Detector de esquinas en tiempo real implementado en arquitectura SIMD para sistema autónomo de captura de imágenes aéreas

Ing. Irving Manuel Tolosa Garma

Maestría en Ciencias de la Computación
Facultad de Matemáticas
Universidad Autónoma de Yucatán
2014

Resumen

El presente proyecto se desarrolla a partir de un sistema autónomo de captura de imágenes aéreas , el cual determina el instante en que se deben de capturar las fotografías para la creación de un mosaico de imágenes. El sistema emplea un modelo geométrico, el cual determina si la imagen a capturar se traslapa con alguna de las anteriores; si el área es adecuada, se captura la fotografía. El modelo de captura que utiliza el sistema depende únicamente del modelo geométrico, y no considera la información presente en las imágenes. Para garantizar que el área de traslape entre las imágenes ha capturar tengan suficientes puntos de interés, se propone complementar el sistema con un algoritmo de visión para detectar esquinas, el cual determina si existe suficiente información en el área de traslape, que permita automatizar la creación del mosaico de imágenes. El algoritmo debe de ejecutarse en tiempo real, por lo cual es paralelizado y ejecutado en la unidad SIMD(singlas en inglés de Single Instruction Multiple Data) NEON que se encuentra embebida en la familia de procesadores cortex ARM.

Dedicatoria y agradecimientos

A Dios, ser maravilloso que me dio fuerza e inspiración para poder concluir este trabajo de tesis, por bendecirme hasta donde he llegado, por permitirme cumplir las metas que me he propuesto y por llenar mi vida de felicidad.

El presente trabajo de tesis está dedicado con cariño a las siguientes personas:

- A mis padre José, que Dios lo tenga en su gloria, por sembrar en mi el deseo de superación personal y lograr una carrera de éxito.
- A mi madre, la cual con mucho sacrificio me brindo educación y por apoyarme en todo para cumplir mis sueños, de lo cual es reflejo y recompensa lo que he logrado hasta el día de hoy.
- A mi amada esposa que ha estado a mi lado desde el comienzo de mis estudios de Ingeniería, por ser el pilar principal para el seguimiento de la misma, que con su apoyo constante, amor incondicional, su compañía en todo momento, palabras de animo y consejos me ha inspirado a cumplir una meta más en mi vida.
- A mi Hermanito Daniel por su amor y apoyo incondicional, a mis Abuelos por guiarme e inculcarme valores y a toda mi familia que me ha apoyado y animado a continuar con mi preparación.
- A Mis perros, bruno y kira, los cuales estaban a mi lado en las largas jornadas de trabajo y a altas horas de la noche, los cuales al ver que ya se habían dormido me indicaba que ya era muy tarde y que debía descansar también, por brindarme distracción y desestrés con sus juegos después de las jornadas de trabajo o momentos de frustración en que las cosas no salían muy bien.

De igual forma quiero agradecer y reconocer de manera muy especial a las siguientes personas, quienes contribuyeron de una forma u otra al desarrollo de la presente tesis:

- Al Dr. Arturo Espinosa Romero, quien fue director de la presente tesis, quien me ha orientado, apoyado y corregido en mi labor científica, toda mi admiración y respeto.
- Al M. Johan Estrada López quien fue mi asesor de tesis y profesor durante la maestría, gran consejero y asesor.
- A mis compañeros Francisco Quintal, Luís Aguilar, Erberth Castillo los cuales me apoyaron, ayudaron y dieron alternativas para resolver problemas relacionados al trabajo de tesis.
- A todos los profesores de la maestría, quienes transmitieron el conocimiento y la formación académica que hace posible el presente trabajo. En especial al Dr. Ricardo Legarda, ya que la única materia que me impartió fue de gran utilidad en el desarrollo de la tesis y me enseño a trabajar buscando la excelencia.
- Al Consejo Nacional de Ciencia y Tecnología (CONACYT), que me otorgó la beca con la cual pude sustentarme económicamente durante el estudio de la maestría.

Declaración

Por este medio declaro que yo escribí esta tesis y que describe el trabajo de mi tesis de Maestría en Ciencias de la Computación.

Ing. Irving Manuel Tolosa Garma
Mérida, Yucatán
México
15 de febrero de 2015

Índice general

Índice de figuras

Lista de Algoritmos

Índice de cuadros

Capítulo 1

Introducción

1.1. Introducción

La fotografía aérea consiste en capturar imágenes de la tierra desde puntos elevados, como pueden ser algunos accidentes geográficos, estructuras artificiales entre los que destacan postes y torres o siendo más común, vehículos aéreos y satélites. Las opciones anteriores pudieran no contar con la resolución necesaria ni el enfoque deseado para un uso específico, adicionalmente los costos de emplear vehículos aéreos o satélites suelen ser altos.

La fotografía aérea es utilizada comúnmente en geografía, vigilancia, arqueología y estudios ambientales. Una alternativa de bajo costo para obtener imágenes aéreas es usar globos o cometas con el propósito de elevar la cámara. Sin embargo, en este tipo de sistemas existe incertidumbre de lo que la cámara observa y no se tiene un control de la posición de la misma ya que se ve afectada por el viento. Las alternativas que se han desarrollado para ofrecer un mejor funcionamiento a este tipo de sistemas son: trabajos en los cuales la operación de la cámara se realiza mediante control remoto desde tierra o adquirir fotografías de manera periódica. Sin embargo, es un problema mucho más interesante el desarrollo de un sistema autónomo (que opere sin necesidad de la acción de un operador externo). Un ejemplo de este tipo de sistema se ha desarrollado en la universidad Autónoma de Yucatán (Sánchez, 2013; López, 2013), el cual es capaz de obtener fotografías adecuadas para realizar un mosaico de imágenes de manera autónoma. Este sistema se basa en la determinación de la posición y orientación de la cámara, lo cual se conoce como pose, con respecto a un marco de referencia

global. La pose se obtiene a través de sensores conectados al sistema, entre los cuales se utilizaron: unidades GPS (Global Positioning System), acelerómetros, giróscopos y magnetómetros, los cuales se encuentran embebidos en una tarjeta, la cual se conoce como IMU(Inertial Measurement Unit). Esta información es empleada por un modelo geométrico, el cual determina si la imagen a capturar se traslapa con alguna de las imágenes anteriores, si esta área es adecuada se captura la fotografía. Debido a que el modelo es completamente geométrico no garantiza que las fotografías tengan suficientes puntos de interés para la construcción del mosaico de imágenes. Este proyecto plantea complementar el sistema con un algoritmo de visión para detectar esquinas, el cual se empleará para determinar si existe información en la imagen que permita realizar el mosaico de imágenes con las fotografías capturadas. Este algoritmo debe de ejecutarse en tiempo real, por lo cual será paralelizado y ejecutado en la unidad SIMD (Single Instruction Multiple Data) NEON que se encuentra embebida en la familia de procesadores cortex ARM, tipo de procesador de la computadora embebida beagleboneblack[1], plataforma en la cual se ha desarrollado este proyecto.

1.2. Objetivos

A continuación se presentan los objetivos de la presente tesis: el objetivo general que es la meta a conseguir en el presente trabajo y los objetivos particulares a completar durante el desarrollo de la metodología, así también se presenta la estructura de la tesis y la organización de los capítulos.

1.2.1. Objetivo General

Diseñar e implementar un algoritmo paralelo para la detección de esquinas en imágenes utilizando la arquitectura SIMD de un sistema embebido para el análisis *in situ* de imágenes aéreas en tiempo real.

[1] `http://beagleboard.org/Products/BeagleBone`

1.2.2. Objetivos particulares

- Determinar el mejor método para detección de esquinas que se adapte a la arquitectura disponible.
- Implementar algoritmo de detección de esquinas en la plataforma de cómputo móvil.
- Integrar el detector de esquinas con el sistema de captura de imágenes basado en la pose.
- Realizar pruebas de campo para evaluar el funcionamiento de todo el sistema.
- Realizar un mosaico de imágenes con la fotografías obtenidas.

1.3. Estructura de la tesis

El presente trabajo se encuentra dividido por capítulos de la siguiente forma:

- El capítulo dos presenta el estado del arte, *i.e*, una recopilación de la información actualmente disponible sobre el tema.
- El capítulo tres presenta el marco teórico, antecedentes y fundamentos que se requieren y se utilizan en el desarrollo del proyecto.
- El capítulo cuatro está dedicado al desarrollo de la metodología, presentando la planificación, la teoría, el diseño y la implementación del sistema de automatización.
- En el capítulo cinco se presenta el diseño y resultados experimentales llevados a cabo implementando el sistema.
- En el capítulo seis se presentan las conclusiones del proyecto y el trabajo futuro.

Capítulo 2

Estado del Arte

En el presenta capítulo se reseña parte de la historia de la fotografía aérea, en especial la evolución de la fotografía con papalotes; se presentan algunos de los trabajos que se han hecho y sus aplicaciones hasta tiempos actuales. También se presentan los mosaicos de imágenes y algunos de los estudios que se han hecho con ellos, y por último se presenta una breve historia de los detectores de esquinas o puntos de interés, sus principales aplicaciones y trabajos que se han realizando para mejorar su tiempo de ejecución y lograr obtener aplicaciones en tiempo real.

2.1. Fotografía aérea

La primer fotografía aérea data del año 1858 y se le atribuye al francés Gaspar-Felix Tournachon, conocido como "Nadar", el cual capturó una imagen sobre el poblado Petit-Bicêtre en Francia desde un globo aerostático a una altura de 80 metros. El mismo Nadar había patentado la idea de utilizar la fotografía aérea para realizar los levantamientos topográficos y la elaboración de mapas (Alemán, 2013).

En el año de 1888 el fotógrafo Francés Arthur Batut tomó la primera fotografía aérea empleando un papalote para adquirir fotografías de su granja citewokipi, quien dos años más tarde publicaría el libro *La photographie aerienne par cerf − volant* (Fotografía aérea con papalotes) (Aber, 2008).

La primer fotografía aérea desde un avión data del 24 de abril de 1909, la cual fue capturada cerca de Roma durante uno de los vuelos de entrenamiento de la armada

italiana. Mas tarde, durante la primera Guerra mundial esta técnica fue empleada para levantar, corregir y mejorar sus mapas (Hepp et al., 2012).

En la siguiente imagen se puede observar la fotografía aérea más antigua que ha sido conservada, la cual consiste en una toma de la ciudad de Boston a una altura de 650 metros, la cual fue capturada por James Wallas Black en 1860.

Figura 2.1: Fotografía aérea más antigua que se conserva.

2.1.1. Fotografía aérea con cometas KAP (Kite aereal photography)

La primer persona en obtener una fotografía aérea utilizando un papalote fue Arthur Batut en 1888, el cual realizó una toma de su granja. El papalote empleado fue hecho de madera y papel, tenía una dimensión de 2.5 por 1.75 metros, la cámara empleada fue de una capa de vidrio fotosensible de 9 por 12 cm, mientras que el obturador era accionado por la acción de una mecha que, al consumirse después de haber elevado el papalote, lo liberaba y a su vez se desplegaba una bandera, esto con el fin de indicar que la fotografía había sido capturada.

Figura 2.2: Papalote utilizado por Arthur Batut y fotografía capturada.

En 1902 el ingeniero Ruso R. Thiéle creó el panoramógrafo, el cual es una estructura en forma hexagonal, en el cual se montaban 7 cámaras, una en el centro del mismo y las otras 6 en cada lado del hexágono, este instrumento fue empleado para obtener imágenes panorámicas (Mouchague & Mouchague, 2010).

Figura 2.3: Panoramógrafo.

Al transcurso de los años, los mecanismos de accionamiento por combustión del obturador de las cámaras fueron reemplazados por disparadores mecánicos y eléctricos, las cámaras fueron mejoradas, se desarrollaron estructuras para poner varias cámaras

en ellos y se desarrollaron sistemas de suspensión para la misma. Entre estos podemos mencionar el modelo Picavet, el cual debe su nombre a su creador Pierre L. Picavet en el año 1912, este sistema de cuerdas permite mantener la cámara viendo siempre a la superficie.

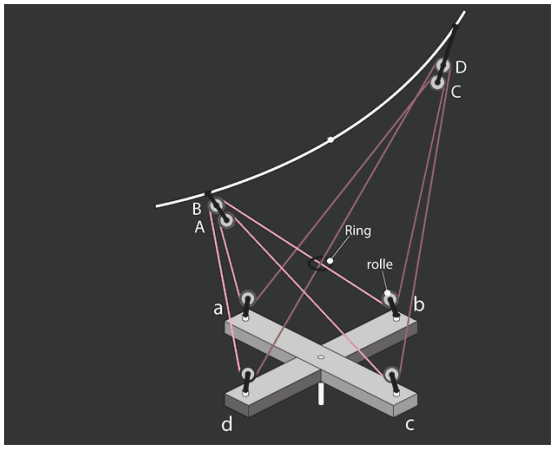

Figura 2.4: Picavet.

Desde ese entonces hasta el día de hoy, la fotografía aérea con papalotes ha evolucionado a la par con la tecnología. La fotografía aérea ha sido utilizada en diversas áreas, los Dres. (Aber & Aber, 2003) describen varias aplicaciones para esta técnica, la Dra. (Debra Eberts, 2005) ha empleado esta técnica para monitoreo de la vegetación, (M.Yakar et al., 2010) lo han empleado para realizar mediciones del nivel de agua en un lago, en arqueología ha sido empleado por el Dr. (Anderson, 2001), en arquitectura ha sido empleado para reconstrucción 3D por (Alemán, 2013), (Hepp et al., 2012) han empleado esta técnica para mejorar la coordinación y labores de rescate en zonas de desastres naturales, en fin, las aplicaciones que se le da a esta técnica son muy diversas y es aplicable a cualquier actividad en la cual las imágenes aéreas ayuden al desarrollo de la misma.

En la actualidad se han desarrollado diferentes trabajos para mejorar este tipo de sistemas, entre los cuales podemos mencionar el trabajo de (Oh & Green, 2004) quienes presentaron un sistema de visión teleoperada para la adquisición de imágenes aéreas, (M.Yakar et al., 2010), el cual emplea control remoto para el accionamiento de la cámara. En la Universidad Autónoma de Yucatán se ha desarrollado un sistema autónomo

capaz de determinar el momento para capturar una imagen; esto se basa en determinar la posición del sistema, con lo cual se estima el área a fotografiar y el traslape entre las imágenes, dado que el modelo de la toma de decisión para capturar una fotografía es completamente geométrico, no se garantiza que las imágenes capturadas tengan la suficiente información para poder realizar una construcción automática del mosaico de imágenes, para lo cual, en el presente trabajo se integra al sistema un algoritmo de visión computacional para analizar las imágenes *in situ* con una segunda cámara VGA y garantizar que las imágenes obtenidas cuenten con la información suficiente para generar un mosaico de imágenes de manera autónoma.

2.2. Mosaico de imágenes

Un mosaico de imágenes es la combinación de varias imágenes de una misma escena con elementos comunes; estos elementos repetidos que aparecen en las imágenes se usan para mezclar las imágenes individuales y producir una nueva imagen de mayor tamaño. Los orígenes de la creación de mosaico de imágenes va a la par de la aparición de la fotografía aérea, como se menciona en la sección anterior; este tipo de imágenes sirvió para realizar levantamientos cartográficos, con la obtención de varias imágenes que se unían para formar sus mapas. Para el año de 1902 el ingeniero Ruso R. Thiéle creó el panoramógrafo, sistema especialmente diseñado para obtener 7 imágenes desde diferente perspectiva con el objetivo de generar imágenes panorámicas, la creación de mosaico de imágenes también fue empleada para realizar levantamientos topográficos, tal como realizó el ingeniero geógrafo José Soriano Viguera en el Pirineo Central.

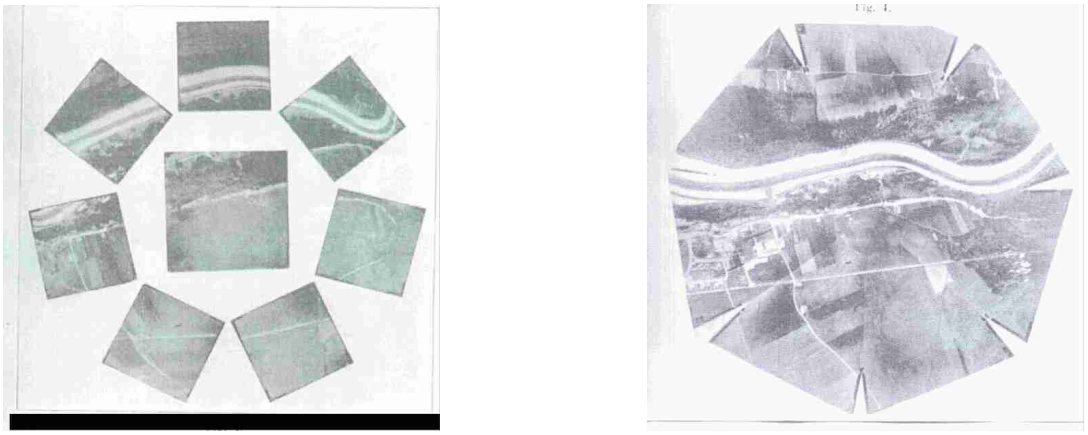

Figura 2.5: Fotos obtenidas por panoramógrafo y panorámica generada.

En la actualidad el uso de las imágenes panorámicas se han vuelto muy común, se cuenta con teléfonos celulares y cámaras capaces de realizar este tipo de imágenes, servicios que empleamos como *Google maps*[1] *y street view* usan esta misma tecnología para generar los mapas que nos presentan así como la navegación en el medio. (Gumustekin, 1999) nos presenta una lista de aplicaciones para la creación de mosaico de imágenes, entre las cuales se encuentran la creación de imágenes de alta resolución a un bajo costo, creación de espacios navegables virtuales, reconstrucción 3D, entre otros. Esta técnica también ha sido empleada para el desarrollo de aplicaciones médicas por el Dr. (Loewke KE et al., 2011).

2.3. Detectores de esquinas

Las esquinas son características locales importantes en una imagen, las cuales generalmente son alrededor del 0.05 % del total de los píxeles (Chen et al., 2009). De manera general podemos decir, que una esquina son aquellos puntos que tienen alta curvatura y que se encuentran en la unión de regiones con diferente brillo en la imagen.

De acuerdo con (Nixon & Aguado, 2012) un detector de esquinas puede ser considerado como un buscador de puntos en los cuales la curvatura tiene un valor muy alto. La curvatura representa la razón de cambio de la dirección del contorno de un objeto

[1] `https://maps.google.com`

y la forma más sencilla de encontrar este valor en imágenes digitales es mediante la medición del cambio del ángulo a lo largo de la curva. Esta aproximación fue considerada en las primeras técnicas de detección de esquinas: el cálculo consiste en encontrar la diferencia de dirección en el borde entre los píxeles conectados y puede expresarse como $K = \varphi_{t+1} - \varphi_{t-1}$, donde la secuencia de $\varphi_{t-1}, \varphi_t, \varphi_{t+1}, \varphi_{t-2}$..... representa la dirección del gradiente de una secuencia de píxeles que definen una curva.

Los detectores de esquinas son empleados como paso previo para encontrar correspondencias entre múltiples imágenes, ya que permite disminuir el número de píxeles a evaluar para encontrar dichas correspondencias, lo cual resultando muy útil en procesamiento de imágenes y aplicaciones de visión computacional. Algunas de las aplicaciones más notables son:

- Correspondencia entre imágenes estéreo
- Registro de imágenes
- Creación de mosaico de imágenes
- Detección y reconocimiento de objetos
- Seguimiento de movimiento
- Navegación de robots

(Parks & Gravel, 2004) nos presenta una variedad de diversas técnicas para encontrar esquinas, entre los cuales podemos mencionar: Beaudet, Moravec, Forstner, Plessey, SUSAN (siglas en inglés de smallest univalue segment assimilating nucleus) entre otros. Cada una de las técnicas tiene sus ventajas y desventajas, por lo cual cada método se adaptará de mejor o peor manera a cada aplicación en particular.

De acuerdo con (Chen et al., 2009), todos los detectores de esquinas pueden dividirse en dos clases principales: basados en contornos y basados en intensidad. Los métodos basados en contornos, primero recuperan los contornos de la imagen, posteriormente recorren los mismos en busca de curvaturas máximas o puntos de inflexión. Los métodos basados en intensidad estiman una medida para indicar la presencia de una esquina directamente de los valores de la imagen en escala de grises; este tipo de

algoritmos se caracterizan por ser más rápidos que los que se basan en contornos. Los 2 métodos basados en intensidad más representativos y más usados en la práctica son: SUSAN y el método desarrollado por (Harris & Stephens, 1988). (Chen et al., 2009) desarrolló un trabajo para comparar el desempeño de ambos algoritmos para encontrar correspondencia entre imágenes y unir las mismas, en su estudio se demuestra que para este tipo de aplicación Harris es mejor en: tiempo de ejecución, inmunidad al ruido, y repetibilidad (Chen et al., 2009).

(Teixeira et al., 2008) demostró que es posible acelerar el tiempo de ejecución del algoritmo de (Harris & Stephens, 1988) empleando unidades de procesamiento gráfico. También demostró que este algoritmo puede ser modificado para acelerar más el tiempo de ejecución, obteniendo como resultado aplicaciones en tiempo real.

De acuerdo a lo anteriormente mencionado se ha decidido para este trabajo implementar la detección de esquinas mediante el algoritmo de Harris.

Capítulo 3

Marco teórico

En el presente capítulo se describen las bases y fundamentos necesarios para un mejor entendimiento del desarrollo de la metodología del presente trabajo, así como los conceptos necesarios para entender el funcionamiento del sistema anterior sobre el cual se basa el presente proyecto y las adecuaciones que se hacen al mismo.

3.1. Generación de mosaicos

La generación de un mosaico de imágenes en visión computacional se refiere a la combinación de varias imágenes de una misma escena con elementos comunes. Estos elementos repetidos que aparecen en las imágenes se usan para mezclar las imágenes individuales y producir una nueva imagen de mayor resolución. Para encontrar estos elementos repetidos o correspondencias entre imágenes se deben examinar cada posible par de píxeles, lo cual tiene un alto costo computacional; en general, para dos imágenes de n píxeles cada una, la complejidad es $O(n^2)$ (Sonka & Boyle, 2008). Esta complejidad puede ser simplificada si se examina un menor grupo de puntos conocidos como puntos de interés (Sonka & Boyle, 2008).

De acuerdo a (Capel, 2001) todo algoritmo para hacer un mosaico de imágenes consta de tres fases principales:

- Registro de imágenes *(image registration)*
- Reproyección de imágenes *(reprojection)*

- Fusión de imágenes *(blending)*

El registro de imágenes consiste en encontrar un modelo geométrico que permita relacionar cada imagen con respecto a un sistema coordenado único global, el cual contendrá la escena completa. Posterior al registro se procede a realizar la reproyección, la cual consiste en transformar cada punto de cada imagen hacia el sistema coordenado global. Finalmente se procede a realizar la fusión de imágenes, que combina las imágenes en una sola usando las áreas de traslape como puntos en común para mezclarlas y finalmente obtener una imagen de mayor tamaño.

3.2. Detector de esquinas de Harris

En una esquina se esperan dos efectos importantes. La primera es tener valores de gradientes grandes. La segunda es que en un vecindario pequeño, la orientación del gradiente debe variar bruscamente.

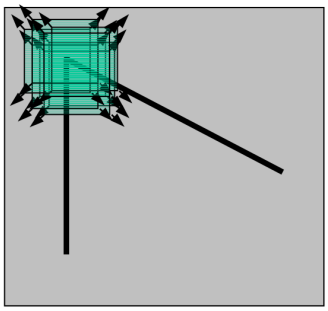

Figura 3.1: Características en una esquina.

El cambio de intensidad en cualquier dirección puede ser calculado mediante una sumatoria de diferencias de cuadrados:

$$D(u,v) = \sum (I(i+u, j+v) - I(i,j))^2 \tag{3.1}$$

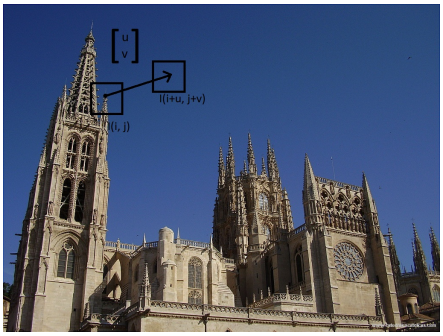

Figura 3.2: Cambio de intensidad.

Si consideramos u y v muy pequeños

$$
\begin{aligned}
I(i+u, j+v) &\approx I(i,j) + I_x u + I_y v \\
I_x = \frac{\partial I}{\partial x} \quad I_y &= \frac{\partial I}{\partial y} \\
(I(i+u, j+v) - I(i,j))^2 &= (I(i,j) + I_x u + I_y v - I(i,j))^2 \\
&= (I_x u + I_y v)^2 \\
&= I_x^2 u^2 + 2 I_x I y u v + I_y^2 v^2 \\
&= [u \ v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\
D(u,v) &= [u \ v] \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}
\end{aligned}
$$

$$
G = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \tag{3.2}
$$

donde la matriz G es conocida como tensor de la imagen y caracteriza los cambios de intensidad. Esta matriz nos ofrece una buena idea del comportamiento de la orientación en una ventana (Forsyth & Ponce, 2012). Los eigenvalores de esta matriz nos dicen que tipo de píxel tenemos de acuerdo a lo siguiente:

- Si $\lambda_1 \approx 0$ *y* $\lambda_2 \approx 0$, no es un punto de interés
- Si $\lambda_1 \approx 0$ *y* $\lambda_2 >> 0$ ó $\lambda_1 >> 0$ *y* $\lambda \approx 0$, es un borde
- Si $\lambda_1 >> 0$ *y* $\lambda_2 >> 0$, es una esquina

A continuación se describe el desarrollo algebraico para encontrar los eigenvalores de la

matriz G.

$$Gv = \lambda v$$

$$Gv - \lambda v = 0$$

$$v(G - \lambda I) = 0 \quad donde\ I\ es\ la\ matriz\ Identidad\ y\ \ v \neq 0$$

$$\det(G - \lambda I) = 0$$

$$si\ a = \sum I_x^2 \qquad b = \sum I_x I_y \qquad c = \sum I_y^2$$

$$\det\left(\begin{bmatrix} a & b \\ b & c \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) = 0$$

$$\det\begin{bmatrix} a-\lambda & b \\ b & c-\lambda \end{bmatrix} = 0$$

$$(a - \lambda)(c - \lambda) - (b)(b) = 0$$

$$ac - a\lambda - c\lambda + \lambda^2 - b^2 = 0$$

$$\lambda^2 + (-a - c)\lambda + ac - b^2 = 0\ aplicando\ formula\ general$$

$$\lambda_{1,2} = \frac{-(-a-c) \pm \sqrt{(-a-c)^2 - 4(1)(ac - b^2)}}{2(1)}$$

$$\lambda_{1,2} = \frac{a + c \pm \sqrt{a^2 - 2ac - 4b^2 + c}}{2}$$

$$\lambda_1 = \frac{\sum I_x^2 \sum I_y^2 + \sqrt{(\sum I_x^2)^2 - 2(\sum I_x^2)(\sum I_y^2) - 4(\sum I_x I_y)^2 + \sum I_y^2}}{2}$$

$$\lambda_2 = \frac{\sum I_x^2 \sum I_y^2 - \sqrt{(\sum I_x^2)^2 - 2(\sum I_x^2)(\sum I_y^2) - 4(\sum I_x I_y)^2 + \sum I_y^2}}{2}$$

(Harris & Stephens, 1988) se dieron cuenta que el cómputo exacto de los eigenvalores pudiera ser muy costoso por el cálculo de la raíz cuadrada, por esta razón propusieron la siguiente aproximación:

$$C(G) = det \mid G \mid -k * traza^2(G) \tag{3.3}$$

donde k es una constante usualmente pequeña, la cual favorece el gradiente de la imagen en una dirección o en ambas. Para darnos cuenta del efecto que tiene la constante k

supongamos los eigenvalores λ_1, λ_2 tenemos:

$$\begin{aligned} C(G) = \lambda_1\lambda_2 + k(\lambda_1 + \lambda_2)^2 &= \lambda_1\lambda_2 + k(\lambda_1^2 + 2\lambda_1\lambda_2 + \lambda_2^2) \\ &= \lambda_1\lambda_2 + k\lambda_1^2 + 2k\lambda_1\lambda_2 + k\lambda_2^2 \\ &= (1 + 2k)\lambda_1\lambda_2 + k(\lambda_1^2 + \lambda_2^2) \end{aligned}$$

notemos que si k es una valor grande y uno de los eigenvalores lo es, entonces $C(G)$ también será grande. Con esto aquellas regiones que tuviesen variación en al menos una dirección hará que $C(G)$ supera un umbral y nos indicará que tenemos una esquina, cuando en realidad tenemos un borde. Si k fuese pequeño es necesario que ambos eigenvalores sean lo suficientemente grandes para superar el umbral, de esta manera se asegura la variación del gradiente en ambas direcciones, y aquellas respuestas que superen este umbral serán nuestras esquinas. La literatura reporta que para valores de $0.04 < k < 0.15$ la aproximación de Harris demuestra resultados confiables.

3.3. Gradiente de la imagen

El gradiente de la imagen se denota como:

$$\nabla I(x, y) = [I_x(x, y), I_y(x, y)]^T$$

el cual se encuentra definido por un vector, el cual tiene como componentes sus derivadas parciales

$$I_x(x, y) = \frac{\partial I}{\partial x}(x, y), \qquad I_y(x, y) = \frac{\partial I}{\partial x}(x, y)$$

las cuales pueden ser estimadas mediante

$$\frac{\partial I}{\partial x} = \lim_{\delta_x \to 0} \frac{I(x + \delta_x, y) - I(x, y)}{\delta_x} \approx I_{i+1,j} - I_{i,j}.$$

De manera análoga se tiene que $\frac{\partial I}{\partial y} \approx I_{i,j+1} - I_{i,j}$, a esta forma de estimar las derivadas se le conoce como diferencias finitas. El cálculo de las derivadas parciales mediante la aproximación anterior tiende a amplificar el ruido, dando como resultado un gradiente ruidoso, la manera de lidiar con esto es suavizando primero la imagen, para lo cual, comúnmente se utiliza un filtro Gaussiano (Forsyth & Ponce, 2012), la cual se utiliza para convolucionar con la imagen, posteriormente se procede a derivar, esto reduce el

ruido en los resultados. Sin embargo, debido a las propiedades de la operación convolución es posible diferenciar el filtro Gaussiano y posteriormente aplicar a la imagen, obteniendo el mismo resultado, lo cual se expresa como:

$$\frac{\partial(g_\sigma \circledast I)}{\partial_x} = \frac{\partial g_\sigma}{\partial_x} \circledast I \tag{3.4}$$

donde g_σ es el filtro Gaussiano.

3.4. Filtro diferencial Gaussiano

Un filtro Gaussiano de una dimensión incluyendo la desviación estándar como parámetro se define:

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi} * \sigma} e^{\frac{-x^2}{2\sigma^2}} \tag{3.5}$$

Si derivamos la expresión anterior obtenemos un filtro diferencial Gaussiano, el cual se define como:

$$\frac{\delta g_\sigma}{\delta x} = -\frac{x}{\sqrt{2\pi} * \sigma^3} e^{\frac{-x^2}{2\sigma^2}} \tag{3.6}$$

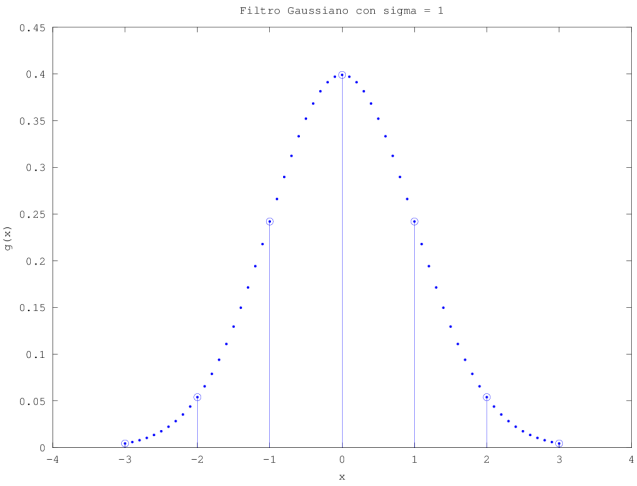

Figura 3.3: Gráfica de la función de Gauss con $\sigma = 1$.

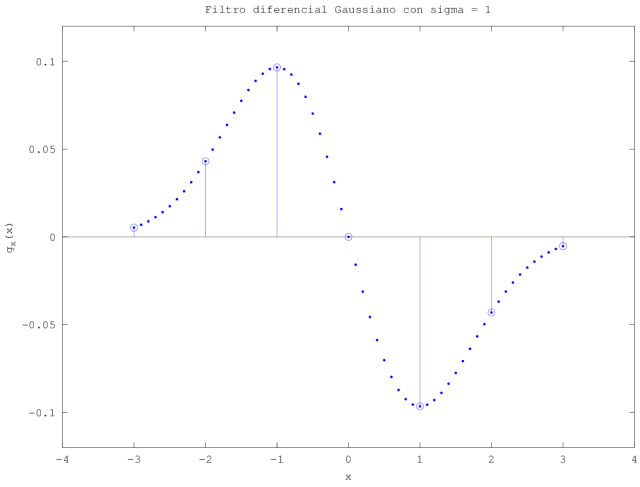

Figura 3.4: Gráfica de la derivada de la función de Gauss con $\sigma = 1$.

En una distribución de Gauss, el 99.6 % del total del área bajo la curva se encuentra en un rango de 6 desviaciones estándar, las cuales van de -3σ a 3σ, por tal motivo el tamaño de nuestro filtro se encuentra dado por:

$$g_{size} = 2(ceil(3\sigma)) + 1. \tag{3.7}$$

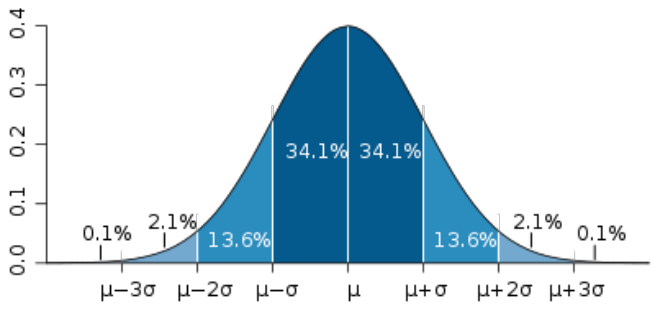

Figura 3.5: Gráfica de una distribución normal de Gauss.

3.5. Convolución

La convolución es un operador matemático que transforma dos funciones f y g en una tercera función, la cual representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

La convolución de f y g se denota como $f \circledast g$ y se define como la integral de ambas funciones después de desplazar una de ellas una distancia n.

$$f(t) \circledast g(t) = \int_{-\infty}^{\infty} f(n)g(t-n)dn$$

3.5.1. Convolución discreta

Para las funciones discretas, tal como el caso de nuestra representación discreta de las imágenes, la convolución es expresada como:

$$f[m] \circledast g[m] = \sum_{n} f[n]g[m-n] \tag{3.8}$$

3.6. Imagen integral

En el presente trabajo hacemos uso de la matriz integral para acelerar el tiempo de ejecución del algoritmo de detección de esquinas de Harris, ya que es necesario realizar una sumatoria de los valores de los píxeles de las derivadas parciales en una ventana de tamaño $n \times n$, lo cual implica un total de $(n \times n) - 1$ operaciones, empleando la matriz integral, el cálculo de esta sumatoria se reduce a tres operaciones.

La imagen integral $I_i(x, y)$ en el punto(x, y) es igual a la sumatoria de todos los píxeles que se encuentran a la izquierda y por encima del mismo, es decir, la sumatoria de todos los píxeles del rectángulo comprendido entre el primer píxel$(0, 0)$ y el píxel (x, y) de la imagen.

$$I_i(x, y) = \sum_{x'=0,\ y'=0}^{x'=x,\ y'=y} I(x', y') \tag{3.9}$$

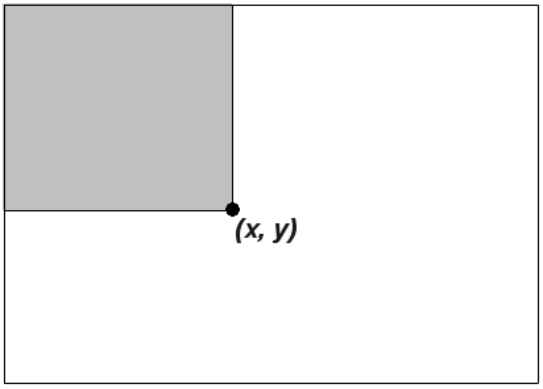

Figura 3.6: El valor de la imagen integral en el punto (x, y) es igual a la sumatoria de los píxeles en el área sombreada de la imagen original.

La matriz integral puede ser calculada mediante:

$$I_i(x,y) = I(x,y) + I_i(x-1,y) + I_i(x,y-1) - I_i(x-1,y-1) \tag{3.10}$$

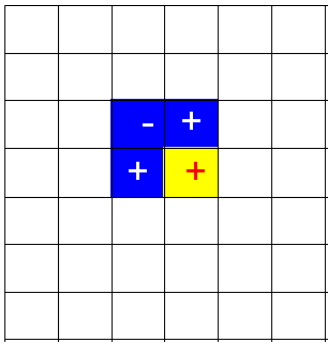

Figura 3.7: Elementos que intervienen en el cálculo del elemento (x,y) de la imagen integral I_i, donde los píxeles en azul pertenecen a I_i y el elemento amarillo pertenece a la matriz original I.

Otra alternativa para calcular la matriz integral es realizar primero una sumatoria por filas y posteriormente realizar una sumatoria por columnas (Pradip et al., 2010).

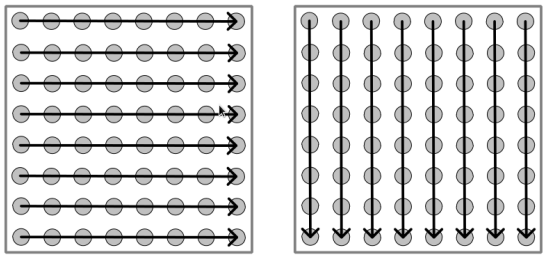

Figura 3.8: Cálculo de matriz integral.

Para el cálculo de la sumatoria por filas se usa la siguiente ecuación:

$$I_i(x,y) = \begin{cases} I(x,y) & x = 0 \\ I(x,y) + I_i(x-1,y) & x \neq 0 \end{cases} \tag{3.11}$$

Para el cálculo de sumatoria por columnas utilizamos la siguiente ecuación:

$$I_i(x,y) = \begin{cases} I(x,y) & y = 0 \\ I(x,y) + I_i(x,y-1) & y \neq 0 \end{cases} \tag{3.12}$$

Empleando la matriz integral es posible calcular la sumatoria de cualquier área rectangular de la imagen con 4 accesos a memoria y tres operaciones (Franklin, 1984; Paul Viola, 2001).

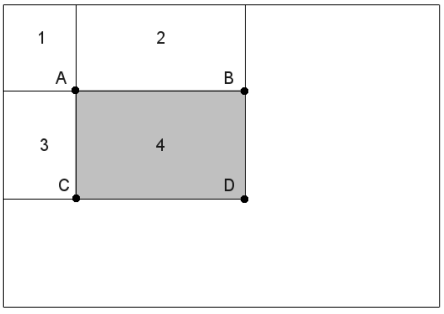

Figura 3.9: La sumatoria del área comprendida por el rectángulo 4 puede ser calculada empleando los valores de la imagen integral en los puntos A,B,C y D. El valor en el punto A representa el área del rectángulo 1, el valor en el punto B representa el área de los rectángulos A+B, el valor del punto C representa el área de los rectángulos A+C y el valor del punto D representa el área de los rectángulos A+B+C+D, el área del rectángulo 4 puede ser calculado como A-B-C+D.

3.7. Arquitectura de computadoras

De acuerdo con la taxonomía de Flynn, los procesadores pueden clasificarse de acuerdo al numero de operaciones concurrentes y en los flujos de datos disponibles en la arquitectura (Hwang et al., 1987). De acuerdo a esta clasificación los procesadores pueden ser:

- SISD (siglas en inglés de Single Instruction Single Data)
- SIMD (siglas en inglés de Single Instruction Multiple Data)
- MISD (siglas en inglés de Multiple Instruction Single Data)
- MIMD (siglas en inglés de Multiple Instruction Multiple Data)

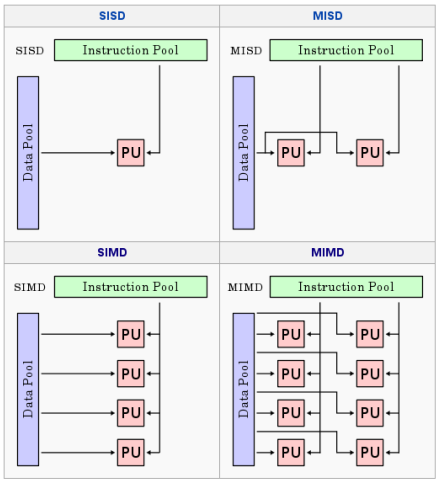

Figura 3.10: Taxonomía de Flynn. En donde cada PU representa una unidad de procesamiento.

3.7.1. Arquitectura SIMD

Los procesadores SIMD son un tipo de unidad de procesamiento en paralelo. En una arquitectura SIMD el paralelismo se logra por múltiples unidades de proceso llamadas Unidades de Procesamiento(PU por sus siglas en inglés), cada una de las cuales es capaz de ejecutar una operación especializada autónomamente. Este tipo de arquitectura se caracteriza por el hecho de que la misma operación es realizada en un momento dado sobre los datos contenidos en cada uno de las PUs.

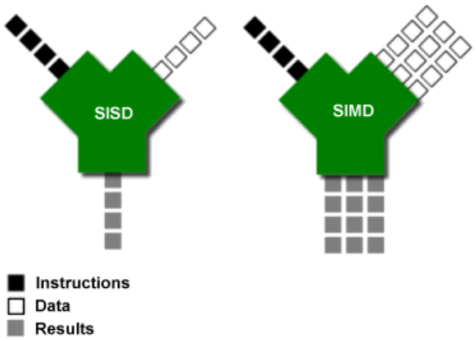

Figura 3.11: Diagrama funcionamiento arquitectura SIMD.

3.8. Conceptos geométricos

3.8.1. Anillo

Anillo será una sucesión de puntos en el plano (x,y), en donde el primer punto es también el último punto y la unión de todos los puntos daran como resultado una figura cerrada.

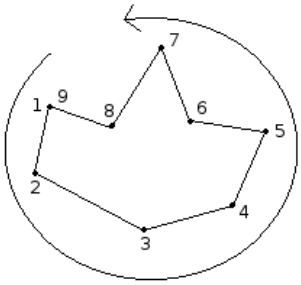

Figura 3.12: Anillo con nueve puntos

3.8.2. Polígono

Polígono será aquella figura plana cerrada definida por un borde exterior(anillo exterior) y que puede o no contener bordes interiores(anillos interiores).

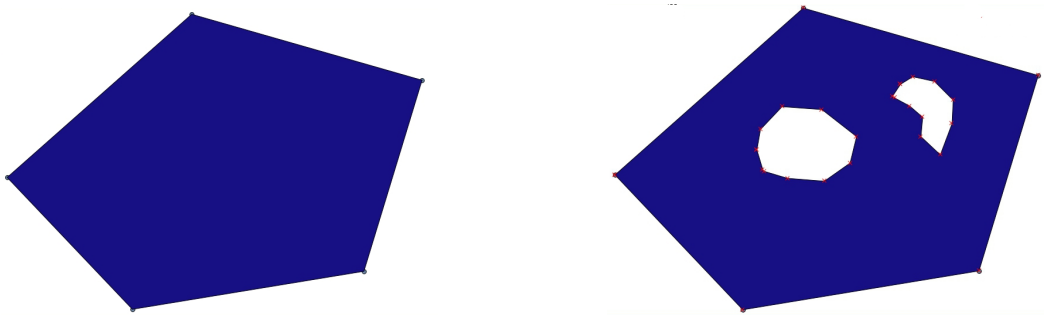

Figura 3.13: Polígono con y sin anillos interiores

Este concepto de polígono con huecos será de utilidad, ya que al ir capturando las fotos para el mosaico de imágenes es posible que se generen huecos entre los traslapes de las imágenes, los cuales serían los anillos interiores del polígono que se va generando al unir las imágenes.

3.8.3. Área de un polígono

El área de un polígono sera igual al área del polígono simple que forman los puntos del anillo exterior menos el área de los polígonos simples que forman los puntos de los anillos interiores.

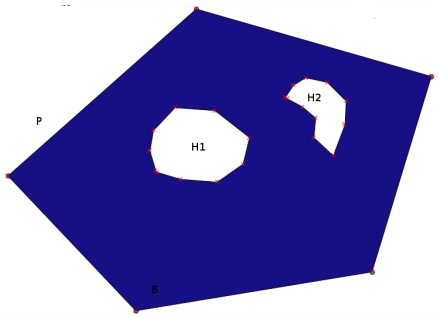

Figura 3.14: Área de polígono P con anillos interiores.El área total de la figura es Area(P) = area(B) - area(H1) - area(H2).

3.8.4. Unión de polígonos

La unión de dos polígonos da como resultado un nuevo polígono que espacialmente ocupará el área ocupada por ambos polígonos, sin importar si es área común o no.

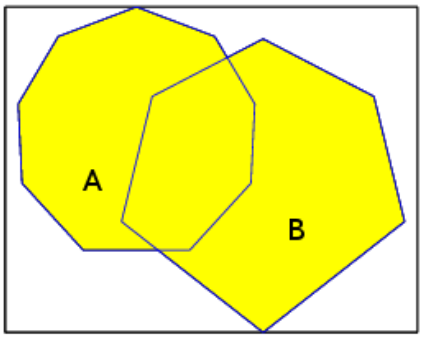

Figura 3.15: Unión de polígonos A y B.

3.8.5. Intersección de polígonos

La intersección de dos polígonos da como resultado un nuevo polígono que espacialmente será el área común de ambos, es decir, donde se superponen.

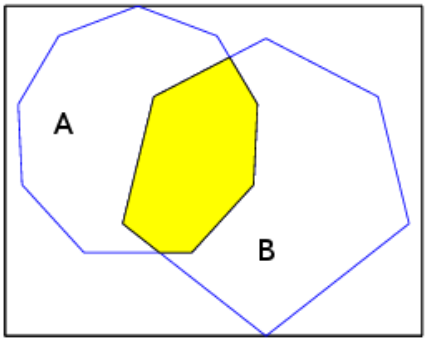

Figura 3.16: Intersección de polígonos A y B.

3.8.6. Diferencia de polígonos

La diferencia entre dos polígonos es el resultado de sustraer a un polígono el área de intersección con otro polígono, lo cual da como resultado un nuevo polígono.

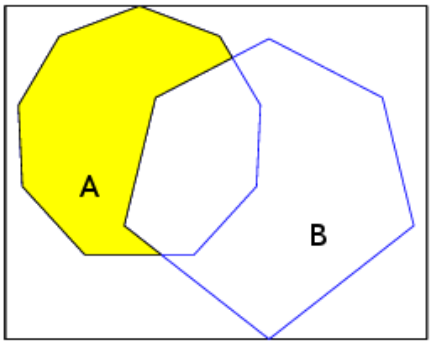

Figura 3.17: Diferencia de polígonos A y B.

Para el uso y manejo de estos conceptos en el presente proyecto se utiliza la biblioteca BOOST escrita en lenguaje C++. Una descripción general de esta biblioteca y las funciones para manejar estos conceptos se da en el apéndice A.

3.9. Geometría proyectiva

La geometría proyectiva es una rama de la geometría que nos ayuda a estudiar cómo se proyectan objetos tridimensionales del mundo real en un espacio bidimensional; a este último se le conoce como espacio proyectivo. Para facilitar el trabajo en este área se utilizan coordenadas homogéneas, las cuales permiten tratar todas las transformaciones geométricas como multiplicación de matrices.

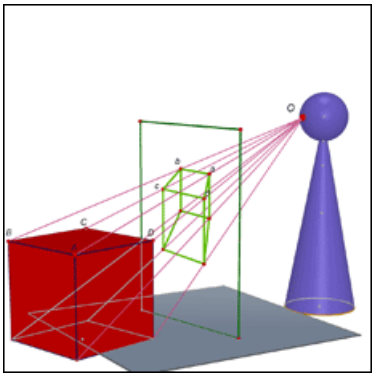

Figura 3.18: Representación de geometría proyectiva.

3.9.1. Coordenadas homogéneas

En coordenadas homogéneas, un punto n-dimensional está definido por $(n+1)$ coordenadas. De tal modo que un punto (x, y), se representa de forma homogénea como $(x/w, y/w, w)$. Por simplicidad se suele dar el valor 1 al tercer valor w.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.13}$$

En el caso de tres dimensiones, para un punto (x, y, z) se introduce una cuarta coordenada y se representa en forma homogénea como $(x/w, y/w, z/w, w)$, donde por simplicidad la cuarta coordenada introducida $w = 1$.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ z/w \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.14}$$

3.9.2. Transformaciones geométricas 2D

Existen 4 tipos de transformaciones geométricas, las cuales son:

- Isometría (Euclidiana)
- Similitud
- Transformación Afín
- Transformación Proyectiva

Isometría

La isometría es una transformación en el plano $\mathbb{R}^2$ que preserva la distancia euclidiana y se representa como:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \varepsilon cos\theta & -sen\theta & t_x \\ \varepsilon sen\theta & cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.15}$$

en donde $\varepsilon = \pm 1$. Si $\varepsilon = 1$ entonces la orientación se preserva y si $\varepsilon = -1$ se invierte la orientación.

Una transformación euclidiana puede ser expresada de manera mas concisa como

$$X' = HX = \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} X, \tag{3.16}$$

donde R es una matriz de rotación de 2×2 (una matriz ortogonal tal que $R^T R = R^T = I$), T es un 2-vector de traslación y 0 es un 2-vector nulo. Para este tipo de transformación existen dos casos especiales, el primero ocurre cuando $T = 0$, se conoce como rotación pura(sin traslación), mientras que el segundo caso ocurre cuando $R = I$ y se habla de una traslación pura(sin rotación). Una transformación euclideana tiene tres grados de libertad, con lo cual son necesarios tres parámetros para definirla, un ángulo θ para la rotación y dos valores t_x y t_y para la traslación.

En esta transformación se preservan la distancia entre dos puntos, el ángulo entre dos líneas y el área.

Similitud

Una similitud es una isometría a la cual se le agrega una variación en escala, con lo cual esta trasformación tiene cuatro grados de libertad: los tres de una isometría y el cambio de escala. Una similitud conserva la forma, los ángulos, las razones entre longitudes y áreas, es representada como:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} Scos\theta & -Ssen\theta & t_x \\ Ssen\theta & Scos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3.17}$$

Esto puede expresarse de forma más concisa como:

$$X' = H_S X = \begin{bmatrix} SR & T \\ 0^T & 1 \end{bmatrix} X, \tag{3.18}$$

donde el escalar S representa el factor de escala.

Transformación afín

Una transformación afín es una transformación lineal no singular seguida de una traslación. Se representa como:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3.19}$$

Se puede expresar de manera general como:

$$X' = H_A X = \begin{bmatrix} A & T \\ 0^T & 1 \end{bmatrix} X, \tag{3.20}$$

donde A es una matriz de 2×2 no singular y puede ser descompuesta como

$$A = R(\theta)R(-\phi)DR(\phi), \tag{3.21}$$

en donde $R(\theta)$ y $R(\phi)$ son rotaciones con θ y ϕ respectivamente y D es la matriz diagonal:

$$D = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}. \tag{3.22}$$

Así, la matriz A es la concatenación de una rotación por ϕ, posteriormente aplicar un cambio de escala por λ_1 sobre el eje de rotación x y λ_2 sobre el eje y, seguido de un regreso en la rotación por $-\phi$ y finalmente otra rotación por θ.

La transformación afín tiene seis grados de libertad, a saber: $\theta, \phi, \lambda_1, \lambda_2, t_x$ y t_y. Esta transformación preserva el paralelismo entre lineas, la razón entre longitudes de lineas paralelas y la razón entre áreas.

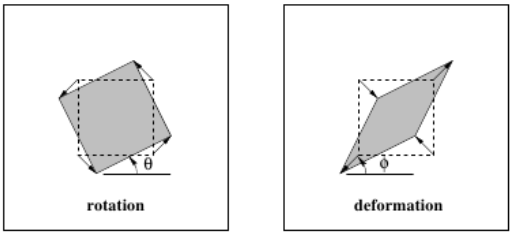

Figura 3.19: Transformación afín.

Transformación proyectiva

Es la forma general de una transformación lineal no singular de coordenadas homogéneas. Esto generaliza una transformación afín. Su representación es:

$$X' = H_P X = \begin{bmatrix} A & T \\ V^T & v \end{bmatrix} X, \tag{3.23}$$

donde $V = (v_1, v_2)^T$ y $v = 1$, lo cual nos deja ocho parametros para definir en la matriz, dando como resultado una transformación de ocho grados de libertad. Esta transformación preserva la colinealidad y la razón cruzada, la cual se encuentra definida como:

$$Cross(x_1, x_2, x_3, x_4) = \frac{|\, x_1 x_2 \,|}{|\, x_1 x_3 \,|} \cdot \frac{|\, x_3 x_4 \,|}{|\, x_2 x_4 \,|} \tag{3.24}$$

en donde $x_1,\ x_2,\ x_3,\ x_4$ son colineales.

Resumen de las transformaciones

A continuación se enlistan los tipos de transformaciones junto con sus propiedades .

Transformación	Matriz H	Invariantes
Euclideana	$\begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix}$	distancia entre dos puntos, ángulo entre dos líneas, área
Similitud	$\begin{bmatrix} SR & T \\ 0^T & 1 \end{bmatrix}$	ángulos entre rectas, razón entre distancias
Afín	$\begin{bmatrix} A & T \\ 0^T & 1 \end{bmatrix}$	razón de áreas y longitudes,el paralelismo
Proyectiva	$\begin{bmatrix} A & T \\ V^T & v \end{bmatrix}$	razón cruzada

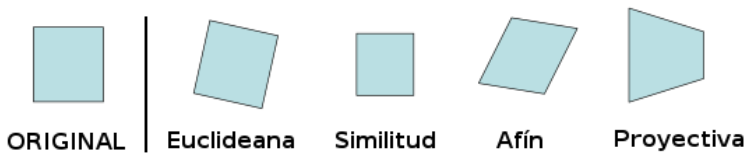

Figura 3.20: Transformaciones proyectivas 2D.

3.9.3. Transformaciones geométricas 3D

Traslación

La traslación se define a partir de un vector 3D con los valores X, Y, Z que se desea trasladar:

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

Si se desea expresar una traslación en 3D en forma matricial se recurre a coordenadas homogéneas, una matriz de traslación está definida como:

$$T = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.25}$$

así, la traslación se hace multiplicando la matriz de traslación por el punto que se desea trasladar en coordenadas homogéneas:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.26}$$

Rotación

Para definir una matriz de rotación se debe designar un eje de rotación respecto del cual se girará, y un ángulo de rotación θ. Una rotación tridimensional se puede especificar alrededor de cualquier línea en el espacio. Los ejes de rotación más fáciles de manejar son aquellos paralelos a los ejes de coordenadas. Una matriz de rotación de es 3×3 y para aplicar la transformación, de nuevo se recurre a coordenadas homogéneas para obtener la forma matricial. A continuación se muestran las formas matriciales para aplicar rotaciones alrededor de los ejes X, Y, Z.

La forma matricial de una rotación alrededor del eje X se define como:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.27}$$

La forma matricial de una rotación alrededor del eje y se define como:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.28}$$

La forma matricial de una rotación alrededor del eje y se define como:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.29}$$

Una matriz de rotación inversa se forma al sustituir el ángulo de rotación θ por $-\theta$. Los valores negativos para los ángulos de rotación generan rotaciones en sentido inverso, de modo que se produce la matriz identidad cuando se multiplica cualquier matriz de rotación por su inversa. Para realizar combinaciones de rotaciones sobre los ejes se multiplican las matrices de rotación de cada eje, resultando así, una sola matriz de transformación para las rotaciones. Sin embargo, hay que tener mucho cuidado, pues el orden en que se multiplican es muy importante, no es lo mismo la matriz $R_1 = R_x R_y R_z$ que, por ejemplo, la matriz $R_2 = R_z R_y R_x$.

Para que una matriz R sea una matriz de rotación debe cumplir dos propiedades: la primera, si la matriz R se multiplica por su transpuesta R^T, el resultado es la matriz identidad I; y segunda, el determinante de la matriz R debe ser igual a 1. Así, se define el espacio de matrices de rotación de 3×3 de la siguiente forma Ma et al. (2004):

$$SO(3) = \{R \in \mathbb{R}^3 | R^T R = I, det(R) = +1\}. \tag{3.30}$$

Matriz de transformación general

En general, una matriz de transformación homogénea es de la forma

$$M = \begin{bmatrix} R_{3\times3} & T_{3\times1} \\ P_{1\times3} & E_{1\times1} \end{bmatrix} = \begin{bmatrix} Rotacion & Traslacion \\ Proyeccion & Escalado \end{bmatrix} \tag{3.31}$$

Sin embargo, se considera que la proyección es nula y el escalado global es 1, así, la

matriz de transformación general queda como:

$$M = \begin{bmatrix} R_{3\times3} & T_{3\times1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Rotacion & Traslacion \\ 0 & 1 \end{bmatrix} \tag{3.32}$$

donde R es la matriz de rotación y T es el vector de traslación:

$$M = \begin{bmatrix} R_{0,0} & R_{0,1} & R_{0,2} & T_x \\ R_{1,0} & R_{1,1} & R_{1,2} & T_y \\ R_{2,0} & R_{2,1} & R_{2,2} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.33}$$

3.10. Funcionamiento del sistema de captura inteligente

El sistema a partir del cual se desarrolla este proyecto consiste en dos partes principales: la estación base y la estación aire. La estación base es una computadora en tierra que ejecuta una aplicación que sirve para enviar comandos y monitorear la estación aire mediante un Xbee(módulo de transmisión y recepción inalámbrica). La estación aire está compuesta por el GPS(global position system), IMU(inertial measurement unit), Xbee, cámara de alta resolución, los cuales se encuentran conectados a la Beaglebone Black, la cual básicamente ejecuta 2 tareas esenciales: la estimación de la pose del sistema y la captura autónoma de fotografías. Los elementos anteriores se encuentran montados en una estructura estabilizadora Picavet, la cual ayuda a que la cámara esté mirando siempre hacia la superficie terrestre.

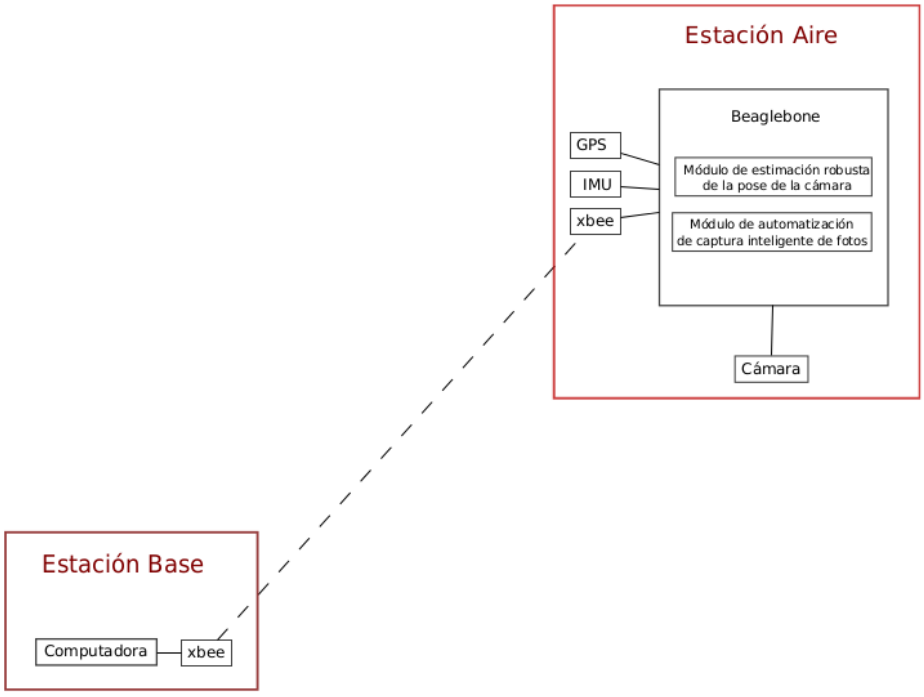

Figura 3.21: Esquema de Sistema.

La estimación de la pose se realiza mediante la lectura de los sensores GPS e IMU, realiza los cálculos necesarios para determinar la pose del sistema con respecto al marco de referencia terrestre en un instante determinado. El proceso de captura autónoma consulta la pose del sistema, en función de esta última realiza una proyección geométrica y calcula el área que se obtendría si en ese instante se capturara una fotografía, sí el área que se obtendría contribuye a tapar algún hueco generado al 100 % o existe algún traslape entre el 40 y el 60 % con alguna de las imágenes capturadas con anterioridad se captura la fotografía. Todo el sistema anterior se encuentra programado en un modelo que simula el funcionamiento de una máquina finita de estados, tal como se muestra en la figura 3.22.

En la Figura 3.22 se observa el diagrama correspondiente a la máquina de estados finitos. Las transiciones épsilon son el flujo normal del proceso, cuando se recibe otra entrada (IO, IP, tomarFoto, terminar) se mueve al estado correspondiente con esa transición. Desde cualquier estado se puede recibir la entrada terminar, lo cual lleva al estado final de la máquina de estados finitos.

Los estados sincronizarSensores y leerSensores corresponden al módulo de estimación de la pose de cámara y el resto corresponden al módulo de captura automática de imágenes.

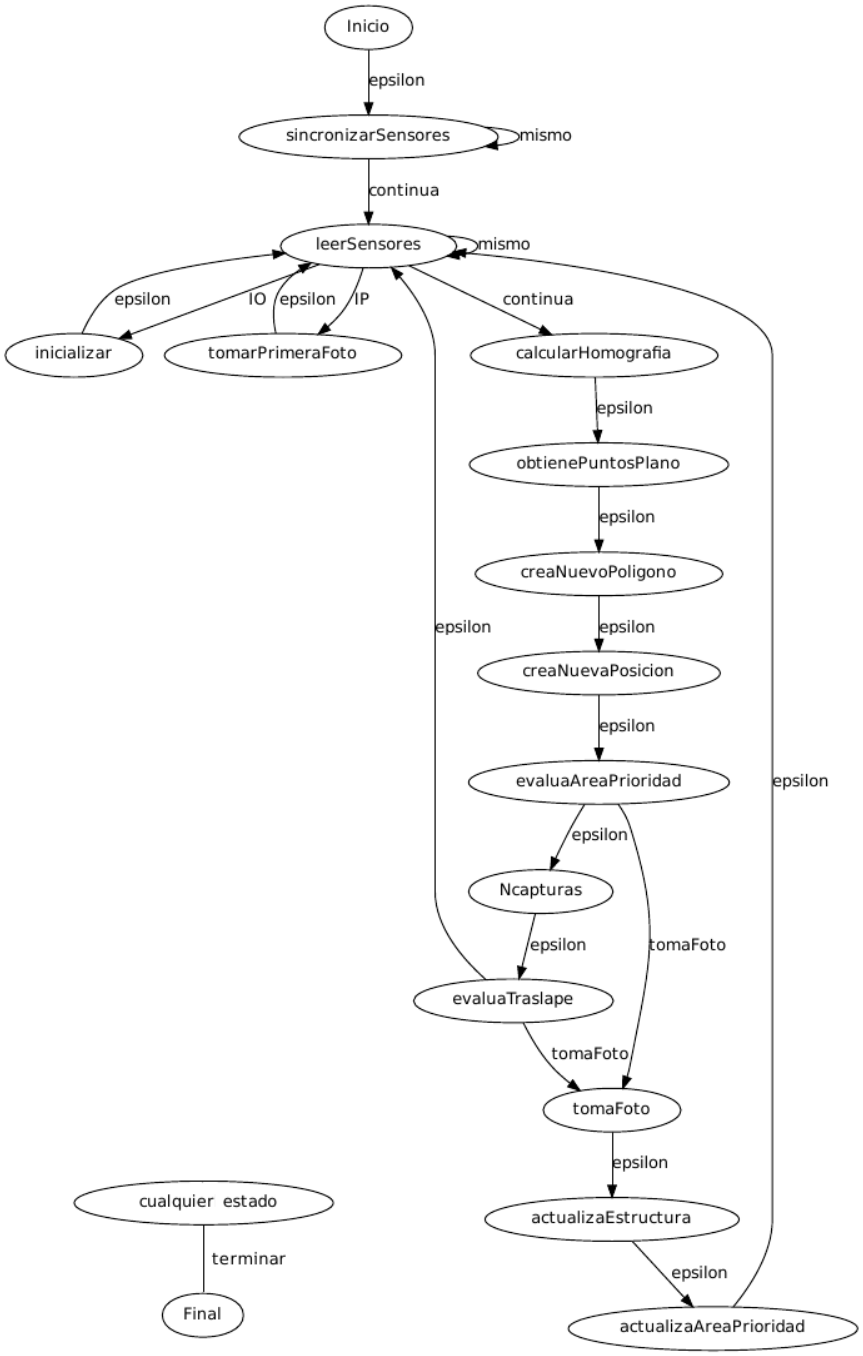

Figura 3.22: Diagrama de la máquina de estados finitos del sistema.

El presenta trabajo pretende complementar al sistema con un algoritmo de visión

computacional, el cual detectará esquinas a partir de imágenes adquiridas por una segunda cámara de resolución VGA(640×480), las esquinas detectadas en estas imágenes serán almacenadas en una estructura de consulta, con esto se busca mejorar la decisión de captura de las imagines de alta resolución, la cual ya no se basará únicamente en un modelo geométrico sino que también consultará la cantidad de información relevante en el área de traslape, con lo anterior el sistema quedará de manera esquemática de la siguiente manera:

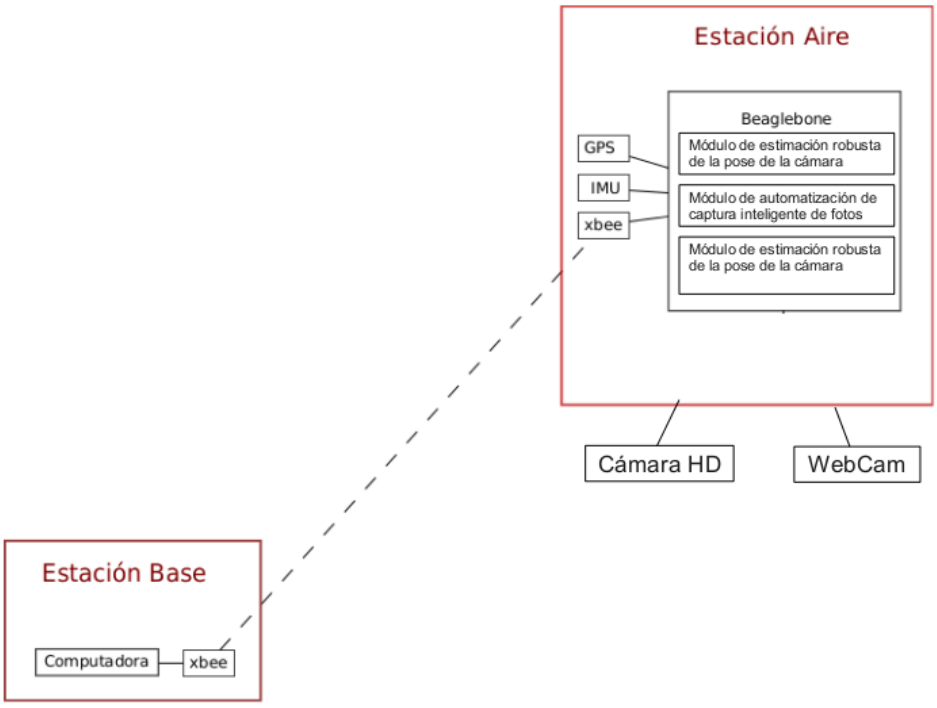

Figura 3.23: Esquema del Sistema Actualizado.

Capítulo 4

Metodología

En el presente capítulo se presente la metodología para el desarrollo y la implementación del detector de esquinas de Harris en la computadora embebida Beagelbone Black.

4.1. Detector de esquinas de Harris

La aproximación propuesta por (Harris & Stephens, 1988) que se describe en la ecuación (3.3) como:

$$C(G) = det \mid G \mid -K * traza^2(G)$$

donde:

$$G = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Para evaluar la ecuación (3.3) es necesario encontrar las derivadas parciales de la imagen I_x e I_y, en función de estas se calcula $I_x I_y$, I_x^2, I_y^2, sobre las cuales se calcula la sumatoria en ventanas. Una vez calculado los elementos anteriores es posible evaluar la ecuación (3.3) para obtener las esquinas.

En la siguiente imagen se ilustra en un diagrama los pasos a seguir para encontrar esquinas, en la cual I es una imagen a escala de grises y f representa el filtro diferencial

gaussiano empleado, el cual nos permite disminuir el ruido de la imagen y derivar al mismo tiempo, por otra parte, el variar el tamaño de la desviación estándar nos permite definir el tamaño de los objetos que deseamos destacar o resaltar en la imagen.

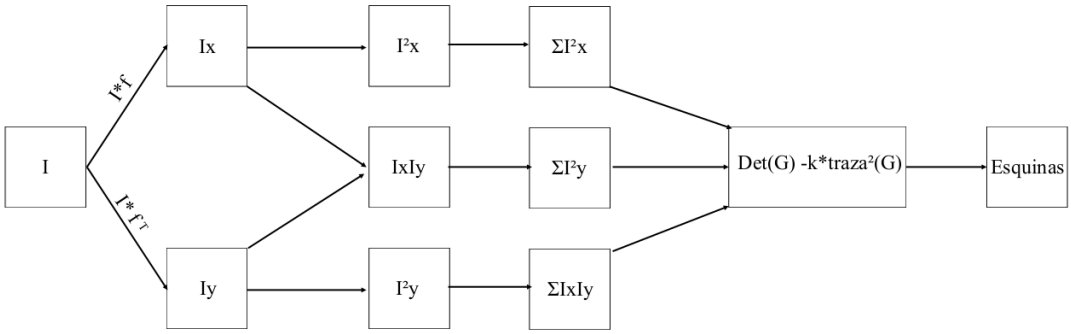

Figura 4.1: Pasos para encontrar esquinas en una imagen.

4.2. Derivadas parciales

Las derivadas parciales son calculadas de acuerdo a las siguientes ecuaciones:

$$I_x(x,y) = I(x,y) \circledast f(\sigma) \tag{4.1}$$

$$I_y(x,y) = I(x,y) \circledast f(\sigma)^T \tag{4.2}$$

4.2.1. Derivada parcial I_x

La ecuación (4.1) puede ser expresada y desarrollada de acuerdo al tamaño del filtro de acuerdo a:

$$I_x(x,y) = \sum_{i=-h}^{i=h} I(x,y+i) f_{i+h}$$

$$I_x(x,y) = I(x,y-h)f_0 + I(x,y-h+1)f_1 + I(x,y-h+2)f_2 \cdots + I(x,y)f_h$$
$$+ I(x,y+1)f_{h+1} + I(x,y+2)f_{h+2} \cdots + I(x,y+h)f_{2h}$$

donde:

g es el tamaño del filtro y $h = \frac{g-1}{2}$

En general para calcular cada elemento de la derivada parcial de la imagen se requieren g multiplicaciones y $g-1$ sumas, lo cual nos da un total de $2g-1$ operaciones.

Tomando en cuenta que en este trabajo se utiliza un filtro diferencial Gaussiano, el cual se encuentra definido por una función impar y que pasa por el origen, tenemos que:

$$\begin{aligned} f_0 &= -f_{2h} \\ f_1 &= -f_{2h-1} \\ f_2 &= -f_{2h-2} \\ &\vdots \\ f_{h-1} &= -f_{2h-(h-1)} \\ f_h &= 0 \end{aligned}$$

con lo cual podemos reescribir la expresión anterior como:

$$\begin{aligned} I_x(x,y) = {} & I(x,y-h)f_0 + I(x,y-h+1)f_1 + I(x,y-h+2)f_2 + \\ & I(x,y-h+3)f_3 \cdot\cdot\cdot\cdot\cdot + I(x,y)f_h - I(x,y+1)f_{h-1} - \\ & I(x,y+2)f_{h-2} + \cdot\cdot\cdot\cdot\cdot - I(x,y+h-2)f_2 - \\ & I(x,y+h-1)f_1 - I(x,y+h)f_0 \end{aligned}$$

$$\begin{aligned} I_x(x,y) = {} & f_0(I(x,y-h) - I(x,y+h)) + \\ & f_1(I(x,y-h+1) - I(x,y+h-1)) + \\ & f_2(I(x,y-h+2) - I(x,y+h-2)) + \\ & \vdots \\ & f_{h-2}(I(x,y-2) - I(x,y+2)) + \\ & f_{h-1}(I(x,y-1) - I(x,y+1)) \end{aligned}$$

de manera general la expresión anterior esta representada por:

$$I_x(x,y) = \sum_{i=1}^{i=h} f_{h-i}(I(x,y-i) - I(x,y+i)) \tag{4.3}$$

La expresión anterior nos permite calcular la convolución de la imagen con el filtro diferencial Gaussiano utilizando solamente la mitad de los coeficientes del mismo, realizar el cálculo de esta manera implica h restas, h multiplicaciones y $h-1$ sumas, para un total de $\frac{3}{2}(g-1)-1$ operaciones, lo cual implica una reducción en el tiempo en que se calcula cada elemento de la derivada parcial I_x.

Si suponemos una imagen $I(x,y)$ de $m \times n$ y tomando en cuenta el manejo de índices para arreglos en C++ tenemos que los valores de x y y se encuentran en el rango:

$$0 \leq x \leq m-1 \qquad 0 \leq y \leq n-1$$

al aplicar la ecuación anterior no existe ningún problema para los valores de x, sin embargo, para los valores de y se obtienen valores negativos para el caso que $y < i$, razón por la lo cual delimitamos el área donde se calculará I_x, para aquellas zonas donde exista un traslape total del filtro con la matriz, con esto, los valores que puede tomar la variable y, se encuentran en el rango:

$$h \leq y \leq n-1-h.$$

En la siguiente imagen se ilustra lo anteriormente mencionado y en color azul se muestra el área en el cual es posible calcular la derivada parcial I_x, los elementos en color blanco son puestos con valor igual a cero.

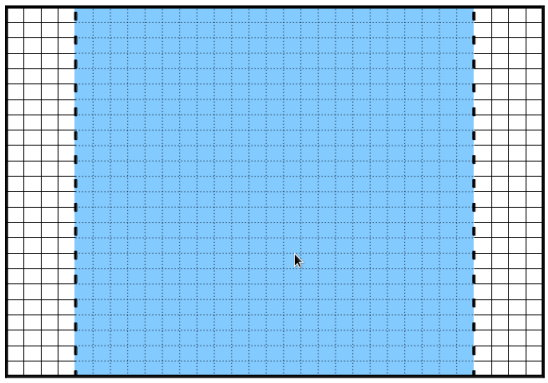

Figura 4.2: Área de la imagen sobre la cual se realiza el cálculo de I_x, el ejemplo ilustra el caso para el cual $h = 4$.

4.2.2. Derivada parcial I_y

La ecuación (4.2) puede ser expresada y desarrollada de manera análoga como se realizó para el calculo de la derivada parcial en I_x, con la diferencia que la variable a la cual se le sumaran los valores del iterador i es la variable y, con lo cual, la expresión para calcular I_y se encuentra dado por:

$$I_x(x,y) = \sum_{i=1}^{i=h} f_{h-i}(I(x-i,y) - I(x+i,y)) \tag{4.4}$$

para este caso, al aplicar la ecuación anterior no existe ningún problema para los valores de y pero si para x, donde se obtienen valores negativos para el caso que $x < i$, razón por la lo cual delimitamos el área donde se calculará I_y, para aquellas zonas donde exista un traslape total del filtro con la matriz, con esto, los valores que puede tomar la variable x, se encuentran en el rango:

$$h \leq x \leq m - 1 - h.$$

En la siguiente imagen se ilustra lo anteriormente mencionado y en color azul se muestra el área en el cual es posible calcular la derivada parcial I_y, los elementos en color blanco son puestos con valor igual a cero.

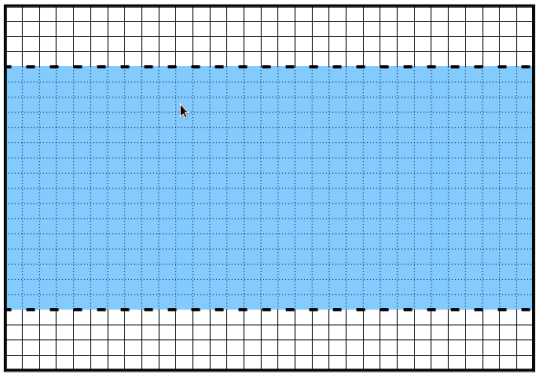

Figura 4.3: Área de la imagen sobre la cual se realiza el calculo de I_y, el ejemplo ilustra el caso para el cual $h = 4$.

4.3. Cálculo de $I_x I_y$

El cálculo del producto entre las derivadas parciales $I_x(x, y)$ e $I_y(x, y)$ es necesario para el cálculo de Harris, el cual se expresa como:

$$I_x I_y(x, y) = I_x(x, y) I_y(x, y), \tag{4.5}$$

de acuerdo a las consideraciones en los rangos de cálculo para $I_x(x, y)$ e $I_y(x, y)$, solo tendremos valores diferentes de cero para $I_x I_y(x, y)$, en aquellas coordenadas donde tanto I_x como I_y sean diferentes de cero, con lo cual podemos reducir el área de trabajo a:

$$h \leq y \leq n - 1 - h.$$

$$h \leq x \leq m - 1 - h.$$

En la siguiente imagen se ilustra lo anteriormente mencionado y en color azul se muestra en el área en el cual el cálculo de $I_x I_y$ es diferente de cero.

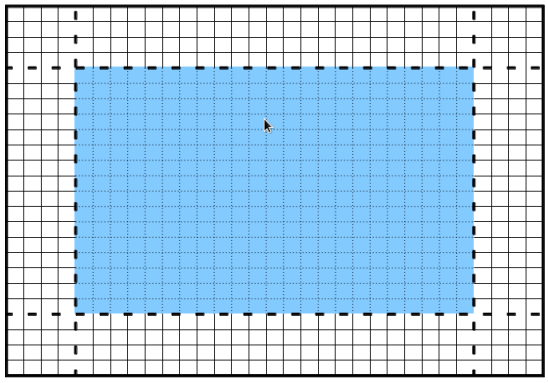

Figura 4.4: Área de la imagen sobre la cual se realiza el calculo de $I_x I_y$, el ejemplo ilustra el caso para el cual $h = 4$.

4.4. Cálculo I_x^2 e I_y^2

Este calculo consiste en elevar cada elemento de I_x e I_y al cuadrado, lo anterior lo podemos definir mediante las siguientes expresiones:

$$I_x^2(x,y) = I_x(x,y) * I_x(x,y) \tag{4.6}$$

$$I_y^2(x,y) = I_y(x,y) * I_y(x,y) \tag{4.7}$$

4.5. Sumatoria en ventanas w de $I_x^2,\ I_y^2, I_xI_y$

La sumatoria en ventanas consiste en sumar todos los elementos en la región que se cubre por la ventana, el resultado de esta sumatoria es colocada en el píxel central de la misma.

En la siguiente imagen se ilustra la ventana o región donde se calcula la sumatoria en la imagen, el resultado es colocado en el píxel central de la misma, el cual se muestra en la ventana de color rojo.

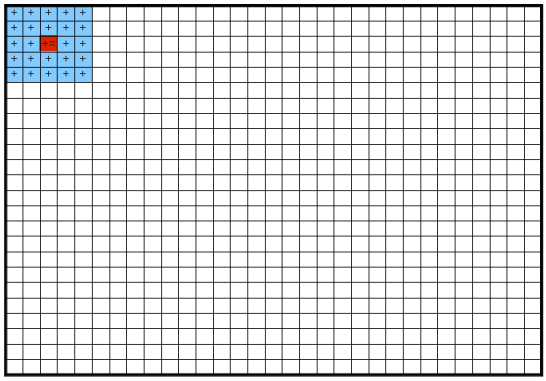

Figura 4.5: Sumatoria en una ventana de 5×5.

De manera general podemos definir el número de operaciones necesarias para realizar el cálculo de esta sumatoria en función del tamaño de la ventana, si suponemos que el tamaño de la ventana es: $w \times w$, el número de operaciones necesarias es $w^2 - 1$.

En el presente trabajo hacemos uso de la matriz integral para el cálculo de esta sumatoria, esto con el fin de reducir el número de operaciones y acelerar el tiempo de cómputo, con el uso de esta matriz el cómputo se reduce a 3 operaciones y 4 accesos a memoria.

4.5.1. Integral de las matrices $I_x^2,\ I_y^2, I_xI_y$

El cálculo de la matriz integral se divide en dos pasos, sumatoria por filas y posteriormente aplicar sumatoria por columnas, sin embargo este orden es posible cambiarlo debido a que este tipo de operación responde al principio de conteo doble. En este proyecto se opta por cambiar el orden con el objetivo de aprovechar la arquitectura NEON para acelerar el tiempo de cómputo.

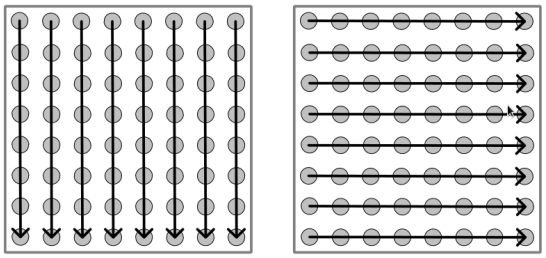

Figura 4.6: Forma de calcular matriz integral en este proyecto.

Para el cálculo de la sumatoria de columnas y filas utilizaremos las ecuaciones (3.11) y (3.12) que se mencionan en el capitulo anterior.

Una vez calculada la matriz integral, el cálculo de la sumatoria en una ventana se reduce a 4 accesos a memoria y 3 operaciones independiente del tamaño de la ventana.

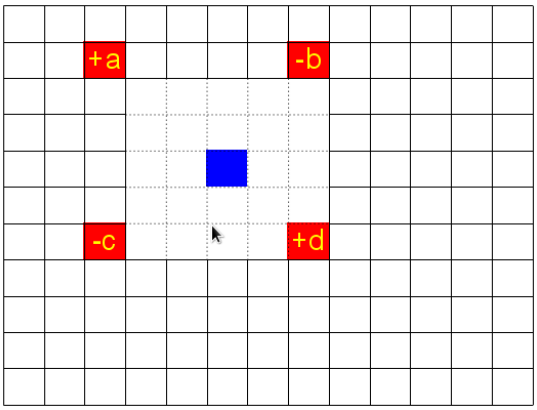

Figura 4.7: Calcular la sumatoria en una ventana consta de sumar los valores de los píxeles $a-b-c+d$, en la imagen se ejemplifica la sumatoria de una ventana de tamaño 5×5.

Si definimos $l = (w-1)/2$ e $Is(x,y)$ como la matriz donde se acumularán las sumatorias realizadas empleando la matriz $I_i(x,y)$, las coordenadas de los elementos están definidos como:

$$a = I_i(x-l-1, \quad y-l-1)$$

$$b = I_i(x+l, \quad y-l-1)$$

$$c = I_i(x-l-1, \quad y+l)$$

$$d = I_i(x+l, \quad y+l)$$

De manera general el cálculo de la sumatoria en ventanas se expresa mediante la siguiente ecuación:

$$\begin{aligned} Is(x,y) = & I_i(x-l-1, y-l-1) - I_i(x+l, y-l-1) - \\ & I_i(x-l-1, y+l) + I_i(x+l, y+l) \end{aligned} \tag{4.8}$$

donde para calcular las sumatorias en áreas donde exista un traslape total entre la ventana y la imagen integral:

$$l \leq x \leq m-l-1 \qquad\qquad l \leq y \leq n-l-1.$$

Sin embargo, aun con estas consideraciones al momento de calcular los elementos de la primera fila y primera columna de la matriz que almacenará los resultados de las

sumatorias, se hace referencia a índices fuera de la matriz integral. Podemos referirnos a 3 casos especiales, el cálculo de el primer elemento, el cálculo de la primera fila y el cálculo de la primera columna.

En la siguiente imagen se ejemplifican los casos anteriormente mencionados para una sumatoria de una ventana de 5×5. En color rojo podemos encontrar el primer elemento factible para calcular la sumatoria en ventanas, en color verde los elementos de la primera fila factible para calcular las sumatorias, en color amarillo los elementos de la primera columna factible y finalmente de color azul se encuentra los elementos generales y que se calculan mediante la ecuación anterior.

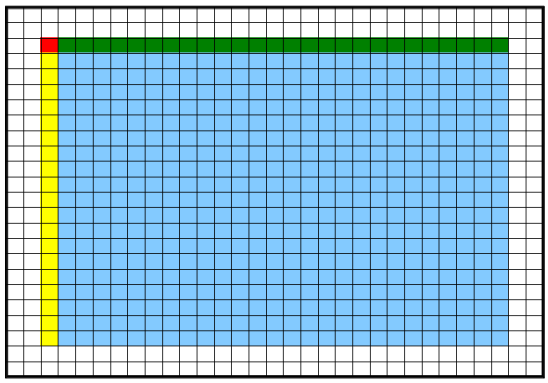

Figura 4.8: Casos para realizar sumatoria en ventanas empleando matriz integral.

Para el caso del primer elemento se tiene que existe un traslape total entre el área de la ventana y el cuadrado formado entre el primer elemento de la matriz integral y la esquina inferior derecha de la ventana, con lo cual, la sumatoria en esta ventana será igual al valor que se encuentra en la esquina inferior derecha en la matriz integral.

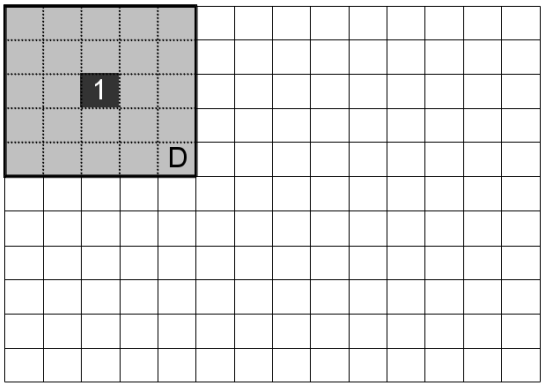

Figura 4.9: Caso 1, en el cual la sumatoria en la ventana es igual al valor D de la matriz integral.

De acuerdo a lo anterior, se establece que para calcular el primer elemento de la matriz de sumatoria por ventanas, el cual estará dado por las coordenadas (l, l), se emplea la siguiente ecuación:

$$Is(l, l) = I_i(2l, 2l)$$

Para el caso del cálculo de los elementos de la primera fila, se tiene que el traslape entre la ventana y el rectángulo que se forma con la esquina de la misma no es total, dividiendo el rectángulo formado en un cuadrado que se traslape totalmente con la ventana y otro rectángulo más pequeño que se encuentra a la izquierda de la ventana, es posible calcular la sumatoria de la ventana realizando una resta entre las 2 áreas de las figuras formadas.

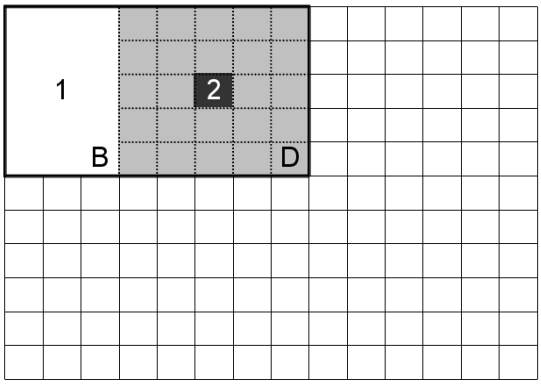

Figura 4.10: Caso 2, la sumatoria de la ventana comprendida por el cuadrado 2 será calculada empleando los valores de los puntos D y C de la imagen integral. El valor en el punto D representa el área del cuadrado 2 más el área del rectángulo 1, el valor en el punto C representa el área del rectángulo 1, con lo cual el área del cuadrado 2 puede ser calculada como D-C.

De acuerdo a lo anterior, se establece que para calcular los elementos de la primera fila de la matriz de sumatoria por ventanas, los cuales estarán dados por las coordenadas (l, y), se emplea la siguiente ecuación:

$$Is(l, y) = I_i(2l, y + l) - I_i(2l, y - l - 1)$$

Para el caso del cálculo de los elementos de la primera columna, se tiene un caso similar al anterior, con la diferencia que al dividir el rectángulo en un cuadrado y en un rectángulo más pequeño, este quedará en la parte superior del cuadrado, con lo cual es posible calcular la sumatoria de la ventana realizando una resta entre las 2 áreas de las figuras formadas.

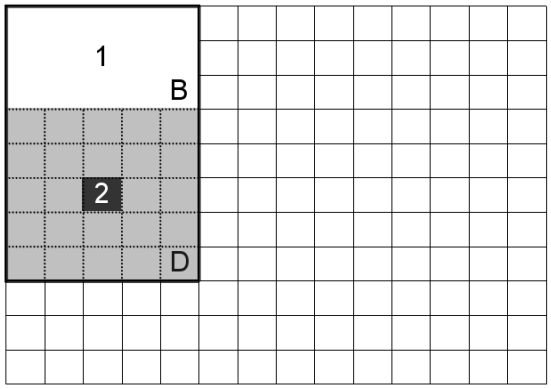

Figura 4.11: Caso 3, la sumatoria de la ventana comprendida por el cuadrado 2 será calculada empleando los valores de los puntos D y B de la imagen integral. El valor en el punto D representa el área del cuadrado 2 más el área del rectángulo 1, el valor en el punto B representa el área del rectángulo 1, con lo cual el área del cuadrado 2 puede ser calculada como D-B.

De acuerdo a lo anterior, se establece que para calcular los elementos de la primera columna de la matriz de sumatoria por ventanas, los cuales estarán dados por las coordenadas (x, l), se emplea la siguiente ecuación:

$$Is(x,l) = I_i(x+l, 2l) - I_i(x-l-1, 2l)$$

El resto de los elementos es calculado con la ecuación (4.8).

4.6. Evaluando la formula de Harris

Una vez que se han calculado las sumatorias en ventanas para I_x^2 I_y^2 e I_xI_y solo falta aplicar la formula de Harris para obtener las esquinas, lo cual se define mediante la siguiente ecuación:

$$\begin{aligned} C_G(x,y) =& (Is_x^2(x,y)Is_y^2(x,y)) - (Is_{xy}(x,y)Is_{xy}(x,y)) - \\ & k((Is_x^2(x,y) + Is_y^2(x,y))^2) \end{aligned} \tag{4.9}$$

la cual nos dará como resultado un valor que tendremos que comparar con un umbral, si el resultado es mayor a este último indicaremos que en este píxel existe una esquina, este umbral es establecido de manera experimental.

4.7. Definición de intervalos de trabajo

De manera general se pueden establecer tres dominios de trabajo diferente, el primero se refiere para el cálculo de las derivadas parciales y productos de las mismas, el cual se encuentra en función del tamaño del filtro. El segundo es para el cálculo de la matriz integral, el cual depende del tamaño del filtro. El tercero se refiere al área de trabajo donde es factible realizar la sumatoria en ventanas, el cual se encuentra en función del tamaño de la ventana.

Dado que para evaluar la formula de Harris son necesarios I_x^2, I_y^2 e I_xI_y, tenemos que los valores de $I_xI_y = 0$ en los bordes de la misma no son los valores reales de los mismos, esto se debe a la forma de calcular las derivadas parciales I_x e I_y, las cuales solo fueron calculadas en aquellas áreas donde existe un traslape total entre la imagen y el filtro, con lo cual se excluía un margen lateral de tamaño h para I_x y un margen superior e inferior a I_y, a los cuales se les asignó el valor de cero, de manera que si usamos los valores de los bordes obtendremos respuestas o esquinas incorrectas en esta área de la imagen, por tal motivo restringimos el área de trabajo para el cálculo de las derivadas parciales y el producto de las mismas, lo cual reducirá el numero de operaciones a realizar y por ende una disminución en el tiempo de cómputo. El primer intervalo de trabajo esta definido por:

$$h \leq x \ \leq m - h - 1 \quad y \quad h \leq y \ \leq n - h - 1$$

En la siguiente imagen se ilustra en color azul el área de trabajo del primer intervalo, en color amarillo se encuentra la zona en donde es posible calcular I_x y que no usaremos, en color rojo se encuentra la zona en donde es posible calcular I_y y que no usaremos, esto con el fin de evitar encontrar esquinas en donde no existen.

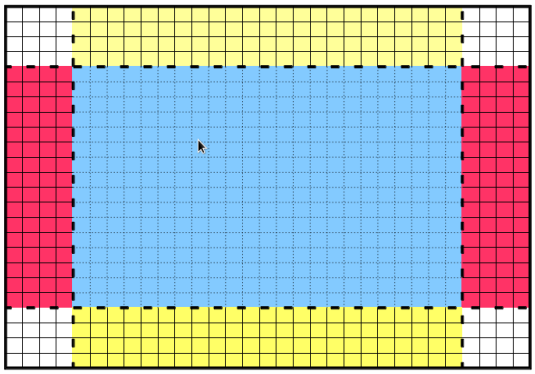

Figura 4.12: Área central de la imagen que se emplea para calcular I_x^2, I_y^2 e I_xI_y, ejemplificando para $h = 4$.

Para el segundo intervalo de trabajo hay que tomar en cuenta que el primer elemento en I_x^2, I_y^2 e I_xI_y, que pudiera ser diferente de cero es el elemento que se encuentra en la coordenada (l, l), por tal motivo no tiene importancia sumar elementos anteriores a este elemento para el cálculo de la matriz integral, con lo cual el segundo intervalo de trabajo esta definido por:

$$h \leq x \ \leq m-1 \quad y \quad h \leq y \ \leq n-1$$

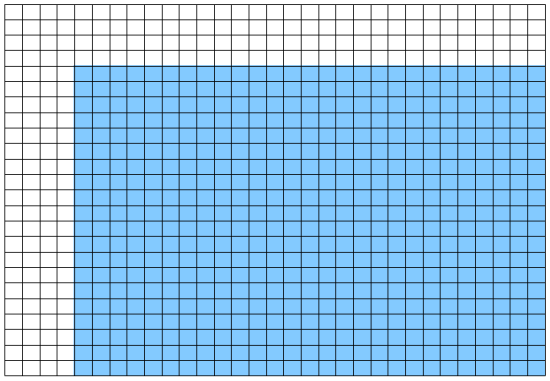

Figura 4.13: Área en la cual se realiza el cálculo de la matriz integral, ejemplificando para $h = 4$.

Para el tercer intervalo de trabajo se toma en cuenta el tamaño de la ventana de la cual se desea realizar la sumatoria, adicionalmente consideramos que el área de trabajo será en las regiones donde existe un traslape total entre parte de la imagen y la ventana,

con lo cual el tercer intervalo de trabajo esta definido por:

$$l \leq x \ \leq m-l-1 \quad y \quad l \leq y \ \leq n-l-1$$

Este último intervalo puede ser modificado para reducir el número de operaciones si $l < h$, con lo cual el tercer intervalo seria el mismo que el primer intervalo de trabajo, y si este fuera el caso, las sumatorias en ventanas se pudiesen calcular utilizando únicamente la ecuación general (4.8) para el cálculo de todos los elementos, ya que a los elementos que se harán referencia serán valores iguales a cero y no contribuirán a incrementar el resultado de la sumatoria.

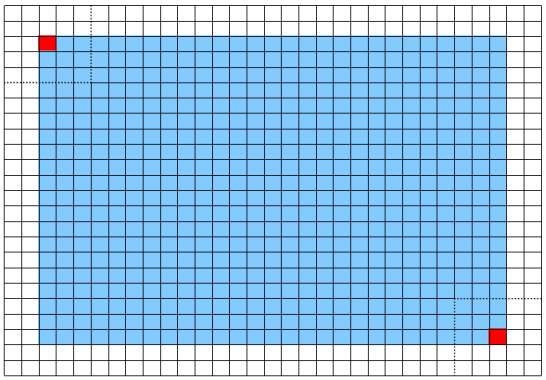

Figura 4.14: Área en la cual se realiza el cálculo de la sumatoria en ventanas, donde los elementos en color rojo representan el primer y último elemento en el cual existe un traslape total de la ventana con la imagen, ejemplificando para $w = 5$.

4.8. Implementación de algoritmo en arquitectura SIMD NEON

El procesador vectorial NEON del procesador ARM cuenta con 16 registros de 128 bits, los cuales pueden ser usados también como 32 registros de 64 bits. Este procesador es capaz de manejar enteros de 8, 16 y 32 bits, así como flotantes de 32 bits.

En el presente trabajo se emplearon flotantes para evitar el desbordamiento que pudiera ocurrir si manejáramos enteros en los cálculos, lo cual nos permite tener como máximo cuatro datos en cada uno de los registros, dado que las operaciones del procesador NEON se aplican a registros, nos permitirá efectuar una misma operación a cuatro datos al mismo tiempo; como máximo podremos tener 64 valores divididos en

16 registros en el procesador NEON en un momento determinado, razón por la cual el procesamiento de una imagen de tamaño $m \times n$ tiene que ser dividido hasta lograr procesar toda la información.

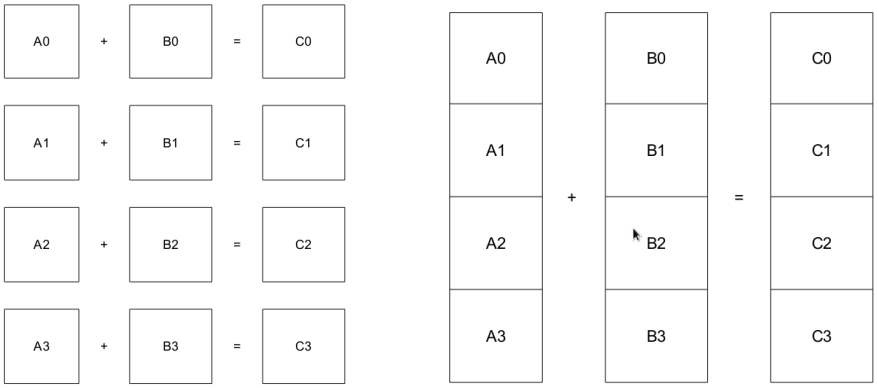

Figura 4.15: Se presenta a manera de diagrama en el lado izquierdo de la imagen la manera en que el CPU realiza una operación a 4 elementos de un vector y se compara con el diagrama de como se realiza la misma operación en el procesador NEON.

El proceso para realizar una operación en el procesador NEON consiste en transferir los valores de la memoria de la CPU al procesador vectorial, una vez que se tienen los valores en algún registro se aplica la operación a todos los elementos del mismo, terminada la operación se regresan los resultados a la memoria de CPU, siendo las transferencias de datos entre la memorias las operaciones más tardadas en el proceso, con lo cual en el presente trabajo se busca reducir el número de transferencias de los datos entre el CPU y la arquitectura NEON.

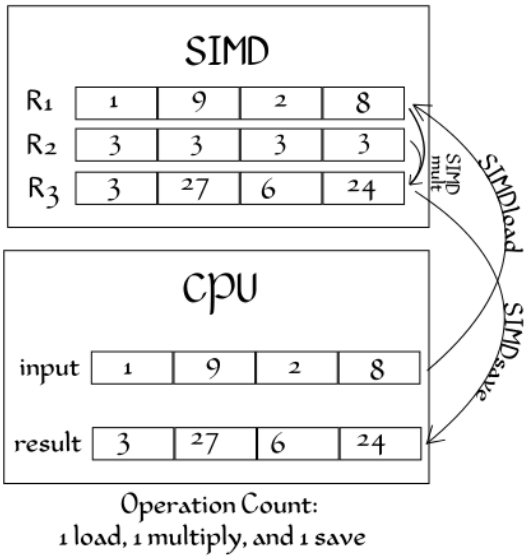

Figura 4.16: Diagrama de flujo de datos para realizar una operación en la arquitectura NEON.

Una de las limitaciones que se tiene a la hora de transferir datos de la memoria del CPU al NEON es que solamente permite cargar datos consecutivos, razón por la cual se opta por emplear una matriz transpuesta de la imagen para poder paralelizar el cálculo de la derivada parcial en x. En el presente trabajo se decidió utilizar un filtro diferencial gaussiano de longitud 5, este valor fue establecido porque se consideró que es un buen tamaño de filtro ya que en la imagen los detalles que se notarán como esquinas serán pequeños debido a la altitud en la que se encontrará el sistema, en caso de necesitar que los objetos a destacar sean más pequeños tenemos la opción de ampliar o disminuir el tamaño en que se capturarán las imágenes.

El proceso que a continuación se describe es la manera en que se implementó el algoritmo de Harris en la arquitectura NEON y que demostró realizar el cálculo en un menor tiempo, en el próximo capitulo se detallará la forma en que se llegó a este algoritmo. El primer paso es capturar la imagen y almacenarla en memoria en una arreglo, mismo que se duplica de manera transpuesta en un segundo arreglo; estos dos arreglos serán la entrada del algoritmo de detección de esquinas.

Para empezar la búsqueda de esquinas se procede a cargar los valores de la imagen

traspuesta a la arquitectura NEON en bloques de 5 filas $\times$ 4 columnas, donde a cada fila le corresponde ser multiplicada por cada uno de los coeficientes del filtro; sin embargo, en esta parte se empleó la ecuación (4.3), con lo que primero se realizó las restas correspondientes entre las filas, el resultado se almacenó en dos registros adicionales, cada uno de estos registro fue multiplicado por el coeficiente correspondiente del filtro, como paso final para el cálculo de I_x se procedió a sumar ambos registros, con esto se ha calculado 4 valores de I_x, los cuales son transferidos a un arreglo que se encuentra en la CPU, este proceso se repite hasta haber calculado completamente I_x, tal como se describe en pseudocódigo en el Algoritmo (2).

Para el cálculo de I_y se procede a pasar los datos de la imagen a la arquitectura NEON, los cuales se operan de manera análoga al proceso anterior para obtener 4 valores de I_y, a diferencia del proceso anterior, el resultado no se transfiere al CPU, en vez de esto se cargan los valores de I_x que corresponden a la misma posición de los valores que se acaban de calcular. Con I_x e I_y en la arquitectura se procede a calcular el producto de ambas para obtener I_xI_y, seguidamente se carga de la CPU el resultado superior de los elementos que se acaban de calcular para realizar la sumatoria entre ellos, siendo este resultado la sumatoria de columnas y el que se regresa a la CPU; este último proceso se utiliza para el cálculo de I_x^2 e I_y^2; todo el proceso anterior se encuentra en forma de pseudocódigo en el Algoritmo (3).

Terminado el proceso anterior se procede a pasar la información de cada una de las matrices generadas en bloques de 4 filas $\times$ 1 columna, la cual se suma con el bloque anterior, el resultado de esta suma se regresa a la CPU en el mismo lugar del bloque actual, al terminar de procesar todos lo bloques de la imagen se habrá concluido de calcular las matrices integrales de I_x^2, I_y^2 e I_xI_y, tal como se muestra en el Algoritmo (4).

Finalizado lo anterior se procede a calcular las esquinas de la imagen en bloques de 1 fila $\times$ 4 columnas, para ello es necesario cargar los cuatro vectores auxiliares correspondientes al bloque de cada una de las matrices integrales, con estos vectores auxiliares se calcula la sumatoria en la ventana de Harris, tal como se muestra en la Figura 4.7, esto se aplica a cada una de las matrices integrales, con estos tres resultados se procede a aplicar la formula de Harris y el resultado se regresa a la CPU, el proceso

anterior se describe en el Algoritmo (5).

Como último paso se normaliza el resultado a una escala de grises, la cual maneja valores entre 0 y 255, esto es para poder comparar el valor normalizado con un umbral que se establece, solo si es mayor al umbral diremos que se ha encontrado una esquina en la imagen. Como resultado final se genera una imagen binaria, en la cual los píxeles en color blanco corresponden a las esquinas detectadas en la imagen. Todo el proceso de encontrar esquinas puede observarse como pseudocódigo en el Algorimo (1).

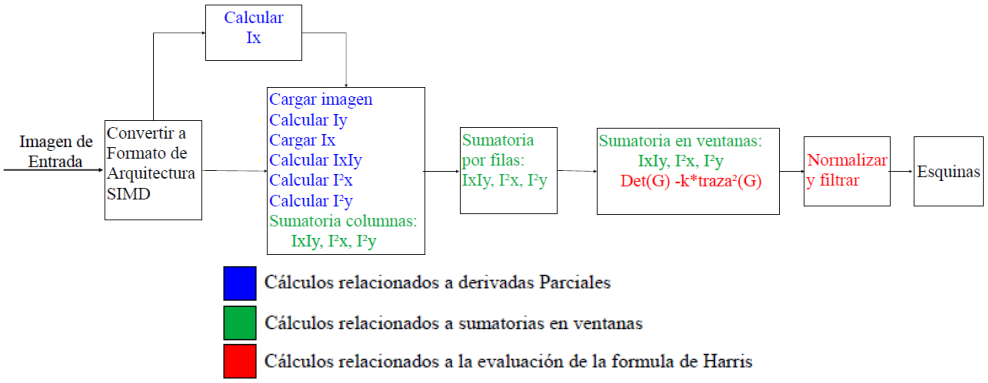

Figura 4.17: Diagrama de algoritmo de Harris implementado en NEON.

Algoritmo 1: Detección de esquinas implementado en arquitectura NEON.

Datos: Una imagen en escala de grises almacenada en un arreglo, valor s de la desviación estándar, tamaño w de ventana de Harris, constate proporcional k de Harris y valor de umbral para determinar si es o no una esquina, esto estará contenido en una estructura dataCornerV4LNEON.
Una estructura webcamV4LNEON que tendrá acceso a un dispositivo de captura de imágenes, y tendrá en un arreglo la información de la imagen capturada.

Resultado: Una imagen binaria, en la cual las esquinas encontradas en la imagen de entrada son representadas con píxeles blancos.

```
1  void Cap_find(dataCornerV4LNEON *dcorner, webcamV4LNEON * cam)
2  {
3      FiltroDiffGaussiano (dcorner->filtro, dcorner->s, dcorner->fsize)
4      cam->captureFrame()
5      ArrayInttoFloat32(cam->gray_image32, dcorner->imagen, dcorner->imagenTrans)
6      filtroIx(dcorner)
7      filtros(dcorner)
8      sumFilas(dcorner)
9      sumWindowNeon2(dcorner)
10     normaliza(dcorner->Corners)
11     filtra(dcorner->Corners, dcorner->resultado)
12 }
```

Algoritmo 2: Cálculo de la derivada parcial Ix (filtroIx)

Datos: Estructura dataCornerV4LNEON
Resultado: Derivada parcial de la imagen contenida en dataCornerV4LNEON.Ix

```
1  fsize = tamaño de filtro
2  filas = número de filas de la imagen - fsize
3  columnas = (número de columnas de la imagen - fsize + 1)/4
4  for (; columnas!=0; columnas- -)
5  |  Se cargan un block de fsize vectores auxiliares de la imagen traspuesta a la
   |  arquitectura NEON
6  |  for (; filas!=0; filas-=fsize)
7  |  |  Se calcula la derivada parcial de 4 elementos de la imagen empleando los
   |  |  vectores auxiliares
8  |  |  El resultado se regresa a la memoria RAM
9  |  |  for (i=1; i<fsize; i++)
10 |  |  |  Se carga un nuevo vector auxiliar de la siguiente fila y se almacena en el
   |  |  |  i-esimo vector auxiliar
11 |  |  |  Se calcula la derivada parcial de los siguientes 4 elementos factibles de la
   |  |  |  imagen empleando los vectores auxiliares
12 |  |  |  El resultado se regresa a la memoria RAM
13 |  |  end
14 |  end
15 |  filas = número de filas de la imagen - fsize
16 end
```

Algoritmo 3: Cálculo de la derivada parcial Iy, productos de derivadas y sumatoria columnas (filtros)

Datos: Estructura dataCornerV4LNEON
Resultado: Se calcula la derivada parcial de la imagen Iy, se realizan los productos IxIy, Ix^2, Iy^2 y se realiza la sumatoria de columnas de cada una de las anteriores, los resultados son almacenados en IxIy, Ix e Iy respectivamente.

```
1  fsize = tamaño de filtro
2  filas = número de filas de la imagen traspuesta - fsize
3  columnas = (número de columnas de la imagen traspuesta - fsize +1)/4
4  for (; columnas!=0; columnas- -)
5      Se cargan un block de fsize vectores auxiliares de la imagen a la arquitectura NEON
6      Se inicializa un vector de resultado anterior de IxIy con puros ceros a NEON
7      Se inicializa un vector de resultado anterior de Ix^2 con puros ceros a NEON
8      Se inicializa un vector de resultado anterior de Iy^2 con puros ceros a NEON
9      for (; filas!=0; filas-=fsize)
10         Se calcula Iy de 4 elementos de la imagen empleando los vectores auxiliares
11         Se cargan 4 valores de Ix correspondientes a la posición de los elementos de Iy
12         Se realiza el producto IxIy
13         Se eleva al cuadrado Ix
14         Se eleva al cuadrado Iy
15         Se suma el resultado anterior de IxIy con el actual
16         Se sobreescribe el resultado anterior de IxIy con el actual
17         El resultado se regresa a la memoria RAM
18         Se suma el resultado anterior de Ix^2 con el actual
19         Se sobreescribe el resultado anterior de Ix^2 con el actual
20         El resultado se regresa a la memoria RAM
21         Se suma el resultado anterior de Iy^2 con el actual
22         Se sobreescribe el resultado anterior de Iy^2 con el actual
23         El resultado se regresa a la memoria RAM
24         for (i=1; i<fsize; i++)
25             Se cargan 4 elementos de la siguiente fila en el i-esimo vector auxiliar
26             Se calcula la derivada parcial de los siguientes 4 elementos factibles de la
               imagen empleando los vectores auxiliares
27             Se cargan los 4 valores de Ix correspondientes a la posición de los elementos
               de Iy calculados
28             Se realiza el producto IxIy
29             Se eleva al cuadrado Ix
30             Se eleva al cuadrado Iy
31             Se suma el resultado anterior de IxIy con el actual
32             Se sobreescribe el resultado anterior de IxIy con el actual
33             El resultado se regresa a la memoria RAM
34             Se suma el resultado anterior de Ix^2 con el actual
35             Se sobreescribe el resultado anterior de Ix^2 con el actual
36             El resultado se regresa a la memoria RAM
37             Se suma el resultado anterior de Iy^2 con el actual
38             Se sobreescribe el resultado anterior de Iy^2 con el actual
39             El resultado se regresa a la memoria RAM
40         end
41     end
42     El último resultado de IxIy, Ix^2 e Iy^2 son copiados en todas las siguientes filas
43     filas = número de filas de la imagen traspuesta - fsize
44 end
```

Algoritmo 4: Sumatoria de Filas de IxIy, Ix^2, Iy^2 (sumFilas)

Datos: Estructura dataCornerV4LNEON
Resultado: Realiza la sumatoria en filas de IxIy, Ix^2, Iy^2, con lo cual se termina de calcular las imágenes integrales de cada una de las anteriores.

```
1  h = (tamaño de filtro -1)/2
2  filas = (número de filas de la imagen - h)/4
3  columnas = número de columnas de la imagen - h
4  for (; filas!=0; filas- -)
5  |  Se inicializa un vector resultado para sumatoria de Ix con ceros en NEON
6  |  Se inicializa un vector resultado para sumatoria de Iy con ceros en NEON
7  |  Se inicializa un vector resultado para sumatoria de IxIy con ceros en NEON
8  |  for (; columnas!=0; columnas- -)
9  |  |  Se carga a NEON una columna de 4 elementos de Ix^2
10 |  |  Se suma el resultado con los elementos recién cargados, el resultado se almacena
   |  |  en resultado Ix
11 |  |  Se regresa el resultado a la RAM en la misma dirección de la que se cargaron
12 |  |  Se carga a NEON una columna de 4 elementos de Iy^2
13 |  |  Se suma el resultado con los elementos recién cargados, el resultado se almacena
   |  |  en resultado Iy
14 |  |  Se regresa el resultado a la RAM en la misma dirección de la que se cargaron
15 |  |  Se carga a NEON una columna de 4 elementos de IxIy
16 |  |  Se suma el resultado con los elementos recién cargados, el resultado se almacena
   |  |  en resultado IxIy
17 |  |  Se regresa el resultado a la RAM en la misma dirección de la que se cargaron
18 |  end
19 |  columnas = número de columnas de la imagen - h
20 end
```

Algoritmo 5: Sumatoria de Ventanas y evaluación de formula de Harris(SumWin2)

Datos: Estructura dataCornerV4LNEON
Resultado: Realiza la sumatoria de las ventanas de IxIy, Ix^2, Iy^2, con esto aplica la formula de Harris y el resultado se almacena en Corners

```
1  h = (tamaño de filtro - 1)/2
2  offset = (h)*número de columnas + h // Primer elemento para evaluar sumatoria
3  wh = (tamaño de ventana - 1)/2
4  filas = número de filas de la imagen - tamaño de filtro + 1
5  columnas = (número de columnas de la imagen -tamaño de filtro + 1)/4
6  for (; filas!=0; filas- -)
7      offsetd = offset + wh + número de columnas * wh
8      offsetc = offsetd - tamaño de ventana
9      offseta = offsetc - tamaño de vantana * número de columnas
10     offsetb = offseta + tamaño de vantana
11     for (; columnas!=0; columnas- -)
12         Se cargan a NEON 4 elementos de Ix a partir de offseta
13         Se cargan a NEON 4 elementos de Ix a partir de offsetb
14         Se cargan a NEON 4 elementos de Ix a partir de offsetc
15         Se cargan a NEON 4 elementos de Ix a partir de offsetd
16         Se calcula la sumatoria de ventanas de Ix // a - b -c + d
17         El resultado se almacena en un vector result_x
18         Se cargan a NEON 4 elementos de Iy a partir de offseta
19         Se cargan a NEON 4 elementos de Iy a partir de offsetb
20         Se cargan a NEON 4 elementos de Iy a partir de offsetc
21         Se cargan a NEON 4 elementos de Iy a partir de offsetd
22         Se calcula la sumatoria de ventanas de Iy // a - b -c + d
23         El resultado se almacena en un vector result_y
24         Se cargan a NEON 4 elementos de IxIy a partir de offseta
25         Se cargan a NEON 4 elementos de IxIy a partir de offsetb
26         Se cargan a NEON 4 elementos de IxIy a partir de offsetc
27         Se cargan a NEON 4 elementos de IxIy a partir de offsetd
28         Se calcula la sumatoria de ventanas de IxIy // a - b -c + d
29         El resultado se almacena en un vector result_xy
30         Se calcula el determinante de G
31         Se calcula la traza al cuadrado de G
32         Se evalúa la formula de Harris
33         Se regresa el resultado a memoria RAM y se almacena en Corners
34         offseta +=4
35         offsetb +=4
36         offsetc +=4
37         offsetd +=4
38         offset +=4
39     end
40     offseta +=4
41     offsetb +=4
42     offsetc +=4
43     offsetd +=4
44     offset +=4
45     columnas = (número de columnas de la imagen -tamaño de filtro + 1)/4
46 end
```

4.9. Integración del detector de esquinas con el sistema de captura inteligente

Para integrar el detector de esquinas al trabajo previo desarrollado por (Sánchez, 2013) se instalará al sistema una segunda cámara, la cual servirá para adquirir imágenes de tamaño VGA, las imágenes capturadas por esta cámara son las que se emplearán para detectar esquinas, ya que si somos capaces de detectar esquinas(detalles) en esta resolución, también se encontrarán en las imágenes de alta resolución que se obtengan con la cámara principal. Las coordenadas de las esquinas que se encuentren en las imágenes son transformadas al punto que corresponden al plano terrestre, para esto se hizo uso de las coordenadas de Plucker. Estos puntos son almacenados en una estructura de árbol que provee la biblioteca Boost Geometry, árbol que el módulo de captura inteligente consultará para verificar si en el área de traslape existe un suficiente número de esquinas que permita una vez bajado el sistema la automatización de la generación del mosaico de imágenes, empleando las fotografías capturadas con la cámara de alta resolución.

El proceso de búsqueda de esquinas en el terreno a fotografiar y el almacenamiento de las mismas se realizará en un hilo secundario, esto para evitar que este proceso afecte el tiempo de respuesta del algoritmo principal, además que permite compartir la estructura del árbol donde se almacenarán las esquinas, mientras el hilo secundario se encarga de agregar esquinas al mismo, el hilo principal consultará el número de esquinas en un área.

El módulo de búsqueda de esquinas funciona de manera similar al modulo de captura inteligente, ya que se buscará que el terreno objetivo sea cubierto a su totalidad con las fotos de tamaño VGA, con lo cual se asegura que se hayan buscado esquinas en todo el terreno. Para ello se emplea un vector de polígonos (pueden contener huecos), los cuales no se intersectan entre si, el cual mantendrá la forma de toda el área que se ha fotografiado hasta ese momento y un polígono que representará el área que se obtendría si en ese momento se capturara una fotografía. La decisión de capturar una fotografía en un instante y buscar esquinas será, sí el área de este polígono no se traslapa con ninguno de los polígonos que se encuentran en el vector; sí se traslapa con un solo polígono del vector y el área de traslape no es mayor al 20 % de la nueva imagen; por último, sí se

traslapa con más de un polígono del vector. El primer caso nos garantiza que el 100 % de la información será nueva, el segundo caso nos garantiza que al menos el 80 % de la información será nueva, el último caso será porque esta imagen nos permitirá unir a todos los polígonos con los que se intersecta, con esto se reducirá el numero de polígonos en el vector. Después de tomar la foto y detectar esquinas en la misma, se agregaran las esquinas que no se encuentren en las áreas de intersección con otro polígono, con esto evitaremos sobre poblar las áreas de intersección con esquinas repetidas. El proceso anteriormente mencionado se muestra en pseudocódigo en el Algoritmo (6).

4.10. Algoritmo final

4.10.1. Pseudocódigo de algoritmo implementado en el hilo secundario

Algoritmo 6: Automatización del sistema de captura, búsqueda y almacenamiento de esquinas basado en la pose de la cámara.

Datos: Una secuencia de posiciones de la cámara (R,T), área total a cubrir.
Resultado: Un árbol que contiene las coordenadas de las esquinas detectadas en el terreno.

```
Crea una estructura para guardar la meta-información de las imágenes.
Crea una estructura que almacenará polígonos, la cual representará el área cubierta
Crea una estructura que controle la webcam, cam
Crea una estructura que almacena la información del detector de esquinas, data_es
Crea una estructura que almacena la información de las coordenadas en donde se
  detectan esquinas
Inicializa variables.
Cap_find(&data_es, &cam) // captura imagen y busca esquinas
Proyectar esquinas detectadas en la imagen capturada
Activar Mutex para bloquear el acceso al árbol mientras se guarda información
Almacenar esquinas en el árbol
Desactivar Mutex para desbloquear el acceso al árbol
while El área total no ha sido cubierta do
    Recibe la posición actual de la cámara, R,T
    Calcula la Homografía
    Crea un nuevo polígono
    Evalúa con cuantos polígonos del área cubierta se traslapa
    if No se traslapa con ninguno de los polígonos then
        Cap_find(&data_es, &cam)
    else
        if Se traslapa con mas de uno then
            Cap_find(&data_es, &cam)
        else
            if Se traslapa con solo un poligono then
                Calcula el área de traslape del nuevo polígono
                if El área de traslape es menor o igual al 20 % then
                    Cap_find(&data_es, &cam)
                end
            end
        end
    end
    if Si la foto fue tomada then
        Proyectar esquinas detectadas en la imagen capturada
        Activar Mutex para bloquear acceso al árbol
        Almacenar esquinas que no se encuentren en alguna zona de intersección con el
        área capturada en el árbol
        Desactivar Mutex para desbloquear acceso al árbol
    end
end
```

Con el Algoritmo (6) se cierra el presente capítulo, en el cual se ha desarrollado la explicación del algoritmo de detección de esquinas que se ha implementado en la BeagleBone Black y que demostró tener un mejor desempeño, así como también se ha descrito la manera en que este último se ha adaptado para la integración con el trabajo previo del sistema de captura inteligente de imágenes aéreas desarrollado por (Sánchez, 2013). La manera en que el algoritmo de detección de esquinas fue cambiando hasta llegar a la implementación que se trata en este capítulo, así como las consideraciones que se hacen para la integración del mismo al trabajo previo se abordarán a detalle en el próximo capítulo.

Capítulo 5

Experimentos

En este capítulo se describen los diferentes experimentos que se realizaron para evaluar varias implementaciones del algoritmo de detección de esquinas, así como la evaluación de diferentes estructuras de datos para el almacenamiento de la información de las esquinas detectadas que permitirán la integración del detector de esquinas al sistema de captura de fotografías autónomo.

5.1. Características del hardware

Todas las pruebas se realizaron en una Beaglebone Black, la cual cuenta con las siguientes características:

- Procesador AM335X 1GHz ARM Cortex-A8
- 512 MB DDR3L RAM a 800MHz
- Debian Wheezy
- SIMD NEON floating-point accelerator 16 registros de 128 bits
- Acelerador Gráfico SGX530 3D, 20M Polygons/S

En la siguiente sección se describen 9 implementaciones del algoritmo de Harris, las cuales contribuyeron significativamente al progreso del presente trabajo de tesis. La primera implementación consiste en capturar una imagen desde una webcam y buscar esquinas, para esto se utilizó la biblioteca OpenCV, en la segunda implementación se

utilizó las matrices integrales para realizar la sumatoria en ventanas. En la tercera implementación se utilizó una imagen de entrada establecida para verificar el funcionamiento del algoritmo y tener tiempos de referencia en que realiza cada paso la biblioteca OpenCV. La cuarta implementación fue realizar búsqueda de esquinas sin utilizar la biblioteca OpenCV, la quinta implementación fue realizar cada paso del algoritmo empleando la arquitectura NEON. En el sexto experimento se utilizan matrices traspuestas para el cálculo de las derivadas parciales en I_x y la sumatoria por filas, en la séptima implementación se utiliza la misma arquitectura NEON para trasponer las matrices. En la octava implementación se disminuye el número de transferencias a la arquitectura NEON, para ello se hace la unión de varios pasos del algoritmo y finalmente en la novena implementación se programa un algoritmo para ciertos valores específicos. A continuación describimos los detalles de cada una de ellas, y se muestran resultados en términos de tiempos de ejecución de cada una de las implementaciones, para los cuales se emplearon imágenes en tamaño VGA(640×480 píxeles).

5.2. Detector de esquinas de Harris

La primera implementación que se realizó del detector de esquinas fue de acuerdo al diagrama que se muestra en la Figura 4.1, para ello se empleó la biblioteca OpenCV[1], la cual se utilizó para controlar la webcam y obtener la imagen de entrada, la cual se convierte a escala de grises, siendo esta última la imagen de entrada para el algoritmo. Las derivadas parciales de la imagen de entrada se obtuvieron mediante la función filter2D proporcionada por OpenCV, la cual aplicó los filtros diferenciales Gaussianos, dando como resultados nuestras derivadas parciales. La sumatoria de valores en una ventana se realizó sumando todos los elementos que se encuentran dentro de la misma, una vez que se obtiene este resultado se aplica la formula de Harris; este proceso se aplica desplazando la ventana por toda la imagen de entrada. Al finalizar el proceso anterior se aplicó la función *normalize* de OpenCV para convertir los valores a escala de grises, donde los valores van de 0 a 255, esto con el fin de poder comparar a un valor umbral y determinar en que píxeles se encuentra una esquina.

[1] `http://opencv.org/`

En la segunda implementación se cambia la manera en que se realiza la sumatoria en una ventana, para ello se utiliza la imagen integral, con esto se aumenta el tiempo de calcular la misma, pero el cálculo de la sumatoria en una ventana es reducido a cuatro accesos a memoria y tres operaciones independiente al tamaño de la ventana.

En la siguiente tabla se pueden observar los tiempos obtenidos de la ejecución del algoritmo de detección de esquinas de Harris, empleando y no la matriz integral para diferentes tamaños de ventana y utilizando la biblioteca OpenCV.

w	**Tiempo sin usar matriz integral**	**Tiempo empleando matriz integral**
3	2373 ms	1260 ms
5	3012 ms	1351 ms
7	4221 ms	1334 ms
9	57534 ms	1322 ms
15	13515 ms	1324 ms

Cuadro 5.1: Tiempos de ejecución para diferentes tamaños de ventana(w) con y sin matriz integral.

De acuerdo a los resultados que se encuentran en el Cuadro (5.1) se demuestra que el uso de la matriz integral para el cálculo de la sumatoria en una ventana hace constante el tiempo en que se calcula, sin importar el tamaño de la misma, lo cual contribuye a la disminución del tiempo de ejecución del detector de esquinas.

Como paso siguiente se procedió a tomar los tiempos de cada una de las etapas o pasos que realiza OpenCV para encontrar esquinas en una imagen, estos resultados se presentan en el Cuadro (5.2) que se presenta a continuación.

Etapa	**Tiempo de Ejecución (ms)**
Cálculo de I_x empleando filter2D	79.76
Cálculo de I_y empleando filter2D	80.22
Cálculo de I_x^2, I_y^2 e I_xI_y	49.06
Cálculo de matriz integral I_x^2, I_y^2 e I_xI_y	203.27
Sumatoria en ventanas y evaluación de Harris	121.39
Normalizar empleando normalize	91.09
Discriminar esquinas(dato uchar)	12.72
Tiempo Total promedio	**637.5**

Cuadro 5.2: Tiempos de ejecución para cada etapa del detector de esquinas usando OpenCV.

El experimento anterior se realizó con el objetivo de obtener los tiempos en que la biblioteca OpenCV realizaba cada una de las etapas para encontrar esquinas, con esto se obtuvieron tiempos de referencia para poder comparar los intentos de mejora y aceleración del algoritmo. Como paso posterior se decidió implementar el algoritmo de detección de esquinas utilizando lo menos posible la biblioteca OpenCV, con lo que solamente se utilizó para cargar una imagen de entrada preestablecida, para normalizar los resultados y para guardar en imágenes los resultados de aplicar cada una de las etapas, esto para ir comparando los resultados de cada etapa al momento de implementar cada una de las etapas en la arquitectura NEON SIMD.

En las siguientes imágenes se muestran todos los resultados del algoritmo de detección de esquinas de Harris para una desviación estándar de 0.6, factor de Harris de 0.08, tamaño de ventana de 5 y un umbral de 150.

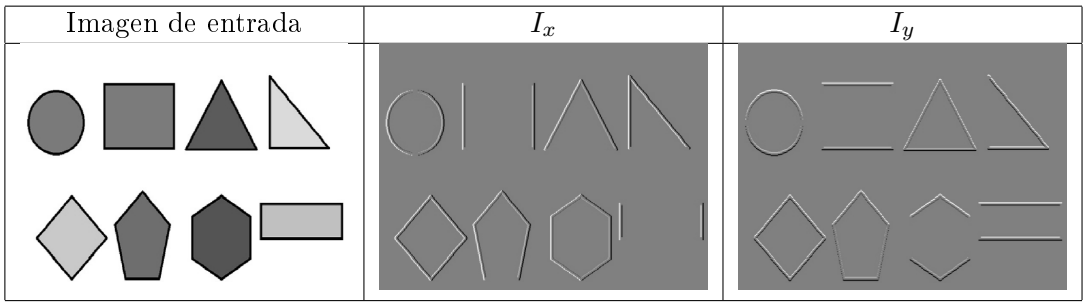

Figura 5.1: Imagen de entrada a escala de grises y derivadas parciales.

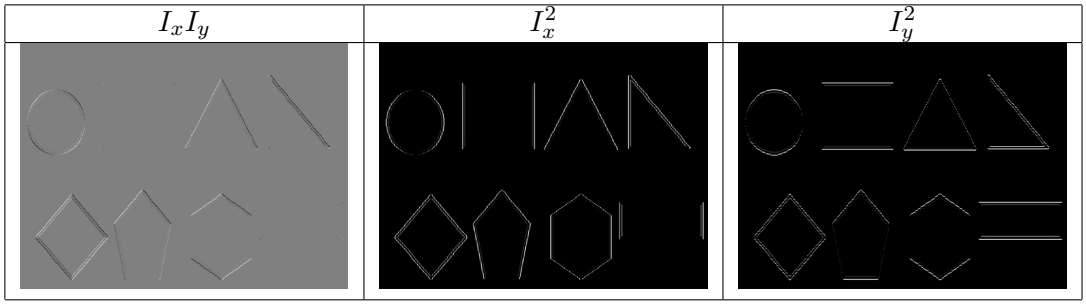

Figura 5.2: Productos entre las derivadas parciales y cuadrados de cada una.

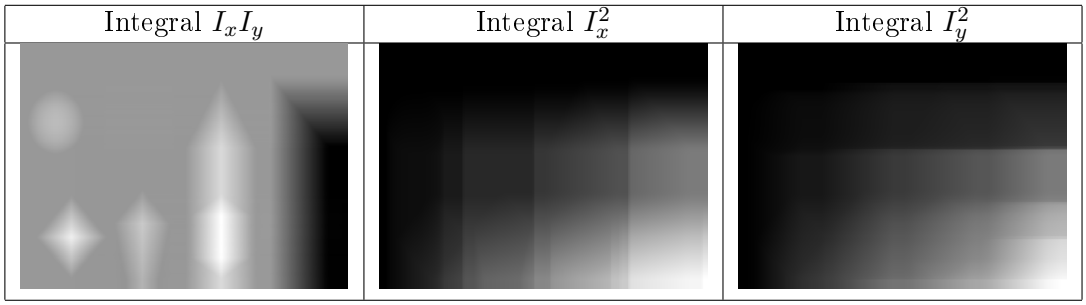

Figura 5.3: Matrices Integrales de I_xI_y, I_x^2, I_y^2.

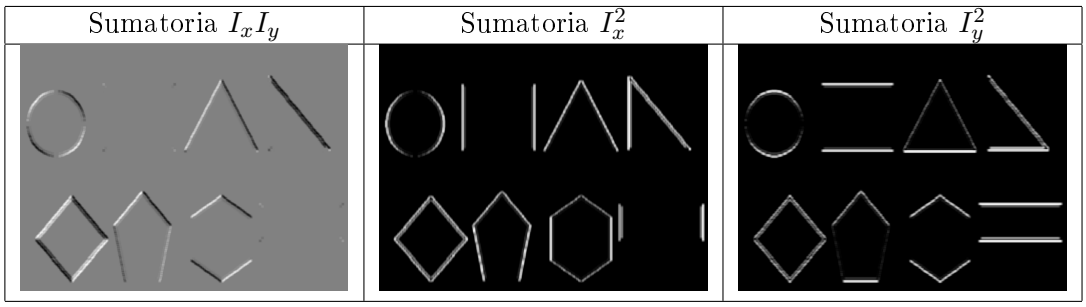

Figura 5.4: Sumatorias en ventanas de matrices integrales de I_xI_y, I_x^2, I_y^2.

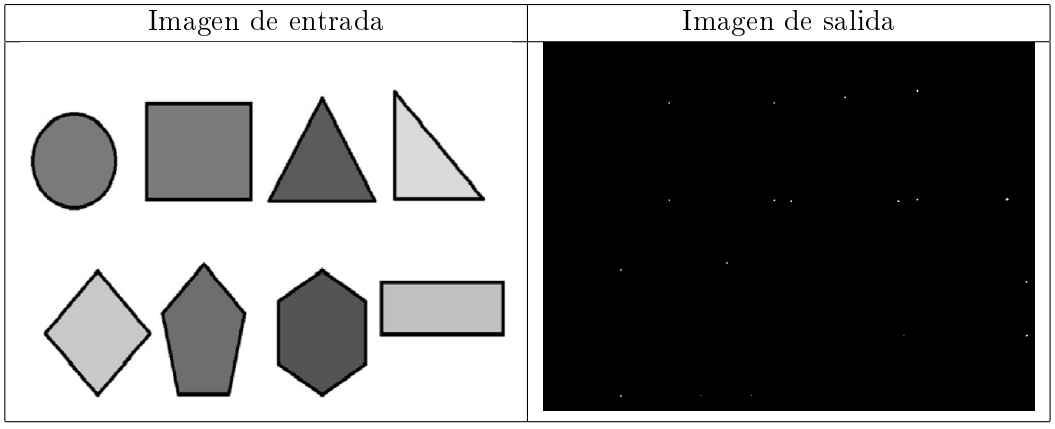

Figura 5.5: Imagen de entrada y de salida.

En el siguiente cuadro se muestran los tiempos en realizar cada etapa en la CPU y en la arquitectura NEON.

Etapa	**CPU (ms)**	**NEON(ms)**
Convertir Mat a arreglo	6.18	6.15
Cálculo de I_x	106.12	32.06
Cálculo de I_y	114.2	10.62
Cálculo de $I_x I_y$	9.51	29.78
Cálculo de I_x^2	36.22	
Cálculo de I_y^2	38.02	
Sumatoria de filas I_x^2	42.63	44.52
Sumatoria de columnas I_x^2		6.22
Sumatoria de filas I_y^2	42.79	44.69
Sumatoria de columnas I_y^2		6.25
Sumatoria de filas $I_x I_y$	42.69	44.57
Sumatoria de columnas $I_x I_y$		6.39
Sumatoria de ventanas de matriz integral I_x^2	8.8	22.68
Sumatoria de ventanas de matriz integral I_y^2	9.02	
Sumatoria de ventanas de matriz integral $I_x I_y$	8.98	
Aplicar formula de Harris	73.55	11.59
Normalizar empleando normalize de OpenCV	84.52	14.39
Discriminar esquinas	34.3	35.02
Tiempo Total Promedio	**651.41**	**314.99**

Cuadro 5.3: Tiempos de ejecución para cada etapa del detector de esquinas usando cpu y arquitectura NEON.

De acuerdo a los resultados mostrados en el Cuadro (5.3) se demuestra que es posible acelerar el algoritmo de detección de esquinas empleando la arquitectura NEON; a partir de este punto se buscó implementar un algoritmo que funcionara para varios tamaños de filtro y de ventana. El cálculo de la derivada parcial I_y se logró implementar para cualquier tamaño de filtro, con la restricción de que el número de columnas de la imagen sea múltiplo de cuatro, esto es porque con la arquitectura NEON se procesan los datos de cuatro en cuatro, mientras que para el cálculo de la derivada parcial en I_x esto no fue posible, esto porque la arquitectura NEON transfiere a sus registros datos contiguos a partir de la dirección indicada, dado que para calcular I_x se hace referencia a elementos contiguos, no se logró establecer una forma general para realizar la transferencia de los valores a la arquitectura NEON, por tal motivo se implementó el cálculo de la derivada parcial I_x para un filtro de cinco coeficientes. El cálculo de la sumatoria en ventanas también se logró implementar para varios tamaños de la misma, con la restricción de que el número de columnas menos el tamaño de la ventana más uno, sea múltiplo de cuatro, esto es por que la arquitectura NEON procesa los datos de cuatro en cuatro.

Después de ejecutar el experimento y notar que el tiempo en que se calcula I_y con respecto a I_x es mucho menor, adicionado a que el cálculo para I_x estaba restringido para filtros con cinco coeficientes, surge la idea de calcular I_x de la misma manera en que se calcula I_y, para ello sería necesario trasponer la imagen, aplicarle el filtro en y, posteriormente volver a trasponer, con lo cual ya se tendría la manera de calcular I_x para filtros de diferentes tamaños, para este caso la restricción es que el número de filas sea múltiplo de cuatro. Adicional a lo anterior se notó que el tiempo de calcular la sumatoria de columnas con respecto a la sumatoria de filas para el cálculo de la matriz integral es mucho menor, análogo al caso de la derivada parcial, se aplicó la idea de trasponer las matrices, con lo que el proceso para calcular la matriz integral sería, trasponer la matriz, aplicar sumatoria de columnas, trasponer el resultado, con lo que se tendría la sumatoria por filas, por último, aplicar la sumatoria por columnas para concluir con el cálculo de la matriz integral, estos resultados se muestra en el Cuadro (5.4) que se muestra a continuación.

Etapa	**Tiempo (ms)**
Convertir Mat a arreglo	6.15
Trasponer imagen	27.81
Aplicar derivada I_y a imagen traspuesta	10.31
Trasponer resultado para obtener I_x	15.22
Calcular derivada I_y de la imagen	10.3
Calcular I_x^2, I_y^2 e I_xI_y	29.87
Trasponer I_x^2	30.78
Aplicar sumatoria por columnas a I_x^2 traspuesta	4.67
Trasponer resultado anterior	21.02
Sumatoria por columnas para obtener matriz integral I_x^2	8.64
Trasponer I_y^2	28.29
Aplicar sumatoria por columnas a I_y^2 traspuesta	4.6
Trasponer resultado anterior	20.92
Sumatoria por columnas para obtener matriz integral I_y^2	6.15
Trasponer I_xI_y	30.85
Aplicar sumatoria por columnas a I_xI_y traspuesta	4.68
Trasponer resultado anterior	20.97
Sumatoria por columnas para obtener matriz integral I_xI_y	6.13
Sumatoria en ventanas I_x^2, I_y^2, I_xI_y	21.78
Evaluación de formula de Harris	11.82
Normalizar	17.14
Discriminar esquinas	41.53
Tiempo Total promedio	**379.72**

Cuadro 5.4: Tiempos de ejecución detector de esquinas utilizando matrices traspuestas.

El experimento anterior demostró que la idea de emplear matrices traspuestas funciona, sin embargo, el tiempo de ejecución aumento en comparación al caso anterior, lo cual nos llevó a rediseñar el algoritmo para reducir el tiempo; se propuso implementar una función que aplicara la derivada I_y y que los resultados se regresaran de manera traspuesta desde la arquitectura NEON, para el caso de las matrices integrales se implementó una función que realizara sumatoria de filas cargando la imagen de manera traspuesta, operando y regresando de la misma manera. Los tiempos de esta implementación se muestra en el Cuadro (5.5).

Etapa	Tiempo en ms
Convertir Mat a arreglo	6.46
Cálculo de I_x con imagen traspuesta	51.56
Cálculo de I_y	9.88
Cálculo I_x^2, I_y^2, I_xI_y	30.43
Cálculo sumatoria de filas I_x^2	10.11
Cálculo sumatoria de columnas I_x^2	6.56
Cálculo sumatoria de filas I_y^2	6.44
Cálculo sumatoria de columnas I_y^2	9.98
Cálculo sumatoria de filas I_xI_y	10.03
Cálculo sumatoria de columnas I_xI_y	6.42
Sumatoria en ventanas I_x^2, I_y^2, I_xI_y	26.11
Evaluación de formula de Harris	12.59
Normalizar	13.96
Discriminar esquinas	32.13
Tiempo Total promedio	**232.71**

Cuadro 5.5: Tiempos de ejecución detector de esquinas utilizando matrices traspuestas desde arquitectura NEON.

De acuerdo a los resultados obtenidos en el Cuadro (5.5) se demostró que utilizar la arquitectura NEON para procesar los resultados traspuestos funcionó y se logró una disminución en el tiempo de ejecución; como siguiente paso se buscó disminuir el número de trasferencia de información entre la memoria RAM y la arquitectura NEON, para ello se buscó unir varios procesos dentro de la arquitectura NEON. En este punto se cambia la forma de calcular la matriz integral; para ello, primero calculamos sumatoria en columnas y posteriormente sumatoria por filas, este cambio no afecta el resultado de la matriz integral. Esta implementación es la que se describe en el capitulo anterior y se muestra en el diagrama de la Figura (4.17); en el Cuadro (5.6) se presentan los resultados obtenidos de esta implementación.

Etapa	**Tiempo en ms**
Convertir Mat a arreglo	6.13
Cálculo de I_x con imagen traspuesta	31.14
Cálculo de I_x^2, I_y^2, I_xI_y y sumatoria de columnas de I_x^2, I_y^2, I_xI_y	20.24
Sumatoria de filas I_x^2, I_y^2, I_xI_y	15.12
Sumatoria de ventanas en I_x^2, I_y^2, I_xI_y y evaluación de formula de Harris	13.73
Normalizar	10.93
Discriminar esquinas	14.78
Tiempo Total promedio	**112.15**

Cuadro 5.6: Tiempos de ejecución detector de esquinas utilizando matrices traspuestas desde arquitectura NEON y reduciendo el número de trasferencias.

Esta última implementación funciona para filtros de tamaño diferente y varios tamaños de ventana, siempre y cuando se cumplan las siguientes restricciones:

- El número de filas de la imagen es múltiplo de cuatro
- El número de columnas de la imagen es múltiplo de cuatro
- El número de columnas de la imagen menos el tamaño de la ventana más uno, es múltiplo de cuatro

Como último paso se decidió implementar un caso especifico para un filtro de cinco coeficientes y una ventana de tamaño tres; para la implementación de este caso se tomaron en cuenta todas las consideraciones que se mencionan en el capitulo anterior, los tiempos obtenidos se presentan a continuación en el Cuadro (5.7).

Etapa	Tiempo en ms
Convertir Mat a arreglo	6.07
Cálculo de I_x con imagen traspuesta	18.15
Cálculo de I_y, I_xI_y, I_x^2, I_y^2 y sumatoria de columnas de I_x^2, I_y^2 e I_xI_y	25.23
Sumatoria de filas I_x^2, I_y^2 e I_xI_y	15.39
Sumatoria de ventanas en I_x^2, I_y^2, I_xI_y y evaluación de formula de Harris	14.03
Normalizar	10.88
Discriminar esquinas	14.72
Tiempo Total promedio	**104.53**

Cuadro 5.7: Tiempos de ejecución detector de esquinas para un tamaño de filtro de 5 y una ventana de tamaño 3.

En los resultados anteriores se puede notar una pequeña disminución en el tiempo de ejecución; a este podemos restarle el tiempo que se emplea para convertir el Mat a un arreglo con el tipo de dato que emplea la arquitectura NEON, esto porque al cambiar la captura de la imagen con OpenCV a V4L(Video for Linux) se implementa que la captura sea almacenada con el tipo de dato que necesita la arquitectura; también se le puede quitar el tiempo de discriminar esquinas, ya que para la unión con el proyecto anterior, es necesario recorrer nuevamente el arreglo para la proyección de las esquinas al plano terrestre y posteriormente agregar al árbol que almacenará las mismas, por tanto, podemos incluir la discriminación de las esquinas en este proceso, con lo cual el tiempo para encontrar esquinas es aproximadamente de 84 ms.

5.3. Captura de imágenes con V4L

En esta sección se presentan los tiempos de ejecución de dos implementaciones para capturar una imagen desde una webcam Logitech C920, la primera implementación emplea la biblioteca OpenCV para inicializar la cámara, capturar una imagen y guardarla en un archivo. La segunda implementación emplea la biblioteca V4L para inicializar la cámara y capturar la imagen, mientras que para guardar la imagen se emplea la biblioteca de entrada y salida estándar de C (stdio.h).

En el siguiente Cuadro (5.8) se presentan los tiempos de estas comparaciones.

Proceso	OpenCV (ms)	V4L (ms)
Inicializar cámara	37.12	26.65
Capturar imagen	779.1	425.67
Guardar imagen	350.17	191.05
Tiempo Total promedio	**1166.39**	**643.28**

Cuadro 5.8: Tiempos de capturar una imagen y guardarla utilizando OpenCV y V4L.

De acuerdo a los resultados que se muestran en el cuadro anterior se decidió emplear la biblioteca V4L para la captura de imágenes ya que presenta un menor tiempo, adicionalmente nos permite que nuestro detector de esquinas no dependa de una biblioteca no nativa del sistema operativo Linux, lo cual facilita la portabilidad del detector de esquinas.

Seguidamente se comenzó con la búsqueda de una estructura que almacenara puntos (x, y) y que nos permitiera saber cuantos puntos existen en una determinada área. En la siguiente sección se presentan cuatro implementaciones para almacenamiento y búsqueda de puntos (x, y). La primera implementación utiliza la biblioteca ANN(Approximate Nearest Neighbor)[2]. La segunda implementación es una idea propia, la cual emplea la partición del espacio de un kdtree, modificando la manera de insertar elementos, esto para no tener que ordenar y buscar la mediana cada vez que agregáramos nuevos puntos, la tercera se basa en un árbol binario no balanceado y la cuarta implementación utiliza la biblioteca Boost Geometry[3].

5.4. Estructura de almacenamiento de coordenadas

Para la integración del detector de esquinas con el trabajo previo, es necesario que las coordenadas del plano terrestre donde se hayan detectado puntos de interés sean almacenados, esto para que el sistema de captura inteligente de fotografías pueda consultar el número de esquinas que existe en el área de traslape entre las imágenes que van a constituir el mosaico, si existe un buen número de esquinas en este área se procederá a capturar la fotografía con la cámara de alta resolución. Para almacenar y realizar esta

[2] `http://www.cs.umd.edu/~mount/ANN/`

[3] `http://www.boost.org/doc/libs/1_57_0/libs/geometry/doc/html/index.html`

búsqueda se decidió emplear arboles, a continuación se describen 4 implementaciones para este propósito.

El primer tipo de árbol que se decidió implementar fue un árbol *kdtree*, para lo cual empleamos la bibiloteca ANN. El objetivo del experimento fue agregar diez mil puntos al árbol, donde los valores de la coordenada en x son $0 < x < 100$, y para cada valor de x, los valores de la coordenada en y son $0 < y < 100$. Una vez hecho lo anterior se procedió a tratar de encontrar un determinado número de puntos en un área determinada. Los tiempos de ejecución de este experimento se muestran en el siguiente cuadro.

Proceso	**Resultados**
Crear kdtree con 10000 puntos	174.66 ms
Realizar búsqueda de puntos en un área circular	101.07
Esquinas detectadas	2500
Tiempo Total promedio	**275.74**

Cuadro 5.9: Tiempos para crear un arbol kdtree con 10000 puntos y buscar 2500 vecinos más cercanos que se encuentren en un circulo con centro (50, 50) y un radio de 25.

Se decidió que estos tiempos no eran lo suficientemente buenos, adicionalmente cada vez que se agregaran nuevos puntos al árbol seria necesario obtener los puntos existentes y almacenarlos en un arreglo, agregar los nuevos puntos al mismo arreglo y construir nuevamente el *kdtree*. Mas aún la búsqueda que nos permite realizar la biblioteca ANN es solo en área circulares, lo cual no resultó muy útil ya que los polígonos de intersección entre imágenes serán de forma irregular o en el mejor de los casos de forma rectangular.

Debido a las razones anteriores se realizó una implementación propia de un árbol, el cual empleó la forma de particionar el espacio de un *kdtree* y se modificó la forma de crear y agregar nodos. Para esto no se tomó en cuenta la mediana de los datos y se tomó como raíz del árbol el primer nodo que se agregara, el resto de los nodos se agregó de la misma manera que en un árbol binario. Esta idea nos permitió ir agregando nodos como se fueran generando y no tener que estar creando un nuevo árbol cada vez que quisiéramos agregar más nodos. Adicionalmente esta forma de agregar los nodos nos permitió buscar esquinas en un área rectangular, sin embargo, no mostró una mejora en el tiempo comparado con la biblioteca ANN.

La tercera implementación también fue una idea propia, la cual se basa en una matriz

dispersa. Se tomó la coordenada x como número de fila o nivel de profundidad en el árbol, la coordenada y del nodo se tomó como el número de columna, con esto, dos nodos que tuvieran la misma coordenada en x,estarían conectados, quedando a la izquierda el que tuviera menor coordenada en y. En el caso de dos nodos con diferentes coordenadas en x, estarían conectados, quedando más arriba el que tuviera un menor nivel de profundidad o coordenada en x.

La cuarta implementación utilizó la biblioteca Boost Geometry, de la cual se empleó un *R-tree* con un algoritmo cuadrático; el cual, de acuerdo a su documentación maneja un balance en el tiempo de agregar y realizar búsqueda de puntos.[4] Adicionalmente, esta biblioteca nos permite realizar búsqueda de puntos en cualquier polígono, regular o irregular, lo cual nos da una mayor flexibilidad y se adaptó mejor para la unión del proyecto.

El experimento de estas tres implementaciones consistió en agregar los mismos 10000 puntos de la primera implementación y la búsqueda se realizo en un polígono de forma rectangular. Los resultados de estas tres implementaciones se presentan en el siguiente cuadro.

Proceso	**kdtree Modificado**	**Matriz Dispersa**	**Boost Geometry**
Crear árbol con 10000 puntos	266.87 ms	219.59 ms	4.25 ms
Realizar búsqueda en 25<= x y <=75	4.9 ms	1.21 ms	3.99 ms
Esquinas encontradas	2601	2601	2601
Tiempo promedio	**271.78 ms**	**220.8 ms**	**8.25 ms**

Cuadro 5.10: Tiempos de capturar una imagen y guardarla utilizando OpenCV y V4L.

De acuerdo a los resultados anteriores se decidió emplear Boost Geometry por mejor tiempo, adicionalmente nos permite ir agregando esquinas conforme se vayan generando y nos permite una perfecta integración con el trabajo previo, el cual evaluaba el área de intersección entre los polígonos, ahora evaluará el número de esquinas que existen en el polígono de intersección, el cual puede tener forma irregular.

[4] `http://www.boost.org/doc/libs/1_55_0/libs/geometry/doc/html/geometry/spatial_indexes/introduction.html`

Capítulo 6

Conclusión

De acuerdo a los resultados obtenidos, se demostró que con el uso de la arquitectura NEON es posible acelerar el tiempo para encontrar esquinas en una imagen. Este algoritmo funcionará específicamente para los procesadores Cortex-ARM y podrá ser utilizado para cualquier proceso que como paso previo requiera de un detector de esquinas, por ejemplo: navegación de robots, reconocimiento de objetos o como lo es para nuestro caso, la creación de un mosaico de imágenes.

Contribución del proyecto

La principal contribución del presente proyecto radica en garantizar al sistema de captura inteligente que se lleve a cabo un buen registro de las imágenes que se capturarán con la cámara de alta resolución; con este trabajo se tomará en cuenta la información de las imágenes capturadas y el área de traslape entre las misma, lo cual asegurará que al momento de generar el mosaico de imágenes haya suficientes correspondencias entre las áreas de traslape y permitan automatizar el proceso de la construcción del mosaico con las imágenes capturadas.

Los aportes del presente proyecto se pueden resumir en los siguientes puntos:

- 1- Un detector de esquinas rápido que funciona para procesadores Cortex-ARM.
- 2- Complementa al sistema de captura inteligente de fotografía aérea.
- 3- Garantiza que el área de traslape entre imágenes cuenten con suficientes co-

rrespondencias para la generación automática del mosaico de imágenes.

Trabajo futuro

Realizar los experimentos de campo para la verificación del funcionamiento del mismo como parte del sistema de captura inteligente de imágenes aéreas. Los experimentos de campo consisten en elevar físicamente el papalote con todo el sistema de captura montado en él. Los dispositivos reales, GPS, IMU, Xbee y Cámaras, deben estar bien calibrados y funcionando. Estos experimentos aún están pendientes de realizar debido a la naturaleza de estos experimentos (que se deben realizar al aire libre, en un terreno extenso); es necesario que las condiciones meteorológicas sean adecuadas para un buen desarrollo de éstos.

Como toda metodología desarrollada, la presentada en este proyecto es susceptible de cambios y mejoras, y puede estar en constante evolución para ir optimizando o mejorando el desempeño del sistema. La linea de continuación del presente proyecto radica en hacer más rápido el proceso de detección de esquinas y desarrollar una biblioteca que permita el uso de diferentes cámaras y se adapte a cualquier tamaño de imagen. La biblioteca deberá permitirá que cualquiera pueda emplear nuestra implementación en cualquier proceso que requiera de un detector de esquinas y que sea pensado para cómputo móvil y que utilice una arquitectura ARM.

Bibliografía

Aber, J. S. (2008). History of Kite Aerial Photography. Unpublished. `http://www.geospectra.net/kite/history/history.htm`.

Aber, J. S. & Aber, S. W. (2003). Applications of Kite Aerial Photography: Property Survey. *Transactions of the Kansas Academy of Science*, *106*(1/2), 107–110.

Alemán, E. S. (2013). Levantamiento fotogramétrico y modelización tridimensional del santuario de nuestra señora de la Fuensanta. Master's thesis, Universidad Politécnica de Cartagena.

Anderson, R. C. (2001). Kite Aerial Photography for Archaeology: An Assessment and Short Guide. *British School at Athens Studies*, *8*, 167–180.

Capel, D. P. (2001). *Image Mosaicing and Super-resolution*. PhD thesis, University of Oxford.

Chen, J., hui Zou, L., Zhang, J., & Dou, L. (2009). The Comparison and Application of Corner Detection Algorithms. *Journal of Multimedia*, *4*(6), 435–441.

Debra Eberts, J. A. S. A. (2005). Applications of kite aerial photography: Biocontrol of salt cedar in the western united states. *Transactions of the Kansas Academy of Science*, *108*(1/2), 63–66.

Forsyth, D. & Ponce, J. (2012). *Computer vision, a modern approach* (Second ed.). Prentice Hall. Pearson Education.

Franklin, C. (1984). Summed-area tables for texture mapping. *Computer Graphics*, *18*, 207–212.

Gumustekin, S. (1999). An introduction to image mosaicing. Unpublished. `http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/An_Introduction_to_Image_Mosaicing.htm`.

Harris, C. & Stephens, M. (1988). A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, *4*, 147–152.

Hepp, C., Delgado, O., & Junquera, P. (2012). Información general sobre la fotografía aérea. `https://sites.google.com/site/apderespanolcastellano/resource-centre/history-of-aerial-imaging`.

Hwang, K., Briggs, F. A., Paloma, A., & Jaime, R. (1987). *Arquitectura de computadoras y procesamiento paralelo*. McGraw-Hill.

Loewke KE, C. D. J. C. S. J., Camarillo, et al. (2011). In Vivo Micro-Image Mosaicing. *IEEE Transactions on Biomedical Engineering*, *58*(1), 159–171.

López, M. L. (2013). Estimación robusta de pose de un vehículo aéreo no tripulado . Master's thesis, Facultad de Matemáticas, Universidad Autónoma de Yucatán.

Ma, Y., Soatto, S., Kosecka, J., & Sastry, S. S. (2004). *An Invitation to 3-D vission*. Springer.

Mouchague, P. & Mouchague, P. (2010). Kite aerial photography. Unpublished. `http://www.wokipi.com/decouverte/aerialpicture.html`.

M.Yakar, Yildiz, F., Metin, A., et al. (2010). Photogrammetric measurement of the meke lake and its enviroment with kite photographs to monitoring of water level to climate change. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *38*(5), 613–616.

Nixon, M. S. & Aguado, A. S. (2012). *Feature Extraction & Image Processing for Computer Vision* (third ed.). Academic Press.

Oh, P. Y. & Green, W. E. (2004). Mechatronic Kite and Camera Rig to Rapidly Acquire, Process, and Distribute Aerial Images . *IEEE/ASME transactions on mechatronics*, *9*(4), 671–678.

Parks, D. & Gravel, J.-P. (2004). Corner detection. `http://www.cim.mcgill.ca/dparks/CornerDetector/harris.html`.

Paul Viola, M. J. J. (2001). Robust Real-time Object Detection. Technical report, Cambridge Research Laboratory.

Pradip, M., Qiong, Y., Lafruit, G., et al. (2010). LOCOCO: Low Complexity Corner Detector. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, (pp. 810–813).

Sánchez, M. I. (2013). Automatización de un sistema de captura de fotografía aérea para la generación de mosaicos de imágenes . Master's thesis, Facultad de Matemáticas, Universidad Autónoma de Yucatán.

Sonka, M. & Boyle, R. (2008). *Image processing, analysis, and machine vision* (third ed.). Cengage Learning.

Teixeira, L., Celes, W., & Gattass, M. (2008). Acelerated corner-detector Algorithms. In *British Machine Vision Conference*. `http://www.bmva.org/bmvc/2008/papers/45.pdf`.

Apéndice A

Biblioteca C Intrínseca NEON

La biblioteca intrínseca de NEON se incluye en las herramientas de compilación de los procesadores ARM. Esta biblioteca nos permite usar el procesador SIMD que se incluye en los procesadores Cortex como simples llamadas a funciones. La biblioteca se encuentra desarrollada en lenguaje C y para su uso es necesario incluir la siguiente cabecera:

```
#include <arm_neon.h>
```

e incluir las siguientes banderas en la instrucción de compilación:

```
-march=armv7-a -mtune=cortex-a8 -mfloat-abi=hard -mfpu=neon -ffast-math
```

En este apéndice se describen las funciones y los tipos de datos que se emplearon para acelerar el detector de esquinas de Harris mediante la arquitectura NEON.

A.1. Vectores en arquitectura NEON

La arquitectura NEON permite acelerar un código realizando una misma operación a todos los elementos de un vector, de manera general los vectores son nombrados de la siguiente manera:

```
<tipo><numero de Bits>X<numero de elementos>_t
```

por ejemplo:

```
float32x4_t vector;
```

es un vector que contiene cuatro elementos, cada uno de los cuales es un flotante de 32 bits, este tipo de vector es el que se empleó en su mayoría en este trabajo.

A.2. Operación de vectores en arquitectura NEON

La arquitectura NEON nos permite aplicar una misma operación a todos los datos de un vector al mismo tiempo, de manera general las instrucciones tendrán la siguiente forma:

```
<nombre de operacion>[q]_<tipo de dato>
```

donde [q] es opcional, si aparece indica que la operación se aplicará a un vector de 128 bits, si no aparece, la operación se aplicará a un vector de 64 bits. En este apéndice se hará referencia a todas la operaciones utilizadas en el detector de esquinas.

A.2.1. Carga de vectores desde un puntero

Antes de poder comenzar a utilizar el procesador NEON, es necesario transferir datos de la memoria RAM a el procesador vectorial, para ello se provee una función, la cual tiene la siguiente forma general:

```
vld1{q}_<tipo de dato>(memoria a partir de la cual se desea copiar datos)
```

Ejemplo:

```
float32x4_t vec1;
float32_t * puntero;
vec1 = vld1q_f32(puntero);
```

el ejemplo anterior transfiere 4 datos del tipo float32_t a la arquitectura NEON y los almacena en un vector.

A.2.2. Carga de vectores con constante

Entre las operaciones de carga existe una función que inicializa o llena todos los valores de un vector con el valor indicado, la cual tiene la siguiente forma general:

```
vdup{q}_n_<tipo de dato>(valor de inicializacion)
```

Ejemplo:

```
float32_t valor;
float32x4_t vec1;
vec1 = vdupq_n_f32(valor);
```

El ejemplo anterior inicializa un vector de 4 elementos con el valor float32_t indicado.

A.2.3. Sumatoria de vectores

La forma general para efectuar una suma de vectores es la siguiente:

```
vadd{q}_<tipo de dato>. Vr[i]:= Va[i] + Vb[i]
```

Ejemplo:

```
float32x4_t vec1, vec2, vecR;
vecR = vaddq_f32(vec1, vec2);
```

en el ejemplo anterior se suman los elementos de 2 vectores de 4 elementos, donde cada elemento es un vector de 32 bits.

A.2.4. Resta de vectores

La forma general para efectuar una resta de vectores es la siguiente:

```
vsub{q}_<tipo de dato>. Vr[i]:= Va[i] - Vb[i]
```

Ejemplo:

```
vecR = vsubq_f32(vec1, vec2);
```

en el ejemplo anterior se le restan los valores de vec2 a vec1, donde cada vector contiene 4 elementos de tipo flotante de 32 bits.

A.2.5. Multiplicación de vector por un escalar

La forma general para efectuar una resta de vectores es la siguiente:

```
vmul{q}_n_<tipo de dato>. Vr[i]:= Va[i] * escalar
```

Ejemplo:

```
float32_t valor;
vecR = vmulq_n_f32(vec1, valor);
```

A.2.6. Máximo de un vector

Otra operación que se nos ofrece es comparar los elementos de 2 vectores, regresando como resultado un vector con los valores máximos entre cada comparación. Esta operación puede expresarse de manera general como:

```
vmax{q}_<tipo de dato>. Vr[i]:= (Va[i] >= Vb[i] ? Va[i] : Vb[i])
```

Ejemplo:

```
Va = [5 8 3 6];
Vb = [9 7 2 4];
Vr = vmaxq_f32(Va, Vb);
Vr = [9 8 3 6];
```

A.2.7. Mínimo de un vector

Otra operación que se nos ofrece es comparar los elementos de 2 vectores, regresando como resultado un vector con los valores mínimos entre cada comparación. Esta operación puede expresarse de manera general como:

```
vmax{q}_<tipo de dato>. Vr[i]:= (Va[i] >= Vb[i] ? Vb[i] : Va[i])
```

Ejemplo:

```
Va = [5 8 3 6];
Vb = [9 7 2 4];
Vr = vmin_f32(Va, Vb);
Vr = [5 5 2 4];
```

A.2.8. Almacenamiento de un elemento de vectores en memoria

Al concluir de operar u obtener el resultado deseado, es necesario regresar esta información a la memoria RAM, para ello se provee una función que regresa un solo valor, la cual tiene la siguiente forma general:

```
vst1q_lane_<tipo de dato>(puntero, vector, posicion de vector(0-3));
```

Ejemplo:

```
float32_t * puntero;
float32x4_t Vr;
vst1q_lane_f32(puntero, Vr, 0);
```

en el ejemplo anterior se transfiere el primer elemento del vector indicado a la dirección de memoria proporcionada.

A.2.9. Almacenamiento de vectores en memoria

Otra función que nos ofrece para regresar elementos a la memoria RAM se presenta a continuación, la cual regresa todos los elementos de un vector a partir de la dirección de memoria indicada, teniendo la siguiente forma general:

```
vst1q_<tipo de dato>(puntero, vector);
```

Ejemplo:

```
float32_t *puntero;
float32x4_t Vr;
vst1q_f32(puntero, Vr);
```

en el ejemplo anterior se transfieren los cuatro elementos contenidos en el vector Vr a la memoria RAM a partir de la dirección indicada con el apuntador.

Para más información sobre el uso de estos conceptos y funciones se recomienda acudir a la guía de usuario del compilador Armcc.

Apéndice B

Biblioteca C++ BOOST

Boost es un conjunto de bibliotecas escritas en C++ que provee soporte para tareas y estructuras para diversas áreas como por ejemplo álgebra lineal, procesamiento de imágenes, geometría, etc. Estas bibliotecas están categorizadas de acuerdo al área para la cual están programadas. En el presente proyecto se usa la biblioteca de Geometría, la cual provee, entre otras funcionalidades funciones para el manejo de polígonos y estructuras para almacenar los mismos.

B.1. Biblioteca de Geometria (*Geometry*)

Boost Geometry forma parte de la colección de las bibliotecas de Boost C++, define conceptos, primitivas y algoritmos para la resolución de problemas geométricos. Algunos de los algoritmos que se pueden encontrar en esta biblioteca son para calcular área, perímetro, centroide, unión, diferencia, e intersección de polígonos, trasformaciones geométricas y estructuras que nos permiten almacenar puntos, polígonos y realizar búsquedas, etc. El código de esta biblioteca está diseñado para ser lo más genérico posible, así, se puede usar en cualquier aplicación en donde la geometría este involucrada, como por ejemplo, el desarrollo de juegos, gráficas por computadora, robótica, astronomía, etc.

Para utilizar la biblioteca sólo se requiere incluirla en la cabecera:

```
#include <boost/geometry.hpp>
```

La mayoría de las bibliotecas Boost son sólo de encabezado o de cabecera: consisten enteramente de archivos de encabezado que contienen plantillas y funciones ya hechas y no requieren ninguna biblioteca compilada por separado o tratamiento especial cuando se vincula (linking). El uso de esta biblioteca se debe a que provee algunas bondades que resultan ser muy útiles para el desarrollo de la metodología del presente proyecto, ahora se verán algunos conceptos y funciones que serán de utilidad para manejar las definiciones vistas en le sección 3.8.

B.2. Tipos de datos ofrecidos por Boost Geometry

B.2.1. Puntos 2D

La base de la geometría utilizada son los puntos en 2D. Boost provee varios tipos de datos para puntos, en el presente trabajo se manejan puntos cartesianos en 2 dimensiones, para lo cual sólo hay que incluir:

```
#include <boost/geometry/geometries/point.hpp>
```

para definir un punto de este tipo se realiza de la siguiente manera:

```
boost::geometry::model::point<
double, 2, boost::geometry::cs::cartesian> punto;
```

para acceder a los componentes x, y del punto se realiza de la siguiente manera:

```
punto.get<0>; //Se accede a la coordenada en x del punto
punto.get<1>; //Se accede a la coordenada en y del punto
```

y para establecer los componentes x, y de un punto se realiza de la siguiente manera:

```
punto.set<0> = a; //Se asigna a como el valor de la coordena en x
punto.set<1> = b; //Se asigna b como el valor de la coordena en y
```

B.2.2. Anillos

BOOST define el conceto de anillo tal y como se definió en la sección 3.8, para poder utilizar esta estructura, solo hay que incluir la cabecera:

```
#include <boost/geometry/geometries/ring.hpp>
```

que contiene funciones para el manejo de anillos exteriores e interiores de un polígono.

B.2.3. Polígonos

Para hacer uso de los polígonos tal y como se definió en el marco teórico, Boost provee el tipo de dato *polygon* que para utilizarlo es necesario agregar:

```
#include <boost/geometry/geometries/polygon.hpp>
```

una vez incluida esta cabecera nos permite el manejo de polígonos con huecos(anillos interiores), tal y como se definió en el marco teórico.

B.3. Operaciones con polígonos de Boost Geometry

La biblioteca Boost geometry ofrece varias funciones que nos ayudan al manejo de los tipos de datos que se mencionan en la sección anterior. A continuación se presentan las funciones que se utilizaron en el presente trabajo.

B.3.1. Corrección de polígonos

La biblioteca Boost ofrece una función que permite corregir el orden de los puntos que forman un polígono, esto se debe a que para poder operar los polígonos, los puntos deben de tener un cierto orden. Para hacer uso de esta función debe de incluirse la cabecera:

```
#include <boost/geometry/algorithms/correct.hpp>
```

una vez incluida simplemente se invoca la función correct(), que recibe como parámetro un polígono. Ejemplo:

```
correct (miPoligono);
```

B.3.2. Área de un Polígono

La misma biblioteca también ofrece una función que regresa el área de un polígono, tomando en cuenta si éste tiene huecos. Para hacer uso de esta función se debe de incluir la cabecera:

```
#include <boost/geometry/algorithms/area.hpp>
```

para hacer uso de esta función simplemente se debe de invocar la función area(), que recibe como parámetro un polígono con los puntos ordenados. Ejemplo:

```
float area_poligono = area(miPoligono);
```

B.3.3. Centroide de un polígono

Boost ofrece una función que permite encontrar el centroide de cualquier polígono, para ello es necesario añadir la cabecera:

```
#include <boost/geometry/algorithms/centroid.hpp>
```

para hacer uso de esta función se invoca la función centroid(), que recibe como parámetro un polígono y un punto en el cual devuelve las coordenadas del centroide. Ejemplo:

```
centroid (miPoligono, posicionCentroide);
```

B.3.4. Unión de Polígonos

Boost también ofrece una función para obtener la unión espacial de dos polígonos, los cuales pueden o no incluir huecos, para utilizar esta función es necesario agregar la cabecera:

```
#include <boost/geometry/algorithms/union.hpp>
```

para hacer uso de esta función se invoca la función union_(), la cual recibe tres parámetros, los dos primeros son los polígonos que se quieren unir y el tercero es un vector de polígonos, el cual contiene el o los polígonos resultantes de la unión, los cuales pueden o no contener huecos. Ejemplo:

```
vector <polygon> poligonos_union;
union_(poligono1, poligono2, poligonos_union);
//poligonos_union = poligono1 + poligono2
```

B.3.5. Diferencia de polígonos

Boost también ofrece una función para obtener la diferencia espacial de dos polígonos, los cuales pueden o no incluir huecos, para utilizar esta función es necesario agregar la cabecera:

```
#include <boost/geometry/algorithms/difference.hpp>
```

para hacer uso de esta función se invoca la función difference(), la cual recibe tres parámetros, el primero es el polígono base, el segundo parámetro es el polígono que se desea sustraer al primer polígono y el tercero es un vector de polígonos, el cual contiene el o los polígonos resultantes de la diferencia, los cuales pueden o no contener huecos. Ejemplo:

```
vector <polygon> poligonos_diferencia;
difference(poligono1, poligono2, poligonos_diferencia);
// poligonos_diferencia = poligono1 - poligono2
```

B.3.6. Intersección de polígonos

Boost ofrece una función para obtener la intersección espacial de dos polígonos, los cuales pueden o no contener huecos, para utilizar esta función es necesario agregar la cabecera:

```
#include <boost/geometry/algorithms/intersection.hpp>
```

para hacer uso de esta función se invoca la función intersection() que recibe tres parámetros, los dos primeros son los polígonos que se quieren intersecar y el tercero es un vector de polígonos, el cual contiene el o los polígonos resultantes de la intersección, los cuales pueden o no contener huecos. Ejemplo:

```
vector <polygon> poligonos_interseccion;
intersection(poligono1, poligono2, poligonos_interseccion);
```

B.4. Árbol de Boost

La biblioteca Boost Geometry ofrece una estructura de árbol R-tree para almacenar puntos y posteriormente realizar búsquedas espaciales. Esta estructura puede ser

operada bajo cuatro diferentes algoritmos, cada uno de los cuales tienen sus ventajas y desventajas, en el presente trabajo se decidió emplear el algoritmo cuadrático. Para hacer uso de este tipo de estructura es necesario agregar la cabecera:

```
#include <boost/geometry/index/rtree.hpp>
```

para definir y hacer uso este tipo de estructura se debe de seguir la siguiente estructura para su declaración:

```
boost::geometry::model::point<float, 2, bg::cs::cartesian> punto;
boost::geometry::index::rtree<punto, algoritmo<elementos>>nombre;
```

Ejemplo:

```
boost::geometry::index::rtree<
punto, boost::geometry::index::quadratic<10000> > mytree;
```

B.4.1. Inserción de puntos al árbol

Una vez que se ha definido el árbol se procede a agregar elementos al mismo, para esto se invoca la función del propia árbol insert(), el cual recibe como parámetro el punto que se desea agregar al árbol. Ejemplo:

```
mytree.insert(punto);
```

B.4.2. Búsqueda de puntos que intersecan un polígono

La biblioteca Boost ofrece varios tipos de búsqueda espaciales con los puntos almacenados en el árbol. En este trabajo se empleo la búsqueda de puntos en el árbol que se encuentren dentro de un polígono. Para hacer uso de esta función se invoca el método query(), el cual recibe dos parámetros, el primer parámetro es el polígono de intersección y el segundo parámetro es un vector que contendrá los puntos del árbol que se intersecan con el polígono de referencia. Ejemplo:

```
std::vector<punto> result_p;
mytree.query(boost::geometry::iindex::intersects(poligono),
                std::back_inserter(result_p));
```

Para más información sobre el uso de estos conceptos y funciones se recomienda acudir a la documentación de la biblioteca Boost Geometry.