
            dist_orig = np.sqrt((dX ** 2) + (dY ** 2))
            desiredDist = (desiredRightEyeX - desiredLeftEye[0])
            print("DesiredDist",desiredDist) # 0.6
            desiredDist =  desiredDist*desiredFaceWidth #0.6 * 256
            scale = desiredDist / dist_orig
            print("Dist_eyes",dist_orig)
            print("DesiredDist",desiredDist)
            print("Scale:",scale)

            #-----  Determinar matriz de rotacion M ----------------------------------------
            # calcular el punto medio entre los ojos

            eyesCenter = ((leftEyeCenter[0] + rightEyeCenter[0]) // 2,
                    (leftEyeCenter[1] + rightEyeCenter[1]) // 2)

            xEyesCenter = eyesCenter[0]
            yEyesCenter = eyesCenter[1]
            cv2.circle(image, (xEyesCenter, yEyesCenter), 3, (0, 0, 255), -1)

            M = cv2.getRotationMatrix2D(eyesCenter, angle, scale)
            M_1 = M.copy() # para visualizar efecto de corte de no modificacion de matriz
```

Para poder hallar las coordenadas para *desiredRightEyeX* hacemos uso de:

- *desiredRightEyeX = 1.0 - desiredLeftEye[0]*

La distancia entre los centroides de la imagen original se halla mediante:

- *dist_orig = np.sqrt((dX ** 2) + (dY ** 2))*

La distancia entre los ojos de la imagen resultado se calcula usando:

- *desiredDist = (desiredRightEyeX - desiredLeftEye[0])*

Hallamos la distancia:

- *desiredDist = desiredDist*desiredFaceWidth*

Para calcular la relación entre las distancias de los ojos de la imagen resultado y la imagen original se usa:

- *scale = desiredDist / dist_orig*

Para poder hallar la matriz de rotación M se debe tener en cuenta que la imagen de salida tendrá unas dimensiones definidas por las variables,

- **desiredFaceWidth:** Ancho de imagen de salida.
- **desiredFaceHeight:** Alto de imagen de salida.

Para la imagen de salida, las dimensiones definidas serán 256x256.En la implementación de la rotación 2D se requiere determinar inicialmente la matriz de transformación (M). Para esto OpenCV proporciona una función cv2.getRotationMatrix2D()

- *M = cv2.getRotationMatrix2D(center, angle, scale)*

que toma el centro de rotación ajustable, el ángulo y la escala como argumentos. La sintaxis de esta función se muestra a continuación.

- **center**: Centro de la rotación en la imagen de origen.
- **angle**: Angulo de rotación en grados. Los valores positivos significan rotación en sentido antihorario (se supone que la esquina superior izquierda es el origen de las coordenadas).
- **scale**: Factor de escala isotrópica.

La estructura de la matriz (2x3) de rotación 2D retornada por la función es la siguiente:

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha)\cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha)\cdot center.y \end{bmatrix}$$

En donde,

$$\alpha = scale \cdot \cos\theta \qquad \beta = scale \cdot \sin\theta$$

Así que inicialmente se determina el factor de escala **scale**, calculando la relación de la distancia entre los ojos en la imagen original **dist_orig** y la distancia entre los ojos en la imagen resultado **desiredDist**:

Figura 31

Distancia entre los ojos en una imagen digital

Para calcular el punto medio entre los ojos usamos:

- *eyesCenter = ((leftEyeCenter[0] + rightEyeCenter[0]) // 2, (leftEyeCenter[1] + rightEyeCenter[1]) // 2)*

Para dibujar el círculo como punto medio entre los ojos hacemos uso de:

- *xEyesCenter = eyesCenter[0]*
- *yEyesCenter = eyesCenter[1]*
- *cv2.circle(image, (xEyesCenter, yEyesCenter), 3, (0, 0, 255), -1)*

Para generar la matriz de rotación M usamos:

- *M = cv2.getRotationMatrix2D(eyesCenter, angle, scale)*

Para visualizar el efecto de corte de no modificación de la matriz hacemos uso de:

- *M_1 = M.copy()*

Figura 32

Código de programación para alineación de rostro

```
# modificacion de matriz de transformacion para evitar corte
tX = desiredFaceWidth * 0.5 # 256*0.5 punto medio ->128
tY = desiredFaceHeight *desiredLeftEye[1] # 256*0.20 ->51.2
M[0, 2] += (tX - eyesCenter[0]) #xEyesCenter
M[1, 2] += (tY - eyesCenter[1]) #yEyesCenter


#---------- Alineacion de rostro -----------------------------------
(w, h) = (desiredFaceWidth, desiredFaceHeight)
faceAligned = cv2.warpAffine(image_copy, M, (w, h),flags=cv2.INTER_CUBIC)

leftEye = shape[lStart:lEnd]
rightEye = shape[rStart:rEnd]
leftEAR = eye_aspect_ratio(leftEye)
rightEAR = eye_aspect_ratio(rightEye)

ear = (leftEAR + rightEAR) / 2.0

leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
cv2.drawContours(image, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(image, [rightEyeHull], -1, (0, 255, 0), 1)

(x, y, w, h) = face_utils.rect_to_bb(rect)

cv2.putText(image, "Face #{}".format(i + 1), (x - 10, y - 10),
    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

cv2.putText(image, "Angle: {:.5f}".format(angle), (30, 30),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
cv2.putText(image, "EyesCenter: {}".format(eyesCenter), (30, 50),
        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
```

Para ajustar cualquier problema de corte de borde de imagen, se requiere ajustar la matriz de transformación:

- *tX = desiredFaceWidth * 0.5*
- *tY = desiredFaceHeight *desiredLeftEye[1]*
- *M[0, 2] += (tX - eyesCenter[0])*
- *M[1, 2] += (tY - eyesCenter[1])*

Para alinear el rostro se debe hacer uso de una transformación afín, la cual una vez calculada la matriz de transformación (M), se le pasa a la función cv2.warpAffine() que aplica una transformación afín a una imagen. Una transformación afín es cualquier transformación que preserva la colinealidad, el paralelismo, así como la relación de distancias entre los puntos (por

ejemplo, el punto medio de una línea sigue siendo el punto medio después de la transformación). No necesariamente conserva distancias y ángulos. La sintaxis de esta función se muestra a continuación:

- Image = cv2.warpAffine (image, M, dsize, flag).

Donde:

- **Image**: imagen de entrada.
- **M**: matriz de transformación 2×3.
- **dsize**: dimensiones de la imagen de salida (w,h).
- **flag**: bandera de combinación de métodos de interpolación (cv2.INTER_CUBIC).

El proceso de alineación viene definido por:

- (w, h) = (desiredFaceWidth, desiredFaceHeight)
- faceAligned = cv2.warpAffine(image_copy, M, (w, h),flags=cv2.INTER_CUBIC)

Para calcular el EAR de la imagen alineada usamos:

- leftEye = shape[lStart:lEnd]
- rightEye = shape[rStart:rEnd]
- leftEAR = eye_aspect_ratio(leftEye)
- rightEAR = eye_aspect_ratio(rightEye)
- ear = (leftEAR + rightEAR) / 2.0

Además, para poder dibujar el contorno de los ojos para hacer más visual la ejecución del programa se hace uso de:

- leftEyeHull = cv2.convexHull(leftEye)
- rightEyeHull = cv2.convexHull(rightEye)
- cv2.drawContours(image, [leftEyeHull], -1, (0, 255, 0), 1)
- cv2.drawContours(image, [rightEyeHull], -1, (0, 255, 0), 1)

Figura 33

Código para mostrar salidas de imágenes

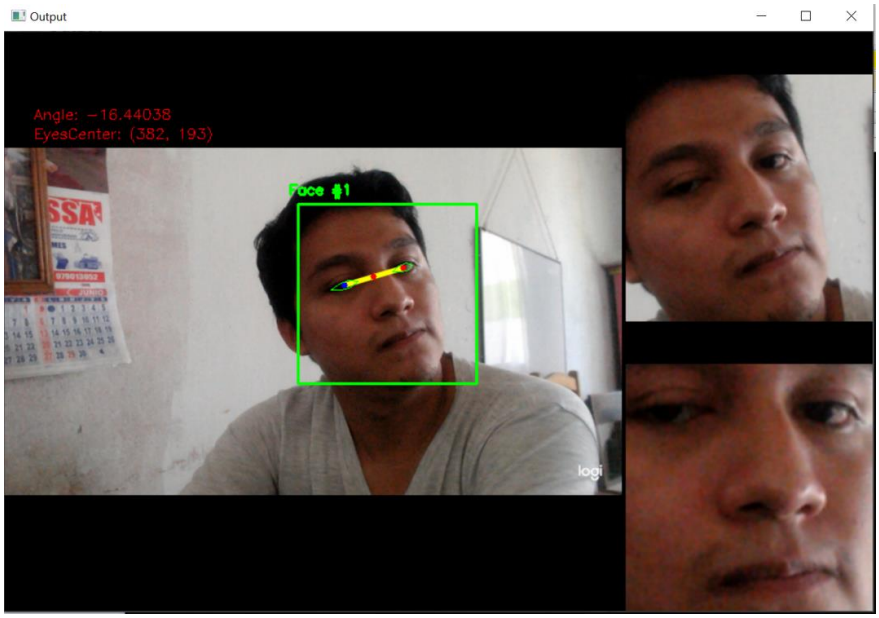
```
                salida = np.zeros((600,900, 3), np.uint8) #imagen de 900 de ancho

                salida[60:540,0:640] = image
                salida[44:300,644:900] = faceOrig
                salida[344:600,644:900] = faceAligned

                cv2.imshow("Output", salida)

        else:

            cv2.imshow("Output", image)


        tecla = cv2.waitKey(1) & 0xFF

        if tecla == 27:
                break

captura.release()
cv2.destroyAllWindows()
```

Como imágenes de salida tenemos:

- salida = np.zeros((600,900, 3), np.uint8)
- salida[60:540,0:640] = image
- salida[44:300,644:900] = faceOrig
- salida[344:600,644:900] = faceAligned

Figura 34

Muestra de salidas de imágenes (ángulo de inclinación y centro)



Para posteriormente tener la opción de cerrar la ejecución del programa mediante la tecla “Esc” o su equivalente “27” en ASCII.

3.4.7. Extracción de ojos en un rostro alineado

En este programa se hizo uso del código antes mencionado, el cual tuvo por finalidad la alineación del rostro, este nos será de utilidad pues nos permitirá realizar la extracción de la región que comprende tanto el ojo izquierdo como el derecho. A continuación, se detallará el proceso para esta extracción:

Figura 35

Proceso de la determinación de los puntos de referencia facial

```
130                 # Determinacion de puntos de referencia facial
131                 shape = predictor(gray, rect)
132                 face = alineacion(shape, image)
133 
134 
135                 #---------------  EXTRACCION DE OJOS --------------------------
136                 gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
137                 rect_face = detector(gray, 1) #RECTS = [[(),()]]
138                 if len(rect_face) == 1:
139                     print('para ojos: ',len(rect_face))
140                     shape = predictor(gray, rect_face[0])
141                     shape = face_utils.shape_to_np(shape)
142 
143                     face_copy = face.copy()
144                     face_copy_1 = face.copy()
145 
146 
147                     #------------ OJO IZQUIERDO ---------
148                     eyeL = face_copy[shape[18,1]:shape[1,1],shape[17,0]:shape[21,0]]
149                     eyeL = cv2.resize(eyeL,(80,80), interpolation = cv2.INTER_LINEAR)
150                     #------------ OJO DERECHO ---------
151                     eyeR = face_copy[shape[25,1]:shape[15,1],shape[22,0]:shape[26,0]]
152                     eyeR = cv2.resize(eyeR,(80,80), interpolation = cv2.INTER_LINEAR)
153 
154                     #-----------------------------------------------------------------
155                     (x, y, w, h) = face_utils.rect_to_bb(rect)
156                     cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
157                     cv2.putText(image, "Face #{}".format(i + 1), (x - 10, y - 10),
158                         cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

Partiendo desde la alineación del rostro, llegado al proceso de la determinación de los puntos de referencia facial, en la cual ya se obtuvo la variable *face* donde tenemos el rostro alineado.

Esta imagen será transformada a grises mediante *gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY),* las ubicaciones de los rostros detectados son retornadas por *rect_face = detector(gray, 1)*

El predictor hace un barrido de la región del rostro mediante *shape = predictor(gray, rect_face[0])* para posteriormente hacer retornar todas las coordenadas de todos los puntos de referencia facial usando *shape = face_utils.shape_to_np(shape)*

Ahora tomamos los puntos que comprenden las regiones que corresponden el ojo izquierdo, en este caso viene siendo dado por:

- *eyeL = face_copy[shape[18,1]:shape[1,1],shape[17,0]:shape[21,0]]*

Y se redimensiona en una imagen de 80x80 usando *eyeL = cv2.resize(eyeL,(80,80), interpolation = cv2.INTER_LINEAR).*

Para el ojo derecho la región que lo comprende bien dada por:

eyeR = face_copy[shape[25,1]:shape[15,1],shape[22,0]:shape[26,0]] y se redimensiona también en una imagen de 80x80 usando *eyeR = cv2.resize(eyeR,(80,80), interpolation = cv2.INTER_LINEAR).*

Para poder dibujar el rectángulo delimitador del rostro primero se debe transformar las coordenadas de tipo (x,y) de Dlib al formato (x,y,w,h) mediante *(x, y, w, h) = face_utils.rect_to_bb(rect)* , posteriormente dibujamos el rectángulo y el texto colocado encima de este usando:

- *cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)*
- *cv2.putText(image, "Face #{}".format(i + 1), (x - 10, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)*

Figura 36

Redimensionamiento de imagen inicial a un formato de 640x480

```
                        # ------------------------------------------------------------
                        #   Imagen de salida
                        # redimensionar imagen de webcam a  640x480
                        image = cv2.resize(image,(640,480), interpolation = cv2.INTER_LINEAR)
                        salida = np.ones((560,640, 3), np.uint8) #480 + 80(ancho de eyeL) = 560
                        salida[:,:]=[255,255,255]
                        salida[0:480,0:640] = image
                        salida[480:560,0:80] = eyeL
                        salida[480:560,560:640] = eyeR

                        cv2.imshow("Output", salida)
                        cv2.imshow("face_align", face)
                        cv2.imshow("OJO", eyeL)

            else:
                image = cv2.resize(image,(640,480), interpolation = cv2.INTER_LINEAR)
                cv2.imshow("Output", image)


            tecla = cv2.waitKey(1) & 0xFF

            if tecla == 27:
                    break

    captura.release()
    cv2.destroyAllWindows()
```

Para las imágenes de salida se redimensionará la imagen inicial a un formato de 640x480 mediante *image = cv2.resize(image,(640,480), interpolation = cv2.INTER_LINEAR)*

Se creará un espacio en blanco usando *salida[:,:]=[255,255,255]*

Las ubicaciones de los ojos izquierdo y derecho vienen siendo dados por:

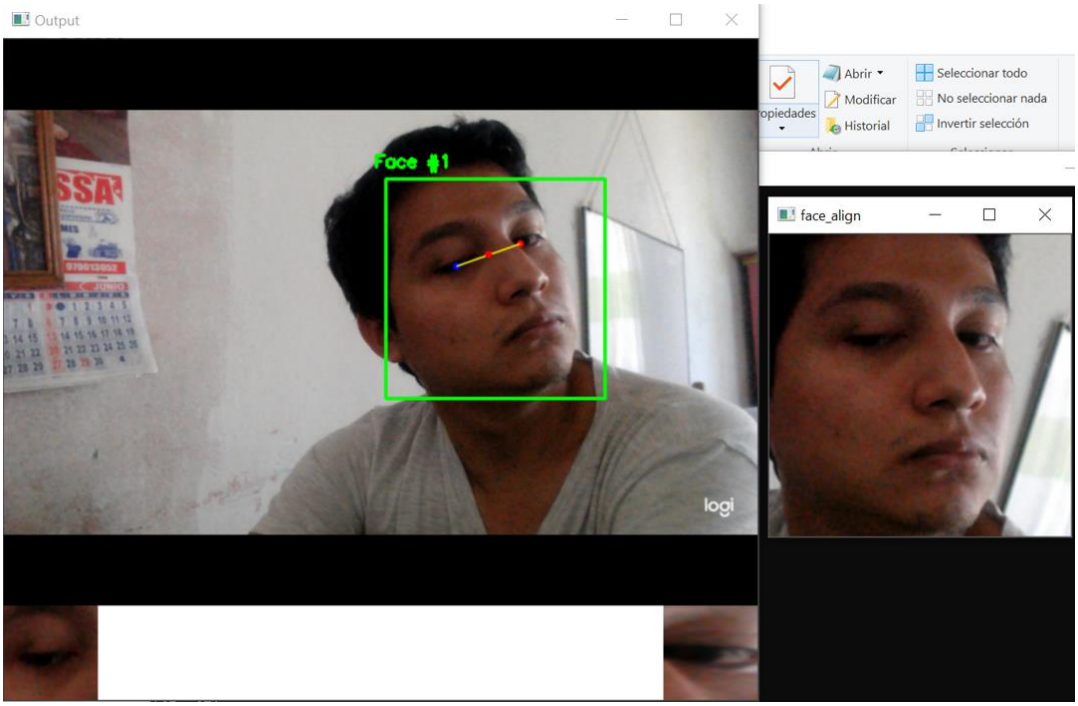
- *salida[480:560,0:80] = eyeL*
- *salida[480:560,560:640] = eyeR*

Posteriormente se muestran las 3 imágenes de salida usando:

- *cv2.imshow("Output", salida)*
- *cv2.imshow("face_align", face)*
- *cv2.imshow("OJO", eyeL)*

Figura 37

Dimensionamiento de 640 x 480



Para posteriormente tener la opción de cerrar la ejecución del programa mediante la tecla "Esc" o su equivalente "27" en ASCII.

3.4.8. Detección de ojos abiertos o cerrados usando una red neuronal preentrenada

Previo al desarrollo de este programa se instaló Keras junto al backend TensorFlow, esto para poder hacer uso de una red neuronal pre entrenada, en este caso "cnn.h5".

Figura 38

Código de programación usando una red neuronal preentrenada

```
import cv2
import os
from keras.models import load_model
import numpy as np
import time



face = cv2.CascadeClassifier('haar cascade files\haarcascade_frontalface_alt.xml')
leye = cv2.CascadeClassifier('haar cascade files\haarcascade_lefteye_2splits.xml')
reye = cv2.CascadeClassifier('haar cascade files\haarcascade_righteye_2splits.xml')



lbl=['Close','Open']

model = load_model('models/cnn.h5')
path = os.getcwd()
image = cv2.imread('1.jpg')
font = cv2.FONT_HERSHEY_COMPLEX_SMALL
count=0
score=0


height,width = image.shape[:2]

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

faces = face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25))
left_eye = leye.detectMultiScale(gray)
right_eye =  reye.detectMultiScale(gray)

cv2.rectangle(image, (0,height-50) , (200,height) , (0,0,0) , thickness=cv2.FILLED )
```

Se inicia el programa llamando a las librerías *cv2*, *os*, *numpy*, *time*. Adicionalmente de carga la función *load_model* de *Keras*, el cual nos permitirá cargar nuestra red neuronal.

Hacemos uso de los clasificadores en cascada para el rostro, ojo izquierdo y ojo derecho respectivamente:

- *face = cv2.CascadeClassifier('haar cascade files\haarcascade_frontalface_alt.xml')*
- *leye = cv2.CascadeClassifier('haar cascade files\haarcascade_lefteye_2splits.xml')*
- *reye = cv2.CascadeClassifier('haar cascade files\haarcascade_righteye_2splits.xml')*

Cargamos nuestra red neuronal y la imagen a procesar:

- *model = load_model('models/cnn.h5')*
- *path = os.getcwd()*
- *image = cv2.imread('1.jpg')*
- *font = cv2.FONT_HERSHEY_COMPLEX_SMALL*

Para poder capturar el ancho y alto de la imagen hacemos uso de *height,width = image.shape[:2]* para luego transformarla a grises mediante:

- *gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)*

Para poder tener una lista de las ubicaciones de los posibles rostros usamos:

- *faces = face.detectMultiScale(gray,minNeighbors=5,scaleFactor=1.1,minSize=(25,25))*

Donde:

  - *gray* es la imagen trasnformada a escala de grises
  - *minNeighbors=5* hace referencia a tener 5 cuadros vecinos
  - *scaleFactor=1.1* esto realiza una pirámide de imágenes
  - *minSize=(25,25)* indica que las dimensiones del recuadro delimitador es de 25x25 pixeles

Asimismo, para obtener unas listas con las ubicaciones de los posibles ojos izquierdos y derechos usamos:

- *left_eye = leye.detectMultiScale(gray)*
- *right_eye = reye.detectMultiScale(gray)*

Posteriormente se dibuja el recuadro delimitador del rostro mediante:

- *cv2.rectangle(image, (0,height-50) , (200,height) , (0,0,0) , thickness=cv2.FILLED)*

Figura 39

Determinación para dibujar rectángulo en rostro

```
for (x,y,w,h) in faces:
    cv2.rectangle(image, (x,y) , (x+w,y+h) , (100,100,100) , 1 )

for (x,y,w,h) in right_eye:
    r_eye=image[y:y+h,x:x+w]
    cv2.rectangle(image, (x,y) , (x+w,y+h) , (0,255,0) , 1 )
    count=count+1
    r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)
    r_eye = cv2.resize(r_eye,(24,24))
    r_eye= r_eye/255
    r_eye=  r_eye.reshape(24,24,-1)
    r_eye = np.expand_dims(r_eye,axis=0)
    #rpred = model.predict_classes(r_eye)
    rpred = np.argmax(model.predict(r_eye))
    #rpred = np.argmax(model.predict(x), axis=-1)
    print('prediccion ojo derecho: ', rpred)
    if(rpred==1):
        lbl='Abierto'
    if(rpred==0):
        lbl='Cerrado'
    break
```

Se hará un barrido por todos los rostros detectados para posteriormente dibujar el rectángulo usando:

- *for (x,y,w,h) in faces:*
- *cv2.rectangle(image, (x,y) , (x+w,y+h) , (100,100,100) , 1)*

Asimismo, se hace un barrido por todos los ojos derechos detectados para luego graficar el rectángulo usando:

- *r_eye=image[y:y+h,x:x+w]*
- *cv2.rectangle(image, (x,y) , (x+w,y+h) , (0,255,0) , 1)*

La imagen del ojo derecho se pasa a escala de grises mediante *r_eye = cv2.cvtColor(r_eye,cv2.COLOR_BGR2GRAY)* para luego redimensionarla a un formato de 24x24 pixeles mediante *r_eye = cv2.resize(r_eye,(24,24))* ya que bajo esas dimensiones para las imágenes fue entrenada la red neuronal usada.

Esta nueva imagen será escalada usando *r_eye= r_eye/255* y se vuelve a redimensionar con *r_eye= r_eye.reshape(24,24,-1).*

La red neuronal reconocerá a *r_eye* como un lote de imágenes mediante *r_eye = np.expand_dims(r_eye,axis=0),* para posteriormente generar una lista con las probabilidades en la predicción y devolviendo el valor máximo usando *rpred = np.argmax(model.predict(r_eye))*

La variable *rpred* se usa como un clasificador binario para determinar si el ojo derecho está o no abierto mediante:

```
if(rpred==1):
    lbl='Abierto'
if(rpred==0):
    lbl='Cerrado'
break
```

Figura 40

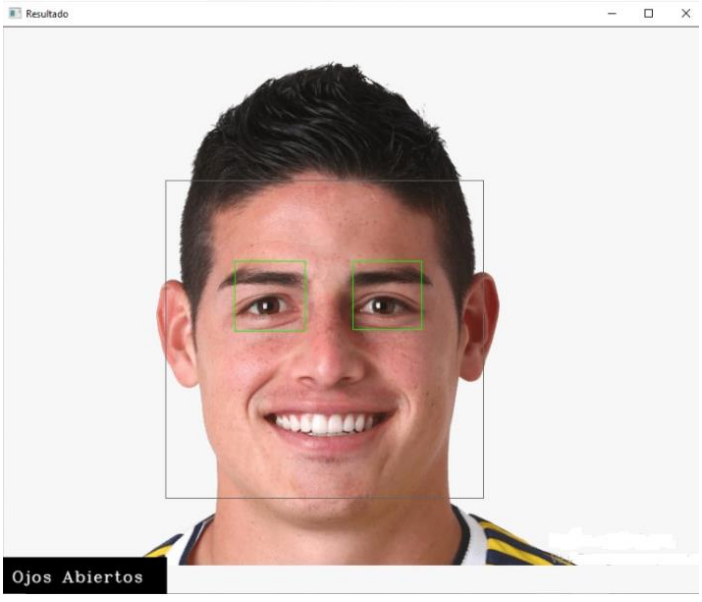
Clasificador binario

```
for (x,y,w,h) in left_eye:
    l_eye=image[y:y+h,x:x+w]
    count=count+1
    cv2.rectangle(image, (x,y) , (x+w,y+h) , (0,255,0) , 1 )
    l_eye = cv2.cvtColor(l_eye,cv2.COLOR_BGR2GRAY)
    l_eye = cv2.resize(l_eye,(24,24))
    l_eye= l_eye/255
    l_eye=l_eye.reshape(24,24,-1)
    l_eye = np.expand_dims(l_eye,axis=0)
    lpred = np.argmax(model.predict(l_eye))
    #lpred = model.predict(l_eye) # esto genera lista con probabilidades [[xxx  xxx]]
    print('prediccion ojo izquierdo: ', lpred)
    if(lpred==1):
        lbl='Abierto'
    if(lpred==0):
        lbl='Cerrado'
    break

if(rpred==0 and lpred==0):
    cv2.putText(image,"Ojos Cerrados",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

else:
    cv2.putText(image,"Ojos Abiertos",(10,height-20), font, 1,(255,255,255),1,cv2.LINE_AA)

cv2.imshow('Resultado', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Para el ojo izquierdo el proceso es el mismo que con el ojo derecho, a lo que para condicionar con ambos ojos se hace uso de las variables *rpred* y *lpred* en las cuales si ambas resultan ser 0 entonces se considera que están en un estado de “ojos cerrados” y si no es 0, que en este caso sería que sean 1 se considera que están en un estado de “ojos abiertos”.

Figura 41

El resultado que se obtuvo al someter una imagen a este código



3.4.9. Detector de somnolencia usando una red neuronal pre entrenada y alarma sonora

Este es el código final desarrollado luego de realizar diferentes pruebas y usando diferentes tipos de algoritmos, el resultado a todo esto concluyó con un código que une tanto el código de extracción de ojos en un rostro alineado unido al de detección de ojos abiertos o cerrados, adicionalmente se le agregó un efecto de "malla" la cual es un arreglo rectángulas entre líneas horizontales y verticales, fue usada para tener una mayor referencia al momento de que el voluntario se pueda ubicar frente a la cámara y quede lo más centrado posible para una mejor lectura de su rostro y ojos, así como una alarma sonora y visual en pantalla. Se detallará en este código el efecto maya y la alarma que se acopló a los códigos del apartado 3.4.7 y 3.4.8.

Figura 42

Código de importar librería y cargar predictor

```
from imutils import face_utils
import numpy as np
import cv2
import dlib
import imutils
from keras.models import load_model
from pygame import mixer
import time, sys

font = cv2.FONT_HERSHEY_COMPLEX_SMALL
predictor_path= "shape_predictor_68_face_landmarks.dat"
model = load_model('models/cnn.h5')

#--------- alarma -----------
mixer.init()
sound = mixer.Sound('alarm.wav')
score = 0
warning = cv2.imread("warning1.png")

#--------- efecto malla ----
def malla(img):
        #cv2.circle(img,(400,320), 6, (0,0,255), -1)
        for l in range(120,380,20): #lineas horizontales
                cv2.line(img,(120,l),(540,l),(230,230,230),1)
        for l in range(120,560,20): # lineas verticales
            cv2.line(img,(l,120),(l,360),(230,230,230),1)

        return img
```

Iniciamos el código llamando a las librerías *numpy, cv2, dlib, imutils, pygame, time.*

Cargamos el predictor mediante *predictor_path= "shape_predictor_68_face_landmarks.dat"* y la red neuronal usando *model = load_model('models/cnn.h5').*

Para la alarma iniciamos con el paquete mixer en el cual se encuentra el audio usado, se crea un contador llamado score el cual permitirá más adelante controlar después de cuántos segundos la alarma se activará. Además, se crea la variable warning la cual luego servirá para activar la imagen de alerta. Este proceso se realiza mediante:

- *mixer.init()*
- *sound = mixer.Sound('alarm.wav')*
- *score = 0*
- *warning = cv2.imread("warning1.png")*

Para el efecto de malla se define la función con el mismo nombre, la cual se dibuja teniendo en cuenta su amplitud tanto horizontal como vertical, en ambos casos serán 20 líneas. Esto viene definido como:

```
def malla(img):
    for l in range(120,380,20): #líneas horizontales
        cv2.line(img,(120,l),(540,l),(230,230,230),1)
    for l in range(120,560,20): # líneas verticales
       cv2.line(img,(l,120),(l,360),(230,230,230),1)
    return img
```

Posteriormente se realiza el proceso de alineación ya explicado con anterioridad, así como la extracción de la región de los ojos izquierdo y derecho, así como la predicción de la red neuronal.

Para el contador *"score"* anteriormente mencionado servirá para controlar la alarma y se detalla a continuación:

Figura 43

Código de programación para contador

```
228                     #-------------------------- resultado final ----------------
229                     if(rpred==0 and lpred==0):
230                         cv2.putText(resultado,"Ojos Cerrados",(10,30),
231                                     font, 0.8,(255,255,255),1,cv2.LINE_AA)
232                         score = score + 1
233                     
234                     else:
235                         cv2.putText(resultado,"Ojos Abiertos",(10,30),
236                                     font, 0.8,(255,255,255),1,cv2.LINE_AA)
237                         score = score - 1
```

El contador queda condicionado de tal manera de que, si resulta que se tiene los ojos cerrados, este empezará a aumentar y si los ojos son detectados como abiertos, este empezará a disminuir.

Figura 44

Código de programación para alarma

```
                    #---------------- analisis de alarma -----------------------

                    if score < 0:
                        score = 0
                    else:
                        if score > 5: # reproduccir la alarma
                            sound.play()
                            resultado[height-100:height,width -100:width]=warning

                    #-----------------------------------------------------------
                    label = 'Score: ' + str(score)
                    cv2.putText(resultado,label,(540,30),
                                font, 0.8,(255,255,255),1,cv2.LINE_AA)



                    cv2.imshow("Output", resultado)
                    cv2.imshow("Aligned", face)
                    cv2.imshow("eyeL", eyeL)
                    cv2.imshow("eyeR", eyeR)
            else:
                image = malla(image)
                cv2.imshow("Output", image)

            tecla = cv2.waitKey(1) & 0xFF

            if tecla == 27:
                    break
```

Para el análisis de la alarma, queda definida como si la variable *"score"* llegara a 0 esta dejará de reproducirse y si fuera mayor a 5, la alarma se reproducirá.

Posteriormente, se ubica la imagen "warning" en la esquina inferior derecha y se posiciona el "score" en la esquina superior derecha.

Cargamos la malla de tal manera que quede superpuesta con la imagen del voluntario en las pruebas y cargamos las imágenes de salida que en este caso son la imagen original, el rostro alineado, así como el ojo izquierdo y derecho.

Como resultado se obtiene lo siguiente:

Figura 45

Detección de ojos abiertos

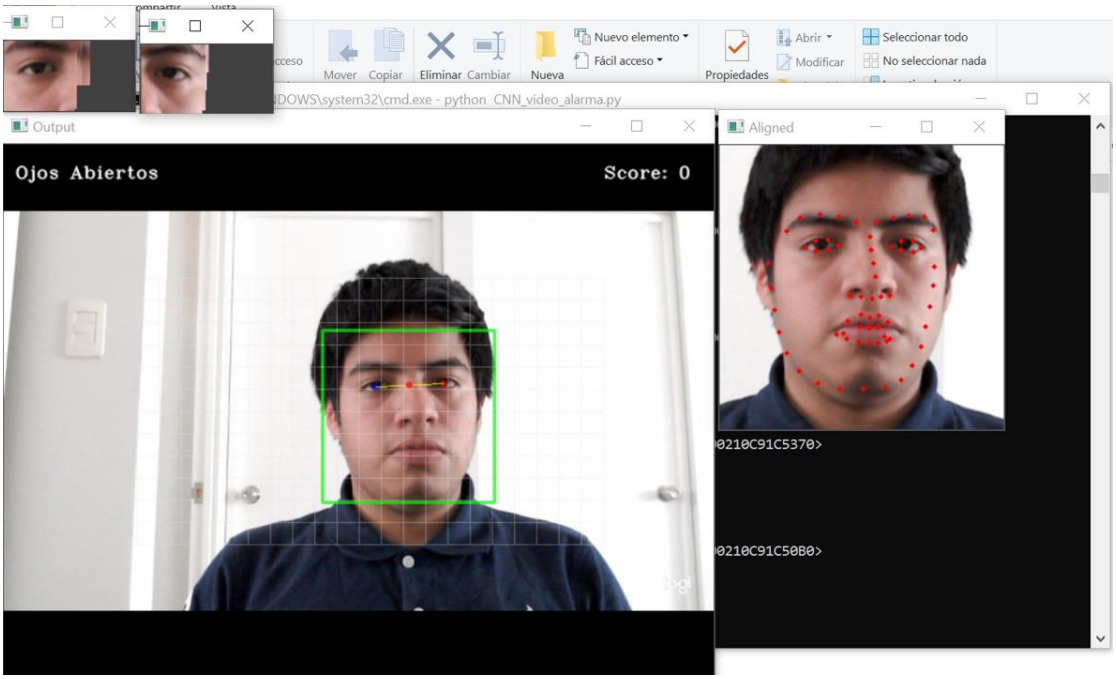
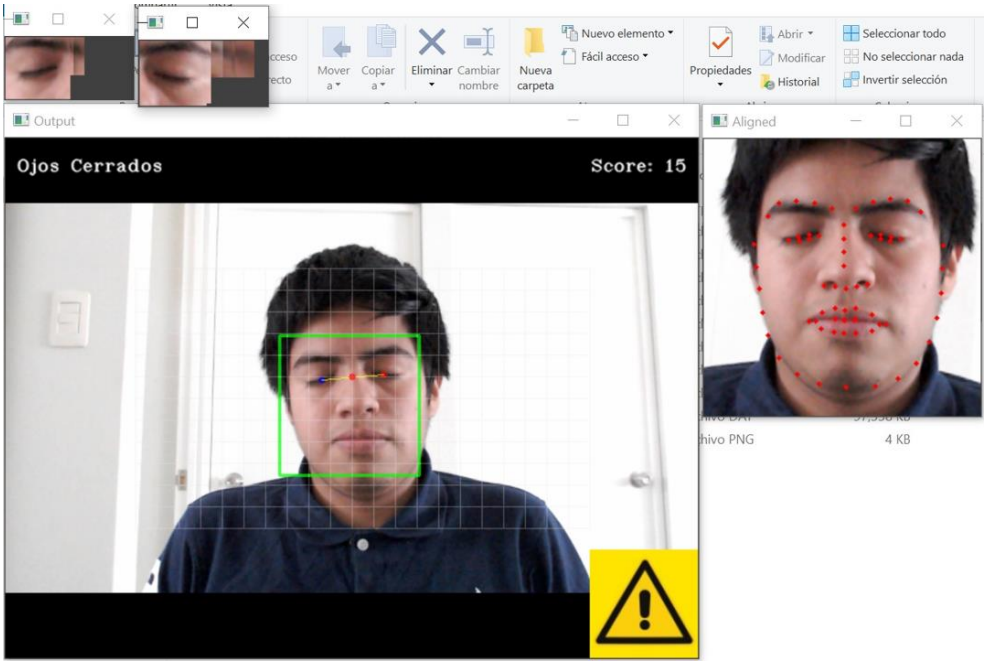


Figura 46

Detección de ojos cerrados y alarma sonora



En la imagen con los ojos abiertos se puede apreciar justamente un score en 0 y sin la alarma activada. Sin embargo, con los ojos cerrados donde ya se tiene un score mayor a 5, en este caso es 15 se tiene activa la alarma tanto en modo visual como sonoro.

Para posteriormente tener la opción de cerrar la ejecución del programa mediante la tecla “Esc” o su equivalente “27” en ASCII.

IV. RESULTADOS

Para analizar los resultados obtenidos en las pruebas realizadas, estas se hicieron tomando las siguientes consideraciones:

- Se realizaron las pruebas a 30 voluntarios en edades que oscilan los 19 a 66 años (estos estando sentados a una distancia aproximada de 40 cm a 50 cm de la cámara, lo más centrados posibles a ella y con iluminación moderada a buena).
- Las pruebas realizadas fueron hechas en un rango de 09:00 am a 06:00 pm, las pruebas realizadas de noche sólo cuando había buena iluminación eran acertadas.
- Se tomaron en consideración 3 tipos de pruebas (rostro sin accesorios, usando lentes y usando gorra), en cada una de estas pruebas se realizaron 3 subrutinas (rostro de frente, rostro con inclinación hacia la izquierda e inclinación hacia la derecha).
- Cada subrutina realizada tuvo una duración de 1 minuto a 2 minutos aproximadamente.
- Se consideró como *prueba válida (PV)* a las subrutinas donde durante esos 1 a 2 minutos no hubo resultados "falsos positivos" o "falsos negativos" (para este caso identificar ojos cerrados cuando en realidad están abiertos y viceversa o simplemente no identificar el rostro en alguna subrutina cuando sí hay presencia de este) y *prueba no válida (PNV)* cuando sí hubo presencia de estos.
- Para calcular el porcentaje de eficacia de las pruebas realizadas hacemos uso del siguiente cálculo:

$$\%\boldsymbol{Eficacia} = \left(\frac{\boldsymbol{PV}}{\boldsymbol{PV + PNV}}\right) * \mathbf{100}\%$$

Para una mejor comprensión de resultados se realizó cálculo de la eficacia en las pruebas a cada voluntario, así como también un cálculo de la eficacia de las pruebas realizadas por rutina a todos los voluntarios en general.

Teniendo todas las consideraciones en cuenta, se detalla a continuación los resultados que se obtuvieron en las pruebas realizadas:

4.1. Resultados obtenidos de cada voluntario:

Tabla 1

Resultados de Voluntario 1

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 1	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 2

Resultados de Voluntario 2

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 2	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 3

Resultados de Voluntario 3

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 3	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 4

Resultados de Voluntario 4

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 4	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 5

Resultados de Voluntario 5

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 5	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 6

Resultados de Voluntario 6

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 6	X	X	X	X	X	X	X		X
Eficacia		100%			100%			66.66%	
Eficacia total									88.89%

De aquí se puede identificar que en la subrutina donde hay inclinación del rostro hacia la izquierda usando gorra es donde hubo presencia de falsas lecturas (falsos negativos y falsos positivos), así que sólo 8 rutinas tuvieron resultados satisfactorios, por lo tanto:

$$\%eficacia = \left(\frac{8}{8+1}\right) * 100\% = 88.89\%$$

Tabla 7

Resultados de Voluntario 7

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 7	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 8

Resultados de Voluntario 8

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 8	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 9

Resultados de Voluntario 9

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 9	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 10

Resultados de Voluntario 10

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 10	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 11

Resultados de Voluntario 11

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 11	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 12

Resultados de Voluntario 12

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 12	X	X	X	X	X		X	X	X
Eficacia		100%			66.66%			100%	
Eficacia total									88.89%

De aquí se puede identificar que en la subrutina donde hay inclinación del rostro hacia la derecha usando lentes es donde hubo presencia de falsas lecturas (falsos negativos y falsos positivos), así que sólo 8 rutinas tuvieron resultados satisfactorios, por lo tanto:

$$\%eficacia = \left(\frac{8}{8+1}\right) * 100\% = 88.89\%$$

Tabla 13

Resultados de Voluntario 13

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 13	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 14

Resultados de Voluntario 14

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 14	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 15

Resultados de Voluntario 15

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 15	X	X	X	X	X	X	X	X	
Eficacia		100%			100%			66.66%	
Eficacia total									88.89%

De aquí se puede identificar que en la subrutina donde hay inclinación del rostro hacia la derecha usando gorra es donde hubo presencia de falsas lecturas (falsos negativos y falsos positivos), así que sólo 8 rutinas tuvieron resultados satisfactorios, por lo tanto:

$$\%eficacia = \left(\frac{8}{8+1}\right) * 100\% = 88.89\%$$

Tabla 16

Resultados de Voluntario 16

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 16	X	X	X	X		X	X	X	X
Eficacia		100%			66.66%			100%	
Eficacia total									88.89%

De aquí se puede identificar que en la subrutina donde hay inclinación del rostro hacia la izquierda usando lentes es donde hubo presencia de falsas lecturas (falsos negativos y falsos positivos), así que sólo 8 rutinas tuvieron resultados satisfactorios, por lo tanto:

$$\%eficacia = \left(\frac{8}{8+1}\right) * 100\% = 88.89\%$$

Tabla 17

Resultados de Voluntario 17

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 17	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 18

Resultados de Voluntario 18

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 18	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 19

Resultados de Voluntario 19

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 19	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 20

Resultados de Voluntario 20

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 20	X	X	X	X	X	X	X	X	
Eficacia		100%			100%			66.66%	
Eficacia total									88.89%

De aquí se puede identificar que en la subrutina donde hay inclinación del rostro hacia la derecha usando gorra es donde hubo presencia de falsas lecturas (falsos negativos y falsos positivos), así que sólo 8 rutinas tuvieron resultados satisfactorios, por lo tanto:

$$\%eficacia = \left(\frac{8}{8+1}\right) * 100\% = 88.89\%$$

Tabla 21

Resultados de Voluntario 21

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 21	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 22

Resultados de Voluntario 22

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 22	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 23

Resultados de Voluntario 23

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 23	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 24

Resultados de Voluntario 24

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 24	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 25

Resultados de Voluntario 25

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 25	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 26

Resultados de Voluntario 26

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 26	X	X	X	X		X	X	X	X
Eficacia		100%			66.66%			100%	
Eficacia total									88.89%

De aquí se puede identificar que en la subrutina donde hay inclinación del rostro hacia la izquierda usando lentes es donde hubo presencia de falsas lecturas (falsos negativos y falsos positivos), así que sólo 8 rutinas tuvieron resultados satisfactorios, por lo tanto:

$$\%eficacia = \left(\frac{8}{8+1}\right) * 100\% = 88.89\%$$

Tabla 27

Resultados de Voluntario 27

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 27	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 28

Resultados de Voluntario 28

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 28	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 29

Resultados de Voluntario 29

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 29	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

Tabla 30

Resultados de Voluntario 30

	Rostro sin accesorio			Lentes			Gorra		
	Frente	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***	***Frente***	***In. Izquierda***	***In. Derecha***
Voluntario 30	X	X	X	X	X	X	X	X	X
Eficacia		100%			100%			100%	
Eficacia total									100.00%

De aquí se puede identificar que en las 9 subrutinas se tuvo resultados satisfactorios por lo que la eficacia total en este caso es del 100%.

4.2. Resultados generales de las pruebas realizadas:

Tabla 31

Tabla General de Resultados Obtenidos

	Rostro sin accesorio			Lentes			Gorra			Eficacia por persona			
	Frente	*In. Izquierda*	*In. Derecha*	*Frente*	*In. Izquierda*	*In. Derecha*	*Frente*	*In. Izquierda*	*In. Derecha*	*E. Rostro S/A*	*E. lentes*	*E. Gorra*	*Efic. Total*
Voluntario 1	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 2	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 3	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 4	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 5	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 6	X	X	X	X	X	X	X		X	100%	100%	66.66%	88.89%
Voluntario 7	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 8	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 9	X	X	X	X	X	X	X	X	X	100%	100%	100.00%	100.00%
Voluntario 10	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 11	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 12	X	X	X	X	X		X	X	X	100%	66.66%	100%	88.89%
Voluntario 13	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 14	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 15	X	X	X	X	X	X	X	X		100%	100%	66.66%	88.89%
Voluntario 16	X	X	X	X		X	X	X	X	100%	66.66%	100%	88.89%
Voluntario 17	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 18	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 19	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 20	X	X	X	X	X	X	X	X		100%	100%	66.66%	88.89%
Voluntario 21	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 22	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 23	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 24	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 25	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 26	X	X	X	X		X	X	X	X	100%	66.66%	100%	88.89%
Voluntario 27	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 28	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 29	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Voluntario 30	X	X	X	X	X	X	X	X	X	100%	100%	100%	100.00%
Eficacia 1	100%	100%	100%	100%	93.33%	96.67%	100%	96.67%	93.33%				
Eficacia 2		100%			96.67%			96.67%					
Eficacia Total por categoría									97.78%				

De donde se puede identificar el porcentaje de eficacia por prueba (categoría) tomando en cuenta a los 30 voluntarios de la siguiente manera:

Pruebas con el rostro sin accesorios:

Las pruebas realizadas a los 30 voluntarios con el rostro sin accesorios (tanto de frente, inclinación hacia la izquierda e inclinación hacia la derecha), todas fueron exitosas, por lo tanto, el porcentaje de eficacia es del 100%.

Pruebas con lentes:

De las pruebas realizadas con lentes a los 30 voluntarios, los hechos con el rostro de frente tuvieron 100% de eficacia, sin embargo, hubo presencia de falsas lecturas (falsos negativos y falsos positivos) en los procedimientos realizadas con el rostro inclinado hacia la izquierda, así como a la derecha. Para calcular el porcentaje de eficacia en ambos casos se realizó de la siguiente manera:

- *Lentes con inclinación a la izquierda* en 2 voluntarios no hubo pruebas del todo exitosas, siendo 28 voluntarios donde estas fueron exitosas. Por lo tanto:

$$\%Lentes(Inc.Izq) = \left(\frac{28}{28+2}\right) * 100\% = 93.33\%$$

- *Lentes con inclinación a la derecha* en 1 voluntario no hubo pruebas del todo exitosas, siendo 29 voluntarios donde estas fueron exitosas. Por lo tanto:

$$\%Lentes(Inc.Der) = \left(\frac{29}{29+1}\right) * 100\% = 96.67\%$$

Por lo tanto, la eficacia total de las pruebas realizadas a los 30 voluntarios haciendo uso de lentes es el promedio aritmético del porcentaje de eficacia del rostro sin accesorios, con inclinación hacia la izquierda e inclinación hacia la derecha:

$$\%Eficacia(lentes) = \frac{\%Lentes\ (Frente) + \ \%Lentes(Inc.Izq) + \ \%Lentes(Inc.Der)}{3}$$

$$\%Eficacia\ (lentes) = \frac{100\% + 93.33\% + 96.67\%}{3}$$

$$\%Eficacia\ (lentes) = 96.67\%$$

Pruebas con gorra:

De las pruebas realizadas con gorra a los 30 voluntarios, los hechos con el rostro de frente tuvieron 100% de eficacia, sin embargo, hubo presencia de falsas lecturas (falsos negativos y falsos positivos) en los procedimientos realizados con el rostro inclinado hacia la izquierda, así como a la derecha. Para calcular el porcentaje de eficacia en ambos casos se realizó de la siguiente manera:

- *Gorra con inclinación a la izquierda* en 1 voluntario no hubo pruebas del todo exitosas, siendo 29 voluntarios donde estas fueron exitosas. Por lo tanto:

$$\%Gorra(Inc.Izq) = \left(\frac{29}{29+1}\right) * 100\% = 96.67\%$$

- *Gorra con inclinación a la derecha* en 2 voluntarios no hubo pruebas del todo exitosas, siendo 28 voluntarios donde estas fueron exitosas. Por lo tanto:

$$\%Gorra(Inc.Der) = \left(\frac{28}{28+2}\right) * 100\% = 93.33\%$$

Por lo tanto, la eficacia total de las pruebas realizadas a los 30 voluntarios haciendo uso de gorra es el promedio aritmético del porcentaje de eficacia del rostro sin accesorios, con inclinación hacia la izquierda e inclinación hacia la derecha:

$$\%Eficacia(gorra) = \frac{\%Gorra\ (Frente) + \ \%Gorra(Inc.Izq) + \ \%Gorra(Inc.Der)}{3}$$

$$\%Eficacia\ (gorra) = \frac{100\% + 96.67\% + 93.33\%}{3}$$

$$\%Eficacia\ (gorra) = 96.67\%$$

Cálculo del porcentaje de eficacia total de pruebas realizadas:

Este será calculado mediante el promedio aritmético de las pruebas realizadas a los 30 voluntarios en todas las categorías (rostro sin accesorios, usando lentes y usando gorra). De los cálculos anteriores se pudo calcular las eficacias necesarias para calcular la eficacia total, la cual sería:

$$\%Eficacia(Total) = \frac{\%Efic(Frente) + \ \%Efic(Lentes) + \ \%Efic(Gorra)}{3}$$

$$\%Eficacia\ (Total) = \frac{100\% + 96.67\% + 96.67\%}{3}$$

$$\%Eficacia\ (Total) = 97.78\%$$

Por lo tanto, se puede concluir que el presente trabajo luego de realizar las diferentes pruebas correspondientes a los 30 voluntarios tiene una eficacia total del 97.78%.

V. CONCLUSIONES

1. Se realizó un estudio entre los diversos métodos de detección de rostros, para lo cual se usaron diferentes tipos de herramientas matemáticas como algoritmos y clasificadores.
2. Se estudiaron las tecnologías más actuales sobre detección de somnolencia como el uso de predictores apoyados en una red neuronal dándole una eficacia y procesamiento de datos de un nivel muy superior al de los clasificadores convencionales, los cuales al mencionarlos y servirán de guía para futuros proyectos de mejora al presentado, puesto que es ampliamente escalable.
3. Se diseñó un sistema detector de somnolencia, el cual advirtió al conductor mediante una alerta sonora y ayudó a reducir la probabilidad de que esta sufra un accidente debido a algún síntoma de fatiga.
4. Se realizaron pruebas simuladas a 30 voluntarios, realizando cada uno de ellos 9 subrutinas para detectar somnolencia logrando obtener una eficacia del 97.78%, de esta manera se confirmó el correcto funcionamiento del sistema.
5. Mediante la implementación de este proyecto se fomentó el desarrollo de nuevas tecnologías en diversas aplicaciones relacionadas con la somnolencia en la vida diaria.

VI. RECOMENDACIONES

1. Teniendo en cuenta que las pruebas realizadas fueron simuladas, si se quisiera hacer un diseño portátil de este sistema se recomienda usar una placa NVIDIA Jetson Nano, puesto que está diseñada para la implementación de proyectos que hagan uso de redes neuronales, puesto que estos demandan una gran cantidad de recursos de procesamiento.
2. Se recomienda que la versión de Python con la cual se desarrolla el proyecto en el software Anaconda se actualice a la par con las librerías usadas, así como de TensorFlow puesto que hay veces que puede haber conflicto de versiones y no se podría desarrollar ningún proyecto hasta solucionar ese problema.
3. Para el uso de una cámara externa se recomienda siempre verificar el puerto asignado desde el administrador de dispositivos, puesto que cuando se actualiza la versión del sistema operativo del ordenador usado, es común que por defecto este cambie el número de puerto.
4. Se recomienda que para el uso de este proyecto sea realizado en un ordenador con un procesador de las versiones Intel Core i5 décima generación, Ryzen 5 serie 4000 o sus versiones superiores, si hubiese la posibilidad de que tenga una tarjeta gráfica independiente sería mucho mejor, puesto que el uso de redes neuronales consume bastantes recursos computacionales al ser compilados.

VII. PROPUESTA

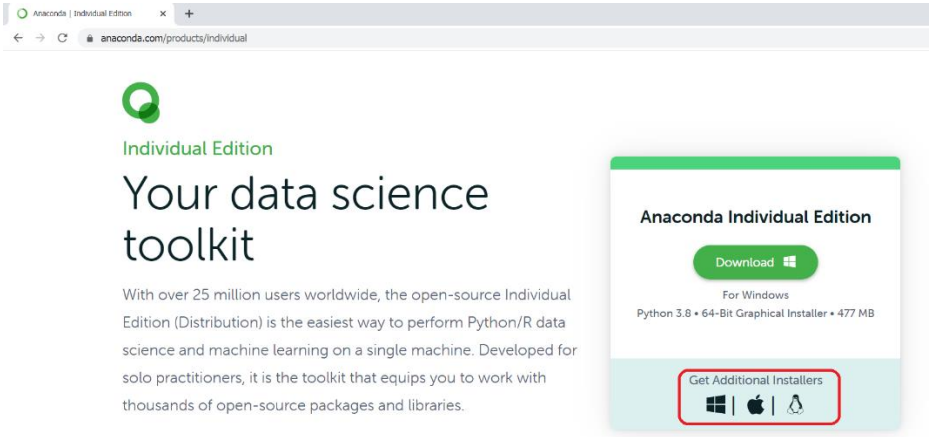
Instalación de entorno virtual en software Anaconda e instalación de librerías

Para instalar el software Anaconda primero nos vamos a dirigir a su página oficial https://www.anaconda.com/products/individual

Después tenemos que escoger nuestro sistema operativo(Windows, Linux, MacOs) según nuestra computadora/laptop (Windows en nuestro caso).

Figura 47

Página web para descarga del software anaconda



Una vez descargado el archivo nos dirigimos a nuestra carpeta de descargas y ubicamos el archivo, para después darle doble clic en ejecutar y esperar que cargue y continuamos con los pasos que se detallan en las siguientes figuras.

Figura 48

Ubicación de descarga de software

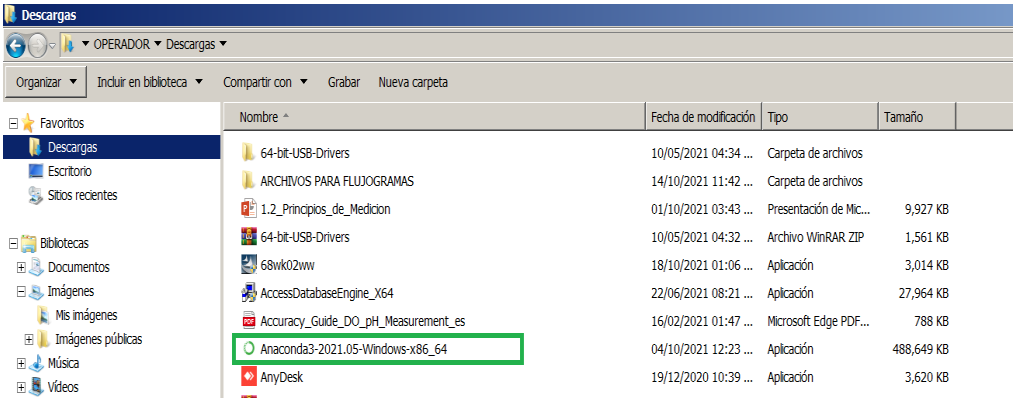


Figura 49

Ejecucion de software para instalación

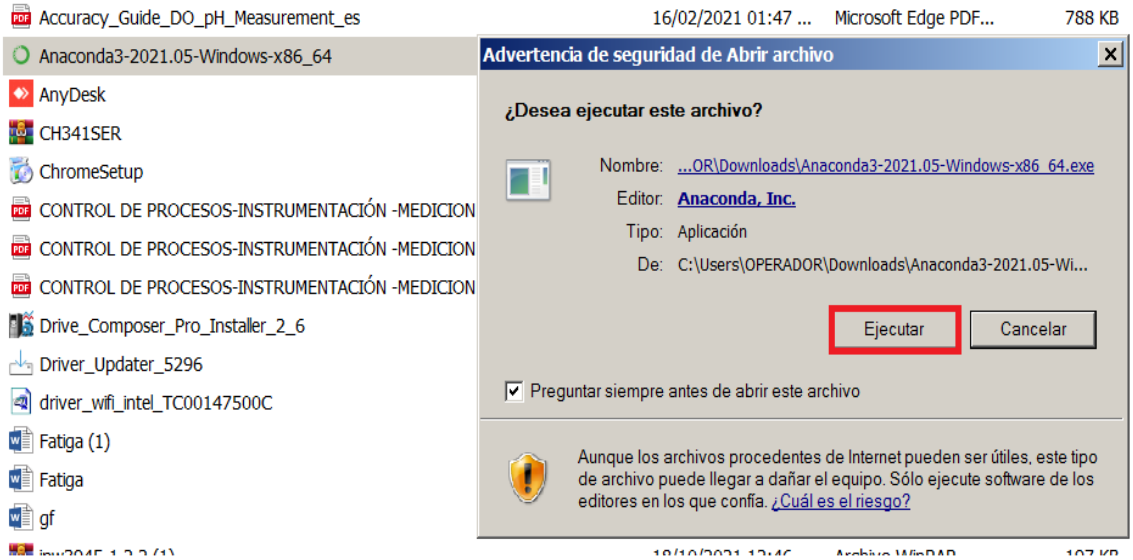
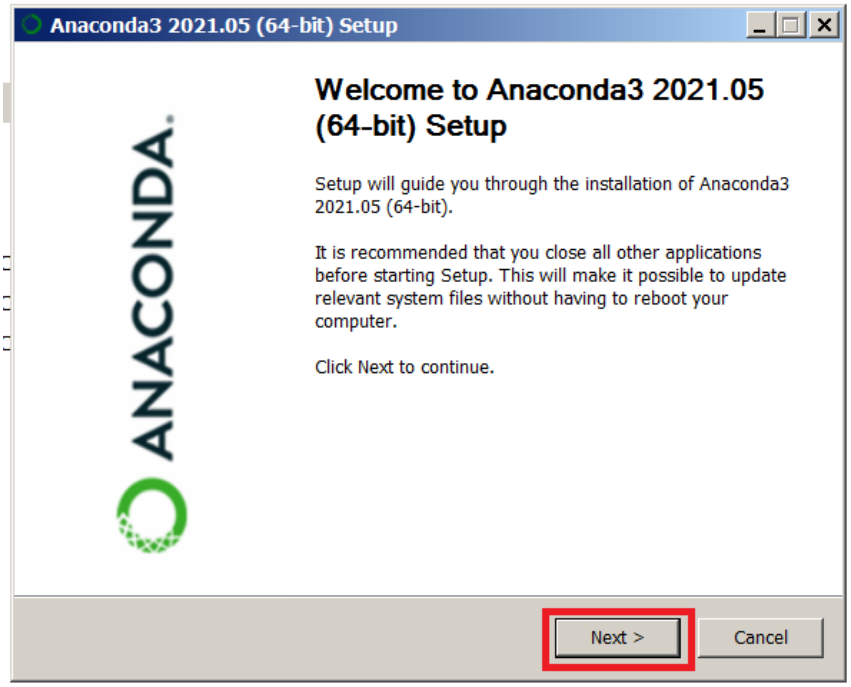


Figura 50

Click en Next para continuar con instalación



.

Figura 51

Selección de I Agree para continuar instalación

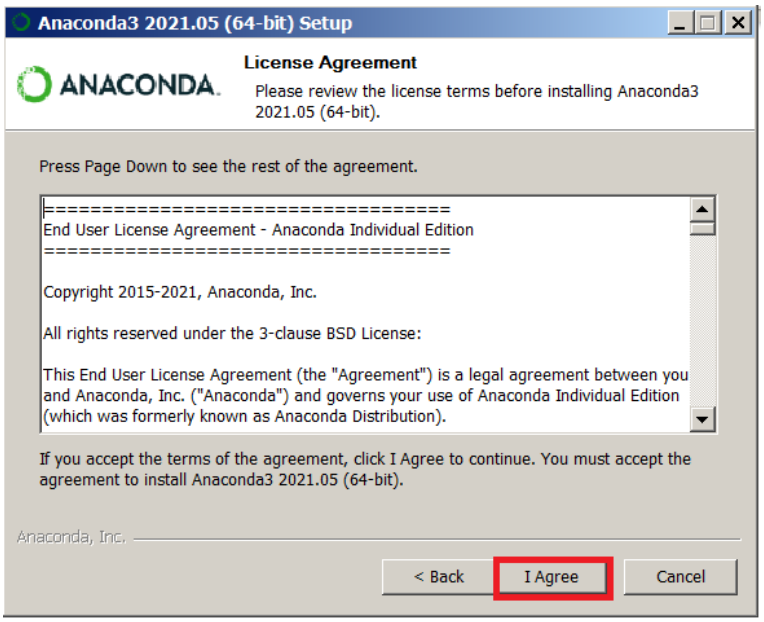
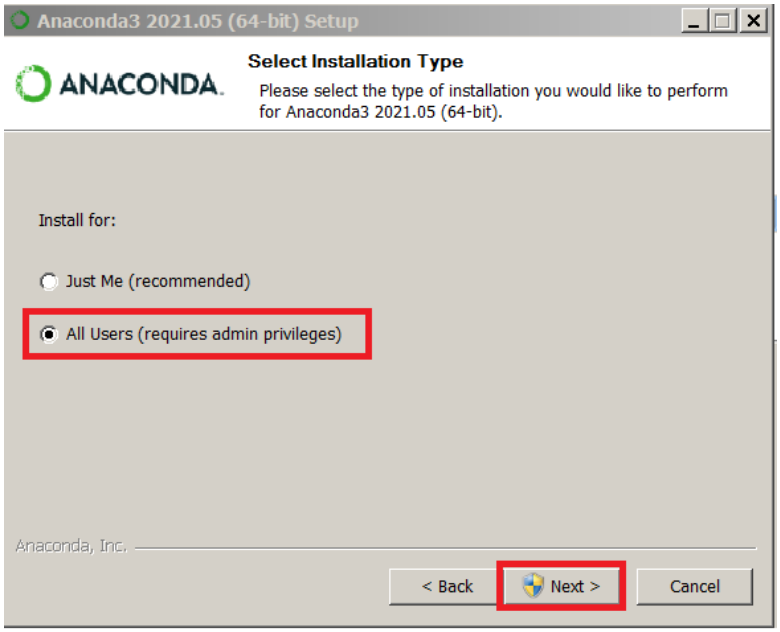


Figura 52

Selección de opción All Users (para todos los usuarios) y next para continuar



Se deja la ruta por defecto que aparece, tambien muestra el espacio requerido para la instalación en este caso 2.9 GB, se da next.

Figura 53

Escpacio a utilizar el software en nuestra instalación

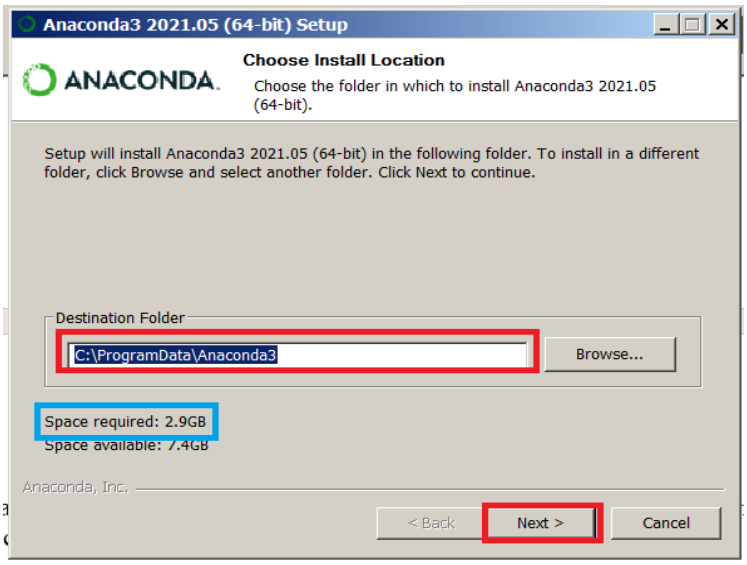
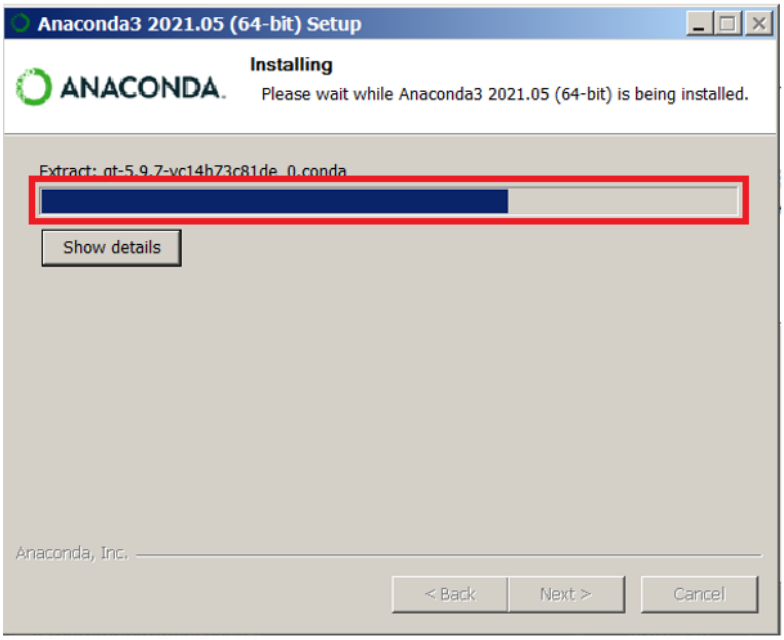


Figura 54

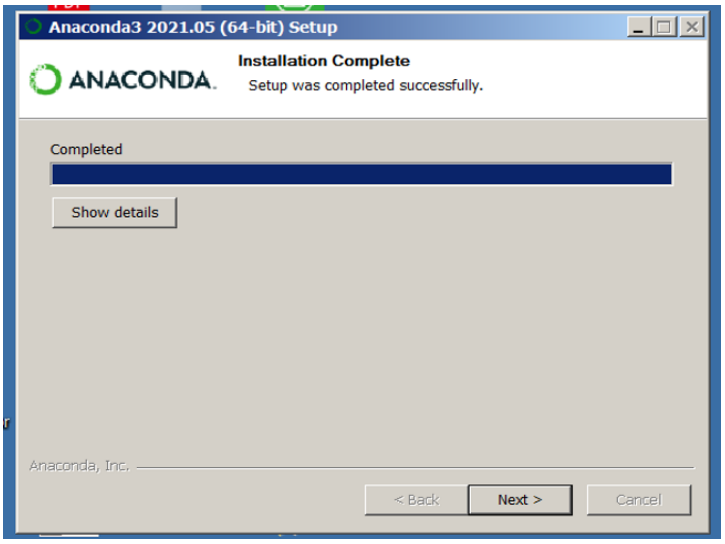
Extracción de sotfware en nuestra computadora



Una vez que termine la instalacion nos mostrara una pantalla como la siguiente en la que nos mostrara que la configuración se completó correctamente.

Figura 55

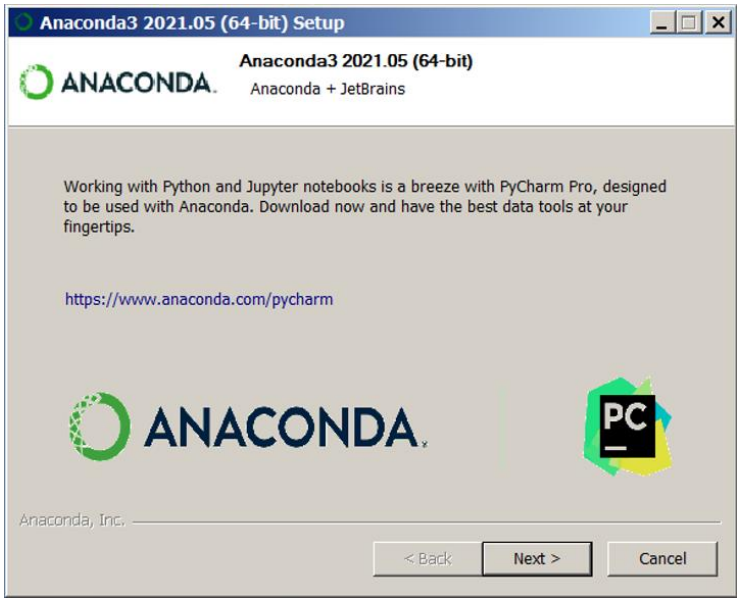
Instalación completa



Nos mostara la siguiente pantalla donde nos recomienda descargar el editor de código PyCharm y nos deja el enlace de descarga. En este caso le pasaremos a dar next.

Figura 56

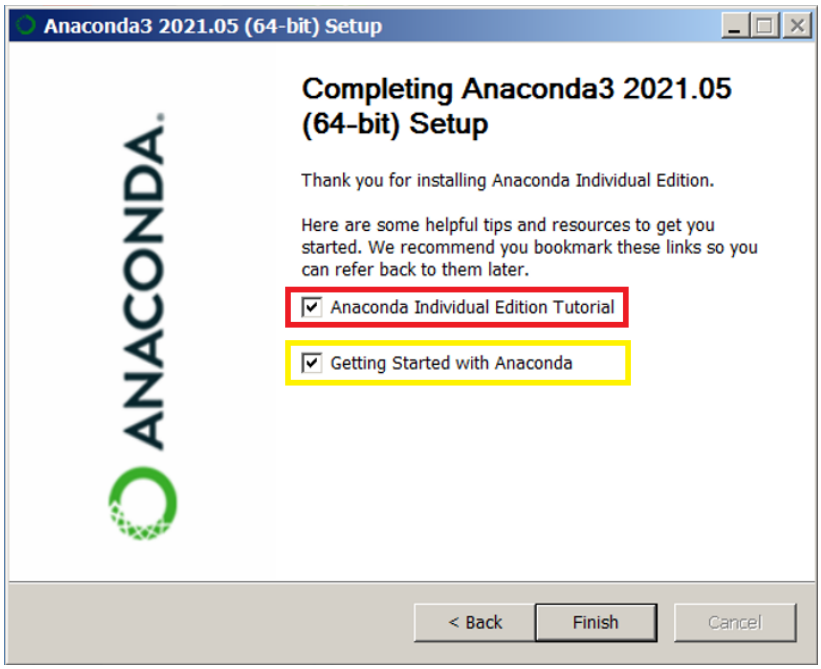
Opción de descarga de PyCharm



Por último nos aparacera una pantalla que nos muestra que la instalación de Anaconda a sido completada. Si dejamos los dos checks marcados tal y como se muestra, el primero(***Anaconda Individual Edition Tutorial***) nos llevara a un breve tutorial acerca del uso de Anaconda y el segundo(***Getting Started with Anaconda***) nos iniciará de forma automatica el programa.

Figura 57

Elección opcional de casillas para breve tutorial de anaconda

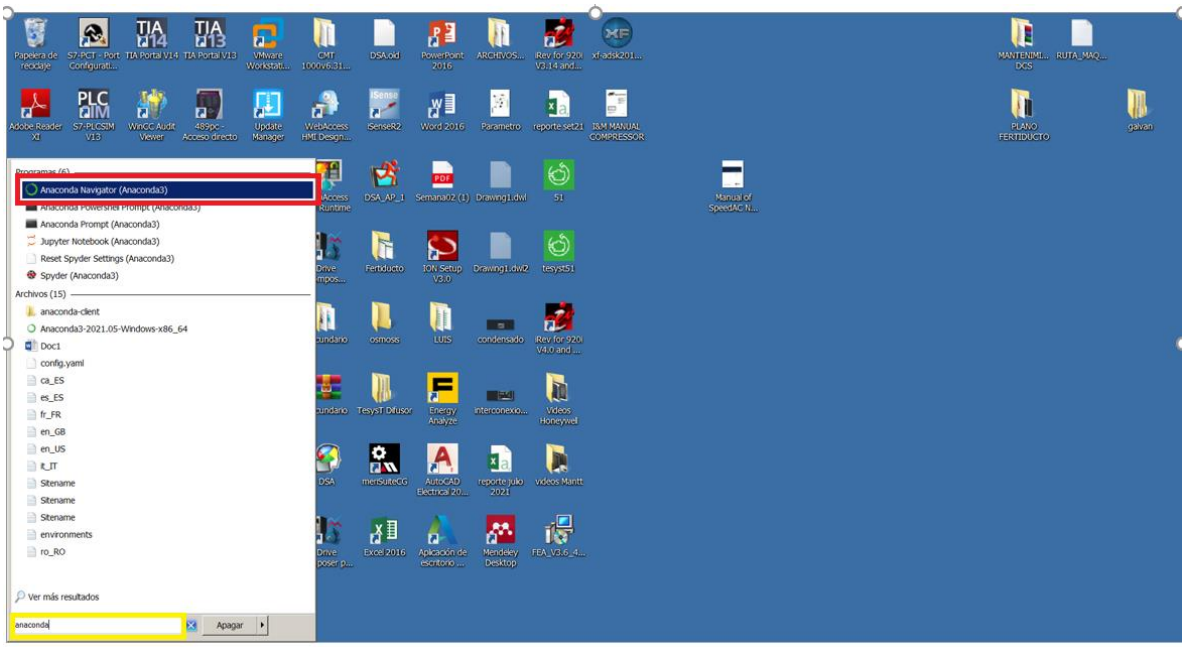


Una vez terminada la instalación de Anaconda procederemos a crear nuestro entorno virtual en donde se realizará la instalación de las librerias necesarias que se detallan a continuación que nos van a permitir desarrollar nuestro proyecto de tesis.

Abriremos el programa anaconda, para ello iremos al inicio de Windows y en la lupa escribiremos el nombre anaconda, nos aparecera tal como se muestra en la pantalla, le damos doble clic y esperamos que se inicialice el programa.

Figura 58

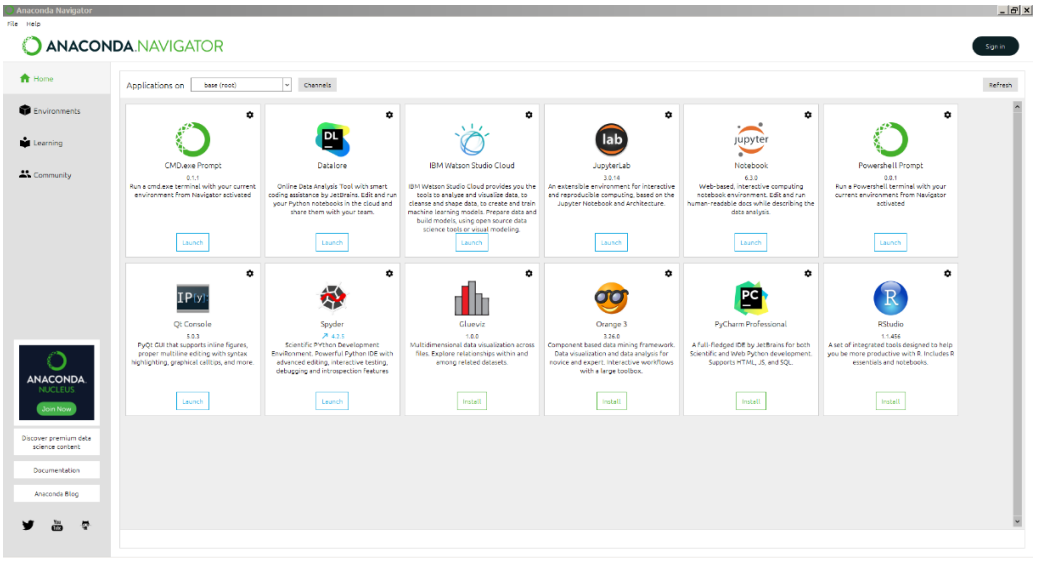
Inicilaizando el software Anaconda



Se nos abrira una pantalla que nos mostrara una suit de programas que vinen dentro de anaconda y su uso sera de acuerdo a la necesidad de cada proyecto.

Figura 59

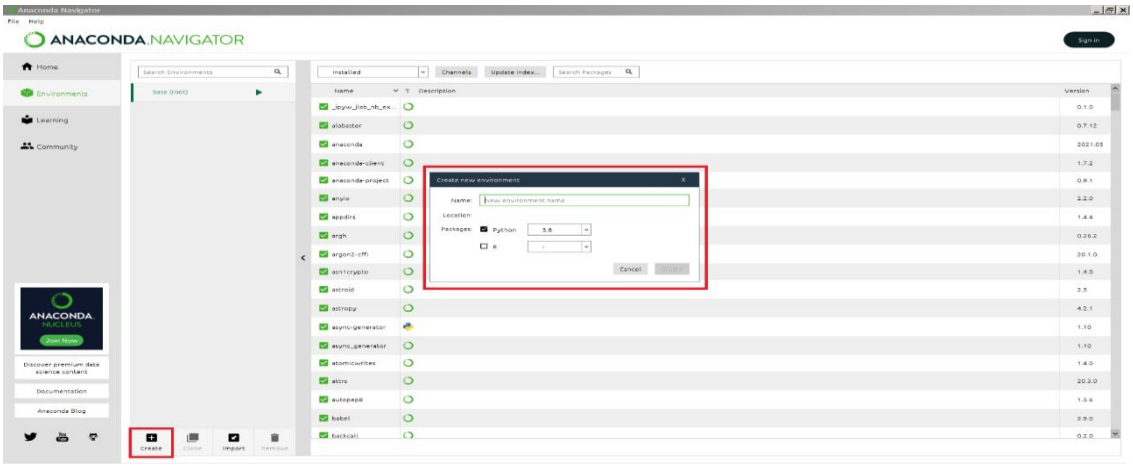
Gama de programas que incluye Anaconda



Para nuestro caso nos vamos a dirigir a Environments y vamos a crear nuestro entorno virtual para proceder a la instalación de las librerias.

Figura 60

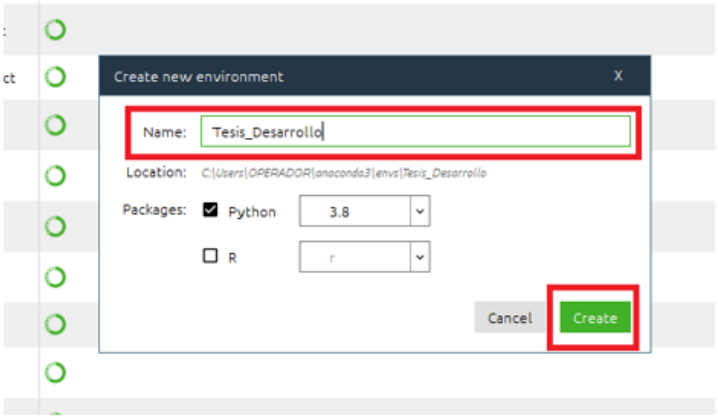
Creación de entorno virtual



Nos dirigimos en el recuadro que dice **create** y nos aparecerá una pequeña pantalla que nos pedirá un nombre, en nuestro caso le colocamos **Tesis_Desarrollo**, a la vez también nos muestra la versión de Python con la que se va a trabajar en esta versión es la 3.8.Una vez asignado el nombre le damos click en create, y esperamos que se que comience a crear nuestro entorno virtual.

Figura 61

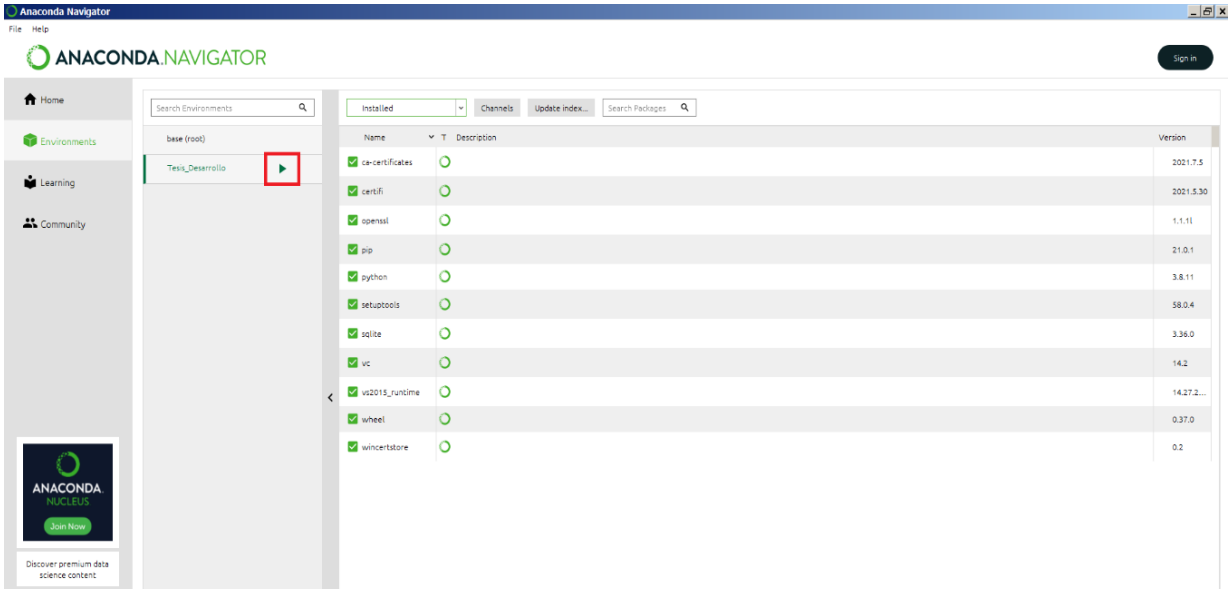
Asignación de nombre de entorno virtual



Finalizada la creación de nuestro entorno virtual nos mostrara la siguiente pantalla en la que le damos click al boton start, tal como se indica en la figura.

Figura 62

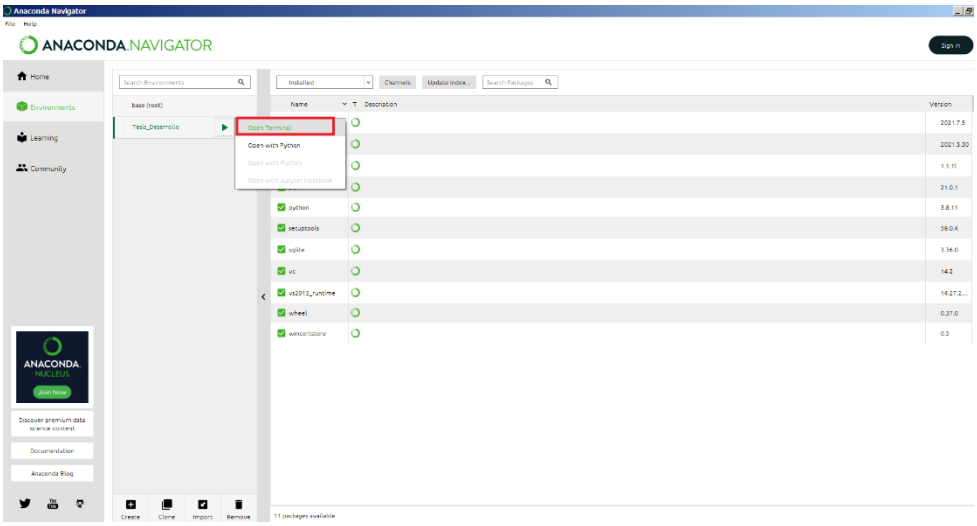
Encendido de entorno virtual



Nos aparecerá dos opciones (***open terminal*** y ***open with Python***), para la instalación de nuestras librerías le daremos click en ***open terminal***.

Figura 63

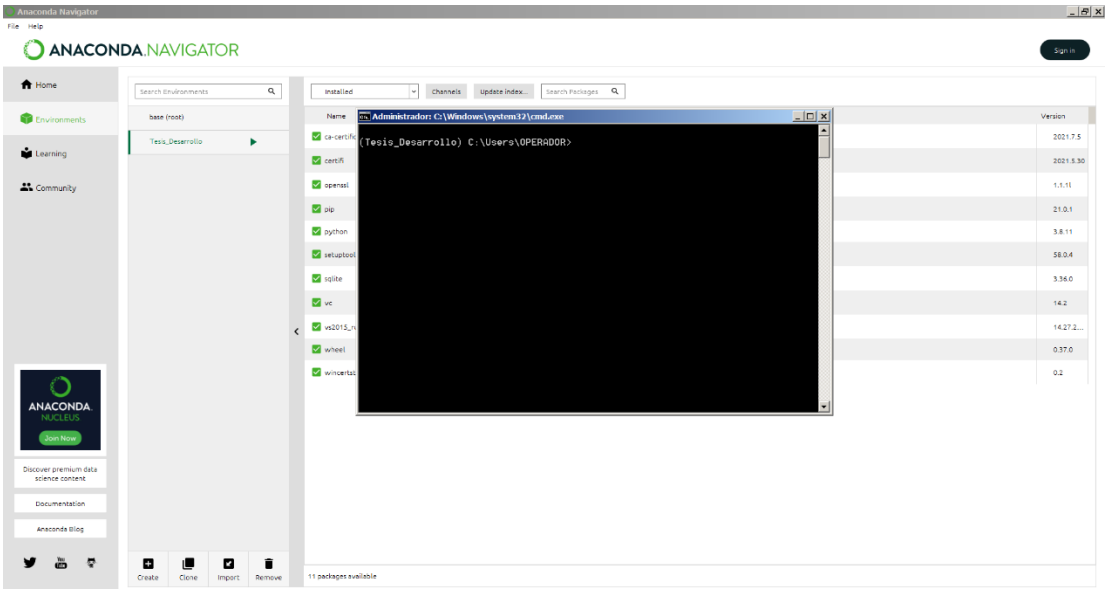
Abriendo terminal con Python



Nos aparecerá la siguiente ventana de comando en la que ya procederemos a instalar las librerías.

Figura 64

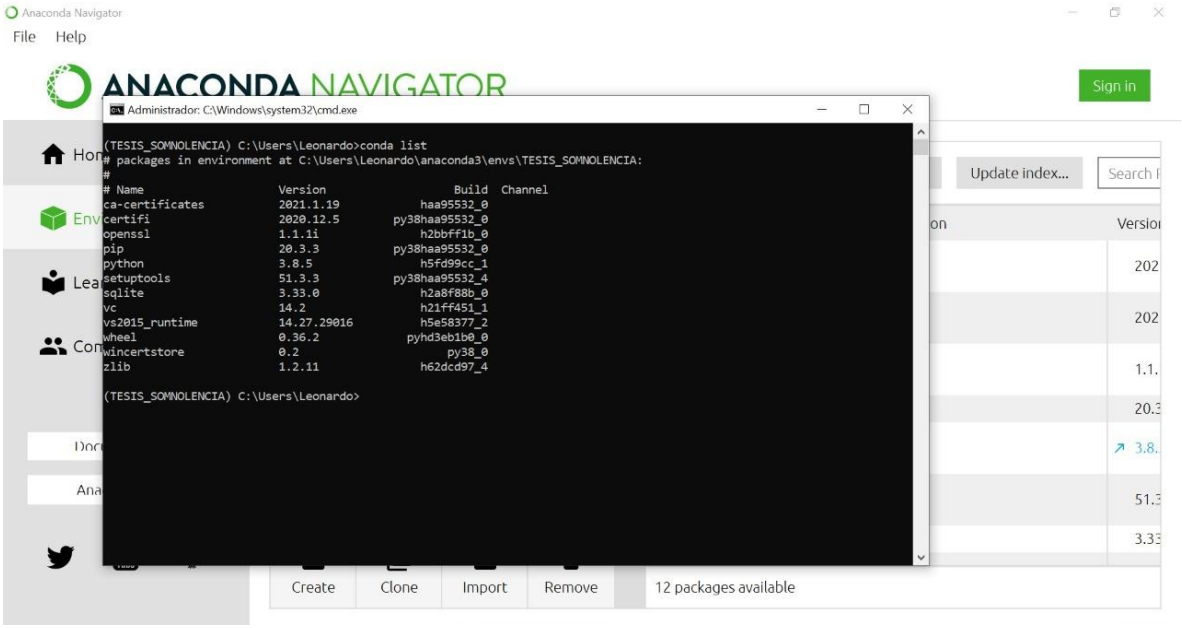
Entorno de comando para instalación de librerías



Previo a la instalación de librerías ejecutamos el comando ***conda list*** para ver la lista de las librerías instaladas en el entorno activo.

Figura 65

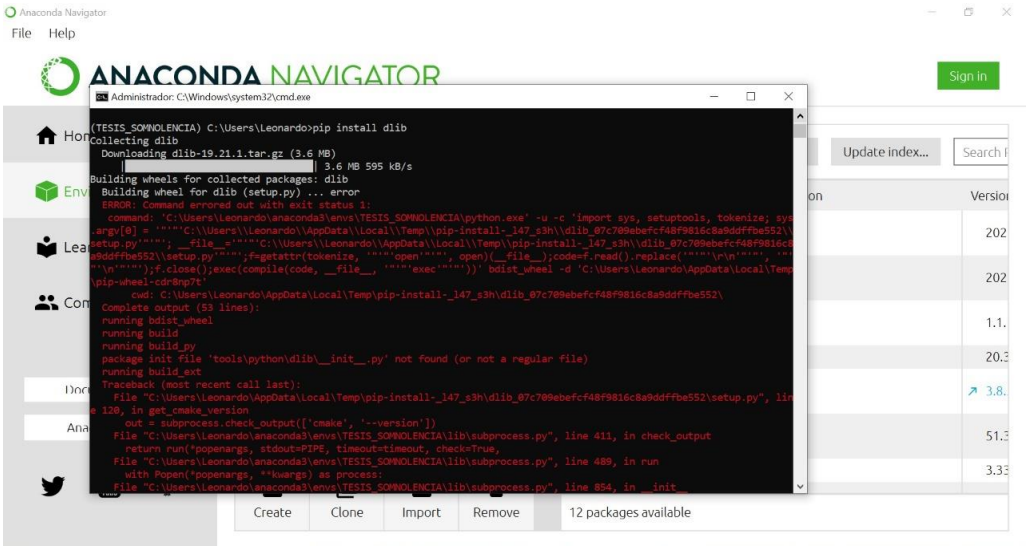
Lista de librerias y paquetes instalados en entorno virual



Procedemos a instalar la librería dlib haciendo uso del comando ***pip install dlib*** (pip es un gestor, el cual permite la instalación y administración de los paquetes en Python).

Figura 66

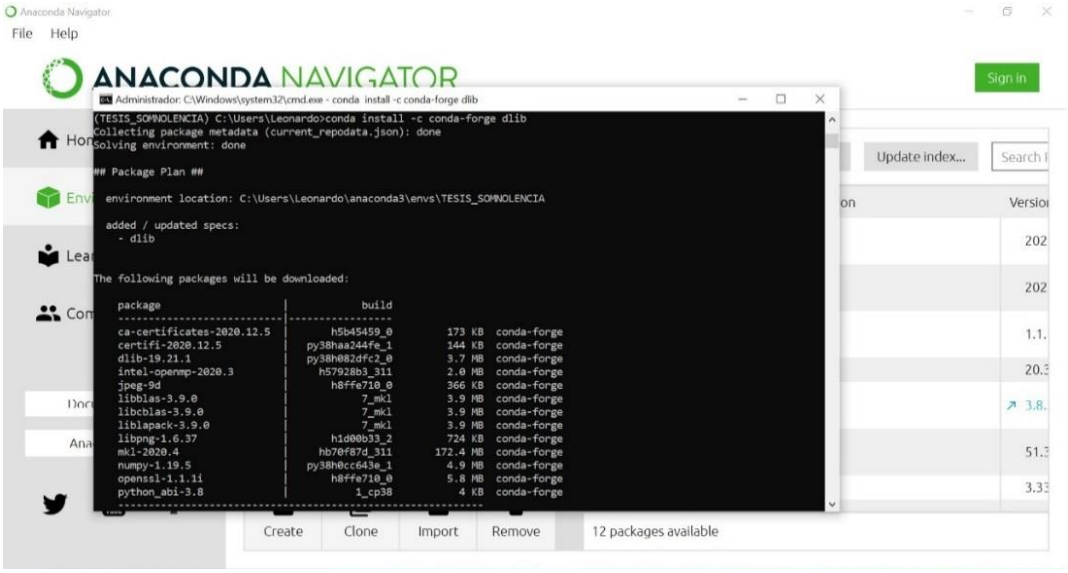
Error al instalar librería dlib



Al momento de ejecutar la instalación nos encontramos con un error, para solucionarlo hicimos uso del comando ***conda-forge dlib*** logrando una correcta instalación.

Figura 67

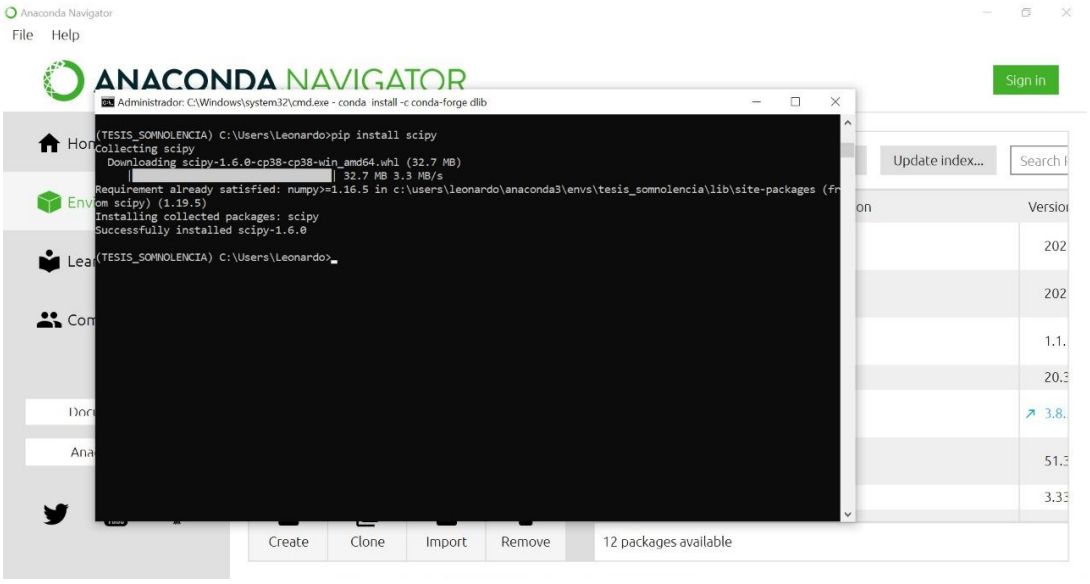
Instalación correcta de librería dlib



Ahora procedemos con la instalación de la librería scipy usando el comando ***pip install scipy***, con el cual no se obtuvo problemas de instalación.

Figura 68

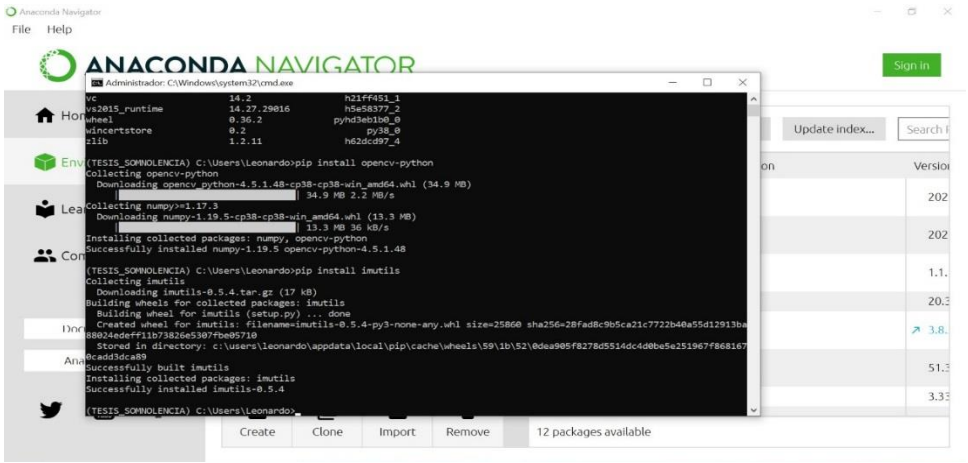
Instalación de librería scipy



Ahora damos paso para la instalación de la librería opencv, por el cual hacemos uso del comando ***pip install opencv-python***.De la misma forma instalamos la librería imutils usando ***pip install imutils***.

Figura 69

Instalación de librería opencv y imutils



Finalmente instalaremos las librerías keras, pygame y tensorflow por el cual hacemos uso del comando ***pip install*** con el respectivo nombre de la librería, teniendo así nuestro entorno listo para la programación.

Figura 70

Instalación de la librería Keras

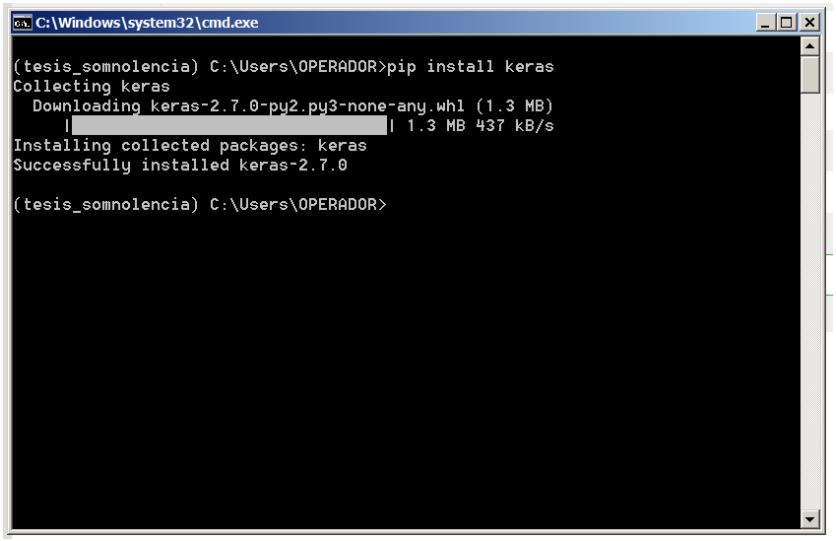


Figura 71

Instalación del módulo pygame

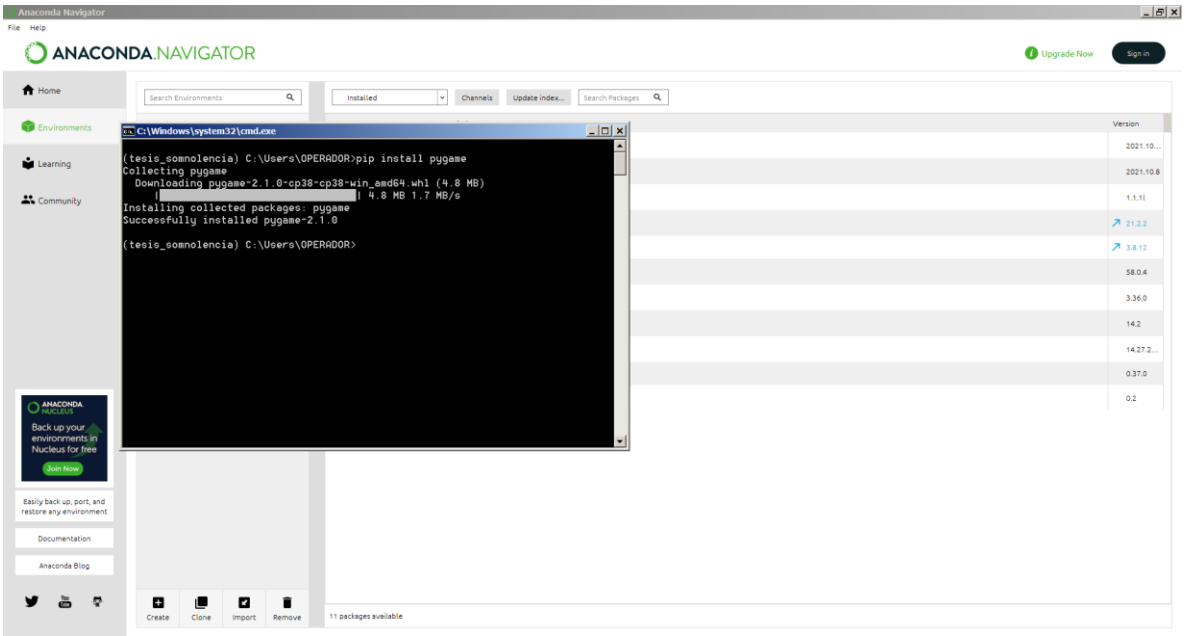
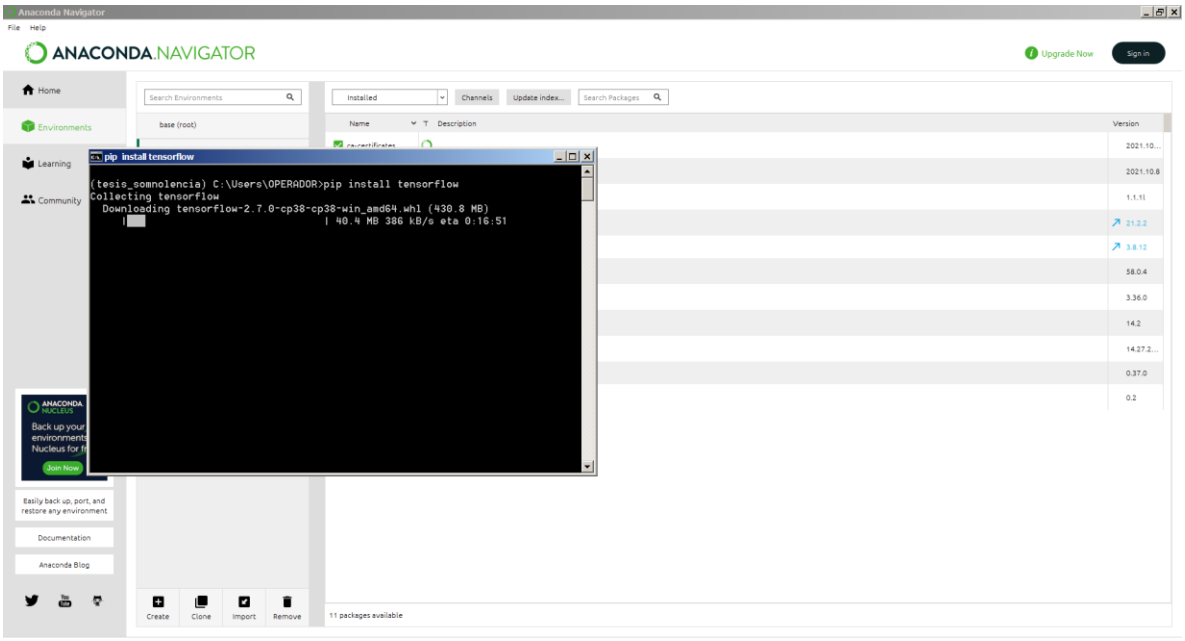


Figura 72

Instalación de la librería tensorflow



REFERENCIAS

García, E. M. i. (2018). Visión artificial. In *Visión Artificial* (p. 30).

Vezhnevets, V., Sazonov, V., & Andreeva, A. (2003). A survey on pixel-based skin color detection techniques. *International Conference Graphicon*, 8.

Dlib. (n.d.). http://dlib.net/

Soukupova, T., & Cech, J. ´. (2016). Real-time Eye blink detection using facial Landmarks. *Center for Machine Perception, Department of Cybernetics*, 8.

OpenCv. (2016). https://opencv.org/

Gonzales, R., & Woods, R. (2013). *Digital image processing* (P. E. INTERNACIONAL (Ed.); 3era Edici).

Gonzalez, H., & Velásquez, S. (2019). Reconocimiento facial utilizando Viola-Jones y patrones binarios. *Centro de Investigación de Redes Neuronales Artificiales y Robotica (CIRNAR)*, 7.

Wikinson, V. E., Jackson, M. L., Westlake, J., Stevens, B., Barnes, M., Swann, P., W.Rajaratnam, S. M., & Howars, M. E. (2013). The Accuracy of Eyelid Movement Parameters for Drowsiness Detection. *Sleep Medicine*, *9*, 10.

Boeglin Naumovic, M. (2015). *Leer y redactar en la universidad.*

García S, I., & Caranqui S, V. (2015). La visión artificial y los campos de aplicación. *Tierra Infinita*, *11*(2602–8131), 94–103.

Quevedo López, N. (2012). *Estudio del parpadeo durante la conducción de vehículos (Aspectos cognitivos y de flujo de información)*.

Fuletra, J. D., & Bosamiya, D. (2013). A Survey on Driver's Drowsiness Detection Techniques. *International Journal on Recent and Innovation Trends in Computing and Communication*, *1*(11), 816–819.

Flores, M., Armingol M, J., & De la Escalera, A. (2011). Sistema avanzado de asistencia a la conducción para la detección de la somnolencia. *Revista Iberoamericana de Automática e Informática Industrial*, *8*, 216–228.

Rondon Condori, L. A., & Nuñez Pucara, F. J. (2013). *Reconocimiento de somnolencia en conductores bajo condiciones simuladas*.

Rosales Mayor, E., & De Castro Mujica, J. R. (2010). Somnolencia: Qué es, qué la causa y cómo se mide. *Acta Médica Peruana*, *27*. http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S1728-59172010000200010

Wedro, B. (2016). *Fatigue Symptoms, Causes, and Treatment*. https://www.medicinenet.com/fatigue/article.htm

Castañeda Escobar, L., & Hernández Malacara, D. (2004). *Diversos instrumentos para el estudio del ojo humano* (p. 3).

Monclus, J., Ortega, J., Pérez, S., & Encinar, R. (2018). *ADAS(Advance Driver-Assitance Systems)*.

Becerra, F. (2019). Patrones de Conducta Facial para Identificar Accesos Informáticos no Autorizados. no. 32. http://repositorio.uss.edu.pe/handle/uss/5638

Crespin, J., & Julián, R. (2019). Sistema detector de somnolencia en secuencias de vídeo de conductores manejando usando visión computacional. 128.

Egas, F. (2017). Sistema de vigilancia al conductor vehicular basado en técnicas de visión artificial e implementado en un smartphone para la detección y alerta de somnolencia. https://repositorio.espe.edu.ec/bitstream/21000/8819/1/T-ESPEL-EMI-0262.pdf

España, L., & Oña, E. (2018). Implementación de un prototipo para la detección de signos de fatiga del conductor aplicando visión artificial en un vehículo liviano en la noche. 1–57. http://dspace.ups.edu.ec/bitstream/123456789/5081/1/UPS-CYT00109.pdf

Follonier, M., & Peroni, L. (2018). AVSA – Asistente vehicular de seguridad.

Lauriac, N. (2016). Diseño e implementación de un sistema de monitoreo. Terre Des Hommes, 46.

Mayon, E., & Limaquispe, R. (2019). Sistema de detección de somnolencia mediante inteligencia artificial en conductores de vehículos para alertar la ocurrencia de accidentes de transito. Repositorio Institucional - UNH, 80. http://repositorio.unh.edu.pe/handle/UNH/2755

Viola, P., Way, O. M., & Jones, M. J. (2019). Aplikasi Pengenalan Wajah Untuk Membuka Pintu Berbasis Raspberry Pi. *Aplikasi Pengenalan Wajah Untuk Membuka Pintu Berbasis Raspberry Pi*, *14*(2), 243–252. https://doi.org/10.35793/jti.14.2.2019.24000

Mrudula, Y., Deepthi, A., & Reddy, C. G. (2013). *Drowsiness Detection in Real Time Driving Conditions*. *2*(10), 46–57.

Jhaquelin, E., Becerril, L., & Juárez, E. T. (2021). *Una red neuronal para la detección de somnolencia en conductores*. *22*, 1–9.

García, U. (2019). *Introducción a las Redes Neuronales Pt. I | Future Lab*. https://futurelab.mx/redes neuronales/inteligencia artificial/2019/06/25/intro-a-redes-neuronales-pt-1/

Rosebrok, A. (2017). *Eye blink detection with OpenCV, Python, and dlib - PyImageSearch*. https://www.pyimagesearch.com/2017/04/24/eye-blink-detection-opencv-python-dlib/

Rosebrok, A. (2017). *Raspbian Stretch: Install OpenCV 3 + Python on your Raspberry Pi - PyImageSearch*. https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

García, E. M. i. (2002). Visión Artificial. *Inteligencia Artificial*, 115.

González, E. J. B. (2017). *Programando redes neuronales paso a paso con Python*.

Mordvintsev, A., & Abid, K. (2017). OpenCV-Python Tutorials Documentation. *OpenCV Python Documentation*, 269. https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf

Rodríguez Montiel, M. (2015). *Estudio de las características del parpadeo, y su relación con los movimientos sacádicos, en distintas condiciones controladas de lectura*. 1. https://upcommons.upc.edu/bitstream/handle/2117/89328/margarita.rodriguez.montiel - marga_rodriguez_montiel_tfm.pdf?sequence=1&isAllowed=y

Castro, J. R. de, Gallo, J., & Loureiro, H. (2004). Cansancio y somnolencia en conductores de ómnibus y accidentes de carretera en el Perú: estudio cuantitativo. *Revista Panamericana de Salud Pública*, *16*(1), 11–18. https://doi.org/10.1590/s1020-49892004000700002

Garcés, M. A., Salgado, J. D. J., Cruz, J. A., & Cañon, W. H. (2015). Sistemas de detección de somnolencia en conductores: inicio, desarrollo y futuro. *Ingeniería y Región*, *13*(1), 159. https://doi.org/10.25054/22161325.717

Aravind, A., Agarwal, A., Jaiswal, A., Panjiyara, A., & Mallikarjun Shastry, P. M. (2019). Fatigue detection system based on eye blinks of drivers. *International Journal of Engineering and Advanced Technology*, *8*(5 Special Issue), 72–75.

Lewandowski, C. M., Co-investigator, N., & Lewandowski, C. M. (2015). Learning OpenCV 3 Computer Vision with Python. In *The effects of brief mindfulness intervention on acute pain experience: An examination of individual difference* (Vol. 1).

Garcia-agundez, A., Ochs, T., Konrad, R., Caserman, P., & Göbel, S. (2018). *Eye Aspect Ratio based Blink Rate detection and its potential use for Parkinsons Disease*.

ANEXOS

Flujograma del sistema de monitoreo

Figura 73

Flujograma – Parte 1

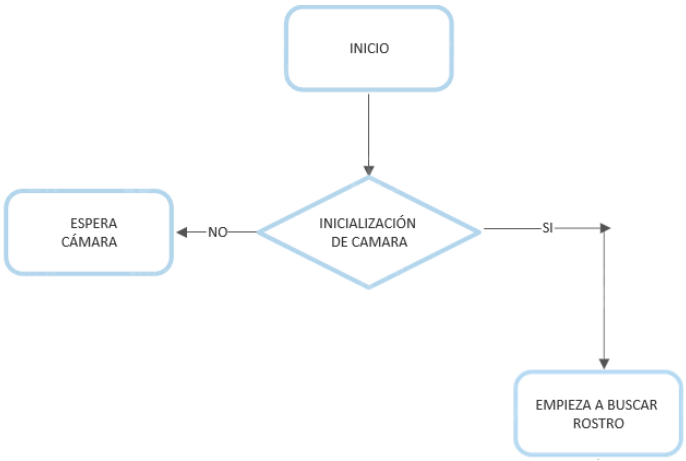


Figura 74

Flujograma – Parte 2

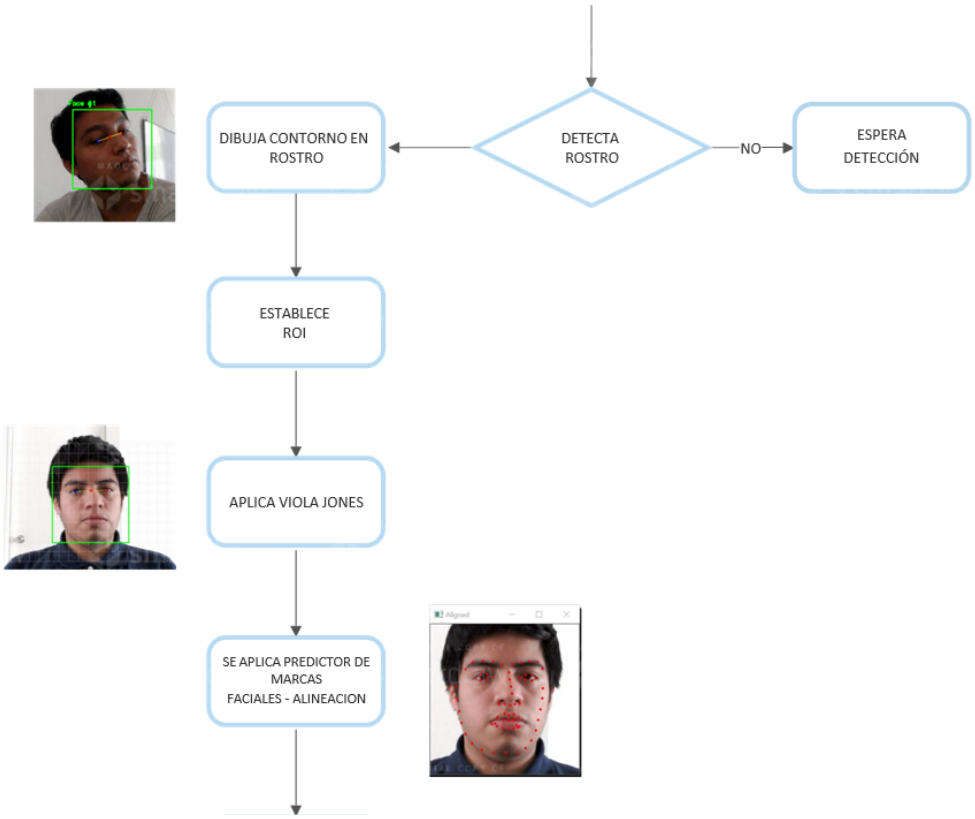


Figura 75

Flujograma – Parte 3