PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

FORMACIÓN DE IMAGEN COMPLETA DE UNA PÁGINA CON TEXTO IMPRESO MEDIANTE PROCESAMIENTO DE IMÁGENES OBTENIDAS DE UN VIDEO

Tesis para optar por el título profesional de Ingeniero Electrónico, que presenta el bachiller:

Ramírez Díaz, José Fernando

ASESOR:

MSc. Ing. Pedro Moisés Crisóstomo Romero

Lima, agosto de 2020

Resumen

En la presente tesis se aborda el diseño e implementación de un algoritmo que permite formar la imagen completa de un documento con texto impreso partiendo de un video que contiene fragmentos de la página en cuestión. Dicho algoritmo recibe como entrada un video registrado empleando la cámara de un teléfono móvil y como resultado retornará la imagen del documento con texto completo; esta imagen puede ser empleada posteriormente en un algoritmo de reconocimiento óptico de caracteres (u OCR por sus siglas en inglés) para recuperar el texto en forma digital.

El enfoque del desarrollo de esta propuesta es el de brindar una solución alternativa, en cuanto a adquisición de imágenes, para las existentes aplicaciones móviles de OCR enfocadas en apoyar a personas con ceguera parcial o total.

Para abarcar el planteamiento y cumplimiento de los objetivos de este proyecto, se ha estructurado el mismo en 4 capítulos. En el capítulo 1 se aborda la actual situación de personas con distintos grados de discapacidad visual en nuestro país y diversos sistemas que buscan apoyarlos en recuperar su autonomía informativa y educativa. Además, se trata detalles sobre el estado del arte en adquisición de imágenes para las aplicaciones OCR existentes en la actualidad y sus falencias. En el capítulo 2 se presenta el marco teórico que avala el desarrollo del algoritmo propuesto, desde la teoría necesaria en procesamiento de imágenes y, también, sobre el registro de vídeos. En el capítulo 3 se trata el diseño e implementación del algoritmo en dos plataformas: inicialmente en Python 3.6 para la etapa de calibración de parámetros en una computadora de escritorio, y en C++ para las pruebas finales en un teléfono con SO Android. En dicho capítulo también se hace presente consideraciones planteadas para la creación del conjunto de videos de pruebas en Python. Finalmente, en el capítulo 4 se exponen las pruebas y resultados obtenidos de la aplicación del algoritmo, en Python, sobre la base de muestras creadas, y los resultados finales del uso de la aplicación en Android. Para estimar el grado de conformidad de la imagen resultante se hará uso de la métrica de Levenshtein o distancia de edición, la cual señala cuántos caracteres detectados en la imagen compuesta son diferentes a los caracteres del texto original.

Dedico este trabajo a toda mi familia,
quienes acompañan y apoyan cada uno de mis pasos.
A mis padres y abuelos,
por demostrar que existen fuerzas capaces de mover montañas.

Contenido

Índice de figuras

Índice de tablas

Introducción

Actualmente en el Perú existen más de 800,000 personas que padecen de algún tipo de limitación visual que les dificulta leer, incluso usando lentes, lo cual ha creado una brecha educativa en esta parte de la población. Esto es debido a que el acceso a la información impresa para estas personas se ve limitado por la escasa tecnología y metodología disponible en centros educativos y otras instituciones destinadas a apoyarlos en estas tareas formativas.

Para aliviar esta situación se han desarrollado diversas soluciones, tanto electrónicas como manuales, que se enfocan en devolverles la autonomía necesaria para informarse y educarse. De las alternativas electrónicas existentes se destacan aquellas que aprovechan las características de los teléfonos móviles, ya que este dispositivo sigue mejorando tecnológicamente y, por disminución en precios, es accesible para todos los sectores económicos en nuestro país.

Específicamente, las aplicaciones móviles de reconocimiento óptico de caracteres (OCR, por sus siglas en inglés) son herramientas con mucho potencial, dado que les facilitan recuperar cierta autonomía al permitirles escuchar el contenido de la imagen capturada de un documento que contiene texto. Sin embargo, este tipo de aplicaciones aún se enfrentan a un reto de funcionalidad: facilitar al usuario la captura de la imagen más nítida posible de una página y que el texto impreso contenido en ésta se encuentre completo para realizar una adecuada conversión de imagen a voz sintética.

Por esta razón, en la presente tesis se plantea como objetivo el diseño e implementación de un algoritmo que permita reconstruir la imagen completa de un documento partiendo de un vídeo que contiene fragmentos de éste para brindar una solución alternativa, en cuanto a adquisición de imágenes, a este tipo de aplicaciones.

Capítulo 1
Situación actual de las personas con limitaciones visuales en el Perú

De acuerdo al Instituto Nacional de Estadística e Informática (INEI), en el Perú existen actualmente más de 1.5 millones de personas que padecen de algún tipo de discapacidad que limita uno o más de sus sentidos y que también pueden llegar a afectar sus capacidades motoras o mentales. En la Encuesta Nacional Especializada sobre Discapacidad (ENEDIS), realizada por esta institución en el año 2012, se puso en evidencia un hecho poco alentador referente a la educación a la que pueden acceder estas personas: sólo el 11.6% pudo alcanzar el nivel superior de estudios y el 22.3% ni siquiera cuenta con el nivel de educación inicial [1]. Esta realidad se refleja en la Figura 1.1, conforme aumenta el nivel de los estudios, decrece el número de personas discapacitadas que pudieron acceder a dichos grados ya sea debido a factores económicos, sociales o por el hecho de que muy pocas instituciones cuentan con la metodología y tecnología necesaria para facilitarles el servicio educativo.

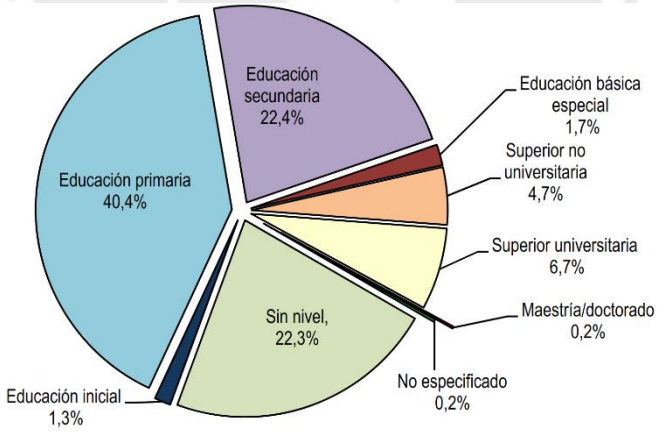


Figura 1.1: Personas con discapacidad de 3 años a más de edad, según nivel educativo, 2012 [1].

El porcentaje más vulnerable ante estas carencias es el de personas con algún tipo de discapacidad visual, pues en la ENEDIS se estima que a nivel nacional existen más de 800,000 personas (51% del total de personas con discapacidad) en estas condiciones. Estas personas padecen de un tipo de limitación permanente que afecta su capacidad para ver incluso usando lentes; dichas limitaciones pueden ser originadas por una enfermedad congénita, un problema genético u otro factor [1]. Sus vidas se ven condicionadas por estas dificultades en su salud ya que reducen su autonomía y les hacen más dependientes de otras personas o herramientas para poder realizar actividades

cotidianas [2]. En el ámbito de la educación y acceso a la información, su desarrollo se ve restringido por la falta de accesibilidad de nuestro entorno, pues no se les brinda las facilidades necesarias para que puedan lograr sus objetivos al igual que sus congéneres.

En vista a ello, muchos investigadores a nivel mundial han diseñado e implementando software que les facilita el acceso a la información impresa, nuestra principal fuente de conocimiento, a las personas con ceguera parcial o total, para contribuir en la normalización del nivel de oportunidades de desarrollo personal mediante el acceso a la información y la educación. El acceso a estas herramientas informáticas se ha masificado gracias a la expansión tecnológica de la que es partícipe nuestro país [3]; sin embargo, el uso de estas se ve limitado por falta de información o por factores económicos. En base a lo previamente mencionado, hace evidente la necesidad de contar con una herramienta funcional que les facilite el acceso a la información impresa de la manera más fácil posible y que sea económicamente asequible para la mayoría de potenciales usuarios.

1.1. Sistemas de apoyo en la lectura para personas con limitaciones de carácter visual

En la actualidad ya existen medios que permiten leer a las personas con limitaciones visuales, estos sistemas apelan al sentido del oído o tacto de los usuarios (en algunos casos apoyan a su visión remanente) para brindarles la información que desean. Estos se pueden dividir en dos clases: manuales y electrónicos. Los sistemas electrónicos poseen mayor potencial dado que les permiten acceder a casi cualquier información contenida en un medio impreso; sin embargo, acceder a sistemas embebidos especializados en esta tarea puede resultar, especialmente en zonas rurales, muy costoso. Para estas situaciones, existen otras opciones de menor costo en forma de programas que se pueden ejecutar en los dispositivos disponibles (computadoras, smartphones, tabletas, etc.) o, por lo general, se hace uso del sistema de lectura y escritura braille (medio manual). Este último es el más empleado dentro de nuestro sistema educativo dado que los materiales requeridos para su enseñanza son muy accesibles; sin embargo, la cantidad de información disponible escrita en braille es significativamente limitada en comparación con la que emplea grafías impresas.

A continuación, se hace un breve repaso sobre los diversos sistemas de apoyo en la lectura para personas con limitaciones visuales.

1.1.1. Sistema de lectura y escritura Braille

El sistema Braille fue ideado en 1825 por Louis Braille para brindarle la posibilidad de leer y escribir a personas con problemas visuales, mediante el sentido del tacto. Louis, ciego desde los 13 años, derivó este método a partir de un sistema de lectura militar que empleaba puntos en alto relieve para registrar y proteger información clasificada.

Este sistema emplea arreglos de 6 puntos que se pueden combinar de distintas formas, 64 en total, para representar un determinado carácter. En la Figura 1.2 se aprecia la distribución de los puntos y arreglos que resulta más accesible para las personas.

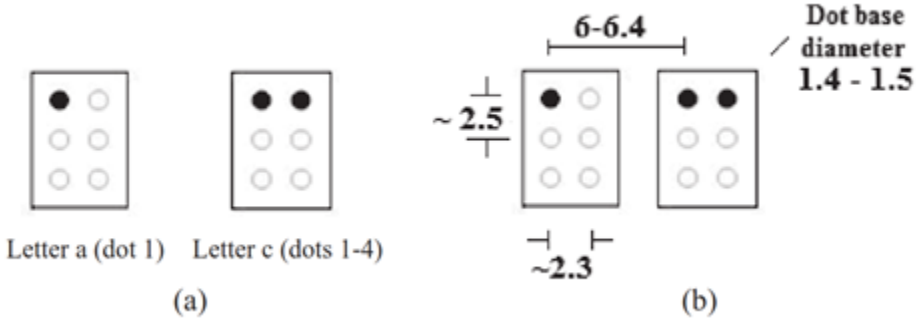


Figura 1.2: (a) Letra ‘a’ y letra ‘c’ representadas en Braille, (b) reglas del formato de símbolos Braille [6].

A nivel mundial es el sistema de apoyo más difundido y por ello se ha empleado para mejorar la accesibilidad de gran parte de la tecnología actual (computadoras de escritorio, teléfonos celulares, TV, etc.). También se ha empleado para desarrollar dispositivos completamente enfocados en asistir a estas personas. Por ejemplo, las líneas Braille (Figura 1.3) transforman el contenido de la pantalla de un computador al sistema Braille para facilitarles el acceso a la información [4] [6] [12].

Por otro lado, a pesar de la presencia global de este sistema, el porcentaje de personas discapacitadas que lo emplea es relativamente bajo. En el año 2012, se estimó que en el Perú tan solo 50,000 personas (6% del total de personas con limitaciones visuales) empleaban el sistema Braille para acceder a la información [5].

Figura 1.3: Línea Braille de la marca Focus Blue [12].

1.1.2. Sistemas embebidos

Actualmente existen muchos dispositivos electrónicos especializados que facilitan el acceso a la información a personas con limitaciones visuales. Estos se pueden clasificar en 3 clases:

- **Máquinas lectoras**

Son dispositivos que digitalizan un medio impreso a través de cámaras o escáneres y luego aplican algoritmos de reconocimiento óptico de caracteres (OCR por sus siglas en inglés) para extraer el texto de estas imágenes en forma digital. Una vez obtenido el texto digitalizado, los dispositivos pueden realizar diversas tareas para transmitir la información al usuario; ya sea como audio reproducido a través de altavoces integrados, como un archivo exportable para ser editado o reproducido en una computadora, o como una lectura en líneas Braille.

Figura 1.4: Algunos sistemas embebidos de apoyo en la lectura [13].

En la Figura 1.4 se presenta algunas de las máquinas lectoras existentes en el mercado actualmente. En (a) se aprecia al Eye-Snap Reader, emplea una cámara integrada ubicada en un soporte para

obtener la imagen del documento y posee altavoces integrados para reproducir el contenido digitalizado. En (b) tenemos al Eye-Scan Reader, una máquina tipo escáner que también posee altavoces integrados para reproducir la información. Finalmente, en (c) podemos observar al Scanning Pen, un dispositivo que se asemeja a un lápiz y está diseñado para personas con baja visión. Al pasar el escáner ubicado en uno de sus extremos sobre una palabra impresa, esta se reproduce en voz alta a través de su altavoz.

Si bien estos dispositivos brindan las facilidades necesarias para su uso y realizan sus tareas con gran precisión, sus limitadas funciones y sus elevados costos hacen que la presencia de estos en nuestro país sea bastante limitada y se encuentren sólo en contadas instituciones.

- **Lupas electrónicas**

Estos dispositivos están diseñados para personas cuya vista no esté gravemente afectada dado que sirven para ampliar el texto impreso que el usuario desee leer. El funcionamiento de estos instrumentos consta de tres etapas: primero, una cámara integrada al sistema digitaliza el documento que el usuario desea leer; luego, se procesa las imágenes y se amplían digitalmente; finalmente, se envían los resultados a una pantalla que puede ser externa o integrada al sistema. En la Figura 1.5 se aprecia una lupa electrónica de la marca Zoomax.

Figura 1.5: Lupa electrónica Zoomax [13].

Estos dispositivos están siendo reemplazados en el mercado por aplicaciones para smartphones dado que estos últimos cuentan con el hardware y software necesario para poder realizar la misma tarea y a un menor costo.

- **Sistemas de información digital accesible (DAISY)**

Estos sistemas están enfocados principalmente en brindar a los usuarios una vasta colección de información en forma de audiolibros que respetan el estándar DAISY. Además, algunos dispositivos añaden las funcionalidades de los equipos listados previamente.

1.1.3. Programas para ordenador

Hoy en día casi todos los sistemas operativos para computadoras de escritorio cuentan con sus propias características de accesibilidad para facilitar su uso a personas con ceguera parcial o total. Por ejemplo, el sistema operativo Windows posee una aplicación que permite ampliar el contenido de la pantalla hasta 16 veces su tamaño original para ayudar a las personas con baja visión; también, cuenta con un lector de pantalla que permite al usuario identificar el contenido del monitor mediante un ayudante de voz. Por su parte, los sistemas operativos desarrollados por Apple también cuentan con aplicaciones semejantes para brindar mayor accesibilidad.

Además de las características integradas en estos sistemas, diversos desarrolladores continúan brindando software (puede ser de pago o gratuito) que amplía las funcionalidades de acceso del ordenador.

Lo mejor de estas aplicaciones es que pueden realizar casi todas las tareas que realiza un sistema embebido a un menor costo y requiriendo poco o ningún hardware adicional. En la tabla 1.1 se encuentran algunos programas para ordenador y sus características.

Tabla 1.1: Programas para ordenador de apoyo en la lectura [20] [21].
Elaboración propia.

Programa	Desarrollador	Funciones	Plataforma	Licencia	Costo
JAWS	Freedom Scientific	Lector de pantalla	Windows	Comercial	$ 179 - $ 1100
VoiceOver	Apple Inc.	Lector de pantalla	Mac OS X, iPhone, iPad, iPods and Apple TV	Libre para todos los productos Apple	Sin costo
ZoomText	Ai Squared	Lector de pantalla y amplificador de texto	Windows	Comercial	$ 100 - $ 600
Dragon	Nuance Communications	Elaboración de documentos mediante voz	Windows, Mac OS X	Comercial	$ 75 - $ 500
NVDA	NonVisual Desktop Access Project	Lector de pantalla	Windows	Libre y de código abierto	Sin costo
Orca	The GNOME project	Lector de pantalla y amplificador de texto	GNU/Linux	Libre y de código abierto	Sin costo

Estas herramientas de software son una muy buena opción de apoyo dado que disponen de un mayor poder de procesamiento para obtener resultados más rápidos y precisos; sin embargo, su uso se ve limitado por el mismo hecho de que menos personas disponen de un ordenador en comparación con las que disponen de un dispositivo móvil [3] [7].

1.1.4. Aplicaciones para teléfonos inteligentes

Durante la última década, los smartphones se han expandido a gran velocidad en todo el mundo y al mismo tiempo han continuado evolucionando para brindar una mejor experiencia a los usuarios. Actualmente, todos los sistemas cuentan con funciones integradas de accesibilidad (ayudante de voz, lupa, conversión de voz a texto, lector de pantalla, etc.) y también tienen la opción de instalar software adicional para realizar tareas complementarias como: extraer texto de imágenes y leerlo en voz alta, detectar fuentes de luz, guiar al usuario al caminar, etc.

El hecho de que estos teléfonos sean portátiles, y cuenten con el hardware necesario, ha permitido crear aplicaciones enfocadas en devolverle su autonomía a personas con discapacidad visual; por ejemplo, en el área de acceso a la información se han desarrollado muchas aplicaciones que hacen uso de la cámara y los altavoces del dispositivo para convertir a dicho móvil en una máquina lectora de bolsillo. Algunas de las aplicaciones enfocadas en esta tarea se listan en la tabla 1.2.

Tabla 1.2: Aplicaciones para smartphones de apoyo en la lectura [16].
Elaboración propia.

Aplicación	Plataforma	Número de descargas	Costo	Valoración de los usuarios	Principal problema
TextGrabber	iOS/Android	Más de 50,000	$ 5.00	4.3/5.0	Errores en el texto digitalizado
KNFB reader	iOS/Android	Más de 10,000	$ 100.00	4.2/5.0	Su costo es muy elevado
Prizmo	iOS	-	$ 10.00	3.6/5.0	No captura el texto completo
Text Detective	iOS/Android	Más de 10,000	Gratuito	3.1/5.0	No captura el texto completo
SayText	iOS	-	Gratuito	2.7/5.0	No captura el texto completo
Talking Camera Pro	iOS/Android	Más de 100	$ 2.00 - $ 3.00	2.5/5.0	Serios errores en el texto digitalizado
Voice	iOS	-	Gratuito	2.5/5.0	No captura el texto completo

En base a este repaso sobre las tecnologías existentes, podemos destacar que el smartphone resulta ser la plataforma más conveniente para el desarrollo de herramientas de apoyo en la lectura para

personas con discapacidad visual. Esto se debe a que el aumento en la distribución de estos dispositivos móviles se ha incrementado considerablemente en todos los sectores económicos del país [3] y se espera que este continúe en los próximos años, lo cual brinda la posibilidad de dar acceso a esta herramienta a una mayor cantidad de personas con discapacidad visual. Además, este sistema móvil posee mayor funcionalidad que un sistema embebido y, a diferencia de un ordenador, es fácilmente portable.

1.2. Estado del arte en adquisición de imágenes para sistemas de apoyo en la lectura

La principal falencia de las aplicaciones OCR para smartphone, como se hizo presente en la tabla 1.2, es que los métodos empleados por la mayoría de estas no le brindan al usuario las facilidades necesarias para que obtenga la imagen completa de la página con texto y esto causa errores cuando se convierte el texto digital a voz sintética [9]. Para reducir este tipo de problemas, cada vez más desarrolladores buscan implementar en sus aplicaciones alguno de los métodos listados a continuación:

1.2.1. Construcción de un mosaico de documento

Es el proceso mediante el cual se construye la imagen completa de un documento partiendo de varias imágenes que tan solo contienen segmentos de este. Estos segmentos poseen puntos característicos (feature points) que contienen información que coincide con la de otros; esta coincidencia de datos (matching) es la que permite superponer los segmentos para crear la imagen completa (los métodos de detección de puntos y emparejamiento de estos se tratarán a profundidad en el siguiente capítulo).

Este proceso consta de dos etapas como se describe en [8] [17] [28]: en la primera etapa se obtienen las imágenes con los segmentos del documento, estas imágenes se pueden adquirir a partir de escaneos o de fotografías. Las imágenes registradas mediante fotografías son más susceptibles a tener variaciones de iluminación y perspectiva en comparación con una escaneada, es por ello que se les aplica las correcciones apropiadas antes de pasar a la segunda etapa. En la segunda etapa se procede a superponer los segmentos mediante la aplicación del algoritmo adecuado con la finalidad de obtener una imagen completa, nítida y precisa. En la Figura 2.1 se aprecia el desarrollo de este proceso: (a) y (b) son los segmentos adquiridos mediante fotografías, (c) y (d) son los segmentos

con correcciones de perspectiva e iluminación y en (e) se aprecia la imagen completa reconstruida. Este proceso también es aplicable cuando se desea componer la imagen completa de un documento grande (un póster, por ejemplo) o de un documento que refleje la luz (como un documento de identidad), pero para estos casos se puede obtener mejores resultados si los fragmentos de imagen se extraen de los cuadros (*frames*) de un vídeo [18]. Trabajos previos como el presentado por Zandifar et al. [11] o Nakajima et al. [19] emplean cámaras digitales para grabar un vídeo del documento y componer el mosaico mediante el procesamiento de los *frames* extraídos de este; sin embargo, la creación de mosaicos empleando smartphones es un terreno relativamente inexplorado. Concursos como el SmartDoc 2017 Video Capture [18] tienen como objetivo promover avances en esta área del procesamiento de imágenes para análisis de documentos dado que existe gran potencial de desarrollo en esta aplicación.

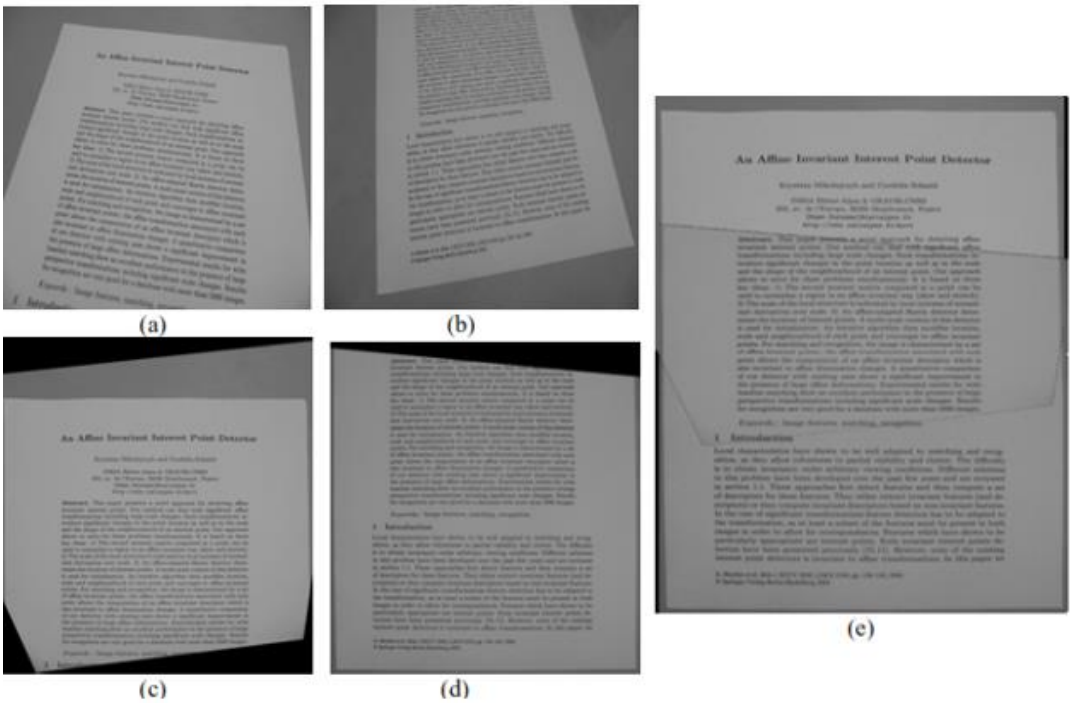


Figura 1.6: Proceso de Mosaico de documento. (a), (b), (c) y (d) son imágenes con secciones del documento. (e) es el mosaico compuesto [17].

1.2.2. Modalidades de interacción con el usuario

De acuerdo a Cutter y Manduchi [10], se puede identificar tres modalidades de interacción con el usuario que emplean las aplicaciones de OCR actualmente y las clasifican de la siguiente forma: modo manual, modo de auto disparo y modo guiado.

- **Modo manual**

En este modo el usuario desplaza el móvil sobre la hoja que desea fotografiar y captura la imagen cuando considera que se encuentra en una posición adecuada. Para esta modalidad, las aplicaciones le indican al usuario si la imagen obtenida contiene la hoja completa o qué partes de esta no se han incluido para que este realice otra fotografía tomando en consideración estas correcciones.

A pesar de dicho sistema de realimentación al usuario, este modo aún depende de la habilidad y entrenamiento del usuario. Si este no ha recibido alguna capacitación para tomar fotografías a documentos, es muy probable que deba repetir la captura de imágenes en varias ocasiones. Por ejemplo, un método para facilitar la toma de fotografías en este modo consiste en colocar el móvil sobre el centro de la hoja; luego, se levanta el celular aproximadamente 20 cm y se procede a tomar la fotografía.

- **Modo de auto disparo**

En este modo, a diferencia del modo manual, la aplicación decide cuándo tomar la fotografía del documento basándose en algún patrón que haya detectado en las imágenes que capta la cámara de forma continua. Por ejemplo, algunas aplicaciones buscan y detectan los bordes de la hoja para decidir, mientras que otras capturan la imagen cuando detectan la presencia de texto en el área enfocada.

Si bien este modo permite reducir las probabilidades de error durante la captura, pueden causar problemas al tomar una imagen con baja resolución o texto incompleto.

- **Modo guiado**

Este modo incluye las funciones automáticas del modo de auto disparo y, en adición a esto, brinda direcciones al usuario mediante un ayudante de voz para guiarlo en tiempo real hasta que se haya detectado la hoja completa o un texto completo.

De acuerdo a los resultados de las pruebas realizadas por Cutter y Manduchi [10], este modo permite reducir el tiempo que requieren los usuarios para poder tomar una fotografía adecuada para OCR. Además, comprobaron que los usuarios luego de emplear esta modalidad cometen menos errores de captura en el modo manual.

1.3. Justificación de la tesis

La necesidad de educarse y progresar es propia del ser humano, esta se puede satisfacer en medida al empeño y esfuerzo que seamos capaces de entregar a dicha actividad; sin embargo, para una persona con discapacidad visual estas cualidades no son el único factor del que depende su desarrollo personal. Las carencias presentes en nuestras instituciones educativas especializadas en discapacidad representan un obstáculo considerable dado que no pueden proveer de la metodología y tecnología necesaria para brindarles una educación adecuada [22] [23].

En vista a esta situación, la tecnología existente en los smartphones actuales nos abre las puertas a soluciones que buscan suplir, en medida de lo posible, las carencias de nuestro contexto educativo. Las aplicaciones gratuitas orientadas a facilitarles el acceso a la información realizan un trabajo decente pero la gran mayoría presenta problemas con la etapa de adquisición de imágenes, esto se debe a que los métodos que emplean para realizar esta tarea están basados solo en el procesamiento de fotografías. Entonces, una solución necesaria para este problema surge del procesamiento de vídeos dado que resulta ser más flexible para los usuarios y permite corregir algunos problemas que son muy complejos de resolver mediante el procesamiento de fotografías [18].

1.4. Objetivos de la tesis

1.4.1. Objetivo general

El objetivo general de esta tesis es implementar un algoritmo que realice la formación de la imagen completa de una página con texto impreso a partir de fragmentos de esta extraídos de un vídeo grabado con un teléfono móvil.

1.4.2. Objetivos específicos

- Estudiar los métodos de adquisición de imágenes de las aplicaciones existentes.
- Crear una diversa base de datos con vídeos de muestra tomados grabados con un teléfono móvil.
- Implementar el algoritmo de formación de imagen completa.
- Probar el algoritmo sobre la muestra de vídeos registrados.
- Evaluar los resultados del algoritmo.

Capítulo 2
Fundamentos teóricos para el desarrollo de la tesis

2.1. Image stitching

La técnica de image stitching consiste en combinar dos o más imágenes que contienen un dominio de información superpuesta con la finalidad de obtener una sola imagen compuesta en alta resolución. Los algoritmos para alinear imágenes y unirlas para componer foto mosaicos se encuentran entre los más antiguos y más usados en el procesamiento de imágenes por computadora. El proceso de image stitching se emplea para producir los mapas y fotos satelitales de alta resolución que empleamos hoy en día. Estos algoritmos también se incluyen en la mayoría de dispositivos que cuentan con una cámara digital, ya que permiten crear fotos panorámicas de hasta 360° (véase Figura 2.1) [24].

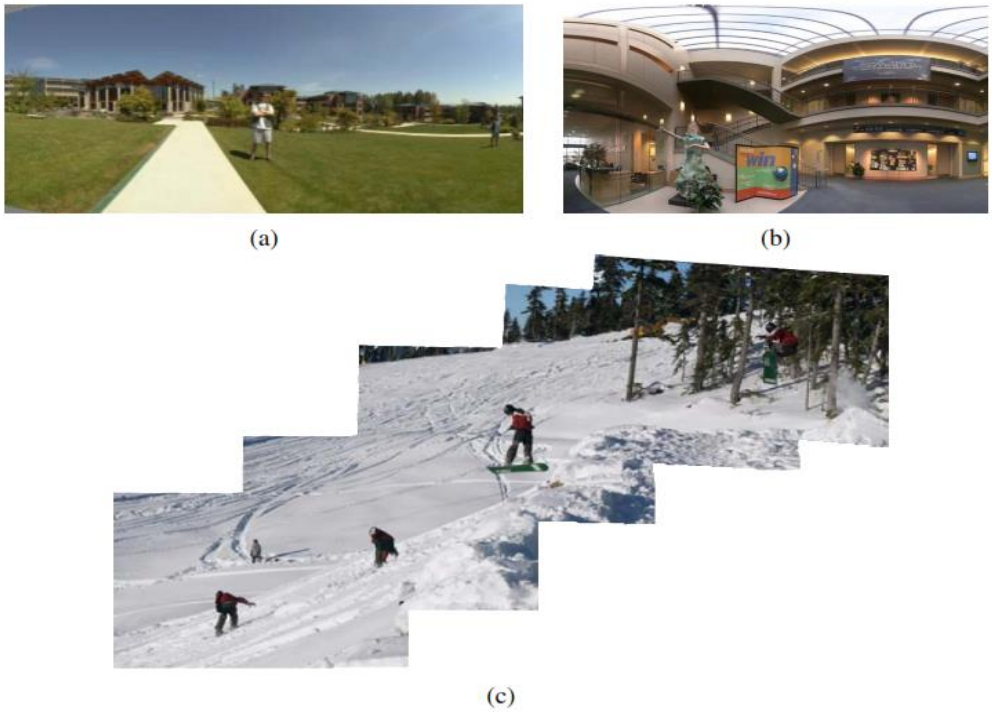


Figura 2.1: Imágenes compuestas mediante Image stitching: (a) Panorama cilíndrico, (b) panorama esférico y (c) imagen secuencial [24].

En la actualidad, el uso de estos algoritmos se ha extendido y desarrollado en el área de reconstrucción de documentos y composición de mosaicos de documentos. Por ejemplo, Wang y Pan emplean estos algoritmos en [25] para reconstruir de forma digital un documento que ha sido

triturado (véase Figura 2.2); mientras que, en aplicaciones como [8] o [19] se emplean para crear un mosaico de documento como se trató en el capítulo previo.

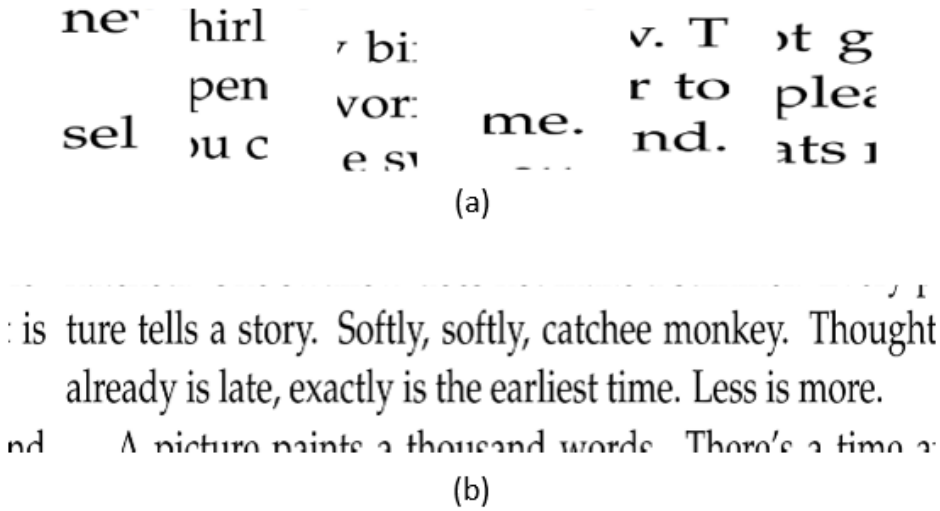


Figura 2.2: Aplicación de Image stitching en reconstrucción de documentos: (a) imágenes de los retazos de un documento y (b) texto reconstruido [25].

El primer paso para realizar esta composición es alinear las imágenes a combinar; es decir, se debe determinar el modelo matemático correcto que relaciona las coordenadas de los píxeles de una imagen con las coordenadas de otra. Esto se puede lograr empleando los modelos básicos de movimiento en dos dimensiones (2D) de las imágenes.

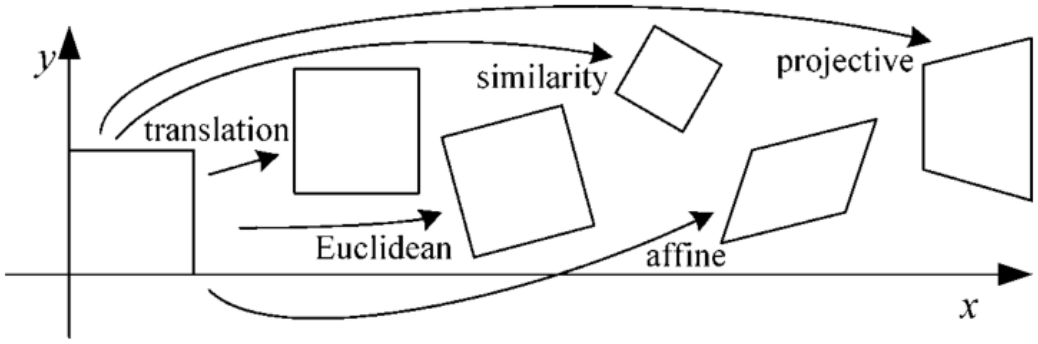


Figura 2.3: Transformaciones básicas en 2D [27].

Se puede realizar la transformación 2D de una imagen de acuerdo a la siguiente relación,

$$x' = M_T \cdot x \tag{2.1}$$

En donde *x'* representa la coordenada transformada de la imagen original, M_T es la matriz de transformación 2D y *x* es la coordenada original. En la tabla 2.1 se puede apreciar el orden de las matrices de transformación, de acuerdo al tipo, y las características que se conservan en una imagen luego de aplicar la transformación respectiva.

Tabla 2.1: Características de cada transformación 2D [27].

Name	Matrix	Preserves	Icon
Translation	$\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times 3}$	Orientation $+\cdots$	
Rigid (Euclidean)	$\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$	Lengths $+\cdots$	
Similarity	$\left[\, s\boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$	Angles $+\cdots$	
Affine	$\left[\, \boldsymbol{A} \,\right]_{2\times 3}$	Parallelism $+\cdots$	
Projective	$\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times 3}$	Straight lines	

El siguiente paso consiste en comparar las imágenes a unir y determinar las coincidencias que se superponen. Para esto existen técnicas en las que se comparan directamente cada uno de los píxeles de ambas imágenes; sin embargo, los resultados de esta técnica pueden tomar mayor tiempo de procesamiento y pueden presentar errores si las imágenes poseen mucha variación de contraste o contenido. Existen otras técnicas que superan estas dificultades, estas se basan en la detección de características (feature detection) y hoy en día han alcanzado un nivel de robustez y confiabilidad que permite su uso en una vasta cantidad de escenarios [24]. Las técnicas basadas en detección de características permiten acelerar la comparación de imágenes de manera significativa, en contraste con los métodos de comparación directa; por ello, representan la mejor opción para desarrollar el algoritmo requerido.

La detección de características se basa en el cálculo de dos elementos dentro de una imagen: sus puntos característicos y sus descriptores asociados. Los primeros elementos, los puntos característicos (feature points o keypoints), permiten detectar un conjunto reducido de las

locaciones que coinciden entre distintas imágenes; dichas locaciones poseen ciertas particularidades como pueden ser contornos, esquinas o algún otro punto distintivo. Una de las ventajas de trabajar con estos puntos es que permiten realizar el emparejamiento de imágenes incluso si existen diferencias de escala, orientación o contraste entre estas. El criterio más general y simple que se puede aplicar para determinar qué regiones de una imagen pueden contener potenciales puntos característicos, consiste en aplicar la siguiente suma de diferencias al cuadrado,

$$E_{AC}(\Delta u) = \sum_i w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2 \tag{2.2}$$

A esta función se le conoce como la autocorrelación de una imagen. Lo que realiza es una comparación entre una determinada región de la imagen (I_0), de coordenadas $x_i = (x, y)$, con una versión de sí misma ligeramente desplazada (Δu), si esta función tiene un valor mínimo lo suficientemente estable se puede decir que en dicha región comparada se puede establecer un punto característico. De esta ecuación se derivan la mayoría de técnicas de detección de puntos. Dichas técnicas parten de la matriz de correlación *A*,

$$E_{AC}(\Delta u) = \Delta u^T \cdot A \cdot \Delta u \tag{2.3}$$

la cual se puede calcular luego de expandir la función E_{AC} mediante la serie de Taylor para obtener un aproximado de la superficie de autocorrelación.

En la Figura 2.4 se puede apreciar la aplicación de este algoritmo: (a) imagen original con tres cruces rojas que representan las zonas de análisis; (b) autocorrelación en la cruz derecha inferior (conjunto de flores), aquí se puede apreciar la existencia de un punto mínimo único lo cual significa que puede existir un punto característico en dicha región; (c) correlación en la cruz derecha superior (borde del tejado), aquí se aprecia la existencia de varios puntos mínimos adyacentes por lo que no existe un punto característico en esta; y (d) autocorrelación en la cruz izquierda, en esta región no existe ningún pico mínimo estable y por ello no se puede identificar un punto característico en ella.

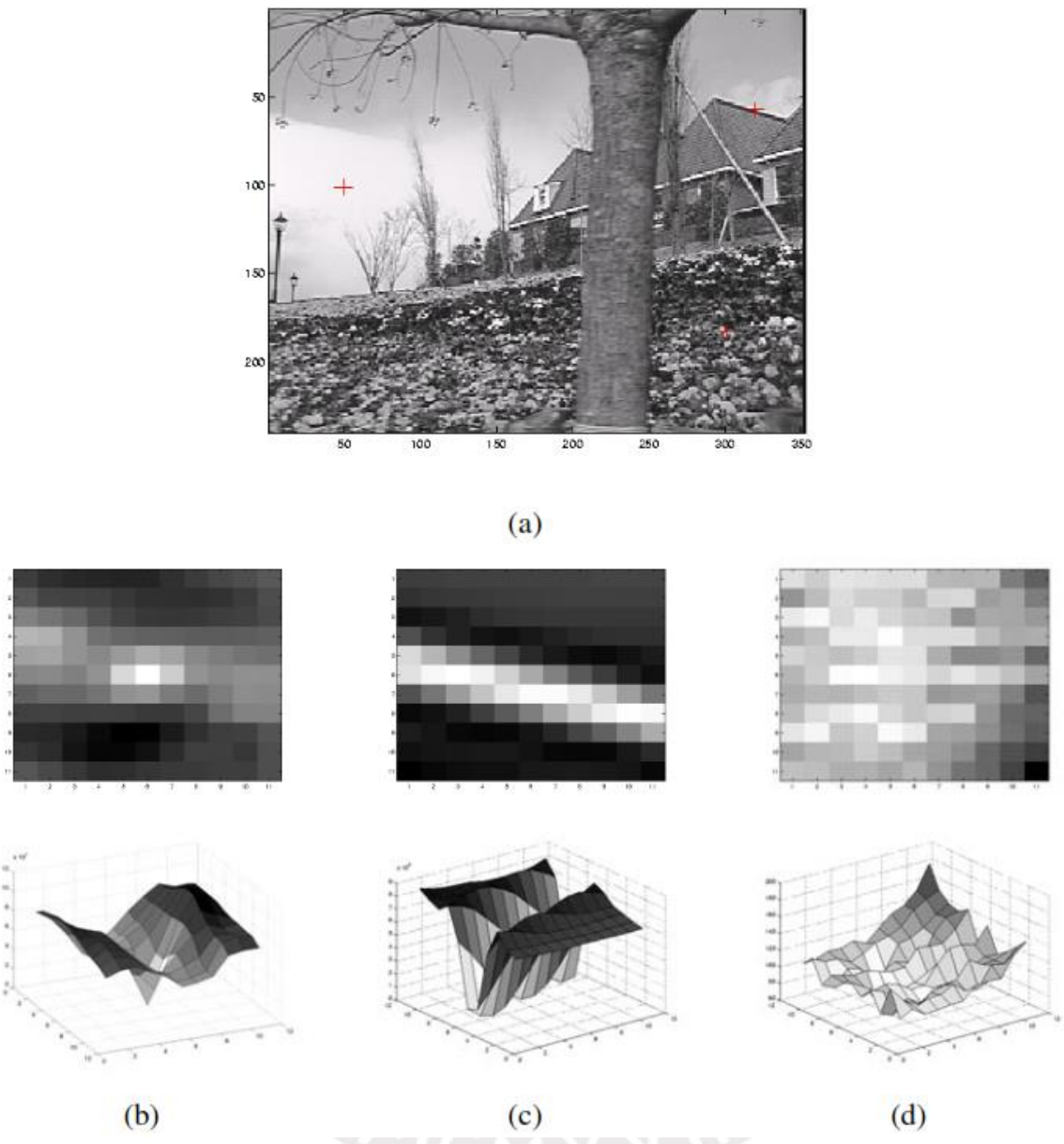


Figura 2.4: Análisis de puntos característicos y superficies de correlación [27].

El segundo elemento presente en la detección de características es el descriptor de un punto característico. Los descriptores permiten identificar la ubicación, escala y orientación de un punto característico dentro de una imagen para poder realizar el emparejamiento de estas. Algunas técnicas de creación de descriptores, como SIFT o GLOH, se basan en el análisis de gradientes de localidades próximas al punto característico (ventana de 16 x 16 píxeles) para determinar la orientación de este (véase Figura 2.5).

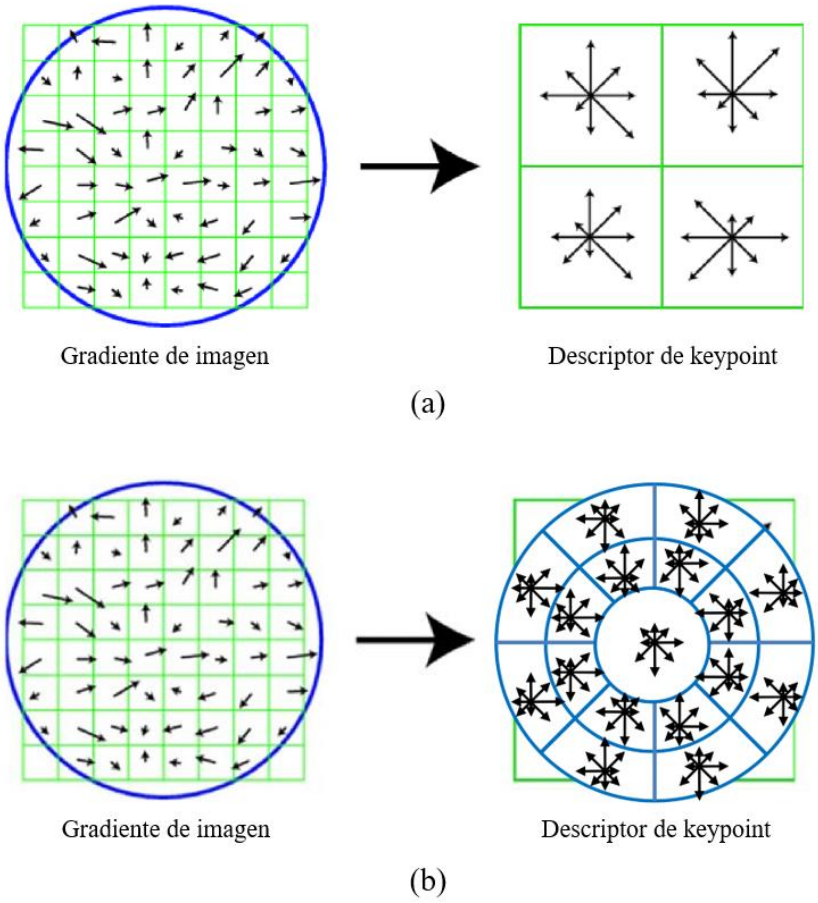


Figura 2.5: Descriptores de un punto característico: (a) usando técnica SIFT y (b) usando técnica GLOH [27].

2.2. Reducción de distorsiones por movimiento

Uno de los problemas más usuales al capturar imágenes ocurre debido al movimiento relativo entre el dispositivo de grabación y el documento a registrar, este efecto genera distorsiones en la imagen. Este tipo de distorsión puede tener formas muy generales dependiendo de la velocidad y dirección en la que se mueva el dispositivo. Para reducir dichos efectos negativos se puede emplear el control manual de exposición; es decir, se controla la cantidad de luz que ingresa al sensor de la cámara. Son tres las características que se pueden manipular en la cámara: apertura de la cámara, tiempo de exposición e ISO [26]. La apertura de la cámara es una medida que indica cuán abierto o cerrado se encuentra el iris de la cámara. Esta se mide en números f y un valor menor indica que el iris se encuentra totalmente abierto, véase Figura 2.6.

Figura 2.6: Apertura del iris de la cámara. Un valor más elevado de f indica menos cantidad de luz ingresando al sensor [26].

El tiempo de exposición hace referencia al tiempo que el obturador de la cámara permanece abierto y permite el paso de la luz hacia el sensor. Este tiempo se representa en forma fraccionaria y un valor elevado indica que más luz ingresará al sensor. Este parámetro es el más influyente en el control de la distorsión por movimiento, dado que con un valor adecuado se puede obtener un resultado más nítido. El parámetro ISO permite controlar la ganancia que se le aplica a las señales obtenidas por el sensor de la cámara, y por ello permite controlar los niveles de luz en esta. El efecto negativo de aplicar una ganancia elevada es que también se amplifica el ruido presente en

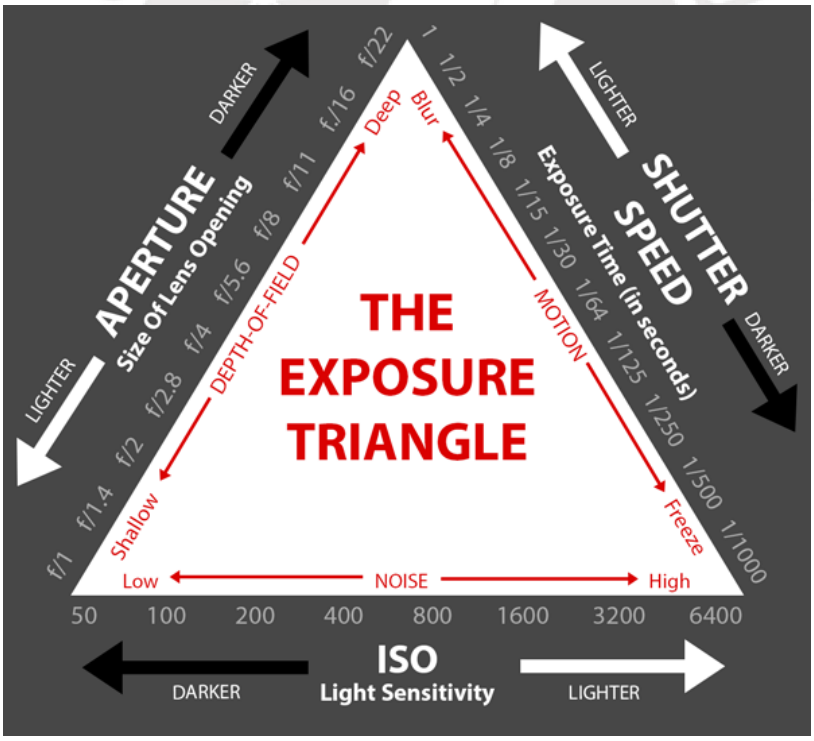
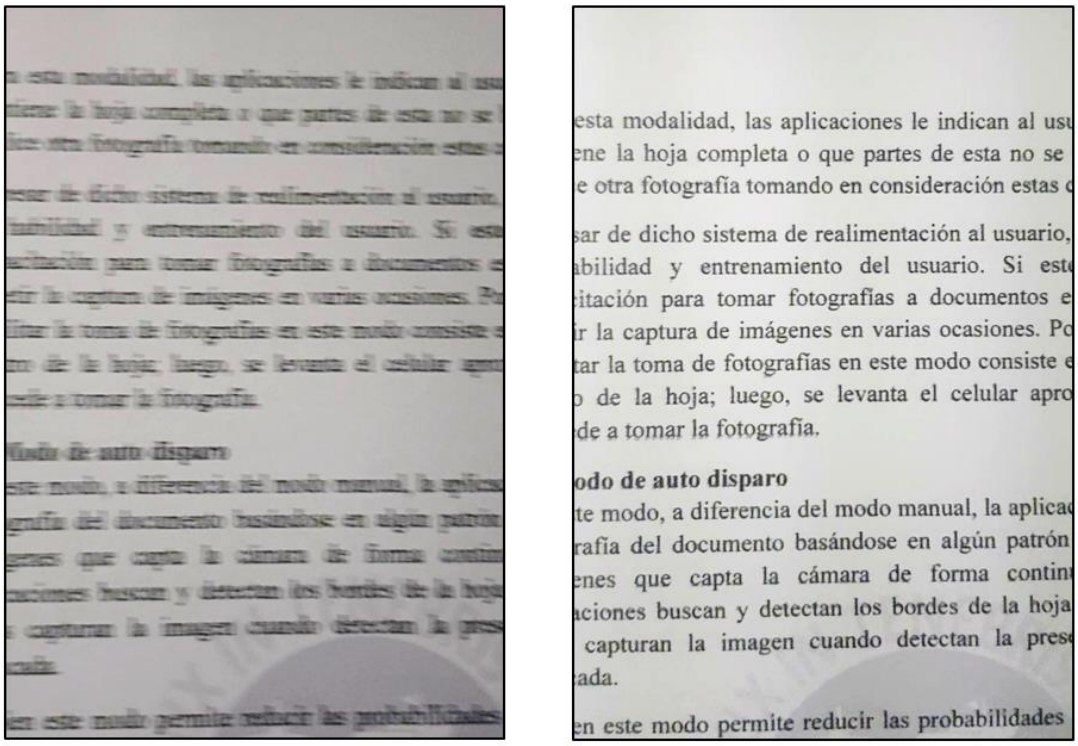


Figura 2.7: "Triángulo de la exposición". La combinación de estos parámetros afecta la luminosidad de la imagen y también la distorsión por movimiento [26].

la imagen. La relación existente entre estos parámetros se puede apreciar en la Figura 2.7. En el denominado “Triángulo de la exposición” se describe la relación complementaria de estas

características y como afectan la luminosidad, nitidez y el ruido presente en las imágenes capturadas. En la Figura 2.8 se muestra una comparación entre imágenes capturadas con diferente combinación en los parámetros de exposición.

Se puede apreciar la drástica diferencia en nitidez entre (a) y (b) al capturar una imagen con el móvil en movimiento. Dado que en (a) el tiempo de exposición es más elevado, no fue necesario incrementar el nivel de ganancia ISO. Por otro lado, en (b) el tiempo de exposición es bajo y se requiere elevar el nivel de ganancia para compensar la luminosidad reducida.



Apertura: f 1.7 , ISO: 64 , SS: 1/20 seg. **Apertura: f 1.7 , ISO: 800 , SS: 1/250 seg.**

(a) (b)

Figura 2.8: Comparación entre dos imágenes bajo las mismas condiciones de luminosidad, pero con distintos parámetros de exposición. Elaboración propia.

Para controlar estos parámetros en la cámara de un móvil con Sistema Operativo Android, este debe cumplir con dos condiciones obligatorias: la versión de Android instalada debe ser superior a la 5.0 y, además, debe contar con una interfaz de programación de aplicaciones (API por sus siglas en inglés) denominada Camera2. Dicha interfaz es implementada por el fabricante de cada teléfono con la finalidad de brindar a los desarrolladores de software la posibilidad de controlar la exposición de las cámaras y el tipo de enfoque. Camera2 puede ser implementada con 4 niveles de

acceso, cada nivel cuenta con su propio grupo de funcionalidades que se pueden controlar en la cámara. A continuación, se detalla las funciones que incluye cada nivel del API:

- Legacy: nivel de acceso mínimo, pensado para brindar retro-compatibilidad a equipos antiguos que poseen muy pocas funciones. Es equivalente al API Camera1 ya obsoleto.
- Limited: nivel de acceso básico para usar los beneficios principales del API Camera2. Puede incluir un subgrupo de funciones propias del nivel Full.
- Full: nivel de acceso que brinda el control manual de la exposición, flash y post-procesamiento de cada *frame*.
- Level_3: nivel de acceso total que incluye las funcionalidades del nivel Full y, además, permite capturar imágenes en formato RAW.

Para conocer el nivel de acceso en el API integrado a la cámara de un teléfono móvil se puede emplear la aplicación Camera 2 Probe. Dicha aplicación brinda información detallada de las funciones disponibles para los desarrolladores. En la Figura 2.9 se puede apreciar un ejemplo de la información que brinda Camera 2 Probe sobre la cámara trasera de un teléfono Huawei Y7 2018.

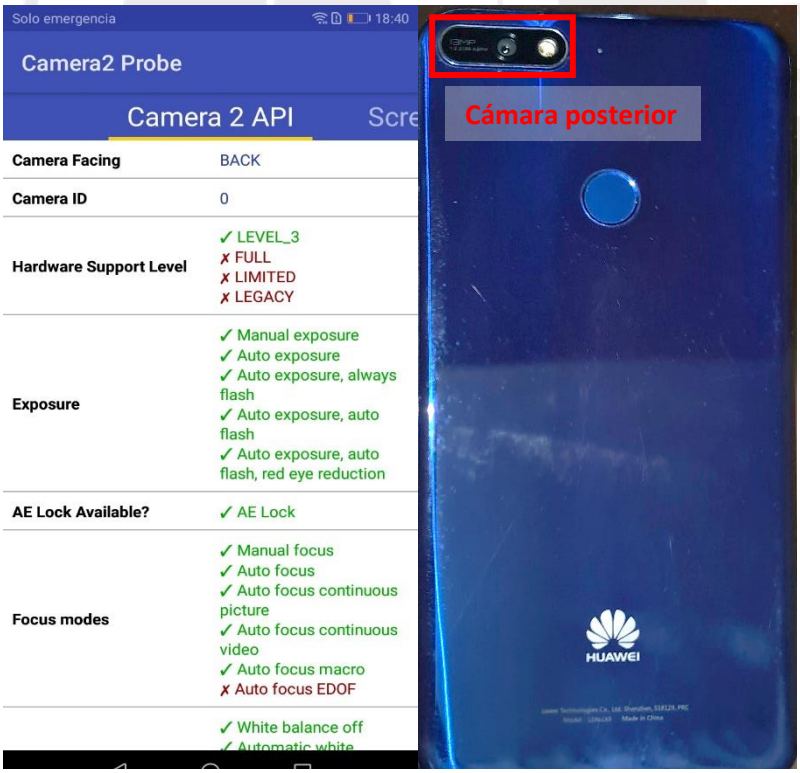


Figura 2.9: Información brindada por la aplicación Camera 2 Probe (Android OS) sobre la cámara posterior de un móvil Huawei Y7 2018. Nótese la capacidad de control manual de la exposición de la cámara.

2.3. Corrección de perspectiva

Como se mencionaba en líneas anteriores, los *frames* extraídos de un vídeo por lo general presentarán cierta diferencia de perspectiva entre sí, esto se debe a que en ciertas ocasiones el plano de grabación dejará de ser paralelo al plano del documento [31]. Este efecto genera diferencias de perspectiva entre los *frames* registrados por la cámara del dispositivo. En la Figura 2.10 se puede apreciar dicho efecto de distorsión que ocurre cuando el plano R (grabadora) y el plano D (documento) no son paralelos.

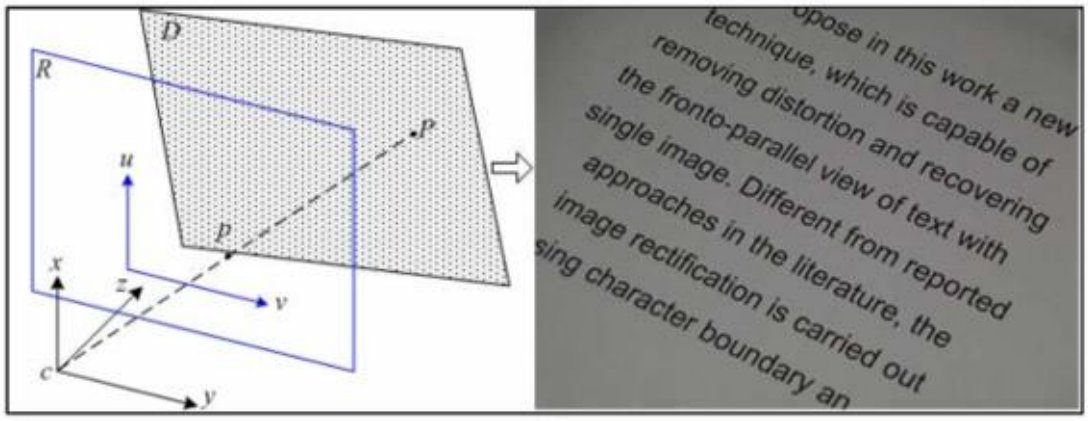


Figura 2.10: Causa de la distorsión por perspectiva, el plano de grabación no es paralelo al documento [31].

La corrección de perspectiva entre dos *frames* se puede realizar aplicando el método de correspondencia de los cuatro puntos tratado en [32]. Dicho método se ilustra en la Figura 2.11. En (a) se aprecia una imagen en la que se han seleccionado 4 puntos de coordenadas *(xi, yi)*, dichos puntos representan los vértices de un cuadrilátero que será proyectado a un nuevo plano con coordenadas *(xi', yi')* como en (b). Para realizar esta transformación es necesario recurrir a la ecuación 2.1 empleando una matriz homográfica de coeficientes *hij* como en la ecuación 2.4.

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{2.4}$$

En donde, las variables x_i' representan las coordenadas x_i proyectadas al plano requerido. Al aplicar la ecuación 2.4 a las coordenadas de los puntos seleccionados en la Figura 2.11 (a) se puede estimar el valor de las nuevas coordenadas de acuerdo a lo expuesto en la ecuación 2.5 y 2.6.

$$y'_i = \frac{h_{21}\,x_i + h_{22}\,y_i + h_{23}}{h_{31}\,x_i + h_{32}\,y_i + h_{33}} \tag{2.5}$$

$$x'_i = \frac{h_{11}\,x_i + h_{12}\,y_i + h_{13}}{h_{31}\,x_i + h_{32}\,y_i + h_{33}} \tag{2.6}$$

Dado que son 4 puntos, se generarán 8 ecuaciones en total que se deben resolver para esta transformación. La inversa de la matriz homográfica puede ser empleada para regresar la imagen a su sistema de coordenadas inicial.

Cabe señalar que una restricción para este método es que los puntos seleccionados en una imagen deben estar en posición general; es decir, no debe existir colinealidad entre más de 3 puntos seleccionados en la imagen original.

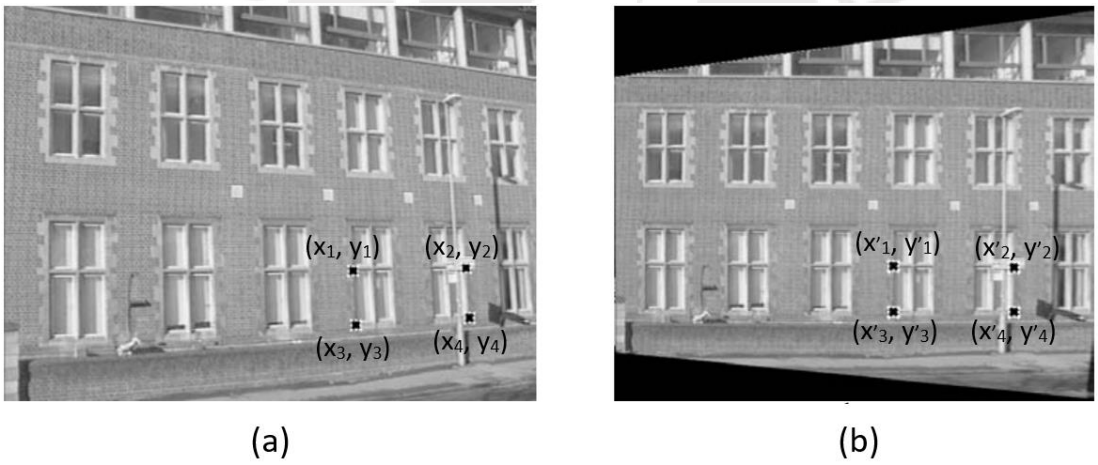


Figura 2.11: Corrección de perspectiva mediante la correspondencia de los 4 puntos. En (a) 4 puntos en el plano original y (b) 4 puntos en el plano proyectado [32].

2.4. Segmentación de imágenes

El análisis de imágenes comprende todos los métodos y técnicas que se utilizan para extraer información de una imagen. El primer paso para ello lo constituye la segmentación de imágenes que se ocupa de descomponer una imagen en sus partes constituyentes, es decir, los objetos de interés y el fondo, basándose en ciertas características locales que nos permiten distinguir un objeto del fondo y objetos entre sí, véase Figura 2.12.

La mayoría de las imágenes están constituidas por regiones o zonas que tienen características homogéneas (nivel de gris, textura, momentos, etc.). Generalmente estas regiones corresponden a objetos de la imagen. La segmentación de una imagen consiste en la división o partición de la imagen en varias zonas o regiones homogéneas y disjuntas a partir de su contorno, su conectividad, o en términos de un conjunto de características de los píxeles de la imagen que permitan discriminar unas regiones de otras. Los tonos de gris, la textura, los momentos, la magnitud del gradiente, la dirección de los bordes, etc., son características a utilizar para la segmentación [29].

Figura 2.12: Imagen original (izquierda) y su segmentación en regiones (derecha). Cada región es un conjunto de píxeles que son similares en color [30].

Cuando se aplica un algoritmo de segmentación de imagen lo que se busca es distinguir si un píxel pertenece, o no, a una región de interés, y por ello, produce una imagen binaria. Estos algoritmos se basan en alguna de las dos propiedades siguientes:

- Discontinuidad:

Mediante este tipo de procesos se busca detectar los cambios bruscos en la tonalidad de gris entre los píxeles de un determinado entorno de la imagen. De este modo se pueden detectar bordes, líneas y puntos aislados. Para realizar la detección de estas características se realiza la convolución de la imagen original con una matriz laplaciana que permite calcular la segunda derivada espacial de esta. La matriz de segundas derivadas posee mayor sensibilidad ante las variaciones de intensidad de tono, es por ello que se emplea para realizar la creación de segmentos (véase Figura 2.13) [34].

- Similaridad:

Este proceso se puede llevar a cabo de dos posibles formas. El primer método, denominado crecimiento de regiones, consiste en agrupar los píxeles o subregiones de la imagen en regiones mayores basándose en un criterio establecido. Normalmente se empieza con puntos “semilla” y se van añadiendo a este los píxeles que se encuentren en un determinado rango de tonalidad.

Por otro lado, tenemos los métodos basados en umbralización. Como lo señala su nombre, estas técnicas buscan modificar el histograma de una imagen creando valles en determinados tonos “umbral”. Los diversos métodos de esta clase poseen distintas maneras de determinar los valores umbrales y es por ello que se debe evaluar cual resulta más conveniente de acuerdo a la situación de aplicación (véase Figura 2.14) [34].

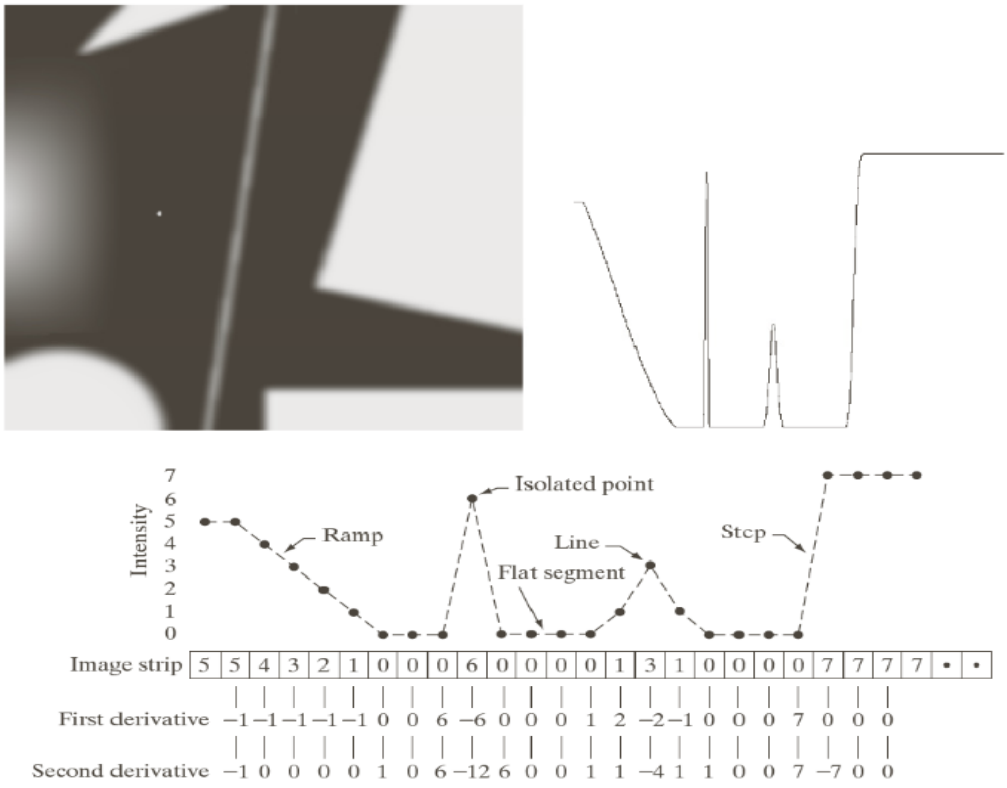


Figura 2.13: Detección de variaciones de la intensidad de tono en un vector horizontal de la imagen (incluyendo el punto de ruido aislado) [34].

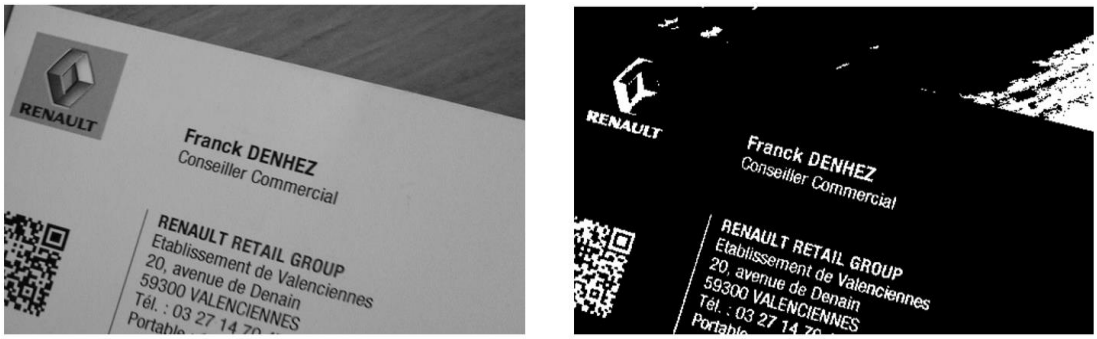


Figura 2.14: Binarización de un documento mediante umbralización adaptiva [35].

2.5. Implementación de algoritmos en OpenCV

OpenCV (Open Source Computer Vision Library) es una librería de código abierto enfocada en machine learning y procesamiento de imágenes por computadora. Esta librería cuenta con más de 2500 algoritmos optimizados, los cuales incluyen tanto funciones del estado del arte como funciones clásicas. Estos algoritmos pueden ser usados para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, seguimiento de movimiento, extracción de modelos 3D de objetos, producción de imágenes compuestas en alta resolución (image stitching), seguimiento de retinas, etc. Posee interfaces para C++, Python, Java, MATLAB y, además, soporta Windows, Linux, Android OS y Mac OS [33].

En primera instancia, con la finalidad de determinar los parámetros adecuados para los algoritmos empleados y realizar modificaciones de forma rápida, se desarrollará la propuesta en lenguaje de programación Python usando videos obtenidos a través de teléfonos móviles; luego, se ejecutará el flujo del algoritmo en una computadora de escritorio. En la siguiente etapa de desarrollo se empleará la interfaz de OpenCV en Android OS para implementar el algoritmo en lenguaje nativo C++ y el flujo será ejecutado directamente en el hardware del teléfono, incluyendo la captura de imágenes.

Capítulo 3
Diseño de la propuesta

3.1. Metodología de desarrollo

El primer paso en el diseño de la propuesta consiste en establecer la metodología que se aplicará para el desarrollo de la misma. Para ello se tomará como marco referencial el proceso general de composición de imágenes a partir de un vídeo. Dicho proceso se sintetiza en [39] y se puede apreciar el esquema en la Figura 3.1.

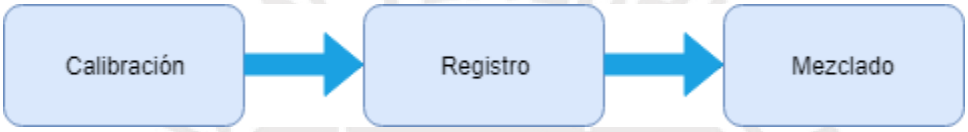


Figura 3.1: Proceso general de composición de imágenes a partir de un video [39].

De acuerdo a este proceso, luego de obtener los *frames* necesarios del vídeo, se debe aplicar algoritmos correctivos a estos para compensar distorsiones ocasionadas por factores intrínsecos y extrínsecos de la cámara (etapa de calibración). Acto seguido, se procede a detectar las características de cada imagen y emparejarlas con las coincidencias de las demás (etapa de registro). Finalmente, se aplican filtros sobre las imágenes alineadas y emparejadas para reducir las evidencias visuales de superposición (etapa de mezclado), en caso sea necesario. Usualmente, esta última etapa va de la mano con la etapa de calibración, ya que en esta también se reducen diferencias visuales entre imágenes.

Este proceso es adaptable a distintas aplicaciones específicas; un ejemplo de ello se puede apreciar en [11], [17], [19] o [28]. De acuerdo a los requerimientos se pueden aplicar distintas técnicas de calibración de imágenes y uno u otro método de detección de características. En el capítulo previo se ha descrito algunas de las técnicas de calibración aplicadas a la composición de imágenes de documentos que serán de utilidad para el desarrollo de este proyecto. Además, en este capítulo se tratará sobre la selección del método de detección de características más adecuado en base a la evaluación de sus respuestas ante distintos tipos de distorsión y el tiempo que toman en ser calculados.

En base a las técnicas tratadas en el capítulo previo y a los requerimientos de la situación en la que se usará el algoritmo propuesto, se puede apreciar en la Figura 3.2 el esquema metodológico que se seguirá para este proyecto.

Figura 3.2: Proceso del algoritmo propuesto. Elaboración propia.

Para seleccionar los *frames* que se han de extraer del vídeo se tomará en cuenta la calidad de la imagen mediante un análisis previo. De esta forma se puede mejorar la robustez de la imagen compuesta.

3.2. Definición de dataset

El algoritmo desarrollado en Python será aplicado sobre una base de vídeos de muestra capturados con un teléfono móvil, ello con la finalidad de comprobar su funcionamiento. Para la elaboración de cada video se tomarán en consideración distintos parámetros que permiten recrear diversas situaciones de grabación. La definición de los parámetros considerados para la grabación de cada video se anexará a este documento de tesis.

3.2.1. Consideraciones

Para el desarrollo de esta propuesta se emplearán muestras de hojas planas o con curvatura reducida como los propuestos en el dataset del concurso SmartDoc 2017 organizado en la Conferencia Internacional sobre Análisis y Reconocimiento de Documentos 2017 (ICDAR 2017, por sus siglas en inglés) [18]. Dos de los documentos propuestos por este comité se incluirán en la construcción de la base de muestras. Además, se escogieron otros documentos de muestra con contenido variado (solo texto, tablas, imágenes, etc.).

3.2.2. Parámetros a controlar

Con el fin de abarcar mayor diversidad de escenarios y condiciones de grabación, se deben tomar en consideración ciertos parámetros. La definición y control de estos permitirá brindar posteriores recomendaciones sobre el escenario óptimo para la aplicación del algoritmo.

3.2.2.1. Velocidad de grabación

Este parámetro hace referencia a cuán rápido se desplaza el teléfono móvil por sobre el documento. Afecta directamente a la duración del vídeo y a la nitidez de los *frames* capturados, por ello es recomendable que el movimiento de grabación se realice de forma suave (evitando movimientos rápidos o impulsivos). Para efectos de esta tesis se clasificará los videos en tres categorías: grabación lenta, media y rápida.

3.2.2.2. Dirección de grabación

La dirección en la que se desplaza el teléfono puede permitir mejorar la resolución del mosaico final a coste de incrementar la duración y tamaño del video registrado. Se tomará en consideración 4 formas de desplazamiento que permiten grabar la hoja completamente: vertical, horizontal, girando sobre la hoja y rotando sobre un eje (véase Figura 3.3).

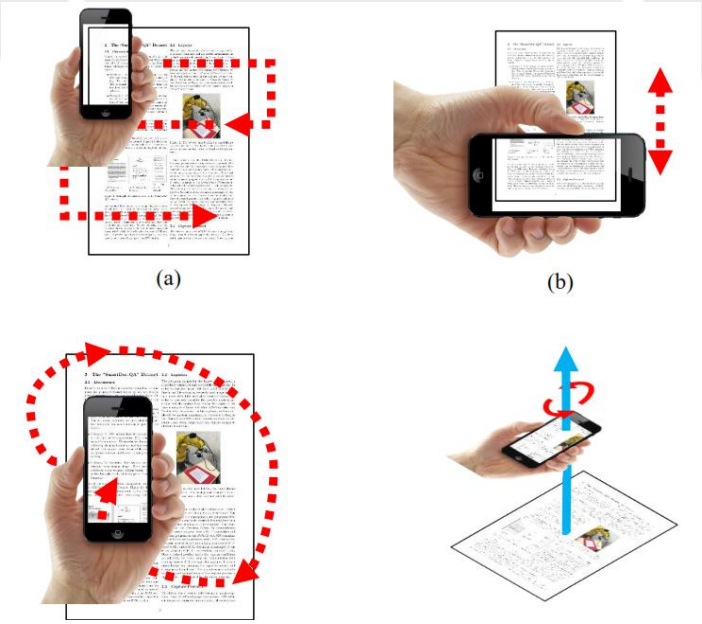


Figura 3.3: Modelos de desplazamiento sobre la hoja.
Elaboración propia.

Si el video se graba en dirección horizontal (a) se puede obtener un mosaico de alta resolución, pero se incrementa el tiempo de grabación y el tamaño del archivo. Aplicando el método de grabación en vertical (b) se puede obtener un mosaico con buena resolución y se reduce el tiempo de grabación. En (c) se puede apreciar la trayectoria helicoidal a seguir, con esta se puede brindar mayor flexibilidad al realizar la grabación y mantener una buena resolución en el mosaico resultante. Finalmente, de acuerdo a (d), se puede mantener el teléfono fijo sobre su eje y girarlo sobre la hoja mientras se inclina la cámara levemente con respecto al documento.

3.2.2.3. Resolución de video

Actualmente los teléfonos móviles de gama media permiten grabar videos en distinta calidad de acuerdo a la resolución del mismo. Al emplear una resolución mayor se puede alcanzar un mosaico final con mayor detalle a coste de mayor espacio para almacenar dicho video. Para la creación de este banco de muestras se tomarán en consideración las siguientes resoluciones:

- HD: 1280 x 720 píxeles.
- FHD: 1920 x 1080 píxeles.
- QHD: 2560 x 1440 píxeles.

3.2.2.4. Contenido del documento

Los documentos se clasificaron de acuerdo a su contenido, el cual puede limitarse a solo texto o texto con imágenes y tablas añadidas. Para *frames* que solo contienen texto pueden ocurrir problemas de detección de puntos característicos como se detalla en secciones siguientes de este capítulo.

3.2.2.5. Velocidad de obturación e ISO

Como se describió en el capítulo 2, el control de estos parámetros permite reducir en gran medida las distorsiones por movimiento en cada *frame*. Se tomaron en consideración las siguientes velocidades de obturación: 1/200, 1/500, 1/750 y 1/1000. Por otra parte, los niveles de ISO considerados son: 80, 125, 200, 320 y 350. Estos parámetros se calibraron en distintas combinaciones para cada grabación.

3.3. Creación de dataset

3.3.1. Grabación de videos

La grabación de los videos se realizó empleando un teléfono celular de la marca Samsung de la familia Galaxy S perteneciente a la gama media/alta de dicha compañía. La tasa de grabación es 30 fps y se realizaron los ajustes predefinidos sobre ISO y la velocidad de obturación. Cabe señalar que las condiciones de luminosidad son variadas de a acuerdo al entorno de grabación: aula de clase, biblioteca, sala de estar, etc.

3.3.2. Medición de la nitidez

Antes de proceder a extraer los *frames* de un video es necesario evaluar cuán nítida es la imagen, ello con la finalidad de obtener un resultado más robusto. Los *frames* se pueden ver afectados por la distorsión ocasionada por movimiento de la cámara o por efecto del desenfoque de la misma. Para este proyecto se empleará el método de análisis de la varianza del laplaciano de cada imagen como fue propuesto por Pacheco et al. en [36]. La ecuación 3.1 muestra la base de este método.

$$\Phi_{i,j} = \sum_{(i,j)\ \epsilon\ \Omega(x,y)} (\Delta I(i,j) - \overline{\Delta I})^2 \tag{3.1}$$

En donde Φ representa la varianza del laplaciano de una imagen. Esta se calcula de la sumatoria de cuadrados de la diferencia del laplaciano y el valor medio de este. Si el valor calculado de la varianza no supera un determinado valor umbral (Φ_{th}) significa que la imagen presenta distorsión; es decir, si Φ es menor que Φ_{th} la imagen se debe clasificar como borrosa. Para determinar el valor umbral necesario se realizó un análisis previo con *frames* de prueba. De acuerdo a los valores de varianza encontrados se pudo establecer un umbral apropiado. Cabe mencionar que los *frames* para este ensayo fueron previamente clasificados, de forma manual, como borrosos o nítidos para obtener referencias sobre los valores de sus varianzas (véase Figura 3.4).

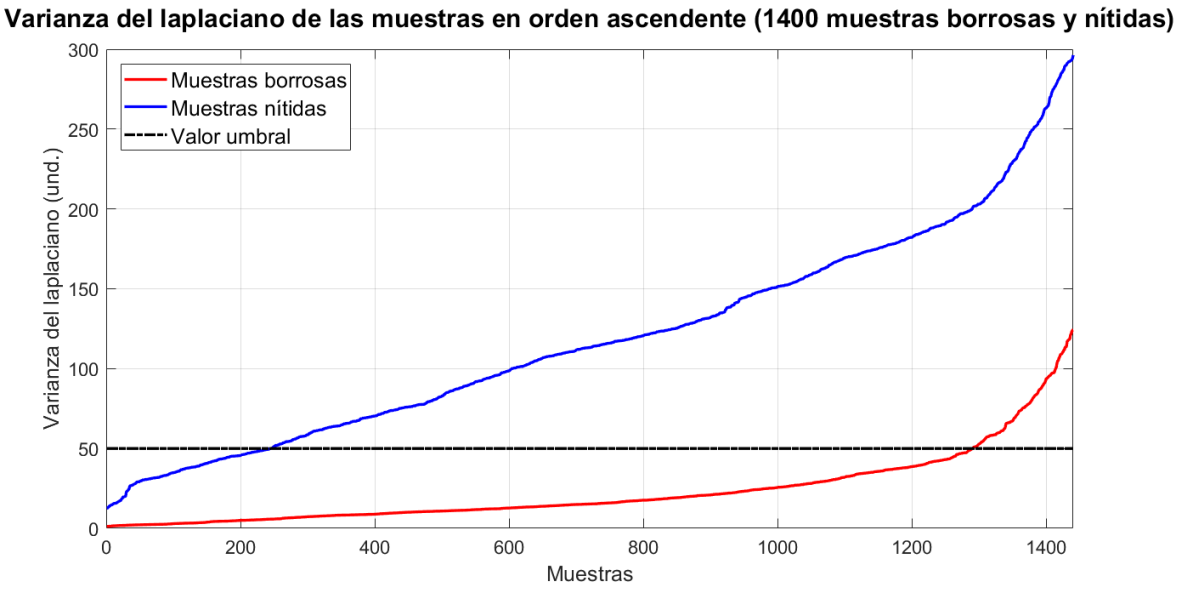


Figura 3.4: Varianza del laplaciano de muestras del dataset. El valor umbral se estima próximo a 50. Elaboración propia.

3.3.3. Extracción de *frames*

Se procede a extraer los *frames* de un video siempre que cumplan con la condición de nitidez impuesta mediante la evaluación de su laplaciano. El conjunto de *frames* extraídos se almacena como un conjunto de imágenes.

3.3.4. Tasa de selección

Como se mencionó previamente, la tasa de grabación de cada video es 30fps; es decir, se capturan 30 imágenes por segundo de video. Sin embargo, para realizar la composición del mosaico de documento no es necesario emplear las imágenes en su totalidad, por ello se necesita establecer una tasa de selección de *frames* para determinar cuántos serán empleados en la composición (ver sección 3.7). Los *frames* serán añadidos de forma secuencial al mosaico de acuerdo a la posición temporal que ocupaban en el video, es por ello que la tasa de selección se debe definir de tal forma que se evite escoger un *frame* que no tenga ninguna coincidencia con los elegidos anteriormente (ver sección 3.7.2) ya que, de darse esta situación, no se podría determinar la ubicación de dicho *frame* dentro del mosaico. En la Figura 3.5 se retrata este problema.

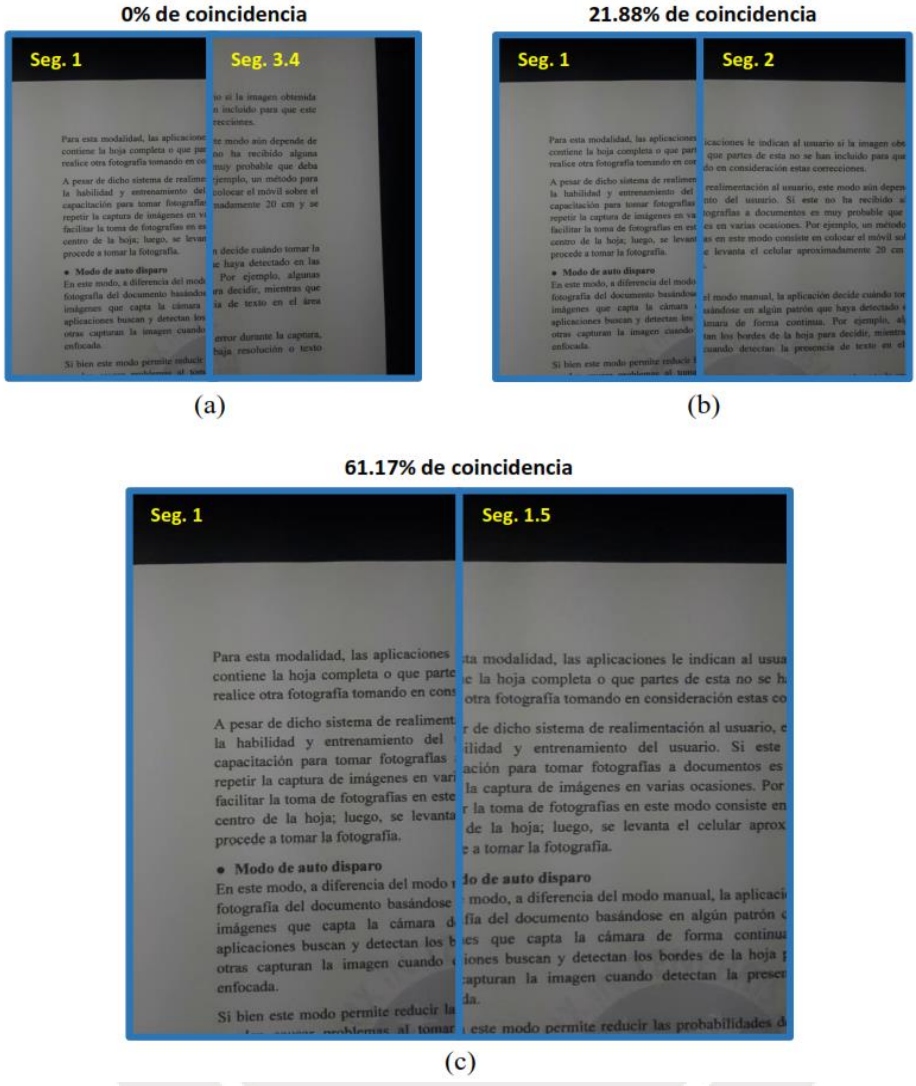


Figura 3.5: Problema de tasa de selección. (a) Para una tasa de 1 de cada 72 *frames* el proceso de emparejamiento no se puede realizar, (b) para una tasa de 1 de 30 *frames* la coincidencia aumenta, (c) la coincidencia óptima ocurre tomando 1 de cada 15 *frames*.

Las pruebas realizadas en la sección 4.2 permiten establecer la tasa de selección óptima como 1 de cada 15 *frames* disponibles en cada video para todas las direcciones de grabación planteadas; sin embargo, para el caso de grabación vertical es posible reducir el número de *frames* necesarios para componer un mosaico. Este tema se tratará con mayor profundidad en el siguiente capítulo.

3.4. Selección de puntos característicos y descriptores

Para detectar puntos característicos en cada *frame* se ha tomado en consideración el uso de tres métodos distintos que realizan esta tarea: SIFT, SURF y ORB. Cada una de estas técnicas destaca

en una característica: SIFT es el método más robusto ante variaciones de rotación y de escala en la imagen; SURF proporciona un nivel de robustez ligeramente por debajo de SIFT, pero es significativamente más rápido en los cálculos. Finalmente, ORB proporciona el cálculo de puntos más rápido, de las tres técnicas, a cuesta de reducida robustez.

En base al escenario sobre el cual se trabajará (imágenes con caracteres repetitivos) se debe determinar qué método resulta más fiable para realizar la detección de características. Además, en cada video se puede apreciar las variaciones de perspectiva, rotación y escala del documento debido al movimiento de la cámara. En [37] se realizó una comparativa de estas técnicas de detección frente a los tipos de distorsión previamente mencionados. De dicho trabajo se destaca SURF por sobre las demás técnicas gracias a que tiene la mejor relación entre robustez y tiempo de cálculo, y por ello es el método que se usará en primera instancia para el desarrollo del algoritmo.

Dado que en un documento los caracteres se repiten con frecuencia entre párrafos, es necesario que cada punto característico permita identificar un área coincidente entre dos imágenes sin arrojar falsas coincidencias positivas. En la Figura 3.6 se puede apreciar la detección errónea de coincidencias debido a la repetición de caracteres.

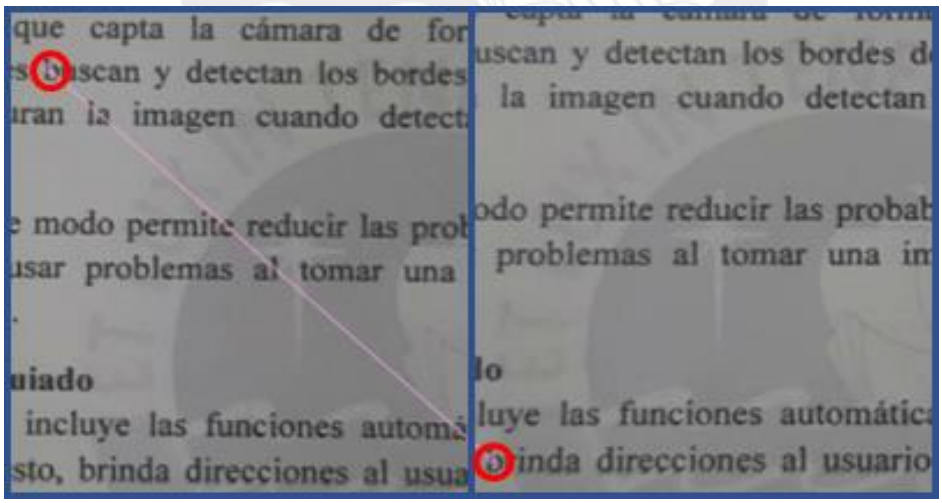


Figura 3.6: Emparejamiento erróneo. La letra "b" aparece en dos párrafos distintos. Elaboración propia.

Este problema ocurre cuando un punto característico es muy pequeño y solo abarca una letra. Por ello, el objetivo es obtener un mayor número de puntos característicos que abarquen sílabas o palabras enteras para reducir las coincidencias erróneas.

3.4.1. Análisis de puntos característicos para SIFT

Para aplicar este método usando OpenCV se debe crear un objeto SIFT usando la función cv.SIFT_create(). Dicha función puede recibir hasta 5 parámetros de control. Estos parámetros son los siguientes:

- **Int** nfeatures: El número de mejores puntos característicos a mantener. SIFT clasifica cada punto de acuerdo a su calificación de contraste local. Por defecto toma el valor de 0 (se mantienen todos los puntos característicos).
- **Int** nOctaveLayers: Indica el número de capas por cada octava. Este valor se calcula automáticamente de acuerdo a la resolución de la imagen. El valor por defecto es 3, de acuerdo con el autor del algoritmo.
- **Double** contrastThreshold: Valor umbral que filtra los puntos característicos generados en zonas de bajo contraste. Un valor mayor reduce el número de puntos detectados. Su valor por defecto es 0.04.
- **Double** edgeThreshold: Valor umbral que permite filtrar los puntos característicos detectados en zonas como bordes o contornos. Un valor mayor aumenta el número de puntos detectados. Su valor por defecto es 10.
- **Double** sigma: Valor de la constante sigma del filtro gaussiano aplicado a la imagen en la octava número 0. Su valor por defecto es 1.6, de acuerdo con lo propuesto por el autor del algoritmo.

Empleando el detector SIFT con sus parámetros por defecto en un *frame* de muestra se obtiene el siguiente resultado (véase Figura 3.7).

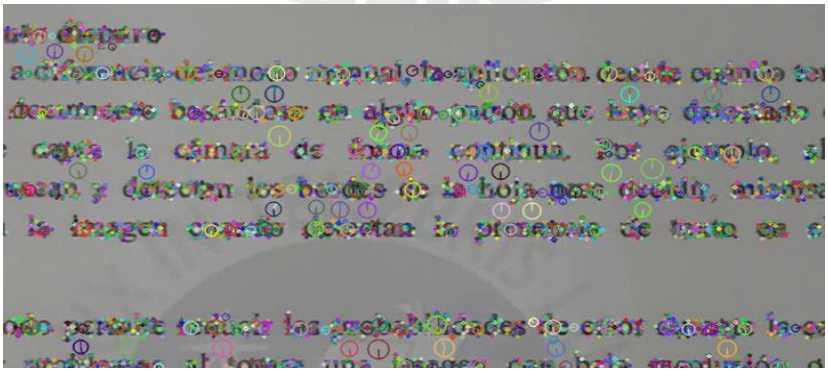


Figura 3.7: Puntos característicos detectados con SIFT y sus parámetros por defecto. Elaboración propia.

De la imagen previa se puede destacar el elevado número de puntos detectados en cada carácter. Estos son de tamaño reducido y generan el problema de emparejamiento erróneo como lo visto en la Figura 3.5. Cabe señalar que, si se tiene un número elevado de puntos, la velocidad del algoritmo decrece dado que la etapa de cálculo de homografías debe hacer más iteraciones para determinar los emparejamientos correctos (ver sección 3.6). Por ello, es necesario estimar los parámetros adecuados para el detector SIFT.

3.4.2. Selección de parámetros óptimos para SIFT

En adición a los 5 parámetros previamente mencionados se añadirá, mediante la programación necesaria, un parámetro denominado SizeFilter. Este tiene por finalidad filtrar los puntos característicos que no tengan un radio mayor al mínimo requerido. Esta etapa de filtrado adicional se explica posteriormente en la sección 3.5.1.

De forma experimental se ha evaluado los valores óptimos para cada parámetro del detector y se ha obtenido un rango aproximado para cada uno (ver tabla 3.1); luego, se procede a ejecutar una serie de iteraciones variando los valores de cada parámetro dentro del rango establecido para determinar un valor óptimo fijo para cada uno (ver Figura 3.7). Este valor óptimo se escogerá de acuerdo al número de puntos emparejados correctamente en comparación con el total de puntos detectados. En la Figura 3.8 se puede apreciar la mejora en robustez de los puntos detectados al emplear valores dentro de los rangos establecidos.

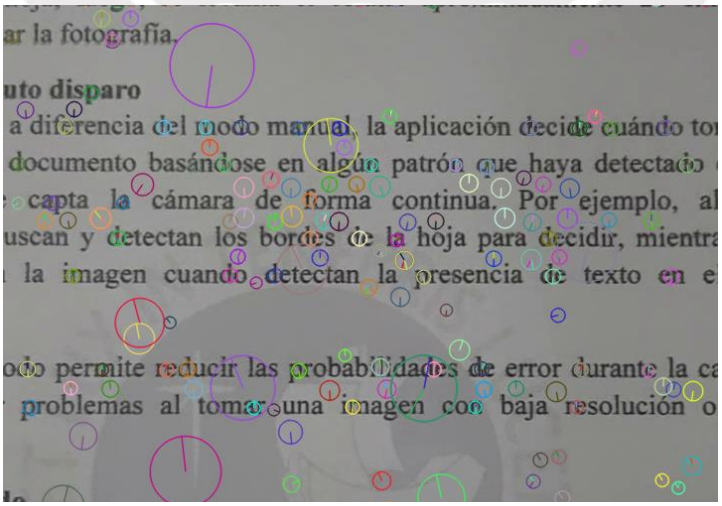


Figura 3.8: Puntos característicos mejorados. Parámetros: SizeFilter = 5, contrastThreshold = 0.02 y los demás en su valor por defecto. Elaboración propia.

Tabla 3.1: Rango de variación de los parámetros y pasos en cada iteración. Elaboración propia.

Parámetro	Rango	Pasos
SizeFilter	6 - 8	0,5
nFeatures	0	-
nOctaveLayers	6 - 8	1
ContrastThreshold	0,02 - 0,002	-0,006
EdgeThreshold	6 - 36	10
Sigma	1,6 - 1,2	-0,2

Para cada iteración se almacenó en un archivo los valores de cada parámetro y el porcentaje de puntos emparejados de forma correcta. De los datos almacenados se extrajo que la mayor tasa de aciertos se logró con los siguientes valores: SizeFilter con 7; nFeatures con 0; nOctaveLayers con 6; ContrastThreshold con 0.02; EdgeThreshold con 26 y Sigma con 1.4.

```
for SizeFilter in np.arange(5,8,0.5):
    for nOctaveLayers in np.arange(6,8,1):
        for ContrastThreshold in np.arange(0.02,0.002,-0.006):
            for EdgeThreshold in np.arange(6,36,10):
                for Sigma in np.arange(1.6,1.2,-0.2):
                    main(SizeFilter,0,nOctaveLayers,ContrastThreshold
                        ,EdgeThreshold,Sigma)
```

Figura 3.9: Iteraciones anidadas para prueba de parámetros. Elaboración propia.

3.4.3. Aplicación de SIFT

Luego de definir los parámetros a usar con SIFT se procede a evaluar visualmente el grado de conformidad en el emparejamiento de los puntos entre dos *frames*. Este procedimiento se realiza mediante el dibujo de las líneas de correspondencia; para este caso se puede comprobar que los emparejamientos correctos generan líneas de correspondencia paralelas (ver Figura 3.10).

3.4.4. Aplicación de SURF

Para emplear el detector de tipo SURF se debe crear un objeto de dicha clase mediante la función cv.SURF_create(). Este descriptor cuenta solo con 3 parámetros: hessianThreshold, nOctaves y

nOctaveLayers. Realizando el análisis de estos puntos de forma análoga a SIFT, se determinó que los valores adecuados para este detector son: hessianThreshold con 100; nOctaves con 4 y nOctaveLayers con 2. El resultado se representa en la Figura 3.11.

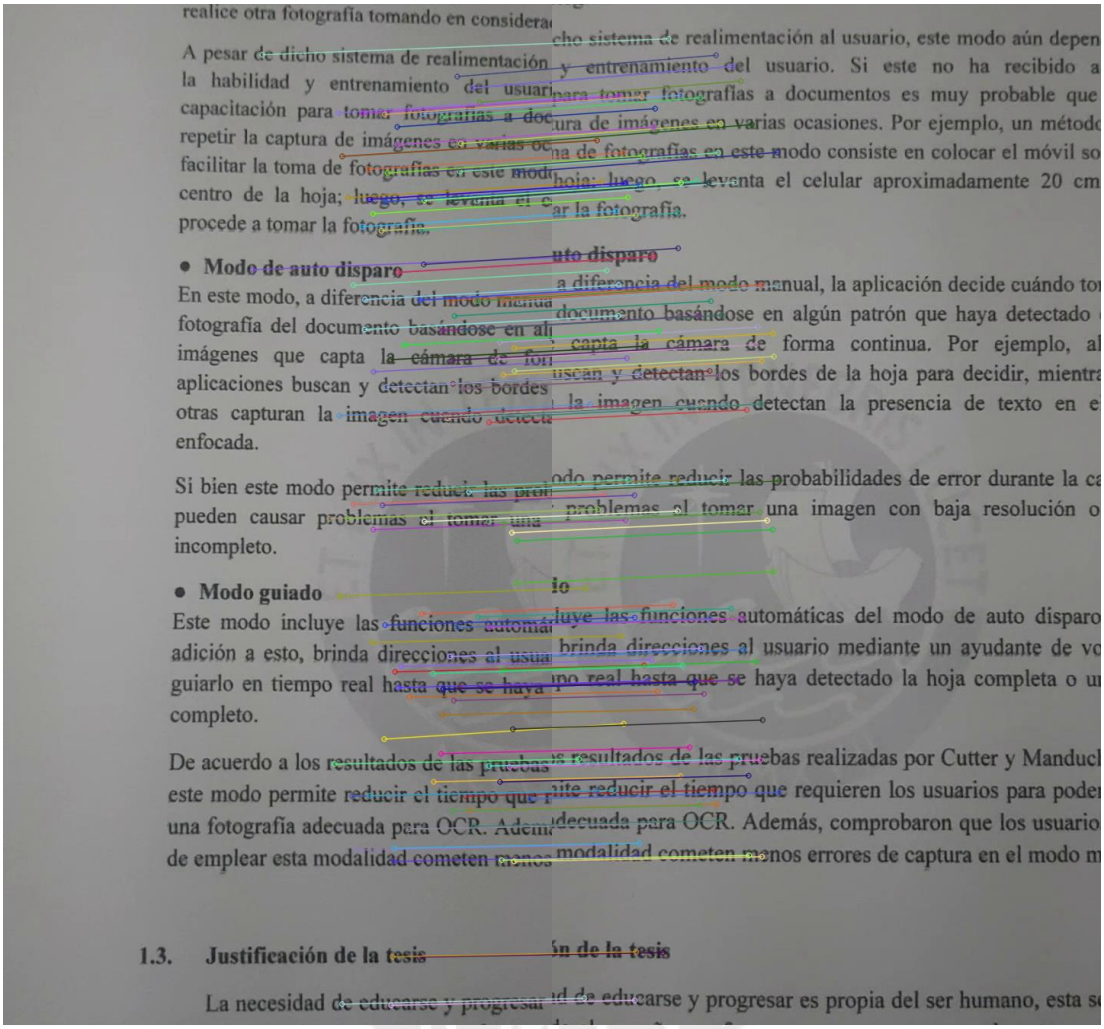


Figura 3.10: Líneas de correspondencia (SIFT) de puntos entre dos *frames*. La mayor parte de las líneas son paralelas indicando un emparejamiento correcto. Elaboración propia.

3.4.5. Aplicación de ORB

Al emplear el método ORB en *frames* que no contienen nada más que texto (sin imágenes ni tablas), los puntos detectados se ubican en las zonas centrales de cada imagen [37] y debido a ello no se puede realizar un correcto emparejamiento. Puesto que la mayoría de coincidencias se ubican

en los bordes de un *frame*, este método no permite identificarlas y debió ser descartado como posible técnica de detección para el desarrollo de esta propuesta (ver Figura 3.12).

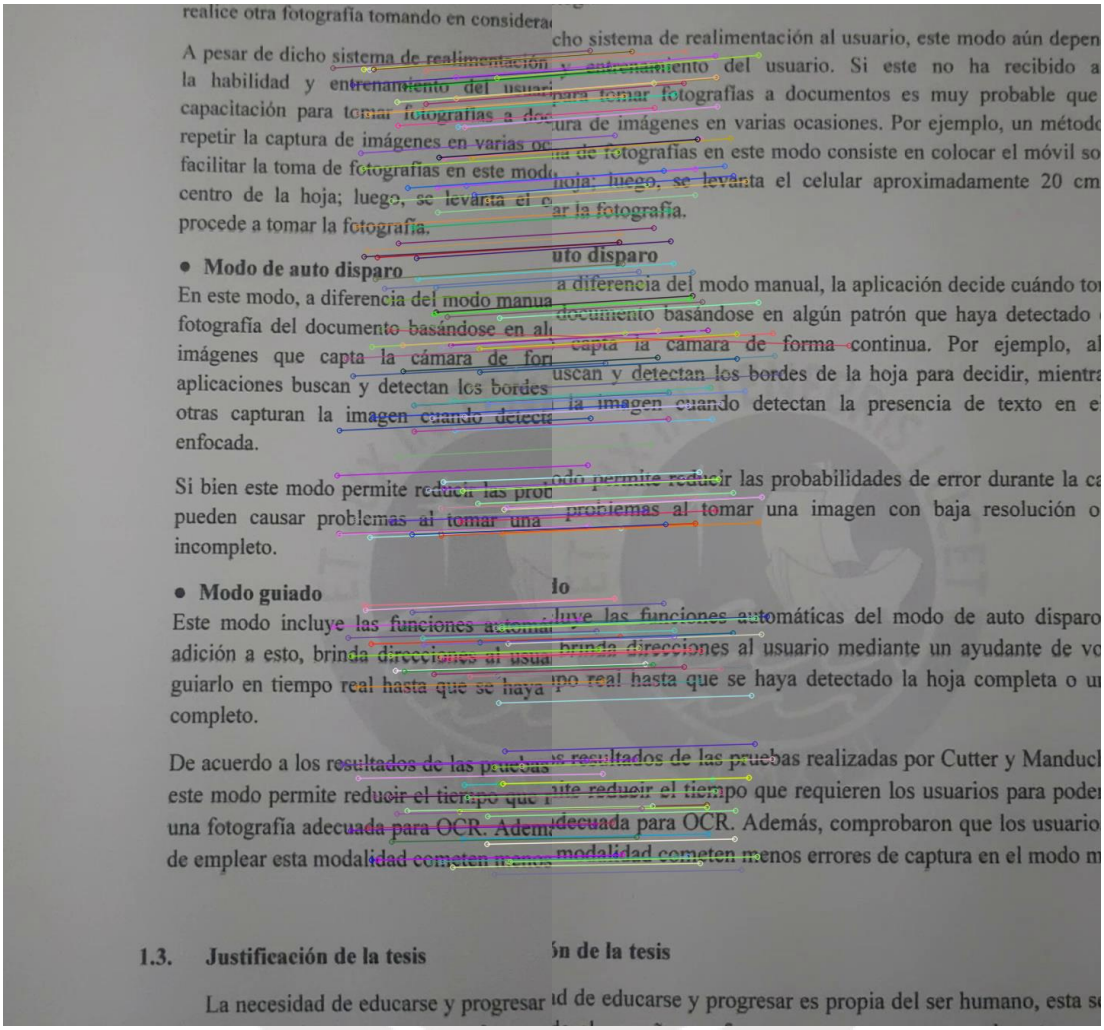


Figura 3.11: Líneas de correspondencia usando SURF. La mayoría de líneas son paralelas. Elaboración propia.

3.5. Emparejamiento de puntos característicos

La librería OpenCV posee herramientas para realizar el emparejamiento de descriptores de puntos característicos. El objeto de clase BFmatcher (Brute Force Matcher) permite comparar los conjuntos de descriptores de dos imágenes y retorna un vector indicando que puntos se relacionan entre sí. Para determinar dicha similitud calcula la distancia euclidiana (para SIFT y SURF) entre dos descriptores y se definen como correspondencias si tienen la mínima distancia calculada.

Si bien esta herramienta devuelve gran parte de correspondencias verdaderas, aún es necesario implementar filtros adicionales (tamaño de puntos característicos y distancia de descriptores) para optimizar los emparejamientos y reducir los errores.

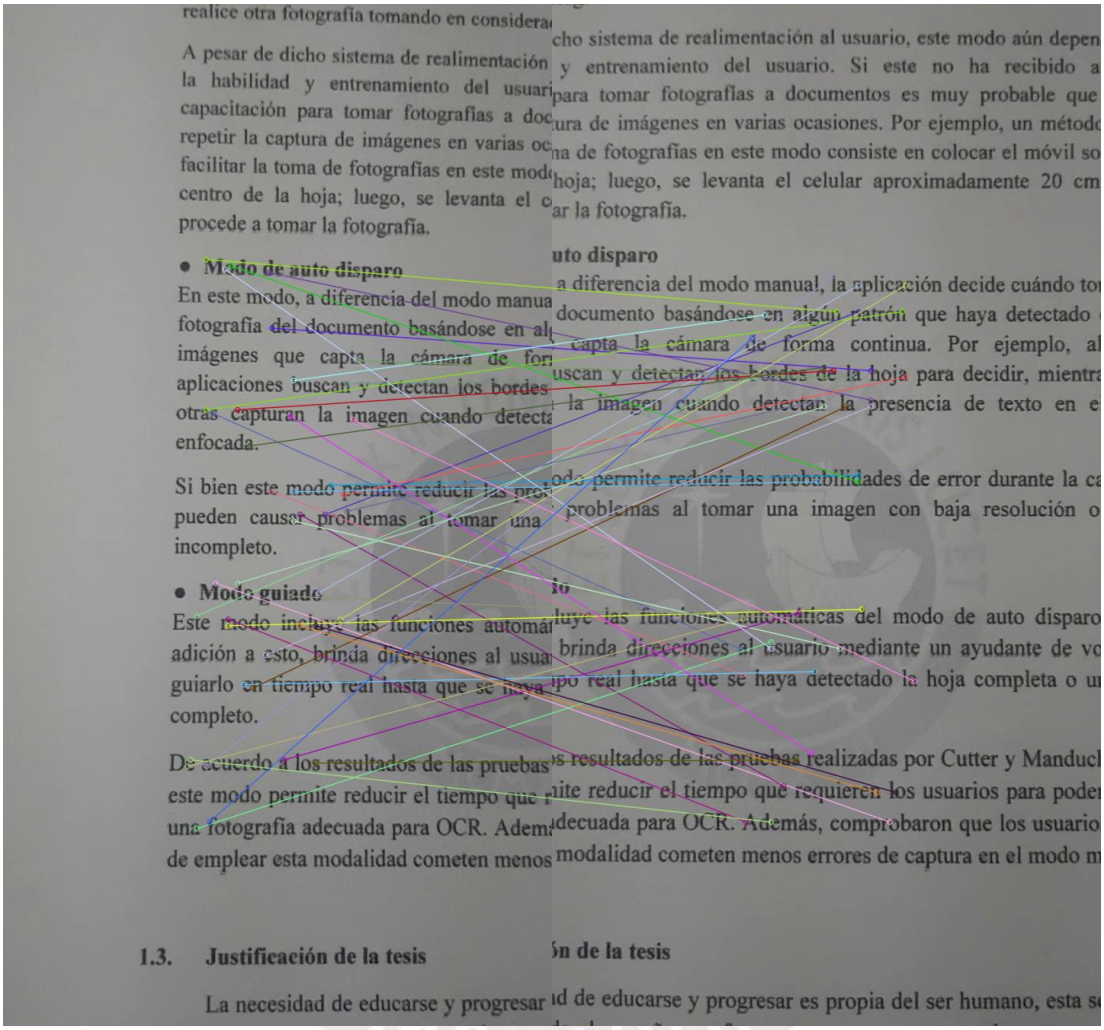


Figura 3.12: Los puntos detectados por ORB se focalizan en la zona central de cada frame y generan errores de emparejamiento. Elaboración propia.

3.5.1. Puntos más importantes

De acuerdo a lo visto en la Figura 3.6, los puntos característicos de un tamaño menor al de un carácter ocasionan errores de emparejamiento debido a que el objeto BFmatcher asocia dos caracteres que pueden presentarse en distintos párrafos. En vista a esta situación, es necesario implementar un filtro de puntos de acuerdo a su tamaño para evitar errores durante el cálculo de homografías.

Cada punto característico pertenece a la clase **KeyPoint** y cuentan con un atributo denominado **Size** (o tamaño), el objetivo es establecer un valor umbral para este atributo a fin de eliminar los puntos innecesarios. El efecto de filtrado se aprecia en la Figura 3.13. En (a) el valor de SizeFilter es 1, ningún punto es filtrado. En (b) se aplica SizeFilter igual a 6.

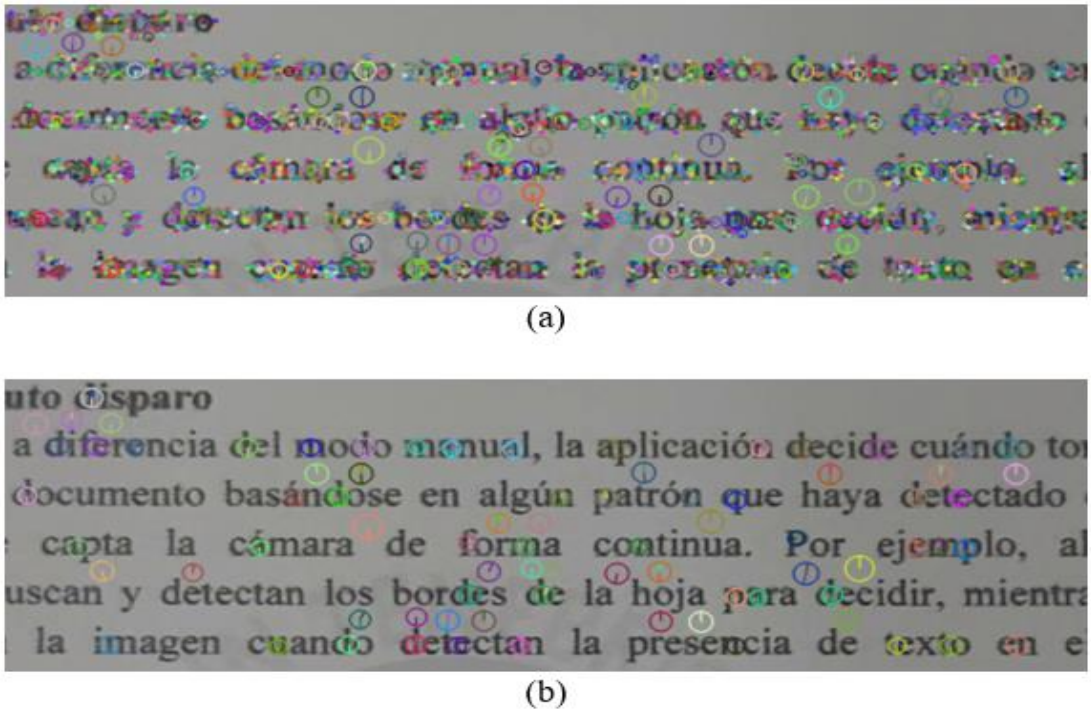


Figura 3.13: Filtrado de KeyPoints en un frame. (a) SizeFilter = 1, (b) SizeFilter = 6. Elaboración propia.

3.5.2. Técnicas y parámetros

También es necesario implementar un filtrado de descriptores emparejados para eliminar ambigüedades como las vistas previamente (Figura 3.6). Para ello se emplea la prueba de la relación de distancias, descrito por D. Lowe en [38].

El criterio consiste en lo siguiente, para los descriptores calculados en una imagen se buscan las dos correspondencias más cercanas (mediante distancias euclidianas) en el conjunto de descriptores de la otra imagen. Luego, se procede a comparar las distancias de estas dos correspondencias mediante la siguiente ecuación:

$$\frac{D1.distance()}{D2.distance()} < Ratio \tag{3.2}$$

En donde, D1.distance() representa la distancia asociada al descriptor más cercano que fue emparejado al descriptor de la imagen original; D2.distance() representa a la distancia del segundo descriptor más cercano y Ratio es el valor umbral que se debe definir para este criterio (0.8 de acuerdo al trabajo de Lowe). Si se cumple esta relación, entonces se puede admitir al descriptor D1 como una buena correspondencia. Por otro lado, si no se cumple la relación ambos descriptores deben ser descartados debido a una posible ambigüedad en los emparejamientos. Si bien Lowe recomienda el valor de 0.8 para Ratio, experimentalmente se ha corroborado que para este escenario (imágenes con alto contenido repetitivo) un valor menor de Ratio brinda mejores resultados y elimina la mayoría de ambigüedades. Por ello, se ha establecido el valor de Ratio en 0.68. En la Figura 3.13 se aprecia la codificación criterio previamente descrito.

```
#Eliminando ambigüedades con "Lowe's ratio test"
verify_ratio = 0.68 #Ratio definido
verified_matches = []
for D1,D2 in Matches:
  #Se admite el discriptor más cercano solo si
  #cumple el ratio definido
  if D1.distance < verify_ratio * D2.distance:
    verified_matches.append(m1)
```

Figura 3.14: Criterio de la relación de distancias de Lowe para eliminar emparejamientos ambiguos. Elaboración propia.

3.6. Alineación de *frames*

Luego de detectar y emparejar los puntos característicos más robustos de ambas imágenes se procede a calcular la matriz de transformación homográfica que permitirá alinear la perspectiva de ambas imágenes (ver puntos 2.1 y 2.4). Para calcular esta matriz se puede emplear la función de OpenCV cv.findHomography(); esta función recibe los siguientes 4 parámetros:

- **Array** srcPoints: Coordinadas de los puntos característicos del plano original.
- **Array** dstPoints: Coordinadas de los puntos característicos del plano de proyección objetivo.
- **Int** method: El método empleado para clasificar emparejamientos adecuados y calcular la matriz homográfica.
- **Double** ransacReprojThreshold: Valor umbral que define el error máximo en la posición de un punto original proyectado al nuevo plano.

El método (method) seleccionado para esta propuesta es RANSAC dado que es el más robusto de los disponibles y permite obviar los emparejamientos erróneos que aún puedan quedar luego de las etapas filtrado tratadas previamente. El valor umbral requerido para este método se estableció en 0 dado que no se debe tolerar error en la proyección de un *frame*. En la Figura 3.15 se ilustra el proceso de alineación de acuerdo a lo tratado en esta sección.

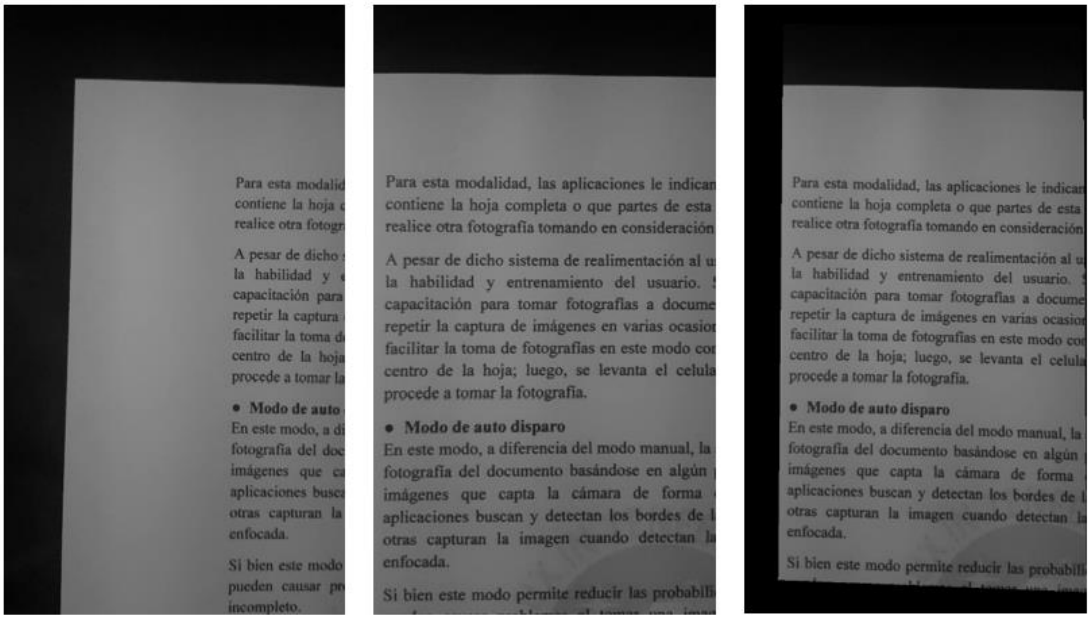


Figura 3.15: Alineación de *frames*. En (a) el *frame* con el plano de proyección objetivo, (b) *frame* a proyectar y (c) *frame* en (b) proyectado al plano de (a). Elaboración propia.

3.7. Creación de mosaico

3.7.1. Técnicas y parámetros

El mosaico final se construye con una secuencia de frames extraídos y alineados de forma consecutiva conforme a su posición temporal en el video. Los frames se seleccionan en intervalos definidos por la tasa de selección determinada en el punto 3.3.3. El proceso de unión y alineación de frames se puede realizar haciendo uso de la función cv.warpPerspective(). Dicha función recibe como parámetros a la imagen del mosaico parcial, el frame que debe ser alineado y añadido al mosaico, y la matriz de transformación homográfica. En la Figura 3.16 se puede apreciar el resultado preliminar de un mosaico de documento compuesto de forma secuencial.

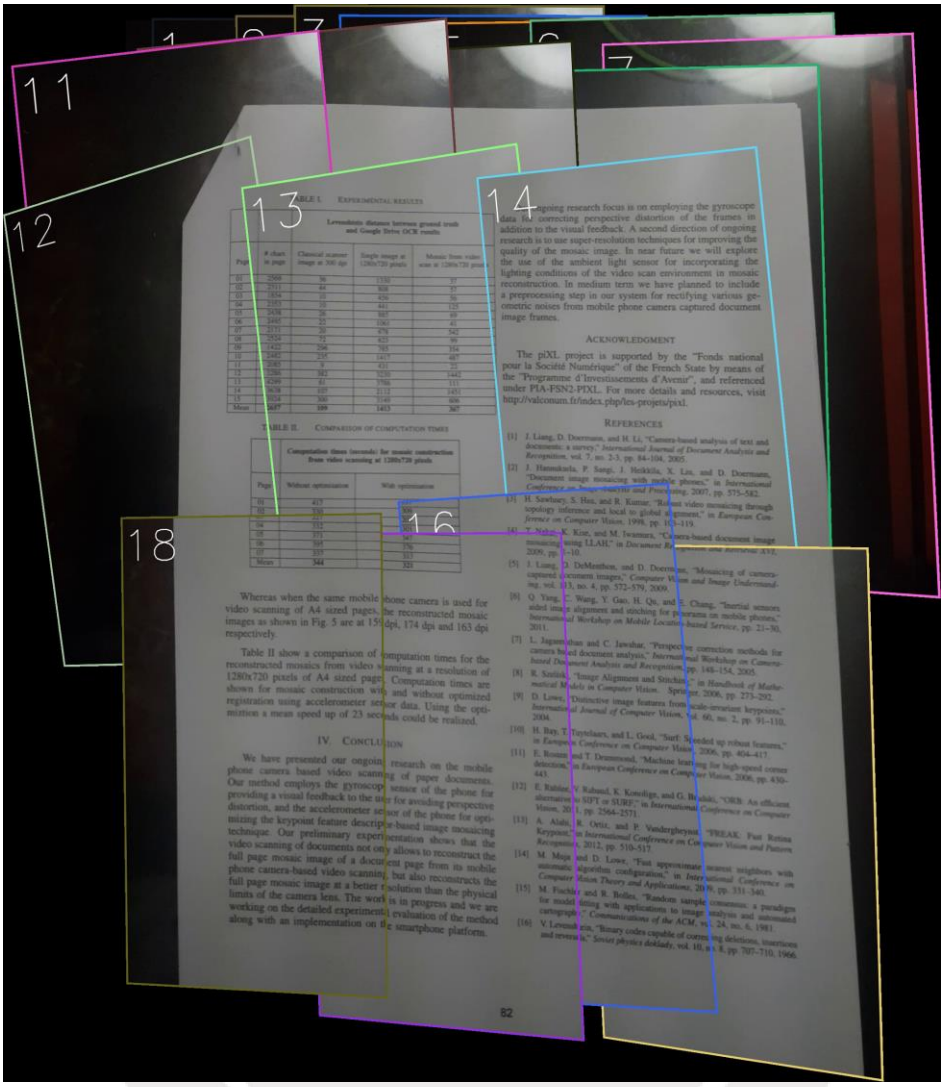
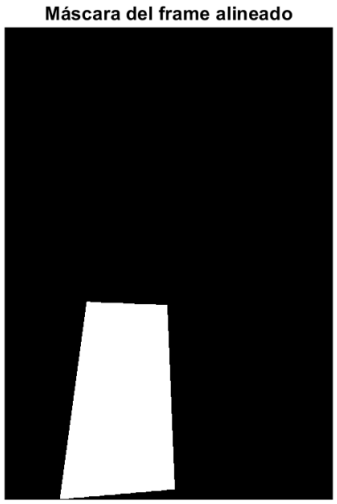


Figura 3.16: Mosaico de documento preliminar compuesto por 18 *frames*. Nótese la superposición de *frames*. Elaboración propia.

3.7.2. Superposición de *frames*

En un video puede ocurrir que una determinada zona de un documento se grabe repetidas veces a lo largo de su duración, ello puede llevar a que muchos *frames* se añadan al mosaico unos superpuestos a otros. Si esto ocurre, dichos traslapes excesivos pueden generar menor precisión durante la detección de puntos característicos y cálculo de homografías llevando el resultado a una menor calidad. Por ello es recomendable que la composición de cada mosaico se realice evitando realizar traslapes innecesarios.

Para reducir las superposiciones se debe determinar qué porcentaje del *frame* que se busca añadir está contenido en el mosaico preliminar, a ello se le denomina porcentaje de coincidencia. Con las siguientes figuras se puede ilustrar el cálculo de dicho porcentaje.



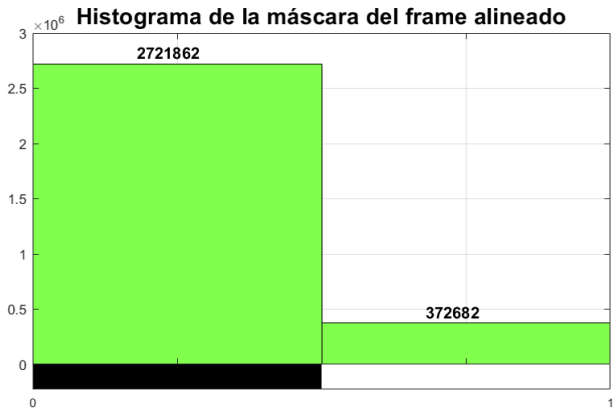
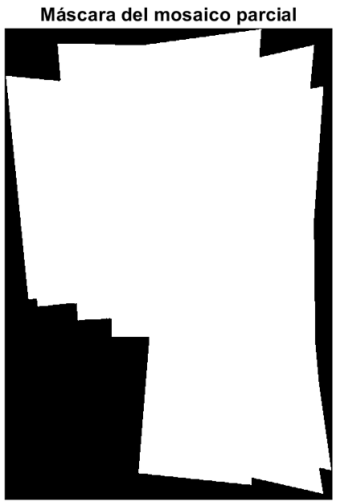


Figura 3.17: Máscara del frame alineado, listo para ser añadido al mosaico parcial, junto a su histograma. Elaboración propia.



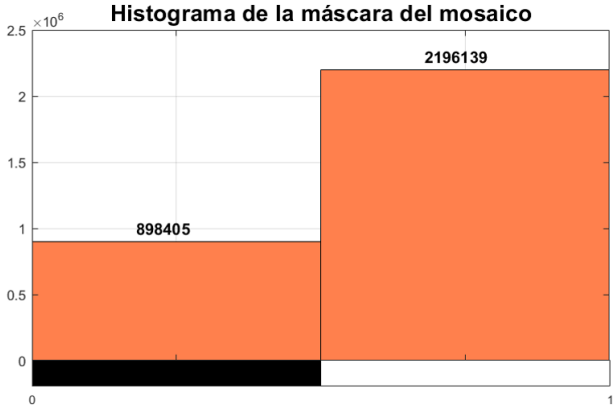


Figura 3.18: Mosaico parcial junto a su histograma. Elaboración propia.

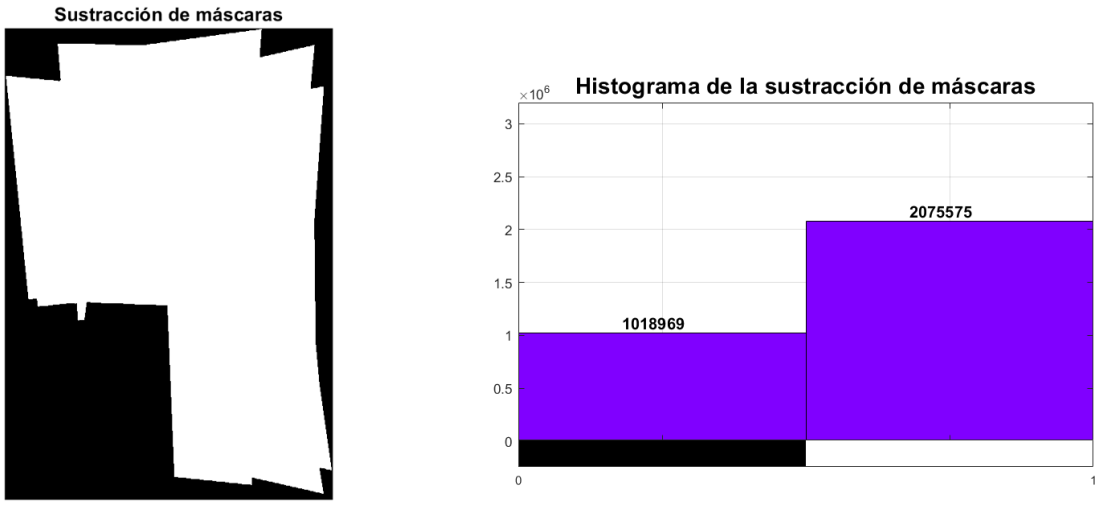


Figura 3.19: Máscara del mosaico parcial a la cual se le ha sustraído la máscara del *frame* alineado, junto a su histograma. Elaboración propia.

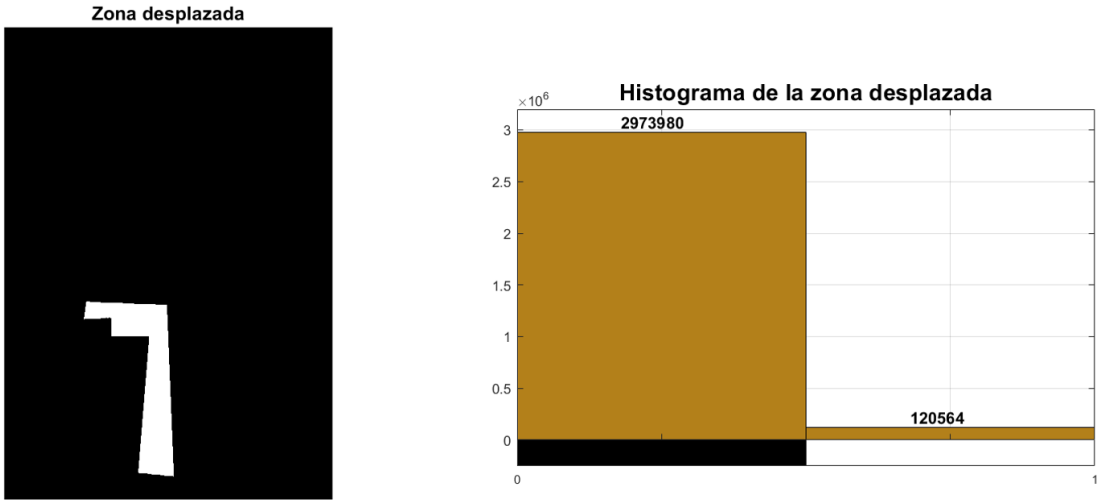


Figura 3.20: Máscara que representa la zona de intersección entre ambas máscaras y la cual se desplaza hacia 0 luego de la sustracción. Elaboración propia.

La máscara del *frame* alineado (Figura 3.17) representa la posición en la cual se añadirá el *frame* ya alineado mientras que, la máscara del mosaico parcial (Figura 3.18) representa el área total que abarcan los *frames* debidamente unidos para componer la imagen resultante. Con estas máscaras se puede calcular la imagen resultante de su sustracción mediante a siguiente función:

$$Subs = cv.substract(mosaic, frame) \quad (3.3)$$

en donde, Subs representa a la imagen resultante de la operación entre el mosaico y el *frame* (Figura 3.19). Esta función permite realizar la operación de sustracción incluyendo la saturación de la imagen resultante; es decir, si la sustracción de dos píxeles produjera un valor menor a 0, este se limita a 0 (color negro en escala de grises). Como se puede apreciar en el histograma de la Figura 3.19, los píxeles con valor 0 han aumentado su número debido a la coincidencia existente entre el mosaico y el *frame*. Este desplazamiento se puede apreciar en el histograma de la Figura 3.20, allí se cuentan todos los píxeles que se han acercado a 0. El número total de puntos desplazados se denota por *K* y el total de píxeles del *frame* sin contar los ubicados en 0 (no se considera el marco de color negro), por *T1*. Entonces, el porcentaje del *frame* que se encuentra contenido en el mosaico parcial se puede hallar mediante la ecuación 3.4:

$$\%Coincidencia = \frac{K}{T1} \cdot 100\% \quad (3.4)$$

Con este valor calculado se ha establecido un rango límite para evitar los traslapes innecesarios. Si el *frame* que se añadirá al mosaico tiene una coincidencia sobre el 80% de su contenido, pero no una nitidez por sobre 150, entonces este no será añadido al mosaico y se continuará con los demás *frames* de la secuencia.

3.8. Programación en Android

De la etapa de diseño se desprenden los siguientes requerimientos para hacer posible la implementación y ejecución del algoritmo en teléfonos móviles:

- Debe contar con una cámara digital con sensor de 13 megapíxeles y soporte para API Camera2 de Android: Actualmente, la mayoría de teléfonos de gama media disponen de una cámara posterior con una cantidad de píxeles no menor a 13 MP; dicha capacidad permite capturar imágenes con una resolución máxima de 4200 x 3100 píxeles.
- Debe brindar soporte para OpenGL ES 3.1 o posterior: Esta API es una versión para sistemas embebidos de la aplicación OpenGL para desarrollo de gráficos 2D y 3D. La finalidad de su uso es ejecutar el renderizado de imágenes en un subproceso dedicado para no afectar la ejecución del subproceso de Interfaz de Usuario (UI por sus siglas en inglés).

- Debe contar con una versión de Android igual o superior a 5.0, puesto que esta versión brinda soporte a API Camera2 y a OpenGL ES 3.1.
- Opcionalmente, debe poseer un LED para mejorar la iluminación al capturar imágenes y desestimar la necesidad de incrementar la ganancia ISO al capturar imágenes.

Durante la implementación de la aplicación se sintetizaron las etapas de grabación, verificación de nitidez y extracción de *frames* en una sola; es decir, a diferencia del algoritmo implementado en Python y descrito en la Figura 3.2 donde la entrada al proceso es un video pregrabado (@ 30 fps) que luego es analizado cuadro a cuadro en busca de los más nítidos y luego se extraen estos, en el algoritmo implementado en Android se analiza continuamente la nitidez de las imágenes que ingresan a través de la cámara, se esté grabando o no, y solo cuando se decide iniciar el proceso de grabación se almacenarán las que superen el umbral establecido para el Laplaciano. En la Figura 3.21 se ilustra esta diferencia.

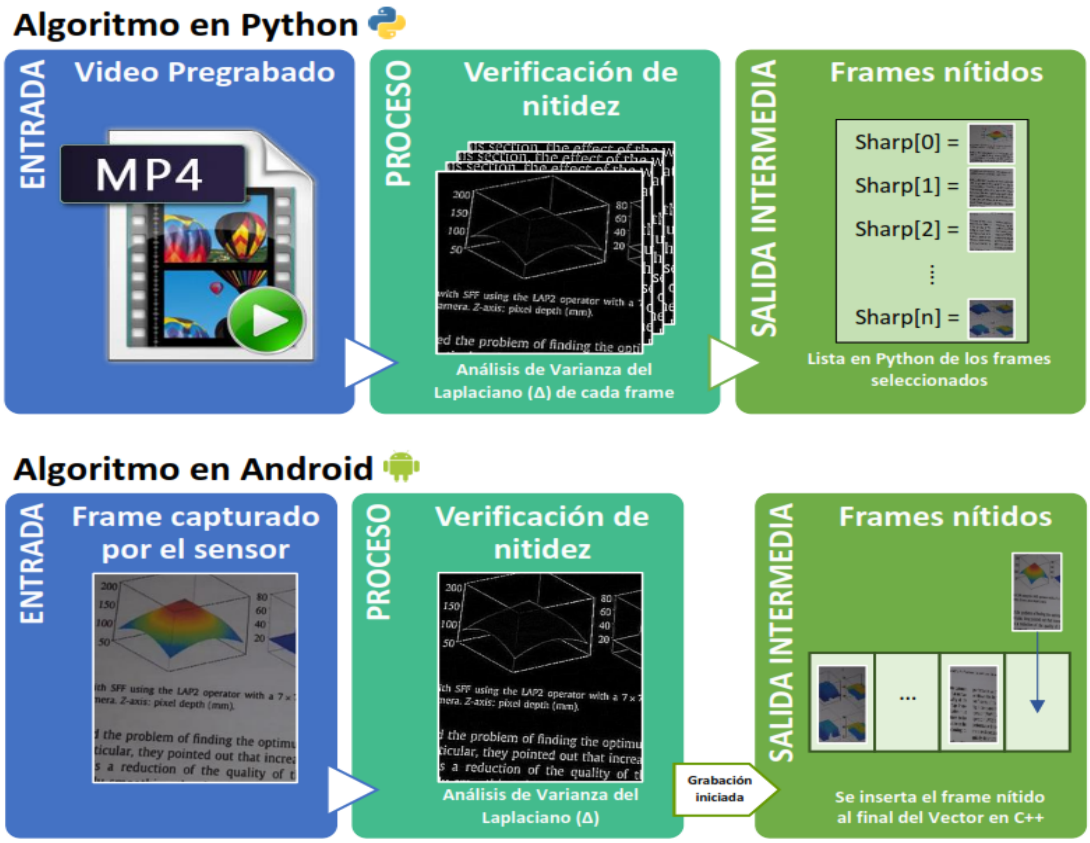


Figura 3.21: La principal diferencia entre los algoritmos, implementados en Python y Android, radica en la entrada de los mismos. En Python se utiliza un video para analizar todos los frames en busca de los más nítidos; mientras que, en Android se captura y analiza continuamente los frames capturados por el sensor en busca de nitidez, solo si se ha iniciado la grabación se almacenarán estos.

Debido a que se procesa cada cuadro que ingresa a la cámara, y ello consume tiempo de procesamiento, no se puede alcanzar una tasa de 30 fps; sin embargo, resulta más conveniente puesto que no se almacenarán cuadros innecesarios con baja nitidez. Para mejorar la tasa de captura es recomendable ejecutar el procesamiento de los cuadros en un subproceso, o thread, dedicado; es por ello, la importancia de usar la API OpenGL ES que facilita el renderizado de las imágenes procesadas con OpenCV.

Finalmente, la configuración de los parámetros de exposición se realiza a través del objeto CameraRequest, el cual permite el control manual de la captura de imágenes de la cámara.

A continuación, se muestra una captura de la interfaz desarrollada para la aplicación en Android (véase Figura 3.22).

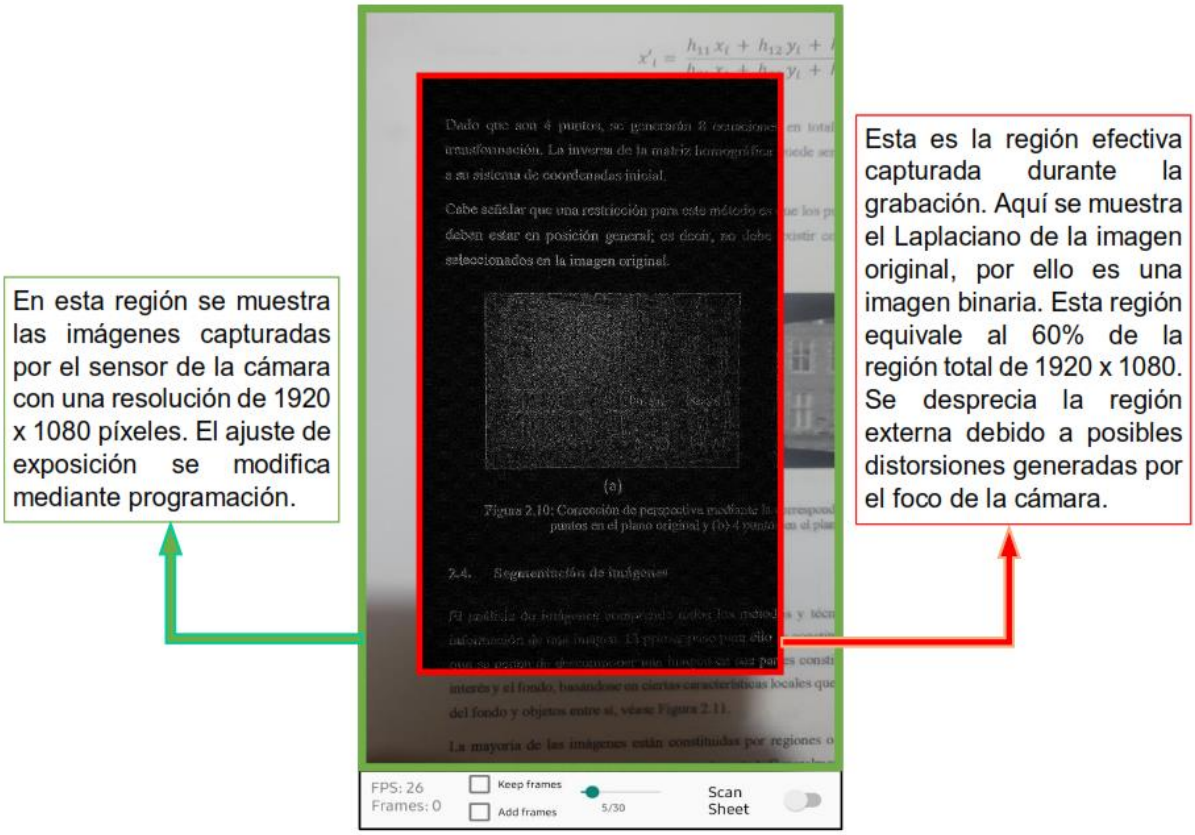


Figura 3.22: Interfaz desarrollada en Android para el uso del algoritmo planteado. En verde se muestra la región total que abarca la imagen capturada por el sensor de la cámara y, en rojo, la región capturada cuando se inicia la grabación (se puede apreciar el Laplaciano de la imagen original).

En la barra de control se incluyeron opciones que permitirán ejecutar las pruebas necesarias con el algoritmo, en la Figura 3.23 se explican dichas funciones.

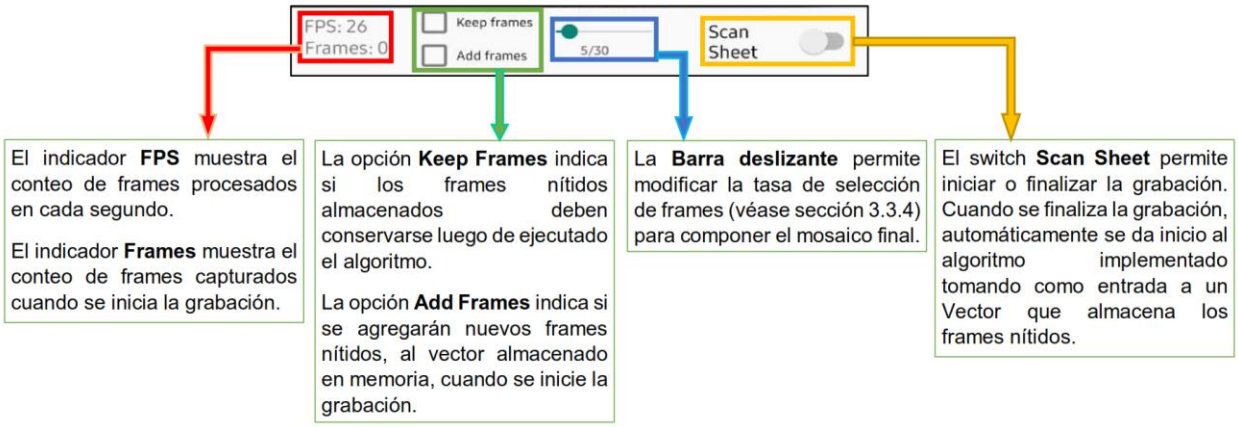


Figura 3.23: Descripción de la barra de control de la interfaz.

Capítulo 4
Pruebas y resultados

4.1. Aplicación en el dataset

La etapa de pruebas inicia con la aplicación del algoritmo creado en Python, dado que facilita el reajuste de los parámetros planteados en la etapa de diseño. El resultado final entregado por el algoritmo es un mosaico, compuesto por fragmentos extraídos de un video, y de la cual se podrá extraer la mayor parte del texto original impreso en el documento objetivo. Como se hizo presente en el capítulo previo, se ha elaborado 21 videos bajo los parámetros de control preestablecidos (dirección de grabación, velocidad de obturación de la cámara, etc.). Cada uno de ellos será procesado por el algoritmo propuesto y, finalmente, el texto extraído de los mosaicos resultantes, mediante un algoritmo OCR externo, será evaluado bajo una determinada métrica; para este contexto se empleará la distancia de Levenshtein. Para identificar correctamente los documentos empleados para estas pruebas, en la tabla 4.1 se presenta brevemente el nombre del documento y el número que permite identificarlo en la tabla 4.2.

Tabla 4.1: Documentos empleados para las pruebas y su número de identificación. Elaboración propia.

N° de documento	Nombre del documento
1	Dataset SmartDoc 2017 - Paper 01
2	Dataset SmartDoc 2017 - Paper 02
3	Primera Encuesta Nacional sobre Discapacidad 2012 - Página 3
4	Image Alignment and Stitching: A Tutorial - Página 1
5	Discapacidad y accesibilidad: La dimensión desconocida - Página 22
6	Analysis of focus measure operators - Página 7
7	Mobile phone camera-based video scanning of paper documents - Página 7
8	La didáctica del Braille más allá del código - Página 167
9	Modelling and control of a buck converter - Página 11
10	Water production from air using multi-shelves solar glass pyramid - Página 6

4.1.1. Mosaicos compuestos

Las imágenes resultantes, en primera instancia de pruebas, se componen empleando la totalidad de *frames* disponibles en cada video; es decir, la tasa de selección de *frames* (tratada en la sección

3.3.3) es igual a 1. En la Figura 4.1 se aprecia 4 de los mosaicos compuestos bajo distintas direcciones de grabación.

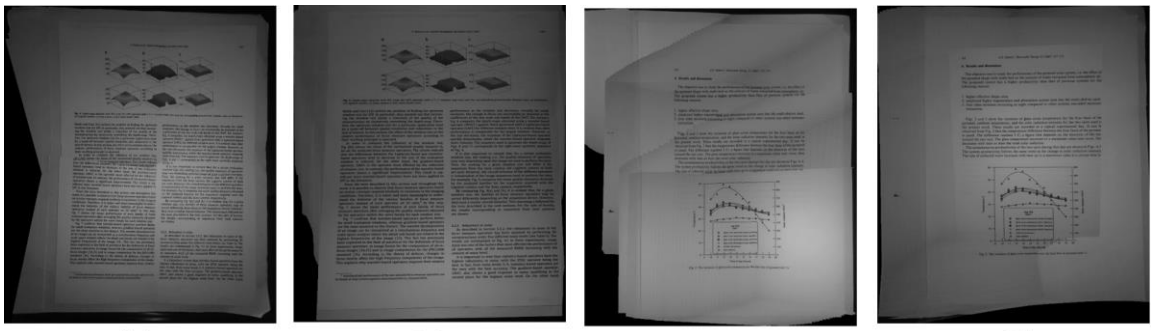


Figura 4.1: Mosaicos compuestos. (a) Mosaico aplicando rotación sobre el eje, (b) mosaico vertical, (c) mosaico horizontal y (d) mosaico helicoidal. Elaboración propia.

Si bien en estos mosaicos aún se puede apreciar el contraste en las zonas de traslape y la diferencia de brillo entre *frames*, durante la experimentación se comprobó que estas no afectan la detección de los caracteres durante la aplicación de OCR. Prueba de ello se obtuvo al realizar la composición de un mosaico binarizado a partir de *frames* filtrados (se les aplicó filtrado bilateral para conservar la nitidez de caracteres) y binarizados (mediante binarización adaptativa). Dicha etapa de pre procesamiento para cada *frame* se puede descartar dado que solo aumenta el tiempo de ejecución del algoritmo y no afecta severamente a la detección de texto. Cabe señalar que la robustez de los puntos característicos ante variaciones de luminosidad permite hallar las matrices homográficas independientemente de estas variaciones. En la sección 4.4 se ilustra este hecho.

En la Figura 4.2 se puede apreciar el proceso realizado (ver sección 3.1) para componer un mosaico desde el video 2110186.

4.1.2. Extracción de texto

El objetivo de estas pruebas es determinar la adecuada correspondencia existente entre el texto que se puede extraer del mosaico compuesto y el texto original, para ello se requiere de una herramienta OCR que procese cada imagen en búsqueda de texto. Google Drive cuenta con una herramienta OCR, de libre acceso, que será empleada para este fin. Cada uno de los mosaicos fue procesado y el texto detectado se almacenó en formato TXT para luego ser empleados en el algoritmo de Levenshtein. Cabe mencionar que el contenido presente en tablas y figuras del documento se ha

omitido durante la conversión del texto. En la Figura 4.3 se ilustra la conversión del mosaico a texto.

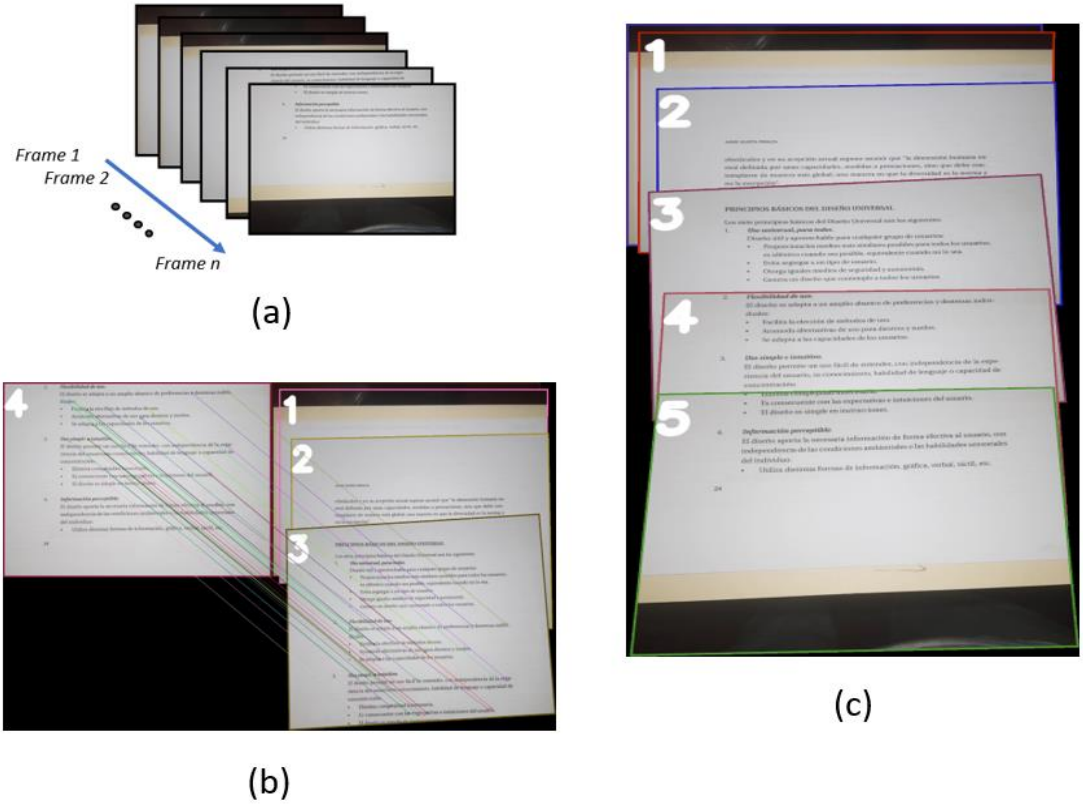


Figura 4.2: Proceso de composición de los mosaicos. (a) Clasificación y selección de frames según su nitidez, (b) detección y emparejamiento de características, y (c) alineación de frames y composición del mosaico.

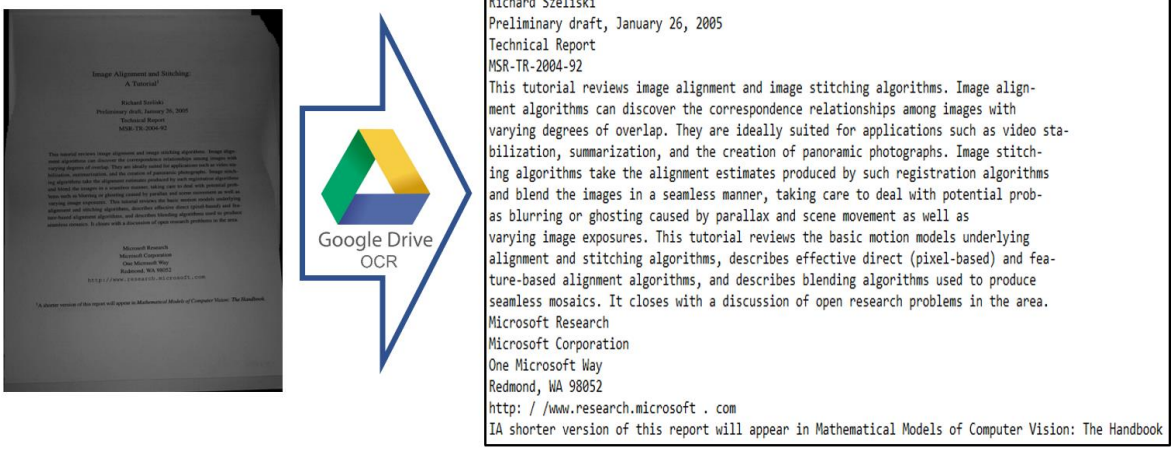


Figura 4.3: Conversión del mosaico compuesto a formato TXT para aplicar el algoritmo de Levenshtein. Elaboración propia.

4.1.3. Distancia de Levenshtein

El texto extraído y almacenado será comparado con el texto original mediante la distancia de Levenshtein o de edición. Esta métrica para cadenas de caracteres permite medir la diferencia entre dos secuencias, su valor representa el mínimo número caracteres que deben ser editados (insertados, eliminados o sustituidos) en una cadena para que sea idéntica a otra. En Python 3.6 se cuenta con la librería Levenshtein, esta incluye la función del mismo nombre que permite comparar documentos de texto y calcular la distancia de edición entre estos. Para comparar los archivos .TXT es necesario dar el formato necesario a cada uno de tal forma que dos líneas del mismo fragmento de texto se encuentren en la misma posición en cada archivo.

4.1.4. Resultados iniciales

En la tabla 4.2 se muestran la distancia de edición calculada para cada uno de los mosaicos compuestos. Para esta prueba se empleó una tasa de selección igual a 1.

Tabla 4.2: Resultados para una tasa de selección igual a 1. Elaboración propia.

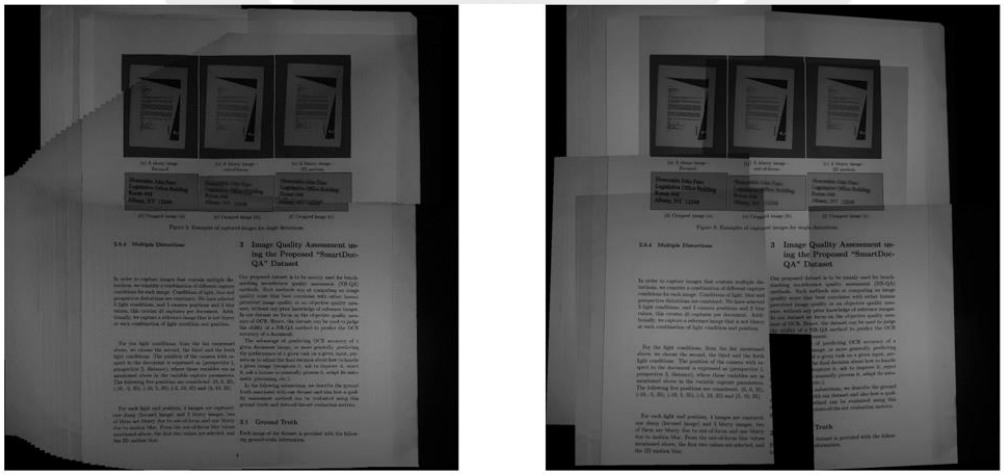
Video	N° de documento	Dirección de grabación	N° de frames	Duración (seg.)	N° caracteres	Distancia de Levenshtein	% error
2410182	3	R. sobre eje	112	3.73	2208	3	0.14%
2410181	3	Horizontal	354	11.80	2208	63	2.85%
2410187	3	Vertical	65	2.17	2208	5	0.23%
2310181	1	Helicoidal	242	8.07	3943	14	0.36%
2210181	5	Horizontal	400	13.33	1582	6	0.38%
2110186	5	Vertical	158	5.27	1582	6	0.38%
2210188	7	R. sobre eje	143	4.77	5080	77	1.52%
3110181	7	Vertical	97	3.23	5080	28	0.55%
3110182	9	R. sobre eje	154	5.13	1796	35	1.95%
0111181	6	R. sobre eje	243	8.10	5506	51	0.93%
0111182	6	Vertical	158	5.27	5506	85	1.54%
0111185	8	Vertical	116	3.87	1839	11	0.60%
0111186	8	Horizontal	276	9.20	1839	14	0.76%
0111187	2	Horizontal	192	6.40	2638	25	0.95%
0211181	4	Vertical	91	3.03	1273	11	0.86%
0211182	4	Horizontal	234	7.80	1273	14	1.10%
1011181	10	Horizontal	295	9.83	1525	30	1.97%
1011182	10	R. sobre eje	200	6.67	1525	10	0.66%
1011183	10	Vertical	154	5.13	1525	6	0.39%
1011185	4	R. sobre eje	172	5.73	1273	18	1.41%
0111188	2	Vertical	84	2.80	2638	14	0.53%

De estos resultados se puede destacar las bajas tasas de error en el texto detectado en los mosaicos. Dicha tasa puede ser tan baja como 0.14% en el mejor de los casos. Por otro lado, si bien se puede emplear todos los *frames* de cada video para realizar la composición de una imagen, el tiempo de procesamiento puede elevarse considerablemente dependiendo de la resolución y duración de cada video; debido a ello, es necesario corroborar que el algoritmo puede componer un mosaico empleando una baja cantidad de *frames*.

4.2. Efecto de la tasa de selección

Las siguientes pruebas se realizaron modificando el número de *frames* empleados para componer cada mosaico y comprobar el efecto que tiene este parámetro en la tasa de error de caracteres. Es necesario mencionar que la ejecución del algoritmo se realizó en una PC con un procesador Intel Core i7 7700 y que los tiempos de ejecución fueron medidos empleando la librería Time para Python 3.6.

El principal efecto de reducir la tasa de selección es un decremento en el tiempo de procesamiento; sin embargo, los errores de proyección tienden a aumentar para los casos de grabaciones helicoidales y de rotación sobre el eje. Ello se debe a la reducción de coincidencia entre *frames* como se presentó en la sección 3.3.3. Otro factor que incrementa la distancia de edición es la pérdida de contenido que puede ocurrir si la tasa de selección es demasiado alta. Este problema se ilustra en la Figura 4.4.



(a) (b)

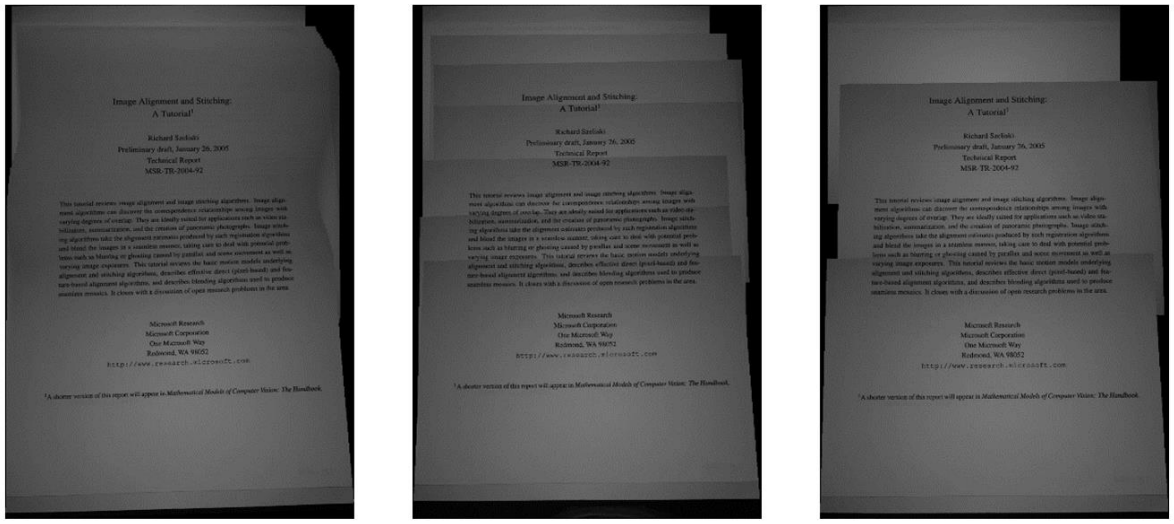
Figura 4.4: Pérdida de información debido a una tasa de selección demasiado amplia. En (a) tasa de selección igual a 1 y en (b) tasa de selección igual a 30. Elaboración propia.

En la tabla 4.3 se presenta una parte de los resultados obtenidos al variar la tasa de selección durante la composición de cada mosaico. Para poder optimizar el tiempo de ejecución del algoritmo es necesario identificar la dirección de grabación óptima a fin de reducir el tiempo de grabación y, además, se busca emplear la menor cantidad de *frames* para poder componer el resultado. Para ello es necesario evaluar las variaciones en la distancia de edición del texto resultante, el objetivo es tener un resultado estable independientemente de la tasa seleccionada. En la tabla 4.3 se resalta en verde las secuencias de valores más estables ante las variaciones.

Tabla 4.3: Variación de la distancia de edición debido a la tasa de selección (tiempo medido en segundos). Elaboración propia.

Video		Tasa de selección						
		1	5	10	15	20	30	45
2410182	D. Lev.	3	47	5	23	10	4	44
	Tiempo	378.3	70.9	34	20.77	21.8	14.22	6.2
	Error	0.14%	2.13%	0.23%	1.04%	0.45%	0.18%	1.99%
2410181	D. Lev.	63	58	60	65	66	60	59
	Tiempo	1215.3	239.33	114.3	76.93	68.71	43.56	29.17
	Error	2.85%	2.63%	2.72%	2.94%	2.99%	2.72%	2.67%
2410187	D. Lev.	5	3	2	3	3	1	3
	Tiempo	165.44	30.31	12.29	7.32	8.22	4.8	6.5
	Error	0.23%	0.14%	0.09%	0.14%	0.14%	0.05%	0.14%
2210181	D. Lev.	6	3	2	7	3	4	6
	Tiempo	573.59	114.85	55.37	36.18	30.82	18.89	12.64
	Error	0.38%	0.19%	0.13%	0.44%	0.19%	0.25%	0.38%
0111181	D. Lev.	51	74	68	267	70	35	35
	Tiempo	1075.4	210.22	102.26	69.01	49.32	32.61	18.84
	Error	0.93%	1.34%	1.24%	4.85%	1.27%	0.64%	0.64%
1011183	D. Lev.	6	8	7	7	4	5	9
	Tiempo	153.62	29.37	13.27	8.56	9.71	4.93	2.85
	Error	0.39%	0.52%	0.46%	0.46%	0.26%	0.33%	0.59%
1011185	D. Lev.	18	15	15	13	13	29	39
	Tiempo	335.32	61.17	30.33	18.72	19.17	12.2	7.66
	Error	1.41%	1.18%	1.18%	1.02%	1.02%	2.28%	3.06%
0111188	D. Lev.	14	25	31	27	16	13	8
	Tiempo	242.3	44.78	17.61	13.16	12.05	9.29	6
	Error	0.53%	0.95%	1.18%	1.02%	0.61%	0.49%	0.30%

Como se aprecia en la tabla previa, la composición de un mosaico vertical se puede realizar empleando una cantidad reducida de *frames* sin incrementar significativamente el porcentaje de error en el texto detectado; por ello, solo este método de grabación será considerado para la etapa de pruebas en Android. En la Figura 4.5 se ilustra el efecto de dicha reducción sobre un mosaico vertical.



(a) (b) (c)

Figura 4.5: Mosaicos verticales del video 0211181 a diferentes tasas de selección. En (a) se emplearon 91 *frames*, en (b) se emplearon 7 y en (c) se emplearon 3.

Cabe mencionar que los videos 2410181 y 2210181 fueron grabados aplicando un movimiento horizontal, mientras que los videos 2410187 y 1011183 se realizaron con un movimiento vertical. Para todos ellos se estableció la ganancia ISO de la cámara en 200 y la velocidad de obturación en 1/750 seg., salvo para el último mencionado que fue realizado en 1/1000 seg.

4.3. Pruebas en Android

Las pruebas finales se realizarán de forma directa en un teléfono móvil mediante la ejecución de la aplicación implementada para el sistema Android (versión 7.0 en adelante); por ello, no se emplearán los videos adquiridos para el dataset, sino que se adquirirán imágenes directamente de la cámara y se aplicará el algoritmo a ellas. El tiempo de ejecución (solo para componer el mosaico) se medirá empleando la librería *time.h* para C++. El equipo empleado en las pruebas es un teléfono Huawei Y7 2018 que cuenta con las siguientes características:

- Resolución de pantalla: 1440 x 720 píxeles.
- Resolución en modo foto de la cámara trasera (principal): 13 megapíxeles.
- FPS máximo de grabación (cámara principal): 30 fps.
- Sistema Operativo: Android 8.0 Oreo.
- Procesador: Qualcomm Snapdragon 430 MSM8937 (1.4 GHz).
- Memoria RAM: 2 GB.
- Memoria interna: 16 GB.
- Soporte para API Camera2 en ambas cámaras.
- Soporte para OpenGL ES 3.1.

Las grabaciones se realizarán a movimiento lento con un ISO de 100 y tiempo de exposición programado en 1/100 seg.; también se empleará el flash para mejorar las condiciones de luminosidad en todo contexto. Adicionalmente, la tasa de selección continuará fijada en 1 de cada 15 frames capturados. Finalmente, como se señaló en el inciso 3.8, se capturará imágenes a una resolución FHD (1920 x 1080 píxeles). Cabe señalar que la evaluación del texto detectado se realizará empleando nuevamente el motor OCR de Google Drive y la librería Levenshtein para Python 3.6. En la Tabla 4.4 se muestran los resultados de estas pruebas.

Tabla 4.4: Resultados de las pruebas realizadas en Android (Huawei Y7 2018). Elaboración propia.

Documento registrado	N° de frames capturados	Tiempo de ejecución	Distancia de Levenshtein	% Error
1	72	5.78	91	2.31%
2	76	5.3	44	1.67%
3	63	5.37	36	1.63%
4	68	3.65	19	1.49%
5	63	4.31	9	0.57%
6	94	8.81	150	2.72%
7	92	9.12	232	4.57%
8	64	6.01	3	0.16%
9	95	7.56	10	0.56%
10	65	3.46	19	1.25%

Del cuadro previo se destaca la baja tasa de error presente en el texto detectado de los mosaicos compuestos, siendo no mayor al 5% en el peor de los casos y con valores semejantes a los obtenidos durante las pruebas en Python.

4.4. Recomendaciones de grabación

Se recomienda realizar las grabaciones en ambientes bien iluminados para poder compensar la reducción de luminosidad producto del incremento en la velocidad de obturación. El rango de la velocidad de obturación puede programarse entre 1/100 seg. y 1/500 seg, dependiendo de la resolución del sensor de la cámara del teléfono y de la apertura del lente. No se recomienda el incremento de la ganancia ISO por sobre 350, dado que amplifica tanto el brillo como el ruido que registra el sensor de la cámara. En el caso que fuese necesario, también se puede hacer uso del flash incorporado en todos los teléfonos de gama media para compensar la reducción de luminosidad.

La resolución de grabación recomendada es 1920 x 1080 (FHD) píxeles, pues permite componer un mosaico con la calidad mínima suficiente para detectar el texto mediante OCR. El contenido del documento objetivo puede contener sólo texto o incluir imágenes y tablas; sin embargo, se debe evitar las fuentes con tamaños reducidos para evitar posibles distorsiones de caracteres causadas por la proyección de los frames.

En base a los datos presentados en las tablas 4.2, 4.3 y 4.4, se puede afirmar que la dirección de grabación óptima es la vertical, puesto que reduce el tiempo que toma realizar la grabación y, también, permite componer un mosaico con la menor cantidad de frames. Ello a su vez facilita la carga de procesamiento y mejora el tiempo de ejecución del algoritmo, ya sea en un computador de escritorio o en un teléfono móvil.

Conclusiones

- En el presente trabajo se ha implementado un algoritmo de formación de imágenes completas de documentos con texto impreso a partir del procesamiento de imágenes con fragmentos de estos que fueron extraídas de un vídeo. El algoritmo ha demostrado ser funcional tanto en su versión para computadoras de escritorio como en su versión para teléfonos con SO Android. Para esta última, si bien las pruebas se realizaron en un teléfono de gama baja y relativamente asequible, los resultados demuestran que la calidad del mosaico y el tiempo de ejecución se encuentran dentro de un margen aceptable para ser empleado en aplicativos de lectura para personas con problemas visuales. Sin embargo, cabe señalar que el algoritmo desarrollado puede aprovechar características de dispositivos superiores en gama, tanto de su mejor capacidad de procesamiento como de la resolución del sensor de la cámara.
- La modificación en el modo de captura de imágenes para la versión de Android permite reducir el tiempo de ejecución total del algoritmo, pues a diferencia de la versión de Python, se procesa imágenes ya clasificadas como nítidas y no se requiere analizar un vídeo en busca de ellas.
- El tiempo necesario para componer un mosaico se puede reducir si se procesa una menor cantidad de imágenes durante la creación. Para este fin, la robustez en detección de puntos característicos es de vital importancia. Además, la aplicación de una adecuada técnica de grabación permite mejorar el rendimiento del algoritmo en términos de tiempo y calidad de la imagen resultante; se ha corroborado que un documento grabado de forma vertical requiere una menor cantidad de frames para componer un mosaico y, además, reduce los errores de proyección de frames. Es necesario entrenar al usuario del algoritmo para que adquiera la pericia necesaria para realizar una grabación cercana a la óptima recomendada; sin embargo, los cuatro métodos de grabación propuestos en este trabajo son soportados por el algoritmo, dado que este cuenta con la robustez necesaria.
- El uso de técnicas más rápidas en detección de puntos característicos (como lo presentado al aplicar ORB) causan mayor imprecisión en la composición de mosaicos con elementos repetitivos. En imágenes que solo contienen texto, ocurre que los puntos característicos calculados con un radio menor al tamaño de una letra se pueden emparejar de forma errónea

a letras pertenecientes a un párrafo distinto en otra imagen. Por ello, es necesario delimitar el tamaño mínimo del radio de estos dependiendo del método de detección de los mismos (SIFT o SURF).

- La etapa de corrección de luminosidad entre frames puede ser obviada dado que no afecta severamente la detección de caracteres en el algoritmo OCR.

Recomendaciones

- Agregar una etapa de guiado activo durante la grabación de los videos puede reducir la destreza necesaria, por parte del usuario, para realizar una grabación óptima.

- La corrección de perspectiva de cada frames, de tal modo que fuesen paralelos a la pantalla del dispositivo, puede reducir el error total de proyección existente en el resultado final.

- En caso fuese necesario, se puede añadir a este algoritmo una etapa de composición selectiva de mosaicos (véase [28]) para reducir la evidencia de traslape en el resultado final.

- Posteriormente, se puede desarrollar un algoritmo de OCR que trabaje en conjunto con el presentado en esta tesis.

Bibliografía

[1] Instituto Nacional de Estadística e Informática, "Primera Encuesta Especializada sobre Discapacidad", Lima, Perú, 2014.

[2] J. Huerta, *Discapacidad y accesibilidad: La dimensión desconocida*, 1ª ed. Lima, Perú, 2006.

[3] Compañía peruana de estudios de mercados y opinión pública, "Evolución del mercado de smartphone y smart TV en el Perú", Lima, Perú, 2017.

[4] Comisión Braille Española, *La didáctica del Braille más allá del código*, 1ª ed. Madrid, España, 2015.

[5] K. La Hoz, "Solo 50 mil peruanos ciegos leen en Braille", Publimetro, Lima, Perú, 2013.

[6] T. Anupama y E. Rufus, "Alternate braille display designs: A review", Vellore, TN, India, 2016.

[7] Instituto Nacional de Estadística e Informática, "Estadísticas de las tecnologías de información y comunicación en los hogares", Lima, Perú, 2017.

[8] J. Liang, D. DeMenthon, y D. Doermann, "Camera-Based Document Image Mosaicing", Hong Kong, China, 2006.

[9] M. Cutter y R. Manduchi, "Towards Mobile OCR: How to take a good picture of a document without sight", Santa Cruz, California, USA, 2015.

[10] M. Cutter y R. Manduchi, "Improving the accessibility of mobile OCR apps via interactive modalities", Santa Cruz, California, USA, 2017.

[11] A. Zandifar, R. Duraiswami, A. Chahine, y L. S. Davis, "A video based interface to textual information for the visually impaired", College Park, MD, USA, 2003.

[12] Freedom Scientific Inc., "Blindness Solutions", 2018. [En línea]. Disponible en: http://www.freedomscientific.com/Products/Blindness/JAWS. [Accedido: 01-ago-2020].

[13] MaxiAids.com, "Scan-Read Devices", 2018. [En línea]. Disponible en: https://www.maxiaids.com/scan-read-devices. [Accedido: 01-ago-2020].

[14] N. Stamatopoulos, B. Gatos, y A. Kesidis, "Automatic Borders Detection of Camera Document Images", Atenas, Grecia, 2007.

[15] M. Shamqoli y H. Khosravi, "Border detection of document images scanned from large books", Shahrood, Irán, 2013.

[16] Sensor Tower, "App Intelligence", 2018. [En línea]. Disponible en: https://sensortower.com/. [Accedido: 24-abr-2018].

[17] M. Ligang y Y. Yongjuan, "Automatic Document Image Mosaicing Algorithm with Hand-held Camera", Beijing, China, 2011.

[18] J. Chazalon *et al*., "SmartDoc 2017 Video Capture: Mobile Document Acquisition in Video Mode", Paris, Francia, 2017.

[19] N. Nakajima, A. Iketani, and T. Sato, "Video mosaicing for document imaging", Ikoma, Nara, Japón, 2007.

[20] American Foundation for the Blind, "Screen Readers", 2018. [En línea]. Disponible en: http://www.afb.org/prodBrowseCatResults.aspx?CatID=49. [Accedido: 25-abr-2018].

[21] Independent Living Aids LLC, "Vision Products", 2018. [En línea]. Disponible en: https://www.independentliving.com/category/VISION. [Accedido: 01-ago-2020].

[22] M. Berríos, "160 alumnos invidentes son educados entre graves carencias", Diario La República, Lima, Perú, 17-jun-2013.

[23] L. Cuartero, "66,3% de los jóvenes con discapacidad abandona sus estudios secundarios", Diario Correo, Lima, Perú, 11-mar-2018.

[24] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1ª ed. Londres, Reino Unido: Springer-Verlag London, 2010.

[25] Z. Pan y M. Wang, "A New Method of Shredded Paper Image Stitching and Restoration", Wuhan, China, 2017.

[26] A. Cooke, "The Exposure Triangle: Understanding How Aperture, Shutter Speed, and ISO Work Together", 2015. [En línea]. Disponible en: https://fstoppers.com/education/exposure-triangle-understanding-how-apertureshutter-speed-and-iso-work-together-72878.

[27] R. Szeliski, "Image Alignment and Stitching: A Tutorial", Microsoft Research, Redmond, WA, USA, 2005.

[28] J. Liang, D. DeMenthon, y D. Doermann, "Mosaicing of Camera-captured Document Images", College Park, MD, USA, 2008.

[29] J. Muñóz, "Segmentación de imágenes", Málaga, España, 2010.

[30] L. Shapiro y G. Stockman, *Computer Vision*, 1ª ed. Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[31] S. Lu y C. Lim, "Camera Document Restoration for OCR", Kent Ridge, Singapur, 2005.

[32] R. Hartley y A. Zisserman, *Multiple View Geometry in computer vision*, 2ª ed. NY, USA: Cambridge University Press, 2004.

[33] OpenCV Team, "About OpenCV", 2018. [En línea]. Disponible en: https://opencv.org/about.html. [Accedido: 01-ago-2020].

[34] R. Gonzales y R. Woods, *Digital Image Processing*, 4ª ed. Upper Saddle River, NJ, USA: Pearson, 2017.

[35] B. Marcotegui y A. Belhedi, “Adaptive scene-text binarization on images captured by smartphones”, Fontainebleau, Francia, 2015.

[36] J. Pech, G. Cristobal, J. Chamorro, y J. Fernandez, “Diatom autofocusing in brightfield microscopy: a comparative study”, Madrid, España, 2000.

[37] E. Karami, S. Prasad, y M. Shehata, “Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images”, St. John’s, Canadá, 2015.

[38] D. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints”, Vancouver, Canadá, 2004.

[39] S. Pravenaa y R. Menaka, “A Methodical Review on Image Stitching and Video Stitching Techniques”, Chennai, India, 2016.

Anexos

Anexo A. Algoritmo en Python

```
#Stitch_Mosaico.py
#Autor: Jose Ramirez Diaz
#Esta aplicacion de Python permite componer un mosaico de documento
#Partiendo de un video previamente grbado con un telefono movil
#Solo se debe indicar la direccion del video y se obtendra la imagen final

#Llamado a las librerias necesarias
import time
import cv2
import numpy as np
import math as mt
import argparse
import os
import shutil


# Calculo del Laplaciano de una imagen
def variance_of_laplacian(image):
    var = cv2.Laplacian(image, cv2.CV_64F).var()
    return var

# Extractor de frames
def extractFrames(pathIn):
    # Se crea el directorio para almacenar los frames
    #os.mkdir(pathOut)
    keep = 0.6

    # Se registra el video solicitado
    cap = cv2.VideoCapture(pathIn)
    count = 0
    ret, frame = cap.read()
    frames = []

    # Obteniedo el ROI
    recX = round(frame.shape[1]*(1-keep)/2)
    recY = round(frame.shape[0]*(1-keep)/2)

    # Mientras existan frames disponibles en el video
    while (cap.isOpened()):
        # Se captura frame por frame
        ret, frame = cap.read()
        if ret == True:
            # Se lleva al espacio de escala de grises
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # Se evalua la varianza de su laplaciano
            fm = variance_of_laplacian(gray)
            # Si supera el valor Threshold se almacena en formato PNG
            if fm > 50:
                print('Read %d frame: ' % count, ret)
                #cv2.imwrite(os.path.join(pathOut, "frame{:d}.png".format(count)), frame)  #
save frame as JPEG file
                fr_roi = frame[recY:frame.shape[0]-recY,recX:frame.shape[1]-recX,:]
                gray = cv2.cvtColor(fr_roi, cv2.COLOR_BGR2GRAY)
                frames.append(gray)
```

```
                count += 1
        else:
            break

    # Se cierra el registro del video solicitado
    cap.release()
    return frames, count

#Funcion para determinar el porcentaje de coincidencia entre el mosaico parcial
#y un frame alineado
def frame_coincidence(img,img2):
  #Calculo del histograma del frame alineado
  hist = cv2.calcHist([img],[0],None,[16],[0,256])

  #Calculo del numero total de pixeles en el mosaico parcial
  we,he = img.shape[:2]
  N_pixels = we * he

  #Calculo del histograma del mosaico parcial
  hist2 = cv2.calcHist([img2],[0],None,[16],[0,256])

  #Mosaico parcial con el frame sustraido y su histograma
  img3 = cv2.subtract(img2, img)
  hist3 = cv2.calcHist([img3],[0],None,[16],[0,256])

  #Numero de pixeles desplazados hacia 0
  Pix_frame = N_pixels - hist[0]
  var = hist3[0] - hist2[0]

  #Relacion final
  rel = float((var/Pix_frame)*100)
  return rel


#Funcion para realizar el stitching de imagenes
def get_stitched_image(img1, img2, mask1, mask2, M):

  #Dimensiones de las imagenes a unir
  w1,h1 = img1.shape[:2]
  w2,h2 = img2.shape[:2]

  #Dimensiones parciales de las proyecciones
  img1_dims = np.float32([ [0,0], [0,w1], [h1, w1], [h1,0] ]).reshape(-1,1,2)
  img2_dims_temp = np.float32([ [0,0], [0,w2], [h2, w2], [h2,0] ]).reshape(-1,1,2)

  #Calculo de la perspectiva relativa de la segunda imagen
  img2_dims = cv2.perspectiveTransform(img2_dims_temp, M)

  #Dimensiones de la imagen final
  result_dims = np.concatenate( (img1_dims, img2_dims), axis = 0)

  #Dimensiones de la matriz transformacion
  [x_min, y_min] = np.int32(result_dims.min(axis=0).ravel() - 0.5)
  [x_max, y_max] = np.int32(result_dims.max(axis=0).ravel() + 0.5)
  transform_dist = [-x_min,-y_min]
  transform_array = np.array([[1, 0, transform_dist[0]],
                              [0, 1, transform_dist[1]],
                              [0,0,1]])

  #Tranformacion de perspectiva del frame
  result_img = cv2.warpPerspective(img2, transform_array.dot(M),
```

```
                                    (x_max-x_min, y_max-y_min),
                                    borderMode=cv2.BORDER_TRANSPARENT)
  #cv2.imwrite('Frame_alineado.png',result_img)

  #Transformacion de perspectiva de la mascara de coincidencia
  mask2 = cv2.warpPerspective(mask2, transform_array.dot(M),
                                    (x_max-x_min, y_max-y_min),
                                    borderMode=cv2.BORDER_TRANSPARENT)

  #Preparando el lienzo de la imagen final
  res_frame = result_img.copy()
  result_img[:result_img.shape[0],:result_img.shape[1]] = 0

  #Mascara del mosaico parcial para el calculo de coincidencia
  mask_canvas = result_img.copy()
  mask_canvas[transform_dist[1]:w1+transform_dist[1], transform_dist[0]:h1+transform_dist[0]]
= mask1
  #cv2.imwrite('Mask_mosaico.png',mask_canvas)

  #Se agrega el mosaico parcial al lienzo
  result_img[transform_dist[1]:w1+transform_dist[1], transform_dist[0]:h1+transform_dist[0]] =
img1
  #cv2.imwrite('Mosaico_parcial.png',result_img)

  #Calculo de la coincidencia empleando las mascaras
  #cv2.imwrite("m1.png",mask2)
  #cv2.imwrite("m2.png",mask_canvas)
  Rate = frame_coincidence(mask2,mask_canvas)
  print('The {:.2f}% of the frame is contained in the partial mosaic' .format(Rate))

  #Mascara resultante del proceso de stitching
  mask_buffer = []

  #Comprobando el grado de coincidencia y nitidez del frame alineado
  #Si la relacion es mayor a 75% pero no supera el threshold de nitidez
  #Entonces se debe omitir el frame en cuestion
  if ((Rate > 75) & (cv2.Laplacian(img2, cv2.CV_64F).var()<200)) :
    return img1,mask_buffer,0

  #Si la relacion es muy baja se debe buscar frames anteriores hasta
  #obtener una coincidencia mayor
  elif Rate < 17:
    return img1,mask_buffer,2

  #Mascara final, imagen resultante e indicador de proceso exitoso
  #cv2.imwrite('Mask_frame.png',mask2)
  mask_buffer = cv2.add(mask2, mask_canvas)
  res2 = result_img
  result_img_copy = cv2.warpPerspective(img2, transform_array.dot(M),
                                        (x_max-x_min, y_max-y_min),
                                        res2,borderMode=cv2.BORDER_TRANSPARENT)

  return result_img_copy, mask_buffer, 1


#Funcion para determinar el angulo de alineacion de emparejamientos
def get_alignment_angle(image1, keyQuery, image2, keyTrain, verified_matches):
  #Dimensiones de las imagenes emparejadas
  w1,h1 = image1.shape[:2]
  w2,h2 = image2.shape[:2]
```

```
  #Numero de emparejamientos verificados en primera instancia
  l = len(verified_matches)

  #Calculo de angulos de cada emparejamiento y su almacenamiento
  angles = []
  for i in range(0,l):
    pos1 = verified_matches[i].queryIdx
    pos2 = verified_matches[i].trainIdx
    x1 = keyQuery[pos1].pt[0]
    y1 = keyQuery[pos1].pt[1]
    x2 = keyTrain[pos2].pt[0]+w1
    y2 = keyTrain[pos2].pt[1]
    angle = abs(mt.atan((y2-y1)/(x2-x1)))*180/mt.pi
    angles.append(angle)

  #Vector de angulos calculados
  return angles


#Funcion para calcular la matriz de homografia SURF
def get_surf_homography(img1, img2):

    #Inicializando detector SURF
    surf = cv2.xfeatures2d.SURF_create()

    #Deteccion de puntos y filtrado de los mismos
    Filter_Size = 1

    #Detectores SURF
    u1 = surf.detect(img1)
    u2 = surf.detect(img2)

    #Ordenamiento ascendente
    p1 = sorted(u1, key = lambda x:x.size)
    p2 = sorted(u2, key = lambda x:x.size)

    #Revision y filtrado de los puntos
    k1 = []
    k2 = []
    for i in range(len(p1)):
      if (p1[i].size >= p1[0].size * Filter_Size):
        break
    k1 = p1[i:len(p1)-1]
    for i in range(len(p2)):
      if (p2[i].size >= p2[0].size * Filter_Size):
        break
    k2 = p2[i:len(p2)-1]

    #Calculo de descriptores
    k1,d1 = surf.compute(img1,k1)
    k2,d2 = surf.compute(img2,k2)

    #Emparejamiento con BruteForceMatcher
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(d1,d2, k=2)

    #Aplicando el criterio de Lowe para eliminar redundancias
    verify_ratio = 0.6
    verified_matches1 = []
    for m1,m2 in matches:
        # Add to array only if it's a good match
```

```
        if m1.distance < verify_ratio * m2.distance:
            verified_matches1.append(m1)

    #Seleccion de emparejaientos alineados
    matched_angles = get_alignment_angle(img1, k1, img2, k2, verified_matches1)
    median_angle = np.median(matched_angles)
    verified_matches = []
    for Q in range(0,len(verified_matches1)):
      if (matched_angles[Q]<=median_angle*1.12) & (matched_angles[Q]>=median_angle*0.88):
        verified_matches.append(verified_matches1[Q])

    #Minimo numero de emparejamientos
    min_matches = 8
    if len(verified_matches) > min_matches:

        #Extraccion de coordenadas emparejadas
        img1_pts = []
        img2_pts = []
        for match in verified_matches:
            img1_pts.append(k1[match.queryIdx].pt)
            img2_pts.append(k2[match.trainIdx].pt)
        img1_pts = np.float32(img1_pts).reshape(-1,1,2)
        img2_pts = np.float32(img2_pts).reshape(-1,1,2)

        #Calculo de matriz homografica aplicando RANSAC
        M = np.zeros((3,3))
        M, mask = cv2.findHomography(img1_pts, img2_pts, cv2.RANSAC,0)

        #Mostrando y almacenando puntos emparejados
        #m_image = np.array([])
        #img_res = cv2.drawMatches(img1,k1,img2,k2,verified_matches,m_image, flags=2)
        #cv2.imwrite('Matches.png',img_res)

        #Matriz calculada y operacion exitosa
        return M, 1

    else:
        #Indicador de pocos emparejamientos y operacion fallida
        M = np.zeros((3, 3))
        print ('Error: Not enough matches')
        return M, 2


#Funcion principal
def main(addr,frate):

  #Se inicia el cronometro de ejecucion
  start = time.time()

  #Direccion de los frames extraidos del video
    #Los frames se extraen en una carpeta temporal, luego se eliminan
  #sp1 = addr.split("/")
  #fileName = sp1[-1].split(".")[0]
  #direc = "FramesTMP/"+fileName+"TMP/"
  frames, N = extractFrames(addr)

  #Tiempo de extracción
  fin_ext = time.time()
  print("Frame extraction took: {:.2f} seconds".format(fin_ext-start))

  #Parametros de la iteracion
```

```
i = 0 #Numero de iteracion
tasa = frate #Tasa de seleccion de frames
K = frate #Compensacion de avance/retroceso de seleccion
fin_bucle = 0 #Indicador de fin de bucle

#Calculo del numero de iteraciones necesarias
N_secciones = 1
Rang_seccion = round(N/N_secciones)-1
Secuencia = round(Rang_seccion/tasa-1)
if (Secuencia<1):
  Secuencia = 1

#Inicializacion del mosaico parcial y mascaras
result_image = frames[0] #Mosaico parcial
mask_principal = np.array(result_image,copy=True)
mask_principal[:result_image.shape[0],:result_image.shape[1]]=255
mask_sec = np.array(mask_principal,copy=True)

#Inicio del bucle de formacion de imagen
while i <= Secuencia-1:

  #Inicializacion de las imagenes a unir
  img2 = result_image
  #img1 = imread_pros(direc + 'frame'+str(i*tasa+K)+'.png')
  img1 = frames[i*tasa+K]
  print ('Now reading: frame '+str(i*tasa+K))

  #Calculo de homografia SURF y retorno de operacion
  M, Check_val =  get_surf_homography(img1, img2)

  #Revision de operacion previa
  #Indicador de operacion fallida
  if Check_val == 2:
    Check_val = 2

  #Operacion exitosa
  else:
  #Realizar el calculo de coincidencia, stitching y estado de operacion
    result_buffer, mask_buffer, Check_val = get_stitched_image(img2, img1,mask_principal,
                                            mask_sec, M)


  #Revision de operacion previa
  #Si el frame tiene un traslape muy elevado y baja nitidez se omite
  if Check_val == 0:
    print('Skipping frame...')
    K = tasa

  #Si la coincidencia es muy baja se recupera frames anteriores
  elif Check_val == 2:
    print('Recovering previous frames...')
    i = i-1
    K = K-3
    if K == 0:
      K = tasa

  #Si la operacion es exitosa se almacena el resultado y su mascara asociada
  else:
    result_image = result_buffer.copy()
    mask_principal = mask_buffer.copy()
    K = tasa #Reinicio de compensacion
```

```
    #Avance en el bucle secuencial
    i +=1

    #Revision de fin de bucle y seleccion del frame final
    if ((i == Secuencia) & (fin_bucle == 0)):
      K = (N-1) - ((i-1)*tasa)
      i -=1
      fin_bucle = 1

  #Creacion del mosaico final y escritura en el disco
  result_image_name = 'Mosaico_Final.png'
  cv2.imwrite(result_image_name, result_image)
  result_image_name = 'Resultados/'+addr.split("/")[-1].split(".")[0]+"_"+str(frate)+'.png'
  cv2.imwrite(result_image_name, result_image)

  #Se eliminan los archivos temporales
  #shutil.rmtree(direc,ignore_errors=True)

  #Se detiene el cronometro de ejecucion
  end = time.time()
  print("Stitching process took: {:.2f} seconds".format(end-fin_ext))
  print("The entire process took: {:.2f} seconds".format(end-start))

#Llamado a la funcion principal
if __name__=='__main__':
    # Construcción de la secuencia de argumentos que ingresan a la función principal
    # Se solicita la dirección de la imagen del logotipo y la cadena a grabar en la imagen
    # En caso que no se ingrese una cadena, se almacenará una por defecto
    ap = argparse.ArgumentParser()
    ap.add_argument("-a", "--address", default="Videos/Vid2210188.mp4",
                    help="path to video")
    ap.add_argument("-f", "--framerate", default=15,
                    help="number of frames between each one of the selected to create the
final result")

    args = vars(ap.parse_args())

    # Ejecución de la función principal
    main(args["address"], args["framerate"])
```

Anexo B. Algoritmo en Android

Las plantillas originales de los archivos aquí presentados pueden ser descargadas de la página de GitHub perteneciente al usuario Jonathan Reynolds o a través del siguiente enlace: https://github.com/J0Nreynolds/AndroidOpenCVCamera.

En dicho proyecto se presenta una aplicación sencilla en Android que emplea la API Camera2 para capturar imágenes y procesarlas con OpenCV. Se puede emplear tanto la cámara trasera como la delantera para obtener las imágenes; el resultado final mostrado es el Laplaciano de cada *frame* capturado.

B.1. AndroidManifest.xml

```
<!-- AndroidManifest.xml -->
<!-- Autor: Jose Ramirez Diaz -->
<!-- Este archivo permite declarar los permisos necesarios para ejecutar la aplicacion -->
<!-- y configuracion basica de la misma: icono, etiqueta, nombre de la actividad... -->


<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.tesis.androidopencvcamera">

    <!-- Min/target SDK versions (<uses-sdk>) managed by build.gradle -->
    <uses-feature android:glEsVersion="0x00020000" android:required="true"/>
    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera2" android:required="false"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:theme="@style/MaterialTheme">
        <activity android:name="net.tesis.androidopencvcamera.CameraActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

B.2. Activity.xml

```
<!-- activity.xml -->
<!-- Autor: Jose Ramirez Diaz -->
<!-- Este archivo señala el layout de la interfaz empleada por la aplicación -->
<!-- Indica posiciones, tamaños, etc. de los CheckBoxes, Deslizador, Switch, etc. -->
```

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.constraint.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:rotation="0">

        <android.support.constraint.ConstraintLayout
            android:id="@+id/constraintLayout"
            android:layout_width="55dp"
            android:layout_height="match_parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintVertical_bias="0.0">

            <CheckBox
                android:id="@+id/AddFrames1"
                android:layout_width="108dp"
                android:layout_height="20dp"
                android:text="Add frames"
                android:layout_marginBottom="130dp"
                android:layout_marginEnd="8dp"
                android:layout_marginStart="30dp"
                android:layout_marginTop="8dp"
                android:rotation="270"
                android:textSize="10dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="1.0"/>

            <CheckBox
                android:id="@+id/KeepFrames1"
                android:layout_width="108dp"
                android:layout_height="20dp"
                android:text="Keep frames"
                android:layout_marginBottom="130dp"
                android:layout_marginEnd="40dp"
                android:layout_marginStart="8dp"
                android:layout_marginTop="8dp"
                android:rotation="270"
                android:textSize="10dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="1.0"/>

            <Switch
                android:id="@+id/camera_switch"
                android:layout_width="108dp"
                android:layout_height="47dp"
                android:layout_marginBottom="8dp"
                android:layout_marginEnd="8dp"
                android:layout_marginStart="8dp"
                android:layout_marginTop="8dp"
                android:rotation="270"
                android:text="Scan Sheet"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.51"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.07999998" />

    <TextView
        android:id="@+id/fps_text_view"
        android:layout_width="64dp"
        android:layout_height="46dp"
        android:layout_marginBottom="18dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:rotation="270"
        android:text="FPS:"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <TextView
        android:id="@+id/fps_text_view2"
        android:layout_width="88dp"
        android:layout_height="46dp"
        android:layout_marginBottom="30dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="44dp"
        android:layout_marginTop="8dp"
        android:rotation="270"
        android:text="Frames:"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <SeekBar
        android:id="@+id/seekBar1"
        android:layout_width="100dp"
        android:layout_height="46dp"
        android:layout_marginBottom="200dp"
        android:layout_marginEnd="24dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:max="30"
        android:min="1"
        android:indeterminate="false"
        android:progress="5"
        android:rotation="270"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0"/>

    <TextView
        android:id="@+id/text_viewSB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="210dp"
        android:layout_marginEnd="30dp"
        android:layout_marginStart="44dp"
        android:layout_marginTop="8dp"
        android:rotation="270"
        android:textSize="10dp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="1.0" />

        </android.support.constraint.ConstraintLayout>

        <net.tesis.androidopencvcamera.MyGLSurfaceView
            android:id="@+id/my_gl_surface_view"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toStartOf="@+id/constraintLayout"
            app:layout_constraintHorizontal_bias="0.0"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </android.support.constraint.ConstraintLayout>

</FrameLayout>
```

B.3. CameraActivity.java

```
//CameraActivity.java
//Autor: Jose Ramirez Diaz
//Esta clase de java permite declarar la actividad principal de la aplicacion
//Además, incializa la configuracion de los elementos de interfaz grafica
//Finalmente, solicita los permisos pertienetes para el uso de la camara

//Paquete al cual pertenece esta clase
//package net.tesis.androidopencvcamera;

import android.Manifest;
import android.app.Activity;
import android.content.Context;
import android.content.pm.ActivityInfo;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.util.Log;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Toast;
import android.widget.TextView;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.SeekBar;
import android.widget.Switch;

import org.opencv.android.CameraBridgeViewBase;

import java.io.File;

//Clase para declarar la actividad principal
public class CameraActivity extends Activity {

    //Sentencia usada para cargar las librerias "native-lib" y OpenCV
    //al iniciar la aplicacion
    static {
        System.loadLibrary("native-lib");
    }
```

```
    //Case que permite el renderizado de imagenes con OpenGL
    private MyGLSurfaceView mView;
    //Switch empleado para iniciar la grabacion
    private Switch mSwitch;
    //Codigo de permiso para la camara
    private static final int MY_PERMISSIONS_REQUEST_CAMERA = 1337;
    //Barra deslizante
    private SeekBar sBar;
    //Visores de conteo de frames
    private TextView tView;
    //CheckBoces para opciones de agregar y mantener frames
    CheckBox AddFrames;
    CheckBox KeepFrames;

    //Se indica que el siguiente metodo se sobrepone al declarado en Activity
    @Override
    //Metodo que realiza la configuracion inicial al abrir la aplicacion
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Se revisan los permisos otrogados a la camara
        final Context context = getApplicationContext();
        if(checkSelfPermission(context, Manifest.permission.CAMERA) !=
PackageManager.PERMISSION_GRANTED) {
            //Si no se ha conseguido el permiso para la camara se muestra un mensaje
apropiado
            if (this.shouldShowRequestPermissionRationale(Manifest.permission.CAMERA))
{
                Toast.makeText(context, "Need access to your camera to proceed",
Toast.LENGTH_LONG).show();
                this.finish();
            } else {
                //Se hace efectiva la solicitud de permiso para la camara
                this.requestPermissions(new String[]{Manifest.permission.CAMERA},
MY_PERMISSIONS_REQUEST_CAMERA);
            }
        }
        //Si ya es efectivo el permiso se procede a configurar la aplicacion
        else {
            setupApplication();
        }

        //Se asigna la barra deslizante y su texto para indicar la tasa de seleccion
        sBar = (SeekBar) findViewById(R.id.seekBar1);
        tView = (TextView) findViewById(R.id.text_viewSB);
        tView.setText(sBar.getProgress() + "/" + sBar.getMax());

        //Se incializa el listener de la barra deslizante, encargado de verificar
        //sus modificaciones y reponder de acuerdo a ello
        sBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
            int pval = 0;
            @Override
            public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromUser) {
                //Si se modifica la posicion de la barra se almacena el nuevo valor y
se
                //muestra el texto
                pval = progress;
                tView.setText(pval + "/" + seekBar.getMax());
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
```

```
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
                //Cuando se termino de modificar la posicion de la barra se muestra el
valor
                //y se modifica la tasa de seleccion
                tView.setText(pval + "/" + seekBar.getMax());
                mView.setFrate(pval);
            }
        });

    }

    //Se configura la interfaz de la aplicacion
    private void setupApplication(){

        //Se establece el modo de visualizacion y el layout de la actividad
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
                WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        setContentView(R.layout.activity);

        //Se asignan los CheckBoxes
        AddFrames = (CheckBox) findViewById(R.id.AddFrames1);
        KeepFrames = (CheckBox) findViewById(R.id.KeepFrames1);

        //Se asigna y configura el switch para iniciar y terminar la grabacion
        mSwitch = (Switch) findViewById(R.id.camera_switch);
        mSwitch.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked)
{
                if(isChecked) {
                    //Si se activo el Switch, la bandera de iniciar grabacion se
coloca en True
                    mView.setScan(true);
                    //Se revisa las opciones de añadir frames y mantenerlos en memoria
                    if(AddFrames.isChecked()){
                        mView.setAddfr(true);
                    }
                    else{
                        mView.setAddfr(false);
                    }
                    if(KeepFrames.isChecked()){
                        mView.setKeepfr(true);
                    }
                    else{
                        mView.setKeepfr(false);
                    }
                }
                //Si no se acciono el Switch, no se modifica la bandera
                else {
                    mView.setScan(false);
                }
            }
        });

        //Se asigna la superficie de renderizado de imagen y la resolucion
```

```
        //maxima de visualizacion
        mView = (MyGLSurfaceView) findViewById(R.id.my_gl_surface_view);
        mView.setMaxCameraPreviewSize(1080, 1920);
        mView.setCameraTextureListener(mView);
    }

    //Metodo para revision de permiso otorgado a la camara
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String permissions[], int[] grantResults) {
        switch (requestCode) {
            case MY_PERMISSIONS_REQUEST_CAMERA: {
                //Si la solicitud se cancelo el arreglo estara vacio
                if (grantResults.length > 0
                        && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    //El permiso fue otorgado satisfactoriamente
                }
            }
        }
    }

    //Metodo llamado al reanudar el uso de la aplicacion
    @Override
    protected void onResume() {
        mView.onResume();
        super.onResume();
    }

    //Metodo llamado al poner la aplicacion en espera
    @Override
    public void onPause() {
        mView.onPause();
        super.onPause();
    }

    //Metodo para revisar permisos
    private static int checkSelfPermission(Context context, String permission) {
        if (permission == null) {
            throw new IllegalArgumentException("permission is null");
        }
        return context.checkPermission(permission, android.os.Process.myPid(),
android.os.Process.myUid());
    }
}
```

B.4. MyGLSurfaceView.java

```
//MyGLSurfaceView.java
//Autor: Jose Ramirez Diaz
//Esta clase de java implementa el listener (o receptor) de las imagenes capturadas
//por la camara, facilita la toma de acciones a realizar cuando la camara ha capturado
//una nueva imagen

//Paquete al cual pertenece la clase
//package net.tesis.androidopencvcamera;

import org.opencv.android.CameraGLSurfaceView;

import android.app.Activity;
import android.content.Context;
```

```
import android.content.pm.PackageManager;
import android.os.Handler;
import android.os.Looper;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.SurfaceHolder;
import android.widget.TextView;
import android.widget.Toast;

import java.io.File;

//Esta clase implementa el listener de texturas
public class MyGLSurfaceView extends CameraGLSurfaceView implements
CameraGLSurfaceView.CameraTextureListener {

    //Nombre para facilitar el registro
    static final String LOGTAG = "MyGLSurfaceView";
    //Contador de frames para el calculo de fps
    protected int  frameCounter;
    //Registro de lapso de tiempo en nanosegundos
    protected long lastNanoTime;
    //Variable que indica el uso de la camara delantera o trasera (NO USADA)
    protected boolean frontFacing = false;
    //Variable que indica el estado actual de grabacion
    protected boolean Scan = false;
    //Variable que determina la tasa de seleccion
    protected int Frate = 5;
    //Visores de texto para la interfaz
    TextView mFpsText = null;
    TextView mFramesText = null;
    //Variable que indica cuantos frames se almacenaron en memoria
    protected int Nframes;
    //Variables para indicar el estado de los CheckBoxes
    protected boolean Keepfr = false;
    protected boolean Addfr = false;

    //Constructor por defecto de esta clase
    public MyGLSurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    //Metodo para reaccionar al desplazamiento del dedo sobre la pantalla
    //(NO SE USA)
    @Override
    public boolean onTouchEvent(MotionEvent e) {
        if(e.getAction() == MotionEvent.ACTION_DOWN)
            ((Activity)getContext()).openOptionsMenu();
        return true;
    }

    //Metodo llamado al crear la superficie de visualizacion (inicio app)
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        super.surfaceCreated(holder);
    }

    //Metodo llamado al destruir la superficie de visualizacion (cierro app)
    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        super.surfaceDestroyed(holder);
    }
```

```
    //Metodo llamado cuando se inicia la visualizacion de imagenes a traves de la
camara
    @Override
    public void onCameraViewStarted(int width, int height) {
        //Se muestra un mensaje corto indicando el inicio de visualizacion
        ((Activity) getContext()).runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getContext(), "onCameraViewStarted",
Toast.LENGTH_SHORT).show();
            }
        });
        //Se inicializa el contador de frames para el calculo de fps
        frameCounter = 0;
        //Se inicializa el conteo temporal
        lastNanoTime = System.nanoTime();
    }

    //Metodo llamado cuando se detiene la visualiazcion de imagenes a traves de la
camara
    @Override
    public void onCameraViewStopped() {
        //Se muestra un mensaje corto indicando la detencion de visualizacion
        ((Activity) getContext()).runOnUiThread(new Runnable() {
            public void run() {
                Toast.makeText(getContext(), "onCameraViewStopped",
Toast.LENGTH_SHORT).show();
            }
        });
    }

    //Metodo llamado para modificar el uso de camara frontal o trasera
    public void setFrontFacing(boolean frontFacing) {
        this.frontFacing = frontFacing;
    }

    //Metodo llamado para modificar el estado de grabacion
    public void setScan(boolean Scanning){
        this.Scan = Scanning;
    }

    //Metodo llamado para modificar la bandera para mantener frames en memoria
    public void setKeepfr(boolean keepfra){
        this.Keepfr = keepfra;
    }

    //Metodo llamado para modificar la bandera para añadir frames al vector nitidos
    public void setAddfr(boolean addfra){
        this.Addfr = addfra;
    }

    //Metodo llamado para modificar la tasa de seleccion
    public void setFrate(int FRate){this.Frate = FRate;}

    //Metodo llamado cuando esta lista una nueva imagen de la camara
    @Override
    public boolean onCameraTexture(int texIn, int texOut, int width, int height) {
        //Se incrementa el contador de frames y se calcula los fps
        frameCounter++;
        if(frameCounter >= 30)
        {
            final int fps = (int) (frameCounter * 1e9 / (System.nanoTime() -
lastNanoTime));
            Log.i(LOGTAG, "drawFrame() FPS: "+fps);
```

```
            //Se actualiza el texto en el visor de fps
            if(mFpsText != null) {
                Runnable fpsUpdater = new Runnable() {
                    public void run() {
                        mFpsText.setText("FPS: " + fps);
                    }
                };
                new Handler(Looper.getMainLooper()).post(fpsUpdater);
            }
            //Se emplea esta sentencia para inicializar el visor de fps
            else {
                Log.d(LOGTAG, "mFpsText == null");
                mFpsText = (TextView)((Activity)
getContext()).findViewById(R.id.fps_text_view);
            }
            frameCounter = 0;
            lastNanoTime = System.nanoTime();
        }

        //Se da inicio a la funcion contenida en native-lib para procesar las imagenes
        Nframes = processFrame(texIn, texOut, width, height, frontFacing, Scan, Frate,
Keepfr, Addfr);

        //Se actualiza el texto del visor de frames almacenados en memoria
        if(mFramesText != null) {
            Runnable framesUpdater = new Runnable() {
                public void run() {
                    mFramesText.setText("Frames: " + Nframes);
                }
            };
            new Handler(Looper.getMainLooper()).post(framesUpdater);
        } else {
            mFramesText = (TextView)((Activity)
getContext()).findViewById(R.id.fps_text_view2);
        }

        return true;
    }

    private static native int processFrame(int tex1, int tex2, int w, int h, boolean
frontFacing, boolean Scan, int Frate, boolean Keepframes, boolean Addframes);


}
```

B.5. Camera2Renderer.java

```
//Camera2Renderer.java
//OpenCV-contrib v.3.4.0
//Esta clase de java implementa el renderizado de imagenes capturadas por la camara
//haciendo uso de la API Camera2 para modificar los diversos parametros de la misma
//En el metodo createCameraPreviewSession se modifican los parametros de exposicion
necesarios
//Debido a la extension del codigo en esta clase solo se presentara la seccion
relevante al
//control manual de la exposicion


    private void createCameraPreviewSession() {
        int w=mPreviewSize.getWidth(), h=mPreviewSize.getHeight();
        Log.i(LOGTAG, "createCameraPreviewSession("+w+"x"+h+")");
```

```
        if(w<0 || h<0)
            return;
        try {
            mCameraOpenCloseLock.acquire();
            if (null == mCameraDevice) {
                mCameraOpenCloseLock.release();
                Log.e(LOGTAG, "createCameraPreviewSession: camera isn't opened");
                return;
            }
            if (null != mCaptureSession) {
                mCameraOpenCloseLock.release();
                Log.e(LOGTAG, "createCameraPreviewSession: mCaptureSession is already
started");
                return;
            }
            if(null == mSTexture) {
                mCameraOpenCloseLock.release();
                Log.e(LOGTAG, "createCameraPreviewSession: preview SurfaceTexture is
null");
                return;
            }
            mSTexture.setDefaultBufferSize(w, h);

            Surface surface = new Surface(mSTexture);

            mPreviewRequestBuilder = mCameraDevice
                    .createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
            mPreviewRequestBuilder.addTarget(surface);

            mCameraDevice.createCaptureSession(Arrays.asList(surface),
                    new CameraCaptureSession.StateCallback() {
                        @Override
                        public void onConfigured( CameraCaptureSession
cameraCaptureSession) {
                            mCaptureSession = cameraCaptureSession;
                            try {
                                //El foco de la camara se establece en modo automatico
y continuo

mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);
                                //Se desactiva el control automatico de la exposicion

mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_OFF);
                                //Se establece la velocidad de obturacion en el valor
deseado

mPreviewRequestBuilder.set(CaptureRequest.SENSOR_EXPOSURE_TIME, Expo_Time_1);
                                //Se establece la ganancia ISO en un valor deseado

mPreviewRequestBuilder.set(CaptureRequest.SENSOR_SENSITIVITY, 100);
                                //Se enciende el flash de la camara durante la captura
de imagenes
                                mPreviewRequestBuilder.set(CaptureRequest.FLASH_MODE,
CaptureRequest.FLASH_MODE_TORCH);



mCaptureSession.setRepeatingRequest(mPreviewRequestBuilder.build(), null,
mBackgroundHandler);
                                Log.i(LOGTAG, "CameraPreviewSession has been
started");
```

```
                            } catch (CameraAccessException e) {
                                Log.e(LOGTAG, "createCaptureSession failed");
                            }
                            mCameraOpenCloseLock.release();
                        }

                        @Override
                        public void onConfigureFailed(
                                CameraCaptureSession cameraCaptureSession) {
                            Log.e(LOGTAG, "createCameraPreviewSession failed");
                            mCameraOpenCloseLock.release();
                        }
                    }, mBackgroundHandler);
        } catch (CameraAccessException e) {
            Log.e(LOGTAG, "createCameraPreviewSession");
        } catch (InterruptedException e) {
            throw new RuntimeException(
                    "Interrupted while createCameraPreviewSession", e);
        }
        finally {
            //mCameraOpenCloseLock.release();
        }
    }
```

B.6. Native-lib.cpp

```
//Native-lib.cpp
//Autor: Jose Ramirez Diaz
//Esta libreria en codigo nativo C++ contiene las funciones necesarias para
//procesar las imagenes adquiridas por la camara del telefono y generar el
//mosaico a partir de ellas.


//Inclusion de librerias necesarias
#include <jni.h>

#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/xfeatures2d/nonfree.hpp>
#include <opencv2/calib3d.hpp>
#include <opencv2/highgui.hpp>

#include <GLES2/gl2.h>

#include "common.hpp"

#include "cmath"

using namespace cv;

//Declaracion de funciones
cv::UMat stitchFrames(const std::vector<UMat>* frames, int FrateDEF);
cv::Mat get_surf_homography(UMat img1, UMat img2, int& check);
std::vector< float > get_alignment_angle(UMat img1,std::vector<KeyPoint> keyQuery,
UMat img2,
                            std::vector<KeyPoint> keyTrain, std::vector<DMatch>
verified_matches);
float medianAngle(std::vector<float> Angles);
int get_stitched_image(UMat img1, UMat img2, UMat mask1, UMat mask2, Mat M, UMat&
Result,
```

```
        UMat& Mask_result);
void get_limits(Mat img1_dims, Mat img2_dims, int& xmin, int& ymin,
        int& xmax, int& ymax);
float frame_coincidence(UMat img, UMat img2);


//Funcion principal invocada desde Java al recibir una imagen nueva
extern "C" JNIEXPORT jint

JNICALL
Java_net_tesis_androidopencvcamera_MyGLSurfaceView_processFrame(JNIEnv *env, jobject
/* this */,
                                                                jint texIn, jint
texOut,
                                                                jint w, jint h,
                                                                jboolean
frontFacing,
                                                                jboolean Scan,
                                                                jint FrateDef,
                                                                jboolean
Keepframes,
                                                                jboolean
Addframes) {
    //Declaracion de matrices para operaciones
   static UMat m, Result, Result_resized;
   //Vector que almacena imagenes nitidas
    static std::vector<UMat> frames;
    //Variable que almacena el estado de grabacion previo
    static bool prevState;
    //Variables que indican el estado de grabacion y
    //termino del proceso respectivamente
    static bool actProcess, showResult;
    //Matriz para la presentacion del resultado
    UMat ResFin;
   //Variable de conteo para mostrar el resultado
   //final en pantalla
    static int Ncicles;

   //Creacion de la matriz para operaciones
    m.create(h, w, CV_8UC4);

   //Se obtienen los pixeles almacenados en el buffer de cuadros
   //y se almacenan en la matriz m
    glReadPixels(0, 0, m.cols, m.rows, GL_RGBA, GL_UNSIGNED_BYTE,
m.getMat(ACCESS_WRITE).data);

    //Determinando la region de interes y excluyendo area
    //fuera de foco
    int p1x = w/2 - 75*w/200;
    int p1y = h/2 - 75*h/200;
    int width = 75*w/100;
    int heigth = 75*h/100;
    Rect rect(p1x,p1y,width,heigth);
    UMat fr_pr, fr_NOpr;
    m(rect).copyTo(fr_pr);
    m(rect).copyTo(fr_NOpr);

    //Calculo de la matriz Laplaciano de la region de interes
    cvtColor(fr_pr, fr_pr, CV_BGRA2GRAY);
    Laplacian(fr_pr, fr_pr, CV_8U);
    multiply(fr_pr, 5, fr_pr);

    //Se evalua si se esta grabando para analizar el Laplaciano
```

```
 if (Scan){
    //Variables para calculo de la varianza del Laplaciano
     Scalar mean, dev;
     UMat m_flatten = fr_pr.reshape(1,1);
     meanStdDev(m_flatten,mean,dev);
     //Varianza del Laplaciano
     float var = pow(dev[0],2);
     //Si la varianza supera el umbral y se selecciono la
     //opcion de agragar imagenes al vector de nitidos
     if ((var > 1500)&&(Addframes)){
         UMat fr_test;
         frames.insert(frames.end(),fr_NOpr.clone());
     }
 }

//Se compara el estado previo de grabacion y el estado actual
//para determinar si se dara inicio a la composicion de mosaico
 if ((prevState == true) && (Scan == false)){
     actProcess = true;
 }
 prevState = Scan;

//Copiando la matriz Laplaciano de nuevamente a la imagen original
//para observarla en pantalla
 cvtColor(fr_pr, fr_pr, CV_GRAY2BGRA);
 fr_pr.copyTo(m(rect));

 //Proceso post-grabacion para componer el mosaico
 if (actProcess){
   //Si se tiene imagenes suficientes almacenadas en el vector, entonces
   //se da inicio al proceso de composicion de mosaico
     if ((frames.size() > 0)&&(float (frames.size())/float (FrateDef) >= 1)) {

        //Se almacena el resultado del algoritmo en la matriz
         Result = stitchFrames(&frames,FrateDef);

      //Se eliminan todos los datos del vector si no se eligio
      //la opcion de conservarlos luego del mosaico
         if (!Keepframes){
             frames.erase(frames.begin(),frames.end());
         }

         //Se indica que el proceso termino y se muestran resultados en
         //pantalla
         actProcess = false;
         showResult = true;
         Ncicles = 200;

      //Se comparan las dimensiones de la imagen resultantes para
      //orientarla y mostrarla adecuadamente en pantalla
         if(Result.cols < Result.rows){
             transpose(Result,ResFin);
             flip(ResFin,ResFin,1);
         }
         else{
             ResFin = Result.clone();
         }

         //Se modifica el tamaño del resultado para abarcar toda la
         //pantalla
         resize(ResFin, Result_resized,Size(w,h));
     }
     else{
```

```
            actProcess = false;
        }
    }

   //Si el proceso termino, se muestra el resultado en pantalla
   //por un tiempo limitado
    if((showResult)&&(Ncicles>0)){
        Result_resized.copyTo(m);
        Ncicles = Ncicles -1;
        //Codigo reservado para almacenar el resultado como PNG en la memoria
        //UMat UXMLres = Result.clone();
        //flip(UXMLres,UXMLres,1);
        //Mat XMLres = UXMLres.getMat(ACCESS_READ);

//imwrite("/storage/emulated/0/Android/data/net.tesis.androidopencvcamera/files/Result
.png",XMLres);
    }
    else{
        showResult = false;
    }

    //La matriz "m" procesada se muestra nuevamente en pantalla
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texOut);
    t = getTimeMs();
    glTexSubImage2D(GL_TEXTURE_2D, 0, 0, 0, m.cols, m.rows, GL_RGBA, GL_UNSIGNED_BYTE,
m.getMat(ACCESS_READ).data);

   //Se retorna el numero de imagenes almacenadas en el vector de nitidos
    int NFrames = frames.size();
    return NFrames;
}

//Funcion para la composicion de mosaicos
cv::UMat stitchFrames(const std::vector<UMat>* frames, int FrateDEF){

   //Se accede al vector de imagenes nitidas
    std::vector<UMat> a;
    a = *frames;
    //tasa: indica la tasa de sleccion, i: conteo de avance, Kc: compensacion de
avance,
    //fin:bucle: indicador del final de iteraciones, Nframes: total de imagenes,
    //Secuencia: conteo de iteraciones a realizar
    int tasa, i, Kc, fin_bucle, Nframes, Secuencia;
    Nframes = a.size();
    tasa = FrateDEF;
    Kc = tasa;
    i = 0;
    fin_bucle = 0;
    Secuencia = round((Nframes-1)/tasa-1);
    if (Secuencia < 1){
        Secuencia = 1;
    }

   //Se incializa la matriz de resultado con la primera imagen del vector
    UMat result_image = a[0];
    cvtColor(result_image,result_image,CV_BGRA2BGR);

    //Se crean las mascaras para calculo de coincidencia
    UMat mask_prin, mask_sec;
    mask_prin.create(a[0].rows,a[0].cols,CV_8UC1);
    mask_prin = 255;
    mask_sec = mask_prin.clone();
```

```
//Matrices temporales para la composicion de mosaico
 UMat img1, img2;
 //Matriz homografica
 Mat Mhomo;
 //Valor indicador del estado de stitching
 int Check_val = 0;
 //Matrices temporales para almacenar resultados
 UMat result_buffer, mask_buffer;

//Iteraciones entre imagenes
 while(i <= Secuencia-1){

     img2 = result_image.clone();
     img1 = a[i*tasa + Kc].clone();
     cvtColor(img1,img1,CV_BGRA2BGR);

   //Calculo de la matriz homografica entre un frame capturado
   //y un mosaico parcial
     Mhomo = get_surf_homography(img1, img2, Check_val);

   //Solo si el resultado del stitching es distinto del codigo 2
   //se calcula un resultado parcial
     if (Check_val != 2){
         Check_val = get_stitched_image(img2,img1,mask_prin,
                 mask_sec,Mhomo,result_buffer,mask_buffer);
     }

   //Si el codigo de salida es 0 se pasa al siguiente frame
   //pues la coincidencia es mayor al umbral
     if(Check_val == 0){
         LOGD("Skipping frame...");
         Kc = tasa;
     }

     //Si el codigo de salida es 2 se retrocede en el contador
     //para emplear frames anteriores y mejorar la coincidencia
     else if(Check_val == 2){
         LOGD("Recovering previous frames...");
         i = i -1;
         Kc = Kc - 3;
         if(Kc == 0){
             Kc = tasa;
         }
     }

     //Si el codigo de salida es 1 se obtuvo un resultado con
     //una coincidencia dentro del umbral y se almacena
     else{
         result_image = result_buffer.clone();
         mask_prin = mask_buffer.clone();
         Kc = tasa;
     }

   //Avance en las iteraciones y calculo de fin de bucle
     i++;
     if((i == Secuencia)&&(fin_bucle == 0)){
         Kc = (Nframes-1)-((i-1)*tasa);
         i = i - 1;
         fin_bucle = 1;
     }
 }
```

```
    //Retorno del resultado final
     return result_image;
}

//Función para obtener la matriz homografica usando caracteristicas SURF
cv::Mat get_surf_homography(UMat img1, UMat img2, int& check){

    //Valor umbral para el detector Hessiano
     int minHessian = 400;

     //Creacion del detector SURF
     Ptr<xfeatures2d::SURF> detector = xfeatures2d::SURF::create( minHessian );

    //Coeficiente para filtrado de keypoints por tamaño
     float Filter_Size = 1;

    //Vectores para almacenar keypoints
     std::vector<KeyPoint> u1, u2, k1, k2;

    //Calculo de puntos caracteristicos en el frame y en el mosaico parcial
     detector->detect( img1, u1 );
     detector->detect( img2, u2 );

     //Seccion reservada para el filtrado de KP por tamaño
     k1 = u1;
     k2 = u2;

    //Calculo de descriptores
     Mat d1, d2;
     detector->compute(img1, k1, d1);
     detector->compute(img2, k2, d2);

    //Objeto matcher para comparar decriptores
     Ptr<DescriptorMatcher> matcher =
DescriptorMatcher::create(DescriptorMatcher::FLANNBASED);
     std::vector< std::vector<DMatch> > knn_matches;
     matcher->knnMatch( d1, d2, knn_matches, 2 );

    //Prueba del ratio de Lowe
     const float verify_ratio = 0.68;
     std::vector<DMatch> verified_matches1;
     for (size_t i = 0; i < knn_matches.size(); i++)
     {
         if (knn_matches[i][0].distance < verify_ratio * knn_matches[i][1].distance)
         {
             verified_matches1.push_back(knn_matches[i][0]);
         }
     }

    //Comparacion de angulos de inclinacion en los emparejamientos
    //para eliminar errores
     std::vector< float > matched_angles;
     matched_angles = get_alignment_angle(img1, k1, img2, k2, verified_matches1);
     float median_angle = medianAngle(matched_angles);
     std::vector<DMatch> verified_matches;
     for (size_t q = 0; q < verified_matches1.size(); q++)
     {
         if ((matched_angles[q]<=median_angle*1.5)&&
         (matched_angles[q]>=median_angle*0.5))
         {
             verified_matches.push_back(verified_matches1[q]);
         }
     }
```

```
   //Verificando emparejamientos minimos y calculando matriz homografica
    const unsigned long min_matches = 8;
    if (verified_matches.size() > min_matches){
        std::vector<Point2f> img1_pts, img2_pts;
        for (size_t p = 0; p < verified_matches.size(); p++){
            img1_pts.push_back(k1[verified_matches[p].queryIdx].pt);
            img2_pts.push_back(k2[verified_matches[p].trainIdx].pt);
        }
        Mat img1_ptsMAT = Mat(img1_pts);
        Mat img2_ptsMAT = Mat(img2_pts);
        Mat M;

      //M: matriz homografica
        M = findHomography(img1_ptsMAT,img2_ptsMAT,RANSAC,0);

      //El codigo 1 indica una operacion correcta
        check = 1;

      //Se retorna la matriz M
      return M;
    }
    else{
       //En caso de no encontrar emparejamientos suficientes
       //El codigo de salida es 2 y la matriz M esta vacia
        Mat M;
        M.create(3,3,CV_64F);
        M = 0;
        check = 2;
        LOGD("Error: Not enough matches.");
        return M;
    }
}

//Funcion para calcular el angulo de alineacion de los emparejamientos
std::vector< float > get_alignment_angle(UMat img1,std::vector<KeyPoint> keyQuery,
        UMat img2, std::vector<KeyPoint> keyTrain, std::vector<DMatch>
verified_matches){

    std::vector<float> angles;

   //Dimensiones del frame y del mosaico parcial
    int w1, h1, w2, h2;
    h1 = img1.cols;
    w1 = img1.rows;
    h2 = img2.cols;
    w2 = img2.rows;

   //Calculo y almacenamiento de angulos calculados
    for(size_t i = 0; i < verified_matches.size(); i++){
        int pos1 = verified_matches[i].queryIdx;
        int pos2 = verified_matches[i].trainIdx;
        float x1 = keyQuery[pos1].pt.x;
        float y1 = keyQuery[pos1].pt.y;
        float x2 = keyTrain[pos2].pt.x + w1;
        float y2 = keyTrain[pos2].pt.y;
        float angle = abs(atan((y2-y1)/(x2-x1)))*180/M_PI;
        angles.push_back(angle);
    }

    return angles;
}
```

```
//Funcion para estimar el valor mediana de los angulos de
//emparejamiento
float medianAngle(std::vector<float> Angles){

    std::vector<float> sortedAngles = Angles;
    std::sort(sortedAngles.begin(),sortedAngles.end());
    unsigned long SizeAngles = Angles.size();

    return sortedAngles[round(SizeAngles/2)];

}

//Funcion para realizar el proceso de stitching
int get_stitched_image(UMat img1, UMat img2, UMat mask1, UMat mask2, Mat M, UMat&
Result,
                       UMat& Mask_result){

    int Check_val;
    float w1, h1, w2, h2;
    w1 = img1.rows;
    h1 = img1.cols;
    w2 = img2.rows;
    h2 = img2.cols;

   //Dimensiones parciales de las proyecciones
    std::vector<Point2f>
img1_dims{Point2f(0.f,0.f),Point2f(0.f,w1),Point2f(h1,w1),Point2f(h1,0.f)};
    std::vector<Point2f>
img2_dims_temp{Point2f(0.f,0.f),Point2f(0.f,w2),Point2f(h2,w2),Point2f(h2,0.f)};
    Mat img1_dimsMAT = Mat(img1_dims);
    Mat img2_dims_tempMAT = Mat(img2_dims_temp);

   //Calculo de la perspectiva relativa de la segunda imagen
    Mat img2_dimsMAT;
    perspectiveTransform(img2_dims_tempMAT,img2_dimsMAT,M);

   //Calculo de la matriz de tranformacion de perspectiva
    int x_min, y_min, x_max, y_max;
    get_limits(img1_dimsMAT,img2_dimsMAT,x_min,y_min,x_max,y_max);
    double transformTEMP[3][3] = {{1,0,(double)(-x_min)},{0,1,(double)(-
y_min)},{0,0,1}};
    Mat transform_array = Mat(3,3,CV_64F,&transformTEMP);
    Mat Mfin = transform_array * M;

   //Mosaico parcial con el frame alineado
    UMat result_image;
    warpPerspective(img2,result_image,Mfin,Size(x_max-x_min,y_max-y_min),INTER_LINEAR,
            BORDER_TRANSPARENT);

   //Mascara del mosaico con la mascara del frame alineado
    UMat mask_result;
    mask_result.create(result_image.rows,result_image.cols,CV_8UC1);
    mask_result = 0;
    warpPerspective(mask2,mask_result,Mfin,Size(x_max-x_min,y_max-y_min),INTER_LINEAR,
            BORDER_TRANSPARENT);

    //Calculo de la coincidencia del frame y el mosaico
    UMat res_frame = result_image.clone();
    result_image = 0;

    UMat mask_canvas = result_image.clone();
    cvtColor(mask_canvas,mask_canvas,CV_BGR2GRAY);
    Rect rect(-x_min,-y_min,h1,w1);
```

```
    mask1.copyTo(mask_canvas(rect));
    img1.copyTo(result_image(rect));

    float Rate = frame_coincidence(mask_result, mask_canvas);
    LOGD("The %.2f percent of the frame is contained in the partial mosaic.", Rate);

   //Evaluacion del umbral de coincidencia
    UMat mask_buffer = UMat();

    if(Rate > 75){//Revisar condicion de nitidez
        Result = img1.clone();
        Mask_result = mask_buffer.clone();
        Check_val = 0;
        return Check_val;
    }
    else if(Rate < 17){
        Result = img1.clone();
        Mask_result = mask_buffer.clone();
        Check_val = 2;
        return Check_val;
    }

   //Si la coincidencia se encuentra dentro del umbral
   //se devuelve el resultado parcial
    add(mask_result, mask_canvas,mask_buffer);
    warpPerspective(img2,result_image,Mfin,Size(x_max-x_min,y_max-y_min),
            INTER_LINEAR,BORDER_TRANSPARENT);

    Result = result_image.clone();
    Mask_result = mask_buffer.clone();
    Check_val = 1;

    //Codigo de salida indicando la operacion correcta
    return Check_val;

}

//FUncion para determinar las coordinadas maxima y minima para el
//calculo de la matriz de transformacion
void get_limits(Mat img1_dims, Mat img2_dims, int& xmin, int& ymin,
                int& xmax, int& ymax){

   //Reorganizando coordenadas para facilitar el analisis
    Mat tempimg2 = img2_dims.reshape(1);
    Mat tempimg1 = img1_dims.reshape(1);
    Mat VconcatTEMP;
    vconcat(tempimg1,tempimg2,VconcatTEMP);

   //Analizando la ubicacion en memoria de los datos de la Matriz coordinadas para
ordenarlos
   //en un vector
    std::vector<float> array;
    if (VconcatTEMP.isContinuous()) {
        array.assign((float*)VconcatTEMP.data, (float*)VconcatTEMP.data +
VconcatTEMP.total());
    } else {
        for (int i = 0; i < VconcatTEMP.rows; ++i) {
            array.insert(array.end(), VconcatTEMP.ptr<float>(i),
VconcatTEMP.ptr<float>(i)+VconcatTEMP.cols);
        }
    }

   //Inicializando valores para la comparacion
```

```
    int x_min = array[0], y_min = array[1], x_max = array[0],
    y_max = array[1];

   //Comparando valores y determinando maximo y minimo
    for(int i = 0; i < 8; i++){

        if(array[2*i] < x_min){
            x_min = array[2*i];
        }
        if(array[2*i] > x_max){
            x_max = array[2*i];
        }
        if(array[2*i+1] < y_min){
            y_min = array[2*i+1];
        }
        if(array[2*i+1] > y_max){
            y_max = array[2*i+1];
        }

    }

   //Se retorna los valores calculados redondeados
    xmin = (int)(x_min -0.5);
    xmax = (int)(x_max +0.5);
    ymin = (int)(y_min -0.5);
    ymax = (int)(y_max +0.5);

}

//Funcion para determinar la coincidencia entre un frame y el mosaico parcial
float frame_coincidence(UMat img, UMat img2){

   //Obteniendo las imagenes a analizar
    Mat hist,hist2,IMG = img.getMat(cv::ACCESS_WRITE),IMG2 =
img2.getMat(cv::ACCESS_WRITE);

    //Parametros para calculo de histogramas
    int histSize = 256;
    float range[] = { 0, 256 };
    const float* histRange = { range };

   //Calculo de histograma para el frame
    calcHist(&IMG,1,0,Mat(),hist,1,&histSize,&histRange,true,false);
    long N_pixels = img.rows * img.cols;

   //CAlculo de histograma para el mosaico
    calcHist(&IMG2,1,0,Mat(),hist2,1,&histSize,&histRange,true,false);

   //Calculo de histograma para la sustraccion de imagenes
   UMat img3;
    subtract(img2,img,img3);
    Mat hist3, IMG3 = img3.getMat(cv::ACCESS_WRITE);
    calcHist(&IMG3,1,0,Mat(),hist3,1,&histSize,&histRange,true,false);

   //Calculo del porcentaje de inclusion
    std::vector<float >vec(hist.begin<float>(), hist.end<float>());
    long Pix_frame = N_pixels - (long)(vec[0]);
    std::vector<float >vec2(hist2.begin<float>(), hist2.end<float>());
    std::vector<float >vec3(hist3.begin<float>(), hist3.end<float>());
    long var = (long)(vec3[0]) - (long)(vec2[0]);

    float rel = (float(var)/float(Pix_frame))*100;
```

```
    return rel;
}
```