

SEP

SES

TMN

## INSTITUTO TECNOLÓGICO DE CHIHUAHUA II



### “MÓDULO DE UNIFICACIÓN DE IMÁGENES PARA MICROSCOPIA ÓPTICA”

TESIS  
PARA OBTENER EL GRADO DE

**MAESTRÍA EN SISTEMAS COMPUTACIONALES**

PRESENTA:

**ALEXANDRO SIFUENTES DÍAZ**



**DIRECTOR DE TESIS:**  
M.I.S.C. JESÚS ARTURO ALVARADO  
GRANADINO

**CO-DIRECTOR DE TESIS:**  
DR. CALEB CARREÑO GALLARDO

CHIHUAHUA, CHIH., MÉXICO; A FEBRERO DE 2016

# Resumen

El procesamiento de imágenes y los algoritmos de análisis de datos han tenido amplio uso en diferentes y variados campos. La presente tesis cubre uno de sus problemas presentados en el ámbito de la microscopía óptica, pretendiendo proporcionar una herramienta rápida y efectiva que permita la unión de imágenes capturadas por segmentos en una sola imagen panorámica.

El origen de las imágenes proviene de la captura desde un microscopio óptico, y una vez listas para su procesamiento, esta tesis evaluará diferentes técnicas de reconocimiento de patrones (*Harris Corner Detector*, *SIFT*, *SURF* y *ORB*) para determinar cual ofrecerá mayor calidad, velocidad, estabilidad y precisión.

También se mostrará la evaluación de diferentes técnicas de empate empleadas para la obtención de las transformaciones a ser aplicadas, contemplando *knn por fuerza bruta* y *ANN*.

Por último se precisará como se realizará el cómputo de los valores de transformación para generar el mapa que comprenderá todas las imágenes originalmente obtenidas.

# Abstract

Image processing and data analysis algorithms have been widely used in different and varied fields. This thesis covers one of the problems presented in the optical microscopy field aimed to provide a quick and effective tool to join image captured by segments to form a single panoramic image.

The origin of this images comes from the capture through an optical microscope, and, once ready for processing, this thesis evaluate different pattern recognition techniques (*Harris Corner Detector*, *SIFT*, *SURF* and *ORB*) to determine which offer higher quality, speed, stability and accuracy.

It will also be shown the evaluation of different techniques used to obtain the necessary transformation that will be applied to each image, including *knn brute force* and *ANN* algorithms.

Lastly it will be shown how to compute the transformation values to generate the map which will include all images originally obtained.

# Dedicación

*A mi futuro yo, que al leer esto recuerde quien era.*

# Agradecimientos

Quisiera expresar mi gratitud al *Centro de Investigación en Materiales Avanzados, S.C.* por facilitar el acceso a sus instalaciones y equipo, sin el cual no habría sido posible realizar la presente investigación.

Muchas gracias a *Ruben Castañeda* por permitirme el acceso y uso de su equipo de cómputo así como la estadía realizada en su laboratorio.

Agradezco a mis amigos que directa o indirectamente ayudaron en el desarrollo de la presente investigación, en especial a *Juan Dario Muñoz* el cual sé que no leerá ésto, y finalmente a mi madre *Eva Díaz Garcia* por su inherente presión psicológica.

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Planteamiento . . . . .	3
1.2. Visión Computacional y Mezcla de Imágenes . . . . .	5
1.3. Alcances y Limitaciones . . . . .	7
1.4. Justificación . . . . .	8
1.5. Objetivo . . . . .	8
<b>2. Estado del Arte</b>	<b>10</b>
2.1. Algoritmos de extracción de características . . . . .	10
2.2. Descriptores binarios . . . . .	11
2.2.1. FREAK . . . . .	11
2.2.2. BRISK . . . . .	11
2.2.3. ORB . . . . .	12
<b>3. Marco Teórico</b>	<b>13</b>
3.1. Representación y operaciones sobre imágenes . . . . .	13
3.1.1. Representación . . . . .	13
3.1.2. Convolución . . . . .	16
3.1.3. Derivación de una imagen . . . . .	17
3.1.4. Gradiente . . . . .	17
3.1.5. Laplaciano . . . . .	19
3.1.6. Imagen Integral . . . . .	20
3.1.7. Filtros . . . . .	21
3.2. Transformaciones en imágenes . . . . .	27
3.2.1. Puntos 2D . . . . .	28
3.2.2. Traslación . . . . .	28

---

## ÍNDICE GENERAL

3.2.3. Rotación + Traslación . . . . .	29
3.2.4. Escalación + Rotación . . . . .	29
3.2.5. Afinidad . . . . .	29
3.2.6. Proyección . . . . .	30
3.2.7. Jerarquía de las transformaciones 2D . . . . .	30
<b>4. Desarrollo</b>	<b>31</b>
4.1. Extracción de Características . . . . .	32
4.1.1. Harris Corner Detector . . . . .	32
4.1.2. SIFT . . . . .	35
4.1.3. SURF . . . . .	42
4.1.4. ORB . . . . .	47
4.1.5. Resultados experimentales . . . . .	51
4.2. Empate de Características . . . . .	57
4.2.1. <i>knn</i> por Fuerza Bruta . . . . .	57
4.2.2. ANN . . . . .	58
4.2.3. Incrementando Precisión . . . . .	58
4.2.4. Resultados Preliminares . . . . .	59
4.3. Estimación de Similaridad . . . . .	60
4.3.1. LS . . . . .	61
4.3.2. LMS . . . . .	63
4.3.3. RANSAC . . . . .	63
4.4. Composición . . . . .	66
4.4.1. Selección de Ordenamiento . . . . .	66
4.4.2. Posprocesamiento . . . . .	68
4.5. Interfaz Gráfica . . . . .	69
4.5.1. Creando una imagen panorámica . . . . .	71
<b>5. Resultados</b>	<b>74</b>
5.1. Porcentaje de mejora . . . . .	75
5.2. Errores de Unión . . . . .	76
5.3. Discusión . . . . .	77
<b>6. Conclusión</b>	<b>79</b>
<b>Bibliografía</b>	<b>80</b>

# Índice de figuras

1.1. Ejemplo de usos para Image Stitching . . . . .	2
1.2. Microscopio óptico convencional y metalúrgico. . . . .	4
1.3. Diferentes algoritmos existentes en Visión Computacional . . . . .	6
2.1. Descriptores binarios . . . . .	12
3.1. Representación visual de una imagen de $20 \times 20$ en escala de grises . . . . .	14
3.2. Pirámide Gaussiana . . . . .	15
3.3. Gráficas de la derivada de una imagen . . . . .	18
3.4. Gradiente de una imagen . . . . .	19
3.5. Representación de la integral de una imagen . . . . .	20
3.6. Subáreas dentro de una imagen integral . . . . .	22
3.7. Aplicación de diferentes filtros sobre una Imagen . . . . .	22
3.8. Primera y Segunda derivada de Gauss . . . . .	25
3.9. Laplaciano de Gauss . . . . .	26
3.10. Aproximación por Diferencia Gaussiana . . . . .	27
3.11. Transformaciones básicas de dos dimensiones . . . . .	28
4.1. Elementos distinguidos en Harris Corner Detector . . . . .	33
4.2. Clasificación basado en eigenvalores para Harris Detector . . . . .	34
4.3. Vulnerabilidad de Harris Detector frente a la escalación . . . . .	35
4.4. Características de un Keypoint en SIFT . . . . .	36
4.5. Gaussian Scale Space de SIFT . . . . .	37
4.6. Extremos de imágenes adyacentes DoG dentro de SIFT . . . . .	38
4.7. Diferencia entre extremos detectados y reales en SIFT . . . . .	39
4.8. Detección de orientación de un Keypoints en SIFT . . . . .	41
4.9. Representación esquemática de un descriptor SIFT . . . . .	41

4.10. Discretización y aproximación de la 2 <sup>da</sup> derivada gaussiana en SURF . . . . .	44
4.11. Scale-Space en SURF . . . . .	45
4.12. Haar Wavelet en SURF . . . . .	46
4.13. Orientación de puntos de interés en SURF . . . . .	46
4.14. Descriptor SURF . . . . .	47
4.15. Keypoint FAST para ORB . . . . .	48
4.16. Descriptor BRIEF para ORB . . . . .	51
4.17. Tiempo de extracción y descripción de características . . . . .	53
4.18. Número de características detectadas por algoritmo . . . . .	54
4.19. Número de descriptores computados por algoritmo . . . . .	55
4.20. Tiempo consumido tras detección de descriptores por algoritmo . . . . .	56
4.21. Algoritmo $k$ -nn de dos dimensiones con $k = 2$ . . . . .	58
4.22. Prueba de <i>fuerza bruta</i> y <i>FLANN</i> . . . . .	60
4.23. Transformación por Similaridad . . . . .	61
4.24. Regresión Linear por Mínimos Cuadrados . . . . .	62
4.25. Sensibilidad del Algoritmo de Mínimos Cuadrados . . . . .	63
4.26. Dos conjuntos para aplicar RANSAC . . . . .	64
4.27. RANSAC con outliers . . . . .	64
4.28. RANSAC con inliers . . . . .	65
4.29. Ejemplo de ordenamiento de las imágenes finales . . . . .	66
4.30. Árbol de decisión de ordenamiento . . . . .	68
4.31. Prototipo de Interfaz . . . . .	70
4.32. Interfaz de usuario . . . . .	70
4.33. Creación de nuevo proyecto . . . . .	71
4.34. Carga de imágenes . . . . .	71
4.35. Creación de panorámica . . . . .	72
4.36. Exportar panorámica . . . . .	72
4.37. Imagen panorámica resultante . . . . .	73
 5.1. Tiempo de ejecución . . . . .	 74
5.2. Tiempo de ejecución entre Software y Usuarios . . . . .	76
5.3. Comparativa entre Algoritmos . . . . .	78

# Índice de tablas

3.1. Jerarquía de Transformaciones 2D . . . . .	30
5.1. Tiempo de ejecución del software . . . . .	75
5.2. Tiempo de ejecución entre Software y Usuarios . . . . .	76

# Capítulo 1

## Introducción

El campo de la microscopía es un área que ha encontrado cobijo con el avance de las herramientas informáticas. El progreso logrado por estas tecnologías a lo largo del tiempo ha permitido la proliferación de técnicas en el procesamiento y análisis de imágenes, uno de estos casos son los sistemas de unificación de imágenes.

La presente tesis *Módulo de Unificación de Imágenes para Microscopía Óptica* expone el entorno desarrollado el cual permite a un usuario común la toma de imágenes provenientes de un microscopio con la finalidad de unirlas, entregando el conjunto ensamblado para un posterior análisis.

Dentro de los diferentes algoritmos existentes, aquellos cuyo propósito se centran en la unión de imágenes representan un conjunto de entre los más viejos y utilizados en el campo de la Visión Computacional (*Computer Vision*) (CV) [1, cap. 9, pág. 377], su aplicación directa tiene una variedad amplia:

- **Imágenes en alta resolución de objetos inmensos**, como el mapeo completo de una ciudad (fig. 1.1a), un planeta o una galaxia (fig. 1.1b)
- **Imágenes con gran detalle de objetos reducidos**, como la captura a cuerpo completo de un *piojo de buitre* (fig. 1.1c).
- **Imágenes de carácter artístico**: como los panoramas de gran apertura para un paisaje.
- etc.

Todos estos ejemplos hacen frente a las limitaciones que presentan las cámaras, debido, entre otras razones, a su reducida resolución o el limitado campo de visión que hasta la fecha presentan.



(a) Mapa de Google Maps. Créditos: Google, Digital Globe. Fuente: <https://www.google.com/maps>



(b) Cartografía fotográfica de la Galaxia de Andrómeda. Un conjunto masivo de 7,398 exposiciones tomadas desde 411 puntos distintos. Fuente: <http://hubblesite.org>



(c) Puijo de Buitre. Una colección de 1,529 imágenes individuales para generar el cuerpo completo de un piojo de buitre. Fuente <http://www.quekett.org/resources/stack-stitch/stitching>.

**Figura 1.1:** Ejemplo de usos para Image Stitching, incluido el sector público y comercial.

Este proceso automatizado de armar imágenes como un rompecabezas es conocido en la CV como los algoritmos de Mezcla y Fusión de Imágenes (*Image Stitching and Blending*) (ISB) y en la actualidad involucran varias técnicas de detección de patrones, técnicas de proyección, ordenamiento, fusión y técnicas de optimización de imagen.

La microscopía óptica es un sector que aplica y explota estos algoritmos precisamente por la necesidad de sus usuarios por obtener una imagen lo más completa y con la mayor información posible.

Muchos microscopios incluyen sus propias interfaces que permiten una comunicación directa con un ordenador, a sí mismo, algunos ya incluyen software o aplicaciones que permiten cumplir con esta tarea, sin embargo, repetidamente la ventaja que presentan se verá mermada por las limitaciones (económicas o computacionales) que surgen durante su uso, ésto debido a la complejidad computacional que acarrea el desarrollo de esta tecnología, sumado al nivel de especificación requerido en su uso.

Existen muchos factores que se deben tomar en cuenta a la hora de desarrollar un sistema de esta índole aplicable al ámbito de la microscopía óptica, tales como las propiedades básicas que rodean al microscopio (la apertura de los lentes, el nivel de aumento, la escala manejada, el tipo de proyección, las deformaciones existentes, luminosidad, etc.), propiedades informáticas (como la carga computacional y el nivel de abstracción) y aspectos referentes a la ergonomía del software (como el usuario final, la portabilidad y la versatilidad del mismo).

Como usuario, una persona dentro de los campos de estudio científico posiblemente se verá forzado a interactuar con un microscopio y con los algoritmos de CV aquí mencionados, así que la disponibilidad y flexibilidad de estos sistemas garantizarán, al menos, un ahorro de tiempo.

En la primera parte de este capítulo se adentrará brevemente al entorno propio de la Microscopía Óptica y al medio que lo rodea, dando una explicación de su funcionamiento, las partes fundamentales que lo componen, su utilidad, así como quiénes se ven involucrados en su uso.

En las siguientes secciones se dará una introducción al proceso de mezcla de imágenes; se explicará su ubicación dentro de la CV, así como los algoritmos involucrados y la metodología seguida con la finalidad de familiarizar al lector con el proyecto. Al final del capítulo se estudiarán los alcances, limitaciones y justificaciones que mueven y motivan a la investigación, finalizando con el objetivo base de la presente tesis.

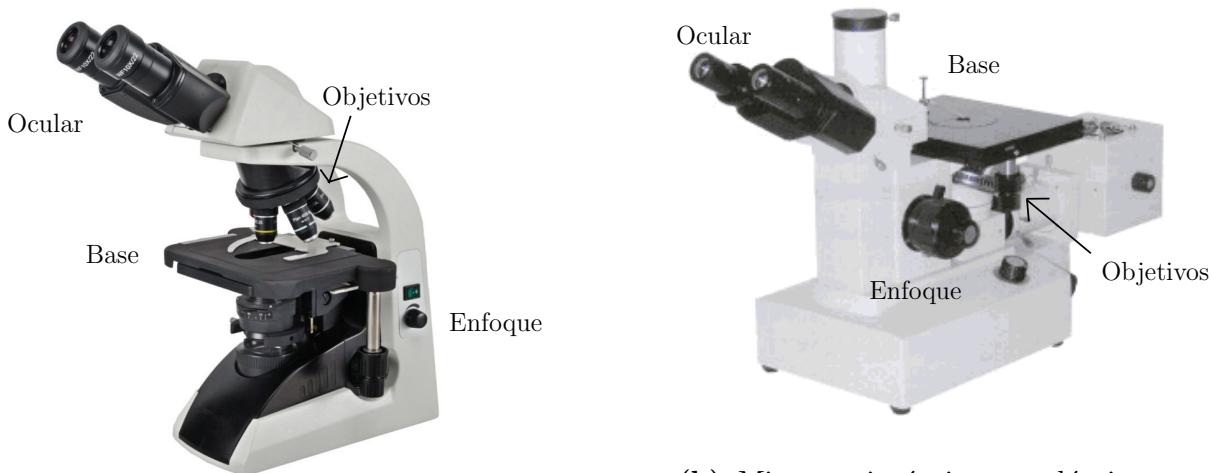
### 1.1. Planteamiento

Un microscopio es un instrumento común dentro del entorno científico y su uso es relativamente sencillo: se toma una muestra del objeto a estudiar y se coloca en una base para que se proyecte una imagen magnificada, lo cual permite estudiar detalles minúsculos que escapan a la capacidad normal del ojo humano. Si el microscopio cuenta con una interfaz

de comunicación hacia una computadora, entonces dicha imagen puede ser almacenada para posteriormente ser procesada y analizada.

Hoy en día existe una amplia gama de microscopios: óptico, simple, de luz ultravioleta, de fluorescencia, petrográfico, de campo oscuro, de contraste de fases, de luz polarizada, confocal, electrónico de transmisión, electrónico de barrido, de iones en campo, de sonda de barrido, de efecto de túnel, de fuerza atómica, estereomicroscopio binocular, invertido, metalográfico, etcétera. El desempeño de estos depende del campo de estudio, aunque generalmente resultan ser una variación de un microscopio óptico o electrónico.

En el caso de los microscopios ópticos, la magnificación total usualmente vendrá determinada por el aumento presente en el ocular y en el objetivo (ver figura 1.2), obtenido mediante una multiplicación de ambos (generalmente el valor viene marcado en el armazón cilíndrico que rodea a estos dos componentes); así por ejemplo, si el ocular está graduado con un aumento de  $40x$  (es decir, incrementa 40 veces a un objeto) y el objetivo con  $10x$ , entonces el aumento final sobre la muestra será de  $40x \times 10x$ , o  $400x$ .



(a) Microscopio óptico convencional, con sus componentes esenciales marcados.

(b) Microscopio óptico metalúrgico, también conocido como microscopio invertido, debido a que la muestra se posiciona en sentido contrario a uno convencional.

**Figura 1.2:** Microscopio óptico convencional y metalúrgico.

Con los microscopios electrónicos el rango de magnificación es sumamente superior y su utilidad se centra en la observación indirecta de fenómenos u objetos a un nivel molecular y/o atómico, donde los aumentos cubren un rango superior a  $500,000x$  (lejos del rango de los microscopios ópticos, donde en el mejor de los casos llegará a los  $1,000x$ ), de cualquier

manera, dichos aparatos escapan al enfoque de la tesis, por lo cual no es necesario un conocimiento más profundo sobre ellos.

Como investigador, estudiante o analista; un usuario afín al estudio de materiales eventualmente hará uso del microscopios óptico durante los proyectos que él/ella realice. Este uso, en un gran número de casos, traerá consigo la forzosa implementación de alguna técnica que le permita unir las imágenes que de ésta herramienta obtenga debido, entre otras razones, a la baja resolución de las cámaras de captura de los equipos.

## 1.2. Visión Computacional y Mezcla de Imágenes

Los aspectos primordiales que permiten establecer un margen de rendimiento óptimo radican en la toma de imágenes en muy alta calidad y con una excelente resolución, ambos aspectos, son cuestiones que vienen limitados por la resolución de la cámara.

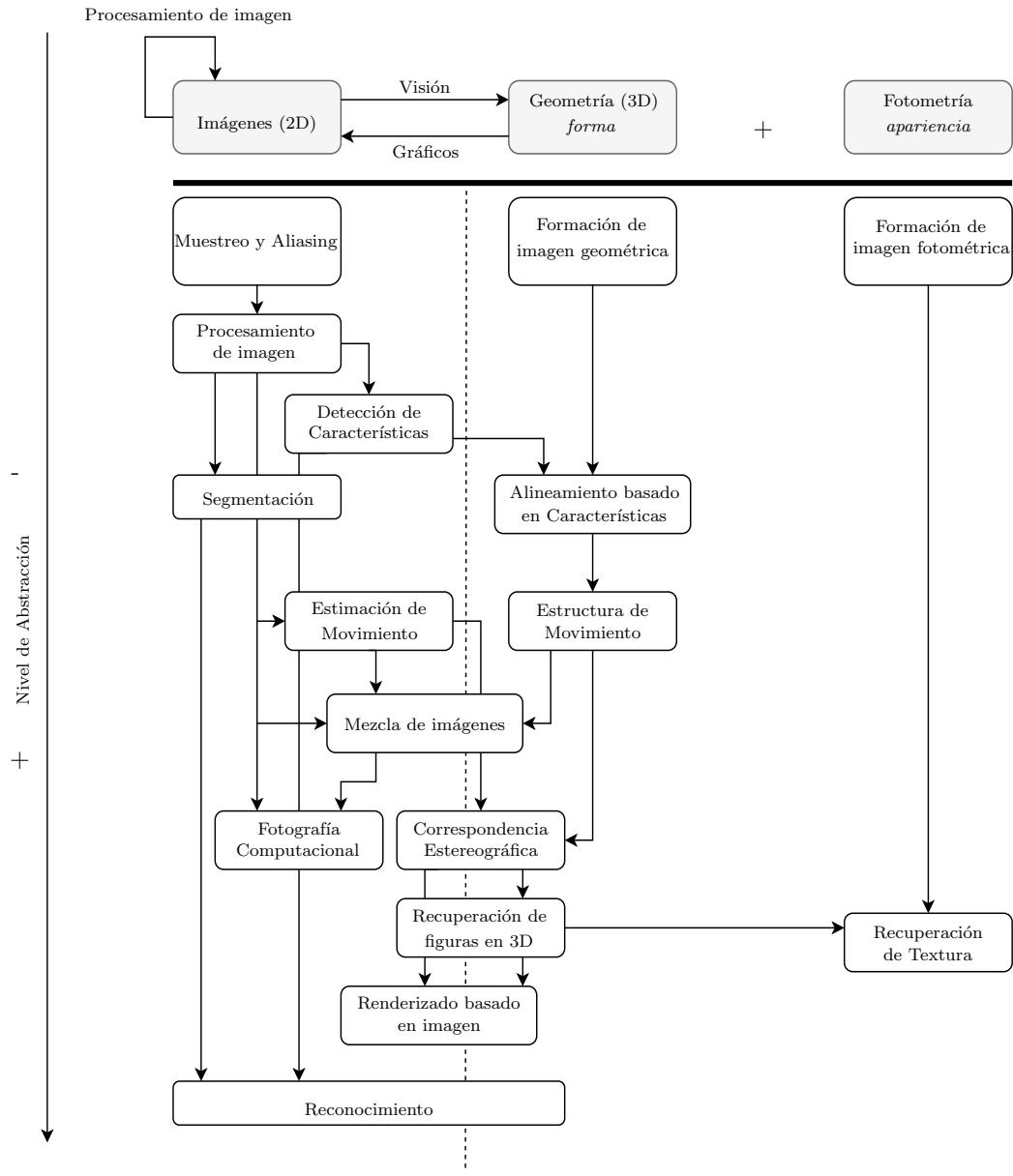
Se podría pensar que entre mejor sean las características inherentes de estos dispositivos, mejor será la toma de imágenes y sería prescindible la utilización de los algoritmos de ISB; pero en la práctica este planteamiento es insostenible, en parte por la inexistencia de cámaras de esta índole, pero mayormente debido a que el costo de dichos dispositivos sobrepasa las ganancias o beneficios obtenidos por su uso.

Existiendo un marco amplio, es normal encontrar investigaciones que enfoquen sus estudios en la especialización de los algoritmos requeridos para satisfacer diferentes demandas existentes en CV; así se pueden encontrar extractores de dimensiones [2], detectores de rostros faciales [3], reconocimiento de caracteres [4] o unión de imágenes [5] siendo estos últimos los de especial interés para el proyecto.

Cada uno de estos ejemplos expone su propio reto computacional que involucra de alguna u otra manera uno o varios conceptos de CV, por fortuna, con la evolución tecnológica, ha proliferado la investigación en el área y buena medida de ellos cuentan ya con al menos uno o varios estudios base.

El área computacional que investiga las técnicas de fusión de imágenes se encuentra en las Ciencias Computacionales (*Computational Science*) (CS), en la ya mencionada CV, representando una familia amplia de algoritmos.

En la figura 1.3 se observa el nivel de abstracción requerido (alto o bajo) y el tipo de análisis que se realiza (a nivel imagen, geométrico o textura). Se puede apreciar cómo los



**Figura 1.3:** Diferentes algoritmos existentes en Visión Computacional. De izquierda a derecha se diferencian en tres categorías: aquellos basados en imagen, geometría o en apariencia; mientras que la posición vertical representa el nivel de abstracción requerido, incrementando entre más abajo se posicione. Algunos algoritmos (como, por ejemplo, la mezcla de imágenes) involucran una manipulación a nivel imagen y geométrico, debido a ello se posicionan entre ambas columnas. Una mayor descripción de la figura puede ser encontrada en [1, cap. 1, pág. 19].

algoritmos de ISB (bajo el nombre de *Mezcla de imágenes*) se posicionan en una zona central en el eje vertical y entre el análisis a nivel imagen y geométrico.

### 1.3. Alcances y Limitaciones

El sistema cubre una necesidad en el entorno de la Microscopía Óptica, pretendiendo dar solución al tiempo muerto presente durante el proceso de estudio de muestras con el microscopio. Dentro de los *alcances* se tienen los siguiente puntos:

- **Mezcla de Imágenes.** El resultado final se ofrece como una imagen básica en formato estándar (*jpeg, png, bmp, etc.*).
- **Captura Optimizada.** El proceso de adquisición de imágenes está controlado por el investigador, la posición puede ser rastreada en tiempo real.
- **Personalización Operacional.** Las características variables de los algoritmos pueden ser ajustadas a diferentes entornos. Esta información puede ser consultada a detalle en las secciones finales del capítulo 4 de la tesis.

Las *limitaciones* existentes en el sistema vienen influenciadas en su mayoría por la capacidad de los algoritmos, por el ecosistema que interactúa con él y con las características del equipo hardware que lo implementa, agrupadas básicamente en la siguiente lista:

- **Microscopía Óptica.** La investigación sólo incluye esta clase de microscopios. Lo cual no garantiza un resultado similar fuera del área.
- **Equipo de Hardware.** Resulta imprescindible la existencia de un medio que permita realizar la captura de imágenes con su respectivo almacenamiento digital. Además, las características del equipo influirán en el tiempo computacional requerido para realizar un proceso. Usualmente un equipo convencional actualizado brinda un resultado aceptable.
- **Ruido en la imagen.** Los algoritmos de detección de características son sensibles al ruido, es posible realizar pre-procesamientos que minimicen ésto, pero su desempeño no es perfecto.
- **Variación en intensidad.** Algunos algoritmos utilizados en ISB son sensibles a la luz, propiamente aquellos de la primera etapa del proceso estudiado en la sección 4.1.

- **Detección de Características.** Para una efectiva mezcla de imágenes, resulta necesario que el objeto a ser estudiado cuente propiamente con suficientes características detectables. Esta variable depende mucho de la textura o relieve presente en la muestra, así como el contraste existente.

## 1.4. Justificación

En este creciente progreso de las tecnologías de la información, destaca el esfuerzo realizado en investigaciones que buscan brindar y explotar sistemas cada vez más personalizados, veloces y de fácil operación. Que existan sectores (tanto industriales, como sociales) donde persista un mínimo de potencial para aplicar nuevas técnicas (o perfeccionar las ya existentes) son cuestiones que priorizan y promueven esta investigación.

Aunque incluso con la presente existencia y la variada opción en el mercado, las posibilidades operacionales de estos sistemas siguen teniendo limitaciones, en parte por la pesada carga computacional que conlleva su utilización (se requiere un equipo relativamente moderno), pero aún más porque los sistemas que realizan ésto lo hacen como una rutina secundaria dentro de un entorno multitareas y su generalización se hace presente, en otras palabras, en busca de cubrir un mayor campo y hacer más rentable una aplicación (como las aplicaciones más conocidas dentro del procesamiento de imágenes) en ocasiones se opta por ofrecer soluciones rápidas incluidas dentro del software que pueden o no adaptarse a entornos de trabajo más específicos, como es el caso de la Microscopía Óptica.

El desarrollo de la presente investigación ofrece una solución a este problema. Crear un módulo específicamente dedicado a los algoritmos de ISB requeridos dentro de la Microscopía Óptica, verá reflejada su mejora directamente en la efectividad de los equipos utilizados en el área, es decir, al reducir el tiempo operacional del microscopio se liberará su uso para tareas de mayor impacto.

## 1.5. Objetivo

El objetivo principal que se tiene con el desarrollo del sistema de ISB es trasladar el trabajo manual existente de la mezcla de imágenes a un sistema automatizado, reduciendo el tiempo invertido en dicha tarea de un rango entre 1 a 10 minutos a un proceso automático de 20 segundos, minimizando la pérdida de tiempo y los cuellos de botella.

Así mismo, para cumplir esta tarea, se deben cubrir de manera más específica los siguientes puntos:

- **Módulo de rastreo y captura de imágenes.** La adquisición de imágenes debe ser permitido desde la aplicación. Así mismo, un rastreo auxiliar viene implementado como objetivo para facilitar la captura de muestras.
- **Módulo de alineamiento de imágenes.** Este es un proceso inherente en los algoritmos ISB, incluye la extracción de características y el empate de éstas, su estudio se aborda en el capítulo 4 y con mayor énfasis en las secciones 4.1, 4.2 y 4.3.
- **Módulo de composición de imágenes.** Aquí se incluye el proceso de aplicación de las transformaciones obtenidas, su estudio inicia en la sección 4.4.
- **Interfaz Gráfica de Usuario (*Graphic User Interface* o *User Interface*) (UI).** Entendido como la entrega final del software donde se explica la herramienta desarrollada.
- **Validación, cuantificación y comparación entre algoritmos.** Al existir varios, es requerido establecer los criterios y parámetros que permita su adecuada funcionalidad, de esta manera se puede establecer el algoritmo respectivo para el ambiente aplicado.
- **Validación, cuantificación y comparación entre entorno antes y después de la aplicación.** Como análisis conclusivo, se debe parametrizar las mejoras existentes en la aplicación del sistema para un caso de estudio.

# Capítulo 2

## Estado del Arte

### 2.1. Algoritmos de extracción de características

La referencia directa cuando se habla de detección de similitudes entre imágenes en sistemas ISB es el descriptor *SIFT* de David Lowe [6],[7]<sup>1</sup>. A pesar de ser un algoritmo de más de 15 años, sigue siendo base para posteriores metodologías, principalmente porque él mismo brinda además de la detección y creación del descriptor, un método para la etapa de empate.

Éste es un algoritmo trabaja basado en una aproximación del *Laplaciano de Gauss* llamado *Diferencia Gaussiana* con la finalidad de obtener un descriptor que fuera robusto frente a cambios de posición y rotación, logrando cierta invarianza frente a la escalación generando una pirámide de estos filtros como se verá más delante. Su descriptor es un vector de 128 dimensiones que puede ser comparado mediante distancia euclidiana para la etapa de empate y aunque muy robusto, hoy en día es considerado lento.

Para mejorar este problema, Bay *et al* [8],[9] crea *SURF* que sigue básicamente el mismo principio que su predecesor, teniendo su mejora en la detección de los puntos de interés mediante lo que se le conoce como *Fast Hessian* que es una aproximación rápida a una matriz Hessiana. Su descriptor es un vector de 64 dimensiones construido a partir de aproximaciones por respuestas a un wavelet *Haar*<sup>2</sup>.

El problema con los algoritmos mencionados es que su implementación se encuentra patentada, lo que en teoría implicaría un coste inicial que el proyecto intenta evitar, además de

---

<sup>1</sup>Su método se aborda propiamente en la sección 4.1, en 4.1.2.

<sup>2</sup>Más a fondo sobre *SURF* en la sección 4.1, subsección 4.1.3

ser descriptores de tamaño considerable. Pero en años posteriores a la creación de *SURF* han surgido otras opciones más que brindan, de alguna u otra manera, soluciones similares a lo que se le conocen como *descriptores binarios*, algunos ejemplos se mencionan a continuación:

## 2.2. Descriptores binarios

Básicamente, un descriptor binario es una secuencia de bits que describe un punto en particular de manera que permita ser reconocido bajo diferentes transformaciones. A diferencia de *SIFT* y *SURF*, los descriptores binarios tienen la ventaja de ser de muy reducido tamaño y de rápido cómputo.

### 2.2.1. FREAK

Mencionado en [10], este algoritmo propone la extracción de descriptores inspirado en la visión humana y más precisamente, en la retina, de allí su nombre (*Fast REtina Keypoint*). Un conjunto de cadenas binarias son computadas comparando intensidad de imágenes sobre un patrón muestra retinal. *FREAK* es usualmente fácil de computar, requiere poca memoria y puede competir con los descriptores antes mencionados.

### 2.2.2. BRISK

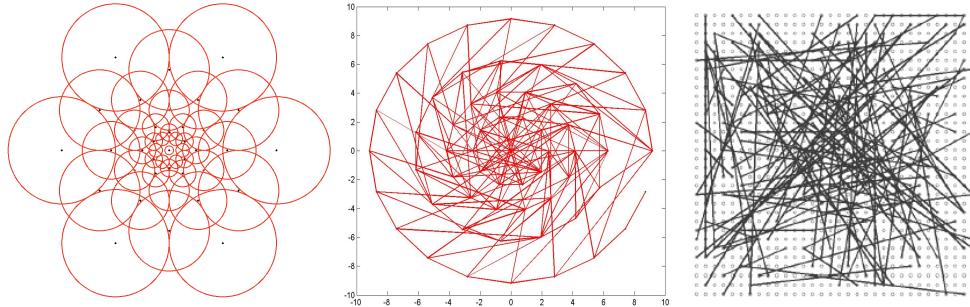
O *Binary Robust Invariant Scalable Keypoints* como se ve en [11], *BRISK* es un descriptor que cuenta con un patrón de muestra compuesto de anillos concéntricos. Al considerarse un punto de interés, se toma un pequeño *patch* al que se le aplica un suavizado gaussiano. Para obtener el vector binario se toma la intensidad de comparación mediante un conjunto de pares discretizando mediante la siguiente comparativa:

$$b = \begin{cases} 1, & I_{p_j^\alpha, \sigma_j} > I_{p_i^\alpha, \sigma_i} \\ 0, & \text{sino} \end{cases} \quad (2.1)$$

Es bueno frente a cambios de perspectiva y algo vulnerable frente a cambios de iluminación, difuminado o compresión de imagen (como en archivos *JPEG*).

### 2.2.3. ORB

*Oriented FAST and Rotative BRIEF*[12]<sup>3</sup>. Es una combinación de un detector de características (*FAST*[13]) y un descriptor (*BRIEF*[14]) a los cuales se les han realizado ciertos ajustes para mejorar su desempeño en cambios de rotación y escalación.



**Figura 2.1:** Descriptores binarios. De izquierda a derecha: FREAK, BRISK y ORB.

---

<sup>3</sup>Más sobre éste algoritmo en la subsección 4.1.4

# Capítulo 3

## Marco Teórico

### 3.1. Representación y operaciones sobre imágenes

Uno de los elementos básicos con el cual se trabaja en el campo de la CV viene siendo la imagen, que puede ser descrita como una representación estática del mundo que nos rodea, obtenida mediante diferentes medios, tales como sensores ópticos, cámaras, radares o algún dispositivo que permita la captura y representación visual de un fenómeno.

Pero, ¿cómo ocurre esta representación? ¿qué es para CV una imagen? y más importante aún ¿cómo se puede comprender la información que contiene? son las preguntas básicas que surgen al iniciar en el entorno del procesamiento de imágenes.

#### 3.1.1. Representación

En CV e ISB, la representación más usual para las imágenes se realiza a través de dos estructuras: **matrices** y **pirámides**.

##### 3.1.1.1. Matriz

Una **matriz** es la representación básica para las imágenes, consta de un arreglo bidimensional numérico de tamaño  $m \times n$ , donde cada uno de los elementos representa un **pixel**, cuyo valor está dado por el rango de 0 a  $(2^n - 1)$ , siendo  $n$  el número de bits involucrados. La figura 3.1 muestra una imagen en escala de grises y su correspondiente arreglo matricial.

La representación de colores, transparencia y otras propiedades de una imagen se logra mediante matrices bidimensionales transpuestas donde cada una de estas expresa la in-



117	125	133	127	130	130	133	121	116	115	100	91	93	94	99	103	112	105	109	106
134	133	138	138	132	134	130	133	128	123	121	113	106	102	99	106	113	109	109	113
146	147	138	140	125	134	124	115	102	96	93	94	99	96	99	100	103	110	109	110
144	141	136	130	120	108	88	74	53	37	31	37	35	39	53	79	93	100	109	116
139	136	129	119	102	85	58	31	41	77	51	53	53	33	37	41	69	94	105	108
132	127	117	102	87	57	49	77	42	28	17	15	13	13	17	41	53	69	88	100
124	120	108	94	72	74	72	31	35	31	15	13	15	11	15	13	46	75	83	96
125	115	102	93	88	82	42	79	113	41	19	100	82	11	11	17	31	91	99	100
124	116	109	99	91	113	99	140	144	57	20	20	15	11	15	17	63	87	119	124
136	133	133	135	138	133	132	144	150	120	24	17	15	15	17	20	115	113	88	150
158	157	154	149	145	133	127	146	150	116	35	20	19	28	105	124	128	141	171	
155	154	156	155	146	155	154	154	147	139	148	150	138	120	128	129	130	151	156	165
150	151	154	162	166	167	169	174	172	167	177	166	164	140	134	120	121	120	127	172
145	149	151	157	165	169	173	179	176	166	166	157	145	136	129	124	120	130	163	168
144	148	153	160	158	158	165	172	165	169	157	151	149	141	130	140	151	162	169	167
144	141	147	155	154	149	156	151	157	157	151	144	147	147	149	159	158	159	166	165
139	140	140	150	153	151	150	146	140	139	138	140	145	151	149	156	156	162	162	161
136	134	138	146	156	164	153	146	145	136	139	139	140	141	149	157	159	161	169	166
136	133	136	135	144	159	168	159	151	142	141	145	139	146	153	156	164	167	172	168
133	129	140	142	146	159	167	165	154	151	146	141	147	154	156	160	161	157	153	154

**Figura 3.1:** Representación visual de una imagen de  $20 \times 20$  en escala de grises. Cada pixel en dicha imagen entra dentro de un rango de  $0$  a  $(2^8 - 1)$  o lo que es igual de  $0$  a  $255$  tonalidades diferentes de negro a blanco. Desde el punto de vista de CV ambas representaciones son homónimas (la visual y su arreglo matricial). Fuente [15, cap. 2, pág. 17].

tensidad de una propiedad, así por ejemplo, el formato popular *RGB* expresa una imagen a color a través de tres matrices donde el valor final se forma mediante la combinación en intensidad de las diferentes tonalidades de los colores rojo, verde y azul (*Red*, *Green*, *Blue* de allí el nombre).

Una matriz puede ser expresada matemáticamente como una tabla numérica, o mediante una letra (usualmente mayúscula), la ecuación 3.1 muestra la representación más usual.

$$A_{m,n} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \quad (3.1)$$

### 3.1.1.2. Matriz Simétrica

Una matriz simétrica es una matriz cuadrada ( $m = n$ ) y además es igual a su transpuesta.

$$A = A^T \quad (3.2)$$

### 3.1.1.3. Matriz Hessiana

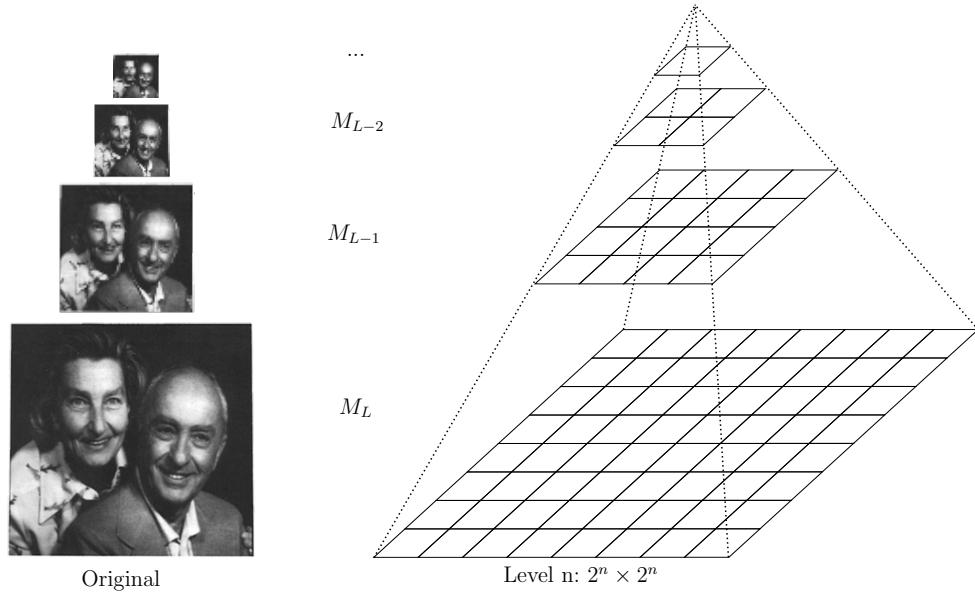
Una matriz hessiana ( $\mathcal{H}$ ) es una matriz cuadrada que contiene las segundas derivadas parciales de una función.

$$\mathcal{H}(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \quad (3.3)$$

### 3.1.1.4. Pirámide

Para minimizar el tiempo computacional requerido, se suelen usar métodos para reducir la resolución de una imagen, esta estructura generada se le conoce como: ***pirámide gaussiana***; la figura 3.2 muestra el esquema de ésta, en ella se puede apreciar el efecto provocado en la imagen, así como la apilación “piramidal” que se genera.

Una pirámide gaussiana es una secuencia  $\{M_L, M_{L-1}, \dots, M_0\}$  de imágenes, donde  $M_L$  tiene la misma dimensión y elementos que la imagen original, y  $M_{L-1}$  es obtenida de  $M_L$  difuminando y reduciendo su resolución a través de filtros gaussianos ([16, cap. 4, pág. 107] y [17]). Este es un proceso rápido y no ocupa demasiado espacio, pero dificulta el análisis, ya que para detectar los cambios en la imagen original se debe agrandar e interpolar entre matrices.



**Figura 3.2:** Pirámide Gaussiana. La imagen original es repetidamente filtrada y submuestreada para generar la secuencia de imágenes de resolución reducida. Fuente [17].

Para la obtención de los elementos dentro de la *pirámide gaussiana* se sigue el siguiente

procedimiento:

1. Teniendo una imagen  $I$ , el primer elemento  $M_L$  representa una copia de  $I$  (ecuación 3.4).
2. El siguiente elemento  $M_{L-1}$  se obtiene mediante la **convolución** de  $M_L$  con un filtro Gaussiano (ecuación 3.5).
3. Una vez realizado este proceso, la imagen sufrirá un difuminado o fuera de foco debido a la aplicación del gaussiano (ver tema 3.1.7.5), para reducir su dimensión se procede a eliminar las columnas y renglones nulos, de esta manera se obtendrá una reducción de 1/4 de la imagen original.
4. Los siguientes elementos se obtienen de manera similar como se muestra en la ecuación 3.6.

$$M_L = I \quad (3.4)$$

$$M_{L-1} = w * M_L \quad (3.5)$$

$$M_{L-2} = w * M_{L-1} \quad (3.6)$$

donde  $w$  es un filtro gaussiano como el mostrado en la ecuación 3.29. Para una expresión más formal, dado un kernel de  $5 \times 5$  (visto en la ec.3.29), la convolución estará dada por [18, 17]:

$$M_{L-1}(x, y) = \sum_{m=-2}^2 \sum_{n=-2}^2 w(m, n) M_L(2x + m, 2y + n) \quad (3.7)$$

Véase los puntos 3.1.2 y 3.1.7 para una mejor comprensión de la ecuación superior.

### 3.1.2. Convolución

Una convolución es el operador matemático que representa la superposición de dos funciones (en este caso dos matrices o imágenes), expresada formalmente en la ecuación 3.8 mostrada a continuación [19]:

$$h(i, j) = \sum_{a=a_{start}}^{a_{end}} \sum_{b=b_{start}}^{b_{end}} g(a, b) f(i - a, j - b) \quad (3.8)$$

o bien

$$h = f * g \quad (3.9)$$

donde  $h(i, j)$  es la convolución de las funciones  $f(i, j)$  y  $g(a, b)$  representado por el símbolo  $*$ ; siendo usualmente  $g(a, b)$  un **filtro**.

### 3.1.3. Derivación de una imagen

En numerosas ocasiones resultará necesario saber en qué parte de una imagen hay cambios de intensidad, en qué dirección va este cambio, así como la magnitud y el ángulo de este cambio, toda esta información es obtenida a través de la derivada de una imagen.

Una derivada de una imagen ofrece información que permite determinar la magnitud en relación a una dirección (ver figura 3.3). Dentro de CV es común el uso de los ejes horizontales y verticales de una imagen, y mediante cálculos determinar la dirección y el ángulo, de cualquier manera, es posible realizar la derivada de una imagen sobre cualquier dirección.

Al igual que en cálculo, la derivada puede ser de un orden determinado, aunque en CV usualmente se tratan las derivadas de primer orden (**gradiente**) y segundo orden (**laplaciano**). En los temas 3.1.4 y 3.1.5 se mencionan estos casos.

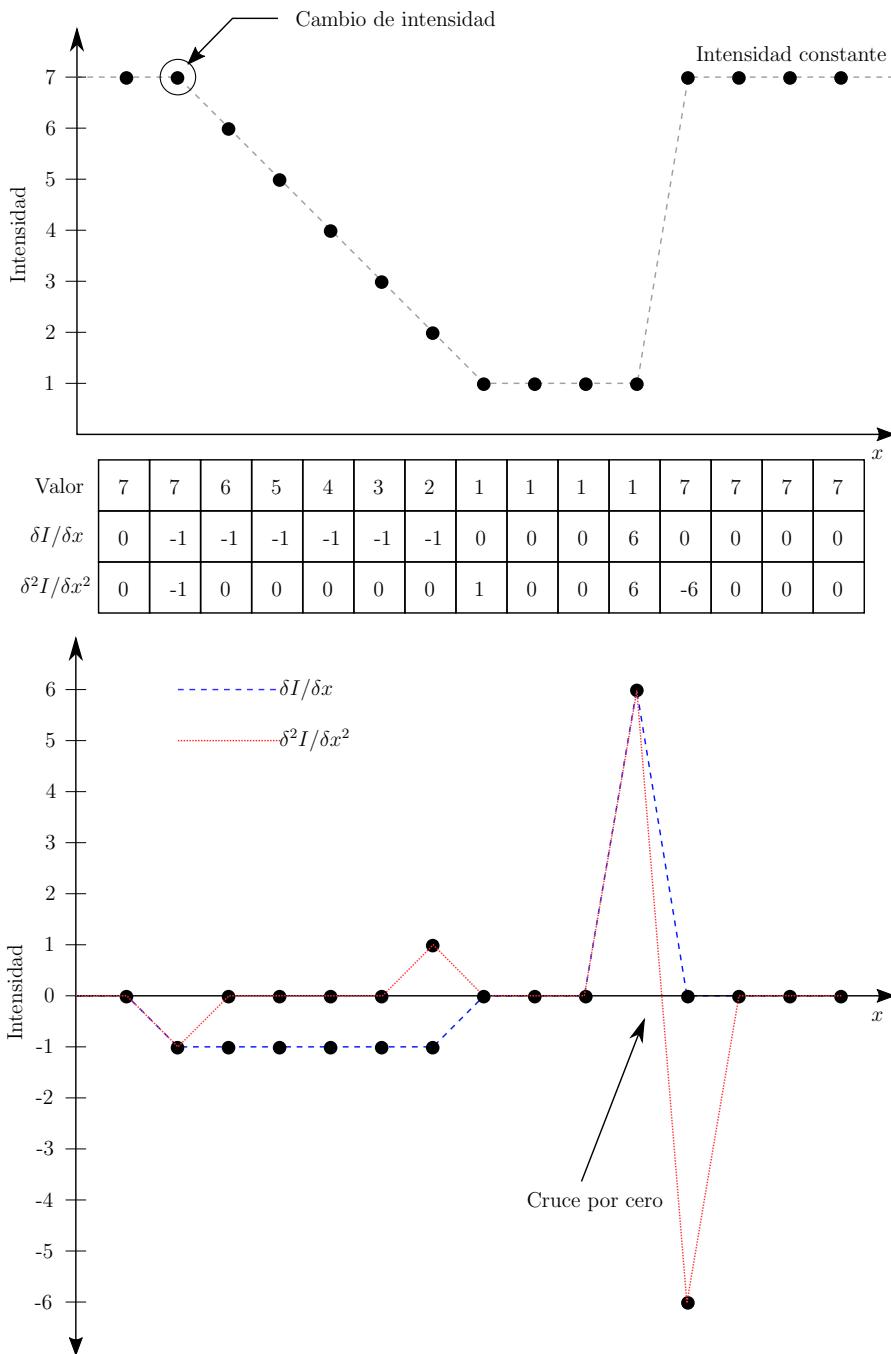
El comportamiento de la derivada de primer y segundo orden sobre una imagen se aprecia en la figura 3.3, aquí se presenta un cambio de intensidad tenue y fuerte; cuando sucede este último surge un **cruce por cero** (*zero-crossing*), que en filtrado de imagen resulta en una buena forma de detectar un borde.

### 3.1.4. Gradiente

La representación matemática que permite determinar la relación entre la derivada de primer orden horizontal y vertical de una imagen es el *gradiente* ( $\nabla$ ), el cual es representado por el vector:

$$\nabla I(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{bmatrix} = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \quad (3.10)$$

y presenta dos propiedades importantes: la magnitud (ec. 3.11, ver fig. 3.4d), que describe qué tan fuerte cambia la intensidad; y el ángulo (ec. 3.12), que indica la dirección de la

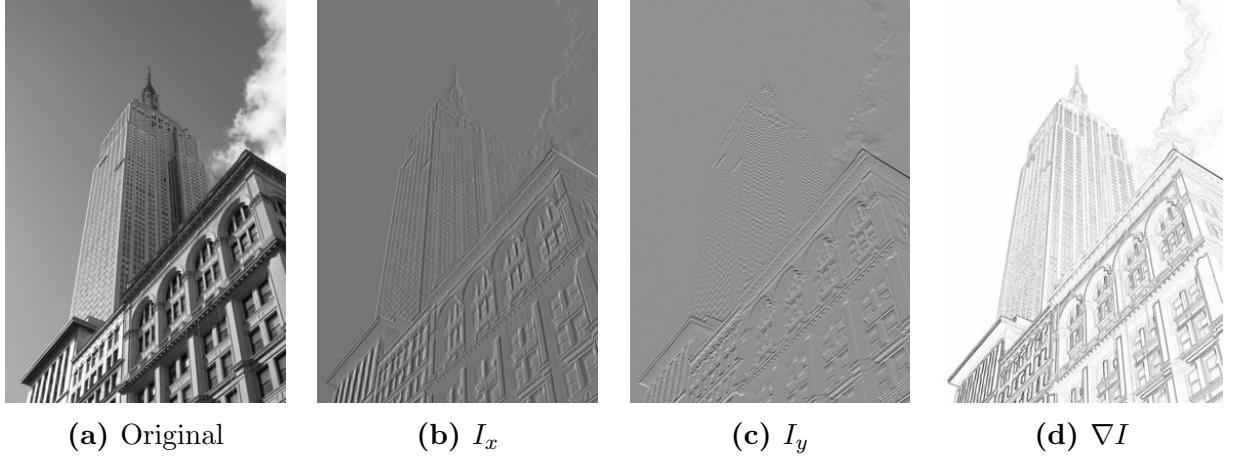


**Figura 3.3:** Gráficas de la derivada de una imagen. La primera gráfica muestra cómo cambia la intensidad a lo largo de un eje (en este caso el eje  $x$ ), la segunda gráfica muestra el comportamiento observable por la primera y segunda derivada. Se puede apreciar que el cambio fuerte de intensidad (comúnmente los bordes) genera un *cruce por cero* en la segunda derivada.

mayor intensidad en cada punto de la imagen.

$$|\nabla I| = \sqrt{I_x^2 + I_y^2} \quad (3.11)$$

$$\alpha = \arctan(I_y/I_x) \quad (3.12)$$



**Figura 3.4:** Gradiente de una imagen. Muestra de la magnitud del gradiente de la imagen 3.4a, en este caso fue utilizado un filtro Sobel. Fuente [20, cap. 1, pág. 34].

Para computar el gradiente, se suelen utilizar aproximaciones discretas usualmente con **convolución** de **filtros** para obtener las derivadas de los ejes  $x$  y  $y$ :

$$I_x = I * G_x \quad y \quad I_y = I * G_y \quad (3.13)$$

donde  $G_x$  y  $G_y$  son filtros como **Prewitt** o **Sobel** que aproximan derivaciones de primer orden (ver tema 3.1.7 donde se detallan ambos filtros).

### 3.1.5. Laplaciano

El equivalente de un gradiente para una derivada de segundo orden es el *laplaciano*, y representa la relación de la derivada de segundo orden sobre los ejes  $x$  y  $y$ , expresado por el símbolo  $\nabla^2$ . A diferencia del gradiente, el *laplaciano* es **isotrópico**, lo cual lo hace invariante a la rotación [16, cap. 5, pág. 133].

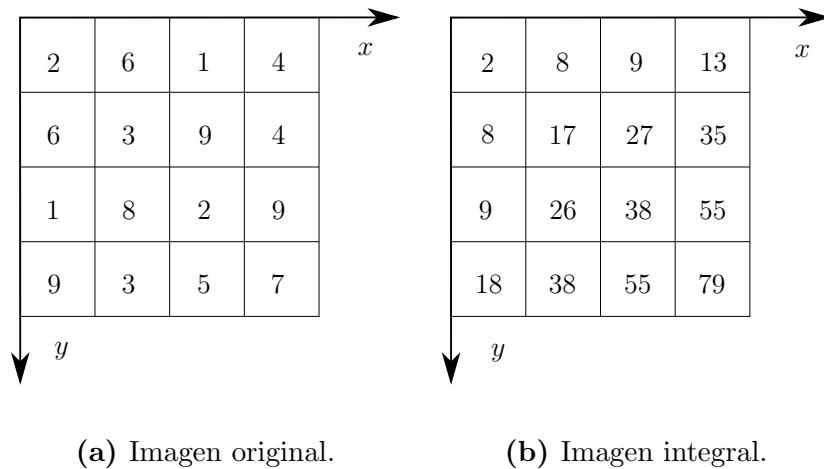
El laplaciano es un operador que esta dado por:

$$\nabla^2 I(x, y) = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2} \quad (3.14)$$

Su aproximación discreta se menciona en el tema 3.1.7.4 y su utilidad está presente en la mayoría de los algoritmos de extracción de características principalmente por su cualidad isotrópica.

### 3.1.6. Imagen Integral

Mencionadas en [21] por Paul Viola y Michael Jones, básicamente, una imagen integral (*integral image* ó *summed area table*) puede ser descrita como la sumatoria de los valores de cada pixel de una imagen. Si se observa la figura 3.5b se puede apreciar que para cualquier punto de la imagen original (fig. 3.5a), el mismo punto en 3.5b representa la suma de todos los valores izquierdos-superiores que le precedieron más él mismo [22]. La principal utilidad de esta representación encuentra cabida en los algoritmos de extracción de características como se ve en el mismo artículo citado con anterioridad y como se verá en secciones posteriores.



**Figura 3.5:** Representación de la integral de una imagen.

Por ejemplo, al tomar el pixel (2, 2) de la imagen integral (fig. 3.5b) su valor estará dado por la suma de  $2 + 6 + 1 + 6 + 3 + 9 + 1 + 8 + 2 = 38$  de los pixeles de la imagen 3.5a.

Generalizando este procedimiento, en [9] hacen uso de la siguiente ecuación:

$$I_{\Sigma}(x) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j) \quad (3.15)$$

donde  $I_{\Sigma}(x)$  expresa la imagen integral que abarca desde (0,0) hasta el punto  $x = (x, y)^T$ <sup>1</sup>.

<sup>1</sup>Apreciar la diferencia existente entre  $x$  y  $x$

Una vez obtenida  $I_\Sigma$ , el valor para cualquier pixel se puede calcular mediante:

$$s(x, y) = I(x, y) + s(x - 1, y) + s(x, y - 1) - s(x - 1, y - 1) \quad (3.16)$$

siendo  $s(x, y)$  un punto dentro de la imagen integral e  $I(x, y)$  es el mismo punto dentro de la imagen original.

Ahora bien, si se desea calcular un área interna (como el área dentro de los puntos  $A, B, C, D$  en la figura 3.6b), se aplica la siguiente fórmula:

$$I(x', y') = s(A) + s(D) - s(B) - s(C) \quad (3.17)$$

siendo  $I(x', y')$  el área buscada, y  $s(A), s(B), s(C), s(D)$  valores que se obtienen fácilmente ya con la imagen  $I_\Sigma$ , la obtención del área es cuestión de sumas y restas:

$$I(x', y') = s(A) + s(D) - s(B) - s(C) \quad (3.18)$$

$$= 17 + 79 - 38 - 35 \quad (3.19)$$

$$= 23 \quad (3.20)$$

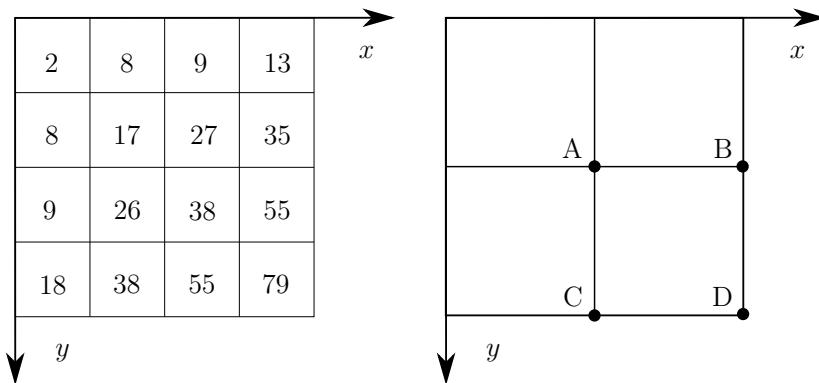
Regresando a la figura 3.5a y sumando manualmente el área, se obtendrá el mismo resultado.

Esta clase de operaciones ofrece una ventaja sobre la complejidad operacional al no requerirse un cálculo más complicado para determinar áreas (como la utilización de operaciones de orden superior).

### 3.1.7. Filtros

En CV, un *filtro* (también conocido como *núcleo*, *kernel*, *máscara*, *operador* o *función de dispersión de punto* (*point-spread function*, [19])) es una matriz 2D pequeña que, mediante la **convolución** permite, entre otras cosas, el pre-procesamiento de una imagen para la extracción de bordes, difuminado o agudizado, etcétera (ver figura 3.7).

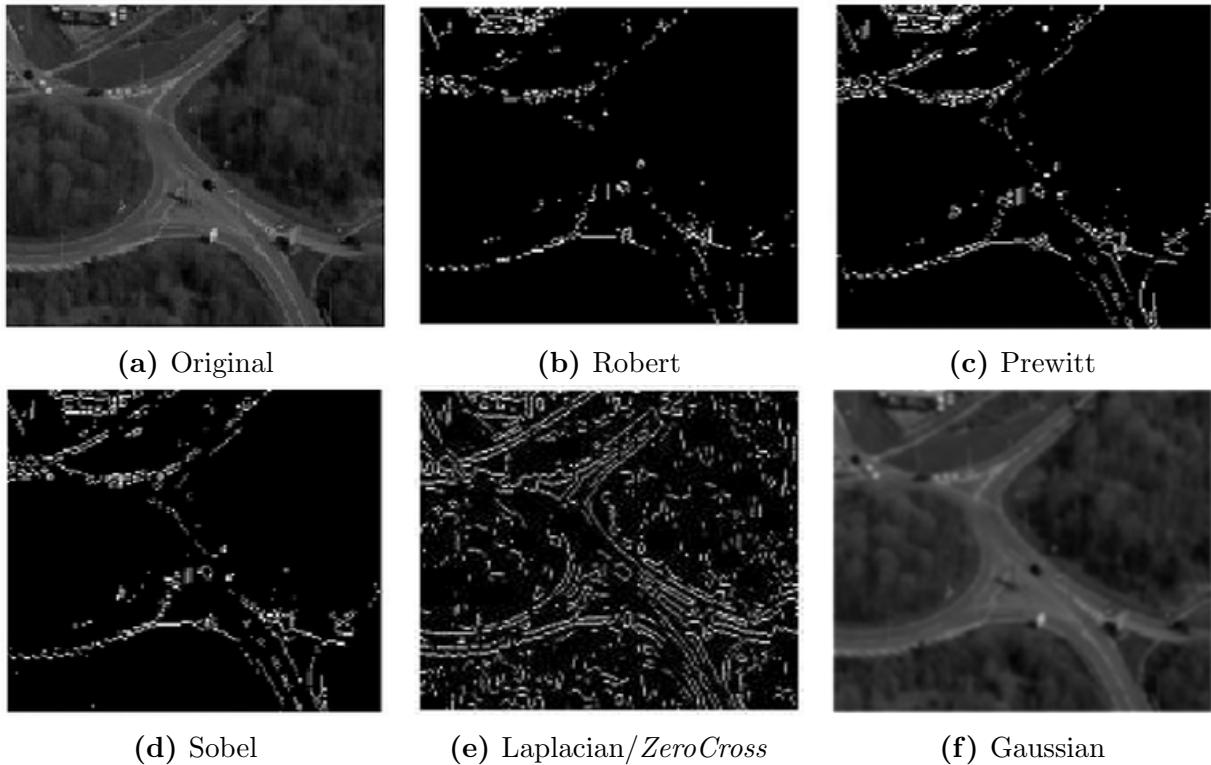
A continuación se mencionan los filtros más utilizados en el proyecto (un enfoque más amplio puede ser encontrado en [16, cap. 5, pág. 135-138], [23] y [24, cap. 3]).



(a) Imagen integral.

(b) Subáreas.

**Figura 3.6:** Subáreas dentro de una imagen integral. La extracción de los valores para las áreas dentro de una imagen integral obedece la misma fórmula mencionada (ec. 3.16).



**Figura 3.7:** Aplicación de diferentes filtros sobre una Imagen. 3.7a, para *robert*, *prewitt* y *sobel* se han juntado las convoluciones del gradiente horizontal y vertical. Fuente [23, pág. 34].

### 3.1.7.1. Robert

Uno de los filtros más viejos (figura 3.7b), utilizado para resaltar bordes, sus máscaras de convolución son:

$$G_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{y} \quad G_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad (3.21)$$

### 3.1.7.2. Prewitt

Al igual que *Robert*, es utilizado como extractor de bordes (figura 3.7c), funciona similar a *Sobel* y permite aproximar la derivada de una imagen. Prewitt estima el gradiente de las 8 direcciones posibles (dado una matriz de  $3 \times 3$ ). Los primeros 3 están dados por:

$$G_1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}, \quad G_3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}, \quad \dots \quad (3.22)$$

El resto puede ser obtenido rotando la matriz. Usualmente sólo son utilizadas las matrices  $G_1$  y  $G_3$  para la obtención de la derivada parcial horizontal  $I_x$  y vertical  $I_y$  respectivamente.

### 3.1.7.3. Sobel

Figura 3.7d. Al igual que *Prewitt*, se pueden estimar el gradiente de las 8 direcciones posibles. Está dado por el conjunto de filtros similares a:

$$G_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}, \quad G_3 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \quad \dots \quad (3.23)$$

De nuevo, a menudo sólo son utilizadas las máscaras  $G_1$  y  $G_3$  para la obtención de las derivadas horizontal y vertical.

### 3.1.7.4. Laplaciano

Como se mencionó anteriormente, a diferencia de *prewitt* y *sobel*, el *filtro laplaciano* es un operador que se estima mediante la aproximación de la segunda derivada. Utilizado igualmente para estimar bordes de una imagen (ver figura 3.7e). Puede ser computado como la suma de convoluciones de máscaras como:

$$G_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{y} \quad G_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (3.24)$$

### 3.1.7.5. Gaussiano

El efecto más obvio de un *filtro gaussiano* provoca un suavizado sobre la imagen, como si estuviera fuera de foco (figura 3.7f), a menudo es utilizado como previo a otro filtro para remover ruido. Con la función de la ecuación de la convolución en 3.8, para este filtro se deben cumplir las siguientes características [19]:

$$-a_{start} = a_{end} = -b_{start} = b_{end} \quad \text{y} \quad g(a, b) = \gamma(r) \quad (3.25)$$

donde

$$r = \sqrt{a^2 + b^2} \quad (3.26)$$

es la distancia desde el centro del kernel a sus elementos  $a, b$ , siendo  $\gamma$  una función gaussiana como:

$$\gamma(r) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}(\frac{r}{\sigma})^2} \quad (3.27)$$

o bien:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3.28)$$

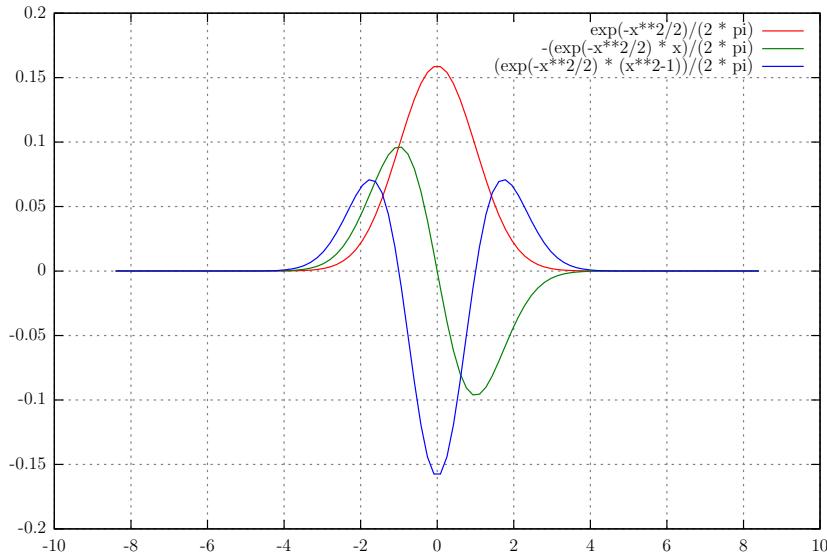
Una aproximación computable de un kernel se muestra en la siguiente ecuación:

$$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \quad (3.29)$$

### 3.1.7.6. Laplaciano de Gauss

El laplaciano por sí solo es muy sensible al ruido. Para optimizar este filtro una opción es utilizar el *Laplaciano de Gauss* (*Laplacian of Gaussian, LoG*).

En la figura 3.8 se observa una función gaussiana básica (rojo), su primera y segunda



**Figura 3.8:** Primera y Segunda derivada de Gauss. En el gráfico se muestran tres funciones, en rojo una función gaussiana con  $\sigma = 1$ , con verde su primera derivada y con azul su segunda derivada. Se puede apreciar el *cruce por cero* en la primera derivada y el *mexican hat* invertido en la segunda.

derivadas son mostradas en la función verde y azul respectivamente. En CV esta última es conocida como *laplaciano de gauss*. Cuando la función es proyectada en un espacio de 3D, el *LoG* asemeja a un sombrero mexicano, este tipo de funciones son conocidas como *mexican hat wavelet* [25, pág. 7] útiles en el procesamiento de señales. Si se observa la figura 3.9 se podrá apreciar la similitud a dicha prenda mexicana.

Debido a que la convolución es una operación asociativa, es posible convolucionar el filtro gaussiano con el laplaciano y con el híbrido filtrar la imagen original. De esta manera se ahorra carga computacional. La fórmula matemática que expresa dicha asociación, obtenida de [1, cap. 3, pág. 135], se muestra a continuación:

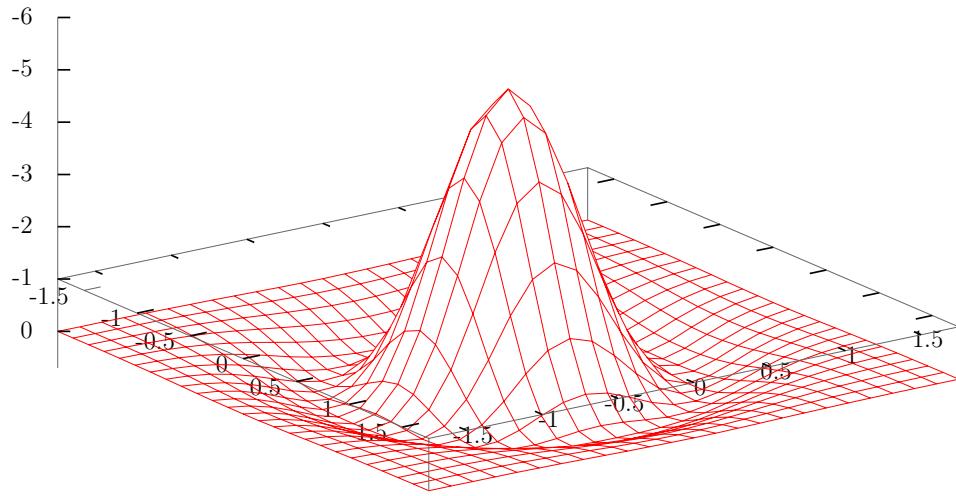
$$\nabla^2[G(x, y, \sigma) * f(x, y)] = [\nabla^2 G(x, y, \sigma)] * f(x, y) = LoG * f(x, y) \quad (3.30)$$

donde  $f(x, y)$  es la imagen,  $G(x, y, \sigma)$  una función gaussiana y  $LoG$  es el filtro dado por [1, cap. 3, pág. 104]:

$$LoG = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (3.31)$$

$$= \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} G(x, y, \sigma) \quad (3.32)$$

El gráfico de la figura 3.9 muestra esta función para  $\sigma = 1$  (nótese como el eje vertical se encuentra invertido, en la práctica esto no afecta su uso pudiéndose invertir la función sin problema).



**Figura 3.9:** Laplaciano de Gauss. Función también conocida como *sombrero mexicano*.

### 3.1.7.7. Diferencia Gaussiana

Como la obtención del *Laplaciano de Gauss* de una imagen es computacionalmente costoso, usualmente se opta por realizar aproximaciones por la *Diferencia Gaussiana (Difference of Gaussian, DoG)* cuyo cómputo es más sencillo y su similitud es sustancial. En la figura 3.10 se puede observar este caso.

La *diferencia gaussiana* es el proceso de obtener la diferencia de dos filtros gaussianos con diferente  $\sigma$ .

$$\nabla^2 G(x, y, \sigma) \approx G(x, y, \sigma_1) - G(x, y, \sigma_2) \quad (3.33)$$

$$LoG \approx DoG \quad (3.34)$$

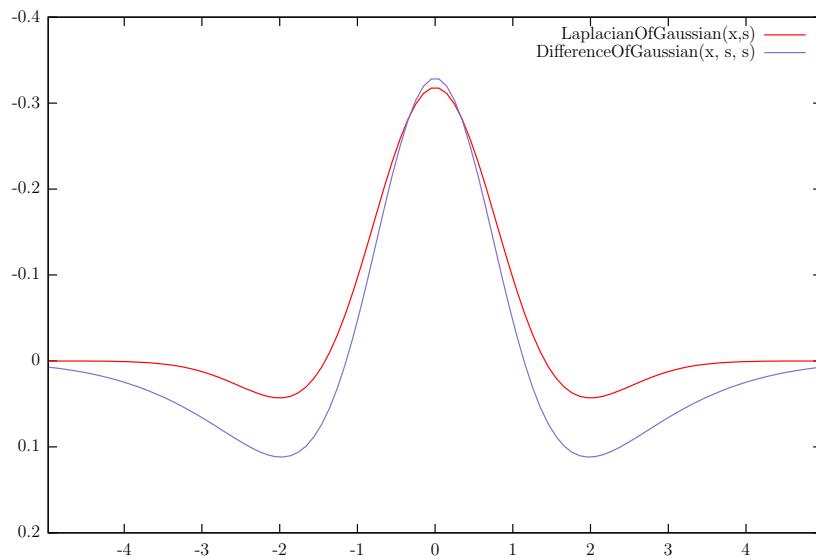
Como son operadores convolucionados, al igual que el *LoG*, comparten propiedades asociativas, lo que permite la agrupación de los gaussianos para luego aplicar el híbrido a la imagen original:

$$G(x, y, \sigma_1) * I(x, y) - G(x, y, \sigma_2) * I(x, y) = [G(x, y, \sigma_1) - G(x, y, \sigma_2)] * I(x, y) \quad (3.35)$$

$$= DoG * I(x, y) \quad (3.36)$$

una mejor aproximación de un *LoG* es obtenida cuando, dada la ecuación 3.33,  $\sigma_1$  y  $\sigma_2$  tienden a (3.37):

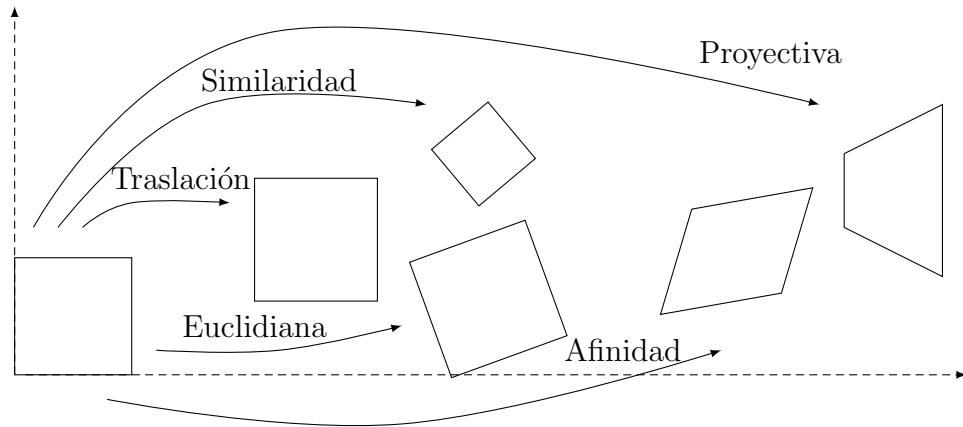
$$\sigma_1 = \frac{\sigma}{\sqrt{2}} \quad y \quad \sigma_2 = \sigma\sqrt{2} \quad (3.37)$$



**Figura 3.10:** Aproximación por Diferencia Gaussiana. En rojo, *Laplaciano de gauss*, azul *Diferencia gaussiana*. Nótese que el eje vertical se encuentra invertido.

## 3.2. Transformaciones en imágenes

Las operaciones básicas que se pueden aplicar sobre las imágenes usualmente involucran dos o tres dimensiones, en la figura 3.11 se muestran las transformaciones usuales de 2D que se aplican en ISB.



**Figura 3.11:** Transformaciones básicas de dos dimensiones. Fuente [1, cap. 2, pág. 33].

### 3.2.1. Puntos 2D

Un punto de 2D representa las coordenadas de pixeles dentro de una imagen, es denotado como un vector:

$$\mathbf{x} = [x, y]^T \in \mathbb{R}^2 \quad (3.38)$$

Éstos a su vez, pueden ser representados usando coordenadas homogéneas,  $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$  y  $\mathcal{P}^2 = \mathbb{R}^3 - (0, 0, 0)^T$  conocido como espacio proyectivo 2D.

Un vector homogéneo  $\tilde{\mathbf{x}}$  puede ser convertido de vuelta a un vector no homogéneo  $\mathbf{x}$  dividiendo entre  $\tilde{w}$ :

$$\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \quad (3.39)$$

$$= \tilde{w}(x, y, 1) \quad (3.40)$$

$$= \tilde{w}\bar{\mathbf{x}} \quad (3.41)$$

donde  $\bar{\mathbf{x}} = (x, y, 1)$  es el vector aumentado.

### 3.2.2. Traslación

Una translación es un movimiento vectorial aplicado sobre un punto 2D (o un conjunto de puntos). Puede ser descrito como  $\mathbf{x}' = \mathbf{x} + \mathbf{t}$  o bien como:

$$\mathbf{x}' = [I \quad t]\bar{\mathbf{x}} \quad (3.42)$$

donde  $I$  es una matriz identidad de  $(2 \times 2)$  ó:

$$\bar{x}' = \begin{bmatrix} I & t \\ 0^T & 1 \end{bmatrix} \bar{x} \quad (3.43)$$

donde  $0$  es el vector cero.

### 3.2.3. Rotación + Traslación

A esta transformación también se le conoce como *movimiento de cuerpo rígido en 2D* o la *transformación euclídea de 2D* (ya que las distancias euclidianas se mantienen). Puede ser escrita como  $\mathbf{x}' = R\mathbf{x} + \mathbf{t}$  o bien:

$$\mathbf{x}' = [R \quad t] \bar{x} \quad (3.44)$$

donde  $R$  es la matriz de rotación:

$$R = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad (3.45)$$

donde  $RR^t = I$  y  $|R| = 1$ .

### 3.2.4. Escalación + Rotación

Conocida como *similaridad*, es expresada como  $\mathbf{x}' = sR\mathbf{x} + \mathbf{t}$  donde  $s$  es un factor arbitrario de escalación. También puede ser escrita como:

$$\mathbf{x}' = [sR \quad t] \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x} \quad (3.46)$$

### 3.2.5. Afinidad

En una transformación *afín*, las líneas paralelas entre sí se mantienen, puede ser descrita como  $\mathbf{x}' = A\mathbf{x}$ , donde  $A$  es una matriz de  $(2 \times 3)$ , por ejemplo:

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{x} \quad (3.47)$$

### 3.2.6. Proyección

En una proyección, se conservan las líneas rectas. También conocida como *transformación proyectiva* u *homografía* es descrita como:

$$\tilde{\mathbf{x}}' = \tilde{H}\tilde{\mathbf{x}} \quad (3.48)$$

donde  $\tilde{H}$  es una matriz arbitraria de  $3 \times 3$  que opera en coordenadas homogéneas. Para obtener coordenadas no-homogéneas se debe normalizar  $\tilde{\mathbf{x}}'$ , por ejemplo:

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}x} \quad y \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}x} \quad (3.49)$$

### 3.2.7. Jerarquía de las transformaciones 2D

La dificultad de una transformación a otra puede permitir clasificar a estas en una jerarquía medida a partir de los Grados de Libertad (*DoL*) requeridos para aplicar dicha transformación. En la tabla 3.1 se observa la jerarquía.

La forma más sencilla de interpretarlas es pensando en ellas como una matriz de  $3 \times 3$  (posiblemente restringida) operando en coordenadas homogéneas de vectores 2D [1, cap. 2, pág. 34], [26, cap. 2, pág. 9].

**Tabla 3.1:** Jerarquía de transformaciones en 2D. Cada transformación preserva la propiedad lista. Las matrices de  $2 \times 3$  pueden ser extendidas con un renglón  $[0^T \ 1]$  para ser de  $3 \times 3$  y poder realizar transformaciones con coordenadas homogéneas. Fuente: [1, cap. 2, pág. 35].

Transformación	Matriz	Preserva	<i>DoL</i>
Traslación	$[I t]_{2 \times 3}$	Orientación	2
Euclíadiana	$[R t]_{2 \times 3}$	Distancias	3
Similaridad	$[sR t]_{2 \times 3}$	Ángulos	4
Afín	$[A]_{2 \times 3}$	Paralelismo	6
Proyectiva	$[\tilde{H}]_{3 \times 3}$	Rectitud	8

# Capítulo 4

## Desarrollo

Esta sección aborda en sí el desarrollo del sistema ISB, encontrándose dividido en una serie de secciones que involucran la metodología seguida por estos algoritmos, de cualquier manera, antes de comenzar se dará un resumen breve del contenido.

El proceso de ISB, o en otras palabras, la creación de imágenes panorámicas es un área que ha tenido un amplio estudio. Recordando la imagen 1.3 de la sección 1, se puede recapitular que, dentro de CV, esta clase de algoritmos constituyen dos de los tres campos que cubre el área (de los problemas a nivel imagen, geométrico y fotométrico, ISB queda en un punto medio de los primeros dos), definiendo así, el problema desde estos dos enfoques:

- Desde un punto de vista de la imagen, el problema principal consiste en determinar el modelo matemático apropiado que permita relacionar el sistema de píxeles de una imagen en el sistema de píxeles de otra [1, cap. 9, pág. 377].
- Desde un punto de vista geométrico, el objetivo es estimar la matriz de la cámara de  $3 \times 3$  u homografía, de cada una de las imágenes [5].

Básicamente, un proceso para unificar imágenes divide su tarea en diferentes etapas. Basado en los trabajos de [5], [9], [27], [28], [7], [12], la metodología básica en la mezcla de imágenes se presenta en los siguientes pasos:

- **Extracción de Características (*Feature Extraction*)**. Siendo el primer paso, en esta etapa se extraen elementos (llamados *feature*) existentes en las diferentes imágenes. La idea es la de obtener puntos únicos que contengan información irrepetible como la posición o el ángulo, conocidos como *keypoint* los cuales sirven de base para la siguiente etapa. El criterio de extracción puede diferir, los métodos más

usados son SIFT [7], SURF [8], ORB [12], entre otros, de cualquier manera, en la sección 4.1 se detalla la teoría interna general dentro de estos algoritmos.

- **Empate de Características (*Feature Matching*)**. Una vez completada la extracción, se procede a emparentar las imágenes. Usualmente se realiza con algoritmos basados en el vecino más cercano (*nearest-neighborhood*). Más acerca de esta etapa puede ser encontrado en la sección 4.2.
- **Estimación de Homografía (*Homography Estimation*)**. En esta etapa se estima la relación existente entre una imagen y otra (llamada homografía u *Homography*) mediante el uso de algoritmos como RANSAC [29]. Al ser obtenida, esta homography permite realizar la transformación requerida (como *traslación*, *rotación*, *escalación*, *proyección*, etcétera) para “encajar” el conjunto de imágenes a modo de rompecabezas. Esto es explicado a fondo en la sección 4.3.
- **Composición (*Compositing*)**. Es la etapa final. Aquí se busca mezclar las imágenes. Si en el proceso anterior existieron inconsistencias tras el traslape de imágenes (bastante usual) se intenta buscar la manera óptima de fusionarlas. Este proceso será discutido en la sección 4.4.

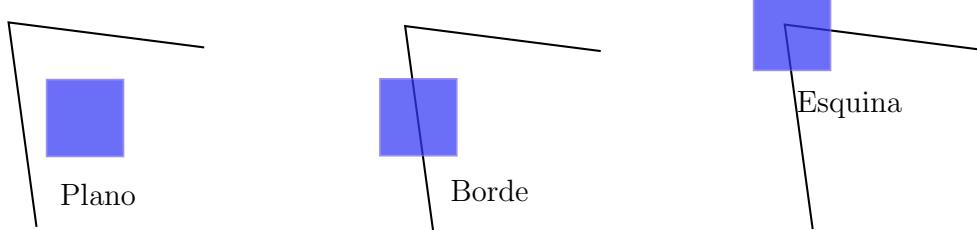
## 4.1. Extracción de Características

Siendo la primera etapa, la extracción de características representa el proceso de conseguir locaciones (a veces llamados *feature*, *keypoint*, *puntos de interés*, *corners*, *edges*, etcétera dependiendo del algoritmo) dentro de una imagen que puedan ser diferenciables y localizables para uso posterior. En CV constituye uno de los pasos fundamentales y se emplean generalmente para alineamiento de imágenes, o bien para correspondencia geométrica. Existen varios métodos para extracción de características, a continuación se presentan los más frecuentes:

### 4.1.1. Harris Corner Detector

El algoritmo *Harris Corner Detector* (o *Harris & Stephens Corner Detector*), desarrollado por Chris Harris y Mike Stephens en 1988 [30] representa uno de los algoritmos de detección de características más simples y conocidos, siendo el sucesor del algoritmo *Moravec Corner Detector*.

*Harris Corner Detector* ofrece un enfoque matemático que permite distinguir básicamente tres tipos de elementos dentro de una imagen: planos, bordes y esquinas (ver figura 4.1), lográndolo mediante la derivada parcial vertical y horizontal para obtener la dirección de los gradientes de las imágenes (este proceso fue estudiado en la sección 3.1.4).



**Figura 4.1:** Elementos distinguidos en Harris Corner Detector. Básicamente tres: planos (sin cambios en ningún eje), bordes (cambio en un eje) y esquinas (cambios en todos los ejes).

Una de las ventajas que ofrece este algoritmo frente a su predecesor es la robustez en la rotación. En *Moravec Detector* se realiza la detección de los elementos en rotaciones de  $45^\circ$ , esto descarta los elementos que no se encuentren en esa posición durante la ejecución del algoritmo. Para resolver este problema, *Harris Detector* realiza una de-rotación de la ecuación de *Moravec* que sigue el siguiente procedimiento [31]:

Dado un cambio  $(\Delta x, \Delta y)$  y un punto  $(x, y)$ , la función de autocorrelación estará definida por:

$$c(x, y, \Delta x, \Delta y) = \sum_{(u, v) \in W(u, v)} w(u, v)[I(u, v) - I(u + \Delta x, v + \Delta y)]^2 \quad (4.1)$$

donde  $I$  denota la imagen,  $W$  es la ventana centrada en el punto  $(x, y)$  y  $w(u, v)$  es una función gaussiana. El cambio en la imagen  $I(u + \Delta x, v + \Delta y)$  es aproximado mediante una expansión de Taylor truncada a sus términos de primer orden:

$$I(u + \Delta x, v + \Delta y) \approx I(u, v) + [I_x(u, v)I_y(u, v)] \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \quad (4.2)$$

siendo  $I_x$  e  $I_y$  la derivada parcial horizontal y vertical respectivamente, obtenidas fácilmente mediante el uso del filtro Prewitt (ver sección 3.1.7.2).

Sustituyendo la aproximación (ec. 4.2) en la ecuación 4.1 y reduciendo llegamos a la representación bilineal:

$$c(x, y, \Delta x, \Delta y) = [\Delta x \quad \Delta y] Q(x, y) \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \quad (4.3)$$

donde  $Q(x, y)$  representa una matriz simétrica de la forma:

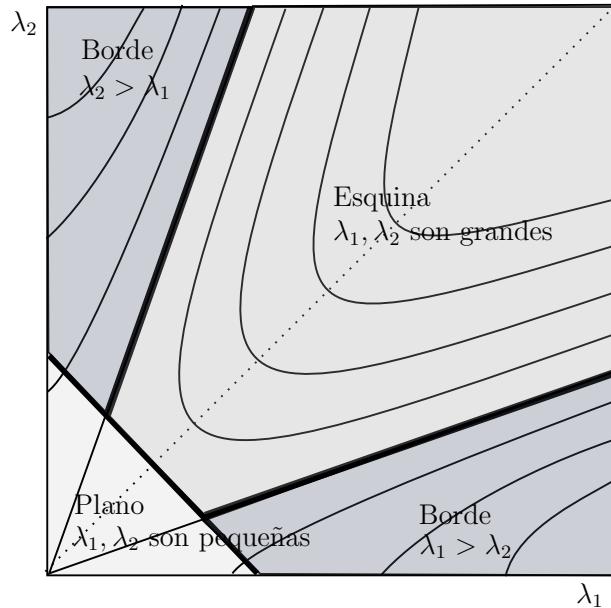
$$Q(x, y) = \begin{bmatrix} A & C \\ C & B \end{bmatrix} = \begin{bmatrix} \sum_W I_x^2 & \sum_W I_x I_y \\ \sum_W I_x I_y & \sum_W I_y^2 \end{bmatrix} \quad (4.4)$$

siendo  $\sum_W$  una abreviación de:

$$\sum_W = \sum_{u,v \in W(x,y)} w(u, v) \quad (4.5)$$

A través de la matriz simétrica  $Q$  en ec. 4.4 se pueden obtener los eigenvalores  $\lambda_1, \lambda_2$  mediante su descomposición espectral (*eigenvalue decomposition* ó *eigendecomposition*). Teniendo estos valores es posible clasificar el pixel a través de un esquema como el que muestra la figura 4.2, existiendo tres casos particulares [30]:

- **Si ambas  $\lambda$  son pequeñas.** Significa que no existió mucha variación dentro de la función ventana, por lo tanto se trata de una región plana.
- **Si una de las  $\lambda$  es mayor que la otra.** Esto indica que dentro de la función ventana se localiza un borde.
- **Si ambas  $\lambda$  son grandes.** Indica la localización de una esquina, siendo ésta la situación esperada.



**Figura 4.2:** Clasificación basado en eigenvalores para el algoritmo *Harris Corner Detector*. Fuente [30].

De cualquier manera, es posible clasificar un pixel sin computar los eigenvalores realmente a través de:

$$R(x, y) = \text{Det}(Q) - k(\text{Tr}(Q))^2 \quad (4.6)$$

donde  $\text{Det}(Q)$  y  $\text{Tr}(Q)$  son el determinante y la traza de  $Q$  respectivamente, dados por:

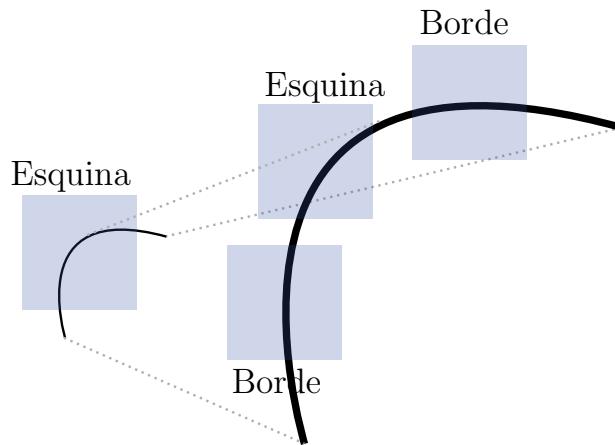
$$\text{Tr}(Q) = A + B \approx \lambda_1 + \lambda_2 \quad (4.7)$$

$$\text{Det}(Q) = AB - C^2 \approx \lambda_1 \lambda_2 \quad (4.8)$$

y  $R(x, y)$  permite medir la respuesta del algoritmo que estima qué tanta esquina (*corner-ness*) existe, siendo  $k$  una constante que permite establecer el umbral de detección donde 0.04 a 0.5 ofrece buenos resultados según reporta [27, cap. 4, pág. 33].

El número de esquinas detectadas en una imagen estará dado por el umbral establecido, al disminuir éste aumentará el otro y viceversa.

Una de las desventajas que presenta este algoritmo frente a los más nuevos (como SIFT y ORB) es su vulnerabilidad frente a la escalación de una imagen tal como se muestra en la figura 4.3.



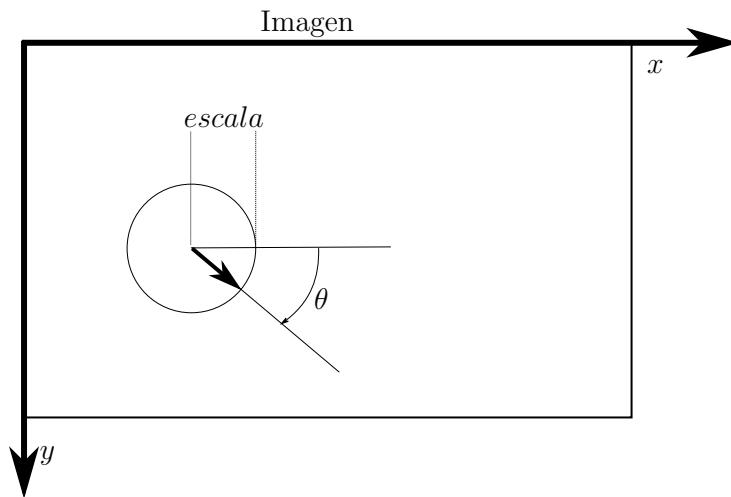
**Figura 4.3:** Vulnerabilidad de Harris Corner Detector frente a la escalación de una imagen.

#### 4.1.2. SIFT

Frente a la necesidad de un método dentro de CV que permita detectar características de una imagen que sean invariantes a la escalación hizo que David G. Lowe en 1999 desarrollara a SIFT (*Scale-Invariant Feature Transform*) [6], un algoritmo que ofrece cierta

robustez a problemas de translación, rotación, **escalación**, proyección y una resistencia parcial frente a problemas de iluminación, ruido y distorsión.

A diferencia de *Harris Corner Detector*, puede decirse que SIFT es un algoritmo que detecta **manchas** (*blob*, *regiones*, *gotas*) a diferentes escalas dentro de una imagen. Su invariabilidad frente a la escalación (o más bien, su covariación) lo logra mediante la localización de keypoints dentro de los máximos y mínimos de un espacio de *Diferencia de Gaussianas (DoG)*. Estas características o *keypoints* son definidos como círculos dentro de la imagen que cuentan con un radio y una rotación, los cuales definen la escala y la orientación respectivamente. En la figura 4.4 se aprecian estas características.



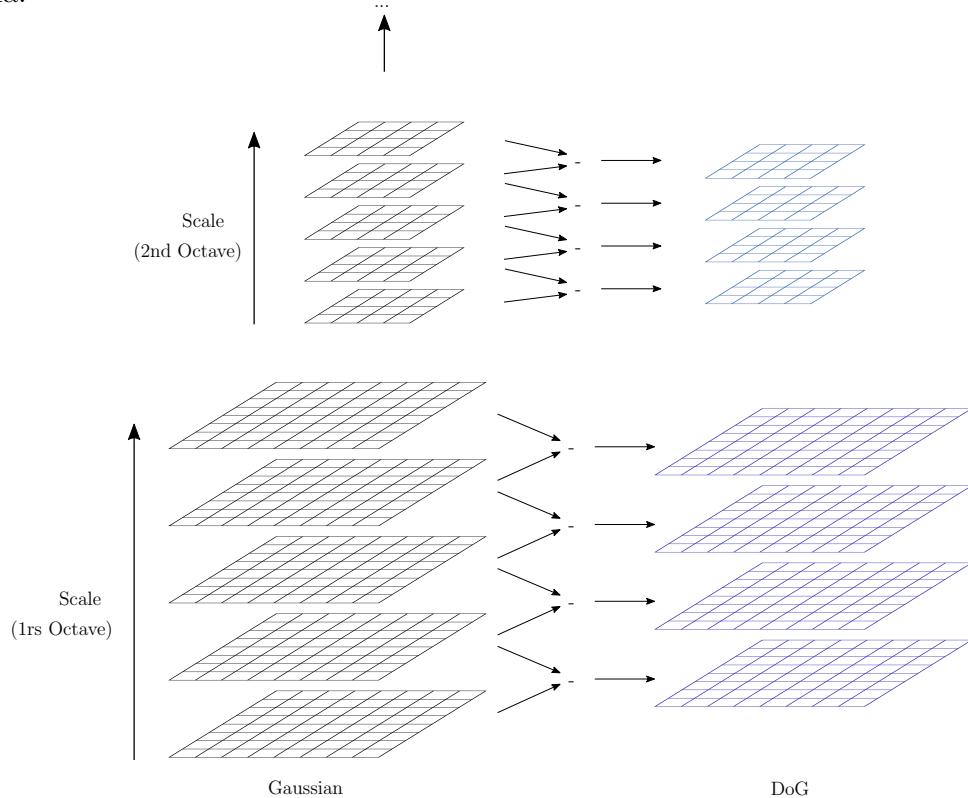
**Figura 4.4:** Características de un Keypoint en SIFT. Un keypoint dentro de *SIFT* está definido como un círculo con una orientación, caracterizado por cuatro propiedades: su localización en el eje *x* y *y*, su escala y su orientación. Fuente [32]

Para la obtención de los *keypoints* y sus correspondientes descriptores en *SIFT*, se sigue la siguiente metodología:

- Crear el Gaussian Scale-space para obtener la pirámide *DoG*.
- Detección del local extrema Scale-space.
- Localización exacta de los Keypoints.
- Asignación de orientación.
- Creación de los descriptores.

#### 4.1.2.1. Gaussian Scale-space y pirámide DoG

La generación del Gaussian Scale-space se logra mediante la convolución incremental de la imagen original  $I(x, y)$  con una función Gaussiana para producir una pila de imágenes  $L(x, y, k\sigma)$  separadas por un factor constante  $k$ , tal como muestra el lado izquierdo en la figura 4.5. Cada octava del espacio está dividida de tal manera que se cumpla  $k = 2^{1/s}$  contando con  $s + 3$  imágenes para permitir completar una octava en la pila de gaussianas [7], por conveniencia, cada una de las imágenes dentro del Gaussian Scale-space representa una escala.



**Figura 4.5:** Gaussian Scale Space de SIFT. Para cada octava del *scale space*, la imagen original es repetidamente convolucionada con Gaussianos para producir el conjunto de imágenes de la izquierda. Con las imágenes adyacentes se obtienen la *DoG* mediante su sustracción, generando el conjunto de la derecha. Después de cada octava, la imagen es reducida con un factor de 2, y el proceso se repite. Fuente [7].

Con la octava completa, el primer elemento de la siguiente es obtenido duplicando el valor inicial de  $\sigma$  y submuestreando el tamaño de la última imagen de la pila a  $1/4$  de la original, una forma de realizar ésto es tomando las columnas y renglones pares, o bien promediando cada cuatro pixeles.

La pirámide *DoG* se obtiene mediante la sustracción de las imágenes adyacentes de cada

octava como se muestra a la derecha en la figura 4.5 generando un espacio de imágenes  $D(x, y, \sigma)$  apiladas. Matemáticamente esto está representado por:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \quad (4.9)$$

$$= L(x, y, k\sigma) - L(x, y, \sigma) \quad (4.10)$$

siendo  $G(x, y, \sigma)$  una función Gaussiana como la vista en la ecuación 3.28.

El uso del operador *DoG* tiene su base en su similitud con *LoG*, la utilidad de este último se presenta frente a su isotropía (propiedad necesaria en el algoritmo *SIFT*). La relación existente entre *DoG* y *LoG* se menciona en los puntos 3.1.7.6 y 3.1.7.7, respectivamente; además, en el mismo artículo, Lowe [7] expone la similitud entre estos dos operadores concluyendo que una mejor aproximación se logra cuando:

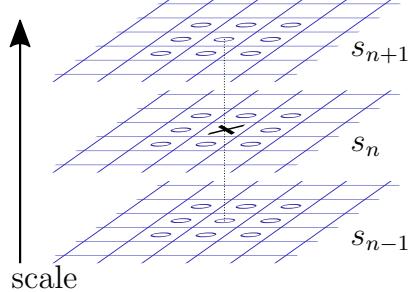
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G \quad (4.11)$$

siendo  $G(x, y, k\sigma) - G(x, y, \sigma)$  el operador *DoG* y  $\sigma^2 \nabla^2 G$  el operador *LoG*.

#### 4.1.2.2. Detección del local extrema Scale-space

Con la pirámide *DoG*, la detección de los puntos extremos se realiza mediante la localización de los valores máximos y mínimos dentro de un conjunto de vecinos de cada pixel para las imágenes  $D(x, y, \sigma)$ .

Cada punto es comparado con sus 8 vecinos de la imagen actual junto con los 9 de la imagen  $D(x, y, \sigma)$  adyacente superior e inferior tal como se observa en la figura 4.6; los puntos localizados son llamados keypoint candidatos.



**Figura 4.6:** Extremos de imágenes adyacentes DoG dentro de SIFT. Los máximos y mínimos son detectados comparando los 26 vecinos de una región de  $3 \times 3$  entre las imágenes adyacentes del pixel actual. Fuente [7].

#### 4.1.2.3. Localización exacta de los Keypoints

Una vez localizados los keypoints candidatos, el siguiente paso es realizar un ajuste más detallado que permita determinar su posición y escala real, así como descartar aquellos con poco contraste o bien, que se encuentran localizados como un borde.

En el primer caso, una manera para encontrar a los máximos y mínimos reales, tal como se muestra en la figura 4.7, es interpolar los puntos sobre las imágenes  $D(x, y, \sigma)$  utilizando una expansión de Taylor truncada a sus términos cuadráticos:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}} \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2} \mathbf{x} \quad (4.12)$$

siendo  $\mathbf{x} = (x, y, \sigma)^T$  (note la diferencia entre  $\mathbf{x}$  y  $x$ ). Al derivar respecto de  $\mathbf{x}$  e igual a 0, se puede obtener el valor real:

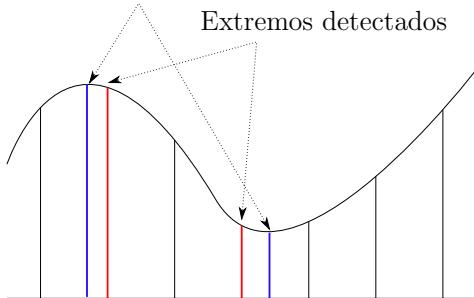
$$\hat{\mathbf{x}} = -\frac{\partial D^{-1}}{\partial \mathbf{x}^2} \frac{\partial D}{\partial \mathbf{x}} \quad (4.13)$$

Sustituyendo la ecuación 4.13 en 4.12 se obtiene la función del valor en el extremo  $D(\hat{\mathbf{x}})$ , la cual resulta útil para rechazar los extremos de bajo contraste:

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \frac{\partial D^T}{\partial \mathbf{x}} \hat{\mathbf{x}} \quad (4.14)$$

donde se toma la magnitud de intensidad y se compara con un valor dado; en [7] si  $|D(\hat{\mathbf{x}})| < 0.03$  entonces ese pixel es descartado.

Extremos reales



**Figura 4.7:** Diferencia entre extremos detectados y reales en SIFT. En repetidas ocasiones, los extremos detectados no representarán a los verdaderos, para localizar éstos, hace falta interpolarlos con una expansión de Taylor.

La eliminación de aquellos keypoints mal clasificados se realiza a través de una matriz hessiana  $\mathcal{H}$  con una clasificación que es similar a como lo hace *Harris Corner Detector*, donde  $\mathcal{H}$  está definida como:

$$\mathcal{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (4.15)$$

Los eigenvalores de la matriz  $\mathcal{H}$  ayudan a clasificar cada pixel, así mismo, su cómputo puede ser reemplazado por la relación existente con la matriz, de tal manera que:

$$Tr(\mathcal{H}) = D_{xx} + D_{yy} = \alpha + \beta \quad (4.16)$$

$$Det(\mathcal{H}) = D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta \quad (4.17)$$

siendo  $\alpha$  y  $\beta$  los eigenvalores con mayor y menor magnitud. Para descartar elipses (o esquinas), se da un margen  $r$  cumpliendo con la condición:

$$\frac{Tr(\mathcal{H})^2}{Det(\mathcal{H})} < \frac{(r+1)^2}{r} \quad (4.18)$$

siendo  $Tr(\mathcal{H})$  y  $Det(\mathcal{H})$  la traza y el determinante de la matriz  $\mathcal{H}$ . Si la desigualdad falla, entonces el keypoint es descartado, Lowe estima que un valor de  $r = 10$  es un buen umbral para eliminar aquellos inservibles.

#### 4.1.2.4. Asignación de orientación

Llegada a esta etapa, los keypoints localizados serán los utilizados como aquellos definitivos de la imagen, los últimos pasos requieren darle una orientación y definir sus descriptores.

Para definir la magnitud y orientación de un keypoint, se utiliza el par de funciones:

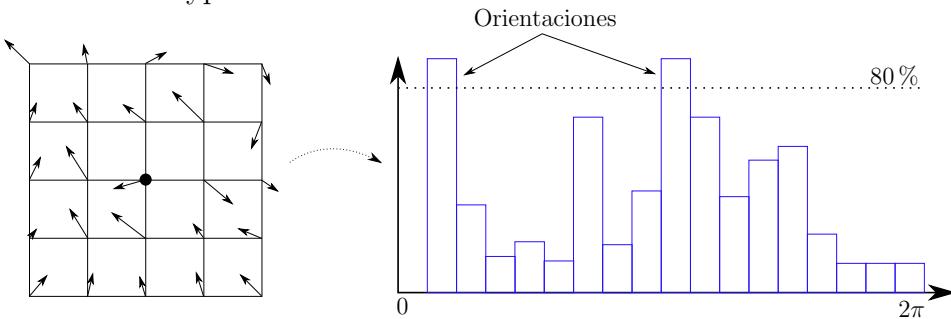
$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (4.19)$$

$$\theta(x, y) = \tan^{-1} \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \quad (4.20)$$

donde  $L(x, y)$  representa la imagen de la pila gaussiana que se encuentra más próxima al keypoint actual.

Para ello, se toma la vecindad de pixeles dentro del keypoint localizado, y se forma un histograma de 36 divisiones (cubriendo los  $360^\circ$ ) que tendrá las orientaciones de cada uno de estos puntos (figura 4.8). Cada vecino será agregado al histograma con el peso de la magnitud de su gradiente cubriendo un área del tamaño de una ventana gaussiana circular con un  $\sigma$  que es 1.5 mayor que la escala respectiva al keypoint.

Si existen dos orientaciones superiores al 80 % dentro del histograma, se toman ambas, siendo cada una un keypoint diferente.

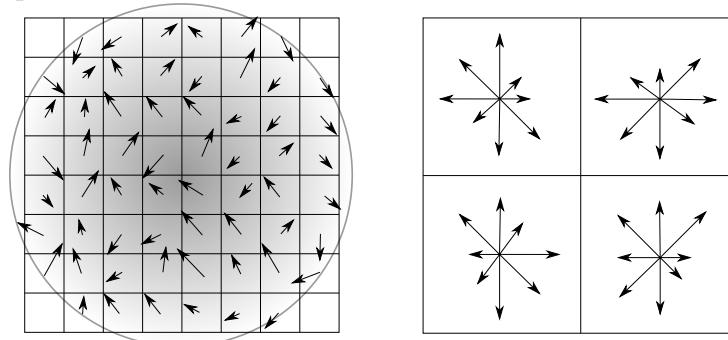


**Figura 4.8:** Detección de orientación de un Keypoint en SIFT. Aquella división que sobrepase un límite del 80 % serán la orientación seleccionada para el keypoint, si más de una sobrepasa el límite, entonces, se crea un keypoint para esa orientación.

#### 4.1.2.5. Creación del descriptor

Un descriptor tiene que cumplir con la propiedad de ser altamente distinguible mientras es lo suficientemente invariante a diferentes cuestiones, como cambios de luminosidad o proyecciones.

El descriptor se obtiene a partir de una vecindad de  $16 \times 16$  alrededor del keypoint, cada elemento es pesado por una ventana gaussiana (véase el círculo de la figura 4.9), éste es dividido en 16 bloques de  $4 \times 4$ , donde se creará un histograma de 8 divisiones. Esto generará un descriptor de 128 valores.



**Figura 4.9:** Representación esquemática de un descriptor SIFT. Mientras esta figura muestra una vecindad de  $8 \times 8$  y un descriptor de  $2 \times 2$ , la implementación real de Lowe involucra una vecindad de  $16 \times 16$  y un arreglo de  $4 \times 4$  de histogramas de 8 divisiones. Fuente: [1, cap. 4, pág. 198]

### 4.1.3. SURF

SIFT ha demostrado ser un algoritmo robusto frente a sus predecesores, pero las limitaciones computacionales (ya sea como equipos con escasa memoria o capacidad de procesamiento, o equipos móviles) han hecho que surjan mejoras que busquen acelerar lo que SIFT hace muy bien. Uno de estos casos es SURF (*Speeded Up Robust Features*), desarrollado por Herbert Bay, Tinne Tuytelaars y Luc Van Gool alrededor de 7 años después de la primera aparición de su predecesor en el 2006 [8].

En SIFT, Lowe aproxima el *LoG* a través de una *DoG* para encontrar el Scale-Space. SURF realiza la misma tarea, pero aproximando el *LoG* con *Box-Filters*, la metodología para obtener los *puntos de interés* (el equivalente de los keypoints en *SIFT*) y sus descriptores se menciona a continuación:

- Detección Fast-Hessian.
- Representación del Scale-Space.
- Localización del punto de interés.
- Asignación de orientación.
- Extracción de descriptores.

#### 4.1.3.1. Detección Fast-Hessian

Como se ha visto en *SIFT*, *SURF* también hace uso de una matriz hessiana que le ayuda con la detección de los puntos de interés, este aspecto es conocido como *Fast-Hessian detector*, su descripción se menciona a continuación.

Dado un punto  $\mathbf{x} = (x, y)$  en una imagen  $I$ , la matriz hessiana  $\mathcal{H}(\mathbf{x}, \sigma)$  para este punto estará definida como:

$$\mathcal{H}(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix} \quad (4.21)$$

siendo  $L$  la convolución de un gaussiano derivado de segundo orden con una imagen  $I$  en el punto  $\mathbf{x} = (x, y)$ :

$$L_{xx}(x, \sigma) = \frac{\partial^2}{\partial^2 x} g(\sigma) * I(x) \quad (4.22)$$

$$L_{yy}(x, \sigma) = \frac{\partial^2}{\partial^2 y} g(\sigma) * I(x) \quad (4.23)$$

$$L_{xy}(x, \sigma) = \frac{\partial^2}{\partial x \partial y} g(\sigma) * I(x) \quad (4.24)$$

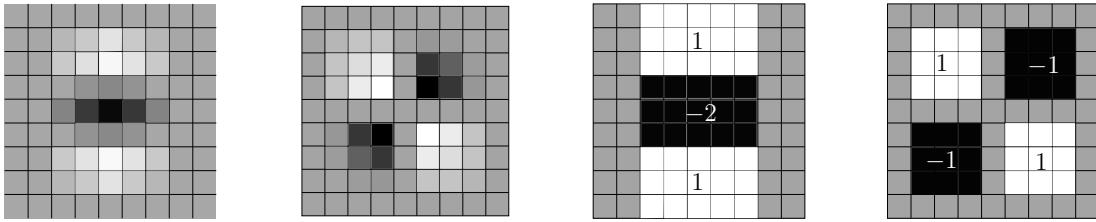
Con los eigenvalores de la matriz hessiana, es posible medir la respuesta del pixel, surgiendo 3 casos particulares:

- Si ambos eigenvalores son positivos, entonces es un mínimo local.
- Si ambos eigenvalores son negativos, entonces es un máximo local.
- Si ambos eigenvalores tienen valores diferentes, entonces hay un punto de equilibrio.

De esta manera, si el producto de los eigenvalores es positivo, significa que ambos, o eran positivos, o eran negativos y nos encontramos en un punto extremo. Esta relación se encuentra en el determinante de  $\mathcal{H}$  en el punto  $x = (x, y)$ , pero el cálculo de éste puede ser aceleradamente aproximado tal como se observa a continuación.

Aplicar un filtro gaussiano (o en este caso, su segunda derivada) a una imagen implica forzosamente una discretización y un recorte, llevando a una pérdida de repetibilidad (ver los dos cuadros izquierdos de la figura 4.10). Ésto lo compensa Bay et al. aproximando la matriz Hessiana con *box-filters* (como los dos filtros de la derecha de la misma figura) pudiendo ser evaluados a muy bajo costo computacional con el uso de una imagen integral, (este concepto se puede repasar en la sección 3.1.6 con la ecuación 3.15).

Al emplear la imagen integral, Bay et al. sugiere iniciar con los filtros de  $9 \times 9$  (mencionados en 4.10) que representan una aproximación de la segunda derivada parcial de un gaussiano con  $\sigma = 1.2$  consiguiendo así  $D_{xx}$ ,  $D_{yy}$  y  $D_{xy}$ , siendo los equivalentes aproximados de  $L_{xx}$ ,  $L_{yy}$  y  $L_{xy}$  respectivamente.



**Figura 4.10:** Discretización y aproximación de la 2<sup>da</sup> derivada gaussiana en SURF. De izquierda a derecha: La segunda derivada parcial gaussiana discretizada y cortada para  $y$  ( $L_{yy}$ ) y  $xy$  ( $L_{xy}$ ); aproximación de la segunda derivada gaussiana parcial para  $y$  ( $D_{yy}$ ) y  $xy$  ( $D_{xy}$ ). El gris es igual a cero. Fuente: [9].

De esta manera, el valor en el punto  $x = (x, y, \sigma)$  puede ser calculado con el determinante aproximado mediante la siguiente expresión:

$$\text{Det}(\mathcal{H}_{\text{approx}}) = D_{xx}D_{yy} - (wD_{xy})^2 \quad (4.25)$$

siendo  $w$  un peso que permite balancear la expresión para el determinante de  $\mathcal{H}$ . Teóricamente, este peso cambia dependiendo de la escala presente, pero Bay et. al. lo mantiene como una constante, afirmando que no representa una variable de un impacto considerable ([8] y [9]).

#### 4.1.3.2. Representación del Scale-Space

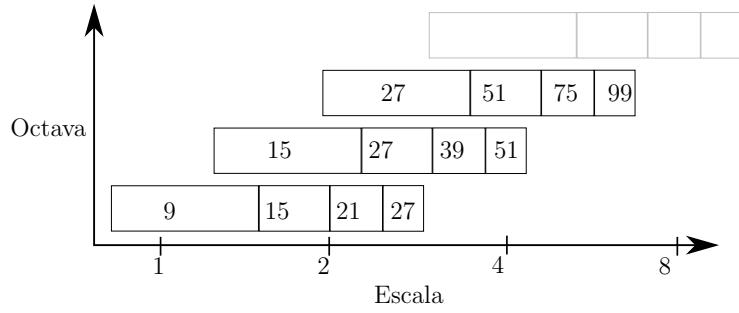
Tal como lo hace Lowe con *SIFT* con la pirámide *DoG*, *SURF* realiza una extracción de puntos de interés a diferentes escalas; aunque debido al uso de *box-filters* e imágenes integrales, *SURF* no tiene que esperar entre la aplicación de cada filtro, ya que es posible aplicar diferentes *box-filters* a la misma imagen, incluso, en paralelo.

El *Scale-Space* de *SURF* se encuentra dividido en octavas (similar a *SIFT*), siendo cada una de éstas una serie de mapas filtrados obtenidos a través de la convolución de la imagen original de entrada con un filtro de tamaño variable. Lo que lo diferencia frente a su predecesor es que la analogía de la “pirámide” se encuentra invertida.

Mientras que en *SIFT* se reduce gradualmente el tamaño de la imagen original, en *SURF* se crean filtros de mayor tamaño pero la imagen permanece igual. Ésto le ofrece cierta ventaja de paralelismo ya que cada filtrado es independiente entre sí.

Para la construcción del Scale-Space, el inicio de la octava tiene como base al filtro de  $9 \times 9$  como la primera escala, seguidas de los filtros de  $15 \times 15$ ,  $21 \times 21$  y  $27 \times 27$ .

Al finalizar cada octava, la siguiente dobla el tamaño (de 6 a 12, y de 12 a 48) tal como muestra la figura 4.11.



**Figura 4.11:** Scale-Space en SURF. Octavas dentro del espacio de escalas. Fuente [9].

#### 4.1.3.3. Localización del punto de interés

El siguiente paso es determinar la localización de los máximos, procedimiento similar al seguido por el algoritmo *SIFT* ([6], [7] y [33]).

Con el Scale-Space, se realiza la supresión de no-máximos, este proceso requiere comparar al pixel con sus 8 vecinos de la escala actual junto con los 9 superiores e inferiores (ver figura 4.6), lo cual, dentro de cada octava, incluye sólo a las dos escalas interiores.

Al ser localizados, el siguiente proceso es realizar la interpolación tal como lo hace *SIFT*, con una expansión de Taylor, lo cual permitirá determinar la localización y la escala real del punto de interés a través de la ecuación:

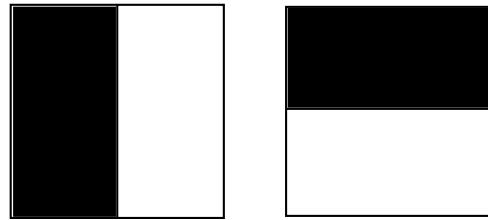
$$\mathcal{H}(x) = \mathcal{H} + \frac{\partial \mathcal{H}^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 \mathcal{H}}{\partial x^2} x \quad (4.26)$$

diferenciando e igualando a 0 se obtiene el punto:

$$\hat{x} = -\frac{\partial^2 \mathcal{H}^{-1}}{\partial x^2} \frac{\partial \mathcal{H}}{\partial x} \quad (4.27)$$

#### 4.1.3.4. Asignación de orientación

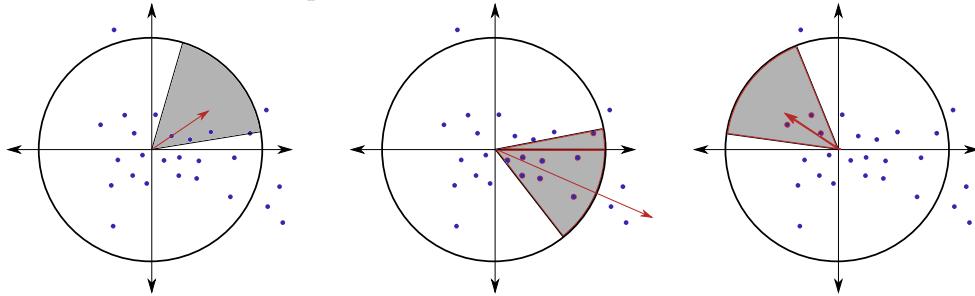
*SURF* hace uso de las respuestas de Haar Wavelet (*Wavelet de Haar, Ondícula de Haar*, ver figura 4.12) sobre las direcciones  $x$  y  $y$  dentro de una vecindad circular de radio  $\sigma = 6s$  alrededor del punto de interés, ésto permite extraer una orientación que es invariante a la rotación de la imagen, donde la variable  $s$  representa la escala en el punto de interés localizado afectando directamente al tamaño del wavelet, ajustándose a  $4s$ .



**Figura 4.12:** Haar Wavelet en SURF. Filtros Haar Wavelet para computar la respuesta en las direcciones  $x$ (izquierda) y  $y$ (derecha), blanco es 1 y negro es  $-1$ . Fuente [9].

Una vez computados los wavelets de  $x$  y  $y$  con las imágenes integrales, se realiza un mapa cartesiano con los valores resultantes. Cada punto es pesado con un gaussiano de tamaño  $\sigma = 2s$  basándose en la distancia hacia el punto de interés en un espacio que mide la respuesta horizontal y vertical.

La orientación dominante es estimada rotando una ventana desde el punto de interés de tamaño  $\frac{\pi}{3}$ . Los puntos dentro de la ventana son sumados generando un vector que variará en relación a la cantidad y distribución de estos (ver figura 4.13). El vector de mayor tamaño será la orientación del punto de interés.



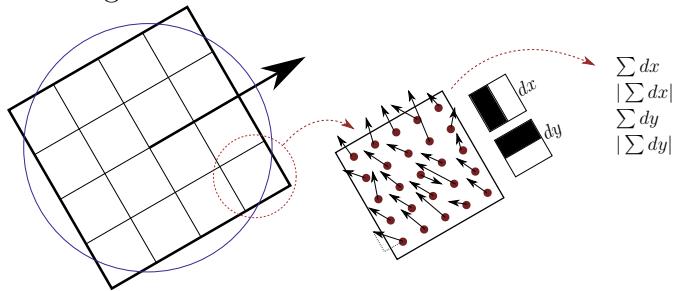
**Figura 4.13:** Orientación de puntos de interés en SURF. El vector con mayor fuerza será el seleccionado como la orientación dominante.

#### 4.1.3.5. Extracción de descriptores

La construcción del descriptor comienza con generar un cuadrado con un tamaño de  $20\sigma$  centrado y orientado en el punto de interés (imagen de la izquierda figura 4.14). Éste será dividido en  $4 \times 4$  regiones que cumplen lo siguiente:

- Cada región es  $5\sigma$  del cuadrado.
- Se computan wavelets Haar de tamaño  $2\sigma$  para los 25 puntos dentro de cada región.
- Para cada región, se computan 4 valores:  $v = [\sum dx, |\sum dx|, \sum dy, |\sum dy|]$ .

El descriptor resultante será un vector de 64 dimensiones que consisten en los 4 valores mencionados de las 16 regiones dentro del cuadrado.



**Figura 4.14:** Descriptor SURF. De izquierda a derecha: Un descriptor SURF consta de un cuadrado con divisiones de  $4 \times 4$ , donde se filtra con los Haar Wavelet para el área interna de esta región. Después se suman y se obtienen los 4 valores mencionados generando un vector de un tamaño de 64. Fuente [9] y [27], cap. 4, pág. 42].

#### 4.1.4. ORB

A pesar de la ventaja ofrecida por los algoritmos hasta ahora mencionados, por desgracia, el uso de *SIFT* y *SURF* se encuentra patentado e hipotéticamente, cualquier uso comercial involucra un pago por su uso<sup>12</sup>. A pesar de esto, con el avance en el área, han surgido algoritmos cuyo uso se encuentra libre de patente, o bien, no requiere (al menos) el pago a terceros, uno de estos casos es *ORB*.

Desarrollado por Ethan Rublee, Vincent Rabaud, Kurt Konolige y Gary Bradski en el 2011 [12], *ORB* (*Oriented FAST and Rotated BRIEF*) es la alternativa libre de *SIFT* y *SURF* que basa su trabajo en el extractor de características de Rosten y Drummond *FAST* [13] y el descriptor de Calonder et al. *BRIEF* [14].

El algoritmo de *ORB* sigue el siguiente proceso:

- Detección de los puntos de interés (o keypoints) *FAST*.
- Asignación de orientación.
- Definición de descriptor *rBRIEF*

<sup>1</sup>Patente SIFT: <https://www.google.com/patents/US6711293>

<sup>2</sup>Patente SURF: <https://www.google.com/patents/CN103640018B>

#### 4.1.4.1. Detección de puntos de interés oFAST

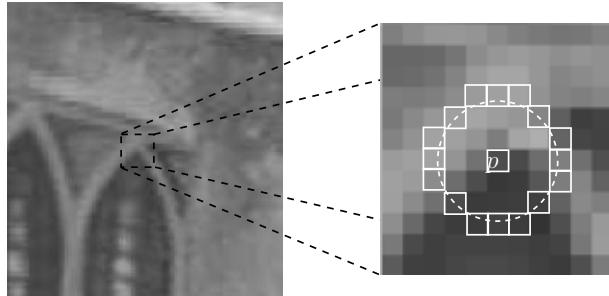
Desde su aparición [13], *FAST* (*Features from Accelerated Segment Test*) ha demostrado ser una alternativa acelerada a los algoritmos de extracción de características que basan su funcionamiento en *LoG* y *DoG* (como lo son *SIFT* y *SURF*).

La detección que realiza *FAST* se enfoca (como se ha visto en los anteriores) en las esquinas y comienza con la creación de un vector circular con centro en el pixel de interés  $p$  tal como muestra la figura 4.15.

Para cada locación en el círculo  $x \in \{1 \dots 16\}$ , el pixel en esa posición relativa a  $p$  (denotado como  $p \rightarrow x$ ) puede tener uno de los siguientes 3 estados:

$$S_{p \rightarrow x} = \begin{cases} d, & I_{p \rightarrow t} \leq I_p - t \text{ más oscuro (darker)} \\ s, & I_p - t \leq I_{p \rightarrow t} < I_p + t \text{ similar} \\ b, & I_p + t \leq I_{p \rightarrow t} \text{ más brillante (brighter)} \end{cases} \quad (4.28)$$

Existiendo una variedad de configuraciones para *FAST*, aquel utilizado por *ORB* es el *FAST-9*, este algoritmo busca la existencia de al menos 9 pixeles contiguos  $I_{p \rightarrow t}$  que tengan una clasificación  $b$  dentro del vector circular, es decir, más brillante que el pixel  $I_p + t$  (donde  $t$  es un umbral definido) tal como muestra la ecuación superior.



**Figura 4.15:** Keypoint *FAST* para *ORB*. En el vector formado por el círculo, la clasificación de una esquina viene determinado por el algoritmo utilizado, ya sea *FAST-9* o *FAST-12*. Fuente [13].

*FAST* no ofrece una forma de medir qué tanta “esquina” (*cornerness*) representa el pixel clasificado como tal. Para ello, *ORB* usa una medición tal como lo hace *Harris Corner* (visto en 4.1.1), midiendo el *cornerness* del keypoint, y mediante otra umbralización, descarta nuevamente aquellos pobremente localizados.

Así mismo, las características *FAST* no son invariantes a la escala (como *SIFT* y *SURF*), para lograr este objetivo, *ORB* desarrolla una pirámide de escalas, obteniendo los keypoints de los diferentes niveles dentro de la pirámide.

#### 4.1.4.2. Asignación de orientación.

La asignación de la orientación de los keypoints detectados se realiza con la medida de intensidad del centroide (*intensity centroid*). Ésto asume que la intensidad de la esquina tiene un desplazamiento en relación a su centro, y este vector puede ser usado para imputar una orientación. Esto es definido como:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \quad (4.29)$$

encontrando el centroide:

$$C = \left( \frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \quad (4.30)$$

pudiendo construir un vector que va desde el centro de la esquina,  $O$ , al centroide  $\vec{OC}$  definiendo la orientación como:

$$\theta = \text{atan2}(m_{01}, m_{10}) \quad (4.31)$$

#### 4.1.4.3. Definición del descriptor rBRIEF.

*BRIEF*, desarrollado por Michael Caloner, Vincent Lepetit, Christoph Strecha y Pascal Fua en el 2010 [14], es un descriptor cuya principal cualidad es su reducido tamaño y su rápido cómputo, donde con un peso de 64 *bytes* afirma ofrecer un resultado similar a los descriptores populares de *SIFT* y *SURF* teniendo cada uno un tamaño de 512 *bytes* y 256 *bytes* cada uno respectivamente [26, cap. 3, pág. 13].

Esta rapidez es lograda debido a que el descriptor *BRIEF* esta constituido por una cadena de *bits*, resultado de un conjunto de test binarios calculados a partir de una imagen integral. Ésto lo vuelve conveniente a la hora de determinar la similitud entre descriptores ya que sólo es requerida la distancia Hamming, proceso calculable bastante rápido por un computador convencional.

De manera formal, el descriptor *BRIEF* es una cadena de bits que define parejas dentro de un área  $\mathbf{p}$  de tamaño  $S \times S$  (llamado *patch*) de una imagen construida a partir de test binarios  $\tau$  definidos como:

$$\tau(\mathbf{p}; x, y) := \begin{cases} 1 & : \mathbf{p}(x) < \mathbf{p}(y) \\ 0 & : \mathbf{p}(x) \geq \mathbf{p}(y) \end{cases}, \quad (4.32)$$

donde  $x$ ,  $e$  y son dos ubicaciones y  $\mathbf{p}(x)$  y  $\mathbf{p}(y)$  son los valores de intensidad del pixel en una versión suavizada de  $\mathbf{p}$  en el punto. De esta manera, cada conjunto de locaciones  $(x, y)$  de  $n_d$  definirán unívocamente el conjunto de *test* binarios sobre el patch  $\mathbf{p}$ . Pudiendo definir el descriptor como la cadena de bits:

$$f_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}; x_i, y_i) \quad (4.33)$$

donde en *ORB* se ha optado por una distribución gaussiana en torno al *patch* con un vector de longitud  $n = 256$ . El tamaño del *patch* es de  $31 \times 31$  pixeles con subdivisiones de  $5 \times 5$ .

Pero *BRIEF* no es un descriptor robusto frente a la rotación, para ello, *ORB* realiza un ajuste del descriptor “guiándolo” con la orientación del keypoint previamente calculado. Para cada característica dentro del vector de longitud  $n$  en la posición  $(x_i, y_i)$  se define una matrix de  $2 \times n$ :

$$\mathbf{S} = \begin{bmatrix} x_1, & \dots, & x_n \\ y_1, & \dots, & y_n \end{bmatrix} \quad (4.34)$$

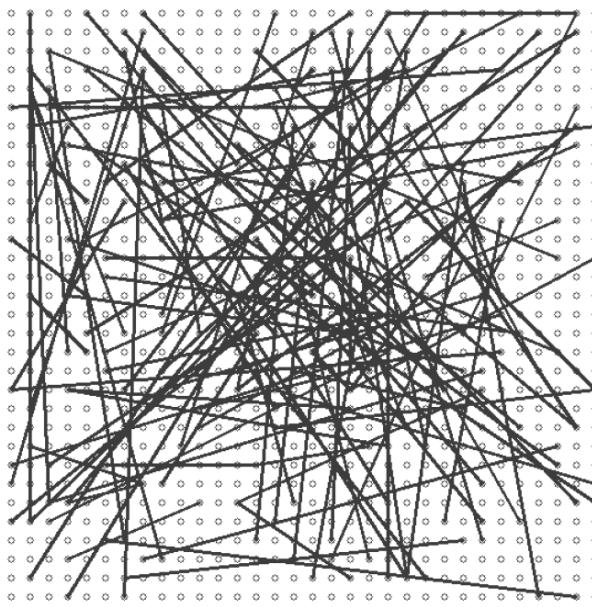
Usando la orientación del patch  $\theta$  y la rotación correspondiente de la matriz  $R_\theta$  se construye una versión “guiada”  $S_\theta$  de  $S$ :

$$\mathbf{S}_\theta = R_\theta \mathbf{S} \quad (4.35)$$

lo que modifica el operador *BRIEF* a:

$$g_n(p, \theta) := f_n(p)|(x_i, y_i) \in S_\theta \quad (4.36)$$

en [12], se discretiza la orientación del ángulo a incrementos de  $2\pi/30 = 12^\circ$  y se construye una tabla de patrones precomputados de *BRIEF*.



**Figura 4.16:** Descriptor BRIEF para ORB. Disposición espacial de los *test* binarios sobre un patch de una imagen. Fuente: [14].

Un detalle a la hora de “guiar” a *BRIEF* es la pérdida de varianza. Ésto resulta en un problema debido a que parte de la robustez de este algoritmo se la debe a esta cualidad, ya que una variación amplia permite extraer características más discriminantes.

Para resolver ésto, *ORB* utiliza una búsqueda codiciosa (*greedy search*) sobre todos los posibles test binarios para encontrar aquellos con gran variación, con media cercana a 0.5 y que no estén correlacionados. El resultado se le conoce como *rBRIEF* y es el último proceso realizado dentro de *ORB*.

#### 4.1.5. Resultados experimentales

Una vez descritos los diferentes métodos para extraer características, se procede a determinar el más óptimo. Se ha optado por la programación de los mismos en lenguaje *C++* en el entorno de desarrollo *QtCreator ver. 3.3.1* que hace uso de *Qt 5.4.1* y el compilador *MinGW*.

Así mismo, se ha hecho uso de las librerías de *OpenCV (Open Source Computer Vision)* las cuales están conformadas por un conjunto de paquetes especiales para el área de CV. La lista completa, así como las características del equipo de cómputo de pruebas son los siguientes:

- Memoria RAM : 3GB.

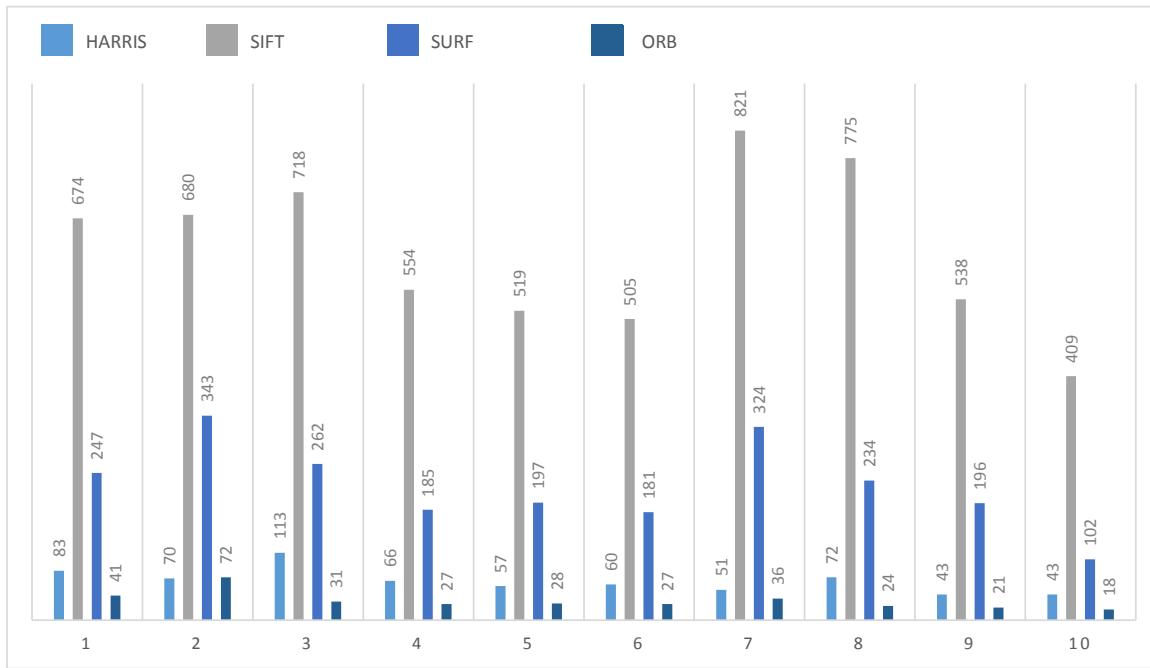
- Procesador: Intel Core i5 2.53GHz.
- Sistema Operativo: Windows 7 Ultimate 64bits.
- Lenguaje: C++.
- Librerías: Qt 5.4.1 y OpenCV 2.49.
- IDE de desarrollo: QtCreator V 3.3.1.

#### 4.1.5.1. Tiempo computacional

Esta prueba se encuentra dividida en dos etapas: la primera de ellas es una extracción de características sobre un conjunto de 10 imágenes sobre las que se correrán 4 algoritmos: *Harris*, *SIFT*, *SURF* y *ORB*. La segunda, correrá sobre una sola imagen a la cual se le aplicarán diversas transformaciones, ruido y deformaciones.

En la figura 4.17 se muestra la corrida de los algoritmos sobre 10 imágenes de prueba. Cada imagen probada es a color (en formato *RGB*) a un tamaño de  $640 \times 480$  (promedio general de una imagen estándar).

Cabe aclarar que el algoritmo *Harris* tiene ventaja en esta prueba, ya que por si solo no ofrece un método para computar sus descriptores, proceso que si realizan *SIFT*, *SURF* y *ORB*. El resultado, expresado en milisegundos, arrojó una considerable ventaja de *ORB* sobre los otros 3 donde sus tiempos pueden estimarse en hasta un 200 % mejor comparando los casos extremos.



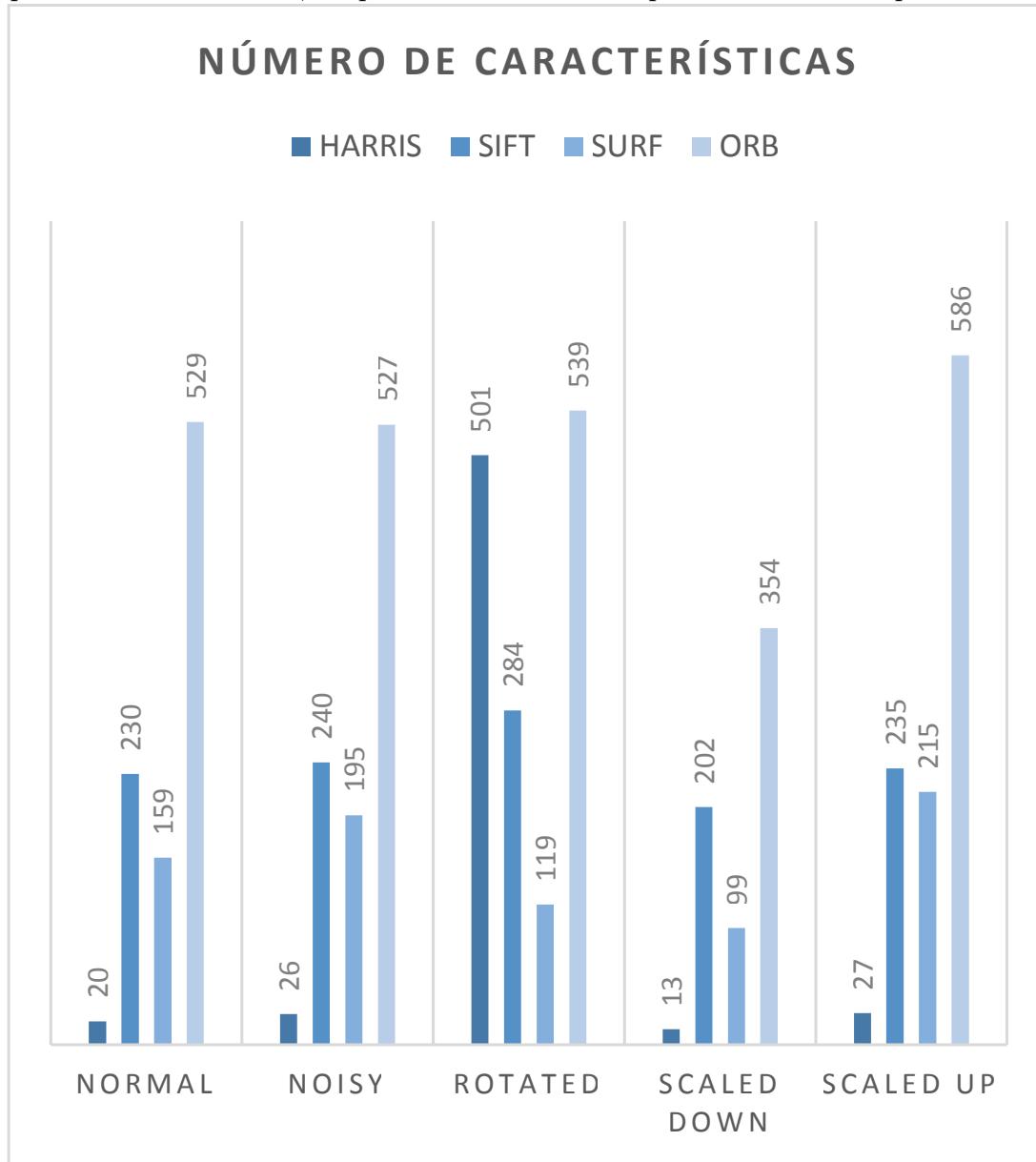
**Figura 4.17:** Tiempo de extracción y descripción de características para los algoritmos *Harris*, *SIFT*, *SURF* y *ORB*, el tiempo se encuentra medido en milisegundos.

En el siguiente test, se realiza una extracción de características y descriptores sobre una misma imagen a la que se le han aplicado diferentes modificaciones enumeradas a continuación:

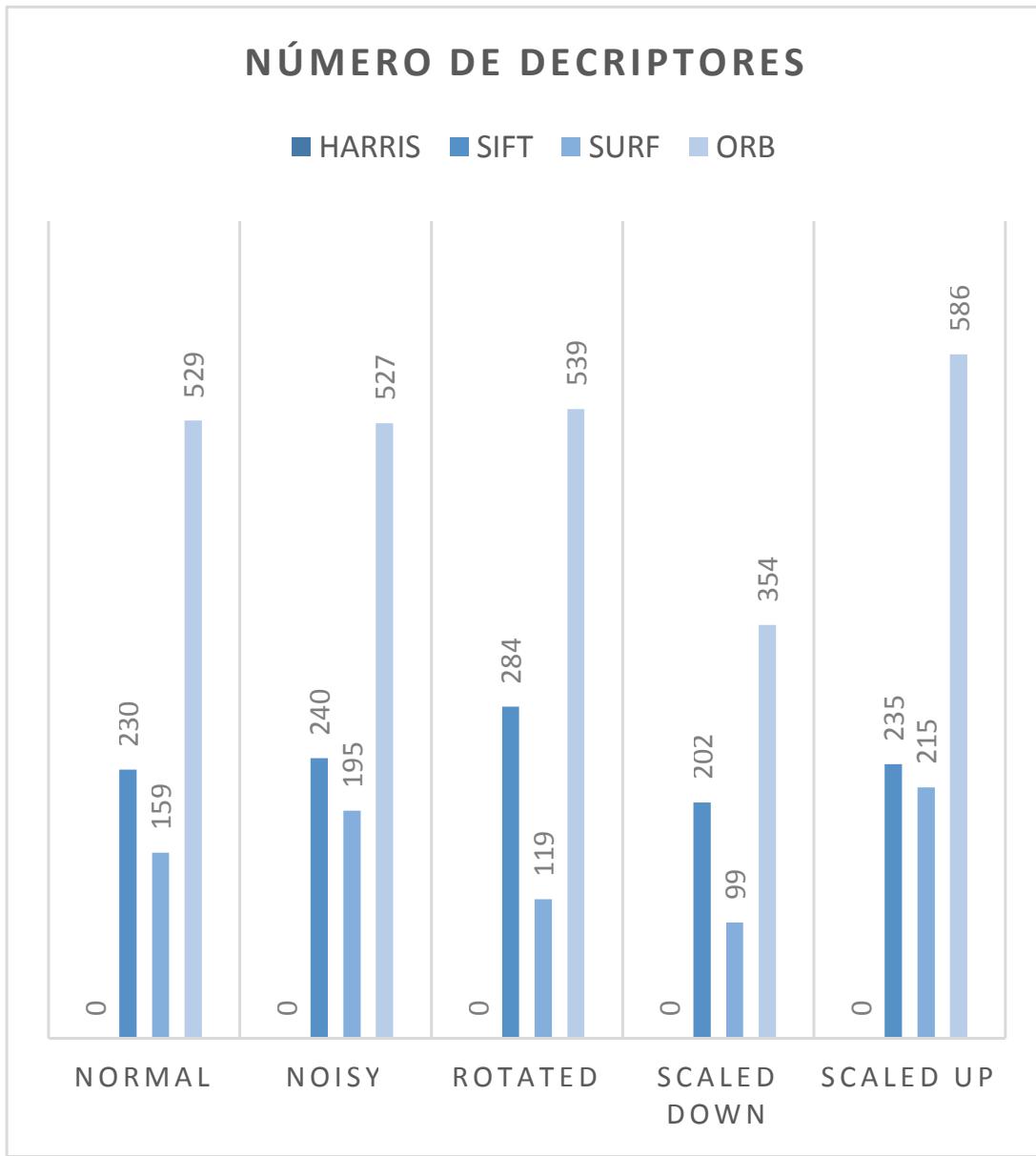
- **Normal.** Imagen a color de  $640 \times 480$  pixeles, tamaño mínimo promedio de las imágenes manipuladas en el área de microscopía.
- **Noisy.** Imagen original más ruido gaussiano con una variación media de 10 y una desviación estándar de 10.
- **Rotated.** Rotación de la imagen original a  $30^\circ$  contrarreloj.
- **Scaled-down.** Reducción de la imagen original a un 70 % del tamaño original.
- **Scaled-up.** Incremento de la imagen original a un 130 % del tamaño original.

Bajo los resultados observados (figuras 4.18, 4.19, 4.20) se aprecia la escasa variación existente entre una imagen con ruido y la imagen original para los 4 algoritmos en las 3 mediciones realizadas (número de características detectadas, número de descriptores computados y tiempo consumido a lo largo del algoritmo).

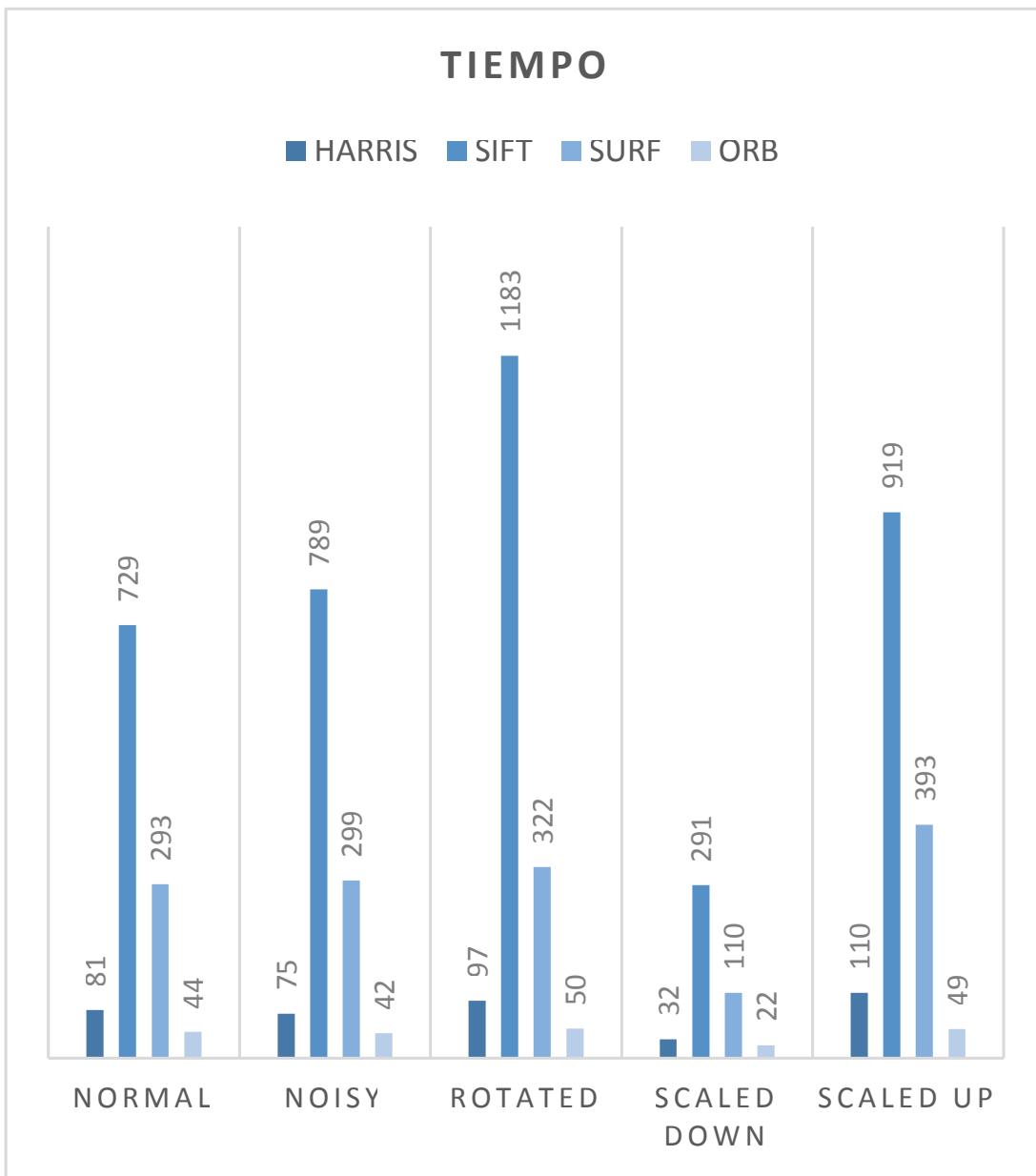
La mayor variación se encuentra entre la imagen escalada superior e inferior, donde los algoritmos de *Harris*, *SIFT* y *SURF* sufren el mayor impacto. El algoritmo que resultó con menor variación fue *SIFT* aunque su tiempo computacional sea mayor al doble de cualquiera de los otros tres, lo que lo desfavorece en aplicaciones de tiempo real.



**Figura 4.18:** Número de características detectadas por algoritmo sobre una sola imagen a la que se le aplicó de manera separada ruido gaussiano y tres transformaciones: Rotación, Escalación inferior y superior.



**Figura 4.19:** Número de descriptores computados por algoritmo sobre una sola imagen a la que se le aplicó de manera separada ruido gaussiano y tres transformaciones: Rotación, Escalación inferior y superior.



**Figura 4.20:** Tiempo consumido tras detección de descriptores por algoritmo sobre una sola imagen a la que se le aplicó de manera separada ruido gaussiano y tres transformaciones: Rotación, Escalación inferior y superior.

En conclusión, bajo este marco, los algoritmos con mayores prestaciones son tanto *SURF*, como *ORB*. Cabe aclarar que aún cuando *SIFT* ofrece muy buenos resultados, su tiempo computacional es una limitante de gran efecto en el proyecto si lo que se busca es un rastreo en tiempo real. De cualquier manera, su estudio seguirá en posterior análisis para verificar estabilidad y vulnerabilidad.

## 4.2. Empate de Características

Tras la experimentación con los diferentes descriptores y los resultados preliminares que arrojaron los mismos, se opta por continuar el proyecto de la tesis teniendo como base el uso de los descriptores *ORB*.

En este proceso intervienen diferentes algoritmos que dependen del descriptor de entrada encargados de obtener el vecino más cercano (*knn* ó *k-nearest neighbor*) a través de algoritmos como la *fuerza bruta* o *aproximados*.

El principio detrás de estos métodos se centra en encontrar un número predefinido de muestras más próximo a un punto dado. Éstas pueden ser constantes definidas por el usuario (por ejemplo, *knn*) o variar en relación a la densidad local de puntos (*radius-based neighbor learning*). La distancia resultante es cualquier medida métrica, siendo la distancia euclíadiana la más común.

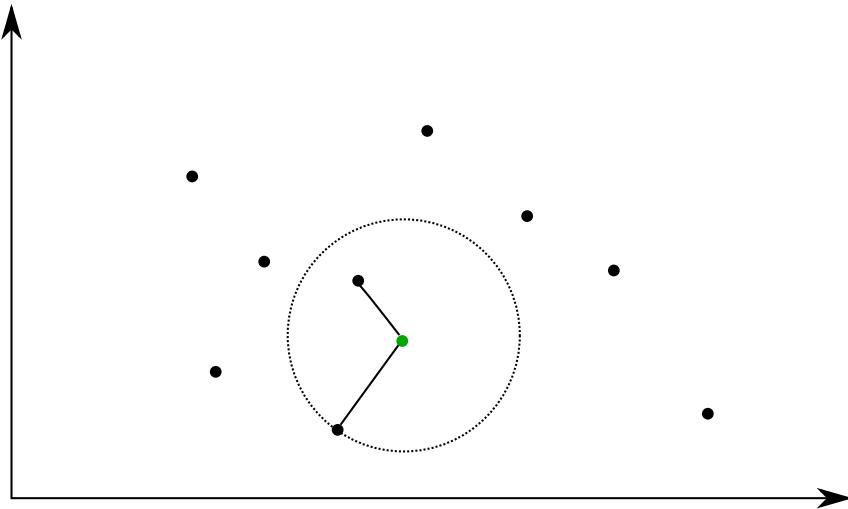
A pesar de su simplicidad, los métodos de vecino más cercano han sido empleados satisfactoriamente en un número de problemas referentes a la *clasificación* y *regresión*. En la primera parte de la sección se estudiará el algoritmo del k-vecino más cercano por fuerza bruta (*knn*), seguido del vecino más cercano aproximado acelerado ([34]). Finalizando con un estudio que permita determinar cual de los dos métodos será seleccionado y porque.

### 4.2.1. *knn* por Fuerza Bruta

El más sencillo dentro de la familia de vecinos más cercanos. Es el cómputo de todas las distancias entre los diferentes puntos dentro de un conjunto.

Para  $P$  muestras de  $D$  dimensiones, este enfoque tiene una complejidad de  $O[DP^2]$ .

En una definición más formal y extraído de [27, cap. 5, pág. 47]: si tenemos un conjunto de puntos  $P$ , constituido por  $P = \{p_1, p_2, \dots, p_m\}$  en un espacio  $d$ -dimensional, y suponiendo que  $q$  es el punto a clasificar dentro del mismo espacio, entonces el algoritmo se encargará de determinar el/los punto/s  $k$  más cercano/s a  $q$  dentro de  $P$  tal como se observa en 4.21.



**Figura 4.21:** Algoritmo  $k$ -nn de dos dimensiones con  $k = 2$ .

Un algoritmo eficiente de fuerza bruta puede ofrecer excelentes resultados para un conjunto pequeño de datos, pero, a medida que el tamaño de  $P$  o  $D$  incrementa, este algoritmo se vuelve inviable.

### 4.2.2. ANN

Cuando el número de dimensiones en el conjunto de puntos  $P$  es mayor a 8, computar el vecino más cercano exacto puede resultar una difícil tarea. Pocos métodos parecen ser significativamente mejores que un cómputo por fuerza bruta. De cualquier forma, se ha observado que es posible realizar este proceso en tiempos significativamente menores (en un orden de 10 a 100) con un margen de error pequeño tal como se muestra en el algoritmo encontrado en [34], resumido en ocasiones con el nombre de *ANN* o *FLANN* ([35]).

*ANN* es un algoritmo que prioriza la búsqueda del vecino más cercano a través de la creación de árboles jerárquicos de  $k$ -medias. El punto más cercano no necesariamente empatará con el punto base, así que pruebas posteriores deberán ser realizadas para incrementar la precisión de empate (como la *prueba de radio* o la *prueba de simetría*).

### 4.2.3. Incrementando Precisión

El uso de cualquiera de los algoritmos anteriores (*fuerza bruta* o *ANN*) genera varios empates falsos. Antes de estimar la homografía, es necesario remover estos falsos empates para que no generen ruido durante su proyección. Para ello se usan las pruebas de *razón* y *simetría*.

#### 4.2.3.1. Prueba de Razón

Para una búsqueda  $knn$ , donde el algoritmo encontrará los 2 puntos más cercanos (es decir,  $k = 2$ ). Una prueba de razón o proporción se encargará en determinar proporción de distancia entre estos dos puntos.

Definiendo un umbral  $T$  y suponiendo que  $d_1$  y  $d_2$  son las distancias del punto base a los puntos de empate más cercanos, donde  $d_1 < d_2$ . Los puntos rechazados (y que no serán tomados en cuenta) serán aquellos en los que se satisfaga la siguiente condición:

$$\frac{d_1}{d_2} > T \quad (4.37)$$

Si dos puntos tienen aproximadamente la misma distancia, la proporción tenderá a ser mayor, implicando un falso empate. La prueba debe asegurar que un punto estará más próximo, mientras el otro alejado. A mayor umbral, mayor número de empates (buenos o malos), y a un menor umbral, los empates serán más acertados pero puede que no sean suficientes. Un óptimo desempeño existe dentro de un valor de 0.7 a 0.8.

#### 4.2.3.2. Prueba de Simetría

En la prueba de simetría se computa los empates entre el par de imágenes en ambas direcciones, seleccionando aquellos que cumplan con el siguiente enunciado:

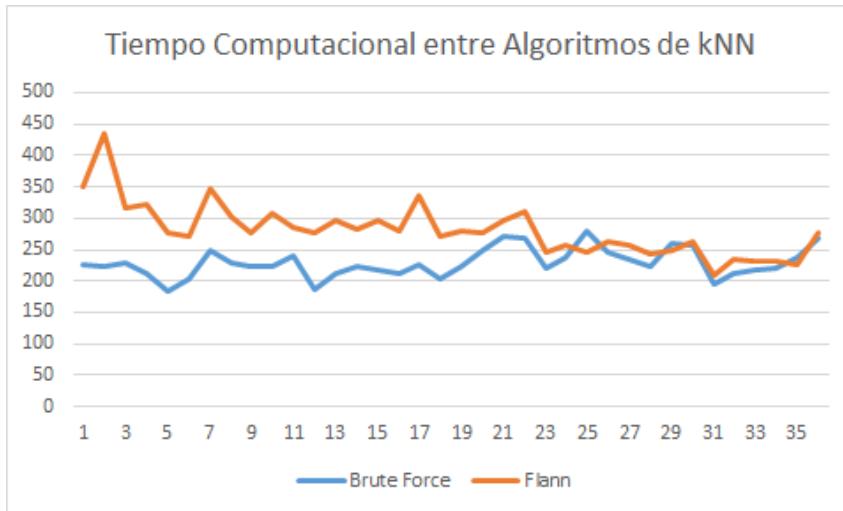
Suponiendo que se tiene un punto  $p_1$  en la primera imagen que empata con el punto  $p'_1$  de una segunda, obteniendo la pareja  $\{p_1, p'_1\}$ , el punto  $p'_1$  de esta última imagen deberá empatar con el punto  $p_1$  de la primera.

Si  $p'_1$  obtiene otro punto que no sea  $p_1$ , entonces esta pareja será descartada y no habrá pasado la prueba de simetría.

#### 4.2.4. Resultados Preliminares

Bajo un total de 8 imágenes, de un tamaño de  $640 \times 480$  se computaron los descriptores *ORB*, obteniendo 1000 puntos de interés por imagen a ser comparados entre sí a los cuales se les computará los vecinos más cercanos.

La imagen 4.22 muestra los resultados del cómputo de los algoritmos *knn por fuerza bruta* (azul) y *ANN* (naranja), donde el eje  $y$  indica el tiempo en milisegundos y el eje  $x$  indica la pareja evaluada, siendo un total de 36 parejas.



**Figura 4.22:** Prueba de *fuerza bruta* y *FLANN*. Se realizó una prueba a 8 imágenes entre sí, probando la imagen 1 con la 2, 3, 4, 5..., la 2 con la 3, 4, 5..., etcétera, teniendo un total de 36 parejas de imágenes con al menos 1000 puntos de interés para cada una. A ambos algoritmos se les ha realizado, además, una prueba de razón y simetría, con un umbral de 0.7.

Como se observa, los resultados arrojaron tiempos menores bajo el algoritmo de *fuerza bruta*.

Aún cuando *FLANN* puede ser un método optimizado, estos resultados indican que éste no puede ser lo mejor para el enfoque en cuestión, debido al uso del descriptor *ORB*, ya que *FLANN* es un algoritmo orientado a *SIFT* y *SURF*.

El resultado de este proceso obtendrá diferentes parejas entre pares de imágenes que indicarán las similitudes entre ellas. E incluso con las pruebas para aumentar la precisión, existirán elementos mal clasificados que, de no ser manejados apropiadamente, generarán problemas de alineación.

Esta similitud entre los puntos clave entre imágenes reciben el nombre de *inliers* para aquellos correctamente clasificados, y *outliers* para aquellos que no y toman un papel importante en la estimación de la matriz de similaridad.

### 4.3. Estimación de Similaridad

Con la finalidad de obtener los diferentes parámetros de transformación necesarios para unir las imágenes en una sola, se debe estimar la matriz de similaridad.

Dentro del enfoque de estudio, las imágenes fuente sufrirán efectos de traslación, rotación y escalación, teniendo un total de 4 grados de libertad. Una transformación de similaridad

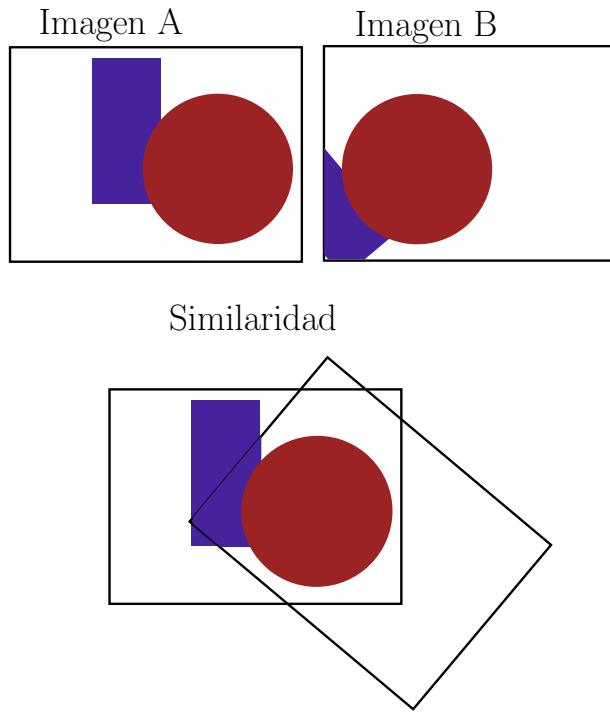
(y como se ve en 3.2.4), es una matrix  $x'$  de  $2 \times 3$  que contiene estos parámetros y está dada por la formula:

$$x' = [sR \quad t] \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x} \quad (4.38)$$

La figura 4.23 muestra el objetivo de obtener una matriz de similaridad.

Dadas dos imágenes, esta matriz permitirá determinar el nivel de transformación requerido para ser acopladas correctamente.

Para la estimación de estos parámetros existen varios métodos que engloban conceptos de regresión lineal, tales como *LS*, *LMS* y *RANSAC*.



**Figura 4.23:** Transformación por Similaridad para obtención de parámetros de traslación, rotación y escalación con la finalidad de obtener la imagen panorámica.

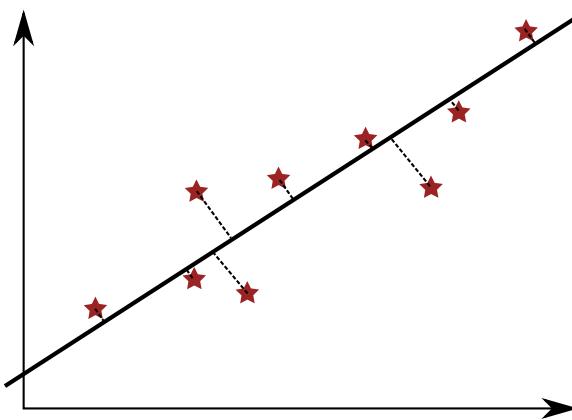
### 4.3.1. LS

El algoritmo de *Mínimos Cuadrados* (ó *LS*, por *Least Squares*) es un método para realizar una regresión lineal que permite ajustar una función sobre una dispersión de puntos.

Conociendo la relación lineal  $y(x) = a + bx$ , en los *mínimos cuadrados* se deben encontrar los parámetros  $a$  y  $b$  que provean la mejor solución que minimice la suma de cuadrados residuales:

$$f(a, b) = \sum_{i=1}^n (y_i - a - bx_i)^2 \quad (4.39)$$

Buscando la línea  $l = \{(x, y) \in \mathbb{R}^2 | y = a + bx\}$  más cercana al conjunto de puntos con la finalidad de minimizar la distancia vertical, tal como muestra la figura 4.24.



**Figura 4.24:** Regresión Lineal por Mínimos Cuadrados. Regresión lineal mediante el algoritmo de mínimos cuadrados en una gráfica con dispersión de puntos.

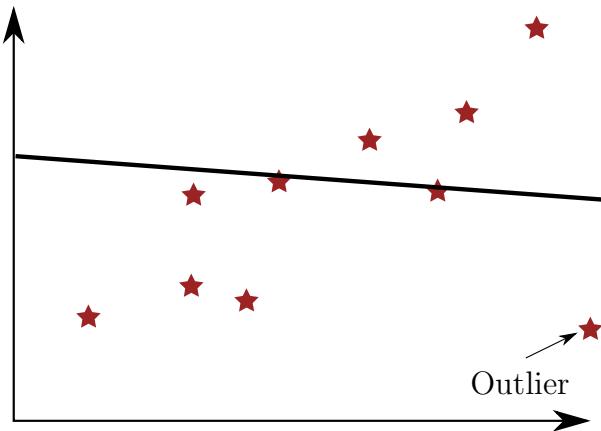
En un enfoque al problema de imágenes panorámicas, el algoritmo anterior puede ser implementado como [36]:

$$[A^*|b^*] = \underset{[A|b]}{\operatorname{argmin}} \sum_i \|dst[i] - Asrc[i]^T - b\|^2 \quad (4.40)$$

donde  $dst$  y  $src$  corresponden al conjunto de puntos 2D obtenidos en el capítulo anterior, y  $[A^*|b^*]$  toma la forma de la matriz de similaridad como la ecuación 4.38, reduciendo el problema a la resolución de la matriz:

$$[A^*|b^*] = \begin{bmatrix} a_{1,1} & -a_{1,2} & b_1 \\ a_{1,2} & a_{1,1} & b_2 \end{bmatrix} = \mathbf{x}' \quad (4.41)$$

El algoritmo de Mínimos Cuadrados es robusto frente a elementos alineados (*inlier*) pero entorpece a medida que aparecen elementos no alineados (*outlier*) tal como se observa en la figura 4.25.



**Figura 4.25:** Sensibilidad del Algoritmo de Mínimos Cuadrados. En la gráfica se puede apreciar cómo la existencia de pocos *outliers* puede alterar en gran medida la estabilidad de la función obtenida.

Para solucionar este problema se deben implementar algoritmos más robustos, como *LMS* ó *RANSAC*

### 4.3.2. LMS

Introducido por Peter J. Rousseeuw en 1984 [37], el algoritmo de Mínimos Cuadrados Promedios (*LMS*, *Least Mean of Squares*) es una mejora a *LS* que clama tener una robustez frente a datos que presentan hasta el 50 % de información corrupta (*outliers* en este caso).

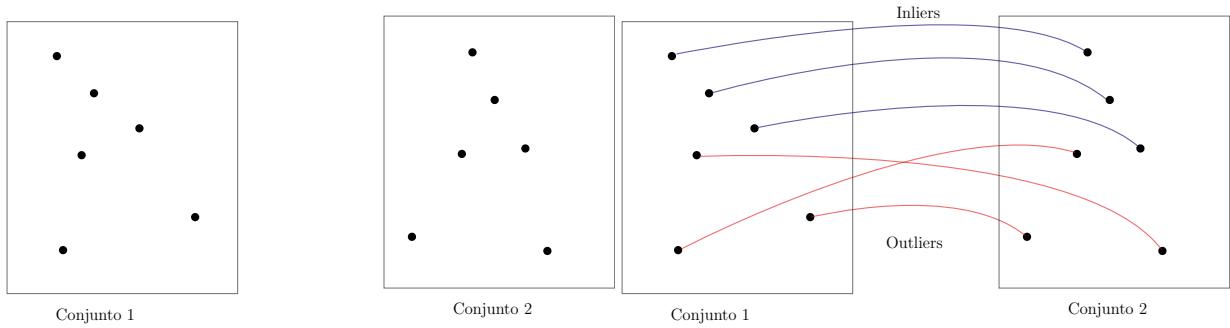
Dado  $n$  puntos  $(x_i, y_i)$ , al igual que *LS*, el objetivo es computar la linea que minimice dicha dispersión de puntos a través de la media de los cuadrados residuales. En este caso, se deben buscar los valores  $\alpha^*$  and  $\beta^*$  que minimice la función [38]:

$$f(\alpha, \beta) = \text{median}(|y_i - (\alpha + \beta x_i)|) \quad (4.42)$$

### 4.3.3. RANSAC

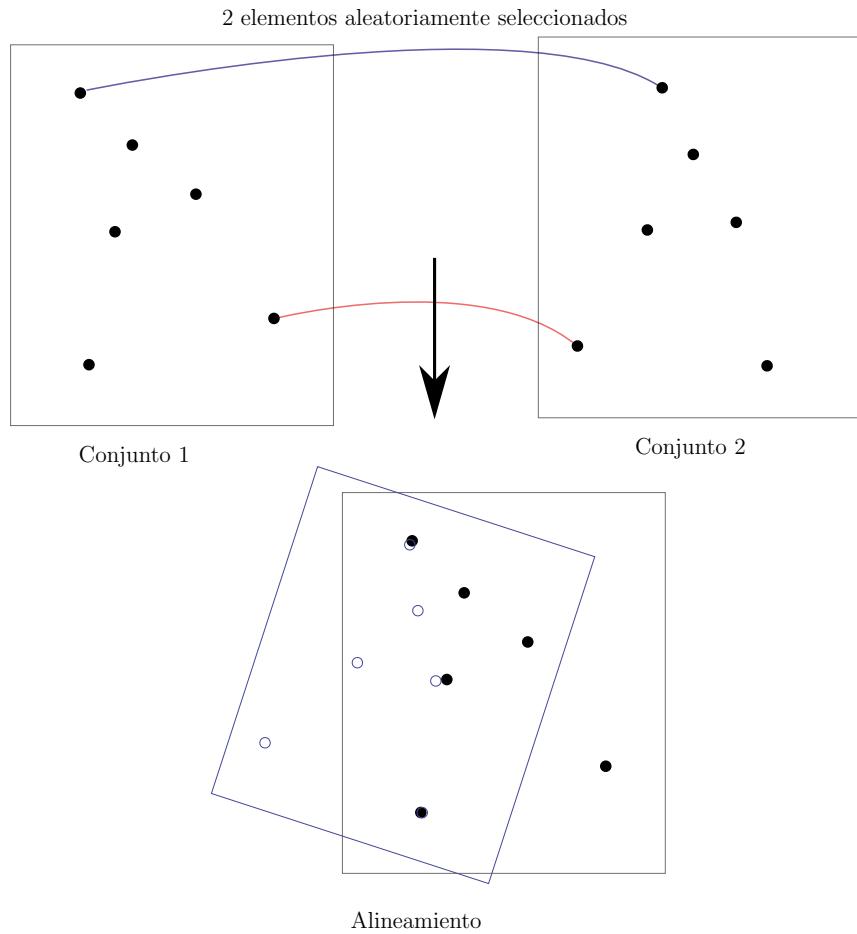
*RANdom SAmple Consensus (RANSAC)* es un método para obtener una función lineal sobre una dispersión de puntos que posee *outliers* desarrollado por Fishler, *et al* en 1981 [39] y es el método más utilizado para la estimación de parámetros de transformación como la homografía (o en este caso, el cálculo de la similaridad).

*RANSAC* es un buen algoritmo para detectar *outliers* y presenta una gran robustez cuando existe una gran cantidad de éstos (como se ve en [40]).



**Figura 4.26:** Dos conjuntos para aplicar RANSAC. Dado estos conjuntos a alinear, donde existen *outliers* presentes, el primer paso será seleccionar aleatoriamente dos elementos.

Este método selecciona aleatoriamente un número definido de correspondencias y computa la función de similaridad  $x'$  como se observa en la figura 4.26, 4.27 y 4.28. Después, los demás puntos se clasificarán como *inlier* y *outlier* dependiendo de su correspondencia con  $x'$ .

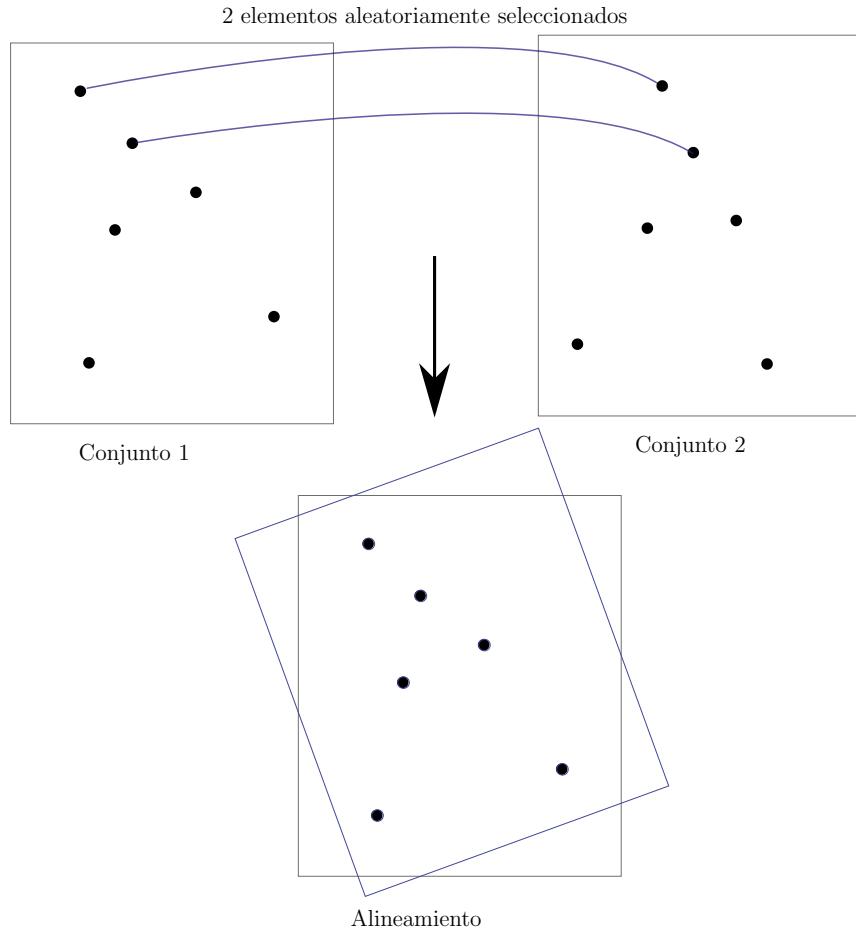


**Figura 4.27:** RANSAC con outliers. Cuando los elementos seleccionados contienen *outlier*, la alineación basada en estos puntos dejará un gran número de *outliers*.

Éste proceso será repetido varias veces y la función que otorgue el número mayor de *inlier* será seleccionada como el mejor modelo de transformación.

La clasificación entre *inlier* y *outlier* viene dada por un umbral  $T$  y si no cumple la condición, el punto es considerado *outlier*.

A mayor umbral, mayor número de *inlier* se obtienen tal como se muestran en las figuras 4.27 y 4.28.



**Figura 4.28:** RANSAC con inliers. Si la selección es de *inliers*, la alineación basada en estos puntos generará el mayor número de *inliers*.

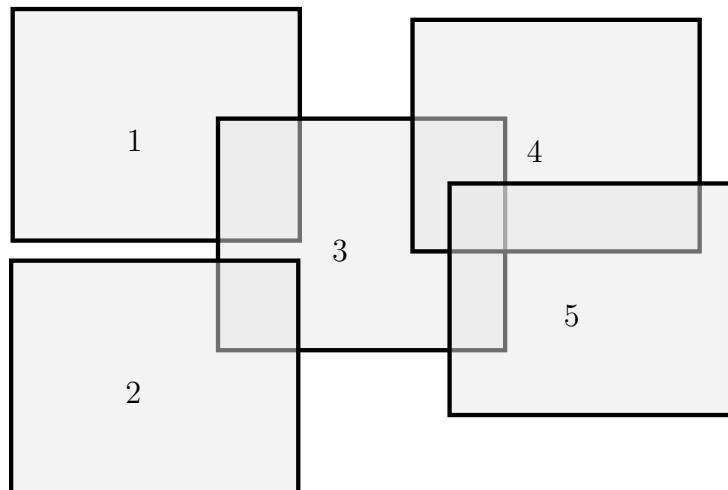
Aún cuando *RANSAC* es un algoritmo de más de 25 años, cabe destacar que su uso sigue vigente y ha dado las bases para un gran número de algoritmos que basan su funcionamiento en éste, tales como: *MSAC*, *MLESAC*, *QDEGSAC*, *NAPSAC*; *PROSAC*, *GASAC*, *MAPSAC*, *AMLESAC*, *uMLESAC*, etcétera. Por simplicidad se seguirá sólo con el original, pero un estudio más profundo puede ser encontrado en [40].

## 4.4. Composición

Teniendo los parámetros de transformación de las secciones anteriores, la siguiente parte centra su atención en la aplicación de dichos parámetros para la obtención de la imagen final.

Es importante resaltar las siguientes asunciones:

- El orden de las imágenes no se conoce y no viene dado por el orden de entrada (como se observa en la fig. 4.29).
- Existe un número máximo  $n - 1$  de posibles emparentados entre imágenes (dónde  $n$  es el número total de imágenes). En otras palabras, una sola imagen puede estar relacionada a todas las demás o con ninguna.



**Figura 4.29:** Ejemplo de ordenamiento de las imágenes finales. Nótese como el número de coincidencia (o el traslape entre imágenes) puede variar. La numeración marcada indica el orden en el que fueron recibidas más no el orden en el que fueron trasladadas.

Con ésto en consideración, se procede a iniciar el proceso de ordenamiento.

### 4.4.1. Selección de Ordenamiento

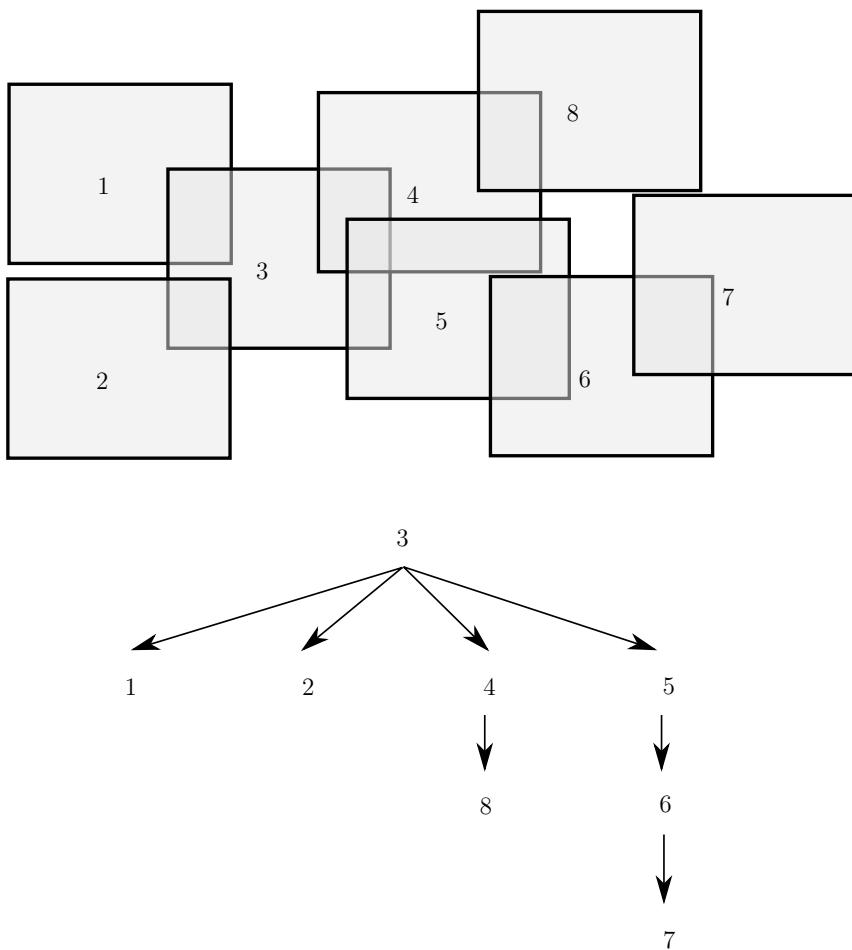
Cuando el número de imágenes base es mayor a 2, es necesario ofrecer un método que permita elegir la imagen pivote (la imagen que posee la posición [0, 0]), así como el orden de acomodo, que consta de la siguiente metodología:

1. Calcular los puntos de interés *ORB* de cada par de imagen entre sí. El número de combinaciones posibles viene dado por:

$$\sum_{i=1}^{n-1} i \quad (4.43)$$

siendo  $n$  el número total de imágenes.

2. Sumar la cantidad de puntos de interés de cada una de las imágenes entre sí.
3. Seleccionar aquella con mayor número de puntos de interés (ó aquella con mayor coincidencias entre imágenes) e identificarla como la imagen pivote. (índice 3 en la figura 4.30).
4. Posicionar esta imagen en la coordenada  $[0, 0]$  e identificar aquellas otras que contengan puntos de interés con la imagen pivote (imágenes con el índice 1, 2, 4 y 5 de la figura 4.30).
5. Una vez identificadas, posicionar las imágenes en base a los valores de transformación obtenidos de la matriz de similaridad.
6. Repetir el proceso de identificación de puntos de interés con cada una de las imágenes resultantes, agregando los valores de transformación de la imagen a la cual hacen referencia hasta que, o bien ya no haya disponibles, o ya no existan puntos de interés (imágenes 6, 7 y 8 de la figura 4.30).



**Figura 4.30:** Árbol de decisión de ordenamiento. En la parte superior de la figura se encuentra el posicionamiento real que tendrán 8 imágenes aleatorias capturadas. La numeración es un índice de identificación. La parte inferior de la figura señala la secuencia de ordenamiento que seguirá el sistema.

#### 4.4.2. Posprocesamiento

Existen técnicas que permiten suavizar las transiciones entre imágenes, así también técnicas de recorte para optimizar dicha unión, pero por la naturaleza del proyecto no son contempladas debido al ahorro de tiempo computacional y a la edición manual de imágenes (de cualquier manera, se puede consultar [41] para más información), dejando como último paso la entrega de la imagen panorámica en un formato cualquiera con o sin transparencia (como *jpeg* o *png*).

## 4.5. Interfaz Gráfica

La culminación de la investigación se expresa como la entrega de un software ofreciendo, de forma amigable, a un usuario común una herramienta que le permita realizar esta tarea de unión de imágenes para cualquiera que sea su fin. El desarrollo de dicho software tiene como base los siguientes puntos:

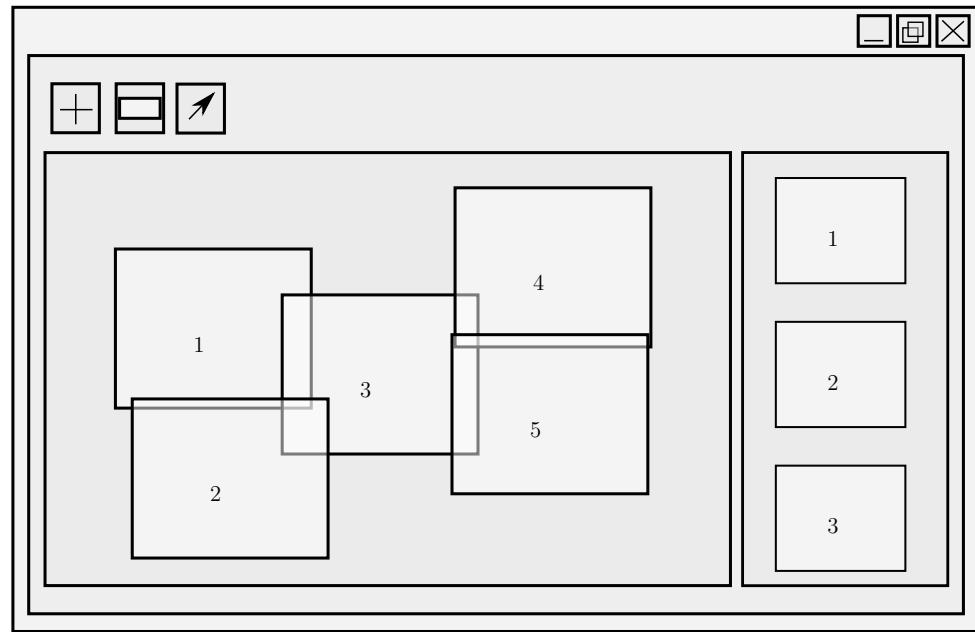
- Lenguaje de programación: C++.
- IDE: QT Creator 3.3.1.
- Patrón de diseño: Modelo-Vista-Controlador.
- Compilador GNU o GCC.

Un rápido prototipado sugiere una interfaz que cuente con un diseño como el mostrado en la figura 4.31<sup>3</sup> donde se implementen los siguientes accesos:

- Un botón para agregar aquellas imágenes provenientes de un microscopio óptico.
- Un botón que active la función de generación de panorámica.
- Un botón que permita exportar la imagen generada en un formato de imagen.
- Un área de trabajo donde se observarán las imágenes transformadas (la imagen panorámica).
- Una lista que muestre las imágenes disponibles.

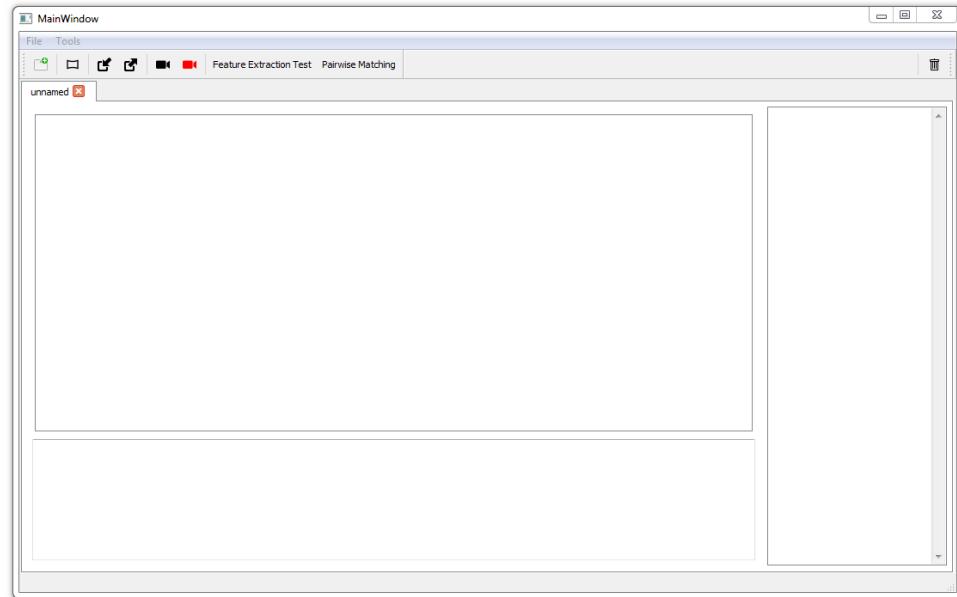
---

<sup>3</sup>También llamado Mockup, el cual es un modelo rápido que sirve como referencia para el desarrollo de interfaces



**Figura 4.31:** Prototipo de Interfaz. Boceto en el que se aprecian los elementos más sobresalientes que debe tener el sistema en cuestión.

El desarrollo tiene como resultado una interfaz como la de la figura 4.32, donde, además de lo estimado, se agregó una pequeña sección que indica el estado actual del sistema:

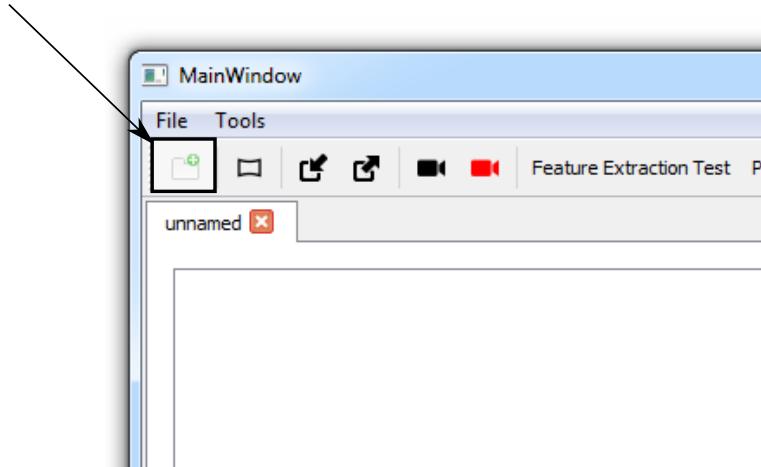


**Figura 4.32:** Interfaz de usuario. Básicamente cuenta con las mismas opciones que el mockup de la figura 4.31.

#### 4.5.1. Creando una imagen panorámica

Para la creación de una imagen panorámica, seguir el siguiente proceso<sup>4</sup>:

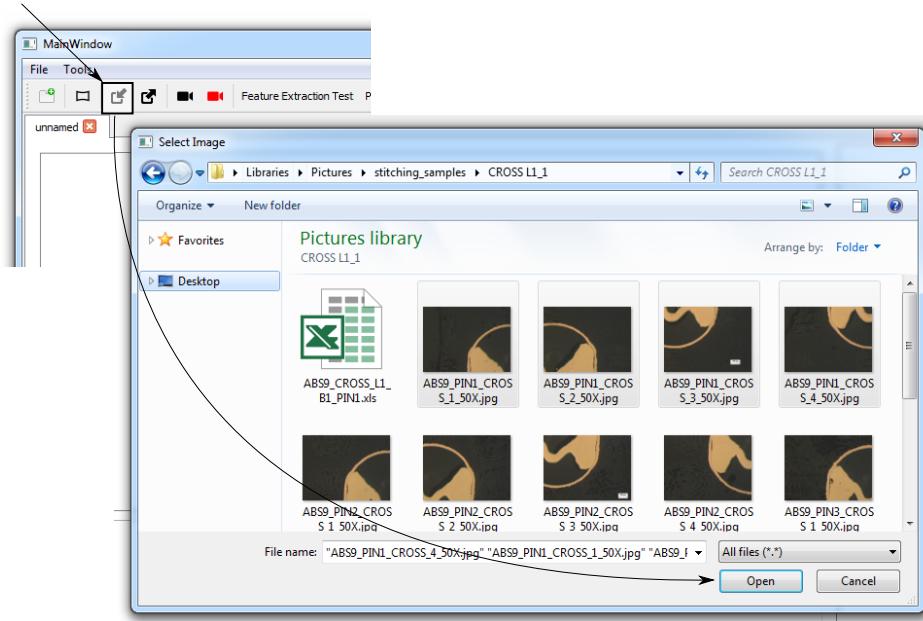
1. Crear un nuevo proyecto.
- Nuevo Proyecto



**Figura 4.33:** Creación de nuevo proyecto.

2. Cargar las imágenes a unir.

Carga de imágenes

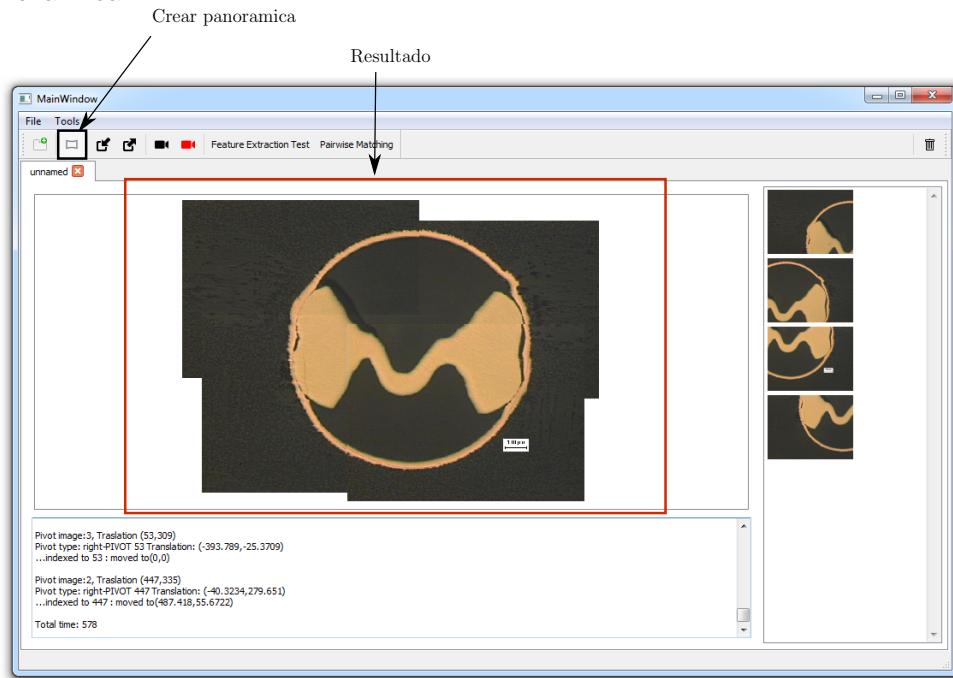


**Figura 4.34:** Carga de imágenes.

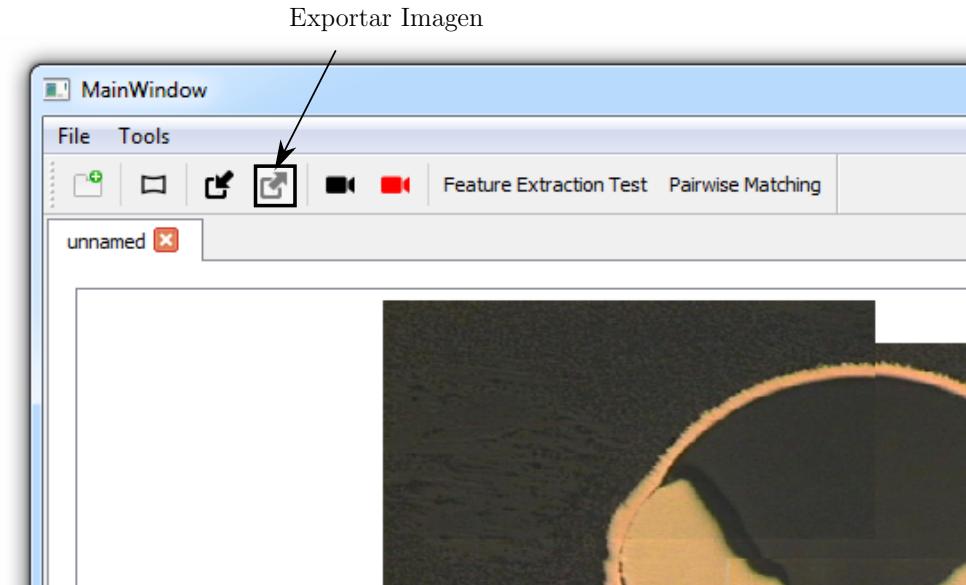
---

<sup>4</sup>Interfaz sujeta a cambios

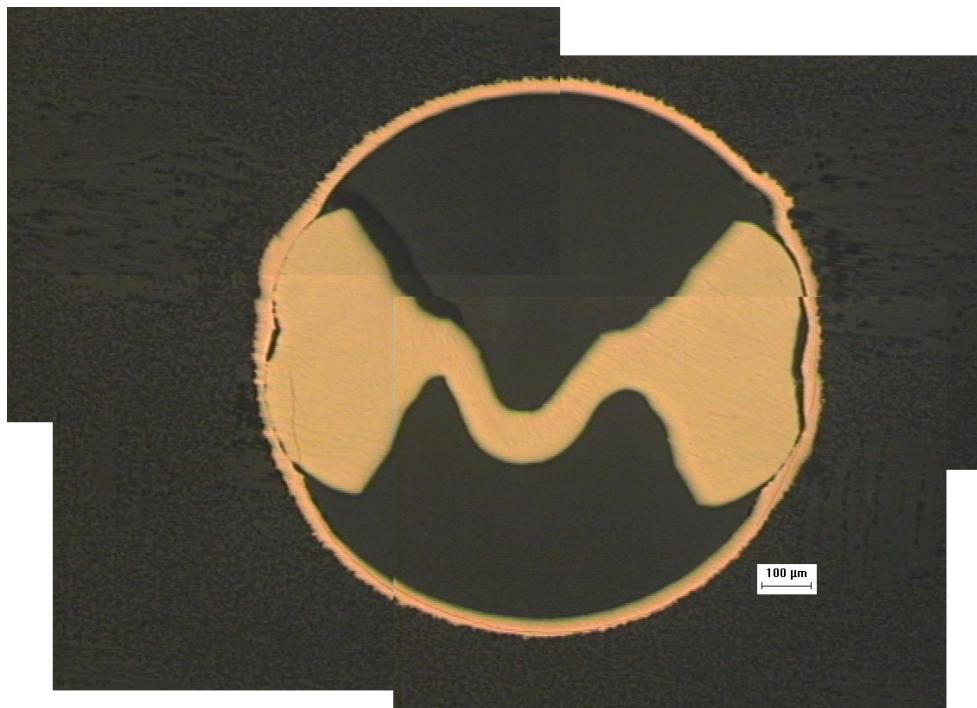
## 3. Crear panorámica.

**Figura 4.35:** Creación de panorámica.

## 4. Exportar resultado.

**Figura 4.36:** Exportar panorámica.

El resultado final ofrece una imagen similar a la de la figura 4.37.



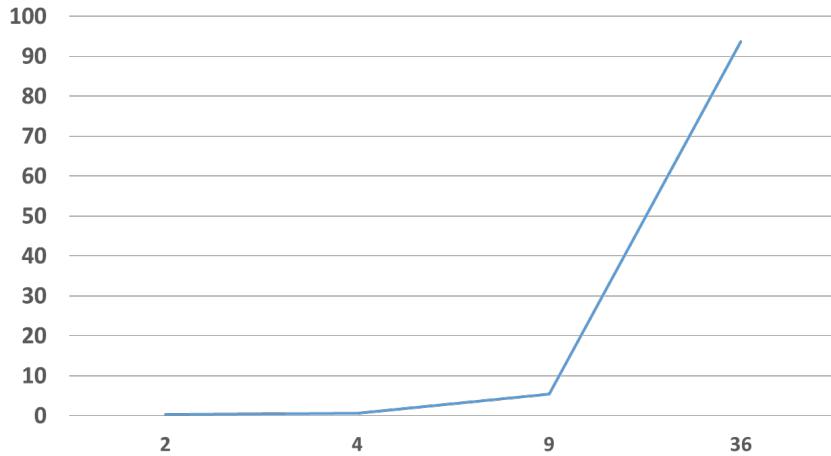
**Figura 4.37:** Imagen panorámica resultante.

# Capítulo 5

## Resultados

Tras la conclusión del software se realizaron diferentes pruebas sobre muestras recibidas de un laboratorio que cuenta con un microscopio óptico, así también sobre diferentes muestras recabadas de sitios especializados en la validación de esta clase de software.

Desde simples uniones de 2 imágenes, hasta un caso particular en el que se unieron 36 imágenes, el software pareció responder de manera positiva. Es interesante resaltar el comportamiento promedio del sistema donde, a mayor cantidad de imágenes, el tiempo incrementa a un nivel exponencial (véase figura 5.1), aún así permaneciendo dentro de un tiempo tolerable.



**Figura 5.1:** Tiempo de ejecución. El eje vertical representa el tiempo medido em segundos, mientras que el eje horizontal indica el número de imágenes que fueron unidas.

En la tabla 5.1, que no es más que la información de la gráfica 5.1, se pueden apreciar el tiempo promedio de ejecución para 2, 4, 9, y 36 imágenes, la segunda columna indica cuantas pruebas fueron realizadas donde intervinieron este número de imágenes y la tercera

columna indica el tiempo promedio en segundos.

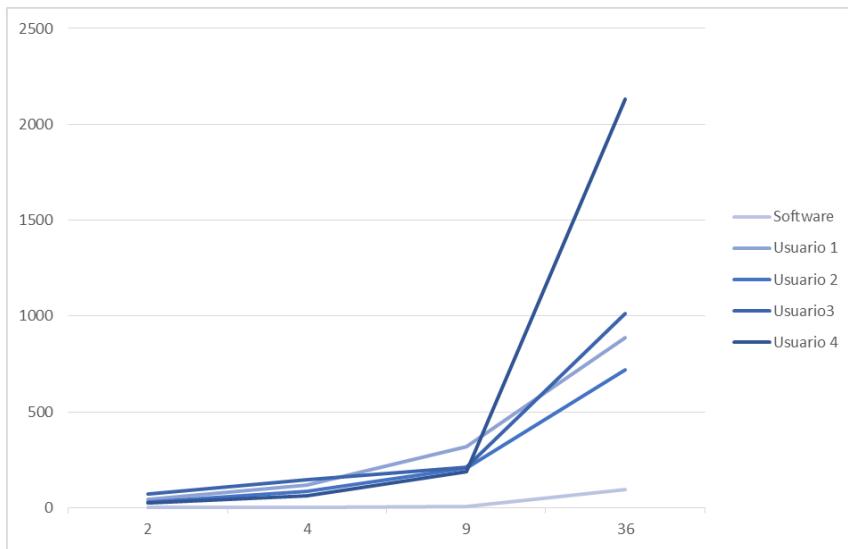
**Tabla 5.1:** Tiempo de ejecución del software.

Número de imágenes	Número de pruebas	Tiempo de ejecución
2	13	0.366seg
4	11	0.587seg
9	2	5.355seg
36	1	93.672seg

## 5.1. Porcentaje de mejora

Con los mismos datos obtenidos por el software se hizo una comparativa con diferentes individuos a los cuales se les pidió que realizaran la misma tarea usando las diferentes herramientas que hasta antes de la presente solución tenían. Se utilizaron soluciones tales como: *Microsoft Word*, *Microsoft Power Point* y *Photoshop*, el cronómetro comenzaba cuando tocaban la primer imagen y finalizaba una vez que todas fueran unidas.

Los resultados variaron de persona a persona, principalmente debido a la experiencia que el usuario pueda tener sobre el software utilizado, en la tabla 5.2, así como en la gráfica 5.2 se pueden apreciar los resultados arrojados mostrando una clara reducción de tiempo que va desde el 99 % para 2 imágenes, hasta un 93 % en el caso de las 36 imágenes.



**Figura 5.2:** Tiempos de ejecución entre Software y Usuarios. El eje vertical representa el tiempo medido en segundos, mientras que el eje horizontal indica el número de imágenes que fueron unidas.

**Tabla 5.2:** Tiempo de ejecución entre Software y Usuarios. Cada uno implementó un programa editor de imágenes a su elección para 2, 4, 9 y 36 imágenes.

Usuario	2 img <sup>1</sup>	4 img	9 img	36 img
Software de Tesis	0.366seg	0.587seg	5.355seg	93.672seg
U1 MS Word	46.266seg	121.151seg	321.484seg	889.446seg
U2 MS PowerPoint	31.343seg	88.087seg	208.901seg	719.212seg
U3 Photoshop	74.736seg	147.626seg	212.113seg	1012.312seg
U4 MS Word	23.137seg	62.215seg	187.211seg	2131.968seg

## 5.2. Errores de Unión

El núcleo del sistema está presente en el descriptor *ORB*, y el correcto funcionamiento del mismo viene dado por la naturaleza de las imágenes. Durante las experimentaciones realizadas se pudo ver, en varias ocasiones, uniones entre imágenes donde el resultado no era el esperado, y tras un análisis sobre dichos casos (que representaban en un principio

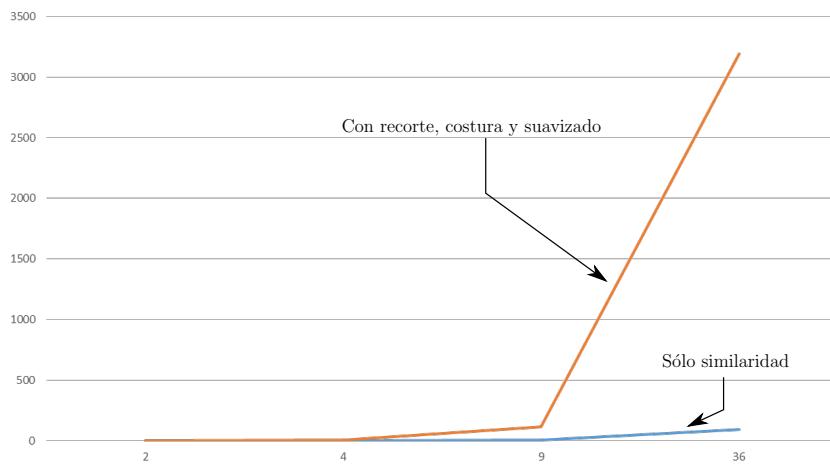
<sup>1</sup>Imágenes a mezclar

alrededor del 40 % de las muestras) se pudo llegar a las posibles causas que están causando dichos problemas (y el cómo evitarlas) .

- **El traslape entre imágenes es reducido:** Para detectar similitudes entre las diferentes imágenes de entrada hace falta encontrar una buena cantidad de puntos de interés. Cuando la intersección entre imágenes es inferior al 40 % en archivos de tamaño  $640p \times 480p$  la fiabilidad del algoritmo comienza a decaer. Esta intersección debe ser mayor a medida que las imágenes van reduciendo su tamaño.
- **No existe suficiente variación en el relieve de la imagen:** Si las imágenes en sí carece de cambios significativos a lo largo de las mismas, la detección que realice *ORB* decaerá debido a la falta de *características únicas*, es decir, a mayor ruido en la imagen (entendido éste como el cambio repentino de tonalidad, saturación o intensidad) la detección será realizada con mayor precisión.

### 5.3. Discusión

Si se toma la imagen de la figura 4.37 se puede apreciar a simple vista que el software no mezcla las imágenes como una sola, y se limita simplemente a interponerlas en un plano de tamaño indeterminado, lo que hace que resulte, en varios casos, visible a simple vista en dónde comienzan y terminan las imágenes ensambladas. Durante el desarrollo de la tesis existió la posibilidad de aplicar técnicas que permitan ofrecer una imagen más homogénea pero el tiempo consumido por su aplicación hacia inviable dicha “optimización” tal como se observa en la figura 5.3, ésto generó un dilema sobre el sacrificio de tiempo o calidad, generando una limitante al sistema.



**Figura 5.3:** Comparativa entre algoritmos. En naranja se muestra el mismo algoritmo al que se le aplicaron técnicas de recorte, costura y suavizado para generar una imagen panorámica homogénea. En azul, el mismo algoritmo sin dicho proceso (el cual fue el que se implementó finalmente).

Además *ORB* es un algoritmo que se puede utilizar en transformaciones de más de 4 *DoF*, y aunque el sistema desarrollado puede reconocer patrones en imágenes con dichas transformaciones, el cálculo de la matriz de similaridad implementará solo transformaciones de traslación, escalación y rotación, ésto por el enfoque del sistema, usualmente las imágenes obtenidas por un microscopio carecen de deformaciones que pueden existir en otros medios.

Ésto solo expone el hecho de que aún hay trabajo por realizar. Estas limitaciones ofrecen oportunidades de investigación a futuro que permitan entregar de manera rápida una imagen panorámica homogénea. Así mismo, aun cuando no forma parte del enfoque de la investigación, es posible incrementar grados de libertad realizando estimación de homografías, ampliando o modificando el área de trabajo a diferentes sectores de interés.

# Capítulo 6

## Conclusión

Esta tesis investigó y evaluó diferentes métodos y algoritmos que pueden ser usados para la unión de imágenes provenientes de microscopios ópticos. Como se observó a lo largo del documento, éste proceso involucra varios pasos y de no ser estudiados, evaluados y mejorados por separado, diferentes incrementos de tiempo habrían aparecido a lo largo de la aplicación que pueden afectar el rendimiento del sistema.

El alineamiento de imágenes implementó el algoritmo que mejor resultados dio en el momento de un grupo de 4 para la detección de similaridades (*Harris*, *SIFT*, *SURF* y *ORB*). Se observó como *Harris* ofrece una buena velocidad de detección de características, pero la falta de un descriptor, así como su vulnerabilidad frente a la escalación hizo que fuera descartado. De igual manera, *SIFT* y *SURF*, aunque buenas alternativas también, se observó la existencia de algoritmo protegido por patente, y el gasto que pueda implicar su uso obligó a la búsqueda de otra alternativa. Llegando a *ORB* y, aunque no es la única, por sí solo ofreció una solución óptima.

Tras la extracción de características, prosiguió el empate de las mismas mediante algoritmo de búsqueda de vecinos más cercanos donde fueron analizados al menos 2 opciones *knn por fuerza bruta y ANN*, y contrario a lo que pueda parecer, la fuerza bruta obtuvo una ventaja aprovechable por el sistema. Y con los datos arrojados, se prosiguió a la estimación de matriz de similaridad con el uso de *RANSAC* para poder obtener los parámetros de transformación y entregar la imagen panorámica.

El uso de estas técnicas fue enfocado a la reducción del tiempo a coste de otros parámetros (como homogeneidad) y demostraron ser una solución viable para la reducción de tiempos muertos en el uso de microscopios mientras el objetivo principal de un investigador sea la precisión y el tiempo.

# Bibliografía

- [1] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2011.
- [2] L. Zhang, B. Curless, and S. M. Seitz, “Spacetime stereo: Shape recovery for dynamic scenes,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Madison, WI, June 2003, pp. 367–374.
- [3] B. Moghaddam and A. Pentland, “Probabilistic visual learning for object representation,” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, Washington, DC, 1997, pp. 696–710.
- [4] e. a. Adam Coates, Blake Carpenter, “Text detection and character recognition in scene images with unsupervised feature learning,” in *ICDAR ’11 Proceedings of the 2011 International Conference on Document Analysis and Recognition*, Washington, DC, 2011, pp. 440–445.
- [5] M. Brown and D. G. Lowe, “Automatic panoramic image stitching using invariant features,” *International Journal of Computer Vision*, 2007.
- [6] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Proceedings of the International Conference on Computer Vision*, ser. ICCV ’99, vol. 2. Washington, DC, USA: IEEE Computer Society, 1999, pp. 1150–.
- [7] ——, “Distinctive image features from scale-invariant keypoints,” in *International Journal of Computer Vision*, vol. 60, Vancouver, B.C., Canada, 2004, pp. 91–110.
- [8] T. T. Herbert Bay and L. V. Gool, “Surf: Speeded up robust features,” in *ECCV (1)*, vol. 3951. Springer, 2006, pp. 404–417.
- [9] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, and B. K. U. Leuven, “Speeded-up robust features (surf),” *Computer Vision and Image Understanding*, vol. 110, pp. 346–359, June 2008.

- [10] R. Ortiz, “Freak: Fast retina keypoint,” in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 510–517.
- [11] S. Leutenegger, M. Chli, and R. Y. Siegwart, “Brisk: Binary robust invariant scalable keypoints,” in *Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2548–2555.
- [12] K. K. Ethan Rublee, Vincent Rabaud and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *ICCV ’11 Proceedings of the 2011 International Conference on Computer Vision*, ser. ICCV ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 2564–2571.
- [13] E. Rosten and T. Drummond, “Machine learning for high-speed corner detector,” in *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*, ser. ECCV ’06. Berling, Heidelberg: Springer-Verlag, 2006, pp. 430–443.
- [14] C. S. Michael Calonder, Vincent Lepetit and P. Fua, “Brief: Binary robust independent elementary features,” in *Proceedings of the 11th European Conference on Computer Vision: Part IV*. Berling, Heidelberg: Springer-Verlag, 2010, pp. 778–792.
- [15] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, 1998.
- [16] V. H. Milan Sonka and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 3rd ed. Thomson Learning, 2008.
- [17] J. B. E.H. Adelson, C.H. Anderson and P. Ogden, “Pyramid methods in image processing,” *RCA Engineer*, vol. 29, pp. 33–44, 1984.
- [18] E. H. A. Peter J. Burt, “The laplacian pyramid as a compact image code,” *IEEE Transactions on Communications*, vol. 31, pp. 532–540, 1983.
- [19] C. Tomasi, “Convolution, smoothing, and image derivates,” May 2003.
- [20] J. E. Solem, *Programming Computer Vision with Python*, 3rd ed. Beijing; Cambridge; Sebastopol [etc.]: O’Reilly, March 2012.

- [21] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference*, 2001, pp. 511–518.
- [22] K. G. Derpanis, “Integral image-based representations,” July 2007.
- [23] M. Avlash and L. Kaur, “Performances analysis of different edge detection methods on road images,” *International Journal of Advanced Research in Engineering and Applied Sciences*, vol. 2, pp. 27–38, June 2013.
- [24] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 3rd ed. Prentice-Hall, Inc, 2006.
- [25] M. Brennan, “Wavelets and signal processing,” November 2006.
- [26] D. Moreno, “mcbrief: un descriptor local de features para imagenes color,” Master’s thesis, Universidad Nacional de Rosario, Facultad de Ciencias Exactas, Ingeniería y Agrimensura, 2011.
- [27] K. Paudel, “Stitching of x-ray images,” Master’s thesis, UPPSALA UNIVERSITET, Institutionen för informationsteknologi, Department of Information Technology, Box 536, 751 21 Uppsala, 2012.
- [28] J. D. Mehta and S. G. Bhirud, “Image stitching techniques,” in *International Conference On Contours Of Computing Technology (THINKQUEST 2010)*, Mumbai, MH, 2010, pp. 74–80.
- [29] M. A. Fishler and R. C. Bolles, *Readings in computer vision: issues, problems, principles, and paradigms*. Morgan Kayfmann Publishers Inc., 1987.
- [30] C. Harris and M. Stephens, “A combined corner and edge detector,” in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151.
- [31] K. G. Derpanis, “The harris corner detector,” October 2004.
- [32] A. Vedaldi. (2007) Scale invariant feature transform (sift). [Online]. Available: <http://www.vlfeat.org/api/sift.html>
- [33] M. Brown and D. Lowe, “Invariant features from interest point groups,” in *Proceedings of the British Machine Vision Conference*. BMVA Press, 2002, pp. 23.1–23.10.

- [34] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *In VISAPP International Conference on Computer Vision Theory and Applications*, 2009, pp. 331–340.
- [35] M. Marius. (2014) Flann. [Online]. Available: <http://www.cs.ubc.ca/research/flann/>
- [36] opencv dev team. (2014) Flann. [Online]. Available: [http://docs.opencv.org/2.4/modules/video/doc/motion\\_analysis\\_and\\_object\\_tracking.html#estimaterigidtransform](http://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html#estimaterigidtransform)
- [37] P. J. Rousseeuw, “Least median of squares regression,” in *The American Statistical Association*, 1984, p. 388.
- [38] J. Steele and W. Steiger, “Algorithms and complexity for least median of squares regression,” in *Discrete Applied Mathematics*, 1986, pp. 93–100.
- [39] M. A. Fishler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, pp. 381–395, JUN 1981.
- [40] S. Choi, T. Kim, and W. Yu, “Performance evaluation of ransac family.” in *BMVC*. British Machine Vision Association, 2009.
- [41] A. Levin, A. Zomet, S. Peleg, and Y. Weiss, “Seamless image stitching in the gradient domain,” in *In Proceedings of the European Conference on Computer Vision*, 2006.