PARALELIZACIÓN DE BUNDLE ADJUSTMENT EN UNA PLATAFORMA EMBEBIDA

JOSEJULIO MARTÍNEZ MAGAÑA

Maestría en Ciencias de la Computación
Facultad de Matemáticas
Universidad Autonoma de Yucatán

Octubre 2016

RESUMEN

El presente trabajo se desarrolla sobre un sistema autónomo de captura aérea para la creación de mosaicos de imágenes. El sistema actualmente utiliza un modelo geométrico e información en las imágenes para determinar si es requerida una captura de imagen. Actualmente no realiza ningún registro entre las imagenes capturadas. Para tener más información que permita realizar el mosaico de imágenes se propone realizar un registro global, utilizando el método de *Bundle Adjustment*. El algoritmo debe ejecutarse en tiempo real, por lo cual se requiere paralelizarlo en el co-procesador *Neon*, el cual sigue la arquitectura SIMD (siglas en inglés de Single Instruction Multiple Data), que se encuentra embebido en la familia de procesadores *ARM Cortex-A*. Se encontró que es posible paralelizar ciertas operaciones, obteniendo una disminución en el tiempo de ejecución, lo cual pemitiría al sistema autonomo a ejecutar este computo en paralelo. Se validan los resultados probando el algoritmo de registro en imágenes de la retina del ojo. Los resultados demuestran que es posible disminuir el tiempo de cómputo. En algunas rutinas logramos tener una aceleración de tres a seis veces en comparación a una ejecución no paralelizada del algoritmo en la misma plataforma.

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.

Josejulio Martínez Magaña
Mérida, Yucatán,
México
Octubre 2016

AGRADECIMIENTOS

A mi esposa, Patricia e hija Eleani, por su amor incondicional que me inspiran a superarme.

A mis padres, Julio y Linda, mis hermanos Rodrigo y Aldo por su apoyo para salir adelante.

Al Dr. Arturo Espinosa Romero, por su paciencia, apoyo y consejos que me ha brindado.

A mis profesores por todos los conocimientos que amablemente me transmitieron.

A mis amigos y compañeros.

Al Consejo Nacional de Ciencias y Tecnología (CONACYT), que me otorgó la beca con la cual pude sustentarme económicamente durante el estudio de la maestría.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

ACRÓNIMOS

SSD Sum of Squared Differences

SISD Single Instruction Single Data

SIMD Single Instruction Multiple Data

MISD Multiple Instruction Single Data

MIMD Multiple Instruction Multiple Data

UP Unidades de Procesamiento

ZNCC Zero Mean Normalized Cross Correlation

1

INTRODUCCIÓN

1.1 INTRODUCCIÓN

Los mosaicos de imágenes dan un panorama más amplio a gran resolución, algo que una cámara convencional no puede tomar o podría ser muy costoso que lo hiciera. Con varias fotografías que abarquen un paisaje es posible formar el panorama completo. Para esto se utilizan técnicas para estimar como casan las imágenes. Con esta información se puede generar una imagen del panorama completo uniendo las partes como si se tratase de un rompecabezas [3].

Por lo regular las imágenes no casan perfectamente, dado a algunos problemas inherentes al medio, tales como la iluminación. Por ejemplo, si se toma una primera imagen cuando el sol ilumina al máximo y otra cuando el sol ya se ocultó, podremos ver el cambio drástico del mismo paisaje. Adicionalmente, las cámaras pueden tener problemas al capturar secciones del panorama, el balance de blancos, objetos en movimiento pueden aparecer distorsionados o incluso no aparecer en otras imágenes.

Los mosaicos de imágenes son útiles para el análisis de objetos los cuales se requieren tener el panorama completo y no es sencillo o es imposible realizarlo mediante una captura directa. Los panoramas formados con imágenes aéreas son ejemplo de esto, para tener una imagen de una zona amplia, requerimos unir varias imágenes. Otra aplicación de los mosaicos de imágenes se puede encontrar en la medicina, para la obtención de una imagen de la dentadura completa.

En este trabajo se plantea resolver el ajuste automático *in situ* de imágenes de manera simultanea para la construcción de un mosaico de imágenes. En el ámbito del análisis de imágenes aéreas o imáge-

nes médicas. Este trabajo forma parte de una línea de investigación que se desarrolla en la Facultad de Matemáticas de la Universidad Autónoma de Yucatán para la generación de mosaicos de imágenes aéreas y su análisis. Las imágenes son capturadas de manera autónoma utilizando vehículos aéreos no tripulados. En dicho proyecto, a la fecha se ha desarrollado tecnología para la estimación de pose para vehiculos áreos no tripulados [9], el diseño y desarrollo de una metodología para la captura automática de imágenes [13] y el uso de un modelo geométrico para extraer la información de las imágenes [19].

Figura 1: Panorama áereo de la Facultad de Matemáticas[13].

Los primeros mosaicos de imágenes se hacían manualemente, cortando y pegando las imágenes físicas para lograr tener una más grande. Con el surgimiento del procesamiento de imágenes se comenzaron a idear algoritmos para automatizar este proceso. En la actualidad existe una gran variedad de algoritmos que tratan de aminorar los problemas que surgen al casar las imágenes.

Las cámaras, como medio de captura de las imágenes, han ido mejorando sus prestaciones, dando mayor calidad de fotografías. En la actualidad, las cámaras se encuentran presentes en casi todas partes e.g. cámaras digitales, celulares, vehículos aéreos no tripulados, etc.

Figura 2: Mosaico de una retina.[1]

Los dispositivos embebidos han tomado un gran auge en la actualidad, ya que los podemos encontrar en casi cualquier lugar; un ejemplo de ellos son los teléfonos celulares. Estos dispositivos tienen limitado su poder de cómputo, en comparación a las computadoras de escritorio o computadoras portátiles. A pesar de esto, gracias a su ubicuidad, son excelentes candidatos para resolver problemas de cómputo.

Los procesadores *ARM* son una familia de procesadores que son ubicuos en los dispositivos embebidos, los podemos encontrar en una gran variedad de aparatos electrónicos tales como teléfonos móviles, placas de desarrollo y aparatos de entretenimiento.

1 https://miac.unibas.ch/BIA/04-BasicsImageRegistration.html#(10)

Los procesadores de la familia *ARM Cortex-A* cuentan con un coprocesador denominado *Neon* este es capaz de acelerar los algoritmos de procesamiento de multimedia y señales, tales como codificación de video, gráficos, audio, procesamiento de imágenes, etc.

Para el desarrollo del presente trabajo se hace uso de un *BeagleBone Black*. El *BeagleBone Black* es un dispositivo embebido que cuenta con el procesador *ARM Cortex-A*[2] . Este dispositivo nos permite acelerar el computo del ajuste para el mosaico de imágenes mediante su el procesador vectorial *Neon*[3].

En el resto de este capítulo se mencionan brevemente que es un mosaico de imágenes, en que consiste el registro de imágenes y el problema de *Bundle Adjustment*. Finalmente se muestran los objetivos y la estructura del presente trabajo de tesis.

1.2 MOSAICO DE IMÁGENES

El mosaico de imágenes es la alineación de múltiples imágenes en una más grande, dando un panorama de visión más amplio en comparación de una sola imagen. Capel [3] afirma que uno de los factores importantes para la construcción de un mosaico de imágenes es la correcta estimación del conjunto de homografías que mapea correctamente las imágenes entre si. El proceso para lograr una correcta estimación del conjunto de homografías se logra realizando un registro entre las imágenes. El registro de imágenes se puede perfeccionar mediante un método de optimización que permita reducir el error de manera global.

1.3 REGISTRO DE IMÁGENES

El registro de imagen es un proceso en el que se encuentra una relación de correspondencia punto a punto entre imágenes. El problema de correspondencia se puede explicar como el problema de saber

2 https://beagleboard.org/black

3 http://www.arm.com/products/processors/technologies/neon.php

que parte de una imagen pertenece a que parte de otra imagen. Estas correspondencias se dan porque las mismas partes de una escena capturada se visualizan en distintas imágenes. Las diferencias entre múltiples imágenes pueden ser dadas por el movimiento de la cámara con respecto a los objetos que se fotografiaron, el movimiento de los objetos dentro de la escena fotografiada o en general cualquier movimiento relativo de los objetos dentro de la escena fotografiada. Esta relación puede ser expresada como una relación geométrica entre las imágenes.

Los métodos para obtener el registro, de acuerdo con Capel [3], se dividen en dos categorías: métodos directos y métodos basados en características. Los métodos directos operan las dos imágenes en su totalidad, con el fin de obtener la medida de la diferencia que existe entre ambas y poder identificar los cambios que se tuvieron. Los métodos basados en características evitan el tener que operar sobre toda la imagen, se elige un conjunto de puntos, a los que se denominan como *puntos característicos*, sobre estos se realiza el cómputo para lograr identificar la relación existente entre las imágenes.

1.4 BUNDLE ADJUSTMENT

Bundle Adjustment es el problema de refinar de manera simultanea las coordenadas tridimensionales de una reconstrucción visual. Triggs et al. [21] realiza un estudio enfocado principalmente a la comunidad de implementadores del área de visión computacional, en el cual corrige conceptos erroneos que se tiene sobre *Bundle Adjustment*, incluye tópicos de importancia, tales como la selección de la función de costo y métodos de optimización númerica.

Shunping et al. [18] hacen uso de *Bundle Adjustment* para la generación de imágenes panorámicas tomadas mediante un vehículo.

Lourakis y Argyros [11] explican a detalle la implementación de *Bundle Adjustment* de manera genérica. En otro trabajo [10] cuestiona si el algoritmo de *Levenberg-Marquardt* [16] es el algoritmo más eficiente para la solución del problema de *Bundle Adjustment*, mostrando

evidencia de que el algoritmo de *Powell's dog leg* [17] es más eficiente para la resolución de este problema. Obtiene los mismos resultados con un decremento del tiempo de ejecución.

Konolige [7] presenta un trabajo en el que implementan una versión más eficiente del trabajo de Lourakis y Argyros [11], considerando que la relación entre las cámaras (lo que llaman en su trabajo, la segunda estructura) es también dispersa. Para este tipo de problemas, logran mejorar el rendimiento por un orden de magnitud.

A pesar de que el problema de *Bundle Adjustment* se define principalmente para la reconstrucción tridimensional, varios trabajos muestran la posibilidad de utilizar este método para la construcción de mosaicos de imágenes. Mclauchlan et al. [15] lo aplica para la construcción de un mosaico de imágenes con el contenido de un video. Para esto, requiere utilizar imágenes planares o aproximadamente planares. Esta consideración es importante para el presente trabajo, ya que se trabaja con imágenes aproximadamente planares, en las que no afectan las distorciones tridimensionales.

1.5 OBJETIVOS

Se presenta el objetivo de la presente tesis y los objetivos particulares a completar durante el desarrollo de la metodología.

1.5.1 *Objetivo General*

Implementación del algoritmo de *Bundle Adjustment* para realizar el ajuste de valores que describen la relación planar entre pares de imágenes utilizando el co-procesador vectorial *Neon* que se encuentra integrando en los procesadores de la familia *ARM Cortex-A*.

Implementación de un algoritmo para ajustar los valores de una relación planar utilizando paralelismo en un sistema embebido.

1.5.2 *Objetivos particulares*

- Implementar *Bundle Adjustment* en un procesador de la familia *ARM Cortex-A*.
- Determinar las partes suceptibles a ser paralelizables del algoritmo *Bundle Adjustment*.
- Realizar las modificaciones para paralelizar el algoritmo *Bundle Adjustment* mediante las instrucciones del co-procesador *Neon* que se encuentra integrado en los procesadores de la familia *ARM Cortex-A*

1.6 ESTRUCTURA DE LA TESIS

El presente trabajo de tesis se encuentra divido en los siguientes capítulos y se detalla a continuación. El capítulo dos presenta el marco teórico en el que el presente trabajo de tesis se apoya para su desarrollo. El capítulo tres describe el desarrollo de la metología que se siguió para cumplir los objetivos del presente trabajo de tesis. El capítulo cuatro muestra el diseño y los resultados experimentales que fueron llevados a cabo a lo largo del presente trabajo de tesis. El capítulo cinco esta dedicado a las conclusiones del trabajo de tesis y el trabajo a futuro.

2

MARCO TEÓRICO

2.1 INTRODUCCIÓN

En el presente capítulo se describen las bases y fundamentos necesarios para un mejor entendimiento de la metodología del presente trabajo.

En la Sección 2.2 se introducen las herramientas matemáticas para la representación del mundo en un ambiente bidimensional, esto nos da herramientas para representar y operar la relación que guarda una imagen con respecto a otra.

En la Sección 2.3 se introduce el método de la interpolación, la cual permite encontrar valores intermedios que no se conocen. Esto es útil porque, dado que las imágenes son representaciones discretas del mundo, se desconoce el valor en elementos que no fueron discretizados, tal es el caso de una fracción de un pixel, que al hacer los cálculos, se requiere saber.

En la Sección 2.4 se introduce la teoría de grafos, la cual se utiliza para hacer representar en conjunto la representación de múltiples relaciones entre las imágenes.

En la Sección 2.5 se introduce el concepto de matrices dispersas y métodos para una representación eficiente de manera computacional de los valores contenidos en estas matrices, lo cual es útil para disminuir la complejidad computacional en tiempo y en espacio de matrices en la que muchos de sus elementos son cero. En la Sección 2.6 se introducen los métodos de optimización de mínimos cuadrados no lineales, estos dan soporte para la resolución del problema de *Bundle Adjustment*. En la Sección 2.7 se introduce el problema de

Bundle Adjustment, el cual es de interés resolver de manera eficiente en un procesador embebido.

En la Sección 2.8 se introduce la taxonomia de Flynn, la cual pone en contexto de sobre que tipo de paralelización se está trabajando, es importante saber cuales son los que existen y que tipo es el que se va a utilizar, ya que, dependiendo este, es la manera en la que se ataca el problema para poder paralelizarlo.

2.2 GEOMETRÍA PROYECTIVA

La geometría proyectiva es una rama de la geometría que estudia como se proyectan los objetos tridimensionales en un espacio bidimensional, el cual se conoce como el espacio proyectivo. Por simplicidad de las operaciones, utilizamos coordenadas homogéneas, que permite manejar las transformaciones geométricas utilizando álgebra lineal.

En esta Sección se muestran las coordenadas cartesianas y las coordenadas homogéneas, las cuales son representaciones de un punto en el plano. También se muestra una operación que se realiza con estas coordenadas, las transformaciones proyectivas, las cuales permiten representar los cambios entre dos imágenes de una superficie planar.

2.2.1 *Coordenadas cartesianas*

Las coordenadas cartesianas sirven para representar, en el caso de dos dimensiones, un punto en el plano, utilizamos la notación: $X_c = (x_{c_1}, x_{c_2})$, estas coordenadas son con respecto a un origen, y cada uno de los componentes es la distancia mínima a un eje de coordenadas, los cuales son ortogonales entre si.

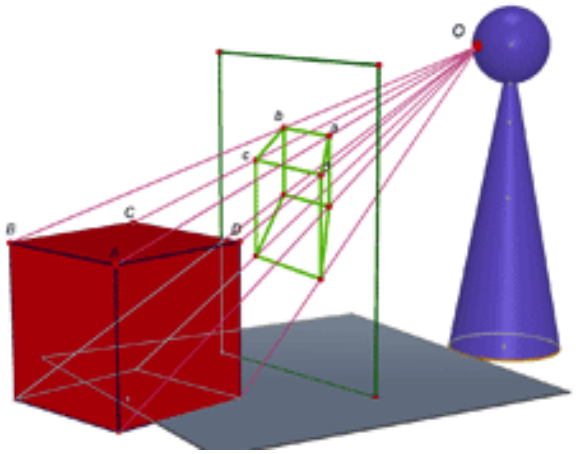

Figura 3: Representación de la geometría proyectiva.

2.2.2 *Coordenadas homogéneas*

Las coordenadas homogéneas son otra manera de representar un punto en el plano, en coordenadas homogéneas usamos la notación: $X_h = (x_{h_1}, x_{h_2}, x_{h_3})$. Las coordenadas homogéneas guardan una relación con las coordenadas cartesianas, la razón entre los componentes de las coordenadas homogéneas corresponde con las coordenadas cartesianas.

$$X_c = \begin{pmatrix} \frac{x_{h_1}}{x_{h_3}} \\ \frac{x_{h_2}}{x_{h_3}} \end{pmatrix} = \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix}. \tag{1}$$

Dada esta forma en la que se plantea la relación entre las coordenadas homogéneas, se puede observar que si multiplicamos una coordenada homogenea por un escalar k distinto de 0, ésta sigue re-

presentando el mismo punto en el plano, esto es porque se cancela el escalar al momento de dividirlos entre kx_{h_3}.

$$
\begin{aligned}
kX_h &= \begin{pmatrix} kx_{h_1} \\ kx_{h_2} \\ kx_{h_3} \end{pmatrix}, \\
X_c &= \begin{pmatrix} \frac{kx_{h_1}}{kx_{h_3}} \\ \frac{kx_{h_2}}{kx_{h_3}} \end{pmatrix} = \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix}.
\end{aligned}
\tag{2}
$$

2.2.3 *Transformación proyectiva*

A continuación se muestra las definición de la transformación proyectiva y los nombres que recibe según las propiedades que cumplen, esta información es una interpretación tomada de Hartley and Zisserman [6].

Se define una transformación planar como una transformación lineal de una coordenada homogénea por una matriz no singular de 3×3 como se muestra en la ecuación (3).

$$
\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \tag{3}
$$

$$x' = Hx.$$

Nótese que si multiplicamos la matriz H por algún escalar k distinto de 0 no se altera la transformación proyectiva, por lo que podemos decir que H es una matriz homogénea que tiene 8 grados de libertad. A esta matriz H se le conoce también con el nombre de homografía.

Las transformaciones proyectivas, se subdividen en 4 tipos, los cuales son:

- Isometría.
- Similitud.
- Transformación afín.
- Transformación proyectiva.

Cada una es una especialización las transformaciones proyectivas, las cuales cumplen ciertas propiedades que nombraremos a continuación.

2.2.3.1 *Isometrías*

Las isometrías son una transformación en el plano $\mathbb{R}^2$ que preserva la distancia Euclidiana. La ecuación (4) muestra la representación de la isometría, en donde $\epsilon = \pm 1$, y si $\epsilon = 1$ la orientación se preserva, y con $\epsilon = -1$ la orientación se invierte.

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} \epsilon\cos\theta & -\sin\theta & t_x \\ \epsilon\sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{4}$$

La isometría se puede representar de manera más sencilla como se muestra en la ecuación (5), donde R es una matriz de rotación de 2×2, la cual es una matriz ortogonal tal que $R^T R = RR^T = I$ y T es un vector de 2×1 en la que se representa la traslación y 0 es un vector de 2×1 nulo. Dos casos especiales de esta transformación ocurren cuando existe una rotación pura ($t = 0$ y cuando es una traslación pura ($R = I$).

$$x' = H_E x = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} x. \tag{5}$$

Una isometría tiene 3 grados de libertad, ya que para definir una se requieren los parámetros de la traslación t_x y t_y así como el parámetro θ el cual considera la rotación.

Entre las propiedades que preserva son la distancia entre dos puntos, el ángulo entre dos líneas y el área de los polígonos.

2.2.3.2 *Similitud*

Una similitud es una transformación isométrica a la que se le agrega un cambio en escala, por lo que tiene cuatro grados de libertad, los tres grados de libertad que se consideran en la isometría (t_x, t_y y θ) y el cambio de escala (S). La representación matricial de la similitud se expresa de la siguiente manera:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{6}$$

La cual se puede expresar de una manera más compacta como:

$$x' = H_s x = \begin{bmatrix} sR & t \\ 0^T & 1 \end{bmatrix}. \tag{7}$$

La similitud no conserva la longitud entre dos puntos, sin embargo la razón entre las longitudes es invariante (ya que la escala es la misma para todos), de manera similar la razón del área es invariante, los ángulos entre las líneas no se ven afectados por la rotación, traslación o escala y las líneas paralelas siguen siendo paralelas.

2.2.3.3 *Transformaciones afines*

Una transformación afín es una transformación lineal no singular seguida de una traslación, la cual podemos representar como:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{8}$$

y la podemos representar de manera más compacta como:

$$x' = H_A x = \begin{bmatrix} A & t \\ 0^T & 1 \end{bmatrix}. \tag{9}$$

donde A es una matriz no singular de 2×2, la cual puede ser descompuesta como:

$$A = R(\theta)R(-\phi)DR(\phi). \tag{10}$$

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

$$D = \begin{bmatrix} \lambda_x & 0 \\ 0 & \lambda_y \end{bmatrix}. \tag{11}$$

Por lo que, la matriz A es un escalamiento de λ_x en el eje x y λ_y en el eje y al resultado de rotar la imagen en ϕ, la imagen se retorna a su rotación original y se rota en θ. Por lo cual se tienen seis grados de libertad, θ, ϕ, λ_x, λ_y, t_x, t_y.

Este tipo de transformaciones no preserva los ángulos entre las líneas, pero si preserva el paralelismo entre las líneas, la razón entre las distancias de las líneas y la razón entre las áreas.

2.2.3.4 *Transformaciones proyectivas*

Es la forma general de una transformación lineal de coordenadas homogéneas. La cual está definida como:

$$x' = H_p x = \begin{bmatrix} A & t \\ V^T & v \end{bmatrix} x. \tag{12}$$

En donde $V = (v_1, v_2)^T$ y $v = 1$, con lo cual se tienen 8 parámetros para definir en la matriz, por lo que se tienen ocho grados de libertad.

La propiedad invariante fundamental de este tipo de proyecciones es la razón cruzada de cuatro puntos x_1, x_2, x_3, x_4 que son colineales, y se encuentra definida como:

$$\text{Cross}(x_1, x_2, x_3, x_4) = \frac{|x_1x_2|}{|x_1x_3|} \cdot \frac{|x_3x_4|}{|x_2x_4|}. \tag{13}$$

2.2.4 *Métricas de similitud en las imágenes*

Existen ciertas métricas para medir la similitud entre secciones de imágenes. Estas métricas ayudan a tomar decisiones al momento de considerar que secciones de una imagen corresponden entre si. Entre las existentes, se analizaron las que tenían un menor costo computacional, ya que el objetivo es implementar el algoritmo en un dispositivo embebido.

2.2.4.1 *Sum of Squared Differences (SSD)*

Sum of Squared Differences (SSD) [1] es una función que cuantifica la diferencia que existe entre dos imágenes, para esto realiza la suma de los errores al cuadrado para los puntos, con esto se puede determinar si los puntos corresponden entre si. Tiene un costo de cómputo muy bajo, ya que únicamente se tiene que efectuar una resta y una multiplicación por punto incluido, sin embargo no es buena cuando existen cambios en la iluminación muy bruscos o existen rotaciones muy grandes.

Al evaluar esta función no solo se evalúa el punto característico, si no que se evaluá una vecindad, de esta forma se toma en cuenta tanto el punto como lo que se encuentra alrededor, para esto se fija una ventana de búsqueda, en la que el punto central es el punto característico, el tamaño de la vecindad se puede incrementar para obtener una exactitud mayor de la relación, sin embargo al hacerlo más grande se eleva el costo computacional de la función, ya que se tiene que evaluar más puntos, en la Figura 4 se muestra un ejemplo una ventana de búsqueda.

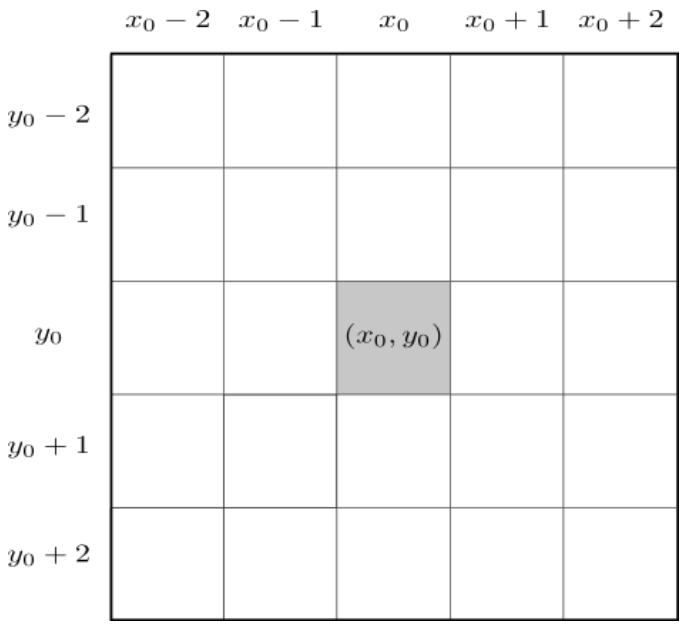

Figura 4: Ventana de búsqueda de 5×5 centrada en el punto característico (x_0, y_0).

La función de SSD está definida en términos del par de imágenes a comparar, la relación que existe entre estas dos imágenes y una ventana de búsqueda:

$$\mathrm{SSD}(I_1, I_2, H, w) = \sum_{x \in w} [I_1(x) - I_2(Hx)]^2. \tag{14}$$

Mientras más pequeño sea el valor de esta función más similares son los puntos, cuando es 0 los puntos son idénticos.

2.2.4.2 *Zero Mean Normalized Cross Correlation (ZNCC)*

La función ZNCC es otra función para hacer la comparación entre puntos, esta función es robusta a los cambios de iluminación [4], sin embargo no es buena cuando existen cambios grandes en las rotaciones o la perspectiva. La función realiza una correlación cruzada, primero normalizando los valores de cada punto, esto se logra restando la media y dividendo el resultado entre su desviación estándar.

La función de ZNCC se define como:

$$
\begin{aligned}
n &= \textit{Numero de elementos de } w \\
\mathrm{ZNCC}(I_1, I_2, H, w) &= \frac{1}{n} \sum_{x \in w} \left[\frac{(I_1(x) - \mu_{I_1(w)})(I_2(x) - \mu_{I_2(w)})}{\sigma_{I_1(w)} \sigma_{I_2(w)}} \right] \\
\mu_{I_i(w)} &= \frac{\sum_{x \in w} x}{n}, \sigma_{I_i(w)} = \sqrt{\frac{\sum_{x \in w} (x - \mu_{I_i(w)})^2}{n}} \\
-1 &\leqslant \mathrm{ZNCC}(I_1, I_2, H, w) \leqslant 1.
\end{aligned}
\tag{15}
$$

Mientras más cercano sea el valor a 1, mayor correlación guarda entre los puntos a evaluar, con lo que se puede decir que son el mismo punto característico.

2.3 INTERPOLACIÓN

En ocasiones es de interés conocer el valor de una función de un punto el cual no está especificado o se desconoce su valor, sin embargo, si se conoce el valor que toma la función en algunos puntos (los cuales pudieron haber sido obtenidos mediante un muestreo) se puede generar el valor del punto que se quiere a partir de estos puntos. Para hacer esto, se puede definir una función más sencilla que nos aproxima el valor que se quiere obtener. Existen varios métodos de interpolación, entre los más sencillos se encuentra la interpolación lineal y sus variante, para funciones de dos argumentos, la interpolación bilineal.

2.3.1 *Interpolación Lineal*

Este es uno de los métodos más sencillos de interpolación para una función unidimensional $f(x)$, se basa en remplazar la función $f(x)$ por una función lineal $L(x)$ en el punto que se desea aproximar, dado que se conoce el valor que toma la función en dos puntos dados x_1 y x_2 tales que $x_1 < x < x_2$.

$$
L(x) = a(x - x_1) + b. \tag{16}
$$

En donde los valores de a y b se escogen de tal manera que los valores de $L(x)$ coincidan con los valores de $f(x)$ para los dos puntos conocidos, el desarrollo se muestra en la ecuación (17).

$$\begin{aligned} L(x) &= a(x - x_1) + b, \\ L(x_1) &= f(x_1), \\ L(x_2) &= f(x_2), \\ L(x_1) &= f(x_1) = a(x_1 - x_1) + b = b, \\ L(x_2) &= f(x_2) = a(x_2 - x_1) + b, \\ a &= \frac{(f(x_2) - b)}{(x_2 - x_1)} = \frac{(f(x_2) - f(x_1))}{(x_2 - x_1)}. \end{aligned} \tag{17}$$

$$L(x) = \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) + f(x_1). \tag{18}$$

Al sustituir los resultados obtenidos de a y b en la ecuación (16) nos queda la formula de la interpolación lineal, mostrada en la ecuación (18), con la que se puede aproximar el valor de $f(x)$ utilizando dos puntos conocidos (x_1, x_2), dado que la aproximación se realiza resolviendo una función lineal, el resultado puede variar del valor correcto en una función no-lineal, se puede observar en la Figura 5 el comportamiento de aproximar el valor utilizando una interpolación lineal.

2.3.2 *Interpolación bilineal*

La interpolación bilineal es una extensión de la interpolación lineal para funciones bivariadas, estrictamente hablando no se trata de una función lineal, pero el desarrollo se basa en el mismo que la interpolación lineal, el cual es realizar la interpolación lineal en un eje y a continuación se realiza en el otro eje. Dado los puntos $P_{i,j} = (x_i, y_j)$, el objetivo es aproximar el valor que toma una función f en un punto $P_{0,0}$, para este método, se considera que se conoce el valor que toma la función f en 4 puntos $P_{i,j}$ para $i, j \in \{1, 2\}$, tal que $x_1 < x_0 < x_2$ y $y_1 < y_0 < y_2$.

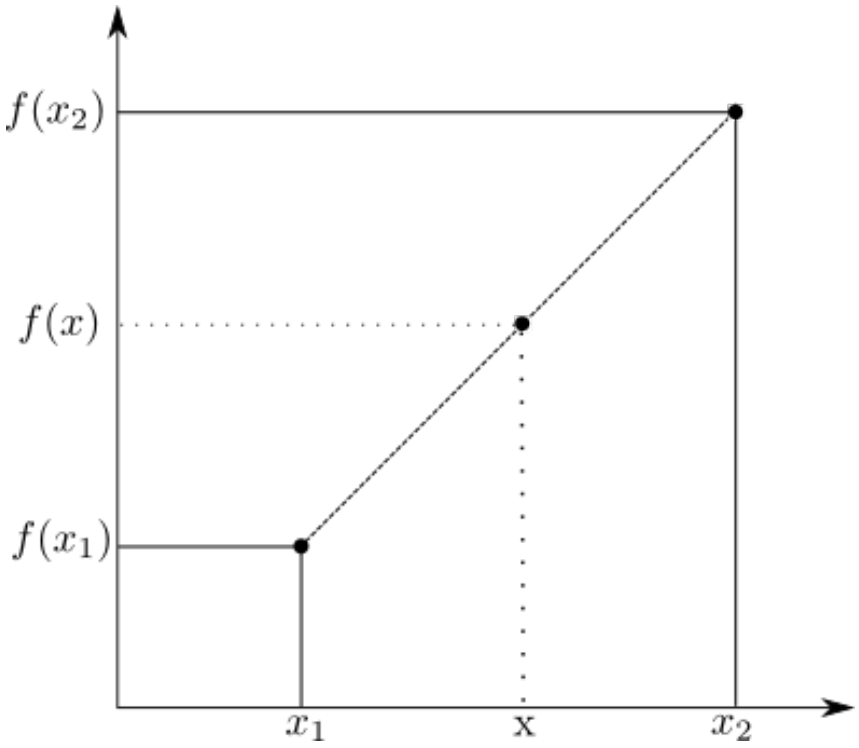

Figura 5: Representación gráfica de la interpolación lineal.

Antes de comenzar con el proceso, se muestra que la ecuación de la interpolación (18) se puede expresar también como se muestra en la ecuación (19), lo que nos servirá para generalizar unos pasos en el desarrollo de la interpolación bilineal.

$$\begin{aligned} L(x) &= f(x_2)\frac{x-x_1}{x_2-x_1} + f(x_1)\left[1 - \frac{x-x_1}{x_2-x_1}\right] \\ &= f(x_2)\frac{x-x_1}{x_2-x_1} + f(x_1)\left[\frac{x_2-x_1-x+x_1}{x_2-x_1}\right] \\ &= f(x_2)\frac{x-x_1}{x_2-x_1} + f(x_1)\frac{x_2-x}{x_2-x_1}. \end{aligned} \tag{19}$$

Se comienza haciendo una interpolación lineal en el eje x, para esto usamos la ecuación (19).

$$\begin{aligned} f(P_{0,1}) &= f(P_{2,1})\frac{x_0-x_1}{x_2-x_1} + f(P_{1,1})\frac{x_2-x_0}{x_2-x_1}, \\ f(P_{0,2}) &= f(P_{2,2})\frac{x_0-x_1}{x_2-x_1} + f(P_{1,2})\frac{x_2-x_0}{x_2-x_1}. \end{aligned} \tag{20}$$

Posterior a esto, se utiliza este resultado para hacer una interpolación lineal en el eje y.

$$
\begin{aligned}
f(P_{0,0}) &= f(P_{0,2})\frac{y_0 - y_1}{y_2 - y_1} + f(P_{0,1})\frac{y_2 - y_0}{y_2 - y_1} \\
&= \left[f(P_{2,1})\frac{x_0 - x_1}{x_2 - x_1} + f(P_{1,1})\frac{x_2 - x_0}{x_2 - x_1}\right]\frac{y_0 - y_1}{y_2 - y_1} \\
&+ \left[f(P_{2,2})\frac{x_0 - x_1}{x_2 - x_1} + f(P_{1,2})\frac{x_2 - x_0}{x_2 - x_1}\right]\frac{y_2 - y_0}{y_2 - y_1} \\
&= \frac{1}{(x_2 - x_1)(y_2 - y_1)}\Big[f(P_{2,1})(x_0 - x_1)(y_0 - y_1) \\
&+ f(P_{1,1})(x_2 - x_0)(y_0 - y_1) \\
&+ f(P_{2,2})(x_0 - x_1)(y_2 - y_0) \\
&+ f(P_{1,2})(x_2 - x_0)(y_2 - y_0)\Big].
\end{aligned}
\tag{21}
$$

La ecuación (21) muestra la fórmula de la interpolación bilineal, y en la Figura 6 se muestra gráficamente el desarrollo de la ecuación y la Figura 7 muestra la interpretación gráfica de este resultado.

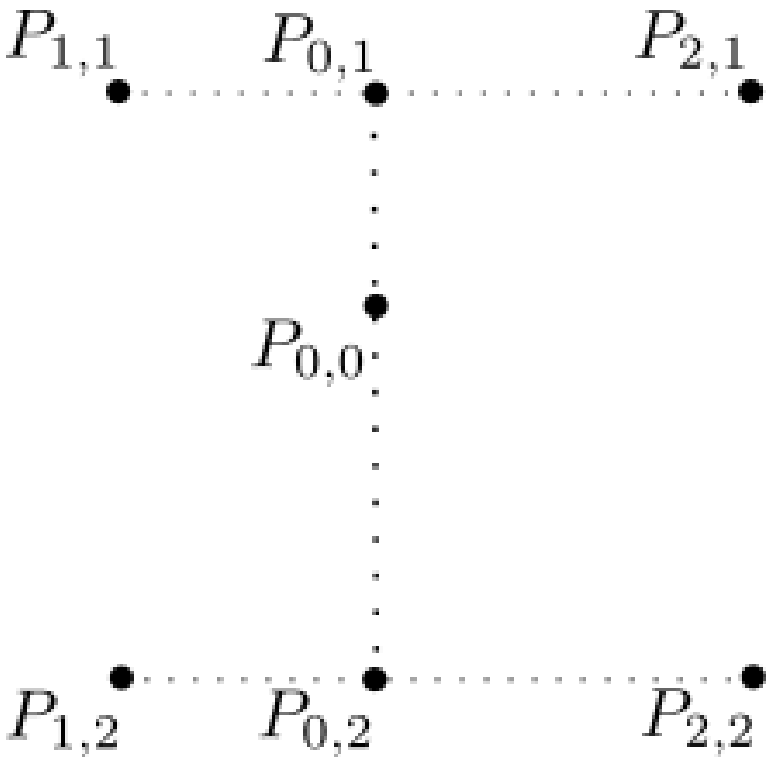

Figura 6: Representación gráfica del desarrollo de la interpolación bilineal por medio de tres interpolaciones lineales. La interpolación lineal entre $P_{1,1}$ y $P_{2,1}$, $P_{1,2}$ y $P_{2,2}$ y finalmente entre $P_{0,1}$ y $P_{0,2}$.

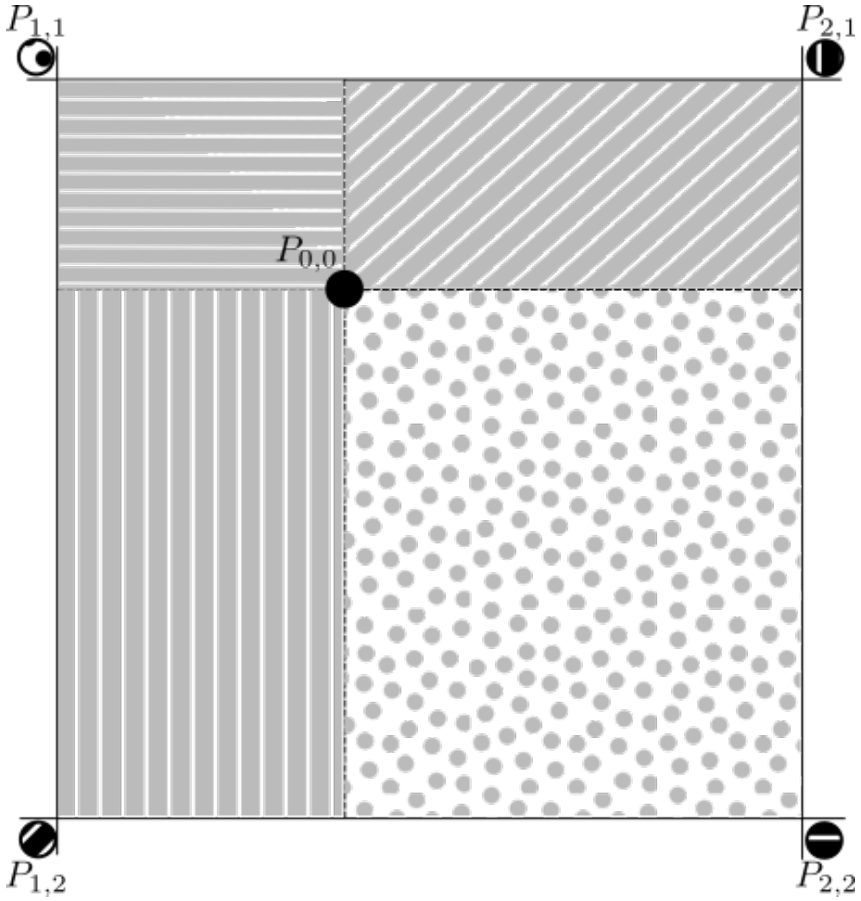

Figura 7: Representación gráfica de la interpolación bilineal. Mientras más cerca se encuentre el punto de una esquina, ésta contribuye más con el resultado final.

2.3.2.1 *Cuadrado unitario*

Si escogemos un sistema de coordenadas en los que los cuatro puntos en los que x es conocido son $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$ podemos simplificar la ecuación como se muestra en (22).

$$\begin{aligned} f(P_{0,0}) = {} & f(P_{1,1})(1-x_0)y_0 + f(P_{1,2})(1-x_0)(1-y_0) \\ & + f(P_{2,1})x_0y_0 + f(P_{2,2})x_0(1-y_0). \end{aligned} \tag{22}$$

2.4 TEORÍA DE GRAFOS

La teoría de grafos es una rama de las matemáticas discretas que estudia las propiedades de los grafos, los cuales son una estructura que consta de dos elementos, un conjunto de nodos (vértices o puntos) y un conjunto de aristas, que pueden estar dirigidos o no.

2.4.1 *Tipos de grafos*

Los grafos, de acuerdo a sus propiedades los podemos clasificar en:

- Grafo simple. Es el grafo que acepta una sola arista uniendo dos vértices cualquiera.
- Grafo dirigido. Son aquellos en los que las aristas tienen una dirección, es decir, empieza en un vértice y termina en otro. Se representa gráficamente por una flecha.
- Grafo etiquetado. Son aquellos a los que se les añade un valor a la arista o un etiquetado a los vértices.
- Hipergrafos. Son aquellos en que las aristas inciden en tres o más nodos.

2.4.2 *Grafo Dirigido Etiquetado*

Igual conocidos como dígrafos etiquetados, son los grafos dirigidos que constan de un valor en cada nodo o arista. Este valor puede ser un costo, una función de relación entre los nodos, etc.

2.5 MATRICES DISPERSAS

En análisis numérico, una matriz se considera dispersa cuando la mayoría de los elementos de la matriz es cero. Para medir la disper-

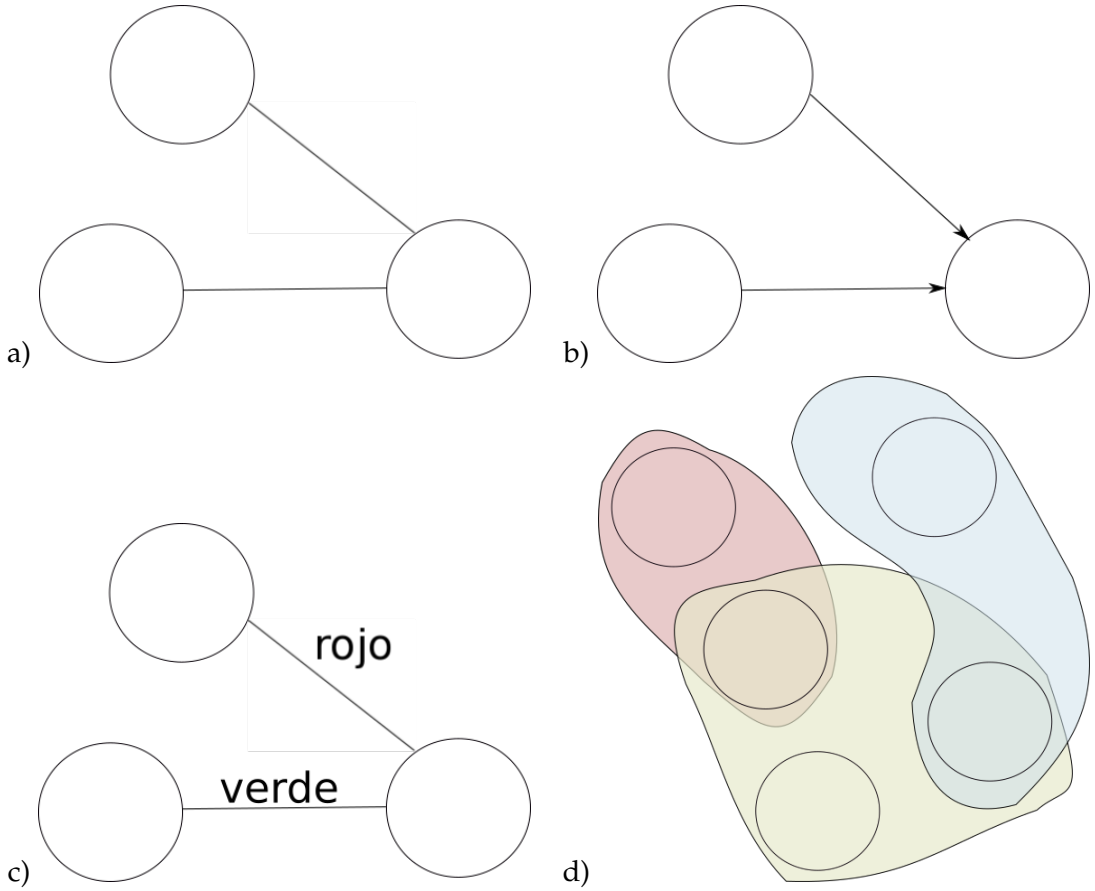

Figura 8: Representación gráfica de los grafos a) Grafo simple; b) Grafo dirigido; c) Grafo etiquetado; d) Hipergrafo.

sión de la matriz, se divide el número de elementos que son cero, entre el número de elementos de la matriz. A esto se le conoce como *grado de dispersión*.

De manera conceptual, este tipo de matrices pueden aparecer al tener sistemas en los que las variables no guardan relación entre si. Un ejemplo de esto puede ser una matriz que representa un grafo no dirigido, cuando dos nodos no se encuentran conectados, se asigna el valor 0 en el cruce de estas variables. Si se tiene un grafo con pocas aristas y muchos nodos, se tendría una matriz dispersa.

Por la naturaleza de estas matrices, es posible idear métodos para comprimir el espacio necesario para almacenar estas matrices.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 52 & 0 & 0 & 0 & 0 \\ 0 & 56 & 0 & 0 & 0 & 14 & 0 & 79 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 31 & 64 \\ 1 & 0 & 0 & 0 & 66 & 0 & 0 & 0 & 0 \\ 0 & 24 & 1 & 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 40 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 33 & 0 & 0 & 51 & 0 & 74 \\ 0 & 7 & 0 & 0 & 0 & 0 & 0 & 46 & 0 \end{bmatrix}$$

Figura 9: Matriz dispersa, se observa que la mayoría de los elementos son igual a cero.

2.5.1 *Almacenamiento de una matriz dispersa*

Una matriz se puede almacenar como un arreglo bidimensional, en el que cada elemento del arreglo, representa una posición $a_{i,j}$ de la matriz. Se sabe que el valor de muchos de estos elementos es cero, podemos pensar en omitir estos valores y solo almacenar aquellos que tienen valor, de esta manera se ahorra espacio, para hacerlo de una manera en la que se puede reconstruir la matriz original o saber la posición que ocupa cada elemento en la matriz, es necesario apoyarse de otras estructuras.

$$\begin{bmatrix} 1 & 0 & 4 & 0 \\ 0 & 3 & 0 & 33 \\ 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Figura 10: Matriz dispersa, la mayoría de los elementos son igual a cero. Se utiliza como referencia para mostrar la forma en que se almacena dependiendo cada formato.

Algunos formatos conocidos para el almacenamiento de matrices dispersas son:

- Diccionario de llaves (DOK).
- Lista de listas (LIL).
- Almacenamiento comprimido por columnas (CCS).

2.5.1.1 *Diccionario de llaves*

La manera de almacenar la información de la matriz en un diccionario de llaves, consiste en utilizar los índices de la fila y columna como llaves de un diccionario, y el valor de la matriz corresponde al valor de la llave (fila, columna), los valores que no se encuentren en el diccionario se toman como si fueran 0. Esta es una manera sencilla de formar el diccionario, y es eficiente al ir agregando nueva información conforme se vayan agregando o quitando valores, sin embargo es poco eficiente al momento de ir iterando sobre los valores diferentes de 0. Normalmente se utiliza este método para tener una base y posterior convertirlo a otro formato que sea más eficiente.

2.5.1.2 *Lista de listas*

Listas de listas, para cada fila se crea una lista, y para valor distinto de 0, se agrega una entrada con el índice de la columna y el valor de la celda. Este es un formato bueno para hacer una construcción incremental de la matriz.

$$
\begin{aligned}
(1,1) &= 1,\\
(1,3) &= 4,\\
(2,2) &= 3,\\
(2,4) &= 33,\\
(3,3) &= 2,\\
(4,1) &= 1.
\end{aligned}
$$

Figura 11: Matriz dispersa (10) representada utilizando un *diccionario de llaves*.

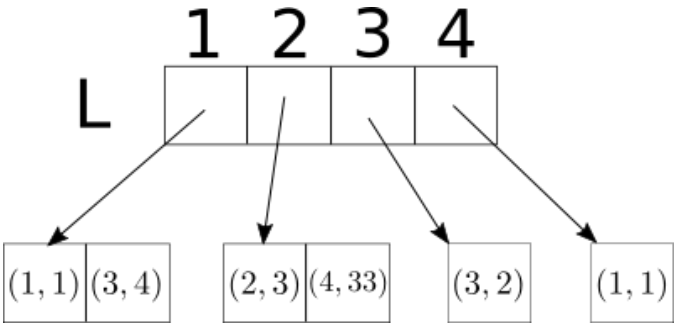

Figura 12: Matriz dispersa (10) representada utilizando una *lista de listas*. Cada elemento de la lista L representa una fila, la cual es una lista con tuplas de la columna y el valor correspondiente a la fila y columna.

2.5.1.3 *Almacenamiento comprimido por columnas*

Este formato pone de manera contigua (en memoria) los valores subsecuentes que no sean igual a cero. Se utilizan tres arreglos, uno almacena los valores distinto de cero de la matriz (val) y dos se utilizan de apoyo para indexar el contenido (ind_filas e ind_cols). Los valores (val) se almacenan de la misma manera que si se recorriera la matriz por columnas, en el primer arreglo de índices (ind_filas) se almacena el índice de la fila en el que se encuentre ese valor, esto es: $val(k) = a_{i,j} \implies ind\_filas(k) = i$ y el otro arreglo ind_cols almacena el índice de val en el que comienza una nueva columna, esto es: $val(k) = a_{i,j} \implies ind\_cols(i) \leqslant k < ind\_cols(i+1)$ y por convención, se agrega un elemento más ind_cols en el que se almacena el número de elementos distintos de cero más 1, la Figura 13 muestra un ejemplo del almacenamiento.

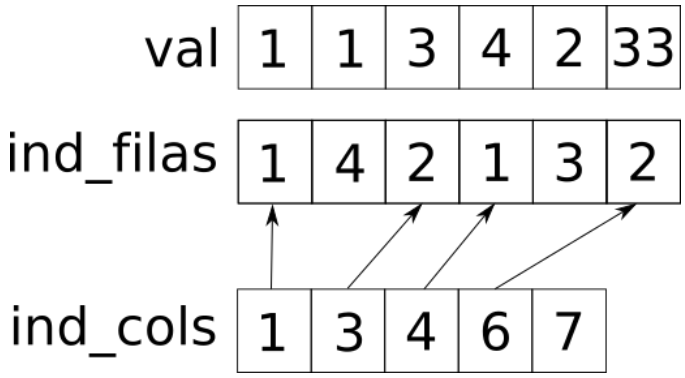

Figura 13: Matriz dispersa (10) representada utilizando el *almacenamiento comprimido por columnas.*

2.6 MÍNIMOS CUADRADOS NO LINEALES

En el desarrollo de esta Sección se describe brevemente algunos métodos de mínimos cuadrados no lineales, esta información está basada en el excelente trabajo de Madsen et al. [14] por lo que se recomienda consultarlo para información más detallada.

Los métodos de mínimos cuadrados surgen al tratar de aproximar la solución de un sistema en el cual se tienen más ecuaciones que incógnitas. En los problemas de este tipo por lo general se trata minimizar la suma de los errores al cuadrado. Estos problemas surgen en muchas áreas, se tiene mucha información pero no se sabe como fue generada, es decir, se tienen muchas muestras y se quiere encontrar el modelo que mejor se ajuste a estos datos.

Estos problemas se pueden resolver utilizando métodos generales de optimización, sin embargo estos no resultan muy eficientes. A continuación se presenta una descripción de algunos métodos que logran un desempeño mejor que una convergencia lineal y algunas veces convergencias cuadráticas, esto sin requerir de la implementación de segundas derivadas parciales.

Estos métodos utilizan un punto de partida, es decir, se requiere indicar una aproximación o un punto de partida inicial, y una serie

de configuraciones que pueden variar dependiendo el problema que se trate de resolver.

$$f(x+h) = f(x) + J(x)h + O(||h||^2). \tag{23}$$

2.6.1 *Método de Gauss-Netwon*

El método de *Gauss-Newton* [2] es una modificación del método de Newton, utilizado para encontrar el mínimo de una función, se implementa utilizando las primeras derivadas de una función, en algunos casos se logra una convergencia cuadrática (al igual que el método de Newton para problemas de optimización). Sin embargo, a diferencia del método de Newton, tiene la ventaja de que no se requiere el cálculo de las segundas derivadas.

2.6.2 *Método de Levenberg-Marquardt*

Este método de optimización, también conocido como el método de *Gauss-Newton amortiguado*, en el cual se agrega un parámetro que actuá como un amortiguador, que asegura la dirección del descenso y el tamaño de los pasos, dependiendo que tan lejos se encuentre la solución.

2.6.3 *Método de Powell's Dog Leg*

El método de *Powell's Dog Leg* [17] trabaja en combinación con el método de *Gauss-Netwon* y el *descenso del gradiente*, controlado explicitamente por un radio, llamado la región de confianza. Se inicializa con un radio, y se va actualizando dependiendo la calidad del modelo lineal, disminuyendo o aumentando este radio de confianza.

2.7 BUNDLE ADJUSTMENT

Bundle Adjustment es el problema refinar de manera simultanea las coordenadas tridimensionales que definen la geometría de una escena, así como los parámetros que definen el movimiento y las características ópticas de las cámaras que fueron utilizadas para adquirir las imágenes de acuerdo a un criterio óptimo, es decir, que los parámetros se encuentran minimizando una función de costo que cuantifica el error total, i.e. que tan mal ajustado se encuentra con respecto al modelo geométrico. El nombre se refiere a los manojos (*bundle*) de haces de luz que salen de cada punto y entran en el centro de cada cámara, los cuales son ajustados tomando en consideración todos los puntos y todas las cámaras.

Se puede aplicar un método de optimización para minimizar el error entre cada par de imágenes, sin embargo, si se tiene una gran cantidad de imágenes, el disminuir el error entre un par de imágenes podría incrementar el error que existe entre otros pares. Por lo que se requiere tomar en consideración a todas las imágenes como un solo conjunto, y hacer el ajuste de todos estos parámetros, tal que la suma de los errores entre pares sea la mínima posible.

Por lo que *Bundle Adjustment* se reduce a minimizar un error de re-proyección entre lo observado y lo que se estimó, el cual puede ser expresado como una suma de las diferencias al cuadrado de un número grande de funciones no lineales, esta minimización se puede lograr utilizando un método de mínimos cuadrados no lineales, tales como Levernberg-Marquardt o Powell's dog-leg. Considerando que estos métodos son iterativos y que en cada iteración se tiene que trabajar con una matriz jacobiana de $N_{observaciones} \times N_{incognitas}$. Con esto queda claro que estos métodos de propósito general son computacionalmente muy demandantes, afortunadamente este tipo de problemas nos generan matrices dispersas, debido a la poca relación que existe entre muchos de los parámetros, i.e. los parámetros de las cámaras no se relacionan entre si, y los parámetros de los puntos solo se relacionan con las cámaras en la que estos puntos son visibles, esta configuración se puede manejar utilizando matrices dispersas.

Lourakis [11, 12] detalla el diseño e implementación de un algoritmo de *Bundle Adjustment* utilizando una versión del algoritmo de optimización *Levenberg-Marquardt* que toma en consideración la poca relación entre varios de los parámetros que se utilizan. Se ha hecho mucha investigación para minimizar el tiempo de ejecución y lograr aumentar la cantidad de imágenes que se puedan procesar.

Lourakis y Argyros [10] realizaron un análisis utilizando el método de *Powell's Dog Leg*, con el que ha mostrado evidencia de que este método brinda mejoras en el tiempo de ejecución en comparación a *Levenberg-Marquardt* al usarlo para resolver el problema de *Bundle Adjustment*, se muestra una tabla con las comparaciones en la Figura 14.

Sequence	# imgs	# vars	initial error	final error		func/jac evals		iter./sys. solved		exec. time	
				DL	LM	DL	LM	DL	LM	DL	LM
"movi"	59	5747	5.03	0.3159	0.3159	11/3	18/18	3/3	18/18	1.06	5.03
"sagalassos"	26	5309	11.04	1.2704	1.2703	16/6	44/33	6/6	33/44	1.29	6.98
"arenberg"	22	4159	1.35	0.5400	0.5399	11/3	25/20	3/3	20/25	0.75	4.95
"basement"	11	992	0.37	0.2448	0.2447	15/9	29/21	9/9	21/29	0.18	0.37
"house"	10	1615	1.43	0.2142	0.2142	10/3	25/19	3/3	19/25	0.11	0.51
"maquette"	54	15999	2.15	0.1765	0.1765	12/3	31/23	3/3	23/31	1.77	12.16
"desk"	46	10588	4.16	1.5761	1.5760	11/3	32/23	3/3	23/32	1.27	9.58
"calgrid"	27	2355	3.21	0.2297	0.2297	10/3	20/20	3/3	20/20	2.61	16.96

Figura 14: Comparación entre los algoritmos *Levenberg-Marquardt*(LM) y *Powell's Dog Leg*(DL) [10]

2.8 TAXONOMIA DE FLYNN

Flynn [5] clasifica las arquitecturas que una computadora pueda tener de acuerdo al número de instrucciones concurrentes que pueda realizar y los flujos de datos que estén disponibles para operar. De acuerdo con esta clasificación, los procesadores pueden ser:

- *Single Instruction Single Data* (SISD).
  Clasifica a las arquitecturas convencionales, en el cual se tiene un flujo de información y un proceso a realizar.

- *Single Instruction Multiple Data* (SIMD).
  Son las arquitecturas que usan un procesamiento en paralelo, la misma instrucción se ejecuta sobre múltiples datos.

- *Multiple Instruction Single Data* (MISD).
  Este tipo de arquitecturas tiene múltiples instrucciones que se aplican sobre el mismo dato.

- *Multiple Instruction Multiple Data* (MIMD).
  Engloba a las arquitecturas que explotan el paralelismo al procesar diversos datos con múltiples instrucciones.

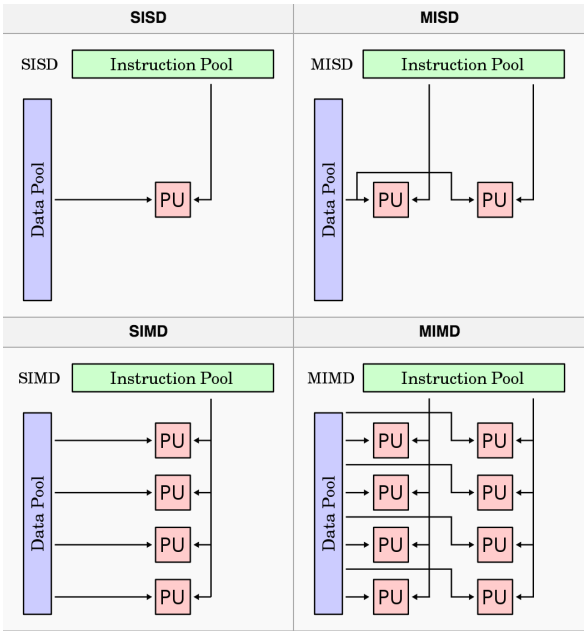

Figura 15: Taxonomía de Flynn

2.8.1 *Arquitectura Single Instruction Multiple Data (SIMD)*

Un tipo de procesamiento en paralelo ocurre en los procesadores SIMD, los cuales explotan el paralelismo al tener varias Unidades de Procesamiento (UP), las cuales pueden ejecutar operaciones en un conjunto de datos. La característica de este tipo de arquitectura es que las Unidades de Procesamiento (UP) ejecutan la misma operación en un momento dado sobre un conjunto de datos.

2.8.1.1 *ARM Neon*

El coprocesador vectorial *Neon* del procesador *ARM* implementa una arquitectura SIMD utilizando 16 registros de 128-bits, que también

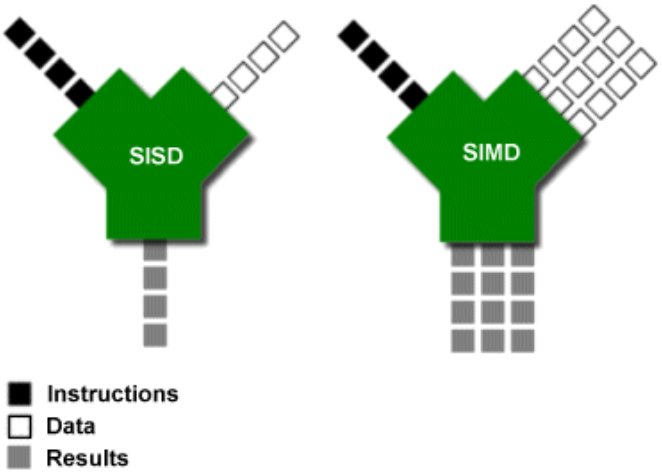

Figura 16: Esquema del funcionamiento de un SIMD, entran múltiples datos a los cuales se les aplica una serie de instrucciones y se obtienen múltiples datos.

pueden ser interpretados como 32 registros de 64-bits. Los registros son considerados como vectores de elementos del mismo tipo, y los tipos que soporta son: enteros con y sin signo de 8, 16, 32 y 64 bits, así como flotantes de precisión sencilla. Soporta una amplia gama de operaciones (e.g. suma, multiplicaciones, multiplicaciones acumulando, valor absoluto, restas.), que se aplican a los vectores, al realizar una operación sobre un vector, la operación se realiza al mismo tiempo en cada uno de los elementos del vector, como se muestra en la Figura 17, una suma de dos vectores.

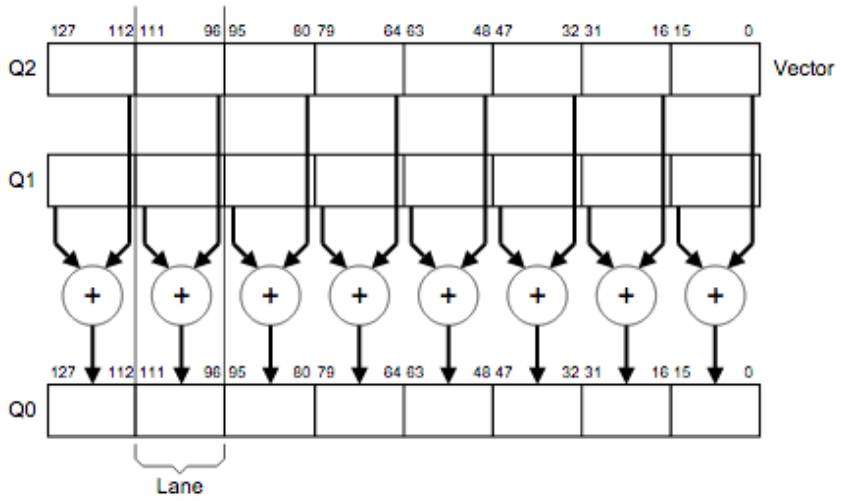

Figura 17: Suma de dos vectores de 8 elementos (16-bits cada elemento) en el procesador Neon.

3

METODOLOGÍA

En el presente capítulo se presenta la metodología para el desarrollo e implementación del algoritmo de *Bundle Adjustment* en el procesador vectorial Neon de un *Beaglebone Black*.

3.1 RELACIÓN ENTRE IMÁGENES

Las imágenes de una misma escena plana guardan una relación geométrica, conocida como Homografía, sin embargo no siempre es posible conocerla a priori. Para encontrar esta relación requerimos identificar puntos entre pares de imágenes para poder estimar la relación geométrica que guardan entre sí. Primero se procede a definir una manera de representar esta relación para posteriormente mostrar la manera de encontrar la relación entre las imágenes.

Una manera de representar la relación que tienen imágenes de una misma escena es utilizando un dígrafo, en el que cada nodo representa una imagen y las aristas representan la dirección en la que la relación geométrica está plasmada, así mismo tienen asignado en cada arista la relación geométrica que mapea una imagen a otra, es decir, si se tienen las imágenes I_1 e I_2 y se sabe que la relación geométrica entre estos de I_1 a I_2 es $H_{1,2}$ entonces se dice que existe una arista dirigida que va de I_1 a I_2 y tiene asignado el valor de $H_{1,2}$. Así mismo, se sabe que estas relaciones se pueden invertir, por lo que, con esta misma representación, sabemos que la relación geométrica para ir de I_2 a I_1 es $H_{2,1} = H_{1,2}^{-1}$.

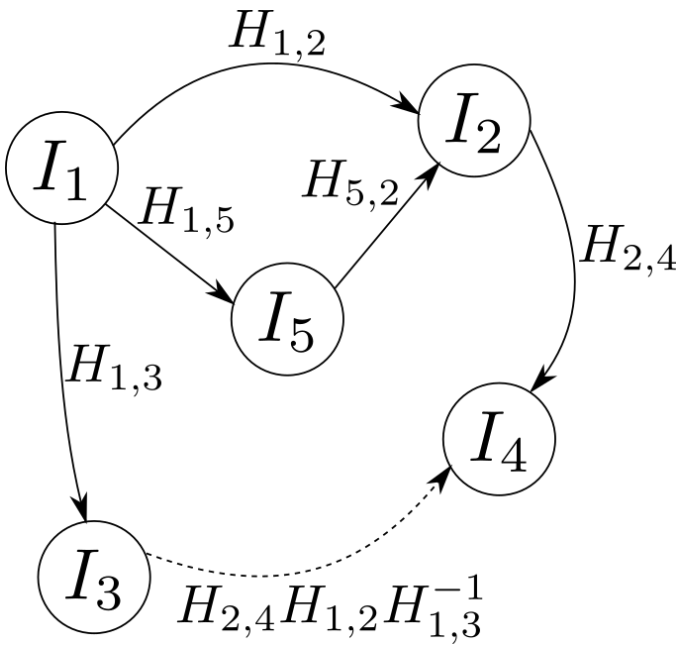

Figura 18: Representación en dígrafos de las relaciones geométricas entre imágenes, nótese que aunque la relación entre I_3 e I_4 no fue encontrada, esta se puede expresar como una concatenación de las relaciones que existen en un camino de I_3 a I_4.

3.1.1 *Búsqueda de correspondencias*

Se utiliza el algoritmo de Harris para la búsqueda de esquinas, los cuales serán los puntos característicos, para las imágenes que presenten un traslape, se tiene una estimación de la relación planar que existe entre ambas imágenes, por lo que, dada una imagen I_a que se traslape en I_b y un punto $X \in I_a$ se puede aproximar un punto $X' \in I_b$

3.2 REFINAMIENTO DE LA TRANSFORMACIÓN PLANAR

Inicialmente se cuenta con una primera aproximación de los cambios que existen entre cada par de imágenes, sin embargo, se sabe que estas aproximaciones tienen un error, el cual se sabe que está relacionado con la manera en la que se hizo la aproximación. Antes de procesar estos datos con *Bundle Adjustment* es deseable minimizar el

error en cada par de imágenes, para de esta manera acercarnos más a la solución.

Para esto se hace una búsqueda exhaustiva en una pequeña vecindad de la transformación, es decir, variando los parámetros que forman la transformación. El saber cuanto variar cada parámetro está relacionado con la información que se puede conocer de como fue el cálculo y un estimado de que tanto es la incertidumbre con la que se generó.

Para hacer el ajuste, se escoge la transformación que nos de la menor suma de las diferencias al cuadrado (SSD). Sin embargo, usar SSD en ambas imágenes puede dar problemas cuando se tienen cambios en escala o rotaciones, ya que esta métrica no considera estos cambios en la imagen, por lo que se optó por poner ambas imágenes en un mismo marco de referencia.

Se evito el tener que hacer la transformación de toda la imagen, por lo que se realizan interpolaciones bilineales para aproximar únicamente los puntos que se van requeriendo, tomando en cuenta estas transformaciones. De esta manera, se aproxima a tener los puntos de las imágenes en un mismo marco de referencia, por lo que el algoritmo de SSD se vuelve más efectivo en este caso.

3.2.1 *Bundle Adjustment*

En la literatura se observa muchos estudios sobre *Bundle Adjustment* como un último paso para una reconstrucción tridimensional, algunos trabajos [15] mencionan los ajustes que se tiene que hacer para utilizarlo en la reconstrucción de un mosaico de imágenes, tomando en cuenta una cámara estacionaria que solamente hace rotaciones o para la reconstrucción de un mosaico de imágenes para una cámara que toma muestras de un plano.

Para el presente trabajo se toma en consideración que se trabaja con imágenes de un plano, ya que, la cámara siempre se mantiene a una altura considerable del suelo, con el lente siempre en dirección a

la tierra, por lo que no se presentan cambios muy grandes y a lo más se tienen transformaciones afines.

El valor que se quiere minimizar es el error de re-proyección, el cual indica que tan alejado se encuentrán nuestras estimaciones con las reales, para esto definimos una ecuación para cada par de correspondencias que mapean mediante una homografía de una a otra imagen:

$$E = \begin{bmatrix} \vdots \\ (P_{j_{k_1}'} - H_{i,j} P_{i_{k_1}})^2 \\ (P_{j_{k_2}'} - H_{i,j} P_{i_{k_2}})^2 \\ \vdots \\ (P_{j_{k_n}'} - H_{i,j} P_{i_{k_n}})^2 \\ \vdots \end{bmatrix}. \tag{24}$$

En donde P_k y $P_{k'}$ son la representación en coordenadas homogéneas de dos puntos que se encuentran en las imágenes I_i e I_j respectivamente, además, $H_{i,j}$ es la relación planar que existe entre I_i e I_j y n es el número de correspondencias que existen entre I_i e I_j.

De manera general y considerando que las coordenadas homogéneas ya se encuentran normalizadas, se tiene que $z_i = z_j = h_{33} = 1$, lo cual nos simplifica la ecuación a:

$$\begin{aligned} E_{P_i,P_j} &= (P_i - H_{i,j} P_j)^2 \\ &= \begin{bmatrix} x_j \\ y_j \\ 1 \end{bmatrix} - \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}. \end{aligned} \tag{25}$$

En donde P_i es un punto en una imagen en I_i que tiene relación con un punto P_j en la imagen I_j. Obteniendo la representación de

los puntos en coordenadas cartesianas y haciendo la suma de los componentes se tiene la siguiente formulación:

$$E_{P_i,P_j} = \left(x_j - \frac{x_i h_{11} + y_i h_{12} + h_{13}}{x_i h_{31} + y_i h_{32} + 1} \right)^2 + \left(y_j - \frac{x_i h_{21} + y_i h_{22} + h_{23}}{x_i h_{31} + y_i h_{32} + 1} \right)^2 . \tag{26}$$

Los parámetros involucrados en estas ecuaciones son las relaciones planares que existen entre las imágenes (X_h) y cada uno de los puntos de interés en las imágenes (X_p).

$$\overrightarrow{X} = \begin{bmatrix} \overrightarrow{X_h} \\ \overrightarrow{X_p} \end{bmatrix} . \tag{27}$$

La cardinalidad de $\overrightarrow{X}$ es M y el número de correspondencias encontradas es N, por lo que, para hacer el planteamiento de *Bundle Adjustment* de de este problema, se tiene que encontrar el jacobiano de manera analítica de la Ecuación 24 con respecto a $\overrightarrow{X}$.

$$\frac{\partial E}{\partial \overrightarrow{X}} = \begin{bmatrix} \frac{\partial E_1}{\partial \overrightarrow{X}_1} & \cdots & \frac{\partial E_1}{\partial \overrightarrow{X}_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial E_M}{\partial \overrightarrow{X}_1} & \cdots & \frac{\partial E_M}{\partial \overrightarrow{X}_N} \end{bmatrix} . \tag{28}$$

Esta formulación da a lugar a una matríz dispersa, ya que, los únicos parámetros involucrados en el error de re-proyección de cada correspondencia son el punto de la primera imagen, el punto de la segunda imagen y la relación planar entre la primera imagen y la segunda imagen.

Bajo este conocimiento, se sabe que un punto tiene dos componentes (el componente en el eje x y el componente en el eje y), y cada relación planar puede ser representado con 8 componentes, por lo que se tiene que en cada fila del jacobiano existen a lo más doce com-

ponentes que pueden ser distintos a cero, dejando una matríz con un *grado de dispersión*[1] igual a:

$$\frac{12 * N_c}{(8 * N_h + 2 * N_p) * N_c}. \tag{29}$$

En donde N_c es el número de correspondencias encontradas, N_h es el número de relaciones entre imágenes conocidas y N_p es la suma del número de puntos en las imágenes.

Con esta información se utilizo la biblioteca **libdogleg**[2] como marco de trabajo, esta biblioteca contiene una implementación de *Powell's dog leg* para la solución de problemas de optimización de mínimos cuadrados no lineales a gran escala. El único requisito es implementar una función que realice la evaluación de la función de error E y el jacobiano de la función del error E con respecto a cada parámetro. Los resultados se deben almacenar en memoria utilizando el formato Almacenamiento comprimido por columnas para la matriz dispersa.

3.2.2 *Construcción del jacobiano para el error de reproyección*

Para la construcción del jacobiano para el error de reproyección, se evaluó de manera analítica la Ecuación 26, y con el conocimiento de que cada fila de este jacobiano tiene solamente 12 elementos distintos de cero, se hizo un algoritmo para hacer el cálculo del jacobiano evaluando solo los elementos que intervienen en cada fila.

Para facilitar esto, se ordenaron los parámetros como se muestra en la Ecuación 27, poniendo primero los parámetros de las homografías, y posterior a esto los parámetros de cada esquina, ordenados por la imagen en la que se encuentra. Además de esto, se cuenta con una estructura de datos que nos ayuda a determinar cuales son las correspondencias y la homografía que le corresponde a cada relación.

1 Como se menciona en el Capítulo 2, este valor nos indica que tan dispersa es una matriz.

2 https://github.com/dkogan/libdogleg

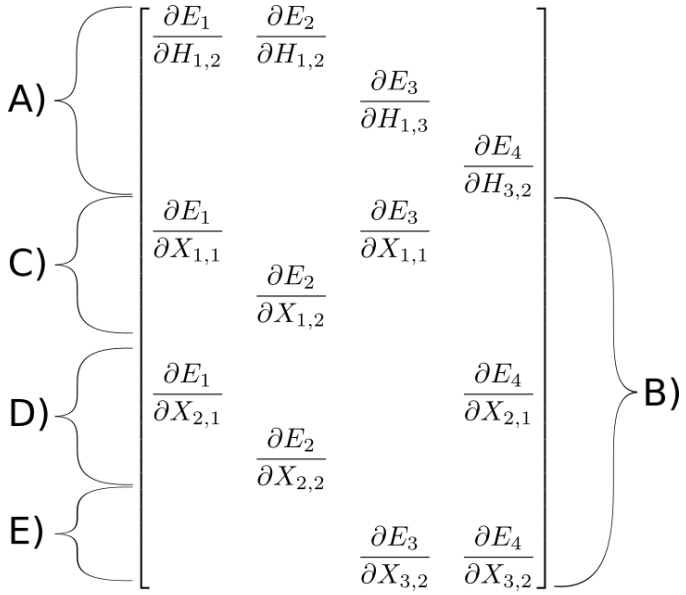

Figura 19: Ejemplo de una matríz jacobiana del error de 32×4, se tienen 3 imágenes (1,2,3) y tres relaciones ($H_{1,2}$, $H_{1,3}$, $H_{3,2}$) con cuatro correspondencias (Dos correspondencias en la primera relación, una para la segunda relación y una para la tercera relación). Los parámetros están organizados de tal manera que tenemos: A) Transformaciones planares B) Puntos en las imágenes. Estos mismos puntos los agrupamos para facilitar el indexado: C) Puntos de la imagen 1 D) Puntos de la imagen 2 E) Puntos de la imagen 3.

Juntando toda esta información, nos permite hacer el cálculo de la matriz jacobiana de la función de error, utilizada en la biblioteca **libdogleg**, permitiendo realizar la optimización de los parametros.

3.3 OPTIMIZACIONES REALIZADAS EN EL PROCESADOR VECTORIAL *neon*

A lo largo del desarrollo, se realizaron mediciones del tiempo promedio que tomaba la ejecución de ciertos algoritmos, revisando los que tomaran más tiempo, se fue decidiendo diseñar versiones optimizadas para el procesador vectorial *Neon* que pudieran dar un mejor rendimiendo en tiempo de ejecución.

3.3.1 *Refinamiento de la transformación planar*

De los algoritmos que se implementaron, se generaron versiones optimizadas para el procesador vectorial *Neon*, de partes que resultaban computacionalmente costosas, tal es el caso del calculo del SSD e interpolaciones bilineales utilizados en la sección 3.2.

Se diseño un algoritmo que tomará como entrada ambas imágenes; la relación planar que existe entre la primera imagen y la segunda imagen; y los puntos esquina de la primera.

El algoritmo extrae los valores de las imágenes, realiza las transformaciones de los puntos y una vecindad de 7×7 e interpola de manera bilineal el valor esperado en la segunda imagen. El algoritmo retorna con los resultados de cada evaluación del algoritmo SSD para cada punto esquina y su vecindad. Con estos valores, se puede decidir que relación planar es la mejor.

3.3.2 *Bundle Adjustment y libdogleg*

Se realiza un análisis de los algoritimos utilizados en la biblioteca **libdogleg** para determinar que secciones se podrían optimizar con el procesador vectorial *Neon*.

Se encontró que se realizan muchas operaciones vectoriales, las cuales son candidatos para la optimización en el procesador vectorial *Neon*. Se diseño un algoritmo para las siguientes operaciones vectoriales que se realizaban en **libdogleg**.

- Norma-l^2.
- Producto punto.
- Vector múltiplicado por un escalar.
- Suma de vectores.
- Resta de vectores.

- Vector multiplicado por -1.
- Norma-l^2 del producto de una matriz dispersa por un vector.

Para el último caso, se diseñaron algoritmos que trabajan específicamente para el planteamiento presentado de *Bundle Adjustment*, ya que, resultaba más óptimo que una rutina de propósito general. Para esto se tomá en consideración que cada columna tiene solamente doce elementos que pueden ser distintos de cero.

4

EXPERIMENTOS

En este Capítulo se define el hardware en el que los experimentos fueron ejecutados, su memoria y el sistema operativo que se utilizó.

También se describen los distintos experimentos que se realizaron para evaluar el procesador vectorial *Neon*, algunas técnicas utilizadas para sacar mayor provecho a estas instrucciones; tales como técnicas de optimización que se usan regularmente en el lenguaje *C* y algunos métodos especificos para la escritura de código en el procesador vectorial *Neon*. Estos experimentos sirven de base para el diseño y codificación del algoritmo de *Bundle Adjustment* en el procesador vectorial *Neon*.

Posterior a eso, se explican los experimentos realizados con la implementación de *Bundle Adjustment* en el procesador vectorial *Neon*, así como los resultados obtenidos con varios casos que se plantearon al momento de realizar una optimización de homografías y puntos de interes entre varias imágenes.

4.1 CARACTERÍSTICAS DEL HARDWARE

Todas las pruebas realizadas se hicieron en un *Beaglebone Black (Rev b)* ensamblado por *element14*, el cual cuenta con las siguientes especificaciones:

- Procesador AM335x 1GHz ARM®Cortex-A8.
- 512 MB DDR3 RAM.
- Memoria de almacenamiento embebida de 2GB 8-bit eMMC.
- Acelerador de punto flotante *Neon* con 32 registros de 64bits.

- Sistema operativo *Debian Wheezy*

4.2 OPTIMIZACIONES GENERALES

A lo largo de la búsqueda de información de como codificar para esta arquitectura, se fueron encontrando técnicas para generar un código que tuviera un mejor desempeño en esta arquitectura, sobre todo al utilizar la vectorización.

A lo largo de esta sección se muestran las que fueron consideradas más importantes al momento de diseñar el algoritmo, se realiza una comparación entre la ejecución de un código utilizando el procesador vectorial *Neon* y un código sin esto.

4.2.1 *Vectorización básica*

Se evaluaron las capacidades del hardware, en especial del procesador vectorial *Neon*, para cuantificar el desempeño, se realizó la siguiente operación de números flotantes, $c = a * b$.

Para preparar la prueba, se generó una lista de 10,000 pares de números flotantes de manera pseudo-aleatoria, los cuales se guardan en memoria. Luego se procede a probar dos casos:

1. Se realiza la multiplicación de ambos pares de números flotantes y almacenamiento en una tercera variable; sin utilizar el procesador vectorial *Neon*.

2. Se realiza la multiplicación de ambos pares de números flotantes y almacenamiento en una tercera variable; utilizando el procesador vectorial *Neon*.

Para estos dos casos, se contabiliza el tiempo en el que tardaban en ejecutarse las operaciones, posterior a la ejecución, se verifica que las operaciones sean correctas para ambos casos.

Los resultados obtenidos fueron los siguientes:

Algoritmo 1: Multiplicación de 10,000 pares de números sin utilizar el procesador vectorial *Neon*

```
1 Inicializacion de los datos;
2 for i<10,000 do
3 |   c[i] = a[i] * b[i];
4 |   i = i + 1 ;
5 end
```

Algoritmo 2: Multiplicación de 10,000 pares de números utilizando el procesador vectorial *Neon*

```
1 Inicialización de los datos;
2 for i<10,000 do
3 |   Cargar a[i] al registro vectorial q0 ;
4 |   Cargar b[i] al registro vectorial q1 ;
5 |   q2 = q0 * q1 ;
6 |   Almacena q2 en c[i] ;
  |   /* El procesador vectorial Neon puede procesar los
  |      flotantes de cuatro en cuatro.                */
7 |   i = i + 4 ;
8 end
```

Figura 20: Algoritmos utilizados para la comparación del rendimiento del procesador vectorial *Neon*, **a** y **b** son arreglos con los factores de la multiplicación y **c** es un arreglo en el que se almacenan los resultados de la multiplicación.

Descripción	Tiempo de ejecución(ms)
CPU	8.35
Neon	3.15

Cuadro 1: Resultados del tiempo de ejecución de las implementaciones de la multiplicación de dos factores.

Se observa un desempeño de un 62 % de mejora en el tiempo requerido para ejecutar las 10,000 multiplicaciones y almacenar los resultados.

4.2.2 *Desenroscado de ciclos*

El desenroscado de ciclos (Del ingles, *Loop unrolling*) es una técnica que trata de disminuir la ejecución de código de control, tales como son la aritmética de apuntadores y pruebas para determinar si ya se llegó al final del ciclo. Esta técnica es válida también en el procesador vectorial *Neon*, y usada apropiadamente puede aportar un mejor desempeño.

Se continua usando el mismo escenario, de entrada tenemos 10,000 pares de números flotantes, la operación a ejecutar es $c = a * b$ y la salida son 10,000 números flotantes, el resultado de la operación.

1. Se realiza la multiplicación de ambos pares de números flotantes y almacenamiento en una tercera variable; haciendo desenroscado de operaciones y sin utilizar el procesador vectorial *Neon*.

2. Se realiza la multiplicación de ambos pares de números flotantes y almacenamiento en una tercera variable; haciendo desenroscado de operaciones y utilizando el procesador vectorial *Neon*.

Los resultados obtenidos fueron los siguientes:

Algoritmo 3: Multiplicación de 10,000 pares de números sin utilizar el procesador vectorial *Neon*, utilizando el desenroscado de ciclos.

```
1 Inicialización de los datos;
2 for i<10,000 do
3     c[i] = a[i] * b[i];
4     c[i+1] = a[i+1] * b[i+1];
5     ⋮
6     c[i+4] = a[i+4] * b[i+4];
7     i = i + 5 ;
8 end
```

Algoritmo 4: Multiplicación de 10,000 pares de números utilizando el procesador vectorial *Neon*, utilizando el desenroscado de ciclos.

```
1 Inicialización de los datos;
2 for i<10,000 do
3     Cargar a[i] al registro vectorial q0 ;
4     Cargar b[i] al registro vectorial q1 ;
5     Cargar a[i+4*1] al registro vectorial q2 ;
6     Cargar b[i+4*1] al registro vectorial q3 ;
7     ⋮
8     Cargar a[i+4*4] al registro vectorial q8 ;
9     Cargar b[i+4*4] al registro vectorial q9 ;
10    q10 = q0 * q1 ;
11    q11 = q2 * q3 ;
12    ⋮
13    q14 = q8 * q9 ;
14    Almacena q10 en c[i] ;
15    Almacena q11 en c[i+4*1] ;
16    ⋮
17    Almacena q14 en c[i+4*4] ;
18    i = i + 4*5 ;
19 end
```

Figura 21: Algoritmos que utilizan el desenroscado de ciclos, son utilizados para la comparación del rendimiento del procesador vectorial *Neon*, **a** y **b** son arreglos con los factores de la multiplicación y **c** es un arreglo en el que se almacenan los resultados de la multiplicación.

Descripción	Tiempo de ejecución (ms)
CPU con desenroscado de ciclos	6.76
Neon con desenroscado de ciclos	3.10

Cuadro 2: Resultados del tiempo de ejecución de las implementaciones de la multiplicación de dos factores, utilizando el desenroscado de ciclos.

Se observa un desempeño de un 54 % de mejora en el tiempo requerido para ejecutar las 10,000 multiplicaciones y almacenar los resultados.

4.2.3 *Otros factores*

Adicionalmente, al mandar a ejecutar una instrucción en el *Neon*, ésta no se completa de manera inmediata, puede tomar más de un ciclo en estar lista, mientras tanto, se bloquean los registros que están involucrados en la operación. Esto se conoce como parada (del ingles, *stall*), y puede ser muy perjudicial para el desempeño de un algoritmo.

Por ejemplo, la instrucción para cargar a un registro del *Neon*, no se completa en un ciclo, por lo que no se recomienda operar ese registro inmediatamente después de mandar a cargarlo. Esto se vuelve un trabajo en el que se tienen que considerar las operaciones que se requieran realizar, sopesar el número de registros que tenemos disponibles para operar y el orden de ejecución requerido de las instrucciones.

Se conoce la siguiente herramienta en la que uno se puede apoyar para diseñar algoritmos en Neon, para verificar que no existan paradas y el desempeño del algoritmo sea el óptimo[1], en esta herramienta se escriben las instrucciones en ensamblador e indica cuandos ciclos se toma en ejecutarse, además de que muestra en donde ocurren paradas y cuanto tiempo se toma en ellas. Con esta información, uno puede ir adecuando el algoritmo para reducir el número de ciclos requeridos para su ejecución.

1 http://pulsar.webshaker.net/ccc/result.php?lng=us

4.2.4 *Algoritmo de interpolación bilineal en el procesador vectorial Neon*

Se hicieron unas pruebas finales, con el algoritmo de la interpolación bilineal (22), se realizaron dos implementaciones, una sin utilizar el procesador vectorial *Neon* y la otra utilizando el procesador vectorial *Neon*, para esto se generan 10,000 entradas para hacer la prueba de la interpolación, cada entrada consta de seis números de punto flotante (cuatro esquinas del cuadrado unitario y dos coordenadas que representan el punto que se desea obtener) y una salida de un número de punto flotante, que contiene el valor de la interpolación bilineal.

Los resultados se muestran en la siguiente tabla:

Interpolación bilineal	
Descripción	Tiempo de ejecución(ms)
Sin utilizar *Neon*	2.74
Utiizando *Neon* con optimizaciones	0.36

Cuadro 3: Resultados del tiempo de ejecución de las implementaciones de la interpolación bilineal.

Se observa un rendimiento del 86 % en la disminución del tiempo de ejecución.

Esto demuestra que utilizar el procesador vectorial *Neon* lográ disminuir el tiempo de ejecución de los algoritmos que puedan ser correctamente vectorizados.

4.3 IMÁGENES DE LA RETINA

Dada las condiciones meteorológicas que imperaban en la localidad no permitieron hacer experimentos utilizando un papalote, se optó por hacer la parte experimental en otro tipo de imágenes, se eligieron un conjunto de imágenes de la retina del ojo humano, dada la similitud que presentan éstas para la formación de mosaico de imágenes.

Se escogieron cuatro imágenes que visualmente guardarán relación entre si; para cada una de ellas se obtuvieron los puntos característicos que contenían utilizando el algoritmo de *Harris*, este algoritmo se utiliza para la búsqueda de esquinas en una imagen. Fue utilizado en trabajos anteriores por Tolosa [19], quien realizó una implementación eficiente de este algoritmo, utilizando el procesador vectorial *Neon*

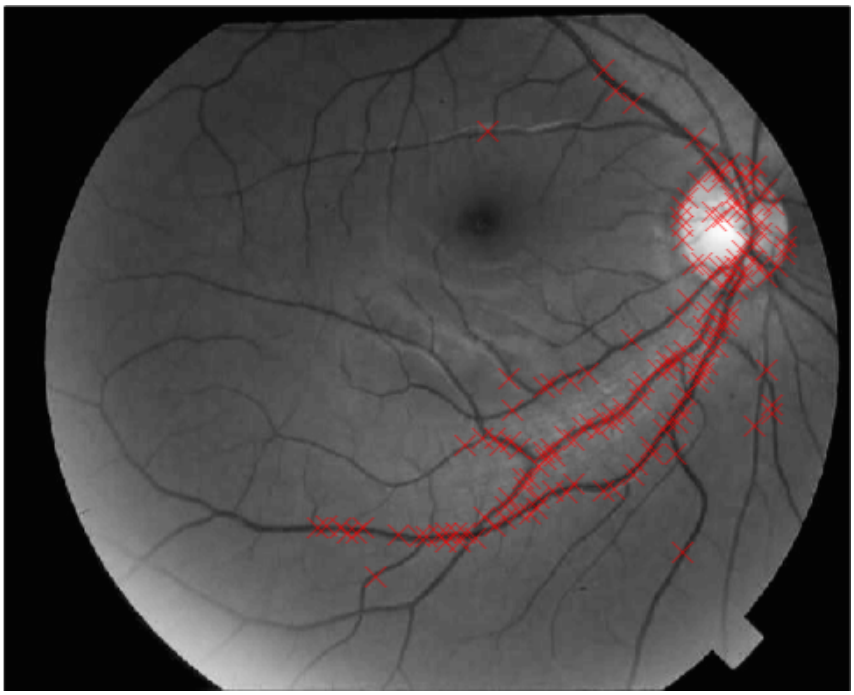

Figura 22: Imagen de una retina con las esquinas hallados por medio del algoritmo de Harris.

Se escogieron pares entre estas imágenes para efectos de probar el algoritmo, el criterio que se utilizó para escoger estos pares fue generar un camino que pase por todas las imágenes. Se escogieron por simplicidad los pares (1,2), (2,3), (3,4).

Para cada par de imágenes, se hizo una búsqueda de correspondencias, para esto se empleó el algoritmo *ZNCC* [4], posteriormente se utilizó el algoritmo *RANSAC* [20] para eliminar los puntos atípicos.

Los puntos resultantes fueron utilizados para el cálculo de la homografía que mejor representa esas correspondencias, el método que se utilizó para hacerlo fue planteando un sistema sobredeterminado, utilizando la Ecuación (3) y las correspondencias. Este sistema se resuelve utilizando un método de mínimos cuadrados.

Ya que estas imágenes se capturan de un ojo, el cual es una esfera, es de esperarse que estas imágenes tengan una cierta distorsión geométrica, la cual se puede reducir estimando el parámetro $\hat{p}$ de la siguiente Ecuación[8]:

$$\hat{p} = p(1 + \sum_{n=1}^{\infty} k_n^{2n} r^2). \tag{30}$$

En donde p es la coordenada cartesiana del punto en la imagen distorsionada, k_n son los parámetros de la distorsión y r es la distancia del centro de la imagen al punto p y $\hat{p}$ es el punto estimado sin la distorsión.

Dado que k_1 aporta el 95 % [8] del error, se decide solo utilizar este parámetro, simplificando la ecuación a:

$$\hat{p} = p(1 + k_1^2 r^2). \tag{31}$$

Para encontrar el parámetro k de cada imagen, usamos la ecuación (3) y (31) para obtener:

$$p_2(1 + k_1^{(2)} r_2^2) = H_{1,2} p_1 (1 + k_1^{(1)} r_1^2). \tag{32}$$

En donde p_i representa un punto en la imagen i, $k_1^{(i)}$ es la k_1 para la i-ésima imagen y r_i es la distancia del punto p_i al centro de la imagen i.

Con el listado de los pares de correspondencias entre pares de imágenes, y la ecuación (32) se forma un sistema de ecuaciones sobre determinado, con el cual se pueden aproximar las $k_1^{(i)}$ utilizando un método de mínimos cuadrados.

Una vez que se cuenta con la aproximación del parámetro de distorsión geométrica, se utiliza para actualizar todos los puntos de las correspondencias en las cuatro imágenes, posterior a esto, se vuelve a repetir el proceso de utilizar *RANSAC* sobre estas correspondencias con los puntos actualizados y posterior a esto se realiza nuevamente el cálculo de la homografía.

Dado que la única variación es la ejecución del algoritmo *Bundle Adjustment*, el valor del parámetro de distorción encontrado es siempre el mismo para todos los casos, $r_1^{I_1} = -1.37e^{-7}$, $r_1^{I_2} = -1.07e^{-7}$,

$r_1^{I_3} = -1.20e^{-7}$, $r_1^{I_4} = -1.93e^{-7}$. Se observa que es una distorción muy pequeña, aparentemente las imágenes utilizadas no contaban con una distorción geometrica.

Antes de proceder a la realización de las pruebas, se quizo determinar que tan sensible es el algoritmo al ruido.

Para generar un ejemplo del ruido que pudiera estar presente en los datos, se consideró que en pasos anteriores no se pudieran encontrar suficientes esquinas. Se simula este escenario generando varios sub-conjuntos del universo de esquinas. Los sub-conjuntos generados contienen k elementos, con $k \in (4, 6, 8, 10, 12, 14, 16, 18)$.

Posteriormente se estima la homografía para cada sub-conjunto de esquinas. Se comparan los sub-conjuntos con la misma k y se obtiene la desviación estandar $\overrightarrow{\sigma_k}$ para cada parámetro de la homografía. Esto nos permitió observar que la desviación estandar era notoriamente más grande cuando $k = 4$, el error crece notoriamente en comparación de utilizar el conjunto completo de esquinas. También se observa que la varianza era muy pequeña cuando $k = 8$ y la diferencia entre la desviación estandar al crecer k no era muy notoria.

Con esta información, se determina utilizar la desviación estandar $\overrightarrow{\sigma_4}$ y $\overrightarrow{\sigma_8}$ para simular los casos en los que existen pocas esquinas presentes en las relaciones.

Considerando los puntos característicos, los cuales fueron modificados utilizando el factor de distorción encontrado; las homografías ajustadas con los puntos característicos mencionados; las correspondencias entre los pares de imágenes y los valores que usamos para simular ruido $\overrightarrow{\sigma_4}$ y $\overrightarrow{\sigma_8}$ procedemos a hacer pruebas utilizando el algoritmo *Bundle Adjustment* para la optimización de las homografías y los puntos característicos.

A continuación se definen dos clasificaciones de los experimentos, con el fin de facilitar la presentación de los mismos.

POR OPTIMIZACIÓN. Dependiendo los parámetros afectados por la optimización, lo podemos clasificar en:

T. Se utiliza para indicar que la optimización se realiza sobre todos los parámetros.

H. Indica que la optimización solo afectará los parámetros de la homografía.

D. Indica que la optimización se hará en dos pasos, primero afectando solo la homografía, posterior a esto optimizando todos los parámetros.

POR RUIDO AGREGADO. Con el fin de observar la manera en la que el algoritmo de *Bundle Adjustment* actua cuando existe ruido en las entradas o cuando el error inicial es mayor, se introduce un ruido artificial a las homografías, previas a la ejecución del algoritmo de *Bundle Adjustment*

o. No se le agrega ningún ruido artificial a las homografías.

cσ_k. Se agrega a las homografías, el ruido $C \times \overrightarrow{\sigma_k}$, donde C es una constante y $k \in (4, 8)$.

Por ejemplo, el caso $H3\sigma_4$ indica que se utilizó el algoritmo de *Bundle Adjustment* solo para optimizar los parámetros de la homografía, agregando un ruido de tres veces $\overrightarrow{\sigma_4}$ a los parámetros de las homografías.

A continuación se muestran los resultados al utilizar el algoritmo de *Bundle Adjustment* con los parámetros

Caso T_0								
Error inicial					951.690			
Error final					0			
Número de iteraciones					28			
Tiempo de ejecución(ms)			Sin Neon		244			
			Con Neon		198			
$H_{0,1}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	−171.568	−105.848	−0.004	1.203	1.276	1.185	0	0
Finales	−171.581	−105.790	−0.006	1.216	1.276	1.185	0	0
Distancia	0.013	0.058	0.002	0.013	0	0	0	0
$H_{1,2}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	0.373	53.322	−0.017	1.019	0.978	0.948	0	0
Finales	0.449	53.205	−0.017	1.011	0.978	0.948	0	0
Distancia	0.076	0.117	0	0.008	0	0	0	0
$H_{2,3}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	166.520	86.029	0.007	0.381	0.754	0.723	0	0
Finales	166.547	86.019	0.006	0.396	0.753	0.724	0	0
Distancia	0.027	0.01	0.001	0.015	0.001	0.001	0	0

Figura 23: Se muestran los resultados del caso T_0, mostrando los errores inicial y final; el número de iteraciones; y la descomposición de cada parámetro, agrupado por homografía, de estos se muestra su valor inicial, el valor final y la distancia entre el valor inicial y el valor final.

Caso H_0								
Error inicial					951.690			
Error final					889.236			
Número de iteraciones					18			
Tiempo de ejecución(ms)			Sin Neon		28			
			Con Neon		24			
$H_{0,1}$	t_x	t_y	θ	ϕ	λ_x	λ_y	ν_1	ν_2
Iniciales	−171.568	−105.848	−0.004	1.203	1.276	1.185	0	0
Finales	−171.012	−104.045	−0.005	1.182	1.271	1.181	0	0
Distancia	0.556	1.803	0.001	0.021	0.005	0.004	0	0
$H_{1,2}$	t_x	t_y	θ	ϕ	λ_x	λ_y	ν_1	ν_2
Iniciales	0.373	53.322	−0.017	1.019	0.978	0.948	0	0
Finales	2.570	54.974	−0.017	0.983	0.973	0.936	0	0
Distancia	2.197	1.652	0	0.036	0.005	0.012	0	0
$H_{2,3}$	t_x	t_y	θ	ϕ	λ_x	λ_y	ν_1	ν_2
Iniciales	166.520	86.029	0.007	0.381	0.754	0.723	0	0
Finales	163.976	84.442	0.004	0.202	0.765	0.738	0	0
Distancia	2.544	1.587	0.003	0.178	0.011	0.015	0	0

Figura 24: Se muestran los resultados del caso H_0, mostrando los errores inicial y final; el número de iteraciones; y la descomposición de cada parámetro, agrupado por homografía, de estos se muestra su valor inicial, el valor óptimo y la distancia entre el valor inicial y el valor óptimo.

Caso D_0								
Error inicial					951.690			
Error final					0			
Número de iteraciones					$18 + 29$			
Tiempo de ejecución(ms)			Sin Neon		257			
			Con Neon		237			
$H_{0,1}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	−171.568	−105.848	−0.004	1.203	1.276	1.185	0	0
Finales	−170.990	−104.090	−0.007	1.207	1.274	1.181	0	0
Distancia	0.578	1.758	0.003	0.004	0.002	0.004	0	0
$H_{1,2}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	0.373	53.322	−0.017	1.019	0.978	0.948	0	0
Finales	2.538	54.564	−0.016	0.988	0.973	0.938	0	0
Distancia	2.165	1.242	0.001	0.031	0.005	0.010	0	0
$H_{2,3}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	166.520	86.029	0.007	0.381	0.754	0.723	0	0
Finales	164.022	84.440	0.004	0.213	0.763	0.737	0	0
Distancia	2.498	1.589	0.003	0.168	0.009	0.014	0	0

Figura 25: Se muestran los resultados del caso D_0, mostrando los errores inicial y final; el número de iteraciones; y la descomposición de cada parámetro, agrupado por homografía, de estos se muestra su valor inicial, el valor óptimo y la distancia entre el valor inicial y el valor óptimo.

Caso H_b2_3std								
Error inicial					6,109M			
Error final					896			
Número de iteraciones					38			
Tiempo de ejecución(ms)			Sin Neon		44			
			Con Neon		33			
$H_{0,1}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	−108.618	−73.245	0.068	0.725	1.499	1.419	0	0
Finales	−179.191	−109.706	−0.007	1.237	1.293	1.212	0	0
Distancia	70.573	36.461	0.075	0.512	0.205	0.207	0	0
$H_{1,2}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	63.323	85.926	0.055	0.541	1.262	1.122	0	0
Finales	4.753	56.750	−0.017	0.948	0.968	0.925	0	0
Distancia	58.570	29.176	0.072	0.407	0.293	0.196	0	0
$H_{2,3}$	t_x	t_y	θ	ϕ	λ_x	λ_y	v_1	v_2
Iniciales	229.470	118.633	0.080	1.474	0.897	1.036	0	0
Finales	157.866	81.281	−0.003	1.341	0.765	0.798	0	0
Distancia	71.604	37.352	0.083	0.133	0.132	0.239	0	0.0

Figura 26: Se muestran los resultados del caso H_b2_3std, mostrando los errores inicial y final; el número de iteraciones; y la descomposición de cada parámetro, agrupado por homografía, de estos se muestra su valor inicial, el valor óptimo y la distancia entre el valor inicial y el valor óptimo.

Tomando en consideración estos datos se puede concluir que:

- Cuando las entradas se encuentran cerca del punto óptimo, el realizar la optimización con todos los parámetros (Parámetros de la homografía y puntos en las imágenes) hace que los cambios a la homografía sean pequeñas.
- Cuando las entradas se encuentran más alejadas del punto óptimo, el utilizar todos los parámetros evita que se llegue a un punto de convergencia.
- El hacer que la optimización solamente afecte los parámetros de la homografía da buenos resultados, disminuyendo el error, aún cuando el error sea considerable, además, se reduce el tiempo de ejecución del algoritmo.
- El hacer una fase de optimización primero solamente para los parámetros de la homografía y posterior a esto para todos los puntos, hace que los puntos se ajusten a esta homografía que se optimizó en el primer paso.

5

CONCLUSIÓN

De acuerdo a los resultados obtenidos, se demuestra que utilizando el procesador vectorial *Neon* de los procesadores ARM es posible acelerar el tiempo de ejecución del algoritmo de *Bundle Adjustment*. Esto permite realizar una mejor aproximación a las homografías y ajustar los puntos carácteristos entre los pares de imágenes. Estos resultados pueden ser utilizados para realizar ajustes de manera periodica sobre los datos, con la finalidad de mantenerlos con el mínimo error posible.

5.1 CONTRIBUCIONES DEL PROYECTO

La contribución de este proyecto radica en proporcionar herramientas para realizar ajustes de los parámetros de manera conjunta, con la finalidad de reducir el error acarreado en las diversas etapas de este cómputo.

El algoritmo proporcionado permite actualizar la información para ajustarla después de cada cierto tiempo, de esta manera se puede reducir al mínimo errores de acarreo. La información proporcionada podría servir como un sensor adicional para realizar los ajustes necesarios a los puntos previamente capturados, así en futuras fases se usaría información más exacta para el cómputo.

Se probó su utilidad en imágenes médicas (imágenes de la retina) para calcular los parámetros que relacionan estas imágenes entre si.

5.2 TRABAJO FUTURO

Si bien se probaron los algoritmos en imágenes médicas (imágenes de la retina), sería deseable hacer la integración al flujo de trabajo que se tiene establecido actualmente para la línea de investigación que se desarrolla en la Facultad de Matemáticas de la Universidad Autónoma de Yucatán para la generación de mosaicos de imágenes aéreas y su análisis.

Igual se propone la realización de pruebas de campo, las cuales consisten en elevar físicamente un papalote con todo el sistema de captura montado en él para observar su desempeño en un ambiente real.

Durante la realización del proyecto no se contaba con un procesador *ARM* que tuviera la capacidad de manejar números flotates de doble presición en su procesador vectorial. Es importante la implementación de este algoritmo en un procesador con las características mencionadas, ya que, además de mejorar la precisión, muchas de las implementaciones que sirven de apoyo para el desarrollo de este algoritmo hacen uso de este tipo de dato.

Para mejorar el desempeño de estos algoritmos, se hace uso de bibliotecas externas que implementan operaciones para matrices dispersas. Se propone revisar estas bibliotecas y generar una implementación para el procesador vectorial *Neon* de las partes que más tiempo toman en el procesador.

BIBLIOGRAFÍA

[1] Padmanabhan Anandan. Measuring visual motion from image sequences. Technical report, Amherst, MA, USA, 1987.

[2] A. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996. ISBN 9780898713602. URL https://books.google.com.mx/books?id=myzIPBwyBbcC.

[3] David Capel. *Image Mosaicing and Super-resolution*. Springer London, London, 2004. ISBN 978-1-4471-1049-1. doi: 10.1007/978-0-85729-384-8. URL http://www.springerlink.com/index/10.1007/978-0-85729-384-8.

[4] Luigi Di Stefano, Stefano Mattoccia, and Federico Tombari. ZNCC-based template matching using bounded partial correlation. *Pattern Recognition Letters*, 26(14):2129–2134, October 2005. ISSN 01678655. doi: 10.1016/j.patrec.2005.03.022. URL http://linkinghub.elsevier.com/retrieve/pii/S0167865505000905.

[5] Michael J. Flynn. Some Computer Organizations and Their Effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960, September 1972. ISSN 0018-9340. doi: 10.1109/TC.1972.5009071. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5009071.

[6] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[7] Kurt Konolige. Sparse sparse bundle adjustment. In Frédéric Labrosse, Reyer Zwiggelaar, Yonghuai Liu, and Bernie Tiddeman, editors, *BMVC*, pages 1–11. British Machine Vision Association, 2010. ISBN 1-901725-40-5. URL http://dblp.uni-trier.de/db/conf/bmvc/bmvc2010.html#Konolige10.

[8] Sangyeol Lee, Michael D. Abràmoff, and Joseph M. Reinhardt. Feature-based pairwise retinal image registration by radial distortion correction. In *Proc. SPIE 6512, Medical Imaging 2007: Image Processing*, volume 6512, pages 1–10. SPIE, 2007. doi: doi:10.1117/12.710676.

[9] Lizbeth Maldonado López. Estimación robusta de pose de un vehículo aéreo no tripulado. Master's thesis, Facultad de Matemáticas, Universidad Autónoma de Yucatán, January 2014.

[10] M. I A Lourakis and AA Argyros. Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1526–1531 Vol. 2, Oct 2005. doi: 10.1109/ICCV.2005.128.

[11] Manolis Lourakis and Antonis Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. Technical report, Technical Report 340, Institute of Computer Science-FORTH, Heraklion, Crete, Greece, 2004.

[12] Manolis I. a. Lourakis and Antonis a. Argyros. Sba. *ACM Transactions on Mathematical Software*, 36(1):1–30, March 2009. ISSN 00983500. doi: 10.1145/1486525.1486527. URL http://portal.acm.org/citation.cfm?doid=1486525.1486527.

[13] Irving Sánchez Machay. 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, January 2014.

[14] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems (2nd ed.), 2004. URL http://f.

[15] Philip F. Mclauchlan, Allan Jaenicke, and Guildford Gu Xh. Image mosaicing using sequential bundle adjustment. In *In Proc. BMVC*, pages 751–759, 2000.

[16] Jorge J. Moré. The Levenberg-Marquardt algorithm: Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116. Springer, Berlin, 1977.

[17] Michael JD Powell. A hybrid method for nonlinear equations. *Numerical methods for nonlinear algebraic equations*, 7:87–114, 1970.

[18] Ji Shunping, Shi Yun, and Shi Zhongchao. Bundle adjustment with vehicle-based panoramic imagery. In *Earth Observation and Remote Sensing Applications (EORSA), 2012 Second International Workshop on*, pages 106–110, June 2012. doi: 10.1109/EORSA.2012.6261145.

[19] Irving Tolosa. Detector de esquinas en tiempo real implementado en arquitectura simd para sistema autónomo de captura de imágenes aéreas. Master's thesis, Facultad de Matemáticas, Universidad Autónoma de Yucatán, January 2014.

[20] P. H. S. Torr and D. W. Murray. Outlier detection and motion segmentation. pages 432–443, 1995.

[21] Bill Triggs, PhilipF. McLauchlan, RichardI. Hartley, and AndrewW. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-67973-8. doi: 10.1007/3-540-44480-7_21. URL http://dx.doi.org/10.1007/3-540-44480-7_21.