CIMAT

Centro de Investigación en Matemáticas, A.C.

Implementación y evaluación del performance del descriptor BRIEF en Android

REPORTE TÉCNICO

Que para obtener el grado de

Maestro en Ingeniería de Software

P r e s e n t a

Marcos Márquez Ovalle

Director(a) de Reporte Técnico

Dr. Jorge Roberto Manjarrez Sánchez

Zacatecas, Zacatecas., 31 de Octubre de 2012

Agradecimientos

A el CIMAT Unidad Zacatecas y a COZCyT por haberme otorgado la beca de colegiatura y la beca de manutención que me permitieron estudiar esta maestría.

A el personal docente y administrativo de CIMAT Unidad Zacatecas por su amabilidad y dedicación que tuvieron asía mi persona.

Al Dr. Jorge Roberto Manjarrez Sánchez

A mi familia y seres queridos, en especial a mi esposa e hijo, por su apoyo durante estos años de intenso trabajo.

RESUMEN

La visión por computadora trata de proporcionar los métodos a las computadoras para que puedan obtener, procesar e interpretar información visual usando dispositivos tales como cámaras. Uno de estos métodos consiste en identificar objetos en imágenes mediante la comparación de los puntos de interés de una imagen inicial con los obtenidos en imágenes posiblemente capturadas al instante o previamente almacenadas. De entre las muchas propuestas para hacer esto, BRIEF sobresale por su rapidez de cómputo, característica que lo hace atractivo de implementar en dispositivos móviles, los cuales no poseen las capacidades de cómputo de una estación de trabajo.

En este estudio describiremos en qué consiste y como se implementó el algoritmo BRIEF en Android. Se analizará el performance de nuestra implementación con las distintas variantes de tamaño de descriptores y se dará una propuesta de solución al problema de rotación y escala que tiene BRIEF. Plantearemos mejoras que pueden ayudar al rendimiento de la implementación. Y por ultimo comentaremos cuales pueden ser los trabajos futuros, con los que se puede continuar este estudio.

ABSTRACT

Computer vision research provides the algorithms which allow a computer to obtain, process and interpret visual information acquired through devices such as mobile's phone cameras. One of these algorithms is point matching and among the many proposals, BRIEF seems to be one of the most viable solutions from a computational cost point of view. This makes BRIEF interesting to use in mobile devices which have processing, storage and energy constraints.

In this work we describe the implementation of BRIEF on Android. We also provide detailed results of the evaluation of its performance and propose a solution to the rotation and scale flaws. Additionally we provide some clues to enhance performance and future development directions of this ongoing work.

Contenido

Índice de figuras

1. Introducción

El objeto de estudio es la implementación de un método para la comparación de puntos de interés en imágenes en un dispositivo móvil.

El problema es que este método requiere de varias etapas de procesamiento computacionalmente costosas y un dispositivo móvil tiene limitaciones en cuanto al poder de procesamiento y duración de la batería.

Una primera revisión del estado del arte nos permite identificar que BRIEF puede ser un método computacionalmente accesible para dispositivos móviles. Entonces, el objetivo es implementar BRIEF para Android y evaluar el tiempo de procesamiento requerido y la capacidad para reconocer objetos.

En este documento nos enfocaremos específicamente a 3 ramas de la visión por computadora que son: extracción de características, descripción de características y el procesamiento de esas descripciones al hacer comparaciones con 2 imágenes.

Para la extracción de características utilizaremos SURF, que es uno de los mejores extractores que existen en la actualidad. Como descriptor de características utilizaremos BRIEF, recientemente creado y que ha llamado la atención por su velocidad de procesamiento.

En las siguientes secciones ofrecemos primero una revisión de los conceptos necesarios para comprender el funcionamiento del método para la comparación de puntos de interés. En seguida detallamos la implementación de nuestra propuesta y finalmente presentamos pruebas de performance de nuestra implementación en Android de BRIEF.

2. Visión por computadora

La visión por computadora (también llamada visión artificial) es una rama de la inteligencia artificial. La visión por computadora es un campo que incluye métodos para la adquisición, procesamiento, análisis, y entendimiento de imágenes. [**1**]

"Como una disciplina de la ciencia, la visión por computador se refiere a la teoría detrás de los sistemas artificiales que extraer información de imágenes. Los datos de las imágenes pueden ser tomadas de muchas formas, secuencias de video, vistas de múltiples cámaras, o de datos multidimensionales tomados de un scanner." [**2**]

"Como una disciplina de la tecnología, la visión por computadora trata de aplicar sus teorías y modelos para la construcción de sistemas de visión por computadora. Ejemplos de aplicaciones de visión por computadora incluyen sistemas para:

- Control de procesos (robots industriales).
- Navegación (vehículos autónomos, robots móviles).
- Detección de eventos (vigilancia visual, conteo de personas).
- Organizar información (indexación de base de datos de imágenes y secuencia de imágenes).
- Modelar objetos o entornos (imagen médica, modelado topográfico).
- Interacción (interacción computadora-humano).
- Inspección automatizada (aplicaciones de manufactura)." [**2**]

La visión por computadora cubre el núcleo de las tecnologías de análisis automatizado que se utiliza en muchos campos.

En la visión por computadora existen varios factores que interfieren con el correcto procesamiento en cuanto al análisis de las imágenes como son:

- Escala
- Iluminación
- Perspectiva
- Rotación

Una pregunta a responder es ¿Qué es un punto de interés o una característica en una imagen? Una característica o punto de interés es una sección o punto de la imagen que sobresale a las demás secciones de la imagen.

2.1. Objetivos de la visión por computadora

Los objetivos típicos de la visión artificial incluyen:

- La detección, segmentación, localización y reconocimiento de ciertos objetos en imágenes (por ejemplo, caras humanas).
- La evaluación de los resultados (por ejemplo, segmentación, registro).
- Registro de diferentes imágenes de una misma escena u objeto, es decir, hacer concordar un mismo objeto en diversas imágenes.

- Seguimiento de un objeto en una secuencia de imágenes.
- Mapeo de una escena para generar un modelo tridimensional de la escena; este modelo podría ser usado por un robot para navegar por la escena.
- Estimación de las posturas tridimensionales de humanos.
- Búsqueda de imágenes digitales por su contenido.

2.2. Tareas que realiza un sistema de visión por computadora.

Comúnmente las funciones que son encontradas en un sistema de visión por computadora son:

2.2.1. Obtención de imagen.

Esta tarea se encarga de abstraer del mundo real, una imagen, y convertirá a una imagen digital. Esto puede ser posible gracias a uno o varios sensores de imagen, ya sean sensores de rango, dispositivos de tomografía, radar, ultrasónicos, etc. Dependiendo de estos sensores, los datos de imagen resultante es una imagen ordinaria 2D, 3D, o una secuencia de imágenes. “El valor del pixel típicamente corresponden a la intensidad de luz en una o varias bandas espectrales (imágenes grises o imágenes de color), pero también puede estar relacionada con diversas medidas físicas, tales como la profundidad, la absorción o reflexión de las ondas sónicas o electromagnética, o resonancia magnética nuclear.” [**3**]

2.2.2. Pre-procesamiento

“Antes de que un método de visión por computadora pueda ser aplicado a los datos de imagen con el fin de extraer alguna pieza específica de información, es necesario procesar los datos con el fin de asegurar que satisface ciertas suposiciones implícitas en el método. Ejemplos de esto es:

- Re-sampling con el fin de asegurar que el sistema de coordenadas de la imagen es correcta.
- La reducción de ruido con el fin de asegurar que el ruido del sensor no introduzca información falsa.
- El realce de contraste para asegurar que la información pertinente pueda ser detectada.
- Representación escala-espacio para mejorar las estructuras de imágenes a escalas adecuadas.” [**4**]

2.2.3. Extracción de características

Obtener las características o puntos de interés más relevantes de una imagen.

Para la extracción de características existen varios tipos de algoritmos tales como:

Edge detection

Es un algoritmo para el procesamiento de imágenes, sobre todo en las áreas de detección de características y extracción de características, que reduce la cantidad de datos a procesar, mientras que al mismo tiempo preserva la información de la imagen. Las

características o puntos de interés que obtiene es donde los cambios de brillo de la imagen son drásticos o tiene discontinuidades [**5**]. Algunos algoritmos de este tipo son:

- Canny-Deriche
- CORF
- Differential
- Sobel
- Prewitt
- Roberts Cross

Corner detection

Al igual que Edge es un algoritmo utilizado en los sistemas de visión por computadora para extraer ciertos tipos de características e inferir el contenido de una imagen. Está basado principalmente en la detección de esquinas a partir de un pixel pivote, buscando ángulos en los pixeles vecinos. Corner detection se utiliza con frecuencia en la detección de movimiento, registro de imagen, seguimiento por cámara, mosaicos de imágenes, composición panorámica, modelado 3D, reconocimiento de objetos y detección de puntos de interés [**6**]. Estos algoritmos son una muestra de este tipo de detector:

- Harris operator
- Shi and Tomasi
- Level curve curvature
- SUSAN
- FAST

Blob detection

El método blob detection al igual que Edge y Corner, tienen por objetivo la detección de puntos o regiones en la imagen, que difieren en propiedades como el brillo o el color en comparación con el entorno. “Hay dos clases principales de detectores de blob: los métodos diferenciales basadas en expresiones derivadas y métodos sobre la base de extremos locales en el panorama de la intensidad. Con la terminología más reciente utiliza en el campo, estos operadores también puede ser denominado como operadores de punto de interés.” [**7**] Los siguientes son algoritmos de blob detection:

- Laplacian of Gaussian (LoG)
- Difference of Gaussians (DoG)
- Determinant of Hessian (DoH)
- Maximally stable extremal regions
- PCBR

Scale Invariant Detectors

Algunos algoritmos son invariantes a cambios de escala significativas, tal es el caso de los algoritmos Scale Invariant. Típicamente, estas técnicas suponer que el cambio de escala es la misma en todas las direcciones, aunque presentan alguna robustez a débiles deformaciones. [**8**]

Affine Invariant Detectors

Puede verse como una generalización de Scale Invariant Detectors. En el caso de una transformación afín la escala puede ser diferente en cada dirección. El escalado no uniforme tiene una influencia sobre la localización, la escala y la forma de una estructura local. Por lo tanto, los detectores Scale Invariant fallan en el caso de importantes transformaciones afines. [**8**] Estos son algunos algoritmos de este tipo:

- Affine shape adaptation
- Harris affine
- Hessian affine

2.2.4. Detección/segmentación

"En este punto del procesamiento se toma una decisión acerca de qué puntos de la imagen o regiones de la imagen son relevantes para su posterior procesamiento.

- La selección de un conjunto específico de puntos de interés
- La segmentación de las regiones de la imagen de uno o múltiples, que contienen un objeto específico de interés." [**4**]

2.2.5. Descripción de características

El contenido de las imágenes es convertido a características, que se encuentran localizadas en coordenadas, y estas son invariantes a escala, rotación o a cambios en los puntos de vista y cambios en la iluminación. Una descripción de una característica, es el valor que se obtiene de aplicar varios cálculos a un punto de interés que se obtuvo de la extracción de características en el paso anterior, este valor puede variar en su tipo dependiendo del descriptor, puedes ser un entero, una cadena de caracteres, un número binario o una matriz de números, etc.

Algunos de los más importantes descriptores son:

Scale Invariant Feature Transform (SIFT)

El algoritmo fue publicado por David Lowe in 1999 y patentando por la universidad de British Columbia. [**9**] Sus características principales son:

- Invariante a escala.
- Invariante a rotación.
- Invariante a la iluminación.
- Invariante a los puntos de vista, para pequeños cambios.
- Rápida detección de puntos de interés.
- Distintivas descripciones para los puntos de interés.
- Debido a punto anterior la velocidad de comparación es mayor.

Speeded-Up Robust Features (SURF)

Fue presentado en 2006 por Herber Bay, quien se inspiro a partir de SIFT. El autor proclama que SURF es varias veces más rápido que SIFT y que conserva la robustez de este. [**10**] Las principales características son:

- Rápida detección de puntos de interés.
- Distintivas descripciones para los puntos de interés.
- Debido a punto anterior la velocidad de comparación es mayor.
- Invariante a comunes trasformaciones en la imagen.
    - Rotación.
    - Escala.
    - Iluminación.
    - Pequeños cambios en la perspectiva.
- SURF describe características más rápido que SIFT.
- SURF no es tan bueno como SIFT para variaciones de iluminación y perspectiva.

Histogram of Oriented Gradients (HOG)

La técnica cuenta las apariciones de orientación gradiente en secciones localizadas de la imagen. Fue presentado por Navneet Dalal y Bill Triggs en Junio del 2005, y el trabajo se centra sobre el problema de detección de peatones en las imágenes estáticas, aunque desde entonces se expandió para incluir a sus pruebas de detección de personas en el cine y vídeo, así como a una variedad de animales comunes y vehículos en las imágenes estáticas. [**11**]

Gradient Location and Orientation Histogram (GLOH)

Algoritmo propuesto como una extensión de SIFT, que demostrado ser superior y con una robustez similar. [**12**]

Local Energy based Shape Histogram (LESH)

"El descriptor LESH se basa en el análisis de la energía local de las características de la percepción, la cual es obtenida por el filtro de Gabor." [**13**] Las características que se obtienen de LESH se pueden utilizar en aplicaciones como recuperación de imágenes basada en la forma, la detección de objetos, y formular estimaciones. [**13**]

Binary Robust Independent Elementary Features (BRIEF)

Un reciente algoritmo para la descripción de características, donde la representación de características es a manera de cadenas de caracteres. Según los autores demostró ser altamente descriptivo, aún utilizando el tamaño mínimo (esto se verá en secciones posteriores) para describir puntos de interés y por lo cual es sumamente eficiente para la comparación de características [**14**]. Algoritmo relativamente nuevo ya que se presentó en 2010.

Los autores declaran que es más rápido que SURF y la variante U-SURF y que el grado de reconocimiento es similar o mejor que estos dos.

2.2.6. Procesamiento de alto nivel

"En este paso las entradas son típicamente un pequeño conjunto de datos, por ejemplo un conjunto de puntos o una región de la imagen que se supone que contiene un objeto específico. Algunas de las actividades que se realizan aquí son:

- La verificación de que los datos satisfacen las hipótesis específicas basadas en el modelo y la aplicación.
- Estimación de los parámetros específicos de la aplicación, como objeto de proponer o rechazar tamaño.
- El reconocimiento de la imagen. La clasificación de un objeto detectado en diferentes categorías.
- Registro de imágenes**.** La comparación y la combinación de dos puntos de vista diferentes del mismo objeto.” [**4**]

2.2.7. Toma de decisiones

Es la última de las etapas en la visión por computadora donde se toman las decisiones a partir de la información recolectada o generada en etapas anteriores [**3**], por ejemplo:

- “Pasa/no pasa en las aplicaciones automatizada de la inspección.
- Concuerda/no-concuerda en aplicaciones de reconocimiento.
- Marcar para revisión humana en médicas, militares, de seguridad y en aplicaciones de reconocimiento.” [**3**]

3. Implementación de la aplicación

La dificultad de compilar, implementar y aprender a utilizar OpenCV, las múltiples versiones liberadas en periodos cortos de tiempo (menos de un mes), así como los bugs que esto trae consigo, nos orilló a crear nuestra propia implementación de BRIEF en Android, sin la necesidad de utilizar bibliotecas externas a este. Con el único propósito de que nuestra implementación de BRIEF pueda ser implementada por cualquier dispositivo con Android sin pasar por todos problemas que OpenCV tiene.

En este capítulo hablaremos acerca de las etapas en las que está compuesta nuestra aplicación y su la implementación, que clases intervienen y que funciones son utilizadas o llamadas en cada etapa.

En este capítulo tratamos de mantener un equilibrio entre lo teórico y lo práctico, es por eso que en cada etapa, se explica, primeramente la teoría y a continuación se da una breve explicación de cómo esta teoría es implementada en código en nuestra aplicación.

En la figura 3.1, se muestran las etapas que se siguen durante la implementación de nuestra aplicación y se explican a detalle en las secciones subsecuentes, en donde se describirá en qué consisten y que factores influyeron en el diseño de cada etapa.

Como puede observase en la figura 3.1, nuestra implementación está cimentada básicamente en SURF y BRIEF. SURF por ser muy robusto en cuanto a extracción de características y BRIEF por ser rápido para describir características.

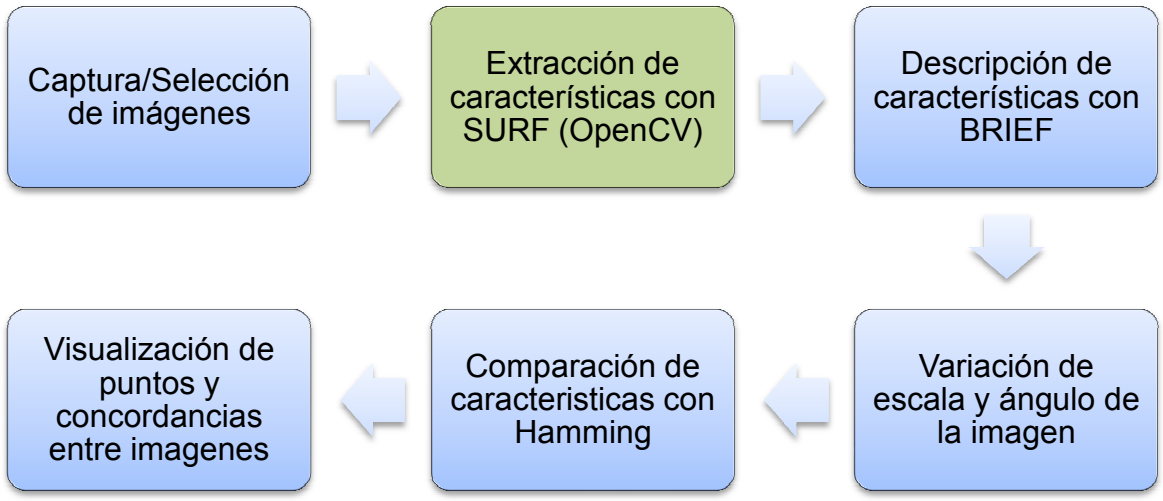

Figura 3.1 Visión general de la implementación

La figura 3.1 ilustra en color azul las etapas que desarrollamos completamente, las etapas de color verde, son etapas en las que fuimos apoyamos de librerías ya existentes.

El diagrama de clases de nuestra aplicación se muestra en la figura 3.2.

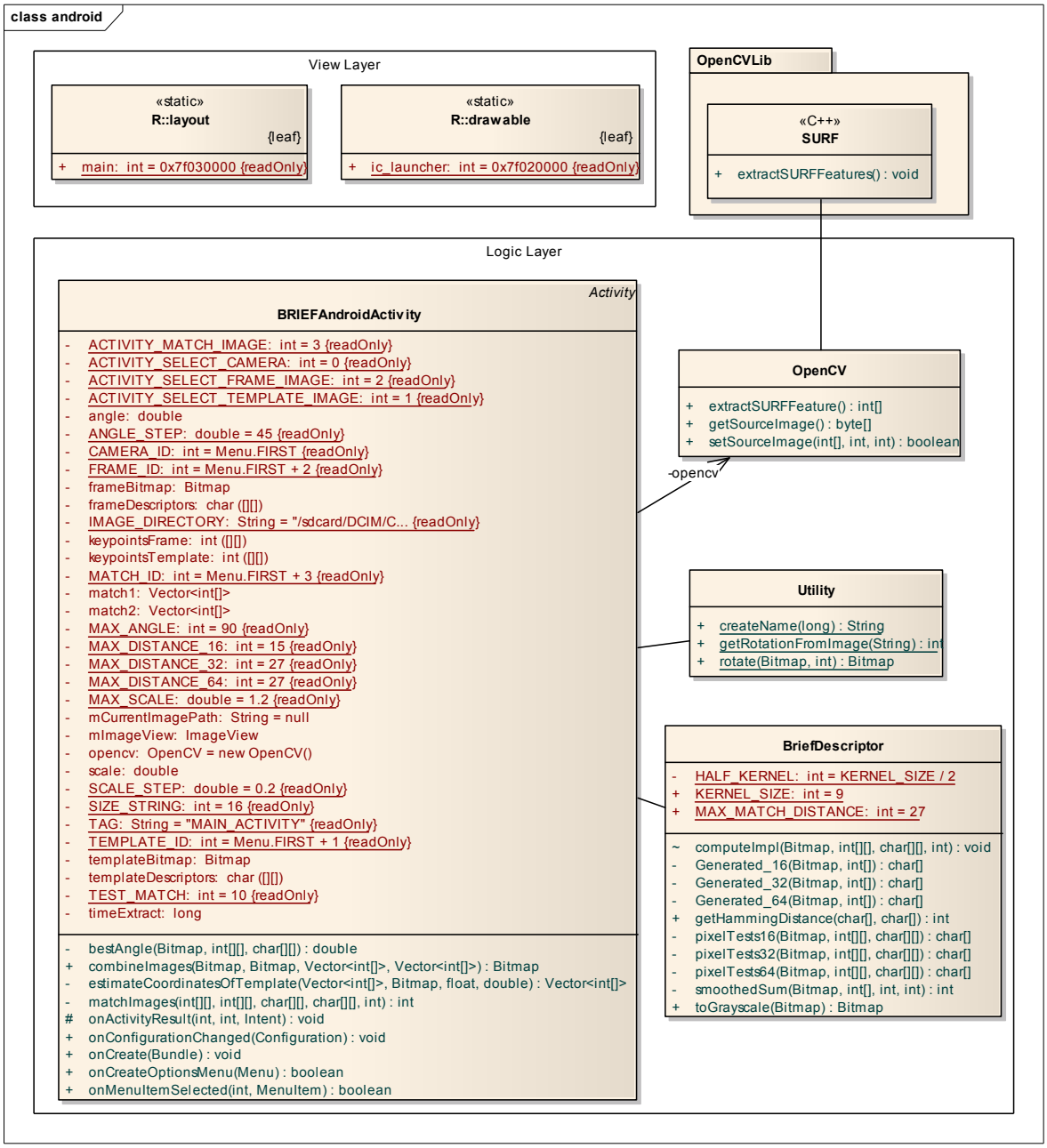

Figura 3.2 Diagrama de clases de aplicación

Está conformado básicamente por 2 capas: vista y lógica. A grandes rasgos, en la capa de vista sólo se encuentra un lienzo donde se dibujará las imágenes a comparar, y donde se ilustrarán las concordancias. En la capa de lógica es donde se hace la extracción, descripción y mapeo de las imágenes. El paquete de OpenCVLib es la librería de OpenCV que contiene varios de los algoritmos de visión que actualmente existen, y que serán mencionados en la sección siguiente. En las siguientes secciones se detalla la implementación de cada una de las etapas de la figura 3.1.

3.1. OpenCV

Cabe hacer un paréntesis para tomar un punto de suma importancia en nuestro estudio, para que sean comprendidas de mejor manera las siguientes etapas es necesario familiarizarse con el termino de OpenCV.

¿Qué es OpenCV? “OpenCV es Open Source Computer Vision Library, que son librerías software de código abierto para visión por computadora y machine learning. OpenCV fue construida para proporcionar una infraestructura común para aplicaciones de visión por computadora y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia BSD, OpenCV hace que sea fácil para las empresas utilizar y modificar el código.” [**15**]

“La librería contiene más de 2500 algoritmos, que incluyen un conjunto completo de algoritmos de ambos, visión por computadora y machine learning. Estos algoritmos pueden ser utilizados para la detección y reconocimiento de caras, identificación de objetos, clasificación de acciones humanas en video, seguimiento de cámara, seguimiento de objetos, extracción en 3D de objetos, unir imágenes para producir una imagen de alta resolución de una escena entera, buscar imágenes similares a partir de una base de datos de imagen, eliminar los ojos rojos de las imágenes tomadas con el uso de flash, seguir los movimientos de los ojos, reconocer paisajes y establecer indicadores para superponer con la realidad aumentada, etc.” [**15**]

La figura 3.3 muestra una visión general de las principales librerías que incluye OpenCV.

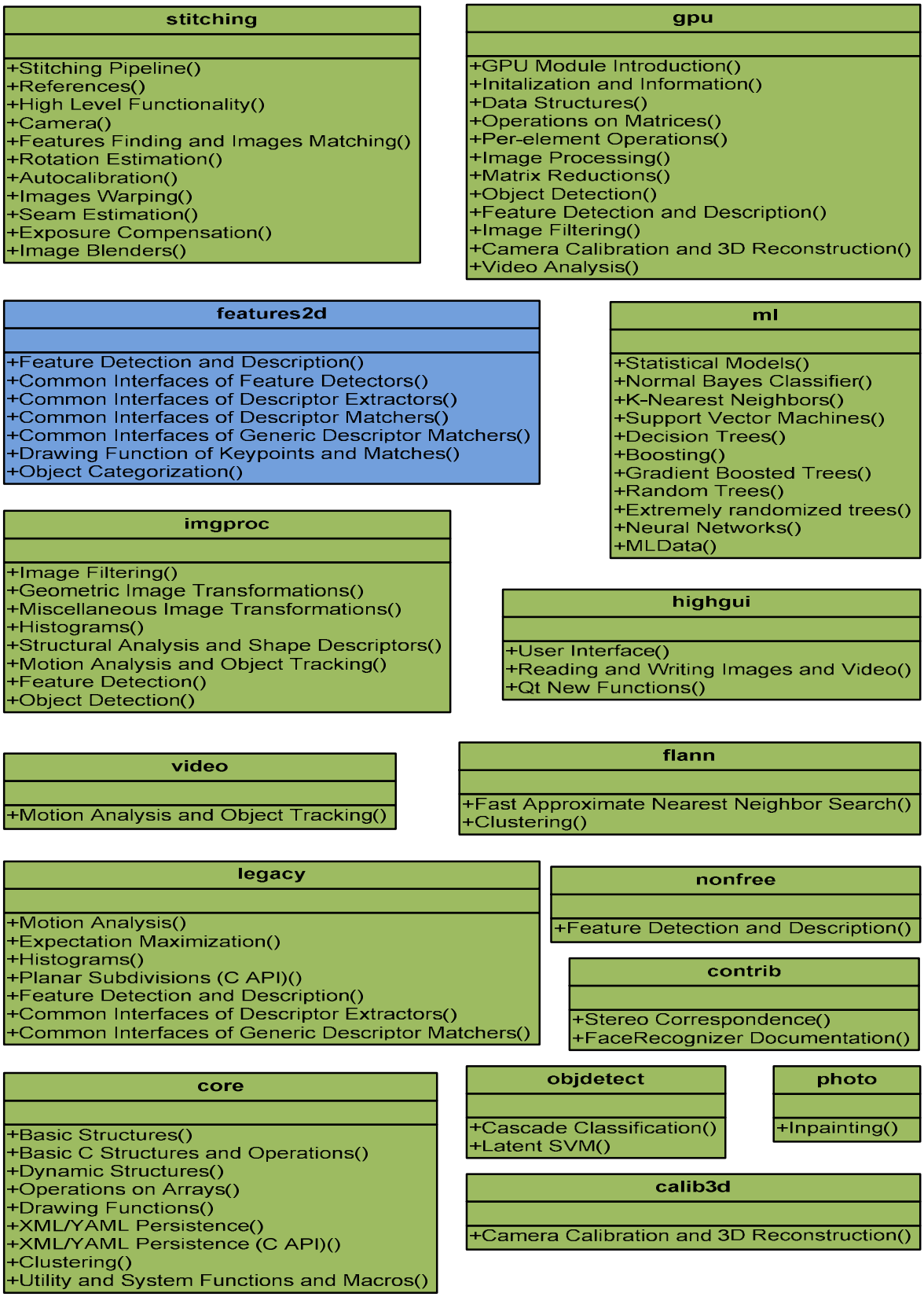

Figura 3.3 Principales librerías en OpenCV

Se hace referencia a OpenCV en este estudio principalmente por dos razones, la primera, porque será utilizado por nuestra aplicación, y la segunda, para mostrar en que parte de OpenCV podría encajar nuestra aplicación. Nuestra implementación encajaría en la librería de color azul **features2d** que es donde se encuentra los algoritmos de extracción, descripción y comparación de características.

3.1.1. Features2d

En esta librería se encuentran los algoritmos de extracción de características, descripción de características, comparación y graficar características, descripción de objetos.

En la extracción de características podemos encontrar los siguientes algoritmos:

- SIFT
- SURF
- FAST
- MSER
- ORB
- FREAK
- STAR
- GFTT
- HARRIS
- Dense
- SimpleBlob

Para la descripción de características se tiene:

- SIFT
- SURFT
- ORB
- BRIEF

Y para la comparación de características existentes en Feature2d:

- BruteForce
- BruteForce-L1
- BruteForce-Hamming
- BruteForce-Hamming-L2
- FlannBased

Como puede observarse, existen varios algoritmos para las etapas de comparación de imágenes. Estás pueden ser intercambiadas, por ejemplo, podemos tener SIFT como extractor, y SURF como descriptor y FlannBased como comparador de descripciones. Lo importante aquí es saber cuáles son las combinaciones que nos darán mejor resultado en cuanto a performances y exactitud, dependiendo de la aplicación que se le dará.

3.2. Captura/selección de imágenes

En esta etapa es donde se obtiene la imagen mediante la cámara del dispositivo o desde la galería de imágenes. Una vez que tenemos la imagen, se le hace el cálculo del tamaño, y la imagen se pasa a escala de grises. La transformación a escalas de grises es necesaria ya que así, la intensidad de los colores de la imagen se reduce a solo dos tonalidades blanco y negro en cada pixel, esto ayuda a atenuar que las fluctuaciones de iluminación sean un factor al momento de identificar los puntos de interés, la conversión de la imagen a escala de grises se usa solo para la extracción de características con SURF (ver sección extracción de características con SURF).

Todo esto se hace con las utilerías existentes en Android sin recurrir a librerías externas.

Para este estudio utilizaremos la imagen estándar utilizada por OpenCV para hacer sus pruebas de performance, la cual se muestra en la figura 3.4.

Figura 3.4 Lena [15]

Las clases y funciones de código utilizados para esta tarea se muestran en la figura 3.5.

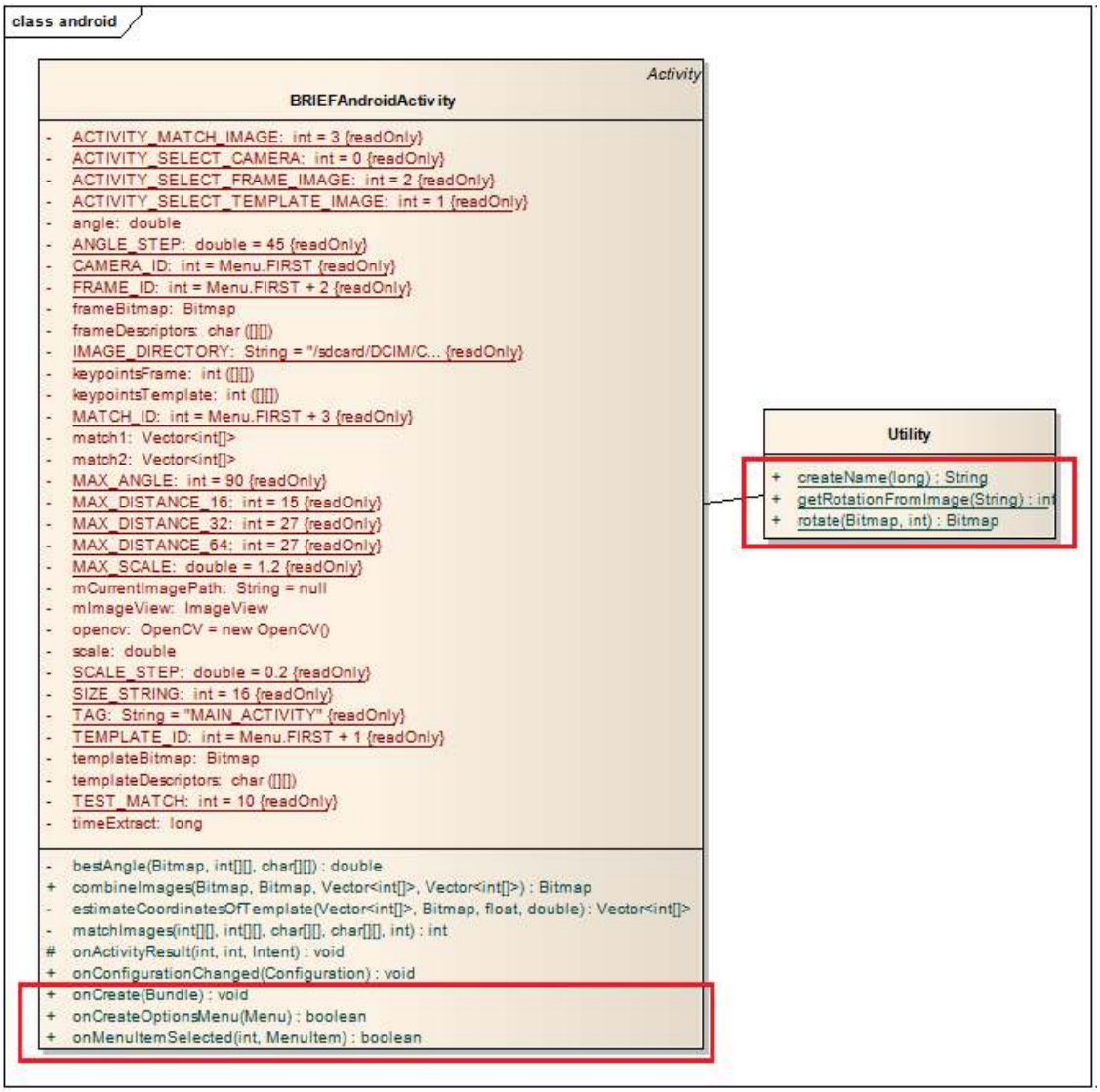

Figura 3.5 Diagrama de clases para captura de imágenes.

La clase BRIEFAndroidActivity, que es la encargada de gestionar todas las peticiones que el usuario hace por medio del menú que se crea en la función **onCreateOptionsMenu**. **onCreate** y **onCreateOptionsMenu** son funciones que se ejecutan al instanciar la clase de manera automática y no es necesario hacer llamado a estas desde alguna otra clases. La función **onMenuItemSelected** es ejecutada en cada ocasión que el usuario seleccione algún menú que se creó en la función **onCreateOptionsMenu**, el menú que dispara la selección de una imagen de la galería es "*Camara*", que es manejada con intents que ya posee Android. Un intent es una ejecución de un módulo ya existente en Android, por ejemplo el módulo de mensajes, módulo de agenda, de calculadora, galería de imágenes, etc.

3.3. Extracción de características con SURF

En esta etapa hacemos uso de la librería de OpenCV (ver sección OpenCV) para la extracción de las características o puntos de interés con SURF.

Las razones por las que se seleccionó SURF son:

- La robustez y capacidad para extraer puntos de interés.
- Por ser uno de los mejores calificados en el mundo de la extracción de características.
- Por contar ya con la implementación del algoritmo de SURF en Android (implementación que se encuentra a en forma de tutorial en la documentación de OpenCV).

Como se había mencionado las librerías de OpenCV del descriptor SURF están hechas en C y C++, por lo que es necesario la utilización de Java Native Interface (JNI) [**16**] que es un framework de programación que permite que un programa escrito en Java y ejecutado en la máquina virtual java (JVM) pueda interactuar con programas escritos en otros lenguajes como C, C++ y ensamblador. Otra herramienta importante para la utilización de esta librería es el NDK: un conjunto de herramientas que permite incorporar los componentes que hacen uso de código nativo en las aplicaciones de Android.

Debido a que las librerías están en C y C++ nosotros optamos por hacer uso de NDK para poder comunicarnos y utilizar las librerías de OpenCV.

La figura 3.6 muestra las clases y funciones que son utilizadas para realizar esta parte de la implementación:

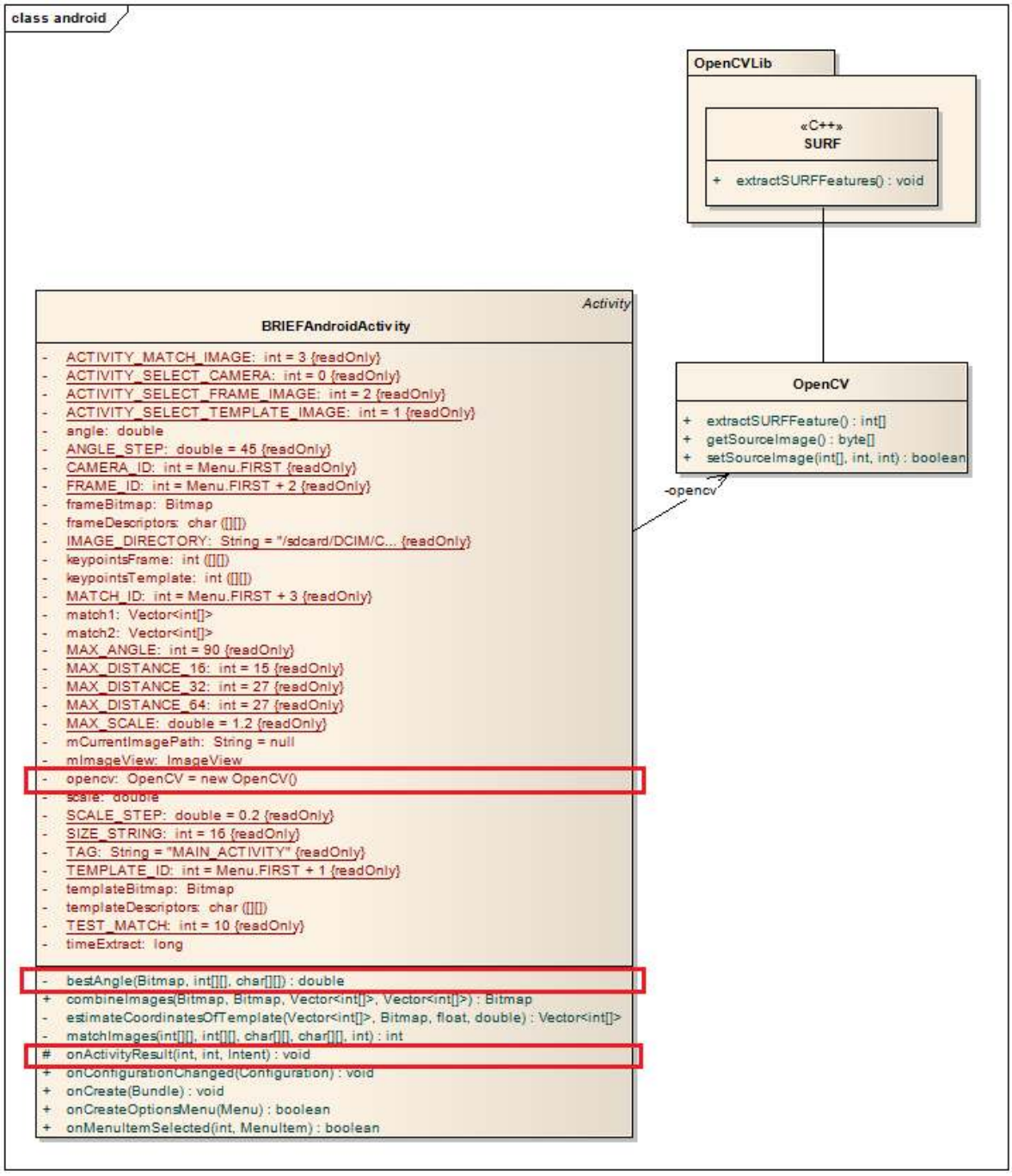

Figura 3.6 Diagrama de clases para SURF

Como se puede apreciar en la figura 3.6, se hace una instancia de la clase de OpenCV, que es la clase que se encarga de comunicarse con la librería de OpenCVLib. La clase OpenCV contiene tres funciones **extractSURFFeatures**, **getSourceImage** y **setSourceImage**.

setSourceImage que es la encargada de almacenar el bit de mapas de la imagen de la cual SURF va a extraer la características y que será utilizado por extractSURFFeatures.

getSourceImagen, que se encarga de obtener la imagen con los puntos de interés graficados en la imagen.

En esta etapa de la implementación la función más importante es esta **extractSURFFeatures**. Esta función es la encargada de llamar a la librería de SURF, que obtendrá los puntos de interés de la imagen seleccionada. El resultado que arroja la librería es un tipo de dato propio de OpenCV, por consiguiente es necesario cambiarlo a un tipo de dato que pueda ser transportado a Android sin ningún problema, como es el caso de una matriz de enteros. Esta matriz nos proporcionará las coordenadas de los puntos de interés encontrados.

La figura 3.7 muestra la selección de 5000 puntos de interés (puntos rojos en la imagen) que detecta SURF, y para este caso toma alrededor de 16 segundos.

Figura 3.7 Imagen con puntos de interés.

3.4. Descripción de características con BRIEF

Este es el punto más importante de nuestro estudio, la implementación de BRIEF para descripción de características.

El algoritmo BRIEF está compuesto principalmente por tres pasos que aplican sus respectivas formulas, pero antes de continuar con estos pasos es necesario explicar algunos términos.

KERNEL: es la longitud de cadena de un número binario, por ejemplo para este estudio se utilizó un Kernel de 8, un número binario con esta longitud tendría un valor parecido a 11101110, los valores para una cadena de bits de longitud 8 convertida a decimal es de 0 a 255, que son los colores que puede tener un pixel en una escala de grises.

NO, NE, SO, SE: Son los pixeles que se encuentra al Noroeste, Nordeste, Suroeste y Sureste del pixel al que se desea obtener la intensidad del color.

A continuación se explicarán los pasos a seguir para obtener una descripción BRIEF:

Paso 1. Calcular la intensidad de dos puntos p y z con coordenadas vectoriales de la imagen x, y. Tómese como ejemplo el ilustrado por la figura 3.8, donde

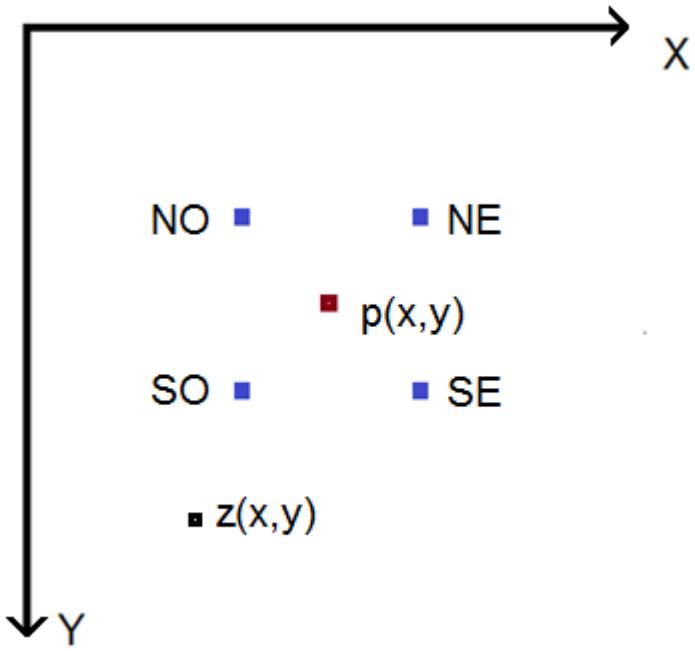

Figura 3.8 Gráfica de intensidad de color.

$$Para\ nuestro\ estudio\ KERNEL = 8$$

$$HALF\_KERNEL\ = KERNEL/2$$

$$NE_{xy} = ((p_x + z_x) + HALF\_KERNEL + 1, (p_y + z_y) + HALF\_KERNEL + 1)$$

$$NO_{xy} = ((p_x + z_x) - HALF\_KERNEL, (p_y + z_y) + HALF\_KERNEL + 1)$$

$$SE_{xy} = ((p_x + z_x) + HALF\_KERNEL + 1, (p_y + z_y) - HALF\_KERNEL)$$

$$SO_{xy} = ((p_x + z_x) - HALF\_KERNEL, (p_y + z_y) - HALF\_KERNEL)$$

Donde los puntos NE, NO, SE, SO son equidistantes.

Entonces para obtener la intensidad de z, con relación a p, denotada como $I(p,z)$ se deben sumar el valor de los pixeles en los cuadrantes como se muestra en la formula siguiente:

$$I(p,z) = NE_{xy} - NO_{xy} - SO_{xy} + SE_{xy}$$

Paso 2. Una vez que tenemos la intensidad de p con respecto de z, tenemos que calcular test (τ) de p con respecto a los puntos x,y podremos aplicar la formula:

$$\tau(p;x,y) = \begin{cases} 1, & if\ I(p,x) < I(p,y) \\ 0, & otherwise \end{cases}$$

Paso 3. Armar la cadena binaria de BRIEF

$$\sum_{1 \le i \le nd} 2^{i-1}\, \tau(p; x, y)$$

Donde, para nuestro estudio utilizamos una longitud de cadena BRIEF nd=16, 32 y 64, ya que son los valores que dan un valor intermedio entre velocidad y precisión [**14**].

Por ejemplo, si *nd* es igual a 16, el resultado que nos arrojará la formula es una cadena de 16 caracteres. Y cada carácter es representado por un valor ASCII del 0 al 255, porque los valores que arroja τ(p;x,y) es un 0 ó 1, y dado que nuestro Kernel es 8, tendríamos un binario de 8 posiciones y quedaría un valor parecido a:

$$nd_i \; binario = 0100\;1101$$

Si el valor de nd_i se pasa a valor decimal el resultado nos daría:

$$nd_i \; decimal = 77$$

Y por ultimo si nd_i se convierte a ASCII:

$$nd_i \; ASCII = M$$

Entonces la descripción para un punto de interés estaría representada de la siguiente forma:

$$nd = M\%\#(=!1?lÑ|:Z<@s$$

Esto es para nd_{size} = 16. Si queremos que nuestra descripciones sean más exactas, tendremos que aumentar el tamaño de nd.

La implementación en código de esta sección es de suma importancia, la figura 3.9 muestra que clases y funciones son utilizadas.

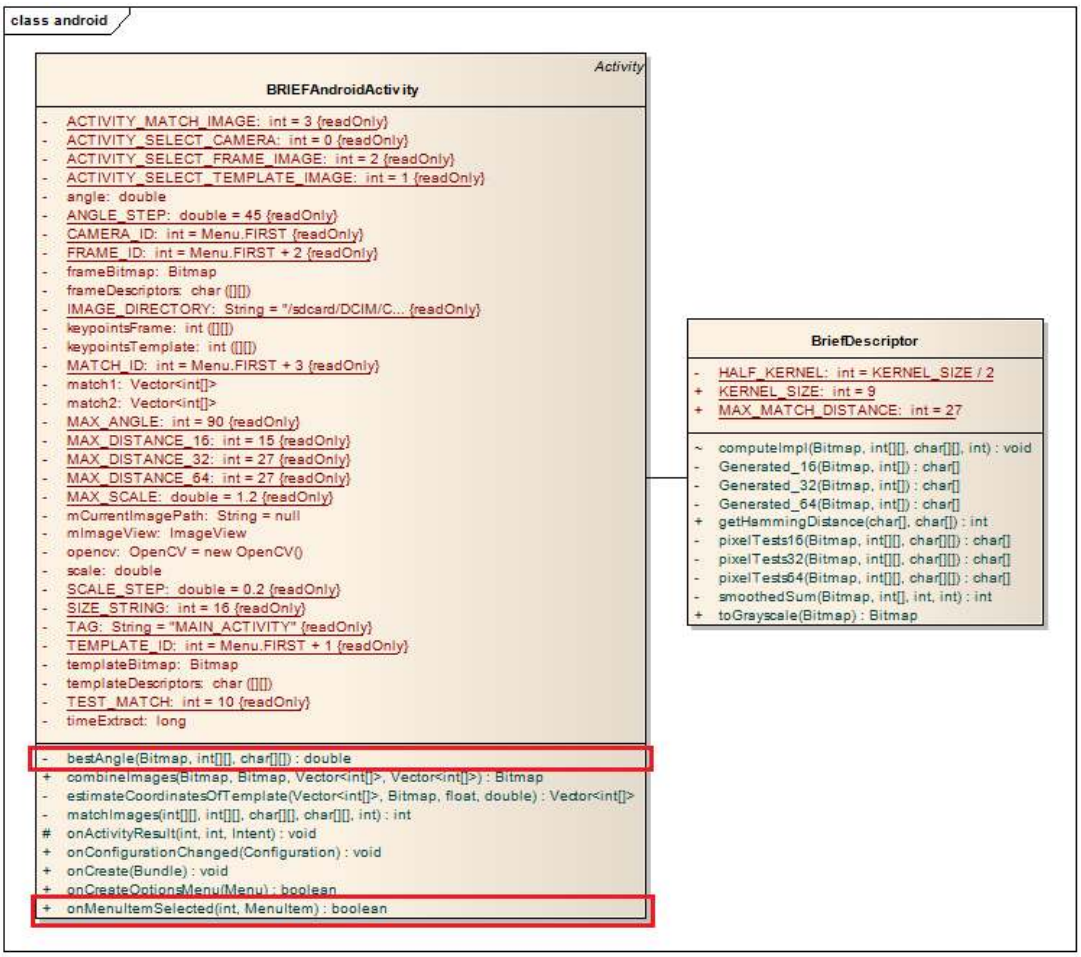

Figura 3.9 Diagrama de clases para BRIEF

Al igual que las etapas anteriores, esta comienza desde el **onMenuItemSelected**, la cual en el caso de la descripción de puntos, puede ser al momento de seleccionar un Template o Frame, en donde se hace la instancia a la clase de **BriefDescriptor**.

La clase **BriefDescriptor**, es la que implementa todas las formulas que se han descrito en esta sección. El método principal en esta clase es **computerImpl**, el cual recibe como parámetros, la imagen, los puntos a describir, y la descripción para estos puntos, y el número de nd que se utilizará. Cabe mencionar que todos los parámetros son pasados por referencia ya que en Java todos los objetos son pasados por referencia a excepción de los valores primitivos. Es por esta razón que este método no regresa ningún valor, ya que todos se almacena en el parámetro de description.

Haciendo la equivalencia de funciones contra formulas quedaría de la siguiente forma:

smoothedSum = $I(p,z) = NE_{xy} - NO_{xy} - SO_{xy} + SE_{xy}$

Generatednd = $\tau(p;x,y) = \begin{cases} 1, & if\ I(p,x) < I(p,y) \\ 0, & otherwise \end{cases}$

pixelTestsnd = $\sum_{1 \le i \le nd} 2^{i-1}\ \tau(p;x,y)$

3.5. Variación de escala y ángulo de la imagen

Ya que BRIEF es un descriptor al cual le afectan tres factores importantes que son: la rotación mayor a 20˚ [**17**], la escala y la perspectiva, se optó por las siguientes soluciones:

3.5.1. Rotación de la imagen

Para atacar este importante problema que afecta a la descripción de los puntos de interés, se llego a la solución más simple posible, rotar la imagen frame (frame es la segunda imagen, es la imagen a la que se hacen variaciones). Esto queda explicado en la figura 3.8.

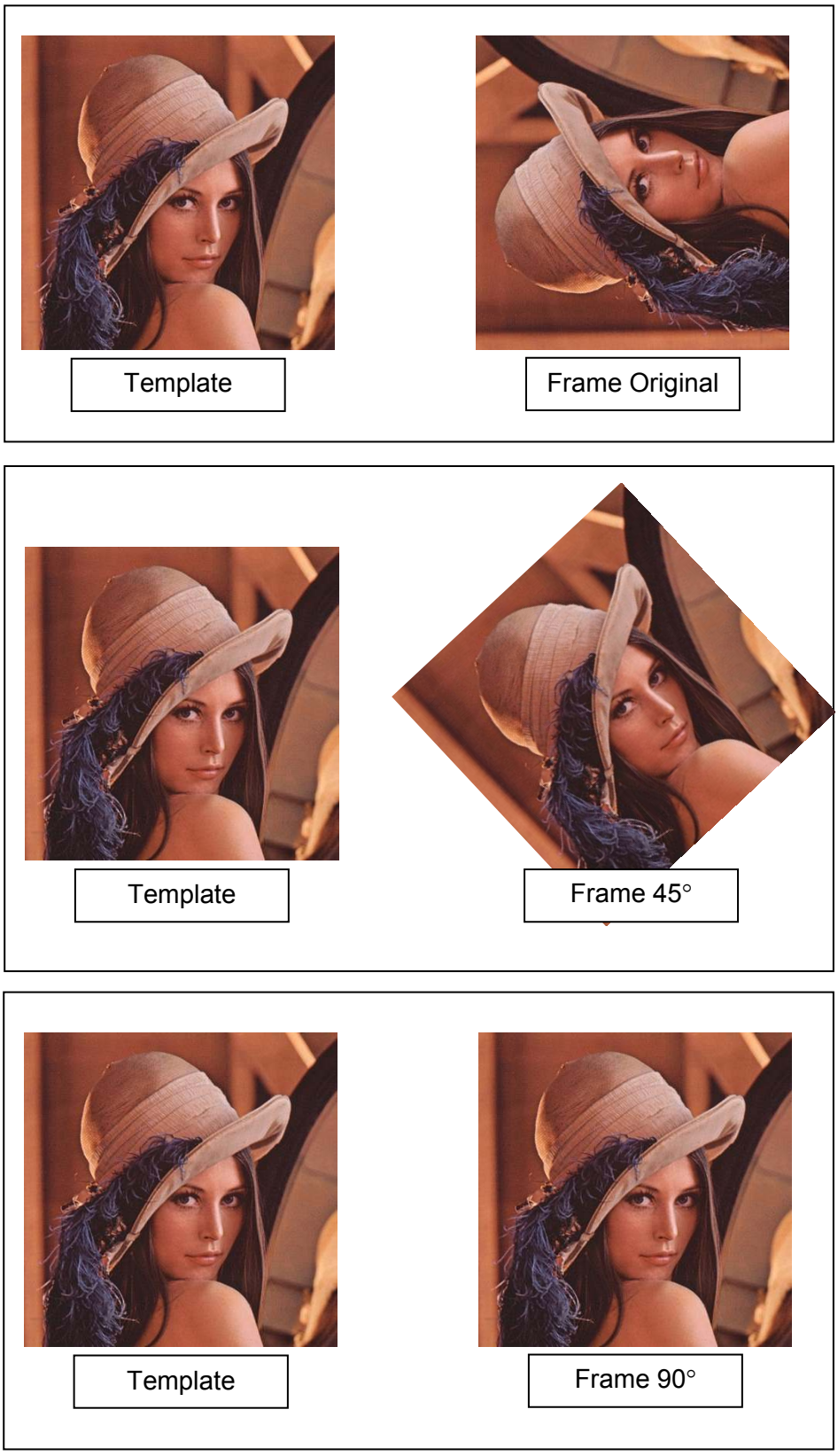

Figura 3.10 Proceso de rotación

Pero rotar las imágenes para hacer coincidir con el template (es la imagen principal a encontrar en la imagen dos), es un proceso muy costoso en cuanto a tiempo. Es por esto que la imagen no se rota en todos los ángulos, sino que, solo se hace la rotación en algunos ángulos representativos.

Aunque solo sean algunos ángulos los que se utilicen para verificar cual es la rotación correcta de las imágenes, sigue siendo un proceso muy tardado. Es por esto que se toma una muestra de los puntos de interés que arrojó el extractor SURF. Para nuestra aplicación se tomaron 10 puntos de interés

Por cada punto de interés se obtiene su característica y cada característica es comparada con todas las características del template para saber si existe coincidencia con algún punto de interés del template con el frame en ese ángulo. Para hacer aún más eficiente la búsqueda de mejor ángulo, si los puntos de interés similares entre el template y el frame son igual al tamaño de la muestra, hemos encontrado el mejor ángulo y no es necesario seguir comparando.

3.5.2. Escala de la imagen

Para atenuar el problema de escala, se procede hacer algo similar a lo realizado en la rotación, ir variando el tamaño de la imagen hasta encontrar el tamaño optimo de la imagen. La escala va de un tamaño menor a un tamaño mayor al tamaño original de frame, pasando por el tamaño original.

Esto queda ilustrado de mejor forma con la figura 3.11:

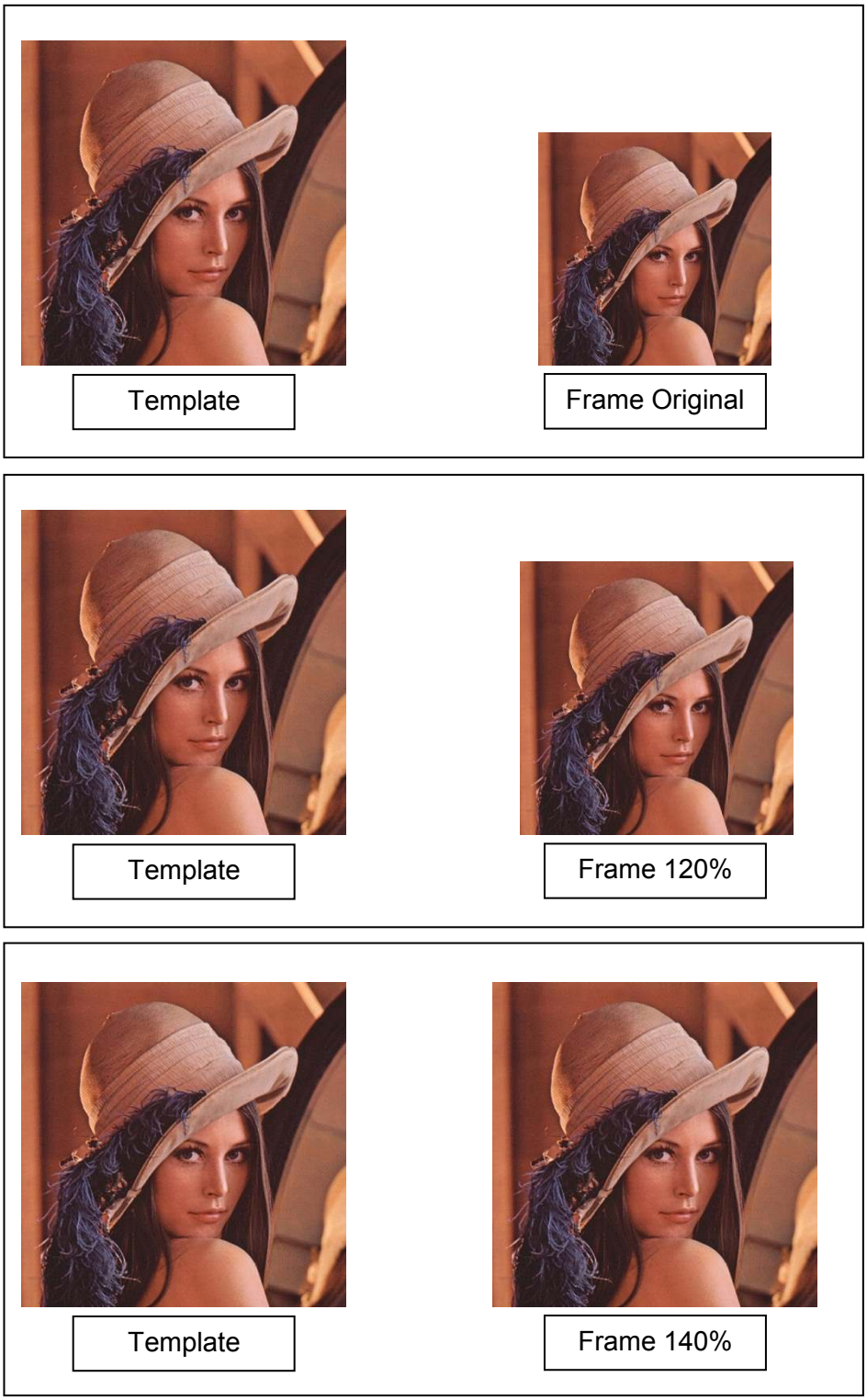

Figura 3.11 Proceso de escala.

Entonces se tiene que hacer las combinaciones de ángulos por cada tamaño, por lo tanto el número de iteraciones totales quedaría:

$$número\ de\ iteraciones = número\ de\ ángulos\ \times número\ de\ escalas$$

Esto si el número de coincidencias es menor al tamaño de la muestra.

En esta etapa de la implementación solo se hace uso de la clase **BRIEFAndroidActivity**, y de las funciones de **bestAngle** y **onMenuItemSelected**, como se muestra en la figura 3.12.

Figura 3.12 Diagrama de clases para Escala y Angulo.

La función **onMenuItemSelected** es la encargada de mandar llamar la función **bestAngle** una vez selecciona el Template. La función **bestAngle** contiene toda la funcionalidad descrita en esta sección: rotación y escalamiento de imágenes.

3.6. Comparación de características con distancia de Hamming

Se denomina distancia de Hamming a la efectividad de los códigos de bloque y depende de la diferencia entre una palabra de código válida y otra. Cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores. A esta diferencia se le llama distancia de Hamming, y se define como el número de bits que tienen que cambiarse para transformar una palabra de código válida en otra palabra de código válida. [**18**]

Si dos palabras de código difieren en una distancia d, se necesitan d errores para convertir una en la otra.

Un ejemplo de esto en relación con nuestra implementación es el siguiente:

Siendo las cadenas:

nd_1 = *M%#(=!1?lÑ|:Z<@s* y

nd_2=*J%#(=!1?lÑ|:M<@s*

La distancia Hamming sería de 2.

Entonces la distancia Hamming es un XOR.count de dos cadenas.

Para nuestro estudio la distancia Hamming máxima varía dependiendo del tamaña de nd, por ejemplo para nd = 16 la distancia Hamming es 15, para nd = 32 es 27, y para nd = 64 la distancia Hamming es de 27. Estos valores fueron evaluados y son los que dan los mejores resultados, esto quiere decir que al comparar una imagen exactamente igual, da el mayor número de concordancias y el menor número de concordancias falsas.

Este punto de la implementación es clave ya que aquí es donde se decide si dos puntos de interés son iguales o solo diferentes. Además es uno de los puntos donde se consume una importante parte del tiempo que se requiere para comparar las dos imágenes.

Lo que se hace aquí es comparar una descripción de un punto de interés del template contra todas las descripciones del frame. Esto se almacena en un vector con las coordenadas de los puntos que concuerdan entre las dos imágenes.

La implementación de la función de la distancia de Hamming queda plenamente identificada en la figura 3.13.

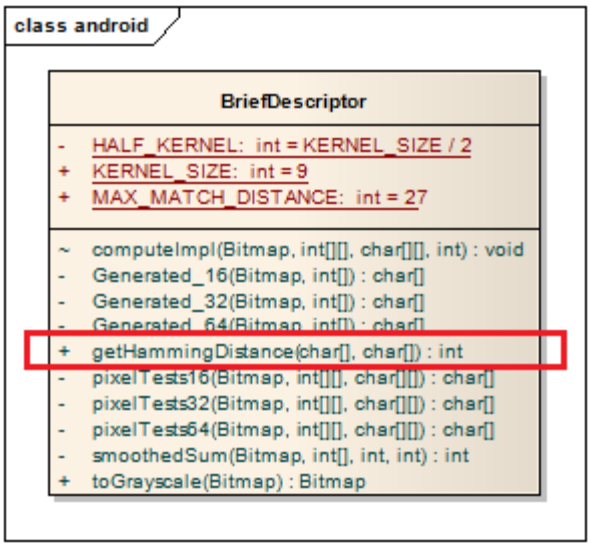

Figura 3.13 Diagrama de clases Hamming

De esta función que contiene la clase BriefDescriptor, no hay mucho que agregar ya que como se menciona en párrafos anteriores de esta sección la distancia Hamming es solo un XOR.count entre dos cadenas de caracteres.

3.7. Visualización de puntos y concordancias entre imágenes

Una vez que se tienen la relación entre los puntos de interés que coinciden, el siguiente paso a seguir es dar los resultados al usuario las concordancias y los puntos de interés que se han obtenido en los puntos anteriores. Es te punto parece un tanto fácil ya que solo tenemos que dibujar líneas y puntos entre dos imágenes, pero tenemos que recordar que para comparar las imágenes fue necesario aumentar/reducir y rotar la imagen del frame para poder buscar y describir los puntos de interés. Entonces tenemos que el tamaño y la rotación de la imagen podría no ser el original, y las coordenadas de los puntos de interés son en relación a la imagen a escala y en un ángulo diferente al original.

Entonces ¿Qué hacer con estas coordenadas? La única opción es escalar y rotar las coordenadas.

Para rotar las coordenadas haremos uso de las siguientes fórmulas:

En primer lugar para poder rotar la imagen tendremos que hacer de la imagen un plano cartesiano, cuyas coordenadas 0,0 serian el centro de la imagen, por lo tanto para cada coordenada X y Y tendremos que restar lo ancho y lo alto de la imagen respectivamente.

La fórmula para obtener las coordenadas en un determinado ángulo es:

En primer lugar cambiamos las coordenadas X y Y a coordenadas cartesianas, dividiendo las coordenadas entre la escala:

$$x' = x/escala - Width/2$$

$$y' = y/escala - Height/2$$

A estas coordenadas aplicamos las formulas de ecuación de rotación de ejes:

$$x'' = x' \cos(angulo) - y' \sin(angulo)$$

$$y'' = x' \sin(angulo) + y' \cos(angulo)$$

Y por último a convertir las coordenadas cartesianas a coordenadas de matriz de la imagen:

$$x_{original} = x'' + Width/2$$

$$y_{original} = y'' + Height/2$$

Para unas coordenadas 84, 225 de una imagen de tamaño 512x512, rotada a 90° y con una escala de 1, el punto de interés se localizaría como se muestra en la figura 3.14:

Figura 3.14 Coordenadas de imagen con rotación 90 y escala 1.

Aplicamos la formulas anteriores:

$$x' = \frac{84}{1} - \frac{512}{2} = -172$$

$$y' = \frac{225}{1} - \frac{512}{2} = -31$$

$$x'' = -172 \times 0 - (-31 \times -1) = -31$$

$$y'' = -172 \times -1 - (-31 \times 0) = 172$$

$$x_{original} = -31 + \frac{512}{2} = 287$$

$$y_{original} = 172 + \frac{512}{2} = 428$$

Como resultado obtendremos que las coordenadas del punto de interés se localizan en 225, 428, como se ilustra en la figura 3.15:

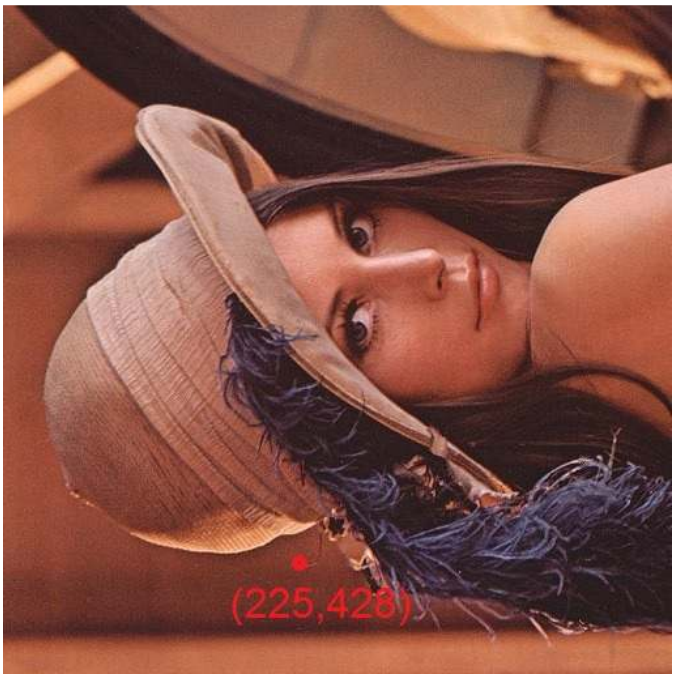

Figura 3.15 Coordenadas de la imagen original.

La implementación del código para esta etapa es con la clase **BRIEFAndroidActivity**, y con la función de **estimateCoordinatesOfTemplate**, que no tienen ninguna otra funcionalidad más que las formulas descritas en párrafos anteriores. Esta función es llamada por **combineImages**, al unir las imágenes y al conectar los puntos de interés de las imágenes, proceso que se describe en la sección 3.8. La figura 3.16 muestra la clase y la función con los parámetros y el valor de retorno.

Figura 3.16 Diagrama de clases Coordenadas

3.8. Uniendo las dos imágenes y conectando puntos de interés similares.

Para unir dos imágenes en una sola se tiene que crear una lienzo nombrado en Android canvas. El primero paso es saber el tamaño que deberá tener la canvas es sumar el ancho de las dos imágenes, para lo alto se tiene que verificar cual de las dos imágenes es la que tiene la altura mayor, y esta será el valor para la canvas.

Ahora que tenemos el tamaño de la canvas podemos unir las imágenes con las herramientas de Android.

El siguiente paso es unir los puntos que coinciden con una línea entre las dos imágenes, esto se muestra en la figura 3.17:

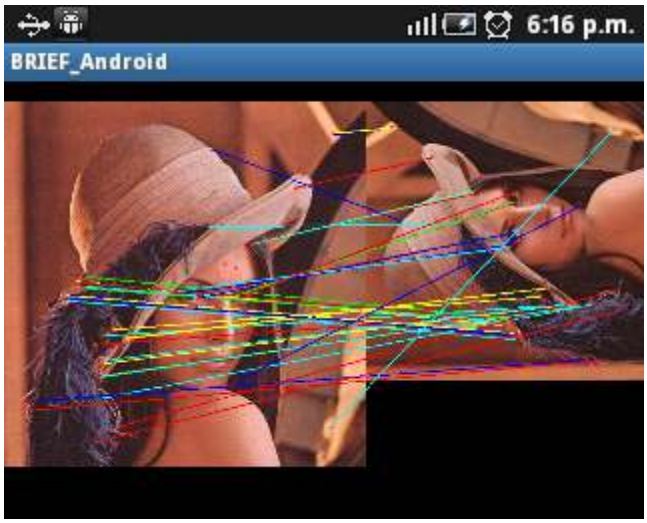

Figura 3.17 Comparación con imagen menor.

La figura 3.18 muestra el resultado final de nuestra implementación de BRIEF en Android. Como puede observarse existen dos imágenes, la imagen izquierda es el template y a la derecha está el frame. Las dos cuentan con tamaños y ángulos diferentes, y como se puede ver aun con esas características nuestra implementación encuentra similitudes entre las imágenes. La figura 3.18 muestra como se ve la combinación de imágenes donde el frame es mayor al template.

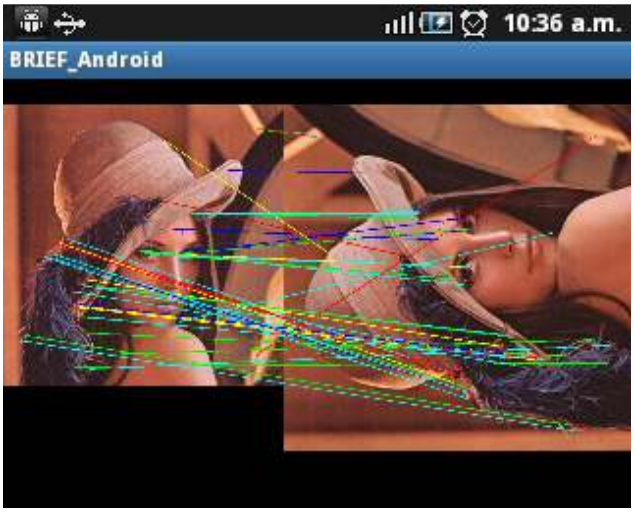

Figura 3.18 Comparación con imagen mayor.

Esta etapa queda implementada como se muestra en la figura 3.19, en las siguientes clases y funciones.

La función **combineImages** de la clase de **BRIEFAndroidActivity**, es la encargada de implementar la funcionalidad de hacer la unión entre las dos imágenes y dibujar los puntos de interés similares entre estas. Las clases de Layout y drawable, son las encargadas de mostrar estas imágenes, ya que sirven como lienzo para plasmar la comparación de las dos imágenes.

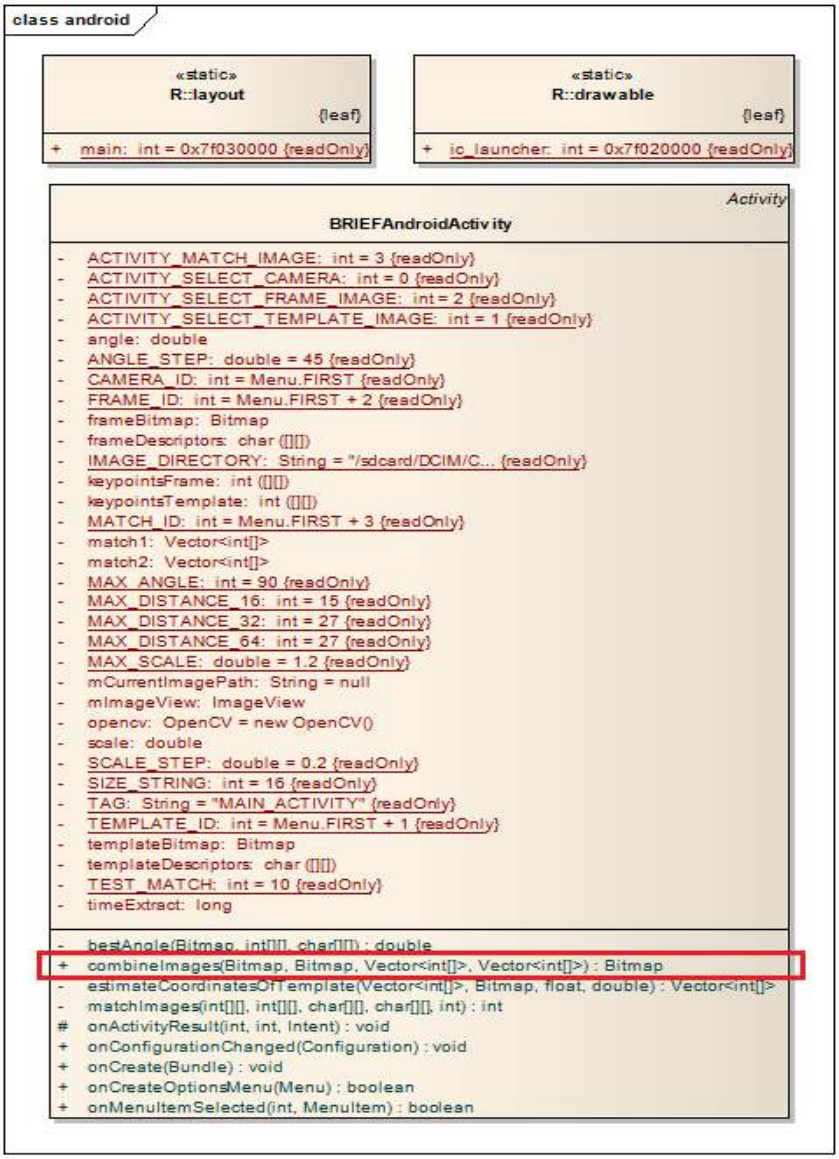

Figura 3.19 Diagrama de clases graficar resultados.

4. Resultados

En esta sección se presentarán los resultados obtenidos de nuestro descriptor para cada una de las combinaciones existentes de rotación y escala. Además se realizará el análisis del comportamiento de BRIEF en los diferentes tamaños de string nd que son 16, 32 y 64, comparando el número de concordancias de puntos y el tiempo que toma para hacer la descripción y la comparación entre imágenes.

4.1. Plataforma de Experimentación.

Antes de entrar con resultados, es conveniente aclarar las características del dispositivo en las que fueron ejecutadas las pruebas.

Las características del dispositivo son:

Tabla 1 Características del dispositivo.

Característica	Valor
Tipo	Celular
Nombre	Samsung GT-S5570L
Procesador	Qualcomm MSM7227 ARMv6 600MHz,
Versión Android	2.2
Memoria interna	160MB memoria interna, 384MB RAM
Cámara	3.15 MP, 2048x1536 pixels, geo-tagging, video QVGA@15fps

4.2. Tiempos totales por tamaño de string

4.2.1. BRIEF-16 sin rotación ni escala

El número de puntos de interés que nos arroja el extractor SURF es de 56 puntos tanto para el template como para el frame.

Los tiempos totales invertidos en cada fase para dos imágenes totalmente iguales, sin variación de tamaños ni ángulo es como sigue:

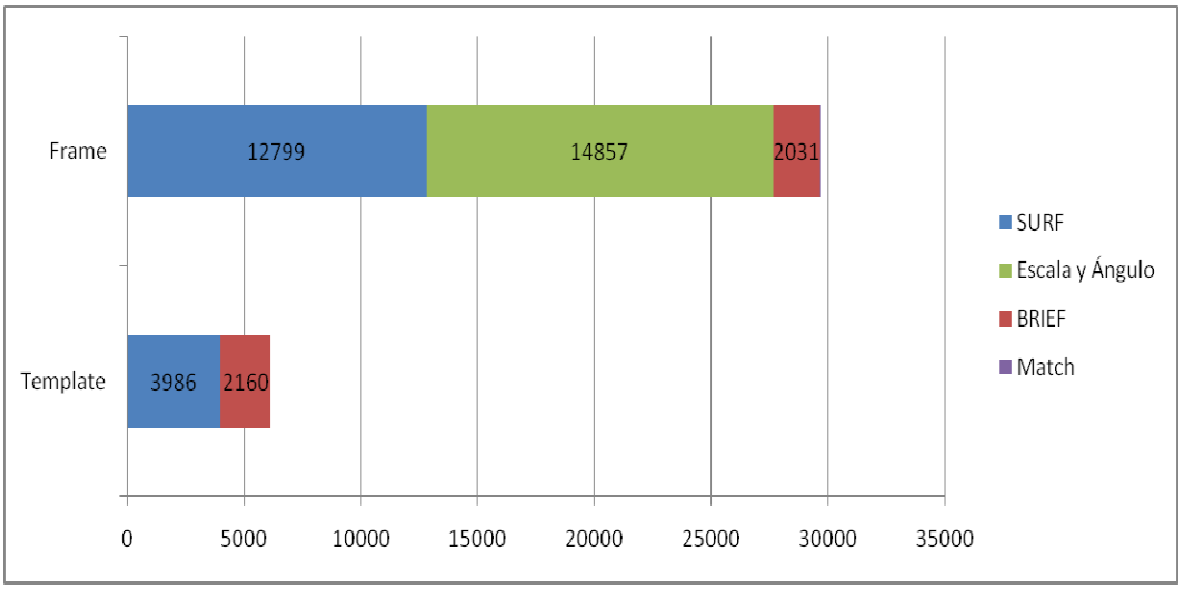

Figura 20 Gráfica de distribución de tiempos en milisegundos para tamaño 16.

Como puede observarse en la figura 4.1, el tiempo donde se consume la mayor cantidad de tiempo es en el extractor SURF, casi siendo el doble que el descriptor. Para el caso del Frame se ve un aumento significativo en el tiempo consumido ya que el tiempo promedio en extraer las características de una imagen, se multiplican por el número de ángulos y el número de escalas que se comparan para encontrar cuales son los óptimos.

El tiempo requerido para encontrar la escala y el ángulo adecuado es de 14857 milisegundos, este tiempo comprende tanto el periodo para extraer como los tiempos para describir y comparar las características, si a este tiempo le restamos el tiempo del extractor SURF 14857-12799 = 2058, tendremos que el tiempo de descripción de las características para todos los ángulos y escalas, es un valor muy reducido, esto debido a dos factores:

1) Solo se obtiene una muestra de 10 puntos de interés a describir y a comparar.
2) Si los puntos de interés en común es igual al tamaño de la muestra, la búsqueda para encontrar el mejor ángulo y escala termina.

El tiempo para hacer la comparación una vez que se obtiene el ángulo y la escala adecuada, es tan insignificantica que no es visible en la gráfica, y es de 22 milisegundos.

El número de puntos de interés similares es de 58, a simple vista podemos observar que existe un error ya que solo existen 56 puntos de interés entre imágenes, por lo tanto podemos concluir que para un tamaño en el string de 16 para BRIEF, este tiende a encontrar concordancias inexistentes entre las dos imágenes.

4.2.2. BRIEF-16 rotación 0° a escala menor

Para un frame con una escala menor la aplicación se comporta como muestra la figura 4.2:

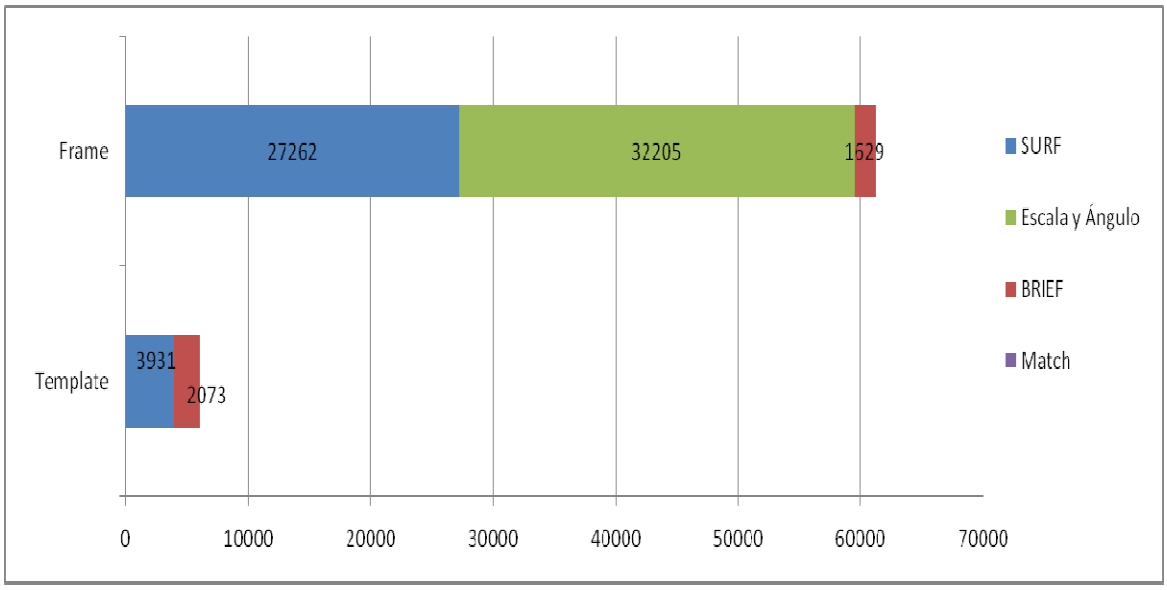

Figura 21 Gráfica de distribución de tiempos en milisegundos para tamaño 16 con rotación 0° y escala menor.

Como muestra la figura 4.2, los tiempos aumentan a pesar de que la imagen es de tamaño más reducido, y por lo tanto el extractor debería de tardar menos tiempo en terminar su proceso, pero como se menciono anteriormente la clave es encontrar el mejor ángulo y escala y no seguir con el proceso de comparación.

4.2.3. BRIEF-16 rotación 0° a escala mayor

Para un frame con una escala mayor la aplicación se comporta como ilustra la figura 4.3:

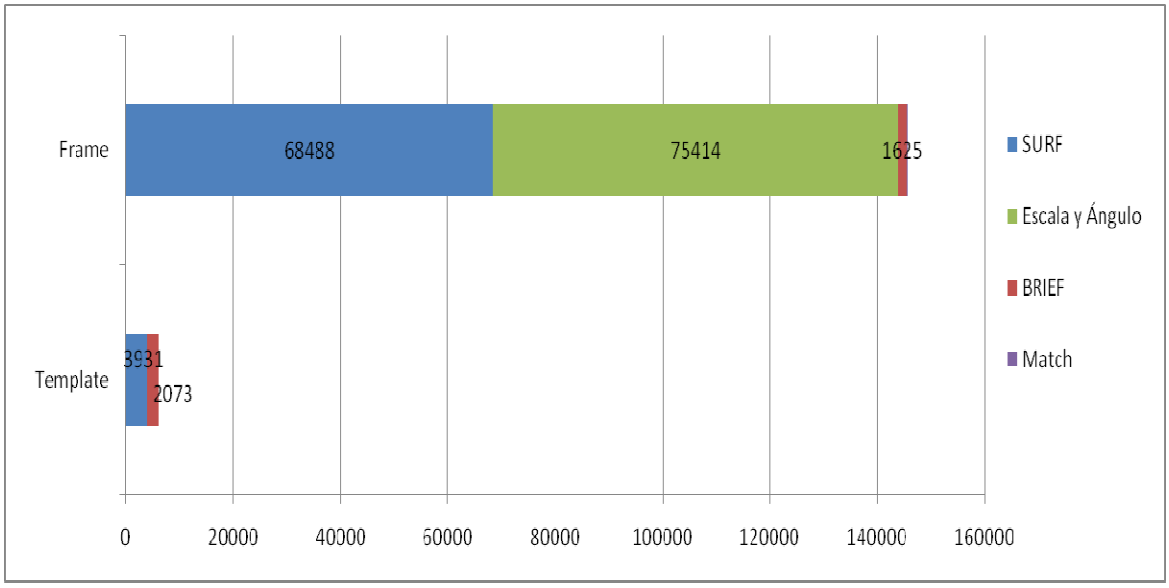

Figura 22 Gráfica de distribución de tiempos en milisegundos para tamaño 16 con rotación 0° y escala mayor

Como se observa en la figura 4.3 el tiempo es mayor que la imagen con escala menor sin ángulo de rotación, esto es debido al descriptor, que tarda más tiempo en analizar las imágenes y obtener los puntos de interés, ya que es mayor la imagen a analizar.

4.2.4. BRIEF-16 rotación 90° sin escala

Como puede apreciarse en la figura 4.4 los tiempos aumentan considerablemente para procesar la imagen del frame. Para el template queda iguales los tiempos ya que en este no existen ninguna variación en cuanto a operaciones adicionales a la imagen.

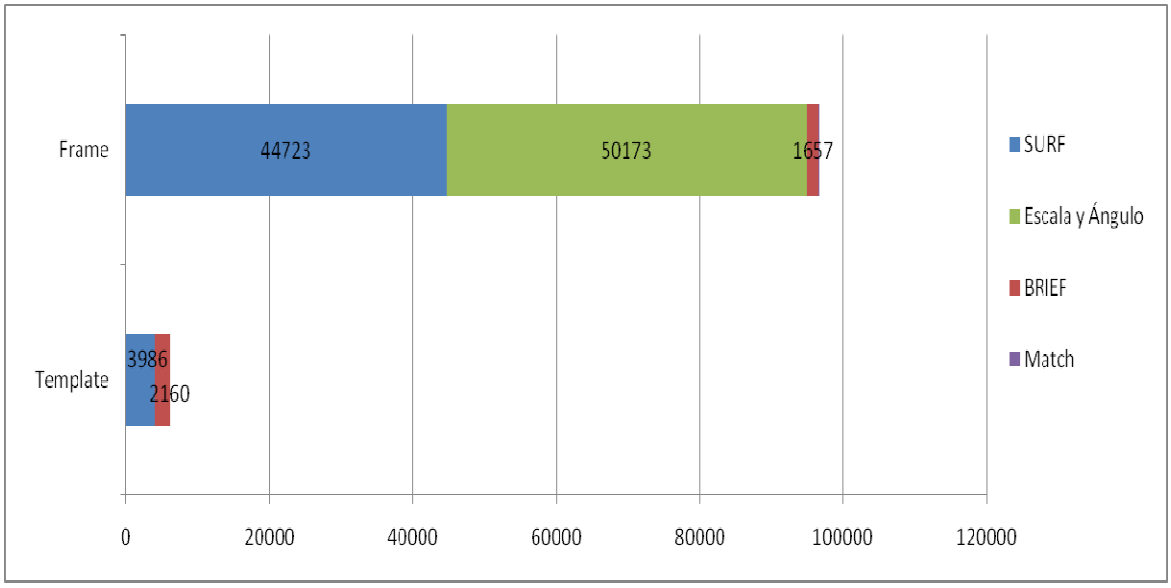

Figura 23 Gráfica de distribución de tiempos en milisegundos para tamaño 16 con rotación 90°.

El aumento en el extractor SURF y en el proceso de encontrar la escala y el ángulo, se debe a que aquí se hizo todos los procesos de descripción y comparación en todos los ángulos y todas las escalas disponibles. Aquí no existió ninguna coincidencia exacta entre el número de concordancias y el tamaño de la muestra.

Pero aún así si haces la resta de tiempos de escala y ángulo contra SURF tendremos que se invirtió solo 5450 milisegundos en hacer las descripciones y comparaciones de las muestras para todos los ángulos y escalas. Si para nuestro estudio se tiene tres escalas diferentes y 3 ángulos de rotación diferentes, el tiempo que se invertiría en describir ese número de combinaciones seria 9 x 1657 = 14913. Con esto queda comprobado que al hacer la descripción y comparación de una muestra de los puntos de interés de la imagen el tiempo invertido se reduce en un 63.45%.

En relación a los puntos de interés similares entre imágenes con una rotación de 90, el BRIEF de tamaño de string de 16, resultó ser sumamente deficiente, al no encontrar ninguna concordancia entre las imágenes.

4.2.5. BRIEF-16 rotación 90° a escala menor

Los resultados para las está pruebas son mostradas en la figura 4.5:

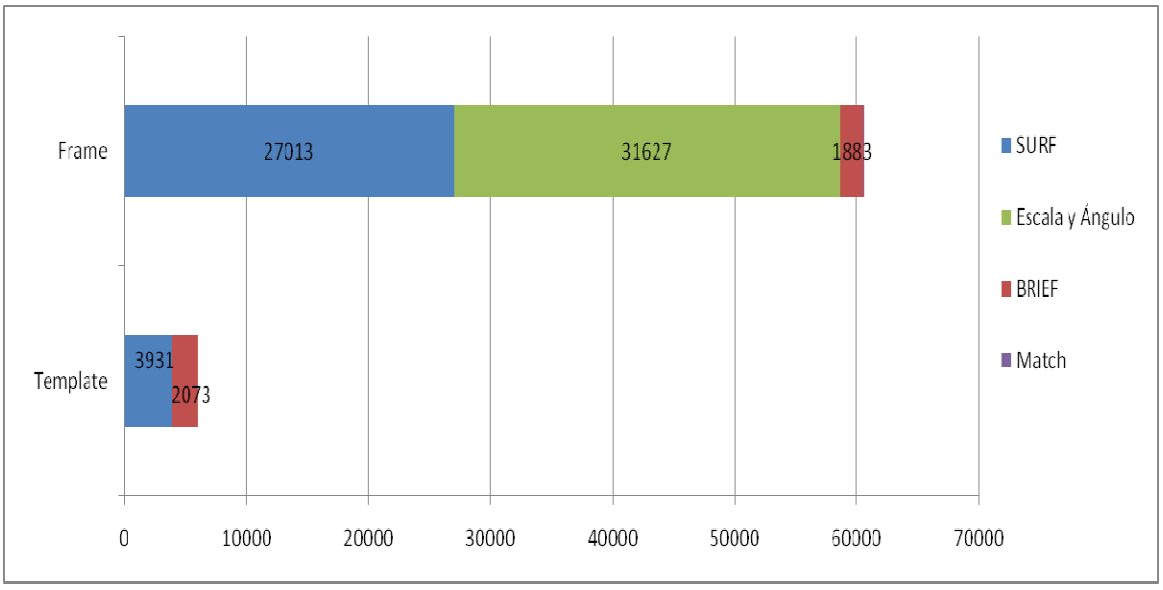

Figura 24 Gráfica de distribución de tiempos en milisegundos para tamaño 16 con rotación 90° y escala menor.

Los tiempos se redujeron en relación a los tiempos de rotación sin escala.

Al igual que las pruebas anteriores con rotación de 90° no encuentra coincidencias en las imágenes.

4.2.6. BRIEF-16 rotación 90° a escala mayor

Como se puede ver en la figura 4.6 los tiempos aumentaron considerablemente con respecto a las demás pruebas con escala.

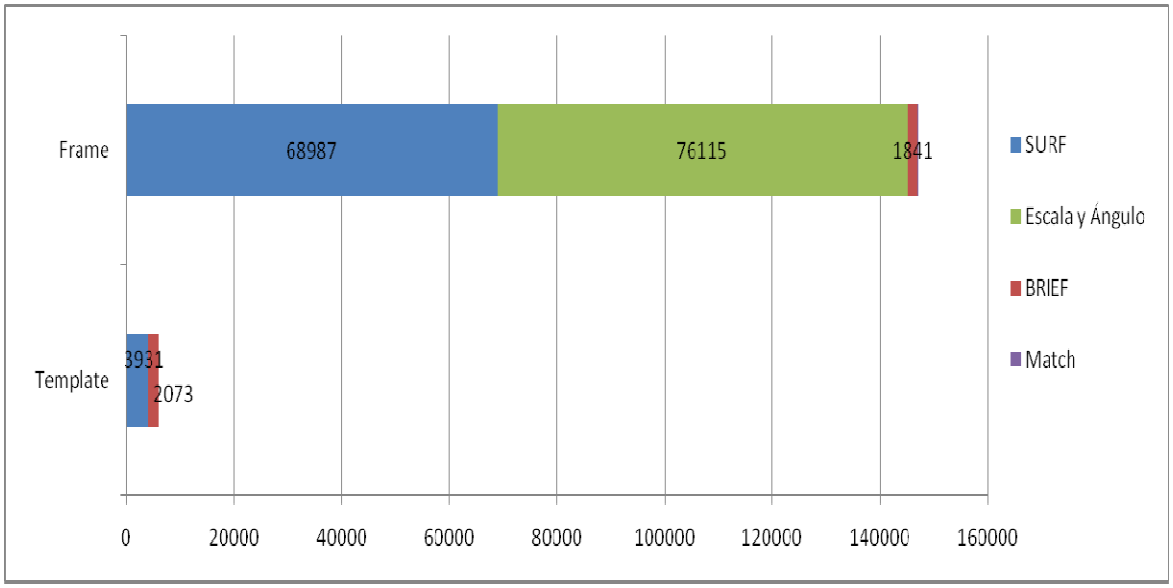

Figura 25 Gráfica de distribución de tiempos en milisegundos para tamaño 16 con rotación 90° y escala mayor.

Entonces encontramos que el tamaño del frame afecta proporcionalmente los tiempos en la extracción, obtención de escala y tamaño, pero no afecta el proceso que realiza el descriptor BRIEF, ya que el tiempo de este sigue constante en las tres escalas.

4.2.7. Tendencias de tiempos para BRIEF-16 en las escalas

A continuación se mostrará la figura 4.7 que refleja las tendencias de tiempos para nuestra implementación:

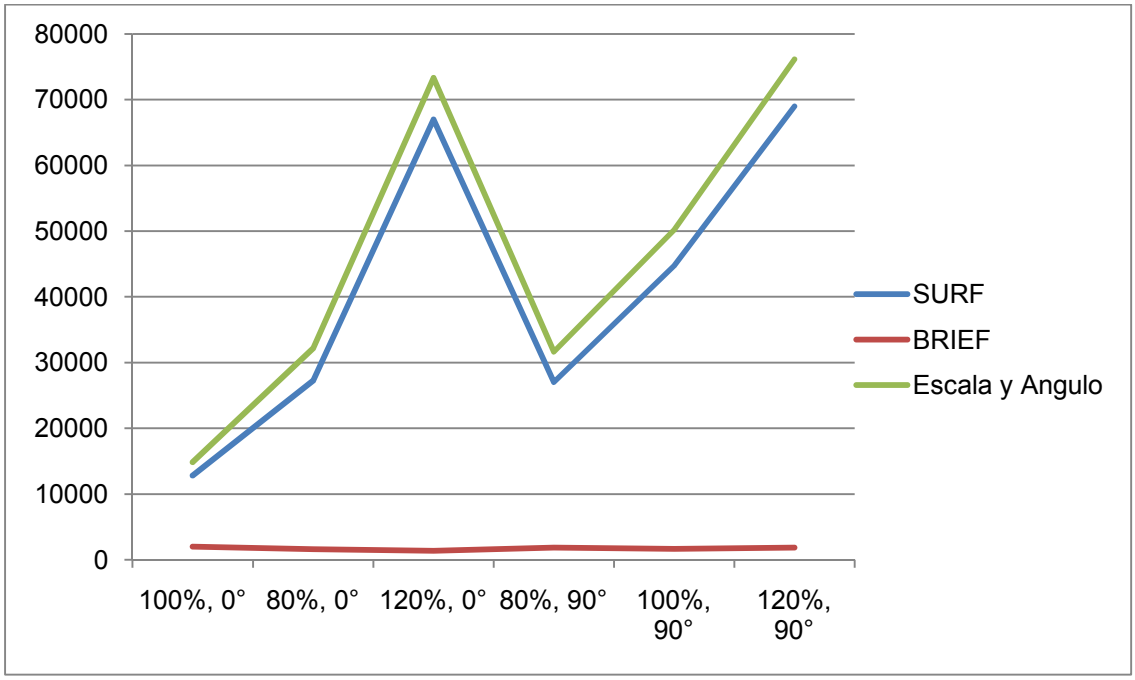

Figura 26 Tendencias de tiempo BRIEF-16.

Como se muestra la gráfica la tendencia de tiempos van en aumento dependiendo de la escala del frame. Cabe mencionar que el ángulo no afecta en el tiempo tanto de SURF como de la obtención del ángulo y la escala.

Otro dato importante que podemos obtener de esta gráfica es que el tiempo que se tarda en describir las características nuestro método BRIEF, es constante, sin afectarle el tamaño ni la rotación de la imagen.

4.2.8. BRIEF-32 sin rotación ni escala

Para el tamaño de string de 32, se muestra un aumento en los tiempos, pero es insignificante, en promedio los tiempos aumentaron en 500 milisegundos para el proceso de la descripción de las características de la imagen.

Si son 56 puntos de interés y en promedio aumento 500 milisegundos, quiere decir que aumento en promedio 9 milisegundos más para describir un punto de interés.

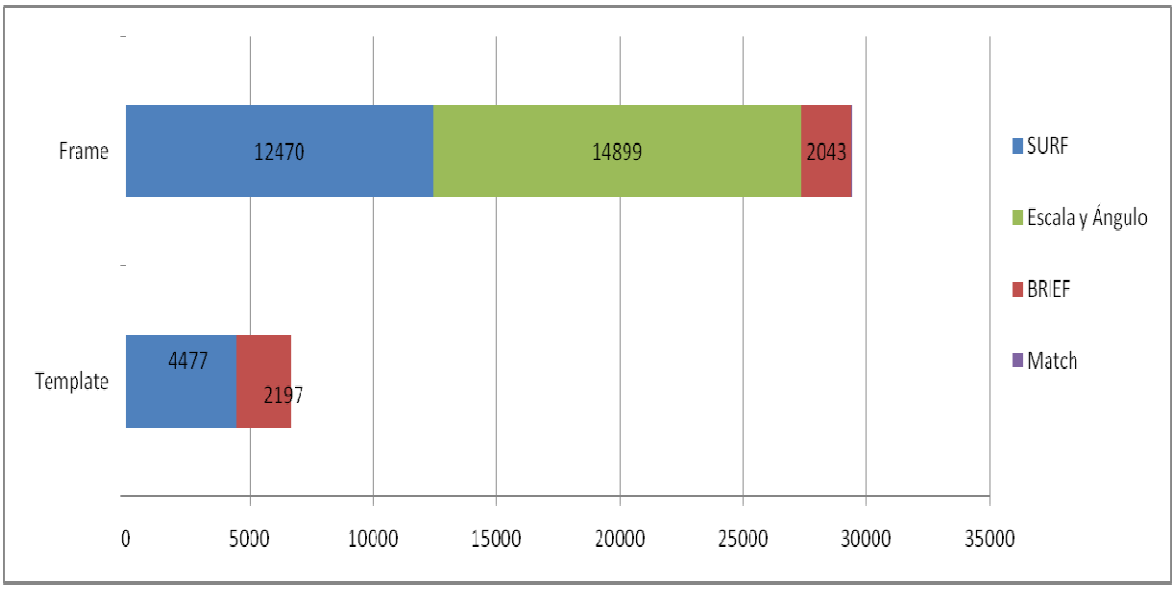

Figura 27 Gráfica de distribución de tiempos en milisegundos para BRIEF-32 entre imágenes idénticas.

La figura 4.8 muestra los tiempos por fase para BRIEF-32, comparando dos imágenes idénticas.

Para la comparación con imágenes similares, las pruebas de comparación de puntos de interés muestran que de 56 puntos, encuentra 56 puntos iguales en un tiempo de 30 milisegundos.

4.2.9. BRIEF-32 rotación 0° a escala menor

Como se ha visto, la tendencia de tiempos es que aumenta mínimamente respecto a BRIEF-16. La siguiente imagen muestra la tendencia de BRIEF-32 para una imagen a escala menor, sin rotación (ver figura 4.9).

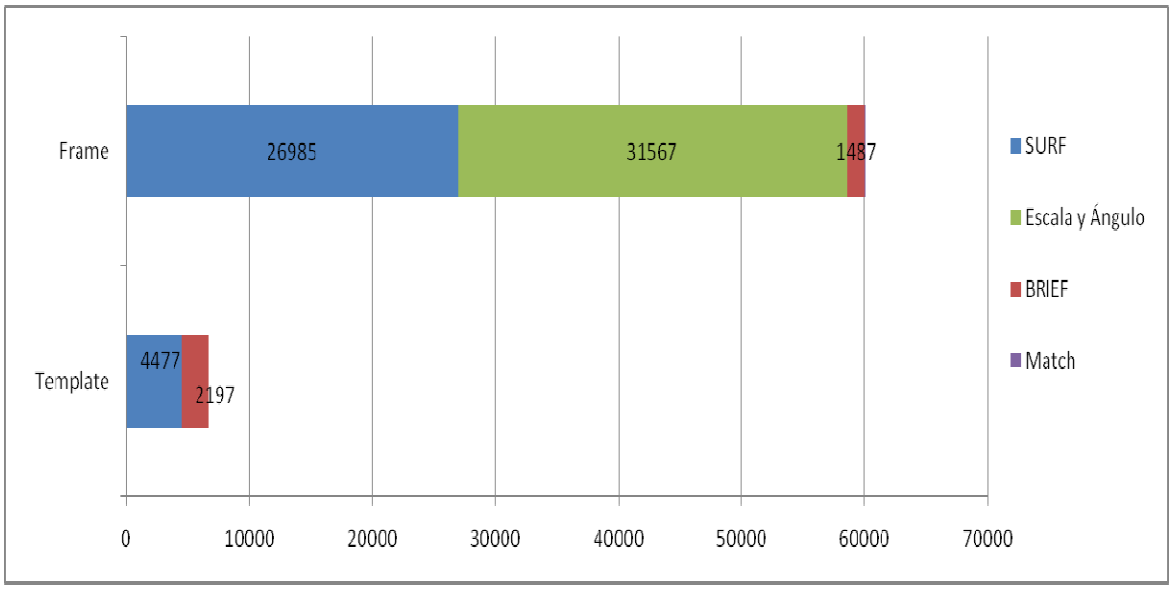

Figura 28 Gráfica de distribución de tiempos en milisegundos para BRIEF-32 rotación 0° a escala menor.

La diferencia principal entre los resultados de BRIEF-16 y BRIEF-32 es que para imágenes a escala y rotación si se encuentran puntos de interés en común ente las imágenes. Por ejemplo para esta comparación de 56 puntos de interés, se encontraron 46 coincidencias en 35 milisegundos.

4.2.10. BRIEF-32 rotación 0° a escala mayor

La figura 4.10 ilustra los tiempos para la extracción, obtención de ángulo y escala, y descripción de puntos de interés.

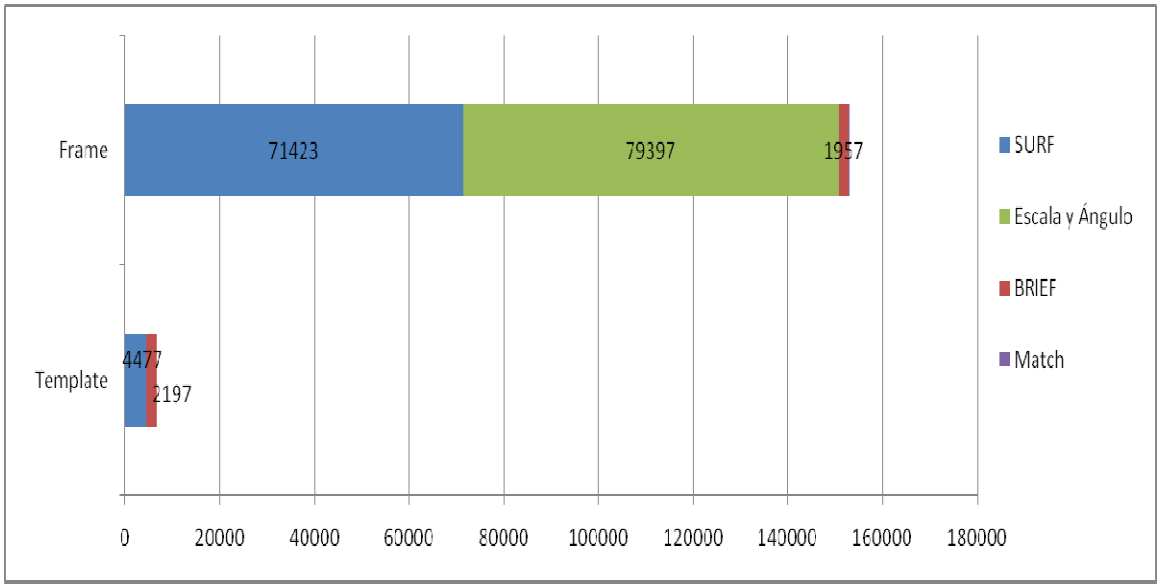

Figura 29 Gráfica de distribución de tiempos en milisegundos para BRIEF-32 rotación 0° a escala mayor.

Como se ha mostrado la tendencia de tiempos es similar para BRIEF-32 y BRIEF-16, pero como se menciono en el punto anterior, aquí si se encuentra puntos de interés en común.

Para esta comparación se encontraron 46 puntos en común en un tiempo de 35 milisegundos.

4.2.11. BRIEF-32 rotación 90° sin escala

Ya no es necesario repetir lo que se ha mencionado en los puntos anteriores. La figura 4.11 muestra los tiempos para esta comparación.

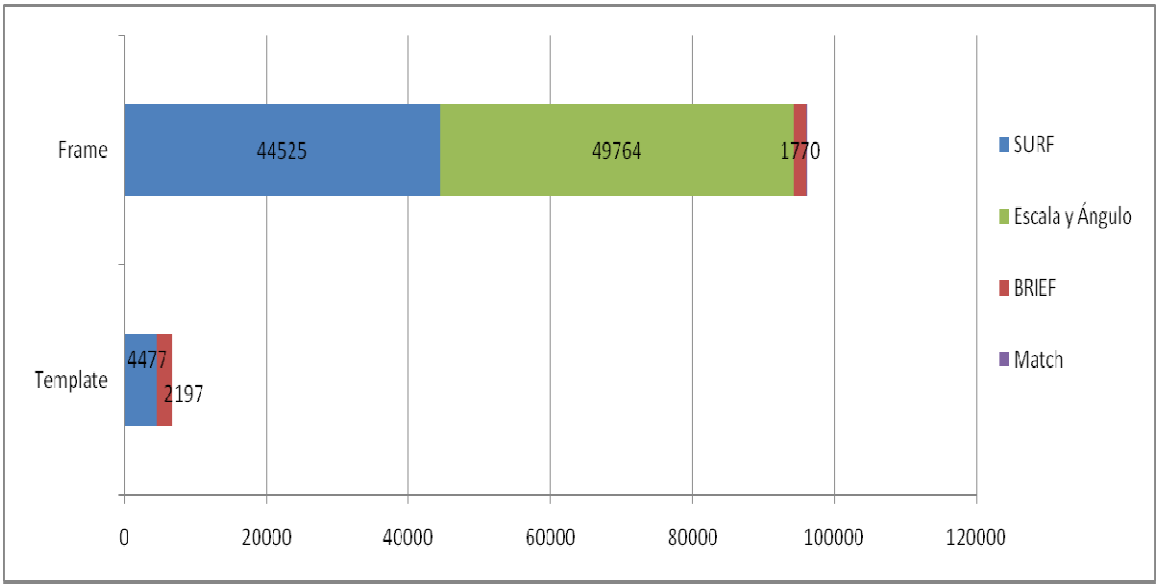

Figura 30 Gráfica de distribución de tiempos en milisegundos para BRIEF-32 rotación 90° sin escala.

Para las comparaciones el tiempo es de 34 milisegundos y se encontraron 51 coincidencias.

4.2.12. BRIEF-32 rotación 90° a escala menor

La figura 4.12 muestra los valores en milisegundos para cada fase.

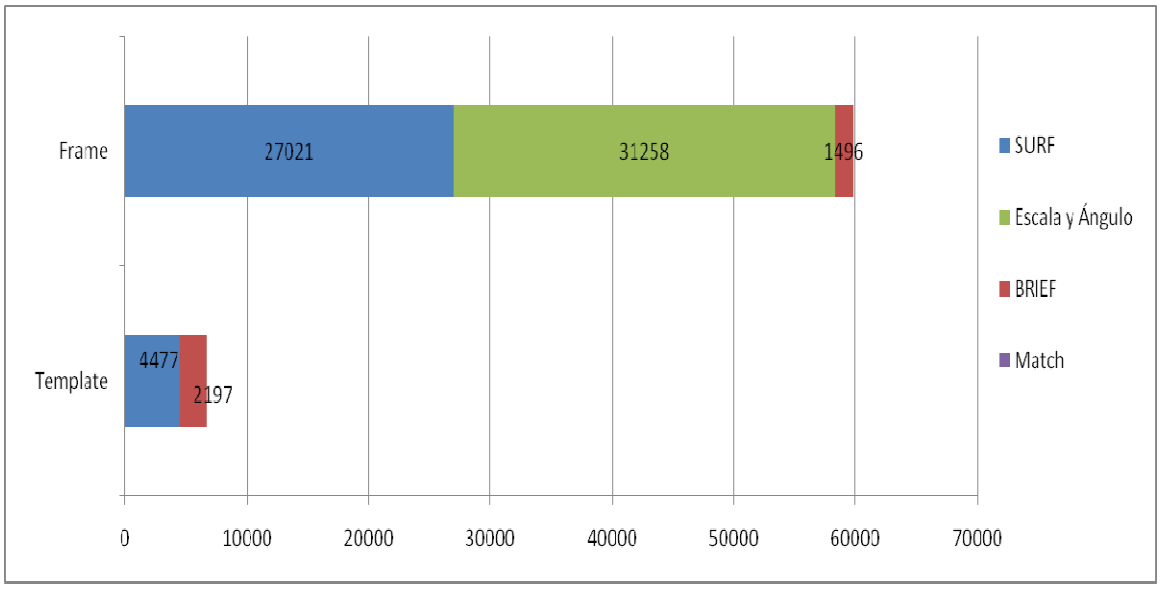

Figura 31 Gráfica de distribución de tiempos en milisegundos para BRIEF-32 rotación 90° a escala menor.

Los tiempos para la comparación es de 29 milisegundos y se encontraron 50 coincidencias.

4.2.13. BRIEF-32 rotación 90° a escala mayor

La figura 4.13 muestra los tiempos para BRIEF-32 para una imagen rotada a noventa grados a una escala mayor.

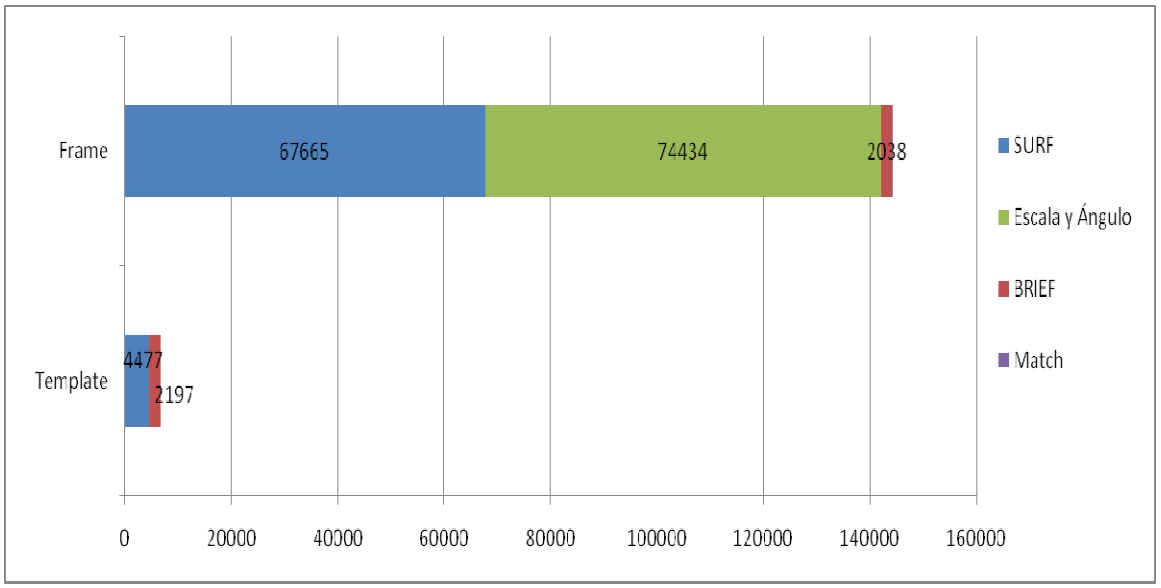

Figura 32 Gráfica de distribución de tiempos en milisegundos para BRIEF-32 rotación 90° a escala mayor.

Los puntos de interés que concuerdan entre las imágenes son 51 puntos, y el tiempo para hacer esto es de 36 milisegundos.

4.2.14. Tendencias de tiempos para BRIEF-32 en las escalas

La figura 4.14 compara los tiempos de los tres procesos de los que se compone nuestra implementación para cada una de las combinaciones de rotación y ángulo.

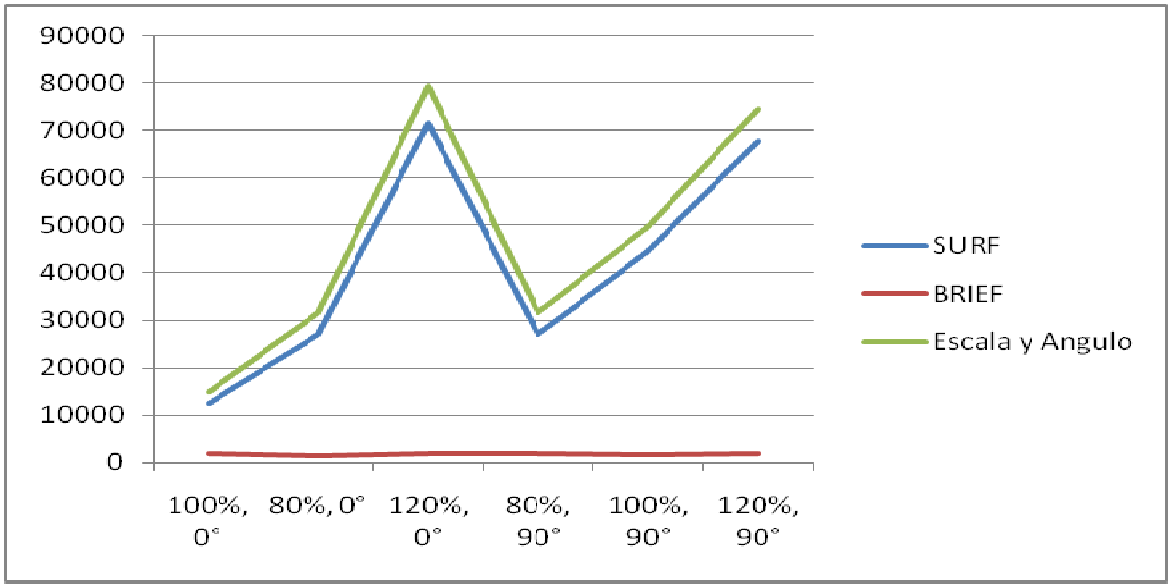

Figura 33 Tendencias de tiempo BRIEF-32.

Como era de esperarse esta gráfica es muy similar a la de BRIEF-16. Se esperaba que los tiempos aumentaran considerablemente debido a la longitud de la cadena de BRIEF, pero esto no pasó.

4.2.15. BRIEF-64 sin rotación ni escala

Para el tamaño de string de 64, se muestra un aumento en los tiempos, pero es insignificante, en promedio los tiempos aumentaron en 1000 milisegundos para el proceso de la descripción de las características de la imagen.

Si son 56 puntos de interés y en promedio aumento 1000 milisegundos, quiere decir que aumento en promedio 17.5 milisegundos más para describir un punto de interés de BRIEF-64.

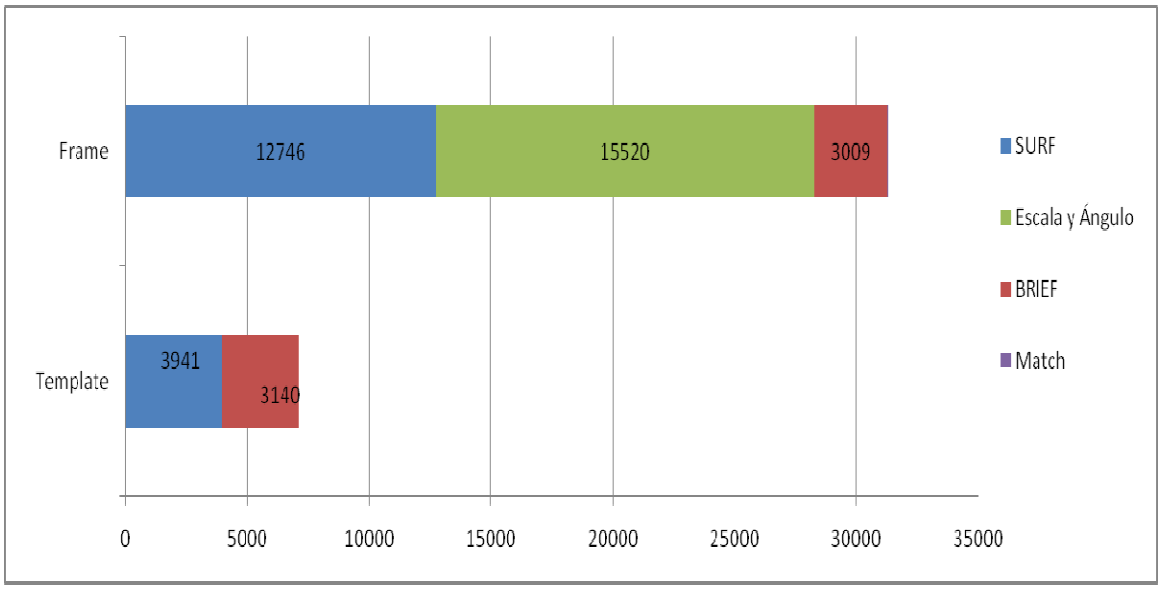

Figura 34 Gráfica de distribución de tiempos en milisegundos para BRIEF-64 entre imágenes idénticas.

La figura 4.15 muestra los tiempos por fase para BRIEF-64, comparando dos imágenes idénticas.

Para la comparación con imágenes similares, las pruebas de comparación de puntos de interés muestran que de 56 puntos, encuentra 56 puntos iguales en un tiempo de 33 milisegundos.

4.2.16. BRIEF-64 rotación 0° a escala menor

Como se ha visto, la figura 4.16 ilustra que aumenta mínimamente respecto a BRIEF-32. La siguiente imagen muestra la tendencia de BRIEF-64 para una imagen a escala menor, sin rotación.

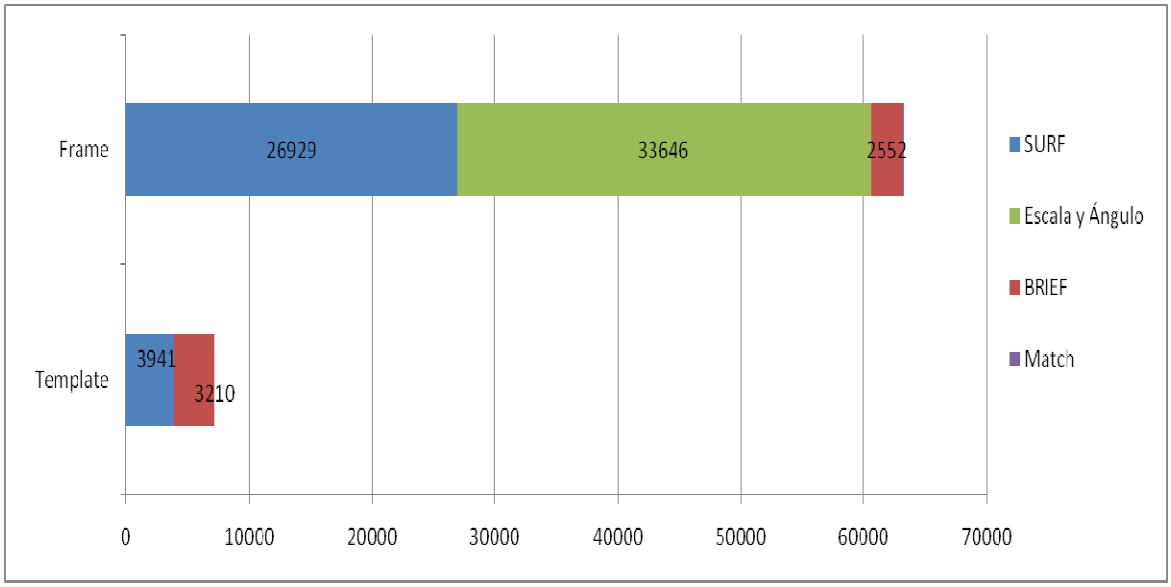

Figura 35 Gráfica de distribución de tiempos en milisegundos para BRIEF-64 rotación 0° a escala menor.

Aquí no existe diferencia en cuanto a número de puntos de interés en común ente las imágenes. Los dos, tanto BRIEF-32 como BRIEF-64 encontraron los mismos puntos de interés en común entre las dos imágenes.

4.2.17. BRIEF-64 rotación 0° a escala mayor

La figura 4.17 muestra los tiempos para la extracción, obtención de ángulo y escala, y descripción de puntos de interés.

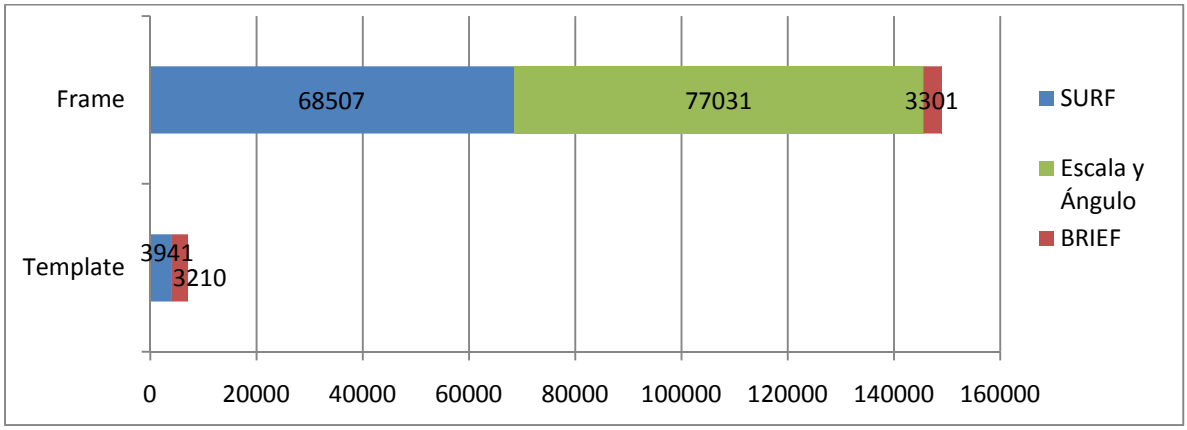

Figura 36 Gráfica de distribución de tiempos en milisegundos para BRIEF-64 rotación 0° a escala mayor.

Como se ha mostrado la tendencia en cuanto a concordancia de puntos de interés es similar para BRIEF-32 y BRIEF-64.

Para esta comparación se encontraron 46 puntos en común en un tiempo de 32 milisegundos.

4.2.18. BRIEF-64 rotación 90° sin escala

Ya no es necesario repetir lo que se ha mencionado en los puntos anteriores. La figura 4.18 muestra los tiempos para esta comparación.

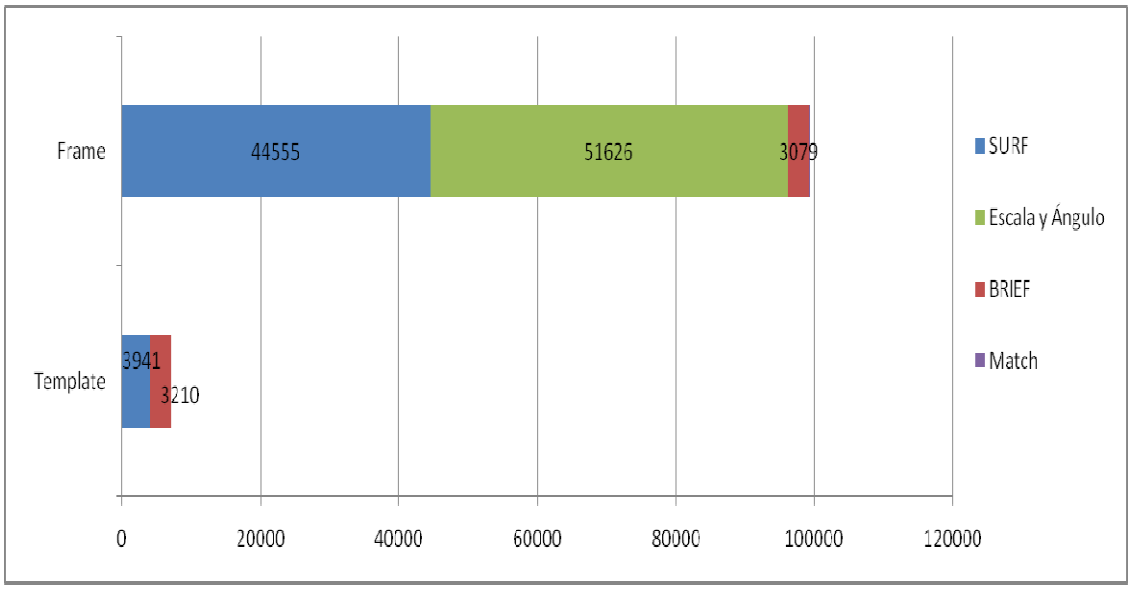

Figura 37 Gráfica de distribución de tiempos en milisegundos para BRIEF-64 rotación 90° sin escala.

Para las comparaciones el tiempo es de 15 milisegundos y se encontraron 51 coincidencias.

4.2.19. BRIEF-64 rotación 90° a escala menor

La figura 4.19 muestra los valores en milisegundos para cada fase.

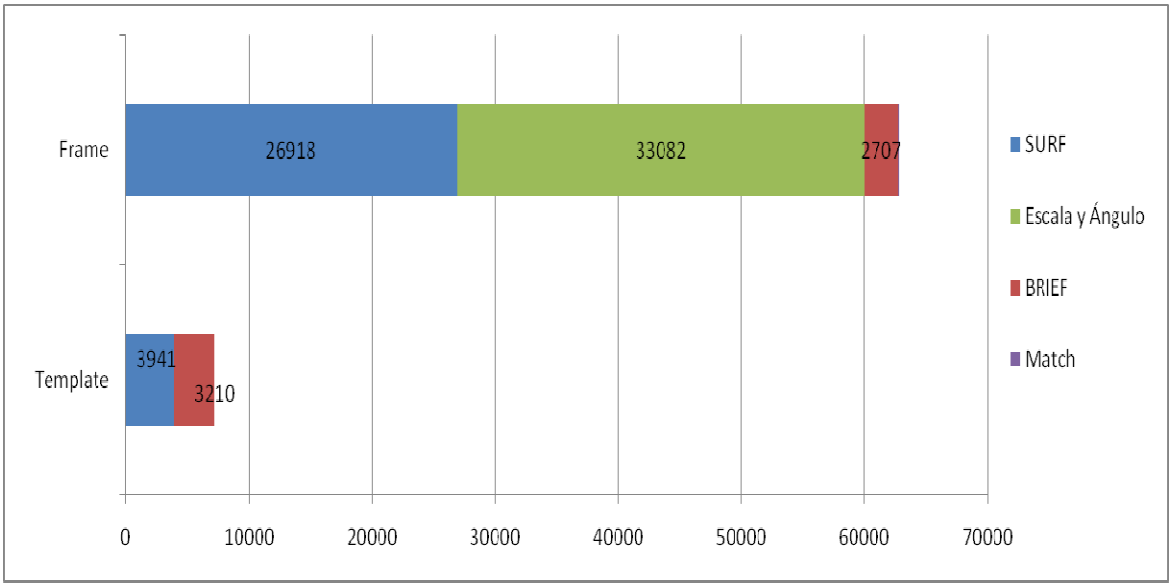

Figura 38 Gráfica de distribución de tiempos en milisegundos para BRIEF-64 rotación 90° a escala menor.

Los tiempos para la comparación es de 25 milisegundos y se encontraron 50 coincidencias.

4.2.20. BRIEF-64 rotación 90° a escala mayor

La figura 4.20 ilustra los tiempos para BRIEF-64 para una imagen rotada a noventa grados a una escala mayor.

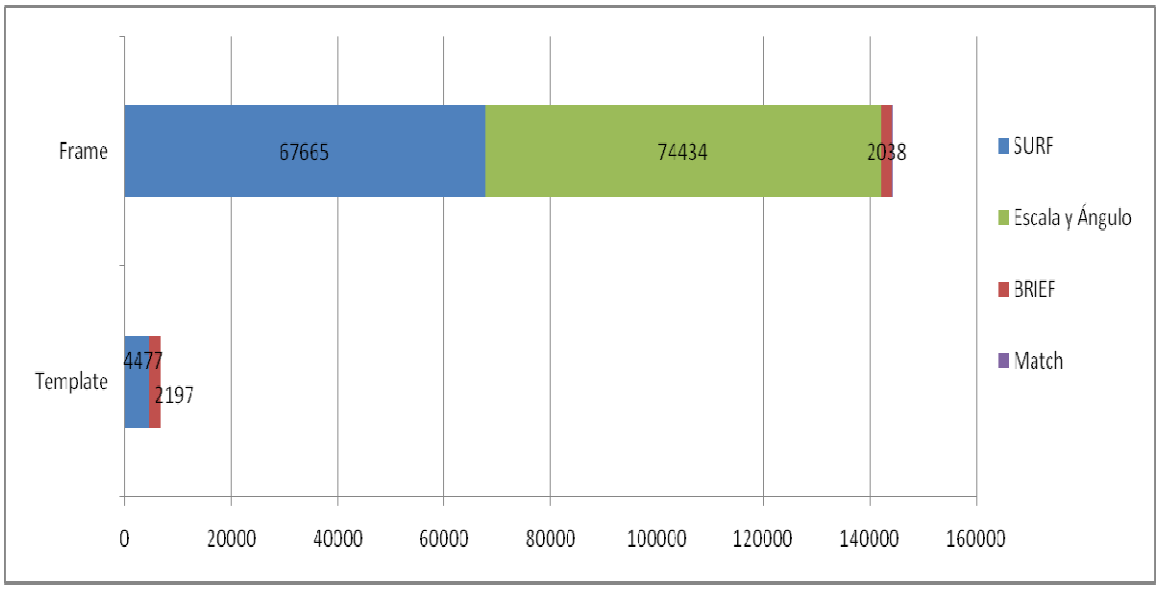

Figura 39 Gráfica de distribución de tiempos en milisegundos para BRIEF-64 rotación 90° a escala mayor.

Los puntos de interés que concuerdan entre las imágenes son de 51 puntos, y el tiempo para hacer esto es de 36 milisegundos.

4.2.21. Tendencias de tiempos para BRIEF-64 en las escalas

La figura 4.21 compara los tiempos de los tres procesos de los que se compone nuestra implementación para cada una de las combinaciones de rotación y ángulo.

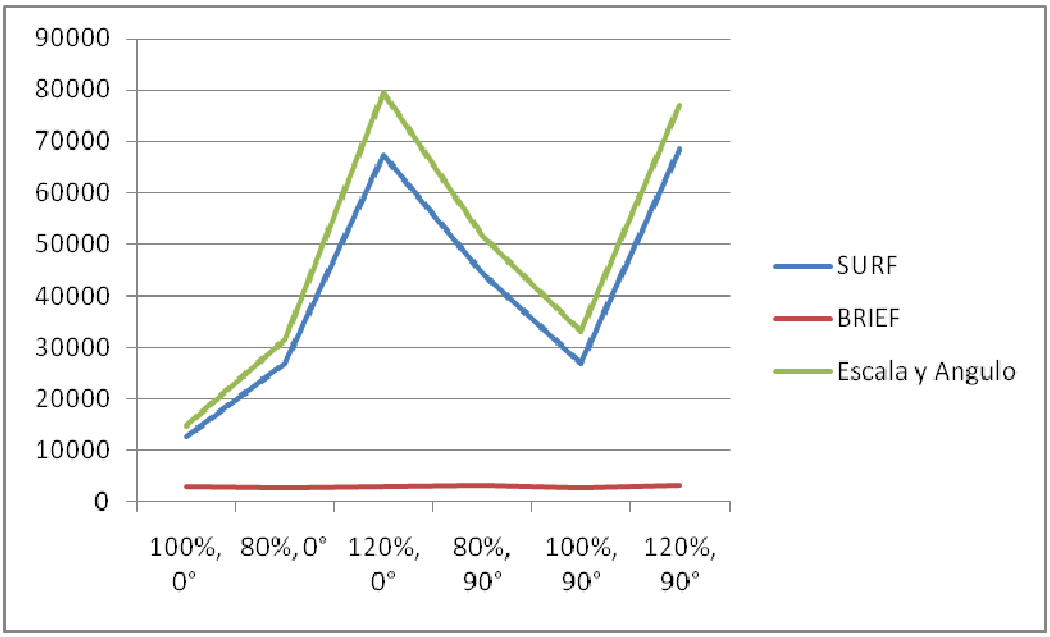

Figura 40 Tendencias de tiempo BRIEF-64.

Como era de esperarse esta la figura 4.21 es muy similar a la de BRIEF-32. Lo que se puede observar en esta gráfica es que la distancia entre los tiempos de SURF y Escala y Ángulo, ha aumentado.

4.3. Comparación de tiempos entre BRIEF-16, 32, y 64

La figura 4.22 muestra la forma en que se comporta BRIEF en sus diferentes tamaños de string.

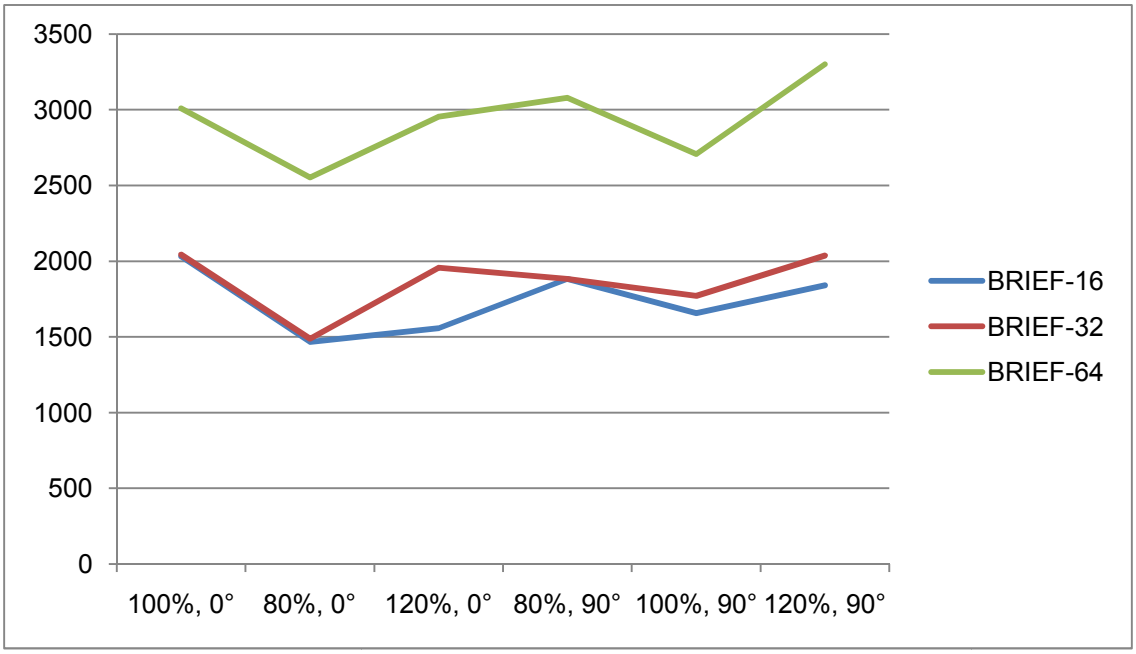

Figura 41 Comparación de BRIEF por tamaño.

La variación de tiempos entre BRIEF-16 y BRIEF-32 es mínima, donde se observa que aumenta el tiempo es en BRIEF-64.

4.4. Resultados de puntos de interés en común detectados

A continuación veremos los resultados para las tres variantes de BRIEF. La figura 4.23 muestra que no existe mucha diferencia entre BRIEF-64 y BRIEF-32, en cuantos puntos de interés similares entre las dos imágenes.

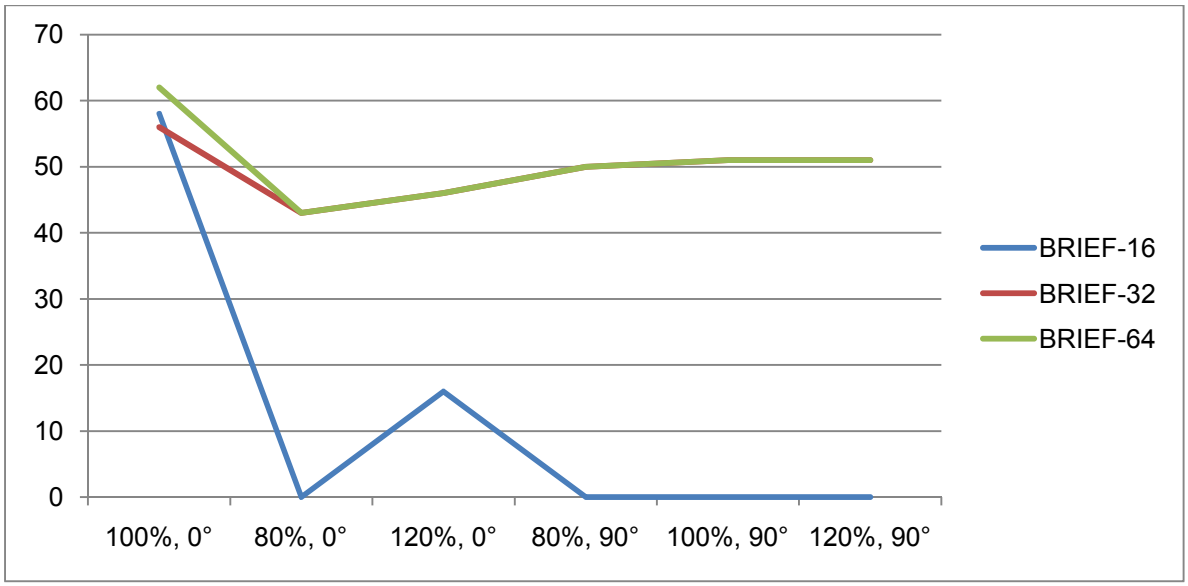

Figura 42 Comparación entre BRIEF-16, 32 y 64 para puntos de interés.

Como puede observarse la BRIEF-16 tiene grandes deficiencias al describir los puntos de interés, ya que solo encontró similitudes al comparar imágenes idénticas, y en imágenes a escala mayor, y tan solo encontró 18 puntos de interés.

Para BRIEF-64 a una imagen idéntica, se obtuvo un número de coincidencias mayor al número de puntos de interés, esto es una inconsistencia ya que las concordancias no pueden ser mayores a los puntos de interés.

4.5. Conclusiones de los resultados

Los resultados anteriores no indican que la mejor opción es BRIEF-32, ya que es mejor en cuanto a descripción de puntos que BRIEF-16 y el tiempo que consume en hacer esto es menor a BRIEF-64 del cual se tiene el mismo número de puntos de interés en común.

Pero algo muy importante en cuanto a los resultados obtenidos es que, en donde nuestra implementación consume la mayor cantidad de tiempo es en SURF y en encontrar la escala y el ángulo correcto de las imágenes.

Otro punto a resaltar es que BRIEF es muy constante en cuanto a tiempos, independientemente del ángulo y la escala de las imágenes.

5. Conclusiones generales

Analizando los resultados arrojados por las pruebas se puede concluir que lo que está afectado al performance de nuestra aplicación es el extractor de características SURF.

Otro proceso donde nuestra aplicación es sumamente lenta es en encontrar el ángulo y la rotación de la imagen. Esto está relacionado con el problema de performance de SURF ya que por cada ángulo y escala se extraen los puntos con SURF.

El menor tiempo que se tardó BRIEF en describir una imagen fue de 1466 milisegundos, esto para BRIEF-16, y el tiempo mayor fue de 3301 para BRIEF-64, esto es casi 2 segundos de diferencia entre BRIEF-16 y BRIEF-64. Pero viendo el mejor rendimiento en relación a tiempo/exactitud, BRIEF-34 es el mejor de las tres variantes.

Cabe mencionar que las pruebas sólo fueron ejecutadas en un solo dispositivo, por lo que la variable de hardware no pudo ser incluida en este estudio. Pero es un punto importante que se tendrá que atender en estudios posteriores.

6. Propuesta

Nuestra propuesta para la aplicación en general es utilizar otro extractor de características como FAST, que según el estudio [**19**] en cuanto a velocidad de extracción, es mucho más rápido que SURF en una proporción de 1 a 15, pero por falta de tiempo no fue posible la implementación para hacer las pruebas sobre este.

Una propuesta para el mejorar el rendimiento en específico para BRIEF es el procesamiento en hilos. Esto es dividir los puntos a describir en varios hilos para que las descripciones sean calculadas en forma simultánea reduciendo el tiempo.

Cambiar los puntos con los cuales se obtiene la intensidad del color de un pixel. Actualmente los puntos son puntos que de cierto modo son dados al azar, nuestra propuesta es que los puntos sean puntos fijos distribuidos en forma circular esto es que, nuestro punto de interés sea el centro de un círculo formado por los pixeles de donde se obtendrá la intensidad. Entonces los puntos de interés de los que se obtendrá su intensidad serán dados por:

$$x = \frac{KERNEL}{2} \times \cos\theta$$

$$y = \frac{KERNEL}{2} \times \sin\theta$$

Donde $\theta \in [0,360]$

La figura 6.1 muestra como estarán distribuidos los pixeles en forma circular para obtener la intensidad de un punto de interés dado por x, y.

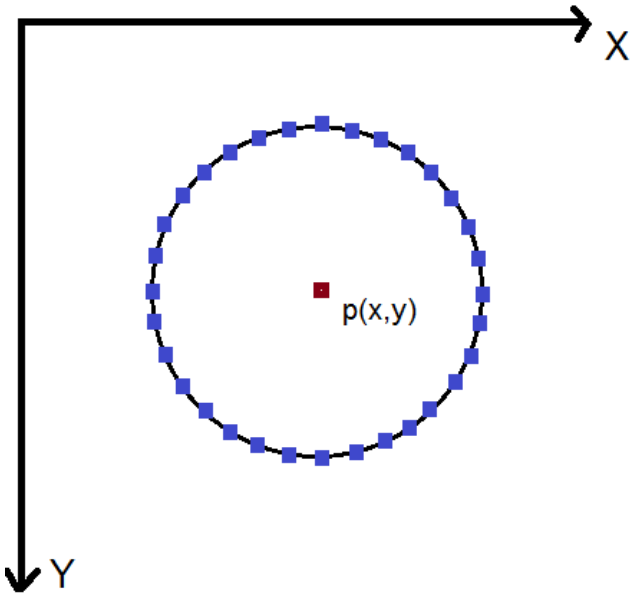

Figura 43 Propuesta de distribución de pixeles al rededor de p(x,y).

Con esto se obtienen puntos equidistantes al centro, entonces, supongamos que describimos obtenemos la intencidad de color de cada pixel, tendriamos por ejemplo la siguiente cadena:

String1="0123456789abcdefghijklmnopqrstuvwxyz"

Suponiendo que la imagen tiene una rotación de 30° la descripción para este punto sería:

String2="3456789abcdefghijklmnopqrstuvwxyz012"

Tendremos dos string con los mismo elementos pero en diferente orden. La distancia Hamming nos daria 36, pero si cambiando las posiciones de las caracteres de la cadena hasta llegar al menor valor de Hamming que seria 0. Con esto tendriamos cubierto la rotación de las imágenes sin la necesidad de extraer puntos, y describirlos por cada rotación, con esta mejora solo se haria una rotación de caracteres en una muestra, y al obtener la óptima se aplicararia a las demas caracteristicas cuando se hace la comparación, eliminado el proceso de obtención de ángulo.

7. Trabajo futuro

Como parte de nuestro trabajo futuro es hacer comparaciones con imágenes tomadas con la misma cámara del celular, siendo que ya está implementada esa opción, sería sólo obtener resultados de estas imágenes, y ver que tanto afecta la calidad de las imágenes para la descripción de los puntos de interés y la comparación de estos.

Otro punto a tomar en cuenta como trabajo futuro es implementar y comparar las propuestas que se dan en el punto anterior, con los resultados del trabajo realizado en este documento.

Existe mucho trabajo por hacer, como comparar nuestra aplicación en diferentes dispositivos, verificar si la capacidad del procesador influye significativamente en los tiempos de nuestra implementación de BRIEF.

Comparar nuestra implementación de BRIEF contra otras implementaciones existentes. Esto es de suma importancia para nuestro trabajo futuro ya que aquí nos indicaría que tan por encima o por debajo está el performance de nuestra aplicación con otra ya desarrolladas, como la de OpenCV.

El punto hasta donde se quiere llegar en el trabajo futuro es que nuestra implementación realice la comparación del template con el stream de video del dispositivo Android. Esto quiere decir que la comparación sea en tiempo real.

Bibliografía

1 Stockman, Linda G. Shapiro and George C. *Computer Vision*. Prentice Hall, 2001.

2 Zhu, Zhongkai. *Computer Vision Research Progress (1st ed.)*. Nova Science Publishers, Inc., NY, USA, 2008.

3 Davies, E. R. *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann, 2005.

4 Sasikumar Gurumurthy, Balakrushna K. Tripathy, M. Priya. Study of Image Recognition Using Cellular Associated Artificial Neural Networks. (Hong Kong 2011).

5 Lindeberg, Tony. *Edge detection and ridge detection with automatic scale selection*. San Francisco, California, 1996.

6 Stephens, C. Harris and M. A combined corner and edge detector. (United Kingdom 1988), The Plessey Company plc.

7 Lindeberg, Tony. Detecting salient blob-like image structures and their scales with a scalespace primal sketch: a method for focus-ofattention. *International Journal of Computer Vision* , 3, 11 (1993), 283-318.

8 Krystian Mikolajczyk, Cordelia Schmid. Scale & Affine Invariant Interest Point Detectors. *International Journal of Computer Vision*, 1, 60 (Enero 2004), 63-86.

9 Lowe, David G. *Object Recognition from Local Scale-Invariant Features*. University of British Columbia, Vancouver, Canada, 1999.

10 Herbert Bay, Tinne Tuytelaars, Luc Van Gool. *SURF: Speeded Up Robust Features*. Katholieke Universiteit Leuven, Zurich, 2006.

11 Navneet Dalal, Bill Triggs. *Histograms of Oriented Gradients for Human Detection*. French National Institute for Research in Computer Science and Control, Rocquencourt, 2005.

12 Krystian Mikolajczyk, Cordelia Schmid. *A performance evaluation of local descriptors*. University of Oxford, Oxford, 2005.

13 Hellwich, M. Saquib Sarfraz and Olaf. *Head Pose Estimation in Face Recognition across Pose Scenarios*. Berlin university of Technology, Berlin, 2008.

14 Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. *BRIEF: Binary Robust Independent*. CVLab, EPFL, Lausanne, Suiza, 2010.

15 Team, OpenCV Developers. OpenCV.org. (2012).

16 Liang, Sheng. *Java Native Interface: Programmer's Guide and Specification*. Addison Wesley Longman, Palo Alto, CA, 1999.

17 Michael Calonder, Vincent Lepetit, Mustafa Özuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua. BRIEF: Computing a local binary descriptor very fast. (Lausanne, Switzerland July, 2012).

18 Hamming, R. W. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, XXIX, 2 (Abril 1950), 393-426.

19 Khvedchenia, Ievgen. http://computer-vision-talks.com/2011/07/comparison-of-the-opencvs-feature-detection-algorithms-ii/. (Odessa, Ukraine July 11, 2011).