



UNIVERSIDAD POLITÉCNICA SALESIANA SEDE QUITO

CARRERA DE COMPUTACIÓN

DESARROLLO DE UN PROTOTIPO MÓVIL PARA IDENTIFICAR
CONTORNO DE OJOS EN IMÁGENES DE ROSTROS HUMANOS
UTILIZANDO PROGRAMACIÓN EN PARALELO PARA UN SISTEMA
DE AUTENTICACIÓN BIOMÉTRICA

Trabajo de titulación previo a la obtención del
Título de Ingeniero en Ciencias de la Computación

AUTORES:

DANIEL MARCELO MOINA CAMPOS

MARIO ALBERTO SUIN CARCHIPULLA

TUTOR:

WASHINGTON ARSENIO RAMÍREZ MONTALVAN

Quito - Ecuador
2022

**CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE
TITULACIÓN**

Nosotros, Daniel Marcelo Moina Campos con documento de identificación N° 1725014490 y Mario Alberto Suin Carchipulla con documento de identificación N° 1727350785 , manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Quito, 18 de marzo del año 2022.



Daniel Marcelo Moina Campos

1725014490



Mario Alberto Suin Carchipulla

1727350785

**CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE
TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA**

Nosotros, Daniel Marcelo Moina Campos con documento de identificación N° 1725014490 y Mario Alberto Suin Carchipulla con documento de identificación N° 1727350785 , expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos los autores del Proyecto Técnico: “Desarrollo de un prototipo móvil para identificar contorno de ojos en imágenes de rostros humanos utilizando programación en paralelo para un sistema de autenticación biométrica”, el cual ha sido desarrollado para optar por el título de: Ingeniero en Ciencias de la Computación, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la biblioteca de la Universidad Politécnica Salesiana.

Quito, 18 de marzo del año 2022.



Daniel Marcelo Moina Campos

1725014490



Mario Alberto Suin Carchipulla

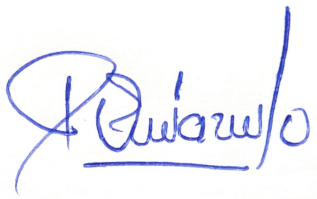
1727350785

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Washington Arsenio Ramírez Montalvan con documentos de identificación N° 1710804681, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DESARROLLO DE UN PROTOTIPO MÓVIL PARA IDENTIFICAR CONTORNO DE OJOS EN IMÁGENES DE ROSTROS HUMANOS UTILIZANDO PROGRAMACIÓN EN PARALELO PARA UN SISTEMA DE AUTENTICACIÓN BIOMÉTRICA, realizado por Daniel Marcelo Moina Campos con documento de identificación N° 1725014490 y por Mario Alberto Suin Carchipulla con documento de identificación N° 1727350785 , obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Quito, 18 de marzo del año 2022.

Atentamente,



Ing. Washington Arsenio Ramírez Montalvan, Ph.D.

1710804681

ÍNDICE GENERAL

Introducción	1
Problema	2
Antecedentes	5
Alcance	8
Objetivos	9
Objetivo General	9
Objetivos Específicos	9
1. Fundamentos Teóricos	10
1.1. Visión por computadora	10
1.1.1. Tratamiento de imagenes	10
1.1.2. Ruido	10
1.1.3. Filtro	11
1.1.4. Problemas de iluminación	12
1.2. Preprocesamiento de imágenes	13
1.2.1. Histograma de gradientes orientados HOG	13
1.2.2. Canny	15
1.2.3. Circulo Hough	16
1.2.4. Filtro mediano híbrido	16
1.2.5. Filtro gaussiano	18
1.3. DLIB	18
1.3.1. shape predictor 68 face landmarks	19
1.4. Computación paralela	21
1.4.1. Programación paralelo	21
1.4.2. Paradigma de programación en paralelo	22
1.4.3. Problemas de programación en paralelo	23

1.4.4.	Granularidad de problemas	24
1.4.5.	Formas de paralelismo	25
1.4.5.1.	Datos Paralelizados y Funciones paralelizadas	25
1.4.5.2.	Segmentación	25
1.5.	OpenMP	27
1.5.1.	Directivas OpenMP	28
1.5.1.1.	Parallel	28
1.5.1.2.	shared	28
1.5.1.3.	reduction	28
1.5.1.4.	Critical	28
1.6.	Calidad de imagen	28
1.6.1.	MSE	28
1.6.2.	Relación pico señal ruido (PSNR)	29
1.7.	Evaluación de rendimiento	29
1.7.1.	Tiempo de ejecución	30
1.7.2.	Memoria RAM (memoria de acceso aleatoria)	30
1.7.3.	CPU (Unidad central de procesamiento)	30
1.7.4.	Threads (Hilos)	30
1.8.	Lenguajes de programación y herramientas	31
1.8.1.	Native C++ en Android Studio	31
1.8.2.	C++	31
1.8.3.	Cmake	31
1.8.4.	Java	31
1.8.5.	Jni	32
2.	Marco Metodológico	33
2.1.	Diseño y construcción de el prototipo móvil	33
2.1.1.	Fase 1: Captura de la imagen 2D	34
2.1.2.	Fase 2: Detección de rostro humano y landmark	36
2.1.3.	Fase 3: Extraer puntos referenciales de la ubicación de los ojos y recorte de la imagen del contorno de los ojos	39
2.1.4.	Fase 4: Aplicar filtro mediano hibrido e identificación de brillo en la imagen	42

2.1.4.1.	Identificar reflejo (reflexión especular)	44
2.1.5.	Fase 5: Crear imagen resultante y evaluación de rendimiento proceso secuencial y paralelo	47
2.1.5.1.	Evaluación rendimiento secuencial	48
2.1.5.2.	Evaluación rendimiento paralelo	49
3.	Resultados	50
3.1.	Herramientas	50
3.2.	Hardware	51
3.3.	Software	51
3.4.	Compilación	52
3.4.1.	Compilación primera etapa reconocimiento facial	52
3.4.2.	Compilación segunda etapa técnicas de procesamiento de imágenes	55
3.4.3.	Evaluación técnicas procesamiento de imágenes Secuencial / Paralelo	58
3.4.3.1.	Evaluación técnicas Reflejo especular Secuencial / Paralelo	61
3.4.3.2.	Evaluación tecnicas Canny Secuencial / Paralelo	64
3.4.3.3.	Evaluación técnicas AdjGamma Secuencial / Paralelo	66
3.4.3.4.	Evaluación tecnicas Circle Hough Secuencial / Paralelo	68
3.4.3.5.	Cantidad de hilos por técnica	71
3.4.3.6.	Comparación evaluación de técnicas	73
	Conclusiones	77
	Recomendaciones	79
	Bibliografía	80
Referencias		80

ÍNDICE DE FIGURAS

1.	Ojo con presencia de reflejo especular.	12
2.	Representación de gradiente	13
3.	Cambio de intensidad horizontal vertical.	14
4.	Vector resultante del cálculo de gradiente.	14
5.	Flujo de trabajo del filtro mediano híbrido.	17
6.	Face landmark detection.	21
7.	Tipos de granularidad	25
8.	Tipos de granularidad	26
9.	Segmentación multiples instrucciones	27
10.	Directiva Critical	29
11.	Fases de diseño prototipo móvil	34
12.	Captura de la imagen 2D	35
13.	Implementación de la técnica Landmark	37
14.	Captura del rostro con la técnica Landmark	38
15.	Detección puntos de referencia de los ojos en un rostro	41
16.	Recorte del sector del ojo.	42
17.	Proceso de recorte imagen, aplicación filtro y tratamiento reflejo especular	42
18.	Imagen tratamiento de filtro y reflejo especular	47
19.	Consumo CPU shape Predictor	52
20.	Consumo Memoria (kb) shape Predictor	53
21.	Consumo Tiempo (ms) shape Predictor	54
22.	Consumo CPU metricas Calidad de imagen	55
23.	Consumo Memoria (kb) calidad de imagen	56
24.	Consumo Tiempo (ms) calidad de imagen	57

25.	MSE	58
26.	Relación PSNR de la señal ruido	59
27.	Consumo Memoria (kb) Reflejo especular	62
28.	Consumo Tiempo (ms) Reflejo especular	62
29.	Consumo CPU evaluacion Reflejo especular	63
30.	Consumo Memoria (kb) Canny	64
31.	Consumo Tiempo (ms) Canny	65
32.	Consumo CPU evaluación Reflejo especular	65
33.	Consumo Memoria (kb) Gamma	67
34.	Consumo Tiempo (ms) Gamma	67
35.	Consumo CPU evaluación Gamma	68
36.	Consumo Memoria (kb) Circle Hough	70
37.	Consumo Tiempo (ms) Circle Hough	70
38.	Consumo CPU Circle Hough especular	71
39.	Total hilos creados por técnica	72
40.	Comparación consumo de CPU secuencial / paralelo	74
41.	Comparación consumo de tiempo secuencial / paralelo	74
42.	Comparación consumo de memoria secuencial / paralelo	75

ÍNDICE DE TABLAS

1.	Data set utilizado	50
2.	Hardware utilizado	51
3.	Software utilizado	51
4.	Resumen consumo recursos computacionales shape Predictor	54
5.	Resumen evaluación calidad de imagen PSNR MSE	57
6.	Tabla resumen de evaluación de técnicas por escenarios	60
7.	Resumen evaluación reflejo especular	63
8.	Resumen consumo recursos computacionales Canny	66
9.	Resumen consumo recursos computacionales Gamma	68
10.	Resumen consumo recursos computacionales Circle Hough	69
11.	Resumen hilos creados por técnica	72
12.	Resultados de la evaluación de las técnicas	73
13.	Porcentaje de cambio de las técnicas	76

ÍNDICE DE ALGORITMOS

1.	Carga de Imagen	36
2.	Lectura Detección de Rostro y Landmark	38
3.	Recorte de ojos (cutImage)	40
4.	Filtro mediano hibrido	43
5.	Identificador reflejo (reflexión especular)	45

RESUMEN

La autenticación biométrica en simples palabras es la verificación de identidad de personas mediante características que son únicas en el cuerpo humano, desarrollar este tipo de sistema de autenticación requiere de una gran carga computacional siendo un problema grande al momento del desarrollo. El presente trabajo se enfoca en la detección de contorno de ojos del rostro humano de una forma eficiente y optima mediante lenguajes de programación ligero como lo es C/C++ aplicando técnicas de programación paralela y visión por computadora. La metodología utilizada sigue los pasos de la lectura de imágenes de rostros las cuales pasaran por el modelo pre entrenado de DLIB shapepredictor, que identifica rostros humanos con ayuda de 68 face-landmark, identificando puntos de referencia del rostro, después se hace el recorte de los ojos y se los pasa por un filtro hibrido medio para eliminar el ruido sal y pimienta a continuación, la detección de reflejos especulares para eliminar el brillo de los ojos. Se utilizó un data set de 500 imágenes de rostros y ojos del repositorio CASIAWebFace y dos datasets propios con 117 y 593 imágenes. Se aplica técnicas de programación en paralelo que ayudaron a optimizar recursos computacionales disponibles, ya que al aprovechar de toda la capacidad del dispositivo se puede acortar tiempos de espera entre procesos en un 20 % y el consumo de CPU en un 35 % debido a la manipulación de tareas repetitivas que pueden ser ejecutadas simultáneamente sin afectar a otros procesos con ayuda de la sincronización de la directiva OPM.

Palabras clave: Programación paralela, reflejo especular, ruido sal y pimienta, programación en C/C++, autenticación biométrica.

ABSTRACT

Biometric authentication in simple words is the identity verification of people through characteristics that are unique in the human body, developing this type of authentication system requires a large computational load, being a big problem at the time of development. The present work focuses on the detection of the eye contour of the human face in an efficient and optimal way through light programming languages such as C / C ++ applying parallel programming techniques and computer vision. The methodology used follows the steps of reading face images which will go through the DLIB shapepredictor pre-trained model, which identifies human faces with the help of 68 facelandmarks, identifying reference points on the face, then cutting out the eyes. and runs them through a medium hybrid filter to remove salt-and-pepper noise then specular highlight detection to remove eye glare. A data set of 500 images of faces and eyes from the CASIAWebFace repository and two own datasets with 117 and 593 images were used. Parallel programming techniques are applied that help optimize available computational resources, since by taking advantage of all the capacity of the device, waiting times between processes can be shortened by 20% and CPU consumption by 35% due to manipulation. of repetitive tasks that can be executed simultaneously without affecting other processes with the help of OPM policy synchronization.

Keywords: Parallel programming, mirror reflection, salt and pepper noise, C/C++ programming, biometric authentication.

INTRODUCCIÓN

La biometría es llamada así ya que habla de cualquier tipo de medición y cálculo corporal, por otro lado, la identificación biométrica verifica que «tú eres tú» se basa en las medidas de tu cuerpo. La autenticación biométrica va un paso más allá y usa esa información para compararlo con una base de datos, en donde un sujeto previamente ha ingresado datos sobre el (Paul Cucu, 2018). Con esta información única del cuerpo humano como el iris se puede crear procesos que detectan patrones únicos utilizadas en sistemas de control de acceso, las tecnologías más utilizadas en los sistemas de control de acceso son: reconocimiento de huellas dactilares múltiples con un 38.3 %, autenticación por huella digital única con un 28.4 % y reconocimiento facial con un 11.4 %. Es por esto que en la actualidad se opta por este tipo de tecnología en sistemas de autenticación (Pérez Lescano, 2018).

En la actualidad existen trabajos como el de (HERNÁNDEZ REYES, 2016) que propusieron un trabajo de autenticación biométrica a través de huellas digitales e iris en una empresa industrial. o como el trabajo de (Ismael Farinango, 2021) el cual es un prototipo que permite la autenticación biométrica utilizando el iris del ojo humano. En estos trabajos nombrados anteriormente, sobre todo el de Farinango podemos notar que el proceso de autenticación de iris conlleva tiempos de ejecución muy largos y más en dispositivos móviles, así como problemas de autenticación en entornos reales por diferentes factores como la iluminación y detección del contorno del ojo. Para mejorar el tiempo de ejecución de estos procesos es necesario implementar métodos de paralelo. Se puede definir el procesamiento en paralelo como una representación

eficiente de procesar datos que hace énfasis en la utilización de concurrencia de eventos en el proceso de cómputo.(García López et al., 2004). Puesto que la concurrencia significa paralelismo ya que, el proceso en paralelo es ejecución concurrente de mas de un proceso en el computador y por ende está en rivalidad con el procesamiento secuencial.

Con las investigaciones previamente realizadas a través del estado de arte demuestran que la mayoría de los trabajos evaluaron la eficacia de distintos algoritmos y técnicas utilizados para la autenticación biométrica. Las métricas evaluadas más comunes son el accuracy y el tiempo de ejecución en ordenadores. Por tal razón se propone el desarrollo de un prototipo móvil de autenticación biométrica en el cual se evalúe el consumo de recursos computacionales como: memoria RAM, CPU, tiempo de ejecución y consumo de energía en los dispositivos móviles. El presente trabajo tiene como objetivo implementar un prototipo móvil que ayude a estos trabajos ya nombrados a mejorar sus tiempos de ejecución y además de mejorar el rendimiento del trabajo de (Ismael Farinango, 2021) a mejorar el rendimiento en ambientes reales.

PROBLEMA

La detección facial es la capacidad de determinar si existe o no un rostro en una imagen digital y determinar su posición, así como sus diferentes aspectos (Modi, 2014), con ayuda de la tecnología de visión por computador que consiste en la captura de imágenes digitales 2D que proporcionan información contenida en una matriz $N \times M$, con valores discretos que representan los niveles oscuros y claros de luminosidad (Alegre, 2016). El procesamiento para visión por computadora consume altos recursos computacionales por lo cual la programación en paralelo permite optimizar esos recursos en dispositivos de escritorios como móviles. La programación en paralelo se trata de una técnica que divide el problema en varias tareas que se ejecutan simultáneamente haciendo uso de múltiples recursos computacionales como la memoria empleando

controles para sincronizar procesos de mayor complejidad ofreciendo un balanceo del rendimiento y costo computacional (Chapman, 2017).

A nivel mundial ocurre que la detección de rostros y características específicas del mismo se ha convertido en punto clave para las aplicaciones que requieren analizar puntos relevantes, es el caso de (Storey, 2018), presenta un método que fusiona arquitecturas de aprendizaje profundo, para optimizar la precisión al realizar la detección de rostros y la localización precisa de puntos de referencia, reduciendo los falsos positivos y demostrando que un pequeño aumento en el error de 0,005 % es nocivo al subconjunto de puntos que forman y bordean la cara.

En América Latina sucede que la visión por computador para la detección de características ha motivado trabajos como el del William Castrillon, la cual automatiza el monitoreo emocional del usuario durante evaluación de eficiencia, eficacia y satisfacción, realizando un seguimiento de las expresiones faciales para obtener indicadores que midan la calidad de un servicio (Castrillon, 2008).

En Ecuador específicamente en la UPS, se ha visto el caso del trabajo de Farinango y Sandoval, el cual es un prototipo que permite la autenticación biométrica utilizando el iris del ojo humano.

Explican que el proyecto presenta problemas de aceptación:

“Las técnicas utilizadas en el presente trabajo proporcionaron buen rendimiento al trabajar con imágenes tratadas como las de base de datos CASIA obteniendo un accuracy de 98.15 % y Ubi-iris (sesión 1) 97.08 %. Sin embargo, cuando procesa imágenes capturadas en ambientes reales y con diferentes dispositivos tiende a bajar los resultados obteniéndose un accuracy de 72.92 %, como consecuencia de la cantidad de ruido, reflejos o la calidad de imagen.” (Farinango, 2021)

El presente proyecto de investigación responderá a la siguiente problemática: ¿En qué medida contribuye el desarrollo de un prototipo móvil para identificar contorno de ojos en imágenes de rostros humanos utilizando programación en paralelo para un sistema de autenticación biomé-

trica?, ya que el trabajo de (Farinango, 2021) consta de cuatro fases. i) localiza el iris en una imagen, ii) normaliza la imagen del iris, iii) extrae características para generar un patrón único y iv) clasifica el contenido comparándolas con diversas plantillas. Proporciona información estadística que mide la aceptación del método utilizado, evidenciando que el reconocimiento facial combinado con el reconocimiento de iris ayudaría a obtener la mínima tasa de error, así como es necesario encontrar métodos para tratar problemas de iluminación y reflejo que se presentan en imágenes que se capturan con dispositivos móviles.

Además, generar procesos en paralelo para medir el rendimiento de las técnicas utilizadas proporciona información estadística que mide la aceptación del método; Evidenciando que el reconocimiento facial combinado con el reconocimiento de iris ayudaría a obtener la mínima tasa de error.

Por tal motivo es necesario encontrar métodos para tratar problemas de iluminación y reflejo que se presentan en imágenes que se capturan con dispositivos móviles, además generar procesos en paralelo para medir el rendimiento de las técnicas utilizadas.

Entre las diferentes causas que pueden estar originando este problema, hemos detectado las siguientes: i) iluminación y reflejo, ii) desenfoque de la imagen, iii) alto cálculo computacional, iv) dimensiones de la imagen inapropiadas, v) ruido en la imagen.

Sin embargo, se consideran las tres primeras causas a solucionarse en el proyecto, ya que la calidad de una imagen se condiciona a la iluminación del ambiente en la cual se realiza la captura; Mientras que el desenfoque produce que la imagen sea borrosa las cuales carecen de nitidez presentado como resultado una mezcla de colores que genera la pérdida de información para el análisis del contenido.

Por último, el alto consumo computacional en dispositivos móviles puesto que al realizar varias tareas secuenciales el tiempo de respuesta aumenta.

Las causas antes mencionadas pueden traer diversos efectos, entre los que hemos destacado están: i) baja calidad de la imagen, ii) imágenes difusas.

Donde la mala calidad en la imagen provoca que el análisis de esta no permita obtener precisamente todas las características y elementos que se encuentran distribuidos.

Por otro lado, están las imágenes difusas las cuales carecen nitidez presentado como resultado una mezcla de colores que simulan la vista a través de una superficie trasparente rugosa.

Es por lo que resulta importante aportar con el desarrollo de un prototipo móvil en lenguaje (C/C++), para extraer contorno de ojos en imágenes de rostros humanos utilizando programación en paralelo.

ANTECEDENTES

(Storey et al., 2018) El propósito general de este estudio es integrar para la tarea general de detectar el rostro y localizar el punto de referencia de las imágenes "In The Wild" para evaluar la efectividad del método propuesto en esta casa (IDM). Diseño de modelo de profundidad (IDM). lugares de trabajo .. Reconocimiento facial. Ofrece varias evaluaciones intermedias y de referencia. El método utilizado consiste en probar y diseñar la integración de dos arquitecturas modernas: la arquitectura de red R-CNN definida individualmente más rápida y una red cara a cara basada en reloj de arena. Utilizaron tres conjuntos de pruebas de reconocimiento facial para una evaluación justa. Es una base de datos de caracteres faciales registrados en el medio natural. Los resultados obtenidos son los siguientes: El rendimiento de FD-CCN es 92,6 mientras que S3FD tiene un 98,2, el método conjunto se encuentra entre el 94,3%. Donde el método sobresale. Concluyendo que el método es comparable a otros métodos de detección de rostros de alto rendimiento.

(Alcántara-Montiel et al., 2019) El objetivo principal de la investigación es desarrollar un detec-

tor de rostros eficiente basado en el descriptor Histogram of Gradient (HOG) y el clasificador en cascada. La metodología utilizada fue cuantitativa, y se empleó un ejercicio de clasificación mediante dos conjuntos de imágenes (positiva y negativa) en proporción 1:3. Para obtener el resultado, se validó el método con diferentes conjuntos de imágenes. Como resultado, obtiene un alto porcentaje de reconocimiento facial del 85 % o más. Concluimos que el reconocimiento facial es en posiciones horizontal, vertical y vertical con fluctuaciones de 30 grados o más con respecto a la vista frontal. Puede afectar el nivel de aceptación y reconocimiento.

(Aldape-Pérez et al., 2013)Esta función demuestra el uso del paradigma de memoria asociativa y la computadora paralela para realizar la función de selección de imágenes. Este enfoque utiliza las redes sociales para capturar el valor de una portada que expone información relevante o redundante para los fines. El rendimiento del algoritmo propuesto se mejora al comparar la precisión predecible encontrada en este modelo con el rendimiento obtenido por otros algoritmos conocidos en la literatura actual. Como resultado, el método de búsqueda se distribuye a una gran cantidad de fabricantes, aumentando el tiempo de ejecución; Este resultado es contrario a lo esperado, ya que si la función de búsqueda se comparte entre un gran número de fabricantes, el tiempo de ejecución debe reducirse. Este comportamiento no deseado ocurre porque el número de configuraciones en la base de datos de Haberman es demasiado pequeño para ser procesado por ocho procesadores simultáneamente; El aumento del tiempo de espera (108,63 milisegundos) se debe a la información pasiva que pasa entre el nodo principal y el nodo de procesamiento. Los resultados experimentales han demostrado que la memoria interoperable se puede realizar en la simulación de estructuras informáticas, lo que reduce el costo de la computadora necesaria para obtener el conjunto perfecto de características que maximizan el rendimiento de los componentes.

(Garcés Núñez, 2017) El proyecto exige la introducción de un sistema de identificación presencial que tiene la ilusión de apoyar a ECU-911 y sensibilizar al público sobre la lista de los más buscados. El método tiene una alta calidad; Es importante señalar que la institución puede lograr desafíos basados en desafíos para sustentar el desarrollo de ingeniería y tecnología. Estadística porque las ideas, teorías y principios científicos se utilizan en el diseño de sistemas, el desarrollo de estándares y cambios que enfatizan la resolución de problemas y los procesos, pruebas y prototipos del proyecto. Como resultado, en una serie de 10 personas no identificadas en la operación, se tomaron 20 fotos de personas buscadas y los criterios fueron: 17 identificados, 3 desconocidos, 10 no buscados. El prototipo completo proporcionó un alto porcentaje de desempeño debido al equipo básico.

(Briones Gárate, 2020) El proyecto tiene como objetivo crear una red neuronal que pueda aprender a reconocer a cada persona utilizando un único patrón de entrenamiento. La metodología de la cognición biométrica consta de cuatro fases: detección, procesamiento, extracción de características, comparación y clasificación, base de datos. El resultado muestra que el 72 % de los que utilizan el reconocimiento aprueban. En definitiva, el uso de técnicas como PCA y cualquiera de sus variantes, CPCA o KPCA, está ampliamente aceptado en el procesamiento visual de rostros humanos.

(Chalén Pacay & Camacho Girón, 2020) Su estudio tiene como objetivo crear un prototipo de aprendizaje automático basado en inteligencia artificial (Machine Learning), que permita integrar a los conductores de una cooperativa de transporte provincial su estado de somnolencia. El método utilizado tiene un método mixto y completo; Los artículos primero y segundo se utilizarán para obtener información fáctica relevante para el proyecto de investigación, tanto teórica

como práctica, de ahí que se plantee que la investigación tendrá investigación e interpretación. Los resultados muestran que el programa en cada fase tuvo un 2% de éxito. Como resultado, se debe principalmente a la cantidad de luz que emite por la mañana, que distorsiona o degrada la imagen cuando se quita la cámara, donde esta distorsión no se producirá si cambias la cámara seleccionada a la cámara infrarroja. sucedió. debido al diseño de la mejor cámara infrarroja.

ALCANCE

Esta sección está organizada en base a los objetivos específicos previamente definidos, donde se describe en detalle las características del prototipo móvil. Como primer paso tenemos el identificar métodos de programación en paralelo para optimizar recursos computacionales móviles. Se identificará métodos de programación paralela en lenguaje C/C++ óptimos para mejorar el consumo de recursos computacionales al realizar tareas simultaneas en dispositivos móviles; Con la finalidad de paralelizar el algoritmo disminuyendo el tiempo de procesamiento mediante la distribución de tareas entre los procesadores y memoria disponibles. De igual manera recopilar estudios de técnicas de reconocimiento facial y extracción sector de los ojos. Se recopilará estudios de técnicas de reconocimiento facial y extracción del sector de los ojos o características en una imagen 2D, para conseguir mejor detalle del rostro humano con el objetivo de presentar claramente los sectores a destacar. Posteriormente construir prototipo para dispositivos móviles de reconocimiento facial y extracción del sector de los ojos con programación en paralelo. Se construirá un prototipo para dispositivos móviles de reconocimiento facial y extracción del sector de los ojos con programación paralela en lenguaje C/C++, el cual permitirá la mejora de un módulo móvil experimental de autenticación biométrica. Como siguiente evaluar los datos de reconocimiento facial y extracción ojos en 2D. Se evaluará los datos obtenidos al

realizar el reconocimiento facial y extracción del sector de los ojos en imágenes 2D para tasar el diseño del modelo empleado con el fin de corroborar si el método empleado optimiza el coste computacional. Finalmente se evaluará el módulo móvil y recursos computacionales aplicando métodos de programación en paralelo, los cuales proporcionarán datos que permitirán establecer el coste computacional que genera el módulo a desarrollar que tiene como finalidad apoyar al reconocimiento biométrico.

OBJETIVOS

Objetivo General

Desarrollar un módulo prototipo móvil que permita identificar contorno de ojos en imágenes de rostro permitiendo mejorar el rendimiento utilizando programación en paralelo para un sistema de autenticación biométrica, haciendo uso de tecnologías y herramientas Open Source.

Objetivos Específicos

- Identificar métodos de programación en paralelo para optimizar recursos computacionales móviles.
- Recopilar estudios de técnicas de reconocimiento facial y extracción sector de los ojos.
- Construir prototipo para dispositivos móviles de reconocimiento facial y extracción del sector de los ojos con programación en paralelo.
- Evaluar los datos de reconocimiento facial y extracción ojos en 2D.
- Evaluar el módulo móvil y recursos computacionales en paralelo utilizando métricas.

FUNDAMENTOS TEÓRICOS

1.1 VISIÓN POR COMPUTADORA

El termino visión por computador ha sido muy sonado en los últimos años, por todas las aplicaciones que tiene en el mundo, puesto que es un grupo de métodos y herramientas por el cual se produce una imagen modificada de una original. Este conjunto de herramientas permite obtener información de las imágenes procesadas, ya sea una medición, interpolación o decisión. La principal función de la visión por computador es la detección de objetos ya que, cuando se emplea estos métodos se puede relacionar los patrones ya aprendidos y, de esta forma, identificar los objetos y diferenciarlos (Mery, 2004).

1.1.1 *Tratamiento de imagenes*

El tratamiento de imágenes es una técnica usada en la visión por computadora que permite corregir y tratar defectos mejorando así el resultado de final de la imagen. Al momento de tratar imágenes que exhiben relación con el ruido se necesita aplicar filtros, por lo general los filtros, los que permiten mejorar diferentes aspectos dependiendo del filtro aplicado (Buemi, 2012).

1.1.2 *Ruido*

Se caracteriza como información que perturba la imagen, este fenómeno se puede dar por varios factores como la transmisión, traducción o recepción de la señal, ya que el paso de infor-

mación de una imagen se debe adaptar al canal por el cual se lo envía esto da como resultado la degradación de la señal.

se contemplan varios modelos de ruido para el tratamiento de imágenes digitales.

- **Ruido Gaussiano o ruido normal** Se da por consecuencia de los equipos electrónicos como son los sensores que no tienen suficiente iluminación para la captura de la imagen o afectados por la temperatura a los que son expuestos, se representa por la función de densidad.
- **Ruido impulsivo o sal y pimienta** Aparece en la transmisión por alteraciones en la señal que produce una imagen.

1.1.3 Filtro

El filtro de imágenes es en pocas palabras es la operación de convolución entre la imagen y la función filtro permitiendo sustituir las convoluciones por productos, aumentando las facilidades de cálculo en el proceso y aumenta la flexibilidad de los intervalos que se requieren ser eliminados (Pinilla et al., 1997).

Para el preprocesamiento de imágenes digitales es necesario aplicar diferentes filtros trabajando directamente en los pixeles de una imagen, estos filtros tienen su propio objetivo como son:

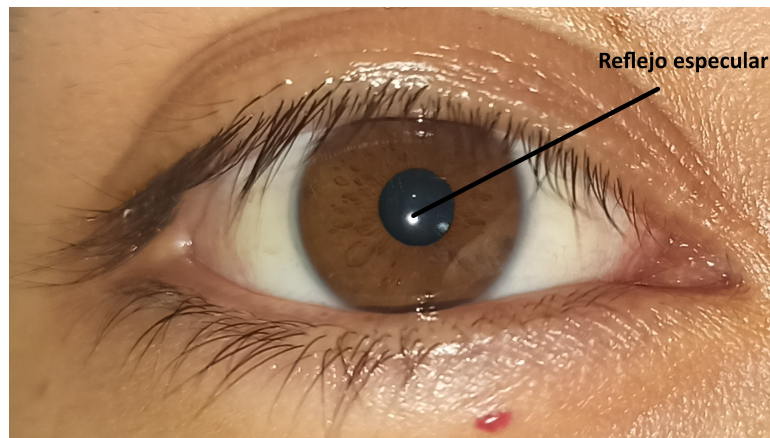
- **Eliminación el ruido** Modifica el valor de intensidad de los pixeles adaptándolos a sus vecinos.
- **Detección de bordes** Analiza y detecta aquellos pixeles en los cuales la diferencia de intensidad es extrema.
- **Suavización de imagen** Disminuye la variación de intensidad de los pixeles apartándolos a los valores de sus vecinos.

1.1.4 Problemas de iluminación

Se ha hablado anteriormente sobre el tratamiento de imágenes y diferentes filtros los cuales se puede aplicar para mejorar las características de las imágenes, a hora debemos hablar sobre las presencias de fuentes de iluminación no controladas al momento de adquirir imágenes, esto puede generar reflexiones especulares o brillos dentro del iris. Este efecto puede reducir la información contenida en el iris. Los reflejos especulares son artefactos que aparecen usualmente como los pixeles más brillantes de la imagen (Valencia-Murillo et al., 2014). La apariencia vidriosa de la córnea, la cual recubre al ojo por completo, es la que causa que la luz se refleje en ella y por ello aparecen los reflejos especulares como se puede apreciar en la Figura 1.

Figura 1

Ojo con presencia de reflejo especular.



Nota: Representación de reflejo especular en pupila del ojo. Elaborado por: Los autores.

1.2 PREPROCESAMIENTO DE IMÁGENES

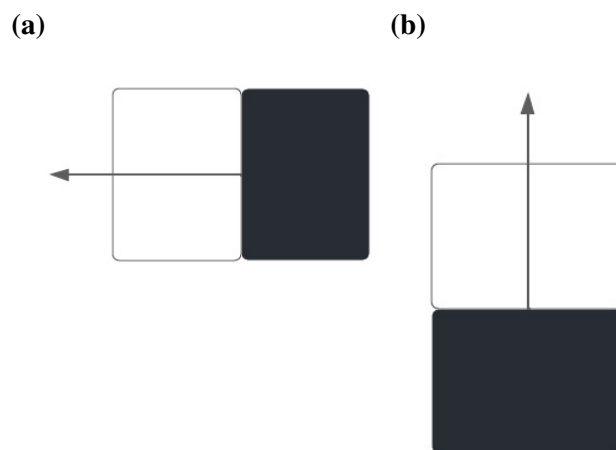
El preprocesamiento de imágenes es una fase crítica para realizar el trabajo que se propone en el proyecto actual a continuación se describe las técnicas a ser usadas:

1.2.1 *Histograma de gradientes orientados HOG*

El descriptor están basados en la orientación de los gradientes que tiene el área de un bloque en una imagen, por ello se define como el cambio de intensidad a la dirección con valores más altos, se establece la dirección del cambio de intensidad y la magnitud de la dirección como muestra la Figura 2a, en la cual se puede observar que la dirección de su modulo si dirige hacia la izquierda, la figura 2b muestra que el cambio de intensidad esta de manera vertical.

Figura 2

Representación de gradiente



Nota: Representación de la dirección de un gradiente por el cambio de intensidad. Elaborado por: Los autores.

Para calcular diferencia de intensidad horizontal y vertical que existe entre los pixeles vecinos como muestra la Figura 3.

Se establece dos ecuaciones generales que permiten definir la dirección del gradiente la

Figura 3

Cambio de intensidad horizontal vertical.

255	255	255	0	0
255	255	255	0	0
255	255	0	0	0
0	0	0	0	0
0	0	0	0	0

Nota: Cambio de intensidad en los pixeles vertical y horizontal. Elaborado por: Los autores.

primera 1.1 es la diferencia horizontal y la segunda 1.2 para el cálculo vertical.

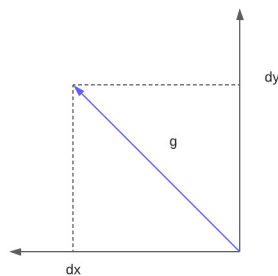
$$dx = M(x - 1, y) - M(x + 1, y) \quad (1.1)$$

$$dy = M(x, y + 1) - M(x, y - 1) \quad (1.2)$$

Como resultante es posible calcular la dirección y magnitud del gradiente como se representa en la Figura 4.

Figura 4

Vector resultante del cálculo de gradiente.



Nota: Vector resultante que corresponde al cambio de intensidad de un segmento de pixeles. Elaborado por: Los autores.

1.2.2 Canny

fue propuesto en 1986, se trata de un método de detección de bordes en una imagen de dos dimensiones, el algoritmo consta de tres fases que son:

- **Cálculo del gradiente** El primer paso es aplicar el filtro gaussiano para suavizar la imagen eliminando el ruido existente para ello utiliza las ecuaciones 1.3 y 1.4 para calcular la magnitud y dirección del gradiente en cada pixel.

$$|G| = \sqrt{G_x^2 + G_y^2} = |G_x| + |G_y| \quad (1.3)$$

$$\phi = \tan^{-1} \frac{G_y}{G_x} \quad (1.4)$$

Obteniendo un par de imágenes con los valores del gradiente y la dirección.

- **Resultado del gradiente** El par de imágenes que se generan al aplicar las ecuaciones antes mencionadas se combinan para formar una imagen con bordes adelgazados.
- **Histéresis umbral a la suspensión no máxima**

La combinación de las imágenes produce máximos locales, los cuales se pueden solucionar eliminando el ruido realizando una serie de pasos que se explican a continuación: i) De la imagen resultante se toma la orientación de los bordes y dos umbrales ii) Posterior por cada punto se toma el borde no explorado que es mayor al segundo umbral conectando las direcciones a la normal del borde.

1.2.3 *Circulo Hough*

Su principal función es localizar círculos en imágenes, como resultado se obtiene el radio y punto central (radio, a, b) estableciendo el valor mínimo y máximo del círculo que lo contiene la ecuación 1.5 define el proceso que realiza.

$$x_c^2 + y_c^2 - r_c^2 = 0 \quad (1.5)$$

Para resaltar el borde del círculo se aplica las fórmulas 1.6 y 1.7 que se representan a continuación.

$$x = a + R \cos \phi \quad (1.6)$$

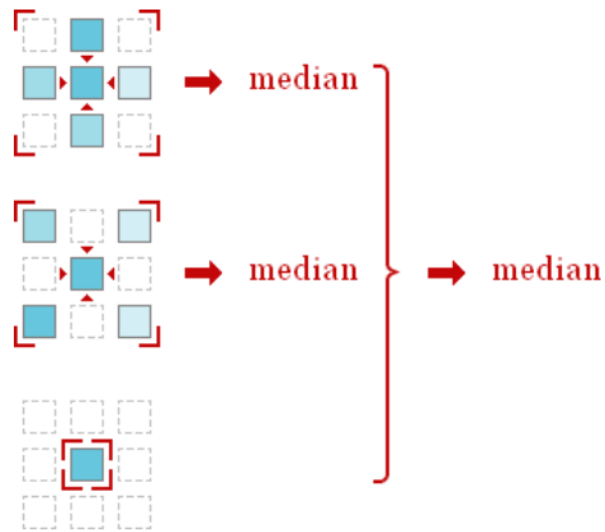
$$y = b + R \sin \phi \quad (1.7)$$

1.2.4 *Filtro mediano híbrido*

El filtro de mediana híbrido es un filtro no lineal, que elimina de manera fácil el ruido de impulso mientras que conserva los bordes (Michael Mukovoz Studio, 15 de Abril de 2018). Comparándolo con la versión normal el filtro mediano, tiene mejores características de preservación de las esquinas. La idea es aplicar para cualquier elemento de señal la técnica de la mediana varias veces variando la forma de la ventana y luego tomar la mediana de los valores medianos obtenidos como muestra la Figura 5.

Figura 5

Flujo de trabajo del filtro mediano híbrido.



Nota: El filtro mediano híbrido no suaviza la imagen en exceso permitiendo conservar los detalles de los bordes de la imagen. Fuente: (Sarmiento-Ramos, 2020).

Haciendo uso de los vecinos y calculando el valor mediano de x , y sumándolo al valor mediano de los vecinos como se evidencia en la Figura 5, se aplica la técnica varias veces variando el tamaño de la ventana para posterior clasificarlos calculando tres valores medianos que son:

- Mediana de los valores horizontales y verticales (MR).
- Mediana de los valores diagonales (MD).
- Mediana del valor central (C).

Para todos los filtros de ventana el problema. Es el tratamiento de bordes. Si se coloca una ventana sobre un elemento en el borde, una parte de la ventana estará vacía. Para llenar el vacío, la señal debe extenderse. Para el filtro mediano híbrido, es una buena idea extender la imagen. En otras palabras, se agregan líneas en la parte superior e inferior de la imagen y columnas a la izquierda y a la derecha de la misma. Entonces, antes de pasar la señal a nuestra función de filtro de mediana, la señal debe extenderse. Anotemos el envoltorio, que hace todos los preparativos.

1.2.5 Filtro gaussiano

El filtro gaussiano ayuda suavizando la imagen de una forma lineal para quitar el ruido gaussiano, por esa razón este filtro es usado para la eliminación de ruido del tratamiento de imágenes. Este filtro es el resultado del promedio ponderado de toda una imagen, obteniendo el valor de cada pixel con el valor del promedio ponderado de si mismo y el de los pixeles vecinos. El proceso que se realiza al usar este filtro es usar una plantilla o mascara para escanear cada pixel en la imagen y usar la escala de grises promedio ponderada de los píxeles en la vecindad determinada por la plantilla para reemplazar el valor del píxel central de la plantilla (Ortiz Rangel et al., 2017).

1.3 DLIB

Dlib es un conjunto de herramientas modernas de C++ que contiene algoritmos de aprendizaje automático y procesamiento de imagen para resolver problemas del mundo real. Se utiliza tanto en la industria tanto como en el mundo académico, se la utiliza en una gran variedad de dominios como la robótica, los dispositivos móviles, dispositivos integrados y varios entornos informáticos. La licencia de código abierto de Dlib le permite usarlo en cualquier aplicación, sin cargo (King, 2009).

1.3.1 *shape predictor 68 face landmarks*

Dlib posee varios modelos entrenados que permiten realizar diferentes tareas de reconocimiento, el que hablaremos a continuación es el shape predictor 68 face landmarks este modelo permite encontrar rostros humanos frontales en una imagen y estimar su pose. La pose toma la forma de 68 puntos de referencia como muestra la figura 6. Estos son puntos en la cara como las comisuras de la boca, a lo largo de las cejas, y en los ojos (King, 2009).

Este modelo es que nos permitirá identificar los ojos con gran precisión puesto que el detector de rostros está hecho con la característica clásica Histograma de gradientes orientados (HOG) combinada con un clasificador lineal, una pirámide de imágenes y un esquema de detección de ventana deslizante.

Su principal objetivo es describir las características de un objeto, para la detección de rostros el descriptor usa una ventana deslizante que se desplaza alrededor de la imagen calculando un descriptor evaluando un conjunto de datos normalizado miden el cambio entre dos puntos. El histograma de gradientes orientados se basa en el aparente de que toda imagen tendrá los mismos cambios en su iluminación sin importar si ésta es clara u oscura, también, tiene un funcionamiento eficiente cuando se evalúan imágenes con cambio de escala; gracias a que otorga una representación de los gradientes dentro de la imagen y la dirección de estos. (Romero et al., 2019) Para determinar el gradiente dlib con una ventana de 8x8 hace uso de las ecuaciones :

$$G_x(x,y) = [-1, 0, 1] * [I(x-1,y), I(x,y), I(x+1,y)]^T \quad (1.8)$$

$$G_y(x,y) = [-1, 0, 1] * [I(x,y-1), I(x,y), I(x,y+1)]^T \quad (1.9)$$

La magnitud esta definida por la ecuación :

$$G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \quad (1.10)$$

Por ultimo la dirección tiene la siguiente ecuación :

$$\phi(x,y) = \arctan \frac{G_y(x,y)}{G_x(x,y)} \quad (1.11)$$

Para profundizar más el modelo se debe mencionar el trabajo de (Kostiantyn S. Khabarlak, 2015), ya que en este trabajo es donde se explica el proceso y metodología para obtener el shape predictor 68 face landmarks. El artículo trata el problema de la alineación de caras para una sola imagen.

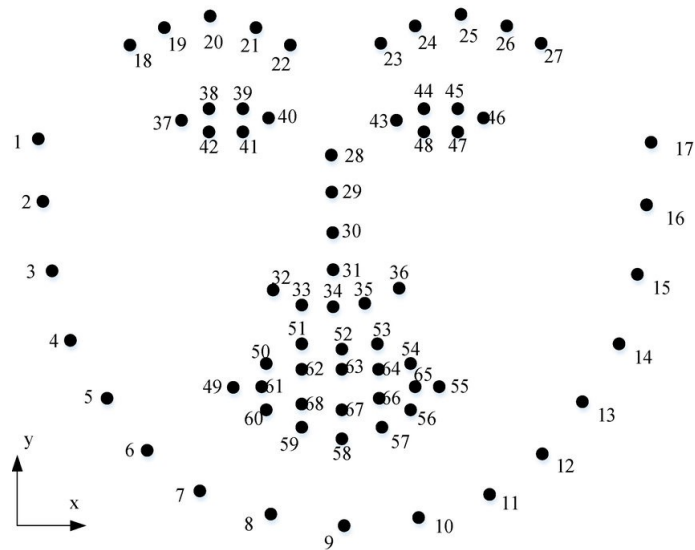
Muestra cómo se puede usar un conjunto de árboles de regresión para estimar puntos de referencia de la cara además que presenta un marco general basado en el aumento de gradiente para aprender un conjunto de árboles de regresión que optimiza la suma de la pérdida de error cuadrática y maneja de forma natural los datos faltantes o parcialmente etiquetados. (King, 2009)

De esta forma se muestra el uso de mejores puntos que activan la estructura de los datos de la imagen lo cual ayuda con la selección eficiente de funciones. Además, se investigan diferentes estrategias de regularización y su importancia para combatir el sobreajuste. También analiza el efecto de la cantidad de datos de entrenamiento sobre la precisión de las predicciones y exploramos el efecto del aumento de datos utilizando datos sintetizados.

El resultado final del proceso y uso de las ecuaciones empleadas en la creación del Face-Landmark, son los puntos de referencia faciales los cuales retornan las posiciones de cada punto del rostro, estos puntos se mantienen en cualquier rostro que pase por el estimador de puntos de referencia como se muestra en la Figura 6.

Figura 6

Face landmark detection.



Nota: Representación de los 68 puntos de referencia. Fuente: (King, 2009).

1.4 COMPUTACIÓN PARALELA

La computación paralela es una forma de cómputo en la que se hace uso de 2 o más procesadores para resolver una tarea. La técnica se basa en el principio según el cual, algunas tareas se pueden dividir en partes más pequeñas que pueden ser resueltas simultáneamente (Aguilar et al., 2004).

1.4.1 Programación paralelo

Durante la última década, surgieron procesadores multinúcleo, por lo que la programación en paralelismo multsimétrico (SMP) resurgió casi una década después, en la que los clúster OpenMPI están en todo lado, y su rendimiento es alto. Actualmente disponemos de 2, 4 y 6 núcleos en el mercado. Los operadores de la celda Be tienen un cabezal de núcleo, Power Processor Element (PPE) y ocho núcleos llamados elemento de procesador sinérgico Synergistic

Processor Element (SPE), con 48 a 448 tipos de cabezales únicos para la comparación cuantitativa. En este proyecto, aprovechamos nuestra experiencia para interactuar con estos sectores, discutiendo oportunidades futuras para implementar software y sistemas digitales de bajo costo (Aslla et al., 2013).

En las primeras etapas del procesamiento informático paralelo, una de las soluciones más comunes para implementar el ensamblaje paralelo fue extender el lenguaje serial con una colección de constructores, a menudo llamados forall. Aunque la mayoría de ellos tienen la misma sintaxis, son diferentes a su manera, semántica y uso. Entre los programas más populares está lo que ofrece Fortran de alto rendimiento (HPF) y OpenMP. Todas las iteraciones se extiendan al lenguaje C. Para todas las iteraciones, el modelo de programación se define como:

- Activar y utilizar diferentes niveles en paralelo:
- Paralelismo anidado.
- Expanda la plantilla de mensaje.
- Le permite asociar el modelo de conexión.

Por tanto, este modelo ofrece por un lado la posibilidad de tendido para todos los operadores y generalizar el patrón de programación de mensajes, por otro lado, te permite analizar y predecir los efectos en función del nivel de complejidad asociado (Fernández et al., 2016).

1.4.2 Paradigma de programación en paralelo

Para analizar el paradigma de programación en paralelo se debe entender los problemas de las actividades secuenciales las cuales tienen como objetivo llevar a cabo n tareas con n recursos disponibles. Las actividades están restringidas a la procedencia es decir que cada proceso no puede iniciar hasta que las actividades procedentes hayan terminado y la restricción de re-

cursos, los cuales son limitados ya que si una actividad hace uso de un recurso dado la siguiente actividad no puede hacer uso hasta que se libere el recurso.

Para resolver estos problemas y aprovechar los recursos se aplica una serie de técnicas:

- **Comunicación:** determina la forma con la cual se realiza la comunicación entre diversos procesos computacionales, para informar cuando un proceso termino o está solicitando algún servicio que debe ejecutarse.
- **Descomposición:** divide una tarea para ser ejecutada en procesadores separados, cada uno ejecuta su tarea y la finaliza dando mayor rendimiento.
- **Sincronización:** Permite determinar el orden que debe tomar un proceso a ser ejecutado es decir si algún conjunto de subprocesos debe esperar mientras otro conjunto trabaja.

1.4.3 Problemas de programación en paralelo

Según la teoría la mayoría de los algoritmos pueden tener un buen performance ya que se tiene un control al ser ejecutados en ambientes de prueba, pero no refleja la realidad del sistema, es así que bajo ambientes reales el rendimiento tiende a bajar al presentar sobrecargas esto hace que sufran problemas que limitan el correcto funcionamiento de un algoritmo paralelizado, las principales dificultades que pueden presentar son:

- **Contención de memoria:** el procesador sufre demoras mientras se intenta acceder a un espacio de memoria, esto puede ser debido a que una variedad de procesadores comparte datos globales.
- **Sectores de código secuencial excesivos:** no todo el código puede ser paralelizado así que por lo general existen segmentos de código secuenciales para llevar a cabo tareas

que deben ejecutarse una sola vez. **La ley de Amdahl** indica que los sectores de código secuencial afectan el speed-up, la representación matemática es la siguiente:

$$Speed-up_{MAX} = \frac{1}{f + \frac{1-f}{p}} \quad (1.12)$$

Donde f : división de operaciones a ser ejecutadas secuencialmente. Y p .: número de procesadores.

- **Creación de procesos:** en ambientes reales, la creación de procesos en paralelo toma una cierta cantidad de tiempo para ser ejecutados, es decir que si el tiempo de creación de estos procesos supera el tiempo de ejecución de ese proceso no se optimiza el algoritmo.
- **Comunicaciones tardías:** este problema se presenta cuando se trabaja con multicomputadoras, ya que los procesadores se comunican a través de mensajes.
- **Falla en la sincronización:** la sincronización de procesos fuerza un proceso a esperar a otros para continuar con su ejecución, esto puede causar un cuello de botella como consecuencia el speed-up general aumenta.
- **Inestabilidad de carga:** Las tareas pueden ser creadas y asignadas dinámicamente a los procesadores disponibles esto tiene como consecuencia que algunos procesadores se encuentren sobrecargados de actividades que pueden manejar mientras que otros no ejecuten ninguna tarea.

1.4.4 Granularidad de problemas

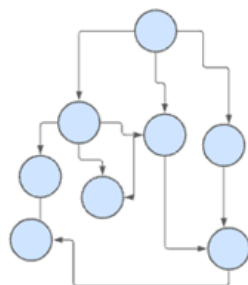
Para tener una clasificación de problemas con programas en paralelo se toma su grado de paralización al cual son sometidos, determina que tan dependientes son otros subprocesos o resultados, si estos tienen una gran dependencia se dice que son de granularidad fina como

muestra en la Figura 7a, ya que poseen un alto grado de comunicación y sincronización en las tareas ejecutadas. Por otra parte, está la granularidad gruesa que contiene menos comunicación entre actividades ejecutadas al igual que menores sectores paralelizados como se puede apreciar en la Figura 8b.

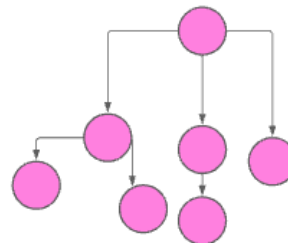
Figura 7

Tipos de granularidad

(a) *Granularidad fina*



(b) *Granularidad gruesa*



Nota: Representación de granularidad fina y gruesa. Elaborado por: Los autores.

1.4.5 Formas de paralelismo

1.4.5.1 Datos Paralelizados y Funciones paralelizadas. Los datos se dividen en bloques para que cada subproceso ejecute una tarea con un segmento de dato que se le fue asignado como muestra en la Figura 8a.

La tarea se dividen actividades individuales como se aprecia en la Figura 8a, es así como cada actividad recibe un subconjunto de datos para ejecutar el proceso que se le asigna.

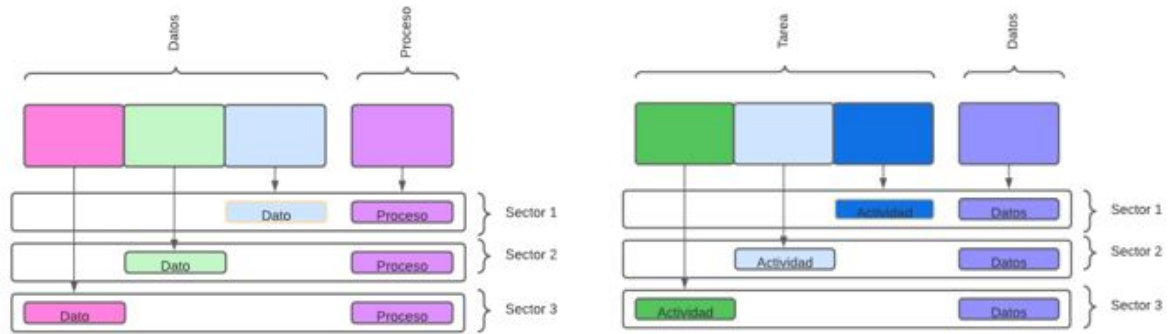
1.4.5.2 Segmentación. Los cálculos se dividen en etapas que trabajan simultáneamente sin importar el origen de los datos, limitando el nivel de paralelismo y aumentando el consumo de recursos, ya que si un proceso necesita un dato resultante de una tarea que se ejecutó al mismo tiempo el proceso espera la respuesta antes de iniciar su tarea como se muestra

Figura 8

Tipos de granularidad

(a) datos paralelizados

(b) funciones paralelizadas

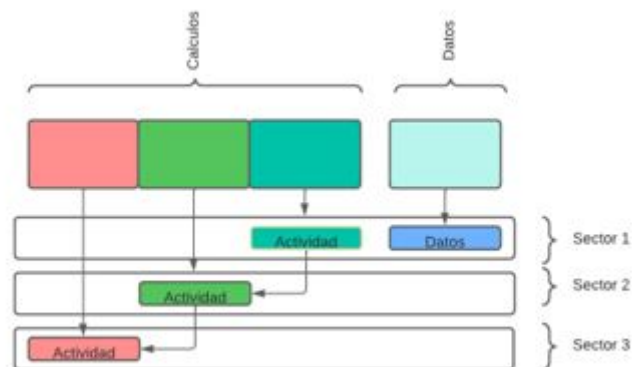


Nota: (a) Diagrama de datos paralelizados, (b) Diagrama de funciones paralelizadas .
Elaborado por: Los autores.

en la Figura 9.

Figura 9

Segmentación múltiples instrucciones



Nota: Técnica de segmentación de instrucciones. Elaborado por: Los autores.

1.5 OPENMP

OpenMP es un modelo de memoria compartida para multiprocesadores, ya que cuenta con esta característica se está convirtiendo en un estándar para paralelizar aplicaciones, en la actualidad organizaciones como Hewlett Packard, Inter, IBM, SUM entre otras hacen uso de las características que brinda OpenMP, dando así el apoyo al estándar para la aplicación en la industria. Proporciona a los desarrolladores una plataforma simple para crear aplicaciones paralelas, siguiendo un modelo fork join, configurando y ejecutando bifurcaciones que se ejecutan en segmentos definidos del programa para posterior unirse y continuar con la ejecución secuencial. La ejecución de los segmentos de código paralelizado depende del hardware disponible es así que dividir un proceso en varios subprocesos no siempre será la mejor opción para generar ganancia ya que esto puede saturar el procesador haciendo que el rendimiento baje (OpenMP, 2013).

1.5.1 Directivas OpenMP

1.5.1.1 Parallel. La directiva parallel inicia un segmento de código paralelizado creando grupos de hilos (Threads), se puede especificar haciendo uso de la cláusula omp_set_num_threads(Número de hilos), dando como resultado una bifurcación de tareas livianas que se distribuyen en los procesadores disponibles permitiendo la paralelización incremental como se muestra en la Figura

1.5.1.2 shared. Declara que todas las variables que se encuentran dentro de la expresión shared se compartirán entre todos los subprocesos permitiendo que las regiones paralelizadas puedan acceder al espacio de memoria que almacena la variable.

1.5.1.3 reduction. Permite que las tareas paralelizadas tengan una copia de la variable por hilo en la que pueden trabajar una operación, al finalizar la región paralela los resultados se combinan generando una variable global.

1.5.1.4 Critical. Esta directiva identifica una sección de código que se ejecutara una sola vez por hilo, como representa la Figura 10 espera el inicio para la ejecución de una región paralela.

1.6 CALIDAD DE IMAGEN

1.6.1 MSE

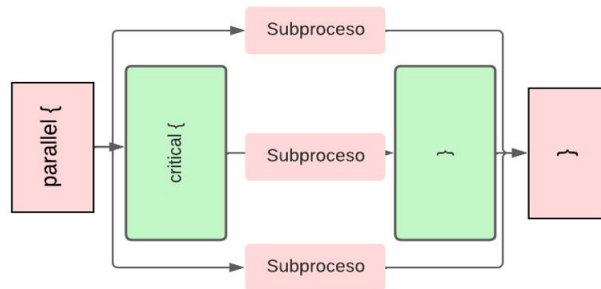
Mide el error cuadrático medio y se calcula con la siguiente formula:

$$MSE = \frac{1}{H \times W} \sum_{i=1}^H \sum_{j=1}^W (X(i, j) - Y(i, j))^2 \quad (1.13)$$

Un valor bajo de mse, representa menos error en la señal nueva respecto a la señal original.

Figura 10

Directiva Critical



Nota: Región crítica ejecutada una sola vez por cada subproceso. Elaborado por: Los autores.

1.6.2 Relación pico señal ruido (PSNR)

Evaluación de la calidad de la imagen que es sensible al error después de realizar alguna acción sobre la imagen original la imagen resultante tiende a presentar variación en la señal que relaciona el ruido para el cálculo de relación pico señal ruido se aprovecha la ecuación 1.14, que identifica las características de la imagen que el ojo humano no puede diferenciar, para el cálculo de hace uso del error cuadrático medio MSE que tiene al imagen original y la imagen resultante la cual se representa por la ecuación 1.13. (?, ?).

$$PSNR = 10 \log_{10} \left(\frac{(2^n)^2}{MSE} \right) \quad (1.14)$$

1.7 EVALUACIÓN DE RENDIMIENTO

Medir el desempeño es vital para establecer el estado de una aplicación por tal motivo el presente proyecto define indicadores que miden el consumo de recursos computacionales.

1.7.1 Tiempo de ejecución

Mide el tiempo que le toma a un proceso en ejecutar las tareas que pueden ser secuenciales o paralelizadas, este tiempo dependerá de la composición del hardware, los recursos que tenga disponibles como la capacidad de memoria RAM o CPU, por otra parte, el código que conforma la aplicación influye en la capacidad de reacción por tal motivo debe estar correctamente estructurado para evitar tiempos largos de procesamiento.

1.7.2 Memoria RAM (memoria de acceso aleatoria)

La principal característica es la capacidad de almacenar temporalmente datos que un programa en ejecución genera o hace uso, ya que se trata de una memoria temporal esta se puede limpiar al término de una tarea o al apagar el dispositivo (FERNÁNDEZ, 17 de marzo 2021).

1.7.3 CPU (Unidad central de procesamiento)

Es la unidad central que procesa instrucciones y ejecuta tareas, la velocidad de procesamiento se la mide por MHz o GHz y esta varía dependiendo de la calidad y capacidad que tienen cada procesador (FERNÁNDEZ, 10 de marzo 2021).

1.7.4 Threads (Hilos)

Los hilos son unidades que contienen instrucciones de procesos, es más eficiente ejecutar un programa con múltiples hilos los cuales se ejecutarán simultáneamente brindando al usuario menor tiempo de respuesta en tareas que debe realizar un proceso, ya que comparten recursos (techlandia, 20 de enero de 2020).

1.8 LENGUAJES DE PROGRAMACIÓN Y HERRAMIENTAS

1.8.1 Native C++ en Android Studio

Android Studio es un entorno de trabajo y desarrollo capaz de trabajar con CMake, que es muy útil para proyectos multiplataforma, lo cual permite realizar trabajos java y c++, además de que permite integrar varias librerías en lenguaje nativo.

1.8.2 C++

Es un lenguaje de programación de nivel medio que descende de C y combina programación estructurada con la programación de objetos. Una de sus principales características es su alto rendimiento puesto que puede hacer llamadas directas al sistema operativo, permite la facilidad de trabajar con memoria dinámica y punteros que permiten el acceso a el espacio de memoria que ocupa una variable (Gary J. Bronson, 2006).

1.8.3 Cmake

Es una herramienta de código abierto utilizada para empaquetar software, brinda una ayuda para administrar y definir el compilador de código. Se usa principalmente para gestionar el proceso de compilación de software (CMake, s.f.).

1.8.4 Java

Lenguaje de programación orientado a objetos, flexible a diversos sistemas operativos, desarrollado por Sun Microsystems en 1995, se lo maneja para organizar una variedad de aplicaciones sin depender de la infraestructura del dispositivo ya que es portable.

1.8.5 Jni

Es la interfaz nativa java, que admite la comunicación a diferentes infraestructuras estableciendo su código en una máquina virtual de java, permitiendo acceder a las aplicaciones y herramientas que java proporciona creando y actualizando los objetos que comparte.

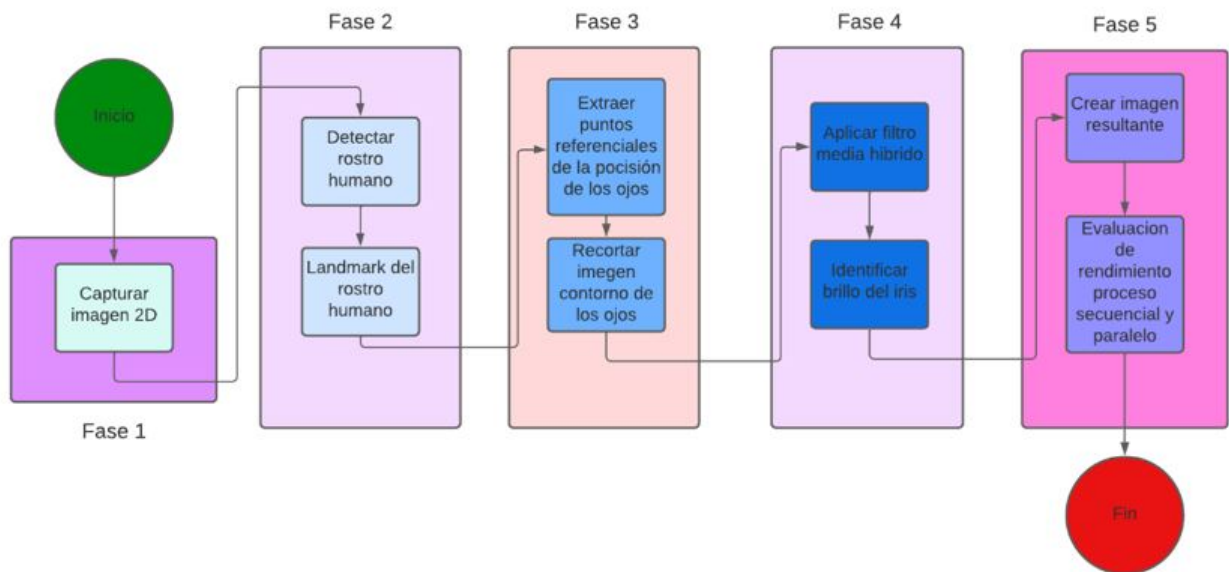
MARCO METODOLÓGICO

2.1 DISEÑO Y CONSTRUCCIÓN DE EL PROTOTIPO MÓVIL

Para extraer el contorno de ojos y mejorar el tiempo de procesamiento se sigue una estructura definida por cinco fases que permiten clasificar las tareas ejecutadas como se muestra en la Figura 11. i) Captura de la imagen 2D. ii) Detección de rostro humano y landmark, para generar puntos de referencia que estima la posición de las coordenadas (x,y), que muestran la ubicación de las características faciales.iii) Extraer puntos referenciales de la posición de ojos y recorte imagen de contorno de ojos, posterior a identificar las coordenadas se genera una imagen que contiene el recorte de la sección de los ojos. iv) Aplicar filtro mediano hibrido e identificar brillo del iris que contine la imagen resultante de la extracción del contorno de un ojo. v) Crear imagen resultante y evaluar el rendimiento del proceso secuencial y paralelo, para definir que paradigma es el adecuado para el procesamiento de la imagen.

Figura 11

Fases de diseño prototipo móvil



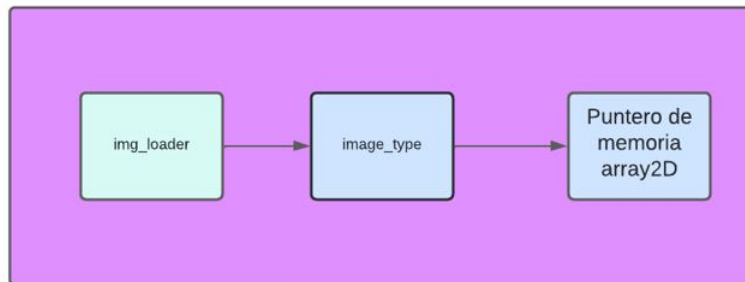
Nota: Fases del diseño para prototipo móvil. Elaborado por: Los autores.

2.1.1 Fase 1: Captura de la imagen 2D

La captura de la imagen en formato JPG se realiza a través de un dispositivo móvil, la cual se almacena en una carpeta nombrada escenarios, la imagen sirve como elemento de entrada, para ello el objeto se tiene un formato array 2D similar a una matriz bidimensional que contiene los píxeles que forman la imagen. La imagen se almacena en un puntero de memoria de este modo la información podrá ser utilizada por diferentes procesos sin afectar el rendimiento en las tareas que se realizan para en preprocesamiento de la imagen, ya que las imágenes pueden tener distintos tipos de formatos el *algoritmo 1* evidencia el proceso a realizar para la captura, identificación del formato de imagen y posterior reserva de espacio de memoria en la cual se ubicara la imagen como muestra en la Figura 12.

Figura 12

Captura de la imagen 2D



Nota: Fase 1 de captura de la imagen, identificación del tipo de elemento y reserva de espacio en memoria . Elaborado por: Los autores.

Cabe recalcar que se realiza la carga de imagen de esta forma puesto que la librería de procesamiento de imagen Dlib lo requiere de esta forma. Como se definió en partes anteriores, esta librería y su modelo preentrenado es el que nos permitirá obtener los puntos de referencia del rostro, los cuales son necesarios para la obtención del contorno de ojos.

Para mejorar el proceso de captura de la imagen el proceso se divide en tres tareas las cuales están definidas por los siguientes métodos: *Puntero array2D*: estructura generada por dlib que almacena punteros en el lugar donde se encuentra almacenada la imagen, para así evitar la falla de sincronización si algún otro proceso desea llamar a la imagen de este modo se mejora el rendimiento al trabajar en cada píxel. *Img_loader*: representa una clase que carga un archivo de imagen almacenada para ello llama a la memoria *imgbuffer* de escala *imgbuffer_size*. *Image_type*: interpreta la interfaz de la imagen que define dlib asegurando que se trata de una imagen.

El objeto imagen tiene funciones generales que se definen: *Num_rows()*: retorna el número de filas que contiene la imagen. *Num_columns()*: retorna el número de columnas que contiene la imagen. *Image_data()*: retorna el puntero que identifica la fila y columna de la imagen.

Algoritmo 1 Carga de Imagen

Input: Profundidad imagen

Output: Puntero de memoria

```
1: Height = 8 | 16 | 24
2: Width = 8 | 16 | 24
3: rgb_pixel p
4: #pragma omp parallel shared(p)
5: for row = Height - 1 hasta row >= 0 do
6:     for col = 0 hasta Width do
7:         if col + 1 < Width then
8:             p.red = red[pixels[i]]
9:             p.green = green[pixels[i]]
10:            p.blue = blue[pixels[i]]
11:            assign_pixel(image[bottomUp Height - row - 1 : row][col+1], p)
12:        end if
13:    end for
14: end for
```

El *algoritmo 1* describe que el elemento de entrada permite definir la profundidad de la imagen que puede tomar el valor de 8, 16 o 24 bits (canal), la línea uno y dos del *algoritmo 1*, define el valor que puede ocupar la imagen para desplazarse entre la matriz resultante de un objeto pixel de tres canales como muestra la línea tres.

Para mejorar el rendimiento del algoritmo se inicia la directiva *parallel* junto a la cláusula *shared* que comparte la variable *p*, facilitando el acceso al espacio de memoria para que todas las tareas a realizar puedan hacer uso de la variable compartida, permitiendo paralelizar procesos que se encuentren dentro de la estructura llevando a cabo tareas simultaneas.

La línea cinco y seis define el rango por el cual se moverá la matriz para asignar los valores de cada pixel, como muestra las líneas ocho, nueve, diez y once. Realizando una comprobación para no salir del límite de la imagen que se genera.

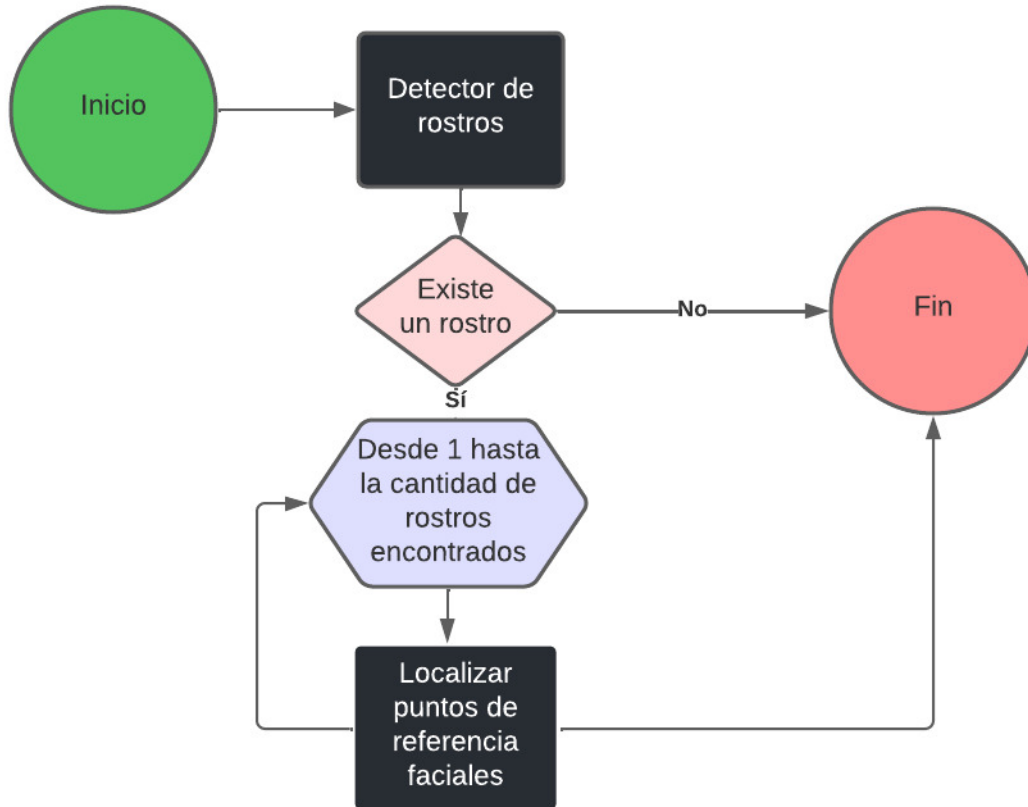
2.1.2 Fase 2: Detección de rostro humano y landmark

Para la detección de rostro se hace uso de la librería *dlib* que permite estimar las coordenadas(x,y) que hacen referencia a la posición de las características faciales como son los ojos,

nariz, boca y contorno del facial, El proceso que se realizó se especifica en la Figura 13.

Figura 13

Implementación de la técnica Landmark



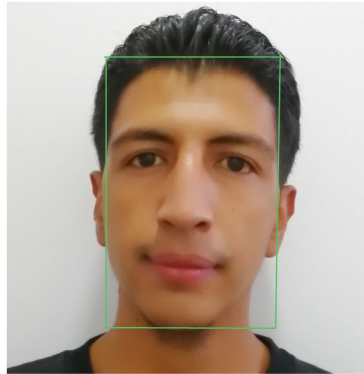
Nota: Proceso para la detección del rostro y extracción de puntos de referencia facial .
Elaborado por: Los autores.

la librería contiene una variedad de herramientas que permiten la lectura de la imagen para obtener un arreglo bidimensional, para su posterior reconocimiento facial con ayuda de la función de histograma de gradientes orientados (HOG), junto con un clasificador lineal y una ventana deslizante establecen los puntos de referencia que identifican los sectores que componen un rostro humano, utiliza punteros de memoria para ubicar los objetos encontrados en después de realizar el landmark de una imagen que contenga un rostro humano el algoritmo 2, evidencia el proceso que realiza para la identificación de rostros en una imagen y su posterior

análisis para identificar los puntos representativos que forman una cara humana, como resultado se obtiene la captura de un rostro en una imagen como se puede observar en la Figura 14.

Figura 14

Captura del rostro con la técnica Landmark



Nota: Proceso para la detección y captura del rostro en una imagen . Elaborado por: Los autores.

Algoritmo 2 Lectura Detección de Rostro y Landmark

Input: Imagen 2D (Matriz bidimensional);

Output: Coordenadas posición características faciales.

```
1:  $dets \leftarrow detector(Imagen\_2D)$ 
2: if  $dets$  is NULL then
3:   return NULL
4: else
5:   for  $i = 0$  hasta  $dets.size$  do
6:     Localizar puntos de referencia faciales
7:     full_object_detection shape = shape_predictor ( $dets$ )
8:     Shapes.push_back(shape)
9:   end for
10: end if
11: return shape (Coordenadas posición características faciales)
```

El algoritmo 2 tiene como entrada un matriz bidimensional, la primera línea tiene como tarea realizar la detección y conteo de la cantidad de rostros humanos que posee una imagen, si al término de la primera tarea no se detecta ningún rostro el algoritmo retornara el valor de nulo de este modo se realiza una parada en la ejecución de las diferentes tareas consecutivas

como muestra la línea dos del algoritmo. Caso contrario al detectar uno o más rostros procede a realizar la localización de las áreas que contienen los rostros esto lo efectúa mediante un ciclo de repetición como muestra la línea cinco del algoritmo 2, continua con la identificación de los puntos representativos que forman el rostro, almacenando en un objeto, finalmente retornando el objeto que almacena las coordenadas estimadas de los puntos de interés.

El proceso de identificación y extracción de coordenadas de los puntos representativos del rostro ejecuta varios métodos como son:

- **detector(imagen 2D):** Retorna la cantidad de rostros encontrados en una imagen.
- **shape_predictor(dets):** Retorna las coordenadas de los puntos que forman un sector del rostro humano (contorno rostro, ojos, nariz, boca)
- **push_back(shape):** Marca los puntos en la imagen.
 - **Dets:** lista de cuadros delimitadores alrededor de cada rostro localizado en la imagen.
 - **shape:** ubicaciones de las partes del rostro.

Al crear un objeto **full_object_predictor**, identifica y almacena las referencias para facilitar la ubicación retorna puntos que contienen las coordenadas que identifican a cada sector que forman un rostro humano.

2.1.3 Fase 3: Extraer puntos referenciales de la ubicación de los ojos y recorte de la imagen del contorno de los ojos

Para realizar la extracción de la imagen que forma el ojo se debe proporcionar al algoritmo los puntos de referencia antes ubicados que forman el contorno del ojo para así realizar un recorte en la imagen original.

El algoritmo 3 muestra los pasos que seguirá para la creación de una imagen que contiene el recorte del sector de los ojos, para realizar la tarea debe recibir la dirección (path) en la cual se almacenara la imagen resultante, al igual que los puntos de referencia que almacenan las coordenadas en la que se localiza el sector de los ojos, para posterior realizar un recorte de la imagen original dando como resultado una imagen nueva como se evidencia en la imagen 15, el contorno que forma el sector de los ojos se encuentra resaltado con líneas que encierran los ojos para así tener una referencia de las coordenadas que encierran los ojos.

Algoritmo 3 Recorte de ojos (cutImage)

Input: path, leftEyep1, leftEyep2, leftEyep3, leftEyep4, leftEyep5, leftEyep6, leftEyep7, leftEyep8;

Output: imagen.bmp.

- 1: $high \leftarrow (LeftEyep2_y - LeftEyep1_y)$
- 2: $width \leftarrow (RigthEyep2_x - LeftEyep1_x)$
- 3: **array2d**< *rgb_pixel* > img
- 4: **array2d**< *rgb_pixel* > resul
- 5: **result.set** (width,high)
- 6: **array2d**< *rgb_pixel* > and *crop_img* = resul
- 7: **array** <point 4> pts =
- 8: **point**(LeftEyep1_x , LeftEyep1_y)
- 9: **point**(RigthEyep2_x , RigthEyep2_y)
- 10: **point**(LeftEyep1_x , LeftEyep1_y)
- 11: **point**(RigthEyep2_x , RigthEyep2_y)
- 12: **extract_image_4points** (img, crop_img, pts)
- 13: **save_bmp** (crop_img, resul/' nombre + .bmp)

- **path:** directorio en el cual se almacenará la imagen resultante
- **array2d:** objeto bidimensional parecido a una matriz que almacena punteros de memoria.
- **point:** objetos que almacenan los puntos de referencia de las coordenadas en la cual se localiza el objeto.

Figura 15

Detección puntos de referencia de los ojos en un rostro



Nota: El resultado de la detección de los puntos de referencia que forman los ojos en un rostro humano. Elaborado por: Los autores.

El algoritmo 3 en las líneas tres y cuatro, establece los arreglos que utilizará para almacenar la imágenes a ser tratadas, mientras la línea cinco inicializa el arreglo resultante que almacena los puntos de referencia en los cuales se encuentra el sector que forma los ojos, para ello las líneas que van desde la número ocho hacia la once especifican los puntos que trazaran las líneas que encierra el contorno de los ojos, la línea doce realiza un corte de la imagen y por último la almacena esta imagen cortada con la extensión .bmp como se muestra en la imagen 16.

Figura 16

Recorte del sector del ojo



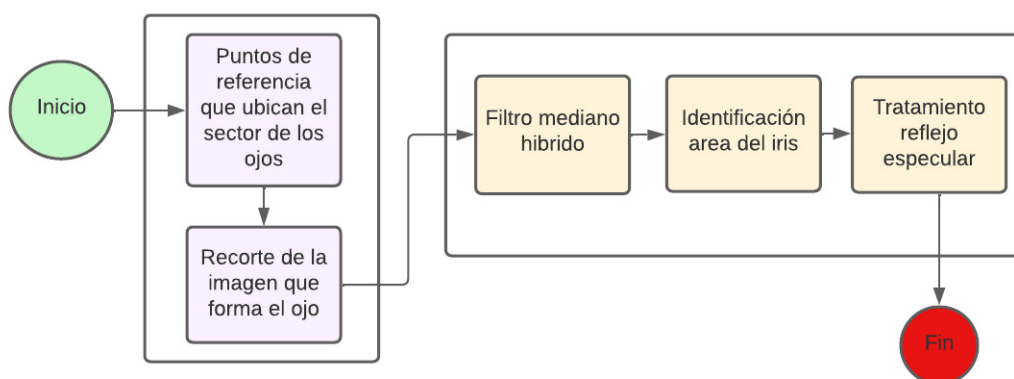
Nota: El resultado de identificar los puntos de referencia que forma el ojo facilita obtener una imagen que contiene el sector del ojo. Elaborado por: Los autores.

2.1.4 Fase 4: Aplicar filtro mediano hibrido e identificación de brillo en la imagen

La implementación del proceso muestra las tareas que realizo en la fase 3 para el recorte de la imagen, a continuación muestra la aplicación del filtro, identificación del círculo que forma el iris y el tratamiento de reflejo especular se especifica en la Figura 17.

Figura 17

Proceso de recorte imagen, aplicación filtro y tratamiento reflejo especular



Nota: Tareas que se realizan para aplicar la identificación y tratamiento del reflejo especular. Elaborado por: Los autores.

El proceso para aplicar el filtro mediano hibrido se contempla en el algoritmo 4 el cual

muestra que se tiene como entrada una imagen que contiene el recorte del ojo, un arreglo resultante y las dimensiones que forma la imagen.

Para optimizar la técnica se comparte las variables que almacenará la imagen resultante después de aplicar el filtro como se muestra en la línea dos del algoritmo, recorriendo de forma horizontal como se muestra en el segmento del algoritmo que va desde la línea siete a la doce y vertical aplicando las operaciones del filtro como muestra las líneas dieciséis hasta la veinte

Algoritmo 4 Filtro mediano híbrido

Input: image, result, N, M;

Output: result.

```
1: Recorre la ventana a través de los elementos de la imagen.
2: #pragma omp parallel shared(window, result)
3: for  $m = 1$  hasta  $M - 1$  do
4:   for  $n = 1$  hasta  $N - 1$  do
5:     window[5]
6:     results[3]
7:     Almacena los elementos diagonales de la ventana
8:     window[0] = image[( $m - 1$ ) * N + n]
9:     window[1] = image[ $m$  * N + n - 1]
10:    window[2] = image[ $m$  * N + n]
11:    window[3] = image[ $m$  * N + n + 1]
12:    window[4] = image[( $m + 1$ ) * N + n]
13:    Obtiene la mediana
14:    results[0] = median(window, 5)
15:    Almacena los elementos de la coordenada x de la ventana
16:    window[0] = image[( $m - 1$ ) * N + n - 1]
17:    window[1] = image[( $m - 1$ ) * N + n + 1]
18:    window[2] = image[ $m$  * N + n]
19:    window[3] = image[( $m + 1$ ) * N + n - 1]
20:    window[4] = image[( $m + 1$ ) * N + n + 1]
21:    Obtiene la mediana
22:    results[1] = median(window, 5)
23:    Almacena el elemento principal
24:    results[2] = image[ $m$  * N + n]
25:    Obtiene los resultados
26:    result[( $m - 1$ ) * (N - 2) + n - 1] = median(results, 3)
27:   end for
28: end for
29: return result
```

El algoritmo anterior hace es colocar una ventana cruzada sobre el elemento de dimensio-

nes especificadas, luego recoge los elementos para proceder a ordenarlos tomando el elemento del medio, realiza los pasos anteriores nuevamente, pero a hora coloca una ventana x sobre el elemento los cuales los recoge y rodona, obteniendo así el del medio.

2.1.4.1 Identificar reflejo (reflexión especular). La reflexión especular degrada la calidad de una imagen, por el fenómeno que produce cuando la luz rebota en superficies que no absorben toda la luz esto de produce cuando se tiene una superficie pulida, para solucionar este problema se propone el algoritmo 5 que identifica los sectores que contienen reflexión especular para luego ser tratadas con ayuda de los pixeles vecinos.

Para llevar a cabo este proceso se hace uso de varias técnicas como la de canny para la detección de bordes en una imagen este proceso se implementa en la línea uno del algoritmo puesto que sirve como eje para proceder con las demás tareas que realiza en algoritmo. en la fase dos se aplica la técnica para cambiar el brillo de una imagen es así que facilita la detección de los círculos que forma el iris del ojo humano para continuar con la extracción de los coordenadas y el radio que las forma como se evidencia en la línea tres que llama a la función detectar circulo. Marcar el contorno que forma el círculo del iris el algoritmo en las líneas cinco hasta la doce recorre los pixeles para marcar el circulo que encierra el objeto esto se puede evidenciar en la Figura 18a se aplica el filtro mediano hibrido y se procede a la detección del círculo del iris resaltando el sector como muestra la Figura 18b.

Algoritmo 5 Identificador reflejo (reflexión especular)

```
1: canny(img, mat_gradient, mat_or, outrow, outcol)
2: adjgamma(mat_gradient, outrow, outcol, g)
3: pcoor = detectar_circulo (afinal, outrow, outcol, r_min, r_max, 0.1)
4: resizeExternalCoor (rx, ry, r, imgCol, outcol, imgRow, outrow, pcoor)
5: for  $i = \text{fila} - \text{radio}$  hasta  $\text{fila} + \text{radio}$  do
6:   for  $j = \text{columna} - \text{radio}$  hasta  $j < \text{columna} + \text{radio}$  do
7:     mat_ojo[conti][contj] = img2[i][j]
8:     results[3]
9:     contj++
10:   end for
11:   contj = 0
12:   conti++
13: end for
14: canny(mat_ojo, mat_gradient2, mat_or2, fil_col, fil_col)
15: adjgamma(mat_gradient2, fil_col, fil_col, g2)
16: paintCircle(img4, imgRow, imgCol, rx, ry, rx2, ry2, r, r2)
17: for  $i = 0$  hasta  $\text{imgRow}$  do
18:   for  $j = 0$  hasta  $j < \text{imgCol}$  do
19:      $d = (\text{pow}((\text{rx}-j),2)) + (\text{pow}((\text{ry}-i),2))$ 
20:      $\text{rad} = \text{pow}(r,2)$ 
21:     if  $d < \text{rad}$  then
22:       acum=acum+img4[i][j]
23:       cont=cont+1
24:       if  $\text{img4}[i][j] > \text{mayor}$  then
25:         mayor=img4[i][j]
26:         cont2=cont2+1
27:         acum2=acum2+mayor
28:       end if
29:     end if
30:   end for
31: end for
32: if  $\text{cont2} \neq 0$  then
33:    $\text{max} = \text{acum2} / \text{cont2}$ 
34:    $\text{promedio} = \text{acum} / \text{cont}$ 
35:    $\text{uref} = \text{promedio} + 0.9 * (\text{max} - \text{promedio})$ 
36:   for  $i = \text{ry} - r$  hasta  $i < \text{ry}$  do
37:     for  $j = \text{rx} - r$  hasta  $j < \text{rx} + r$  do
38:        $d = (\text{pow}((\text{rx}-j),2)) + (\text{pow}((\text{ry}-i),2))$ 
39:        $\text{rad} = \text{pow}(r,2)$ 
40:       if  $d < \text{rad}$  then
41:         if  $\text{img4}[i][j] > \text{uref}$  then
42:            $\text{img4}[i][j] = 255$ 
43:         end if
44:       end if
45:     end for
46:   end for
47: end if
```

El algoritmo anterior muestra el camino para detectar reflejos especulares. (i) Como primer paso se utiliza una técnica de detección de bordes, en este caso canny. (ii) En el paso 2 se utiliza la técnica de detección de círculos de hough para detectar la circunferencia del iris ya trazada por canny. (iii) En el paso 3 con el área del iris ya detectada se procede con la Binarización de la imagen para detectar reflejos especulares el umbral usado está dado por la ecuación:

$$U_{ref} = I_{prom} + p \cdot (I_{max} - I_{prom}) \quad (2.1)$$

En la ecuación 2.1, I_{prom} pertenece al promedio de las intensidades de todos los pixeles de la imagen, mientras que I_{max} corresponde al máximo valor de las intensidades. Para evitar el efecto de unos pocos pixeles muy brillantes sobre el valor de I_{max} , este parámetro se calcula en base al promedio de intensidades del 0,1 % de los pixeles más brillantes de la imagen (cerca de 120 pixeles). La constante de proporcionalidad P se mantiene fija en 0,9. Este umbral se utiliza para binarizar la imagen en escala de grises y así conseguir un mapa de los reflejos especulares de la imagen original.

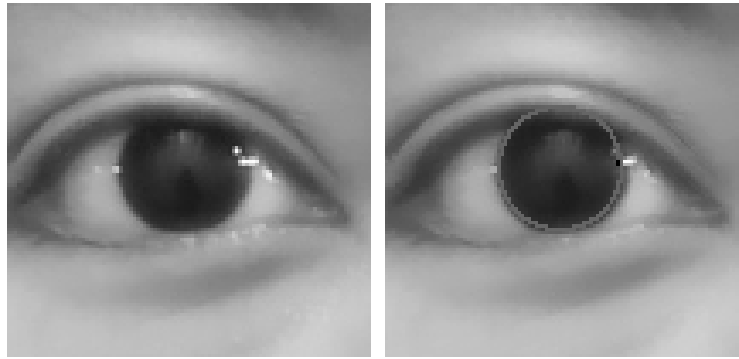
Como paso final al tener identificados los reflejos especulares con el mapa de reflejos, se procede a usar métodos de morfología matemática, en este caso el método de dilatación para abarcar todos los pixeles menos brillantes pero que son parte del reflejo. Las áreas identificadas como reflejos deben ser rellenadas con información de pixeles que las envuelven, de forma de poder reconstruir la imagen.

Al aplicar el filtro híbrido se puede notar que la imagen no pierde sus características que la forman, dando así un mejor entorno para trabajar con los pixeles, la imagen muestra como la detección de bordes conjunto con la alteración de intensidad de los pixeles de una imagen ayudan a identificar el contorno que forma el iris para posterior resaltarlos los resultados se los presenta en la Figura 18a

Figura 18

Imagen tratamiento de filtro y reflejo especular

(a) *imagen aplicando el filtro mediano hibrido* (b) *Imagen identificación y tratamiento del reflejo especular*



Nota: Representación gráfica de aplicar el filtro mediano hibrido conjunto con la detección de bordes para formar la circunferencia que marca el iris del ojo humano. Elaborado por: Los autores.

2.1.5 Fase 5: Crear imagen resultante y evaluación de rendimiento proceso secuencial y paralelo

Al término de la fase 4 se obtiene la imagen que identifica el contorno del iris y trata el brillo especular, se almacena en una carpeta denominada resultados que contiene todas las imágenes que han cumplido los procesos exitosamente.

La evaluación de rendimiento se lo realiza con la ayuda de la toma de datos del consumo de CPU, Memoria Ram y tiempo de ejecución para la parte secuencial, por otra parte la evaluar los procesos en paralelo se contempla la cantidad de hilos que se generan por cada proceso, el tiempo de ejecución de los sectores paralelizados, al igual que la memoria Ram y porcentaje de consumo de CPU.

2.1.5.1 Evaluación rendimiento secuencial. Para realizar el cálculo del rendimiento computacional se toma datos de cada técnica aplicada por el proyecto con ayuda de una clase denominada `system_metrics`, la cual contiene funciones como:

resetCounters(): El función pone los contadores en cero llamando a diferentes métodos que ejecutan la acción, permitiendo iniciar una medida del consumo de CPU, memoria y tiempo de ejecución.

calculate(): Función que calcula las métricas entre invocación `resetCounters` y esta función dejando sentado los atributos: `CalculateCPU,CalculateMemory,CalculateTime`.

`resetCounters()` esta función invoca a diferentes métodos que permiten resetear los contadores que capturan la información del consumo de recursos computacionales como son el cpu, memoria y tiempo de ejecución estos métodos se describen a continuación:

resetCPUCounters(): Método que setea los contadores de tick del CPU para posterior realizar una diferencia al invocar el método que calcula el consumo del método.

Por otra parte la función **calculate()** permite calcular los contadores que capturan la información del consumo de recursos computacionales como son el cpu, memoria y tiempo de ejecución estos métodos se describen a continuación: **calculateCPU():** Este método retorna el porcentaje de CPU usado entre las funciones **resetCouters()** y **calculate()**, el proceso que sigue para proveer del resultado es el tiempo de usuario usado más el tiempo de kernel usado dividido el tiempo total de CPU dividido la cantidad de procesadores. se establecen variables que capturarán el tiempo en el cual se inicia el cálculo del consumo mientras que se calcula el consumo actual para evitar el cálculo que no tiene relación con la ejecución de la técnico que se desea evaluar, además de que se captura el porcentaje de tiempo que usa el kernel, para posterior realizar el cálculo.

calculateMemory(): Método que calcula la diferencia entre el uso de memoria desde la

última llamada del método **resetCounters()** y el cálculo actual de memoria en la técnica o proceso que se desea evaluar.

2.1.5.2 Evaluación rendimiento paralelo. Para la evaluación de procesos en paralelo se hace uso de las métricas de rendimiento como el consumo de CPU, Memoria RAM, cantidad de hilos y tiempo de ejecución por cada hilo que se despliega para ejecutar las funciones o técnicas que se paraleliza.

RESULTADOS

3.1 HERRAMIENTAS

Para la evaluación de las técnicas utilizadas se utilizó el data set denominado "CREDENCIALES" que contiene 180 imágenes con diferentes dimensiones (largo, ancho), estas imágenes fueron capturadas con distintos dispositivos móviles dando como resultado la emulación de un ambiente real al que la aplicación puede ser expuesta, esto que permiten observar el comportamiento de las técnicas para el preprocesamiento de imágenes, adicional se trabajó con el data set de "CASIA", el cual posee alrededor de ochocientas mil imágenes de rostros humanos en diferentes posiciones el cual nos ayuda a evaluar el rendimiento de la técnica de reconocimiento facial y recorte de contorno de ojos. 1.

Tabla 1

Data set utilizado

Nombre	formato	Cantidad
CREDENCIALES	.PNG	130
CREDENCIALES2	.JPG	320
CASIA	.JPG	439320

Nota: *Data set utilizado con detalle y cantidad de imágenes. Elaborado por: Los autores.*

3.2 HARDWARE

Los equipos de cómputo que se utilizaron los detallados en la Tabla 2. Es importante mencionar que para la parte de entrenamiento se utilizó la Máquina 1 y para la parte de testing se utilizó la Máquina 2

Tabla 2

Hardware utilizado

Máquina	Procesador	Núcleos	Frecuencia	Memoria
1	i5	16	2.7 GHz	32 GB
2	Android MT6765	4	2.10 GHz	8 GB

Nota: Equipos utilizados junto al detalle de cada uno. Elaborado por: Los autores.

3.3 SOFTWARE

La librería dlib utiliza varias extensiones de CMake para configurar el compilador g++ propio de c++, para el desarrollo se utilizó la aplicación visual code para crear el backend y Android Studio para la aplicación móvil, a continuación se describe la versión utilizada en la tabla 3.

Tabla 3

Software utilizado

	Nombre	Descripción	Versión
Dependencias	dlib	Librería para manejo de imagenes .	3.7
	g++	Compilador para C++.	9.3
	CMake	Herramienta de configuración del compilador.	3.0
Entornos de desarrollo	Visual Code	IDE para C/C++.	1.64
	Android Studio	IDE para Android.	2022

Nota: IDE de desarrollo, versiones y descripción. Elaborado por: Los autores.

3.4 COMPILACIÓN

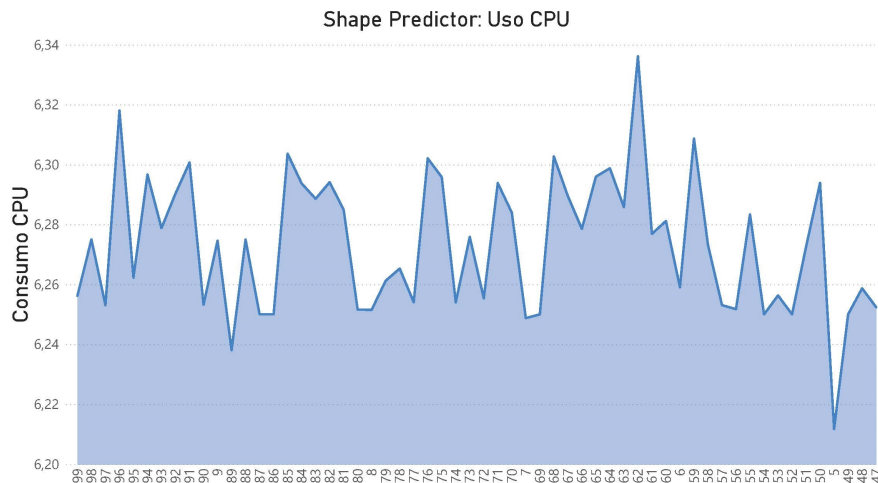
Para las pruebas de funcionamiento de los algoritmos de detección de rostro, preprocesamiento de imágenes se utilizó la maquina uno detallada en la Tabla 2. Para la ejecución se hace uso de la data set CREDENCIALES, para así poder evaluar los resultados de las técnicas aplicadas.

3.4.1 *Compilación primera etapa reconocimiento facial*

La evaluación del modelo inicia con el reconocimiento facial el cual se lo realiza con la ayuda de la librería dlib, se toma en cuenta el consumo de CPU, memoria RAM, Tiempo de ejecución para la parte de programación secuencial.

Figura 19

Consumo CPU shape Predictor



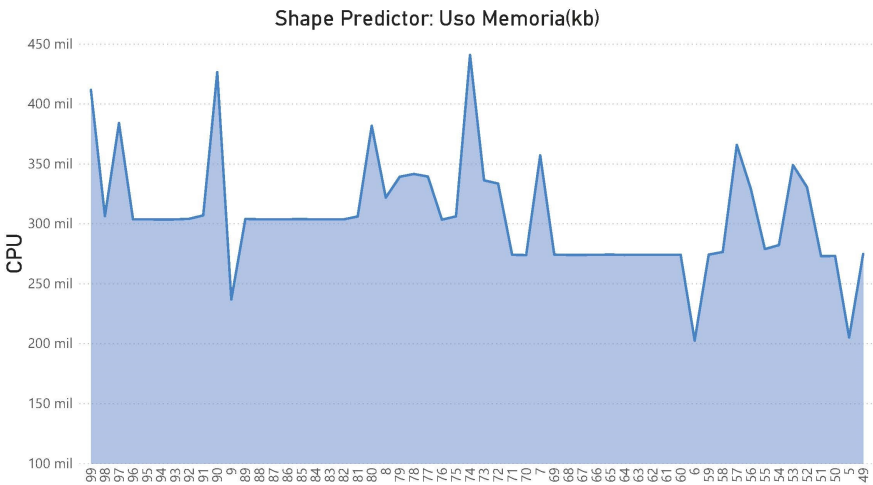
Nota: *Consumo CPU Shape Predictor valores pico de imagen etapa identificación de rostro. Elaborado por: Los autores.*

Para esta etapa la aplicación toma la imagen realizando el mapeo de la misma para identificar si existe un rostro para posterior pasar la imagen por el modelo pre entrenado de la librería que

permite encontrar los puntos de referencia y realizar un recorte de la imagen en la que localizo los ojos, los recursos utilizados en esta etapa se observa en la Figura 19 que muestra la cantidad de CPU consumida por cada imagen procesada, la Figura 20 representa el consumo de memoria (kb) para cada imagen procesada, la Figura 21 indica el tiempo en milisegundos que le tomo al procesador ejecutar la tarea de la librería dlib.

Figura 20

Consumo Memoria (kb) shape Predictor

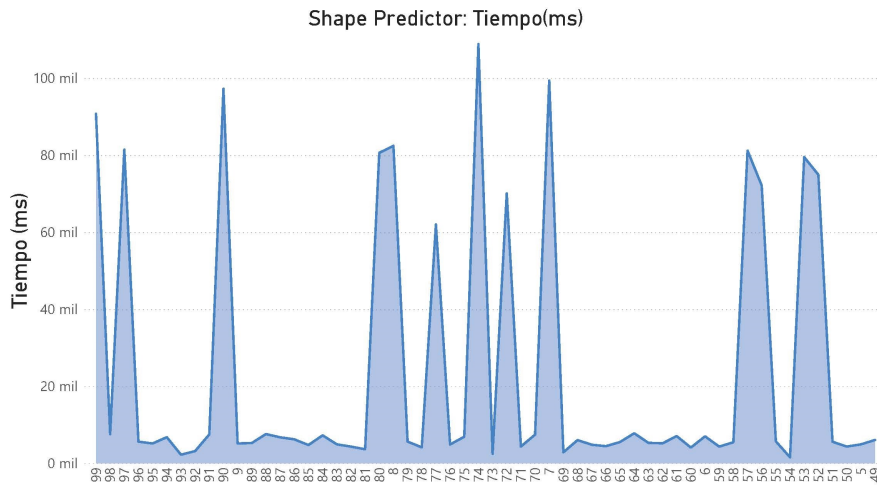


Nota: Consumo Memoria (kb) Shape Predictor valores pico de imagen etapa identificación de rostro. Elaborado por: Los autores.

La Tabla 4 permite identificar los valores que se representan en las Figura 19 que indica el comportamiento de los recursos computacionales, cada columna muestra el recurso computacional utilizado para el análisis de una imagen, ya que el proceso se divide en varias etapas como se describe en la fase uno del proyecto se debe realizar la evaluación de cada etapa del procesamiento de imagen.

Figura 21

Consumo Tiempo (ms) shape Predictor



Nota: Consumo Tiempo (ms) Shape Predictor valores pico de imagen etapa identificación de rostro. Elaborado por: Los autores.

Tabla 4

Resumen consumo recursos computacionales shape Predictor

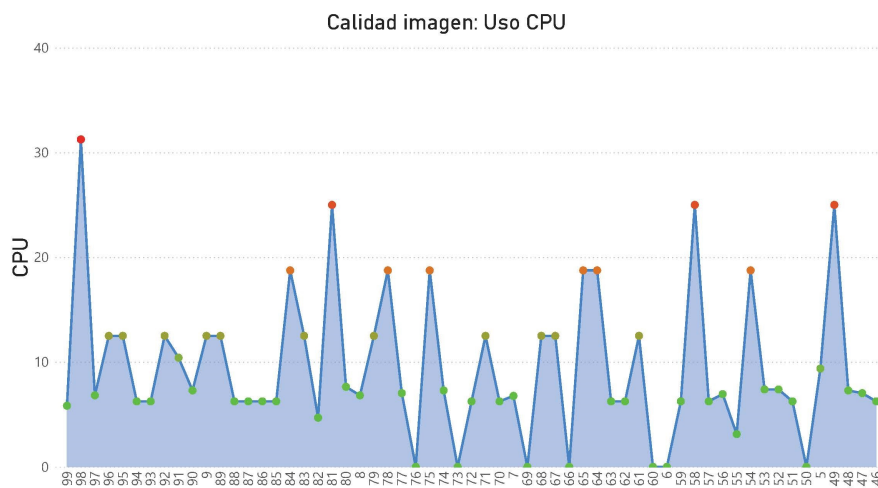
Nombre	CPU	Memoria(kb)	Tiempo(ms)
1	6.26093	174240	34366.8
10	6.27129	234036	5887.65
100	6.28501	306968	3582.67
101	6.25542	371408	69262
102	6.30319	306380	4733.39
103	6.33152	304540	4663.04
11	6.27588	234036	4843.12
12	6.26694	233972	3700.77
13	6.28301	233900	5711.89
14	6.31238	233900	5060.51
15	6.31068	233900	4156.74
16	6.29883	233700	5159.92
17	6.29112	233960	4589.53
18	6.30721	233848	4411.36
19	6.28079	233776	4084.99
2	6.26653	140728	3785.75
20	6.27583	233776	4864.5

3.4.2 *Compilación segunda etapa técnicas de procesamiento de imágenes*

Al llevar a cabo acciones que modifican el estado original de una imagen la misma tiende a perder características propias para cuantificar la afectación de aplicar un filtro a una imagen se utiliza métricas que evalúan el estado de la imagen resultante la Figura 26 muestra la variación de PSNR de las imágenes que se procesa conjunto con su MSE el cual indica la diferencia que tiene la imagen original con la resultante como se aprecia en la Figura 25.

Figura 22

Consumo CPU metricas Calidad de imagen

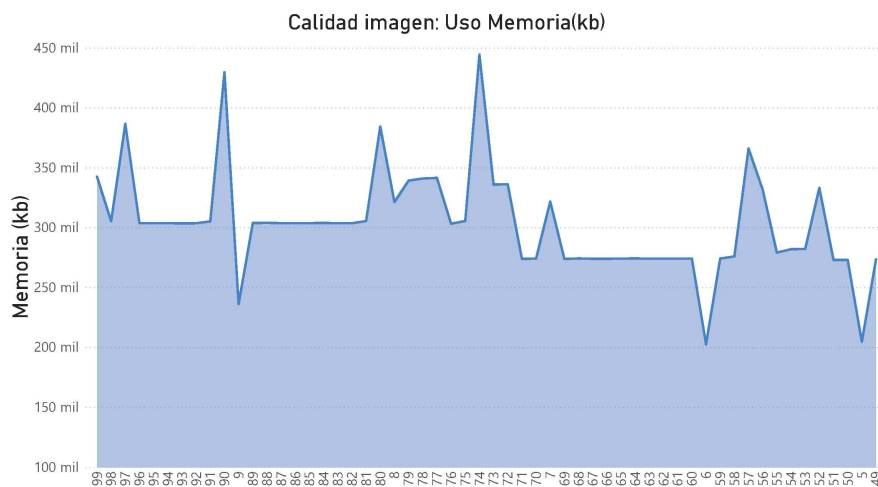


Nota: Consumo CPU metricas de calidad en imagen proceso evaluación MSE PSNR.
Elaborado por: Los autores.

Las métricas de evaluación de la técnica aplicada se aprecia en la Figura 22 la cual indica el consumo de CPU, se puede apreciar que el consumo del recurso se eleva al procesar imágenes con dimensiones que sobrepasan el promedio.

Figura 23

Consumo Memoria (kb) calidad de imagen



Nota: Consumo Memoria (kb) calidad en imagen proceso evaluación MSE PSNR. Elaborado por: Los autores.

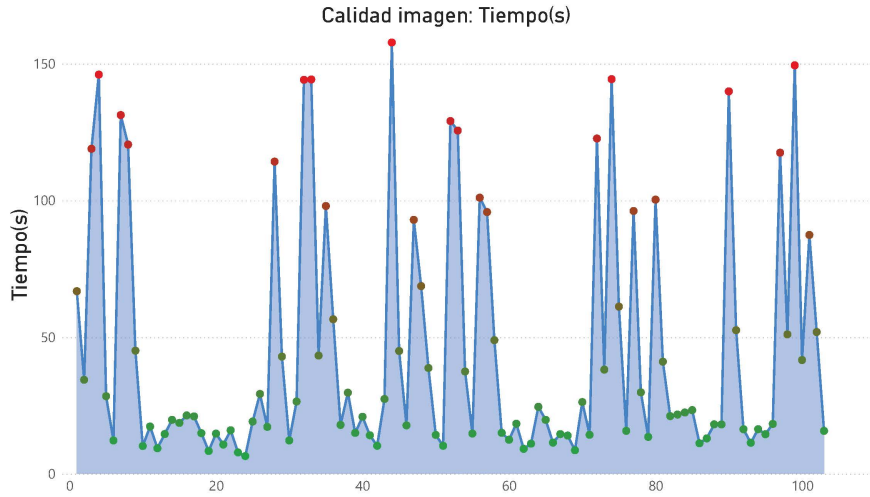
Por otra parte en la Figura 23 se identifica el consumo de memoria (kb) el cual es casi constante al procesar todas las imágenes ya que el proceso lo realiza secuencialmente no tiene afectación mayor a la memoria debido a que no comparte espacios de memoria, por último el tiempo de ejecución que está representado por la Figura 24 tiene una variación constante entre las imágenes que procesa debido a que depende de las características de la imagen para ser procesadas.

La Tabla 5 contiene un resumen de los valores que representa el consumo de los recursos computacionales en la etapa de cálculo de la calidad de la imagen (MSE,PSNR), mientras que la Figura 25 contiene la representación gráfica de los valores resultantes del cálculo MSE en la cual se puede identificar que los puntos de color rojo significan que se pasó del rango aceptable.

Mientras que los puntos de color verde verifican que el estado de la imagen no cambió a gran medida con la imagen original y la Figura 26 indica el máximo valor de señal de ruido en

Figura 24

Consumo Tiempo (ms) calidad de imagen



Nota: Consumo Tiempo (ms) calidad en imagen proceso evaluación MSE PSNR. Elaborado por: Los autores.

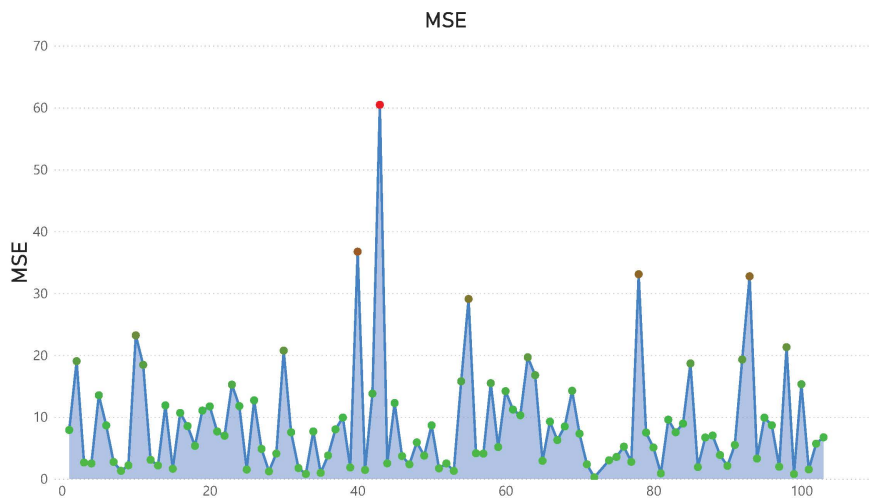
Tabla 5

Resumen evaluación calidad de imagen PSNR MSE

Imagen	CPU	Memoria(kb)	MSE	PSNR	Tiempo(ms)
1	6.25	163680	7.92547	39.1406	66.868
10	6.25	234036	23.2087	34.4743	10.254
100	25	305260	15.3187	36.2786	41.691
101	6.25	373520	1.55061	46.2258	87.45
102	15.625	305436	5.717	40.5591	51.941
103	6.25	304540	6.743	39.8423	15.785
11	6.25	233972	18.4463	35.4717	17.385
12	6.25	233900	3.1017	43.2148	9.438
13	12.5	233900	2.18645	44.7334	14.641
14	12.5	233900	11.8993	37.3756	19.814
15	6.25	233700	1.66068	45.9279	18.714
16	6.25	233960	10.6797	37.8452	21.439
17	0	233848	8.55419	38.809	21.085
18	6.25	233776	5.36778	40.8329	14.957
19	6.25	233776	11.0664	37.6908	8.445
2	18.75	140512	19.0465	35.3327	34.467
20	6.25	233752	11.7333	37.4366	14.786

Figura 25

MSE



Nota: Error cuadrático medio imagen original vs imagen procesada. Elaborado por: Los autores.

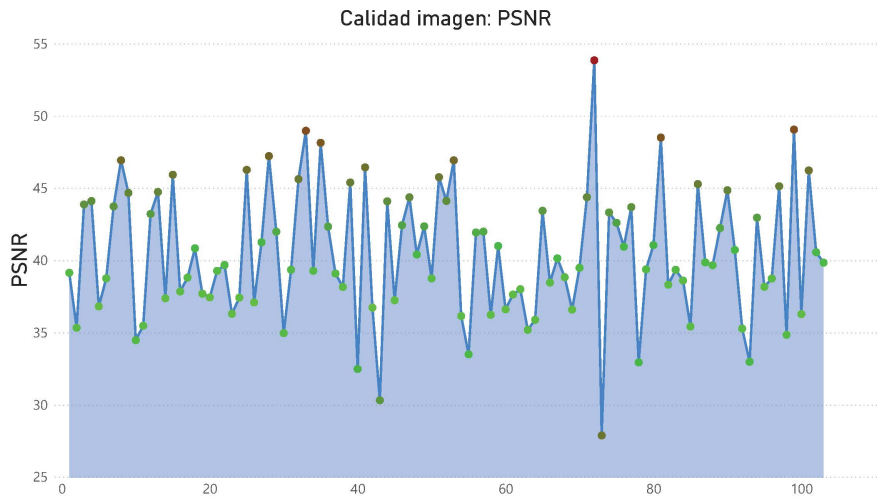
la imagen resultante los puntos de color rojo indican que la señal de ruido aumento al realizar el procesamiento de la imagen.

3.4.3 Evaluación técnicas procesamiento de imágenes Secuencial / Paralelo

La evaluación de las técnicas se realiza con ayuda de la toma de datos del consumo de recursos computacionales tales como: consumo de CPU, memoria (kb), tiempo de ejecución y cantidad de hilos creados en cada proceso como se representa en la Tabla 6 la cual contiene un resumen de los datos almacenados por cada imagen al realizar el proceso en secuencial y paralelo, las técnicas que se evalúan son: Reflejo especular para identificar y tratar el reflejo que contiene una imagen al tener superficies planas, filtro mediano hibrido para mejorar y suavisar la imagen, Gamma ayuda al cambio de intensidad de la imagen para el tratamiento de la misma, Canny que apoya a la identificación de bordes en una imagen y por ultimo Circle Hough que

Figura 26

Relación PSNR de la señal ruido



Nota: Representación grafica del maximo valor de señal ruido imagen resultante. Elaborado por: Los autores.

permite resaltar la circunferencia que forma el iris después de identificarlo con los procesos anteriores.

Se puede apreciar en la Tabla 6 que la selección de imágenes para evaluar los datos se realiza en diferentes escenarios, los valores indican el porcentaje de optimización de la técnica aplicando programación en paralelo, como se indica en la Tabla 6 el porcentaje de CPU y el tiempo de ejecución mejoran en comparación con la ejecución en secuencial debido a que las tareas se despliegan simultáneamente beneficiando el tiempo de respuesta y optimizando los recursos computacionales que ocupa una tarea para su ejecución.

Tabla 6*Tabla resumen de evaluación de técnicas por escenarios*

Escenarios	Técnicas	CPU % GB		Memoria(kb)		Tiempo(s)		Calidad imagen	
		Parelelo	Secuencial	Parelelo	Secuencial	Parelelo	Secuencial	MSE	PSNR
Credenciales	Hough	12.7311765	49.3116471	649360	294345.412	0.55957024	5.15062	2.21652	44.6741
	Canny	18.2598029	47.2794118	648171.294	289731.059	0.01855829	0.057299176		
	Gamma	9.34647059	30.1470588	647622.824	289839.765	0.01012715	0.016219353		
	Reflejo	4.39882353	27.4509824	324253.176	146722.353	1.81202941	2.274005294		
	Filtro HM	3.18	11.46	526576	238003	2.421	5.414		
Credenciales 2	Hough	12.3057445	4.725188	0.13413824	649359.575	294344.986	456.4569209	1.53758	46.2624
	Canny	17.1573899	46.1769988	648170.192	289729.956	-1.08385471	1.045113824		
	Gamma	7.92449059	28.7250788	647621.402	289838.343	-1.41185285	1.853537		
	Reflejo	2.96899853	26.0211574	324251.747	146720.923	0.38220441	0.844180294		
	Filtro HM	2.77514	16.77514	651890	727514	2.67923	4.76329		
CASIA	Hough	16.9435165	49.3116471	649360	294345.412	0.55957024	5.15062	3.79783	42.3355
	Canny	20.5716809	47.2794118	648171.294	289731.059	0.01855829	0.057299176		
	Gamma	11.4941236	33.6256588	651745.824	289839.765	0.01736915	0.020436073		
	Reflejo	5.81103353	28.8631924	324254.589	160134.765	3.22423941	3.686215294		
	Filtro HM	4.812552	15.46412	642142	856223	3.86621	6.97414		

La Tabla 6 indica el porcentaje de cambio que existe entre la métrica evaluada en proceso secuencial y paralelo, como se puede evidenciar el porcentaje de consumo del CPU en procesos paralelos mejora notoriamente con un promedio de 87%, el tiempo de ejecución de las técnicas en un 64%, por último el consumo de memoria incrementa en todos los procesos debido al trabajo de paralelización de memoria compartida, ya que las tareas que se ejecutan simultáneamente realizan peticiones constantes al espacio de memoria que almacena la información dando como resultado un promedio de 186% de incremento en el consumo de memoria.

El detalle de la relación de procesos en secuencial y paralelo se encuentran en las Tablas 7, 8, 5 y 9 las cuales se indica la diferencia de consumo de recursos computacionales que existe entre las diferentes técnicas aplicadas.

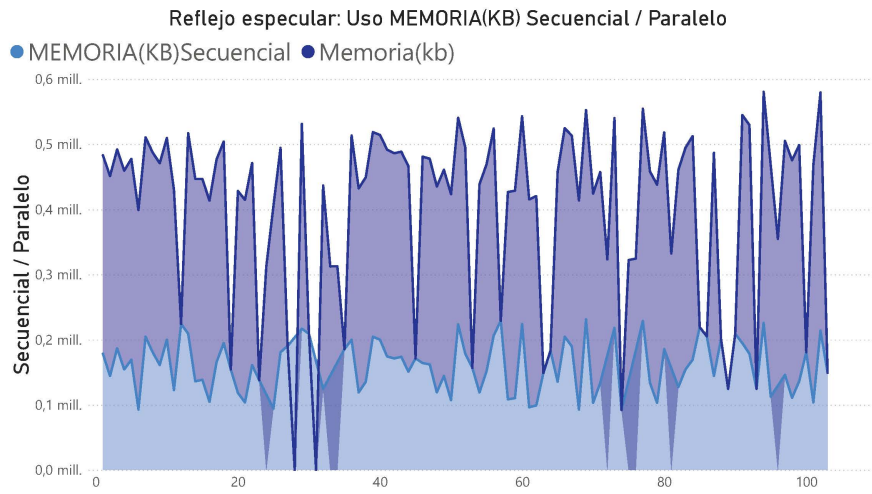
3.4.3.1 Evaluación técnicas Reflejo especular Secuencial / Paralelo. Al aplicar la técnicas de paralelismo se puede evidenciar la diferencia de uso de recursos como muestra la Figura 29 que representa el consumo de CPU, se puede evidenciar que al paralizar tareas el consumo de recursos se optimiza.

El tiempo de ejecución baja notoriamente como se representa en la Figura 28, por otra parte se puede notar que el consumo de memoria aumenta como consecuencia de compartir un espacio de memoria entre tareas que se ejecutan simultáneamente, se puede evidenciar en la Figura 27.

La Tabla 7 muestra un resumen de los resultados de las métricas evaluadas en la ejecución secuencial y paralela, es notorio que el consumo de CPU y tiempo de ejecución mejoran al aplicar técnicas de paralelización dando como resultado un mejor rendimiento y tiempo de espera menor, pero el consumo de memoria se incrementa notoriamente por las peticiones que realiza una tarea que se ejecuta simultáneamente.

Figura 27

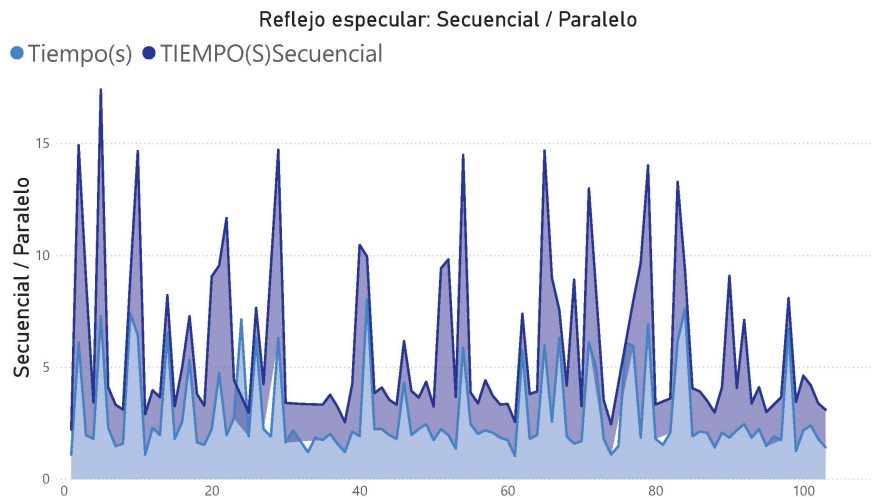
Consumo Memoria (kb) Reflejo especular



Nota: Consumo Memoria (kb) Reflejo especular Secuencia / Paralelo. Elaborado por: Los autores.

Figura 28

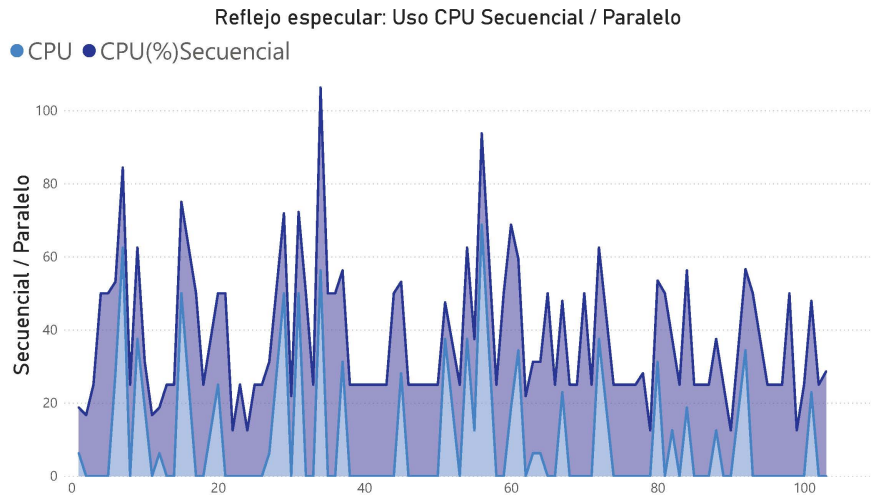
Consumo Tiempo (ms) Reflejo especular



Nota: Consumo Tiempo (ms) Reflejo especular Secuencia / Paralelo. Elaborado por: Los autores.

Figura 29

Consumo CPU evaluacion Reflejo especular



Nota: Consumo CPU etapa Reflejo especular Secuencia / Paralelo. Elaborado por: Los autores.

Tabla 7

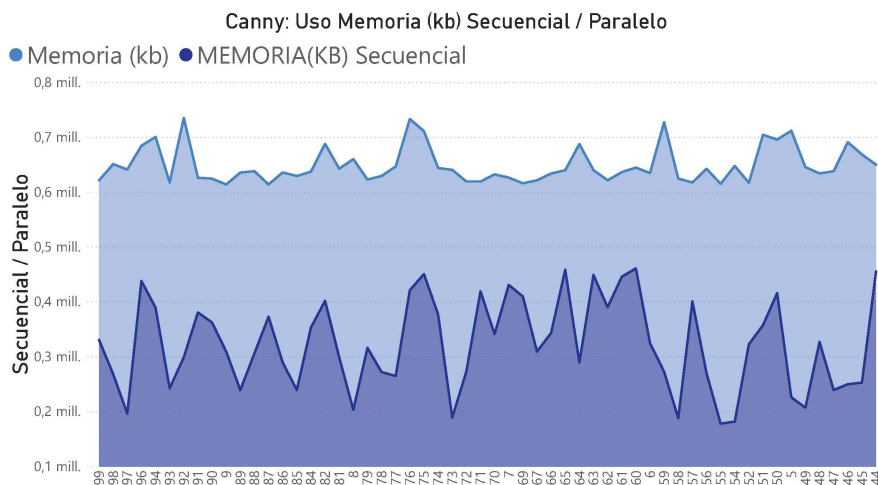
Resumen evaluación reflejo especular

Nombre	CPU Paralelo	CPU Secuencial	Memoria(kb) Paralelo	Memoria(kb) Secuencial	Tiempo(S) Secuencial	Tiempo(s) Paralelo	Total Hilos
1	3.75	25	310952	103808	4,8038	4,73004	6561
10	4.31	16.6667	320868	92952	1,8539	1,90303	1089
100	2.31	25	312588	110408	1,8539	1,19612	225
101	5	25	364524	110936	1,3945	6,68811	14161
102	4.22	25	364524	206284	1,5299	1,90291	900
103	3.56	25	317880	206284	1	2,01174	676
11	1.83	25	318128	108560	1,6515	2,06612	1089
12	1.92	25	318324	110408	1,475	1,84863	49
13	1.85	50	314784	173800	1,8539	2,22918	1369
14	1.63	25	321936	183316	1,9543	1,95739	14161
15	3.26	12.5	316068	124664	1,5299	1,84874	676
16	4.12	25	312132	124664	1,6675	1,68545	400
17	3.27	50	321936	218276	1,723	1,79431	900
18	19.3	25	359044	146048	1,911	1,73999	729
19	6.25	12.5	316068	107240	1,5003	1,73987	441
2	8.2	50	306480	180940	1,523	1,57664	676
20	0	25	316068	185692	1,5791	1,73982	676

3.4.3.2 Evaluación técnicas Canny Secuencial / Paralelo. Muestra de comportamiento de recursos computacionales en la técnica de detección de bordes, al aplicar técnicas de paralelismo se puede evidenciar la diferencia de consumo de recursos como muestra la Figura 32 que representa el consumo de CPU, como se puede observar al paralizar tareas el consulo de recursos se optimiza, el tiempo de ejecución baja notoriamente como se representa en la Figura 31.

Figura 30

Consumo Memoria (kb) Canny



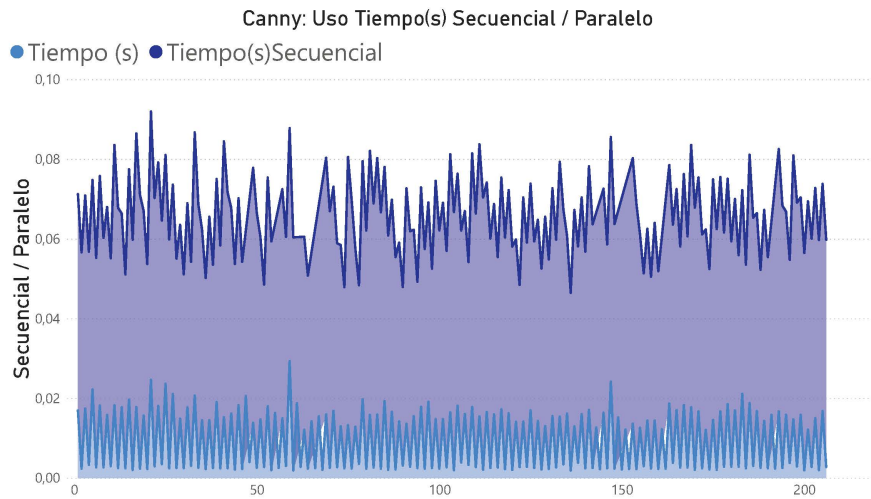
Nota: Consumo Memoria (kb) Canny Secuencia / Paralelo. Elaborado por: Los autores.

Por otra parte se puede notar que el consumo de memoria aumenta como consecuencia de aplicar técnicas de programación en paralelo ya que se comparte la memoria entre tareas que se ejecutan simultáneamente se puede evidenciar en la Figura 30.

Como se puede evidenciar la Tabla 8 presenta un resumen de los valores que genera la técnica para llevar acabo la identificación de bordes en una imagen, los resultados muestran que el consumo de CPU y tiempo de ejecución se acortan, pero el consumo de memoria aumenta por las peticiones simultaneas que realiza una tarea a un espacio de memoria.

Figura 31

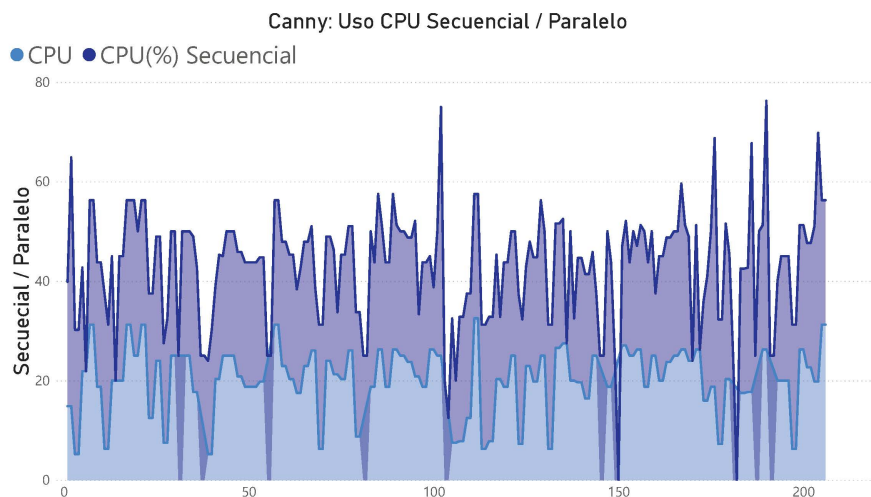
Consumo Tiempo (ms) Canny



Nota: Consumo Tiempo (ms) Canny Secuencia / Paralelo. Elaborado por: Los autores.

Figura 32

Consumo CPU evaluación Reflejo especular



Nota: Consumo CPU etapa Canny Secuencia / Paralelo. Elaborado por: Los autores.

Tabla 8*Resumen consumo recursos computacionales Canny*

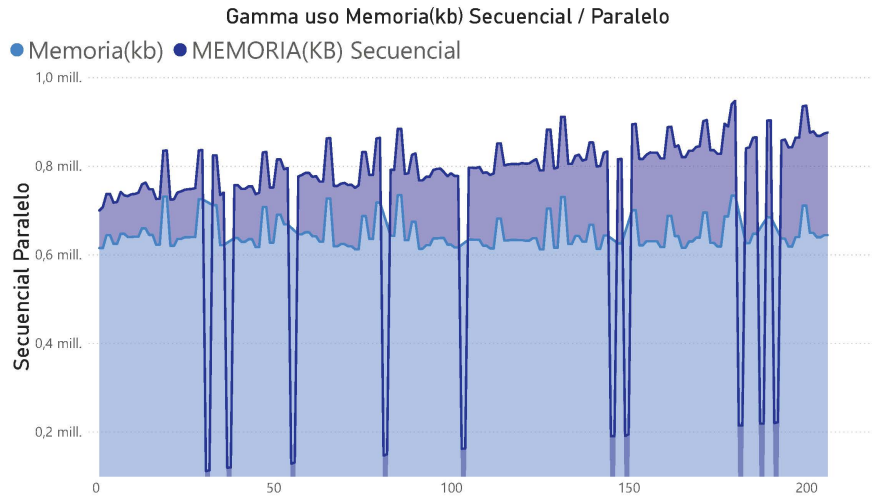
Nombre	CPU Paralelo	CPU Secuencial	Memoria(kb) Paralelo	MEMORIA(kb) Secuencial	Tiempo(s) Paralelo	Tiempo(s) Secuencial	Total hilos Canny
1	31.25	50	622676	207088	0.01764512	0.069207	480
10	5.20833	50	644064	185904	0.01802933	0.044554	445
100	5	45	627272	345224	0.01665206	0.0484	428
101	25	50	724872	221344	0.01959124	0.05096	522
102	18.75	40	626064	247480	0.01880075	0.058107	528
103	23.95833	25	638680	410984	0.01749878	0.068275	524
11	23.95833	50	639004	215800	0.01856144	0.052774	484
12	7.5	45	640440	218968	0.01945834	0.058107	497
13	18.75	50	633168	345224	0.02137367	0.058717	511
14	27.5	25	641944	364256	0.01646167	0.050398	472
15	20.83333	50	707464	247480	0.02178442	0.06228	540
16	18.75	50	627248	249328	0.01443253	0.0484	418
17	17.70833	75	647400	435232	0.01879876	0.056168	528
18	8.75	50	717720	290512	0.01919888	0.065798	520
19	12.5	50	635784	212104	0.01709233	0.059767	486
2	25	56.25	615976	359492	0.02164678	0.057787	522
20	20	42.5	629136	369008	0.01846482	0.064387	528

3.4.3.3 Evaluación técnicas AdjGamma Secuencial / Paralelo. Al aplicar técnicas de paralelismo se puede evidenciar la diferencia de consumo de recursos como muestra la Figura 35 que representa el consumo de CPU, como se puede observar al paralizar tareas el consumo de recursos se optimiza, el tiempo de ejecución baja notoriamente como se representa en la Figura 34, por otra parte se puede notar que el consumo de memoria aumenta como consecuencia de aplicar técnicas de programación en paralelo ya que se comparte la memoria entre tareas que se ejecutan simultáneamente se puede evidenciar en la Figura 33.

la Tabla 9 muestra un resumen de los valores de consumo computacional que genera la técnica AdjGamma que permite variar la intensidad de color de la imagen, es notorio que el consumo de CPU y tiempo de ejecución disminuyen por la aplicación de procesos en paralelo, caso contrario con el consumo de memoria aumenta por las peticiones simultaneas que realiza a un espacio de memoria.

Figura 33

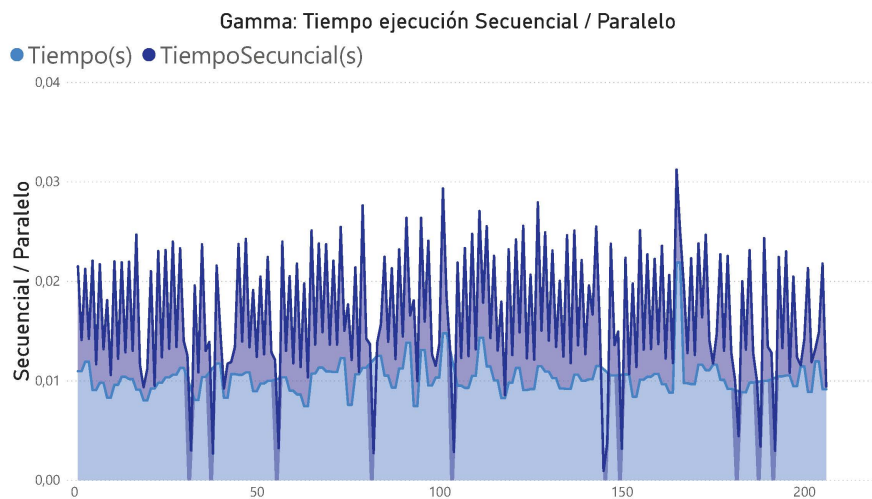
Consumo Memoria (kb) Gamma



Nota: Consumo Memoria (kb) Gamma Secuencia / Paralelo. Elaborado por: Los autores.

Figura 34

Consumo Tiempo (ms) Gamma



Nota: Consumo Tiempo (ms) Gamma Secuencia / Paralelo. Elaborado por: Los autores.

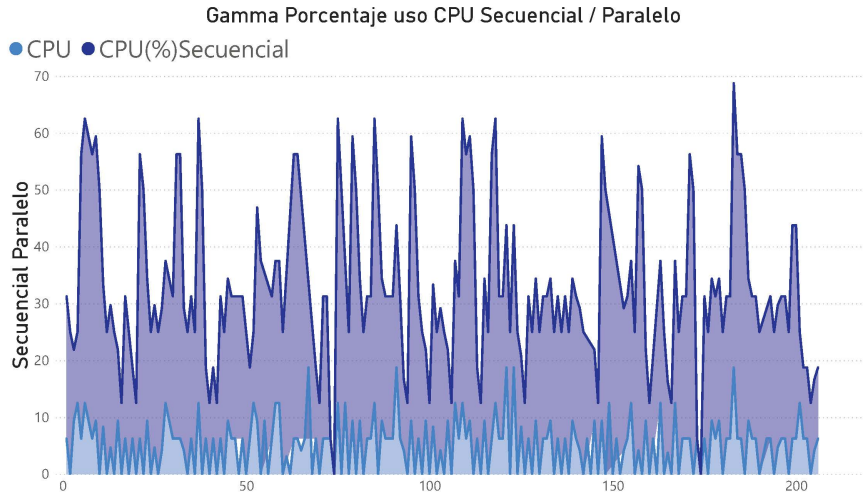
Tabla 9

Resumen consumo recursos computacionales Gamma

Nombre	CPU Paralelo	CPU Secuencial	Memoria(kb) Paralelo	MEMORIA(kb) Secuencial	Tiempo(s) Paralelo	Tiempo(s) Secuencial	Total Hilos Gamma
1	6,25	12.5	621884	207088	0,00908911	0.018041	960
10	6,25	25	643544	185904	0,01189591	0.01161	890
100	10,42	25	626956	249328	0,00893070	0.01804	856
101	10,94	25	724672	221344	0,01127500	0.014707	1044
102	9,38	50	625408	345224	0,00937004	0.016266	1056
103	9,38	50	638020	410984	0,00964080	0.015581	1048
11	9,38	25	638732	215800	0,01031571	0.015686	968
12	18,75	50	639320	218968	0,01058984	0.016151	994
13	12,50	50	632456	345224	0,00977694	0.016207	1022
14	6,25	25	641688	364256	0,00915678	0.01804	944
15	6,25	50	707220	247480	0,01082198	0.015943	1080
16	12,50	50	626624	249328	0,00892465	0.013615	836
17	9,38	12.5	646552	435232	0,00979510	0.016266	1056
18	6,25	25	717328	290512	0,01129485	0.019237	1040
19	9,38	12.5	635112	212104	0,00978085	0.015796	972
2	9,38	12.5	615348	359492	0,01090424	0.017118	1044
20	6,25	12.5	628724	369008	0,01059900	0.017425	1056

Figura 35

Consumo CPU evaluación Gamma



Nota: Consumo CPU etapa Gamma Secuencia / Paralelo. Elaborado por: Los autores.

3.4.3.4 Evaluación técnicas Circle Hough Secuencial / Paralelo. Al aplicar técnicas de paralelismo se puede evidenciar la diferencia de consumo de recursos como muestra la Figura

38 que representa el consumo de CPU, como se puede observar al paralizar tareas el consumo de recursos se optimiza, el tiempo de ejecución baja notoriamente como se representa en la Figura 37, por otra parte se puede notar que el consumo de memoria aumenta como consecuencia de aplicar técnicas de programación en paralelo ya que se comparte la memoria entre tareas que se ejecutan simultáneamente se puede evidenciar en la Figura 36.

Tabla 10

Resumen consumo recursos computacionales Circle Hough

Nombre	CPU Secuencial	CPU Paralelo	TIEMPO(S) Secuencial	Tiempo(s) Paralelo	Memoria(kb) Paralelo	MEMORIA(kb) Secuencial	Total Hilos Hough
1	48,529	12,89	4.91369	0.811826	621316	207616	463
10	50,000	13,19	4.47525	0.5800764	643356	185904	458
100	50,000	13,60	2.00923	1.0152454	726232	221872	438
101	50,000	13,24	5.30641	0.350264	637140	412568	427
102	50,000	12,37	5.71584	0.4937687	637864	217120	458
103	45,000	12,50	0.66226	0.1322312	638880	220816	461
11	50,000	11,82	4.48446	0.639795	631680	347600	460
12	50,000	12,98	4.40808	0.414129	640592	366632	461
13	47,500	14,51	2.94993	0.3077156	707616	249328	464
14	46,654	12,72	51.506.200	0.321308	626472	249328	460
15	50,000	11,14	3.90742	0.6036707	646000	436552	472
16	49,074	12,50	3.331.930	0.215614	720296	292096	466
17	50,000	11,80	4.309.860	0.921468	634392	214480	458
18	50,000	11,93	756.651	1.1828389	614628	361880	450
19	51,087	12,98	6.08603	1.071729	627960	371384	438
2	51,316	13,13	2.490.700	0.1911766	666592	371384	469
20	49,138	13,13	3.43959	0.2598376	618104	277312	464

La Tabla 10 muestra un resumen de los valores de consumo computacional que genera la técnica circle hough que marcar el circulo que forma el iris en la imagen de un ojo.

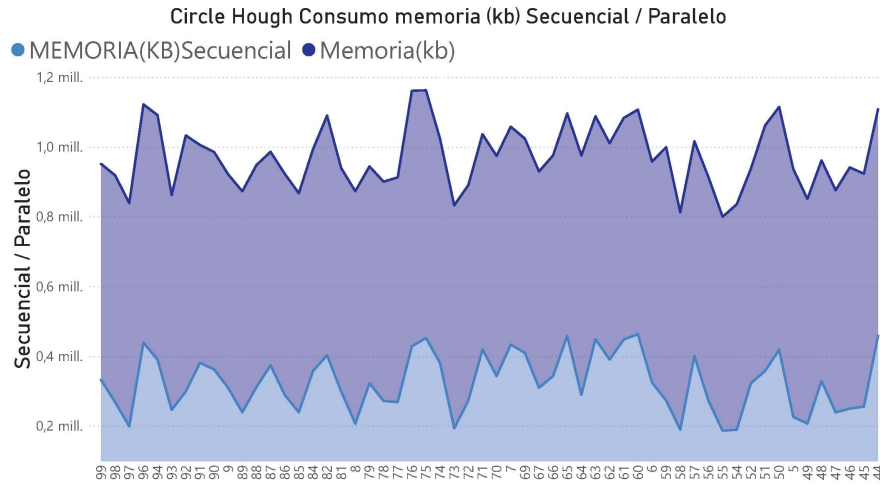
La Figura 36 indica como el consumo de memoria se eleva debido a los procesos en paralelo, dado que la técnica hace uso del mismo espacio de memoria para ejecutar las tareas simultáneamente.

La Figura 37 muestra como el tiempo de ejecución de un proceso disminuye gracias a que las tareas se ejecutan simultáneamente evitando el tiempo de espera que entre acciones repetitivas.

La Figura 38, muestra como el consumo de CPU disminuye al aplicar técnicas en paralelo para procesar tareas repetitivas, al enviar la carga de procesos a la memoria compartida.

Figura 36

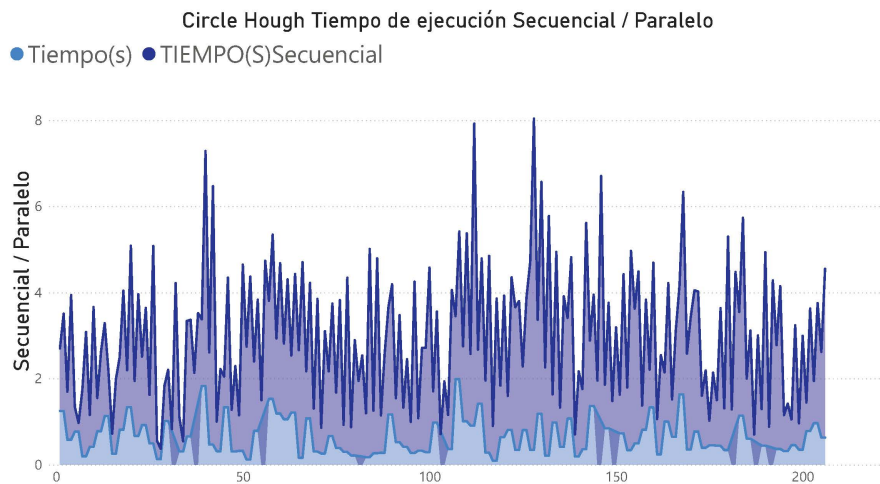
Consumo Memoria (kb) Circle Hough



Nota: Consumo Memoria (kb) Circle Hough Secuencia / Paralelo. Elaborado por: Los autores.

Figura 37

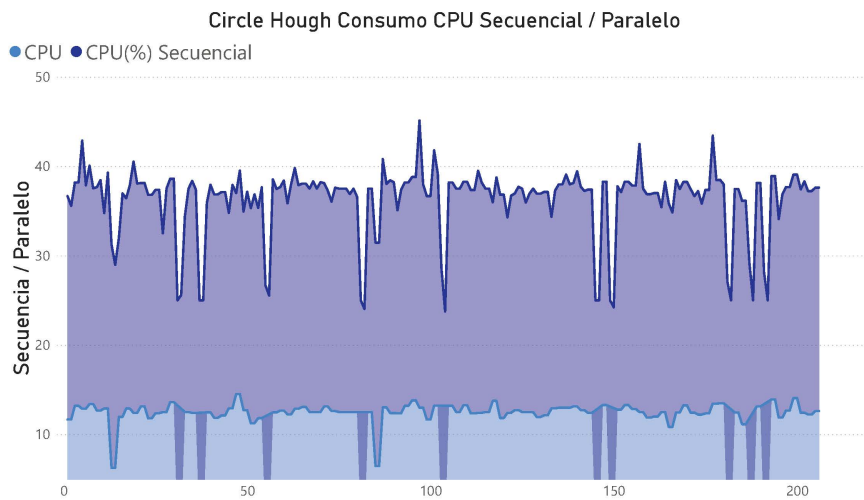
Consumo Tiempo (ms) Circle Hough



Nota: Consumo Tiempo (ms) Circle Hough Secuencia / Paralelo. Elaborado por: Los autores.

Figura 38

Consumo CPU Circle Hough especular



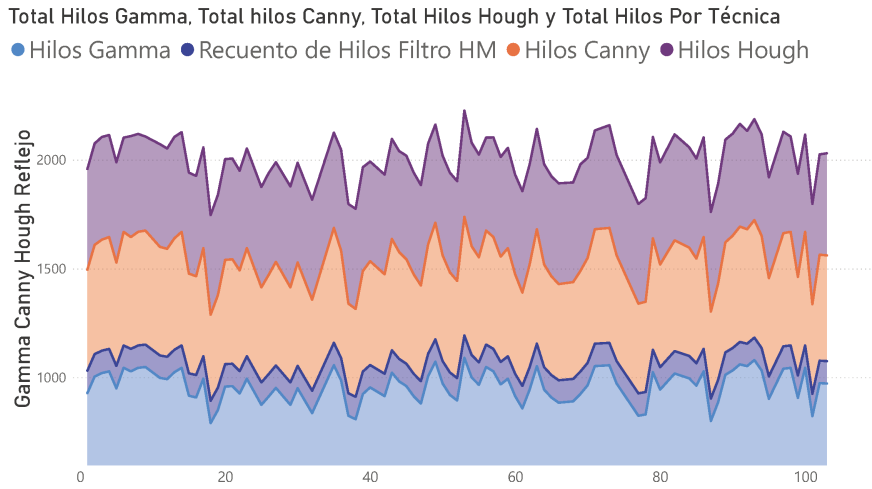
Nota: Consumo CPU Circle Hough Secuencia / Paralelo. Elaborado por: Los autores.

3.4.3.5 Cantidad de hilos por técnica. La Figura 39 representa la cantidad de hilos generados por cada técnica aplicada para el procesamiento de las imágenes, los hilos se crean dinámicamente y depende del tamaño de la imagen, ya que las dimensiones de la misma brindan la cantidad de hilos que el proceso genera para atender las tareas simultaneas que se despliegan, mejorando así el tiempo de respuesta de un proceso, el consumo de CPU tiende a bajar debido a que las tareas hacen uso de la memoria compartida para completar las tareas simultaneas se puede constatar que la técnica que identifica y trata el reflejo especular es la que genera mayor cantidad de hilos ya que realiza varias tareas haciendo uso de diversas técnicas para completar sus tareas.

La Tabla 11 muestra el resumen de la cantidad de hilos creados por cada técnica de procesamiento de imágenes, la cantidad de hilos creados dependerá de las dimensiones de las imágenes haciendo así que la creación de los sectores paralelos sean dinámicos, como se evidencia la

Figura 39

Total hilos creados por técnica



Nota: Total de hilos creados por técnica, los hilos se crean dependiendo de las dimensiones de las iamgens a procesar.. Elaborado por: Los autores.

Tabla 11

Resumen hilos creados por técnica

Nombre	Total Hilos Hough	Total Hilos Gamma	Total hilos Canny	Total Hilos Reflejo
1	463	960	480	6561
10	458	890	445	1089
100	438	856	428	225
101	427	1044	522	14161
102	458	1056	528	900
103	461	1048	524	676
11	460	968	484	1089
12	461	994	497	49
13	464	1022	511	1369
14	460	944	472	14161
15	472	1080	540	676
16	466	836	418	400
17	458	1056	528	900
18	450	1040	520	729
19	438	972	486	441
2	469	1044	522	676
20	464	1056	528	676

técnica del reflejo crea la mayor cantidad de hilos, puesto que el proceso hace uso de diferentes técnicas.

3.4.3.6 Comparación evaluación de técnicas. Los resultados promedios como muestra la Tabla 12 se puede identificar claramente el promedio de cada técnica aplicada, como muestra la Figura 40 el consumo de CPU baja drásticamente al igual que el tiempo de ejecución como se evidencia en la Figura 41, caso contrario de lo que sucede con el consumo de memoria como se indica en la Figura 42 esta tiende a subir por cada proceso que se ejecuta en paralelo dado que se trabaja con memoria compartida.

Tabla 12

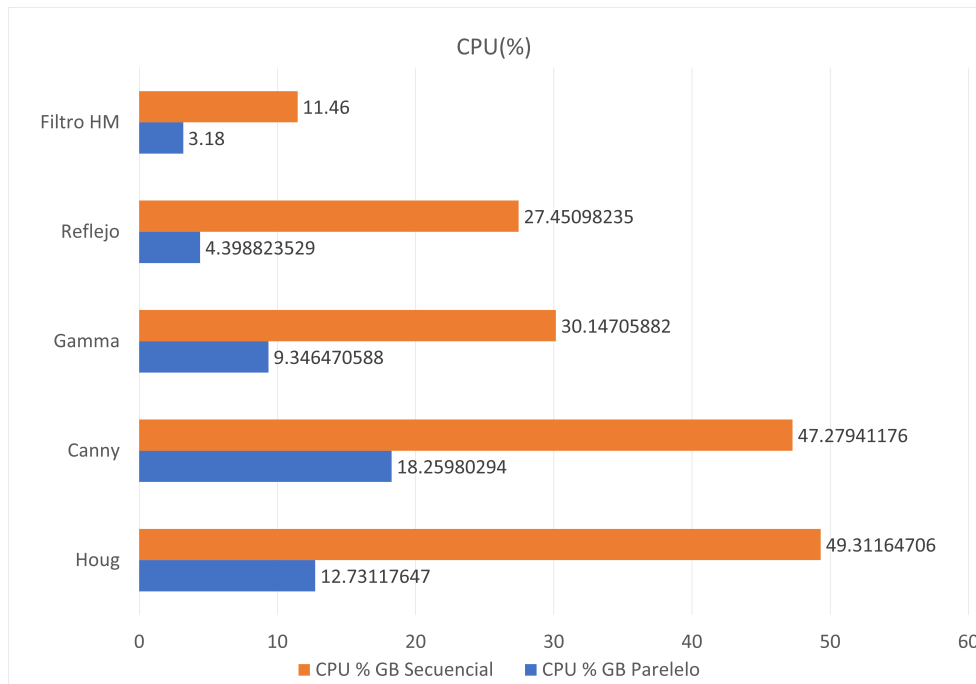
Resultados de la evaluación de las técnicas

	CPU % GB		Memoria(kb)		Tiempo(s)	
	Parelelo	Secuencial	Parelelo	Secuencial	Parelelo	Secuencial
Houg	12.7311765	49.3116471	649360	294345.412	0.55957024	5.15062
Canny	18.2598029	47.2794118	648171.294	289731.059	0.01855829	0.05729918
Gamma	9.34647059	30.1470588	647622.824	289839.765	0.01012715	0.01621935
Reflejo	4.39882353	27.4509824	324253.176	146722.353	1.81202941	2.27400529
Filtro HM	3.183425	11.462356	526576.34	2380034.12	2.4215432	5.4143252

Nota: Resultados de la evaluación de las técnicas de procesamiento de imágenes CPU, Memoria, Tiempo de ejecución. Elaborado por: Los autores.

Figura 40

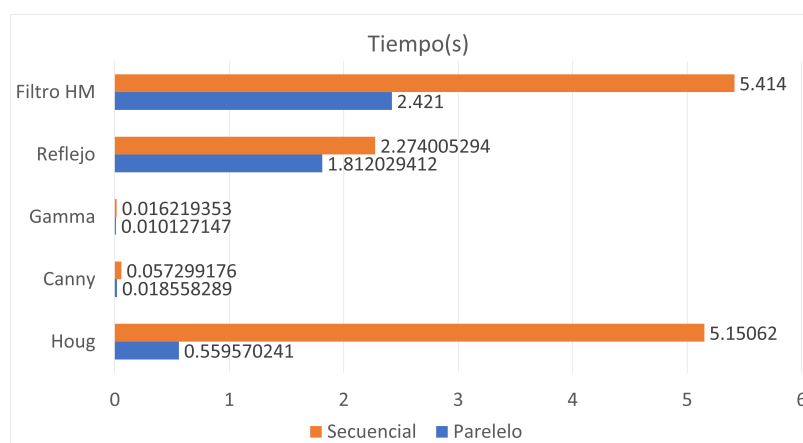
Comparación consumo de CPU secuencial / paralelo



Nota: Comparación del consumo de cpu de las técnicas de procesamiento de imagenes Secuencia / Paralelo. Elaborado por: Los autores.

Figura 41

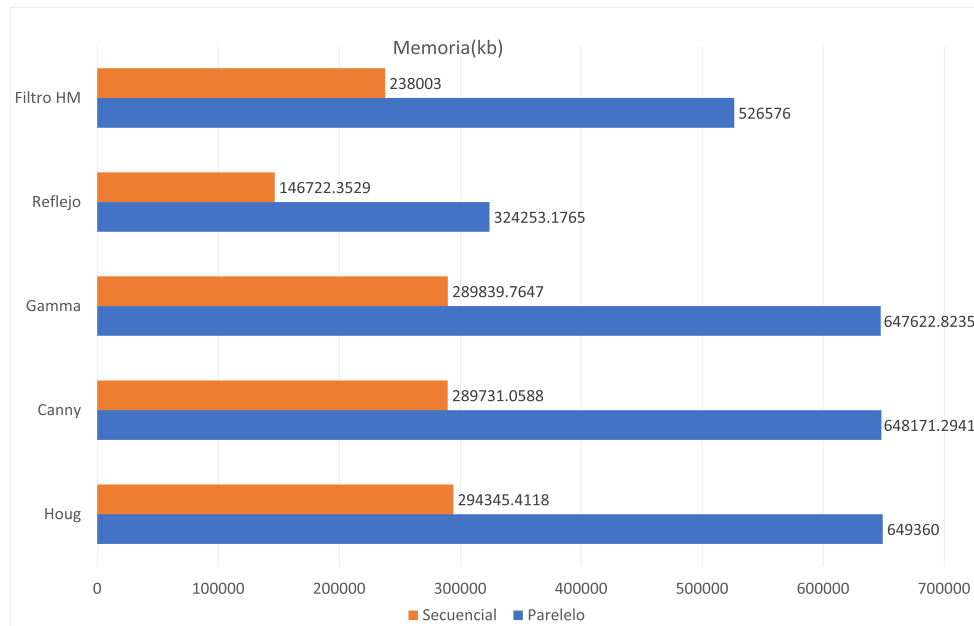
Comparación consumo de tiempo secuencial / paralelo



Nota: Comparación del consumo de tiempo de las técnicas de procesamiento de imagenes Secuencia / Paralelo. Elaborado por: Los autores.

Figura 42

Comparación consumo de memoria secuencial / paralelo



Nota: *Comparación del consumo de memoria de las técnicas de procesamiento de imágenes Secuencia / Paralelo. Elaborado por: Los autores.*

Como resultado el cálculo del porcentaje de cambio que existe entre la programación secuencial y la paralela como muestra la Tabla 13, los valores en negativo representan que existe un consumo mayor en comparación con la ejecución en paralelo del recurso, ya que para el procesamiento de tareas en paralelo la memoria tiende a trabajar el doble de lo habitual por la cantidad de peticiones que cada tarea realiza al espacio de memoria en el que almacena la información.

Queda demostrado que el consumo de CPU baja en promedio 72.13% al aplicar todas las técnicas, el Tiempo de ejecución disminuye en un 53.65% al ejecutar la aplicación usando técnicas de programación en paralelo, puesto que los procesos se realizan de forma simultánea por ende de manera más rápida.

El consumo de memoria se ve afectado dado que se trabaja con un paradigma de paraleli-

Tabla 13*Porcentaje de cambio de las técnicas*

	Porcentaje de cambio		
	CPU	Memoria	Tiempo
Houg	0.74182212	-1.20611558	0.89135866
Canny	0.61378955	-1.23714812	0.67611595
Gamma	0.68997073	-1.23441674	0.37561337
Reflejo	0.83975715	-1.20997803	0.20315515
Filtro HM	0.69214198	-1.2132135	0.50213414
Promedio	0.72133489	-1.22191462	0.53656078

Nota: Porcentaje de cambio entre las técnicas de procesamiento de imágenes. Elaborado por: Los autores.

zación de memoria compartida que da como consecuencia que todas las tareas que se ejecutan simultáneamente hagan uso del mismo espacio de memoria saturándola de peticiones y aumentando su uso en promedio de 122.19% como muestra la Tabla 13 en su apartado de promedios.

CONCLUSIONES

- Los métodos de programación paralela aplicados en el proyecto ayudaron a optimizar recursos computacionales disponibles, ya que al aprovechar de toda la capacidad del dispositivo se puede acortar tiempos de espera entre procesos en 53 % y el consumo de CPU en 72 % debido a la manipulación de tareas repetitivas que se pueden ejecutándolas simultáneamente sin afectar a otros procesos con ayuda de la sincronización de la directiva OPM.
- Sincronizar más de tres tareas que utiliza una variable compartida afecta al porcentaje de memoria que consume, aumentando en un 122 % el consumo de memoria que dedica para llevar la tarea acabo, por tal motivo si un proceso depende del resultado de una tarea que se ejecuta en paralelo este tendrá que esperar a que termine su ejecución para liberar la variable que se comparte.
- Crear dinámicamente los hilos que requiere un proceso asegura que los recursos sean utilizados al máximo, ya que no están restringidos a un número específico de hilos que puede hacer uso, esta actividad mejora el rendimiento si los hilos no sobrepasan de los diez mil, puesto que al crear una cantidad excesiva de hilos el procesador colapsara al tener que atender tareas simultaneas.
- Al evaluar las técnicas aplicadas permitió identificar y clasificar procesos que pueden ser mejorados en desempeño y optimización del consumo de recursos computacionales,

como es el método de reflejo especular que presenta mayor consumo de recursos en la aplicación cuando tiene una ejecución secuencial, pero cuando se lo paraleliza el consumo de CPU baja notoriamente dando como consecuencia que el tiempo de espera se baja a la mitad.

- EL generar al observar los resultados se puede evidenciar que en consumo de CPU disminuye al aplicar técnicas de paralelización, al igual que el tiempo de ejecución, pero no es el caso del consumo de memoria(kb), ya que al compartir variables o espacios de memoria a todos los hilos que genera un proceso, esta tiene que atender varias peticiones que llegan simultáneamente a un espacio de memoria dando como resultado la saturación de memoria.

RECOMENDACIONES

- Para mejorar el rendimiento de los algoritmos se sugiere adaptar el código fuente de la técnica reflejo especular con la menor sincronización de variables compartidas, además de la posibilidad de trabajar los bucles o iteraciones con lenguajes como Fortran o ensamblador para mejorar el tiempo de ejecución y aprovechar los núcleos del CPU al máximo.
- Se recomienda reemplazar la técnica de detección de círculos de Hough por técnicas de mejor rendimiento en detección de iris como lo es la transformada de Dougman. Por alto consumo computacional como es
- Se recomienda identificar técnicas que contengan ciclos de repetición que modifiquen el valor de una variable repetitivamente, para descomponer el problema en tareas más cortas este proceso ayuda a aplicar técnicas en paralelo con memoria compartida.
- Para evitar que el tiempo de espera entre procesos incremente es necesario evitar paralelizar procesos que dependan del resultado de otra tarea, ya que la sincronización hace que se aumente el tiempo y el consumo de recursos computacionales.
- Las imágenes con dimensiones mayores a (400 x 400) mejora el tratamiento he identificación del iris, por tal motivo se recomienda hacer uso de imágenes que superen estas dimensiones.

REFERENCIAS

- Aguilar, J., Leiss, E., y cols. (2004). *Introducción a la computación paralela*. Venezuela: *Gráficas Quinteto*.
- Alcántara-Montiel, C. V., Ortega, J. C. P., Arreguín, J. M. R., Hurtado, E. G., Tovar-Arriaga, S., & Soto, J. E. V. (2019). Detección efectiva de rostros en imágenes utilizando descriptores basados en hog. *Res. Comput. Sci.*, *148*(8), 371–385.
- Aldape-Pérez, M., Yáñez-Márquez, C., Camacho-Nieto, O., & Ferreira-Santiago, Á. (2013). Feature selection using associative memory paradigm and parallel computing. *Computación y Sistemas*, *17*(1), 41–52.
- Aslla, A., Montalvo, R., & Rivera, P. (2013). Revisitando la programación en paralelo.
- Briones Gárate, E. A. (2020). *Sistema web de reconocimiento facial para control de acceso biométrico, utilizando inteligencia artificial* (Tesis de Master no publicada). ESPOL. FIEC.
- Buemi, M. E. (2012). *Tratamiento de imágenes de radar de apertura sintética mediante filtros stack* (Tesis Doctoral no publicada). Universidad de Buenos Aires. Facultad de Ciencias Exactas y Naturales.
- Chalén Pacay, B. F., & Camacho Girón, J. C. (2020). *Desarrollar un prototipo de reconocimiento facial basado en machine learning para detectar estado de somnolencia en conductores de una cooperativa de transporte*. (Tesis Doctoral no publicada). Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas
- CMake. (s.f.). Descargado 2022-02-07, de <https://cmake.org/>
- Fernández, J., Fuentes, M., Piccoli, M. F., & Printista, A. M. (2016). Un modelo de programación paralelo para memoria distribuida. En *Iv workshop de investigadores en ciencias de la computación*.

- FERNÁNDEZ, Y. (10 de marzo 2021). *Xataka* (Inf. Téc.). Descargado de <https://www.xataka.com/basics/cpu-que-como-sirve>
- FERNÁNDEZ, Y. (17 de marzo 2021). *memoria-ram-que-sirve-como-mirar-cuanta-tiene-tu-ordenador-movil* (Inf. Téc.). Descargado de <https://www.xataka.com/basics/memoria-ram-que-sirve-como-mirar-cuanta-tiene-tu-ordenador-movil>
- Garcés Núñez, A. F. (2017). *Sistema de reconocimiento facial con visión artificial para apoyar al ecu 911 con la identificación de personas en la lista de los más buscados* (B.S. thesis). Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas
- García López, F. C., y cols. (2004). Programación en paralelo y técnicas algorítmicas.
- Gary J. Bronson, J. A. G. (2006). *Problemas mediante c++*. Descargado de [https://ayudasingenieria.dotz.com.ar/files/INFORMATICA/\[LIBRO\]%20Bronson%202ed.pdf](https://ayudasingenieria.dotz.com.ar/files/INFORMATICA/[LIBRO]%20Bronson%202ed.pdf)
- HERNÁNDEZ REYES, J. C. (2016). Autenticación biométrica a través de huellas digitales e iris en una empresa industrial.
- Ismael Farinango, L. S. (2021). Desarrollo de un módulo prototipo experimental móvil que permita la autenticación biométrica utilizando el iris del ojo humano. *Universidad Politécnica Salesiana sede quito*.
- King, D. E. (2009). Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10, 1755-1758.
- Kostiantyn S. Khabarlak, L. S. K. (2015). Fast facial landmark detection and applications: A survey. *Dnipro University of Technology, Dnipro, Ukraine*.
- Mery, D. (2004). Visión por computador. *Santiago de Chile. Universidad Católica de Chile*.
- Michael Mukovoz Studio. (15 de Abril de 2018). <http://www.librow.com/articles/article-8> (Inf. Téc.).

- OpenMP. (2013). *OpenMP Application Program Interface*. OPENMP.
- Ortiz Rangel, E., Mejía-Lavalle, M., & Sossa, H. (2017). Filtrado de ruido gaussiano mediante redes neuronales pulso-acopladas. *Computación y Sistemas*, 21(2), 381–395.
- Paul Cucu, A. D. (2018). Textural Feature Based Classification of Mammogram Images Using ANN. *2018 9th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2018*, 10–15. doi: 10.1109/ICCCNT.2018.8493957
- Pérez Lescano, H. V. (2018). *Sistema de control de acceso por reconocimiento de iris para el ingreso de personal a la empresa electroservicios querubín de la ciudad de puyo* (B.S. thesis). Universidad Técnica de Ambato. Facultad de Ingeniería en Sistemas
- Pinilla, C., Alcalá, A., & Ariza, F. (1997). Filtrado de imágenes en el dominio de la frecuencia. *Revista de la Asociación Española de Teledetección*, 8(8), 1–5.
- Romero, R. R., Fernandez, L. P. S., Riverón, E. M. F., & Hernández, J. J. C. (2019). Detección y rastreo de peatones empleando vectores de características de histogramas de gradientes orientados y patrones binarios locales en una máquina de soporte vectorial. *Res. Comput. Sci.*, 148(10), 81–94.
- Sarmiento-Ramos, J. L. (2020, jun). Aplicaciones de las redes neuronales y el deep learning a la ingeniería biomédica. *Revista UIS Ingenierías*, 19(4), 1–18. doi: 10.18273/revuin.v19n4-2020001
- Storey, G., Bouridane, A., & Jiang, R. (2018). Integrated deep model for face detection and landmark localization from “in the wild” images. *IEEE Access*, 6, 74442–74452.
- techlandia. (20 de enero de 2020). *Hilos de un procesador* (Inf. Téc.).
- Valencia-Murillo, J. F., Poveda-Sendales, D. A., & Valencia-Vargas, D. F. (2014). Evaluación del impacto del preprocesamiento de imágenes en la segmentación del iris. *TecnoLógicas*, 17(33), 31–41.