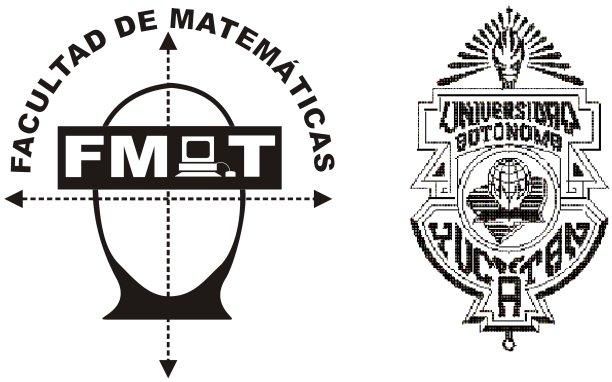
Calibración débil en paralelo a partir de una secuencia de imágenes.

LCC. Ariel Antonio Briceño Coronado.
aranbrico@gmail.com



Maestría en Ciencias Matemáticas.
Facultad de Matemáticas
Universidad Autónoma de Yucatán
2008

Declaración.

En cumplimiento de uno de los requisitos para la titulación en la Maestría en Matemáticas de la Universidad Autónoma de Yucatán, dirijo el presente documento como tesis de Maestría para obtener el grado de Maestro en Ciencias Matemáticas.

Declaro que esta tesis fue realizada enteramente por mi y describe mi propio trabajo de investigación con excepción de las partes que así se indiquen.

LCC. Ariel Antonio Briceño Coronado
Mérida, Yucatán
México
1 de Abril de 2008.

Agradecimientos.

- A Dios, por darme la vida.
- A mi familia, por ayudarme a vivir y enseñarme las cosas buenas de la vida. Gracias también por soportarme.
- A todos los compañeros de mi vida estudiantil, porque pasé gratos momentos con ellos.
- A mi asesor, el Dr. Arturo Espinosa Romero, por ayudarme con su amistad, experiencia y apoyo para el desarrollo de esta tesis. Es un amigo al que estimo mucho y siempre atento a sus enseñanzas.
- Al Dr. Ricardo Legarda Sáenz, por asesorarme en el desarrollo de esta tesis. Durante el tiempo que estuve trabajando con él, fue pasando de ser profesor a ser un amigo cuyos comentarios son siempre valorados.
- Al Dr. Luis Alberto Muñoz Ubando, porque con su amistad mi interés se enfocó al área de visión computacional.
- A los Doctores mencionados un especial agradecimiento porque con su llegada a FMAT no tuve la necesidad de ir lejos de mi lugar de origen para tener excelentes profesores y amigos que me ayudaron a conocer los principios de procesamiento de imágenes y visión computacional
- A la Facultad de Matemáticas de la Universidad Autónoma de Yucatán, porque en su laboratorio LI^2CoViR se desarrolló la mayor parte de esta tesis.
- A todas las personas que con sus recomendaciones, hicieron que la elaboración de esta tesis, se terminara satisfactoriamente.

El trabajo desarrollado en esta tesis fue apoyado por una beca para asistente de investigación del proyecto CONACYT clave "SEP-2004-C01-47893", cuyo responsable técnico es el Dr. Arturo Espinosa Romero.

Resumen.

El objetivo principal de esta tesis es conocer las ventajas del enfoque de algoritmos paralelos en visión tridimensional, que usualmente se plantean como procedimientos secuenciales. El documento presenta un estudio acerca de la paralelización de algoritmos de calibración débil de imágenes, usando una secuencia de 2 o más imágenes.

Para cada etapa en el proceso de calibración, se analiza las oportunidades de paralelización, se muestran los algoritmos secuenciales, los paralelos y se presentan también los resultados de las implementaciones así como las mejoras que se alcanzaron y proporciona una idea clara de lo que se puede lograr con un enfoque de paralelización en aprovechamiento de la computadoras actuales las cuales tienen más de dos núcleos de procesamiento.

El proceso que se describe a lo largo de este documento tiene un enfoque modular, *i.e.*, se pueden cambiar de acuerdo a necesidades, cualquier etapa en el proceso de calibración. Gracias al enfoque modular se tuvo la posibilidad de analizar cada etapa para conocer el comportamiento de los algoritmos paralelos en la arquitectura en la cual se implementaron.

Podemos resumir las fases del proceso de calibración desarrollado en este trabajo en tres etapas principales: detección de esquinas, correspondencia entre las imágenes y depuración de las correspondencias. Cada una de las etapas se describe de manera independiente, pero siempre manteniéndolas dentro del contexto del objetivo central.

Los resultados mostraron mejoras en el rendimiento con la paralelización de los algoritmos implementados y las gráficas en los capítulos desde el 3 hasta el 6 lo demuestran.

Índice general

Índice de figuras

Índice de cuadros

Lista de Algoritmos

Capítulo 1

Introducción.

Grandes esfuerzos se han llevado a cabo durante las últimas décadas para enseñar a las computadoras a ver. En la vida cotidiana del ser humano, cada vez es mas probable encontrar máquinas que realicen actividades peligrosas como por ejemplo acercarse al cráter de un volcán en erupción, desactivar una bomba o en un futuro cercano en el servicio domestico y autos que se conducen solos entre otros; hay que hacer notar que en el caso de los dos primeros ejemplos, estas máquinas son manipuladas a distancia, *i.e.*, no son autónomas, en el segundo caso se puede mencionar al robot humanoide de Honda[1]. Aunque grandes avances se han logrado con respecto a algoritmos desarrollados, aun se está considerablemente lejos de la capacidad que tienen los seres vivos de observar el mundo que los rodea y una gran diferencia es el procesamiento masivo de imágenes que los seres vivos logran llevar acabo y aunque ya se ha avanzado al tener resultados alentadores, en general, se necesita mucho tiempo de procesamiento debido a que comúnmente se hace de manera secuencial, pero gracias al avance de la tecnología ahora ya es muy fácil tener a disposición una arquitectura paralela.

Marr [1982] establece a la *visión computacional* como el proceso que produce a partir de imágenes del mundo externo —escena— una descripción que es útil para el observador y la cual está libre de información irrelevante. Es claro de esta definición, que las máquinas autónomas no son el único objetivo de la visión computacional. Construir una representación geométrica de una escena compleja a partir de una secuencia de imágenes, es uno de los clásicos problemas en visión computacional, como por ejemplo la posibilidad de adquirir representaciones tridimensionales de objetos y escenas realistas y exactas

[1] http://world.honda.com/ASIMO/

es altamente valorado por los profesionales de la herencia cultural en la construcción de modelos computacionales de monumentos y artefactos históricos, así como en industrias cuya imperfección de superficies en los objetos que fabrican deben ser localizados.

En este capítulo se describe la motivación y objetivo de esta tesis, una breve revisión acerca del trabajo relacionado en cómputo paralelo sobre visión computacional, la metodología utilizada en este trabajo, una breve descripción acerca de la computadora paralela y finalmente la descripción general del documento.

1.1. Motivación del trabajo.

Para lograr la titulación en la carrera de Licenciado en Ciencias de la Computación impartida en la Facultad de Matemáticas de la Universidad Autónoma de Yucatán, el trabajo de tesis que desarrollé tuvo que ver con reconstrucción métrica por medio de la recuperación de la geometría epipolar usando un par de imágenes, ver [Briceño Coronado, 2005], y aunque los resultados fueron personalmente inspiradores para continuar trabajando en el área de visión computacional, tuve la impresión de que los cálculos necesitados eran demandantes de tiempo y recursos computacionales. Un año más tarde, un compañero de la facultad terminó el grado de maestría en matemáticas entregando un trabajo de reconstrucción métrica usando una secuencia de imágenes, ver [Patrón P., A., 2006]. Las platicas que tuve con mi compañero acerca de visión computacional, a pesar de estar motivados por lo interesante de las tareas que se pueden lograr, incluían comentarios sobre lo demandante de recursos que puede resultar una tarea compleja. Durante mismo tiempo, el Dr. Arturo Espinosa propuso un proyecto de investigación en visión computacional en paralelo, por el cual resulté aceptado como asistente de la investigación a desarrollar. De esta manera, después de haber cursado satisfactoriamente las materias de la maestría, el trabajo acerca de procesamiento paralelo en visión computacional empezó a tomar forma.

1.2. Objetivo de la tesis.

Por muchos años se han estado desarrollando algoritmos y estructuras de datos con un modelo dominado por la arquitectura de la máquina secuencial de Von-Neumann. Esto es así debido a la simplicidad del control de flujo, consistencia de memoria y semejanza

de las primeras arquitecturas computacionales con este modelo. Visión computacional y procesamiento de imágenes no son la excepción, la gran mayoría de los enfoques en los algoritmos propuestos en la literatura, se basa en esta arquitectura secuencial y esto es comprensible ya que es la manera usual de elaborar algoritmos.

Muchas tareas de visión computacional son muy complejas e intensivas y requerimientos de tiempo real agravan la situación. Bennett [1994] define a un sistema de tiempo real como aquel en el que la correcta operación del mismo depende tanto de los resultados lógicos de cómputo como del tiempo en el cual los resultados son obtenidos. Enfoques en paralelo proporcionan una buena expectativa en este contexto para procesamiento de imágenes médicas, vigilancia y seguimiento automático de objetos por ejemplo, sin embargo, es poco probable que la sustitución directa de las plataformas seriales existentes con máquinas paralelas sea exitosa. Esto se puede ver fácilmente en las computadoras que actualmente tienen más de un núcleo de procesamiento; cuando se lanza una aplicación, esta ocupa una de las unidades de procesamiento y su rendimiento es idéntica a una computadora con un solo núcleo de procesamiento, cuando otra aplicación es ejecutada al mismo tiempo, esta se establece en el núcleo disponible. Si se ejecuta un algoritmo de visión computacional en una computadora de más de un núcleo de procesamiento, este ocupará una sola unidad de procesamiento; por lo tanto programas paralelos para ejecutar tareas de visión deben ser desarrollados. Aunque el uso de compiladores que convierten programas seriales en unos paralelos es una solución, las características de las tareas de visión son básicamente diferentes a la mayoría del computo estructurado que la tecnología de los compiladores puede explotar [Kim K.-N., 1998]. Visión computacional además de procesamiento de imágenes y señales depende también en gran medida de matemáticas avanzadas y técnicas de inteligencia artificial.

Tal como se mencionó al principio, la reconstrucción tridimensional es un tema importante en visión computacional. Actualmente existe una gran variedad de trabajos con diferentes técnicas que se pueden encontrar en la literatura. Jarvis [1983] propone clasificar las técnicas de reconstrucción tridimensional como activas y pasivas; la técnica activa emite energía de manera controlada para sensar el entorno; Salvi et al. [2004] realizó un estudio detallado sobre las diferentes técnicas de reconstrucción tridimensional activa; la técnica pasiva no emite energía, sino que únicamente sensa la energía ambiental (luz)

reflejada en los objetos, ejemplos de tal técnica son los desarrollados por Fitzgibbon et al. [1998] y Pollefeys et al. [2004] cuyas reconstrucciones se logran únicamente a través de una secuencia de imágenes.

El objetivo principal de esta tesis es desarrollar un algoritmo de calibración en paralelo entre pares contiguos de imágenes, calibración que generalmente es común entre todos los enfoques de reconstrucción tridimensional pasiva y para ello se analizará cada etapa de calibración para su paralelización. Finalmente su integración en paralelo así como un algoritmo calibración el cual agrupa procesos se expondrá.

1.3. Trabajo relacionado.

Aunque desde hace ya algún tiempo se han construido computadoras paralelas, estas estaban disponibles a las grandes empresas que podían costearlas. Con el acercamiento de la tecnología a las personas comunes, ahora es posible construir una computadora paralela a un bajo costo y de buen rendimiento. Cada vez mas, personas e instituciones tienen la posibilidad de obtener las bondades de una computadora paralela y los trabajos en paralelo no se han hecho esperar. Con respecto al campo de la visión computacional y procesamiento de imágenes ya existe trabajo que puede ser revisado en la literatura.

Kanade and Webb [1988] presentaron en Carnegie Mellon un reporte de paralelización de visión a bajo nivel y algoritmos de control para vehículos robot sobre una máquina paralela especializada Warp —un arreglo de procesadores Warp, una unidad de interfase y un servidor—. Los objetivos alcanzados fueron algoritmos de seguimiento de caminos, evasión de obstáculos usando visión estéreo y un escáner láser con esta arquitectura. También desarrollaron una biblioteca de visión de bajo nivel de cerca de 80 programas cubriendo por ejemplo detección de bordes, suavizado, operaciones sobre imágenes, transformada de Fourier, etc. Desarrollaron además un lenguaje de programación especializado al cual llamaron Apply para la programación sobre la máquina paralela Warp.

Aunque ya existían algoritmos de correspondencia estéreo de buena velocidad y con resultados confiables, estos algoritmos siempre tenían un déficit importante para requerimientos de correspondencia estéreo en tiempo real, sistemas de navegación, visión robot, etc. Laine and Roman [1991] diseñaron un algoritmo para correspondencia estéreo en pa-

ralelo que se ejecuta sobre máquinas SIMD[2] basado técnicas de clasificación y evaluación de características, aplicación de restricción de ordenamiento y correspondencia basada en relajación, todas ellas reformuladas e integradas en términos de la ejecución paralela sobre una máquina SIMD teórica. El algoritmo propuesto trabaja en dos fases, en la fase uno, correspondencias no probables se descartan en base a restricciones geométricas suaves y el ordenamiento de algunas correspondencias previas; las correspondencias restantes son entonces evaluadas usando algún criterio de similaridad, cada conjunto de correspondencias conteniendo quizá varios candidatos, es clasificado en base al agregado de las correspondencias candidatas previamente evaluadas y finalmente cada conjunto de correspondencias es ordenada y truncada de tal forma que contienen no mas de tres candidatos posibles; en la fase dos se calculan estimados iniciales de la probabilidad de cada posible correspondencia basados en las evaluaciones individuales de las mismas y la clasificación de su conjunto, estos estimados iniciales son refinados entonces durante el proceso de relajación por una regla de consistencia que exitosamente proporciona un incremento o decremento en la probabilidad de las correspondencias si puntos cercanos tienen disparidad similar ó diferente. Este algoritmo está basado en características, *i.e.*, extracción de características como bordes o contornos.

Por otro lado, Fua [1993] mostró cómo un algoritmo estéreo con técnicas más simples que las usadas por Laine and Roman [1991], puede ser implementado en paralelo para producir mapas de profundidad densos ya que tal algoritmo está basado en correlación seguida por interpolación. A diferencia del algoritmo propuesto por Laine and Roman [1991], este es basado en área, el cual a diferencia de los basados en características, son más lentos pero producen mapas de disparidad ó profundidad más densos.

Desde el departamento de computación e ingeniería eléctrica en la universidad Heriot-Watt, Norman Raymond Scaife contribuyó en gran medida al computo paralelo no solo en tópicos de visión computacional, sino en ciencias computacionales en general. En el área de visión computacional uno de los primeros trabajos en el que se vio involucrado fue [Austin and Scaife, 1994], un reporte de trabajo que especificaba un proyecto de sistema completo para visión computacional por el cual se definió el tipo abstracto de datos ADT que se empleó durante todo el proyecto, representando con él los datos de

[2]Ver capítulo 2 para una explicación de este modelo.

entrada y salida del sistema y representando también datos intermedios entre las distintas fases del procesamiento. En general el sistema propuesto involucraba definiciones de funciones para visión a bajo nivel; HIPS —encabezado descriptivo de una secuencia de imágenes 2D—, IMAGE, —representación de arreglos 2D de valores, Canny, Hough, y otros algoritmos—. En [Scaife et al., 1996] presenta un enfoque general de desarrollo de algoritmos paralelos en un lenguaje funcional[3]. Las soluciones propuestas son presentadas en una arquitectura MIMD[4].

Aunque Kim K.-N. [1998] presentan también algoritmos paralelos y planificación de tareas en visión computacional a un bajo nivel como detección de bordes y transformada de Hough, con experimentos en localización de ojos en una secuencia de imágenes, a lo largo del trabajo se explica de manera clara el impacto que tiene el cómputo paralelo en visión computacional así como también las características de las tareas en los distintos niveles de procesamiento —bajo, intermedio y alto—, los ambientes en visión y computación en redes heterogéneas.

En el mismo año, un sistema de detección de caminos y obstáculos generales —Genetic Obstacle and Line Detection, GOLD— sobre un hardware paralelo completamente personalizado, fue planteado por Bertozzi and Broggi [1998] para su uso en vehículos móviles con la intensión de incrementar la seguridad en los caminos. El sistema en cuestión uso visión a bajo nivel *e.g.*, detección de líneas y extracción de características; visión de nivel medio *e.g* eliminación del efecto de perspectiva de producida por las cámaras, calibración de cámaras; y computo de visión de alto nivel *e.g* detección de los senderos y de los obstáculos. El sistema fue probado sobre un vehículo experimental llamado MOBLAB conducido por mas de 3000 km sobre caminos urbanos y freeways bajo diferentes condiciones de tráfico e iluminación a velocidades por arriba de los 80 km.

En su tesis doctoral Scaife Norman R. [2000] mostró un sistema incorporando conceptos de *programación funcional* y *esqueletos algorítmicos*, llevando un estudio minucioso acerca de estos dos tópicos en computación paralela y proponiendo un método para desarrollar algoritmos de visión computacional. Los esqueletos algorítmicos son maneras de desarrollar programas paralelos reutilizando estructuras algorítmicas paralelas exis-

[3] Ver sección 2.2
[4] Ver capítulo 2

tentes. Norman R. Scaife tuvo injerencia en muchos trabajos de computación paralela, entre los que se puede mencionar [Scaife et al., 1997] y [Michaelson and Scaife, 2002].

Reconstrucción tridimensional es un tópico tratado por visión computacional el cual es ampliamente valorado por los profesionales de la herencia cultural. La Red de Europea de Excelencia EPOCH, al combinar los conocimientos de dos de sus laboratorios asociados, ESAT-PSI de la Universidad Católica de Leuven en Bélgica y CNR-ISTI en Italia, instaló un sistema de reconstrucción 3D pipeline[5] para su uso en el campo de la herencia cultural. El servicio proporcionado por este sitio web —http://www.arc3d.be/— consiste en la carga de la secuencia de imágenes hacia el servidor, computo de la reconstrucción 3D, descarga de los resultados del sitio y visualización de las reconstrucciones tal como se describe en la figura 1.1. Como puede observarse, el proceso de reconstrucción automático se ejecuta sobre un servidor conectado a un cúmulo de computadoras. Consiste en tres servicios: las imágenes de la escena a reconstruir son primero verificadas por calidad y entonces cargadas al primer servicio el cual computa mapas de profundidad densos incluida la calibración de cámaras; el segundo servicio es capaz de mezclar estos mapas de profundidad; y el tercer servicio está encargado de la creación de mallas —mesh— y simplificación. Vergauwen and Gool [2006] describe como se lleva a cabo el primer servicio, *i.e.*, el cómputo de mapas de profundidad densos de las imágenes cargadas al servidor por el usuario. Se trata de esquema de reconstrucción jerárquico, masivamente paralelizado y oportunista. El pipeline consiste en general de cuatro pasos:

1. Cómputo de un conjunto de pares de imágenes que pueden ser usadas para correspondencia, incluyendo submuestréo y comparación global de imágenes. En este paso las imágenes son submuestradas (de aquí la naturaleza jerárquica). Debido a que la secuencia de imágenes puede ser cargada en una manera no secuencial, el pipeline tiene que arreglárselas para saber que imágenes puede hacer corresponder, esta es la tarea del algoritmo de comparación global de imágenes.

2. Emparejamiento, Correspondencia de Tripletas Proyectivas y Auto-calibración. En este, paso puntos característicos son extraídos de las imágenes submuestradas. Todos los posibles candidatos del paso anterior son ahora experimentados. Basado en

[5] Arquitectura que consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.

las correspondencias resultantes, todas las tripletas de imágenes son seleccionadas y proporcionan una oportunidad para una reconstrucción proyectiva. Este proceso después de desarrollarse alimenta a la rutina de Auto-calibración, la cual encuentra los parámetros intrínsecos de las posibles cámaras que pudieron haber registrado las escenas.

3. Computo de la reconstrucción Euclidiana y actualización del resultado a la resolución completa. En este paso todas las tripletas de imágenes y correspondencias

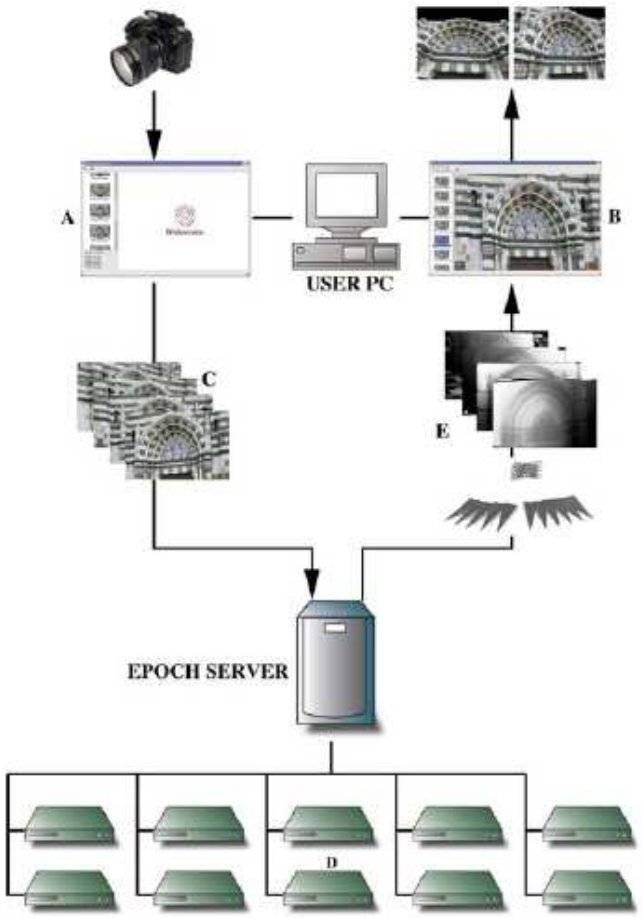


Figura 1.1: *Descripción esquemática del arreglo cliente-servidor, ver Vergauwen and Gool [2006]. La calidad de las imágenes (C) es primero verificada y luego registradas con una herramienta de carga (A) del lado del usuario del servicio; las imágenes son procesadas en un cúmulo de PC's (D); los resultados (E) pueden ser descargados vía ftp y visualizados por el usuario con la herramienta de visor de modelos (B).*

son combinadas en una reconstrucción Euclidiana 3D.

4. El ultimo paso es el responsable de producir los mapas densos de profundidad para cada imagen.

1.4. Metodología.

En esta sección se describen las etapas del proceso de calibración que se estudió en el presente trabajo representadas de manera modular, esto significa que se puede sustituir cualquier algoritmo de los acá estudiados por cualquier otro que produzca la entrada a la etapa siguiente, si se desea aumentar la exactitud de las estimaciones o aumentar la velocidad de la ejecución de los algoritmos.

- *Extracción de puntos característicos ó esquinas en las imágenes.* La extracción de puntos característicos ó esquinas, es una etapa temprana en muchos procesos de visión computacional. Estos puntos proporcionan información acerca de partes especificas de la escena que son fácilmente ubicadas con un detector de esquinas y que representan información la cual es utilizada para comenzar con las estimaciones de lo que ha sido plasmado del mundo real hacia las imágenes. Existe en la literatura una cantidad importante de detectores, entre los cuales podemos destacar los propuestos por Moravec [1977, 1979], el propuesto por Harris and Stephens [1988] así como el propuesto por Smith and Brady [1995]. El detector de esquinas considerado es el propuesto en una serie de trabajos presentados por Lucas and Kanade [1981], Tomasi and Kanade [1991] y por Shi and Tomasi [1994].

- *Búsqueda de Correspondencias entre las imágenes.* Una vez encontradas los puntos característicos en las imágenes, la siguiente etapa del proceso de calibración es la asignación de correspondencias entre cada par contiguo de la secuencia de imágenes. Esto se logra aplicando medidas de similitud o disimilitud como SSD ó ZNCC[6] [Ma et al., 2003]. Este enfoque esta basado en área, en el capitulo 4 se explica de manera detallada el por qué se denomina de esta manera; existe también un enfoque basado en características el cual no es tratado en esta tesis, sin embargo se mencionan algunos trabajos con este enfoque en el capítulo 4.

[6] En el capitulo 4 se proporciona definiciones de estas dos medidas.

- *Estimación de la geometría epipolar.* Esta geometría es estimada con las correspondencias obtenidas en la etapa anterior *i.e.*, es llevada entre cada par contiguo de la secuencia de imágenes. Esta es la culminación de la calibración débil entre los pares de imágenes.

- Finalmente, las etapas anteriores son descritas en un algoritmo general que las engloba para producir la calibración de toda la secuencia de imágenes.

1.5. Acerca de la computadora paralela.

La computadora paralela sobre la cual se hicieron los experimentos se trata de un cúmulo pequeño de computadoras que consta de 3 máquinas, cada una con dos procesadores Athlon64/Opteron con doble núcleo de AMD, *i.e.*, se puede considerar que cada computadora es una máquina paralela de 4 procesadores. En total hay disponibles 12 procesadores para los experimentos de los diferentes algoritmos que a lo largo de esta tesis se describen. También cabe mencionar que los lenguajes de programación sobre los cuales se implementaron los algoritmos son g++/GNU y MPI –Message Passing Interface–, ver sección 2.2. En palabras de Kim K.-N. [1998]:

Se piensa que el enfoque de pase de mensajes en el diseño de soluciones paralelas de alta velocidad para aplicaciones de visión es ahora lo más eficiente

1.6. Descripción del documento.

Este documento está compuesto de la siguiente etapas:

1. En el capítulo 2 se proporcionan algunos conceptos en relación al modelo de computadora secuencial y a los diferentes modelos de computadoras paralelas propuestas en la literatura, lenguajes de programación paralela y medidas de rendimiento de los algoritmos paralelos. También se describe la teoría necesaria para la calibración débil que es implementada en este trabajo.

2. El capítulo 3 es concerniente al detector de puntos característicos –esquinas–, se proporcionan los fundamentos que llevan a la construcción del detector de esquinas; se proporciona también el algoritmo secuencial, el algoritmo paralelo propuesto así como experimentos y conclusiones.

3. En el capítulo 4 se examinan los diferentes enfoques de correspondencias; se da una descripción de los diferentes modelos de deformación de regiones en las imágenes, lo cual permite establecer soluciones al problema de la correspondencia. Un algoritmo secuencial se describe así como su contraparte en el ámbito paralelo. Experimentos son llevados a cabo y resultados y conclusiones son expuestos.

4. El capítulo 5 trata acerca de la estimación de la geometría epipolar; la teoría es descrita así como un algoritmo secuencial y su contraparte en el ambiente paralelo; experimentos acerca de la conducta aleatoria del algoritmo son mostrados para justificar los resultados obtenidos cuando es considerado en forma paralela.

5. La integración de las etapas acá enumeradas 2, 3 y 4 para generar un algoritmo de calibración se describe en el capítulo 6. Se describen los algoritmos secuencial y paralelo y experimentos son llevados a cabo. Un algoritmo paralelo con un enfoque en grupos de procesos es propuesto y comparado con el algoritmo paralelo que no considera grupos.

6. Finalmente en el apéndice se proporciona breve material adicional útil en la implementación de esta tesis. Se presenta desde modelos de cámaras, matrices de proyección, definiciones de puntos y líneas y distancias.

En cada capítulo se presentan experimentos, en donde se prueba el comportamiento de los algoritmos. En los resultados de los algoritmos paralelos, se tomaron tiempos sin considerar la comunicación entre los procesadores con el fin de modelar el tiempo de procesamiento de los algoritmos y también considerando el tiempo de comunicación entre los procesadores. Cuando se haga referencia de una ejecución con un procesador, este debe entenderse como el tiempo de ejecución reportado por el algoritmo secuencial. Cabe mencionar que durante las ejecuciones de los algoritmos, no se consideró los tiempos de preparación de los datos, *e.g.*, lectura y escritura. Corridas del proceso completo de reconstrucción fueron llevadas a cabo considerando tanto tiempos sin comunicación, con comunicación y con preparación de los datos para dar las conclusiones finales de la tesis.

Capítulo 2

Fundamentos.

Por muchos años el computo se ha llevado sobre el modelo de Von-Newman. Kessler [2006] menciona que las tendencias recientes muestran que dicha arquitectura finalmente ha alcanzado sus límites y puntualiza la importancia de enseñar un entendimiento básico de computación y programación paralela desde el inicio en una educación en ciencias computacionales. Para Morrison [2003], el procesamiento paralelo es el método de desglosar grandes problemas en componentes, tareas o cálculos más pequeños que son solubles en paralelo. Bazewicz et al. [2000] menciona que un algoritmo paralelo es un método para resolver un problema computacional el cual permite separarlo en partes mas pequeñas que son resueltas simultáneamente.

La máquina sobre la cual se implementan los algoritmos paralelos que se experimentan, está basada en el modelo $logP$ descrito por Alexandrov et al. [1997] y por Culler et al. [1993], en el cual los parámetros del modelo son:

1. La latencia l ó retardo máximo asociado con el envío de un mensaje.

2. El costo de comunicación –overhead– o, representando la longitud del tiempo por el cual un procesador está ocupado durante la transmisión o recepción de un mensaje; en este intervalo el procesador no puede realizar otras operaciones.

3. El gap g el cual es un límite inferior sobre el tiempo entre la transmisión o recepción de mensajes en el mismo procesador.

4. El número P de módulos de procesadores/memoria.

Martin et al. [1997] ha mostrado que en general, las aplicaciones sobre un cluster son muy sensibles al overhead, esto es, mientras más intensiva sea la comunicación en la aplicación, más debemos esperar que su rendimiento esté afectado por la de comunicación de la maquina.

En este capítulo se presentan dos secciones; la primera presenta conceptos básicos acerca de computación paralela, máquinas paralelas y en la segunda las herramientas matemáticas necesarias para lograr el objetivo de calibración de imágenes.

2.1. Modelos de computadoras

Una *computadora paralela* es un conjunto de procesadores que son capaces de trabajar cooperativamente para resolver un problema computacional. Esta definición es lo suficientemente amplia para incluir supercomputadoras paralelas que tienen cientos o miles de procesadores, redes de estaciones de trabajo y estaciones de trabajos multiprocesadores [Foster, 1995].

2.1.1. Modelo de máquina secuencial.

Como ya se mencionó, el modelo de computadora de Von Neumann tiene una gran influencia debido a que la ejecución secuencial de las instrucciones en cada paso de tiempo es un proceso habitual en la creación de algoritmos de computación. Una computadora de Von Neumann —ver figura 2.1— comprende una unidad de procesamiento central —CPU— conectada a una unidad de almacenamiento conocido como memoria en donde la CPU ejecuta un programa almacenado que especifica una secuencia de lectura y escritura sobre la memoria y debido a que las instrucciones son desarrolladas en secuencia, a este modelo de computadoras se les llama computadoras secuenciales.

2.1.2. Modelos de computadoras paralelas.

En contraste al modelo secuencial, una computadora paralela consiste de varios procesadores, cada procesador es del mismo tipo que el usado en el modelo de Von Neumann, esto es tiene un programa que ejecuta instrucciones en secuencia cooperando con el fin de resolver un problema computacional ejecutando varias instrucciones simultáneamente, i.e., en paralelo.

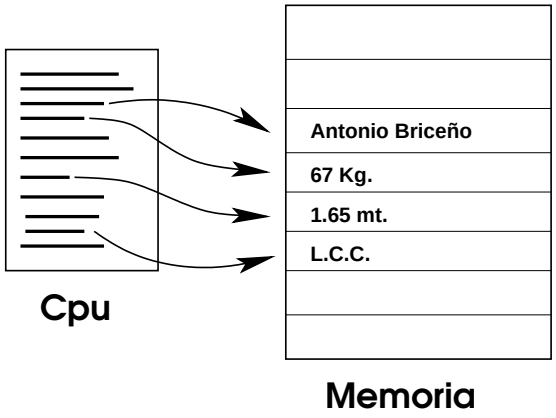


Figura 2.1: *Computadora de Von Neumann. Una unidad de procesamiento central –CPU– ejecuta un programa que desarrolla una secuencia de operaciones de lectura y escritura en una memoria asignada.*

Un *modelo paralelo de computación* es una abstracción de una computadora paralela que ignora detalles irrelevantes de implementación y captura las características importantes, esto permite enfocarse en cuestiones que mas importan cuando se diseñan soluciones paralelas a problemas computacionales. Hay varias maneras de clasificar a las arquitecturas de las computadoras paralelas y por lo tanto existe una multitud de modelos, muchos de los cuales pueden ser consultados en [Quinn, 1994]. Esta fuera del alcance de esta tesis hacer una recopilación de tales clasificaciones por lo tanto solo se mencionarán algunas. Una manera simple de clasificar estos modelos es usar dos familias generales: modelos de *memoria compartida* y *memoria distribuida*.

Modelos de memoria compartida. En esta familia de modelos paralelos de computación, los procesadores comparten una memoria común de la cual todos pueden leer y escribir a través de mecanismos de sincronización. Cada procesador usa la memoria para recibir datos, intercambiar información y depositar los resultados de sus cálculos –ver figura 2.2—. Estos procesadores operan de manera sincronizada y ejecutan la misma secuencia de instrucciones sobre diferentes datos por medio de las siguientes tres fases básicas:

i. Una fase de lectura en la cual cada procesador, si lo requiere, copia datos de la memoria hacia sus registros.

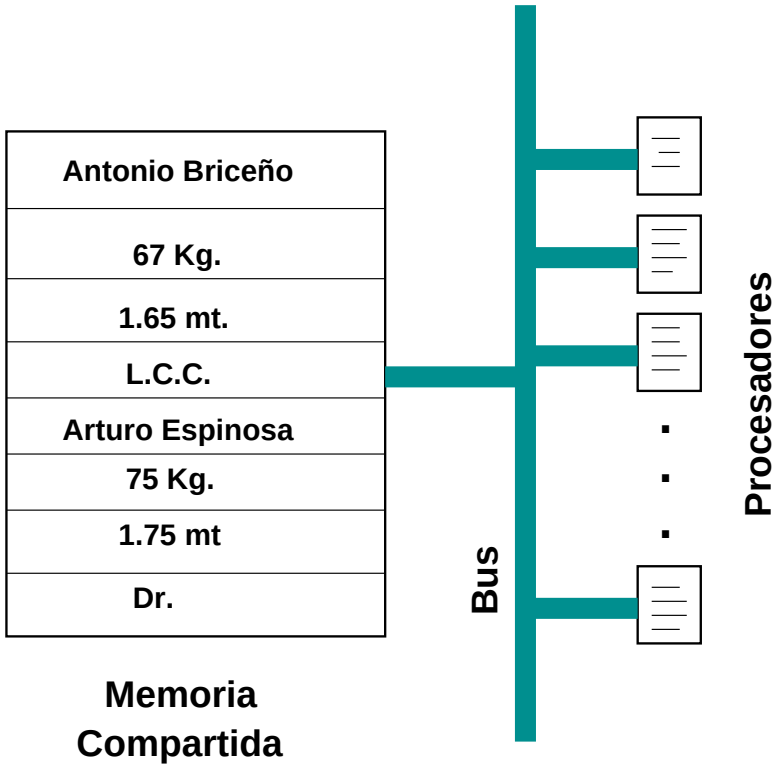


Figura 2.2: *Modelo de computadora paralela con memoria compartida.*

ii. Una fase de computación en la cual cada procesador, si así se requiere, desarrolla una operación lógica ó aritmética sobre los datos almacenados en sus registros.

iii. Una fase de escritura en la cual cada procesador, si así se requiere, copia los datos de sus registros hacia la memoria.

Es importante observar que los procesadores usan la memoria compartida para comunicarse entre si; supongamos que un procesador p_i desea enviar un dato d al procesador p_j, p_i escribe d en una localidad conocida para p_j y luego p_j lo lee de esta localidad en la memoria.

Modelos de memoria distribuida. Los modelos de memoria distribuida consisten de procesadores conectados a través de un bus de comunicación que puede ser por ejemplo una red de comunicación en la cual cada procesador tiene memoria local. La ventaja de tales modelos es que son escalables permitiendo que más procesadores puedan ser

conectados a la computadora paralela a través del bus. Un esbozo de tal modelo se muestra en la figura 2.3.

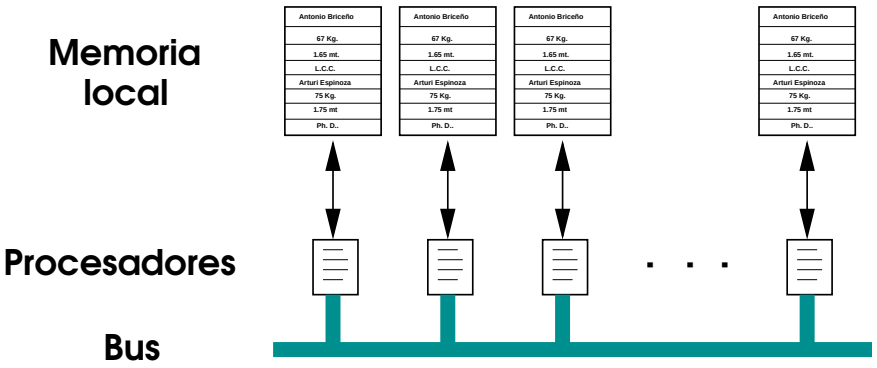


Figura 2.3: *Modelo de Computadora paralela con memoria distribuida.*

2.1.3. Clasificación de computadoras paralelas.

Existe una taxonomía presentada por Flynn [1972], la cual divide a las arquitecturas de las computadoras a lo largo de dos ejes, de acuerdo al número de datos y de instrucciones que la computadora puede procesar simultáneamente. Este es el esquema de clasificación más conocido para arquitecturas de computadoras seriales y paralelas.

Instrucción Única, Dato Único –*SISD*–. Esta es la más extendida arquitectura en la cual un solo procesador ejecuta una secuencia de instrucciones, la computadora de Von Neumann —figura 2.1— es clasificada en esta categoría y aunque la ejecución de las instrucciones puede ser del tipo *pipeline* las computadoras en esta categoría solo pueden decodificar una instrucción por cada unidad de tiempo.

Instrucción Única, Datos Múltiples –*SIMD*–. En esta arquitectura de computadora paralela, todos los procesadores ejecutan la misma instrucción en la misma unidad de tiempo pero sobre datos diferentes. El modelo de programación de computadoras SIMD está basado en conjuntos distribuidos de datos homogéneos sobre el arreglo de procesadores, cada procesador capaz de manipular su propios datos, es decir, en cada unidad de tiempo una misma operación es ejecutada sobre múltiples unidades de procesamiento que manipulan datos diferentes.

Instrucciones Múltiples, Dato Único –*MISD*–. De las cuatro categorías de la taxonomía de Flynn, estas computadoras son las menos intuitivas. ¿Cómo puede una computadora tener instrucciones múltiples pero solamente un solo dato en cada unidad de tiempo? Se piensa que no existe computadoras que no correspondan a esta categoría sin embargo Quinn [1994] y Bazewicz et al. [2000] mencionan ejemplos de máquinas que pudieran tener estas características.

Instrucciones Múltiples, Datos Múltiples –*MIMD*–. Estas máquinas constituyen el núcleo de las computadoras paralelas de hoy en día. Es un modelo reciente de ejecución paralela en la cual cada procesador ejecuta su propia copia de un mismo programa pero usando un conjunto diferente[1] de datos a los otros procesadores. Estas arquitecturas han sido muy exitosas debido a que son mucho más baratas que computadoras paralelas SIMD que son de propósito especial y pueden ser construidas con componentes no tan especializados[2].

Johnson [1988] propuso una clasificación más orientada hacia los diferentes métodos de acceso a la memoria de las computadoras paralelas que brevemente son explicados en esta tesis.

Acceso a Memoria Uniforme –*UMA*– Este tipo de computadoras garantiza que todos los procesadores usan el mismo mecanismo para acceder a todas las posiciones de la memoria y tales accesos son desarrollados con rendimiento uniforme. Computadoras en esta clasificación son de memoria compartida en la cual todos los procesadores acceden a una unidad de memoria central —ver figura 2.2—.

Acceso a Memoria No Uniforme –*NUMA*–. Este tipo de computadoras son máquinas donde cada procesador tiene su propia memoria, sin embargo la interconexión permite a los procesadores acceder a la memoria local de otro procesador. Es un modelo de computadora paralela con memoria distribuida en la cual cada procesador puede acceder la memoria completa de la computadora.

[1] Aunque no en exclusión mutua.

[2] Componentes que fácilmente pueden ser comprados en con algún proveedor de partes de computadora.

Acceso a Memoria No Remota –*NORMA*–. Es una computadora en donde cada procesador tiene una memoria privada y esta es accesible a los demás procesadores únicamente a través de pase de mensajes y por lo tanto no se puede considerar que haya una memoria global como puede suponerse en las computadoras UMA y NUMA. Estas computadoras proporcionan un poder de procesamiento a un bajo costo debido a que no se necesita una infraestructura de hardware asociada a coordinar que múltiples procesadores accedan a la misma unidad de memoria.

Ya que se ha descrito de manera breve la clasificación de las computadoras paralelas según Flynn [1972] y Johnson [1988] así como las arquitecturas –memoria compartida y distribuida–, en la siguiente sección se manejara el asunto en cuanto a los diferentes tipos de lenguajes que pueden ser usados en cuanto al diseño de un algoritmo que será ejecutado en una computadora paralela.

2.2. Lenguajes de programación paralela.

El lenguaje de programación juega un papel importante en la computación debido a que la simplificación en la expresión de un algoritmo complejo depende de la elección del lenguaje de programación elegido. La computación paralela complica aun más la de por si ya desafiante tarea de desarrollar un programa correcto y eficiente. Los lenguajes de *paralelismo de datos* pueden habilitar a ciertos programas con semántica secuencial a ejecutarse eficientemente sobre una computadora paralela, por lo tanto un programador no necesita escribir explícitamente código paralelo, por otro lado, los lenguajes *funcionales paralelos* pueden ser usados para expresar algoritmos altamente concurrentes mientras preservan ejecuciones no deterministas —el lector interesado en este tipo de programación paralela puede revisar [Hammond and Michelson, 2000]—, una forma de lenguaje paralelo de muy bajo nivel son las bibliotecas de *pase de mensajes* que pueden ser usadas para escribir programas portables que aun así logran un muy alto rendimiento sobre computadores paralelas de memoria distribuida y por lo tanto un programador no necesita escribir programas diferentes para distintos tipos de computadoras. Bazewicz et al. [2000] proporciona un estudio detallado acerca de los tipos y clases de lenguajes para programación paralela así como los nuevos paradigmas de programación. El enfoque de programación empleado en esta tesis es el basado en pases de mensajes, como

MPI[3], el cual no es un lenguaje del todo debido a que son implementaciones basadas en llamadas a bibliotecas de funciones.

Interfase de Pase de Mensajes –Message Passing Interface–. El modelo actualmente predominante para programación en supercomputadoras es el pase de mensajes con MPI. El código de pase de mensajes es generalmente no estructurado y difícil de entender, mantener y depurar. En las implementaciones MPI, un grupo de procesos es creado en la inicialización del programa y un proceso es creado por cada procesador, sin embargo estos procesos pueden ejecutar diferentes programas. Por lo anterior los modelos de programación MPI es algunas veces referido como MPMD –Multiple Program Multiple Data– para distinguirlo del modelo SMPD en la cual cada procesador ejecuta el mismo programa. Snir and Otto [1998] y Gropp et al. [1998] hacen una recopilación completa acerca de MPI. Debido a que el estándar MPI soporta modularidad en la programación vía un mecanismo de comunicación, esto permitió diseñar un algoritmo que trabaja con grupos de procesos en el que se mejoró la ejecución del algoritmo de calibración el cual es el objetivo de esta tesis.

2.3. ¿Cómo evaluar los algoritmos paralelos?

Existen varios criterios que se pueden usar a fin de evaluar la calidad de los algoritmos paralelos. En este trabajo de tesis los criterios usadosse se explican brevemente en los siguientes párrafos.

Tiempo de ejecución. La razón primaria para diseñar un algoritmo paralelo es usarlo de tal manera que una tarea computacional pueda ser completada rápidamente. Es por lo tanto natural medir el tiempo de ejecución como un indicador de la bondad de un algoritmo paralelo. Aunque una útil manera de estimar el tiempo de ejecución de un algoritmo está proporcionado por el análisis matemático de las operaciones requeridas, el acceso a memoria para lectura y escritura de datos esto no es considerado. La medición del tiempo de ejecución que se empleará, esta dada por el conteo de las unidades de milisegundos comprendidas entre el momento de inicio y termino de la ejecución de un

[3]Implementación MPI-2.

algoritmo.

Rapidez. En computación paralela la rapidez se refiere a que tan rápido es un algoritmo en comparación con otro que resuelve el mismo problema a través del cociente de sus tiempos. Dos tipos de mediciones de rapidez pueden ser llevadas a cabo; *rapidez relativa* la cual se refiere la razón entre el tiempo medido por una computadora paralela ejecutando un algoritmo paralelo con un procesador y el mismo algoritmo ejecutado con p procesadores y *rapidez absoluta* la cual es la razón de los tiempos entre el algoritmo optimizado para ejecución secuencial y el paralelo. La medición de rapidez que se considerará a lo largo de esta tesis es la absoluta la cual está dado por la siguiente fórmula:

$$S_p = \frac{T_s}{T_p} \tag{2.1}$$

donde T_s es el tiempo de ejecución del algoritmo secuencial y T_p es el tiempo de ejecución del algoritmo paralelo con p procesadores. Cuando se programan algoritmos paralelos una rapidez lineal o ideal se obtiene cuando $S_p = p$ usando p procesadores en el algoritmo paralelo, *i.e.*, cuando se ejecuta un algoritmo con rapidez ideal, al doblar el número de procesadores se consigue el doble de rapidez, cuando esto sucede se considera que se tiene una muy buena escalabilidad.

Eficiencia. Es la proporción del tiempo que se pasa desarrollando trabajo útil. Es una métrica de rendimiento definida como:

$$E_p = \frac{S_p}{p} \tag{2.2}$$

donde su valor esta entre cero y uno [4] estimando que tan bien son utilizados los procesadores en resolver el problema comparado a qué tanto esfuerzo se pierde en comunicación y sincronización. Algoritmos con rapidez lineal tienen eficiencia de 1 mientras que los difíciles de paralelizar tienen una eficiencia de tal manera que $\lim_{p\to\infty} \frac{1}{\log p} = 0$.

Costo. El costo de un algoritmo paralelo es un límite sobre el número total de operaciones básicas ejecutadas colectivamente por los procesadores. El costo de trabajo de

[4] En algunas ocasiones la eficiencia sobrepasa el 100 % cuando se trata de algoritmos heurísticos como se analizará en el capítulo 5.

computación está definido como:

$$C = \frac{T_s}{E_p} = p \cdot T_p. \tag{2.3}$$

Cuando el costo de un algoritmo paralelo para un problema dado corresponde hasta un factor de escala a un límite inferior sobre el número de operaciones requeridas para resolver ese problema, se dice que es de costo *óptimo*.

2.4. Visión geométrica.

A través de las secciones anteriores se ha dado una pequeña introducción acerca de los diferentes tipos de arquitecturas de computadoras paralelas. Aunque el tema de la presente sección es completamente acerca de otra área, es vital para el entendimiento del proceso de calibración el cual es lo que se desea paralelizar en este trabajo de tesis.

2.4.1. Geometría epipolar.

La *geometría epipolar* es la geometría intrínseca entre dos imágenes, es independiente de la estructura de la escena y solamente depende de los parámetros internos de la cámara y del pose de la misma. La geometría epipolar entre las dos imágenes es esencialmente la geometría de la intersección de los planos de las dos imágenes con el plano π formado por los tres puntos C, C' y X —ver figura 2.4(a)—. Las entidades envueltas en la geometría epipolar están ilustradas en la figura 2.4, cuya terminología es:

- El *epipolo*, el cual es el punto de intersección de la línea que une los centros de las cámaras —línea base— con el plano de la imagen. En otras palabras, es la imagen del centro óptico de la otra cámara.
- El *plano epipolar*, el cual es el plano definido por la línea base y el punto $\mathbf{X}$.
- Una *línea epipolar*, la cual es la intersección del plano epipolar con el plano de la imagen. Todas las líneas epipolares intersecan al epipolo. El plano epipolar interseca a las dos imágenes formando líneas epipolares correspondientes.

Esta geometría está usualmente motivada por la búsqueda de puntos correspondientes en visión estéreo y es el mismo motivo por el cual se considera aquí. Hartley and Zisserman

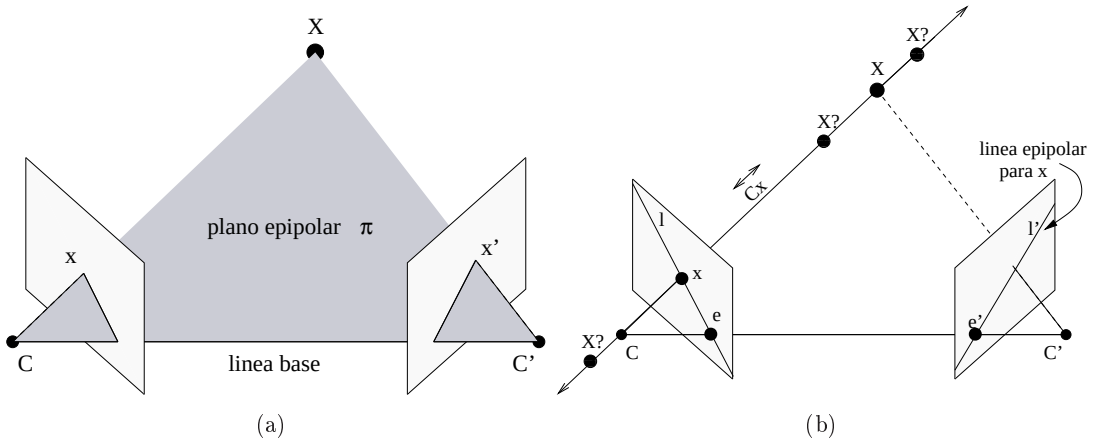


Figura 2.4: *Geometría epipolar. (a) las dos cámaras están indicadas por los centros* $\mathbf{C}$ *y* $\mathbf{C}'$ *y los planos de las imágenes. Los centros de las cámaras, el punto* $\mathbf{X}$ *y sus imágenes* $\mathbf{x}$ *y* $\mathbf{x}'$ *están en un plano* π*. (b) La imagen de un punto* $\mathbf{x}$ *se reproyecta sobre una recta definida por el centro de la cámara* $\mathbf{C}$ *y la imagen* $\mathbf{x}$ *y se proyecta en la segunda imagen como la línea* $\mathbf{l}'$*. El punto tridimensional* $\mathbf{X}$ *el cual proyecta a* $\mathbf{x}$ *en la primera imagen debe estar sobre este rayo, así que la imagen de* $\mathbf{X}$ *en la segunda imagen debe estar sobre la línea* $\mathbf{l}'$*.*

[2004], Ma et al. [2003] y Faugeras et al. [2001] proporcionan tratados completos acerca de la geometría epipolar y el lector interesado debe dirigirse a estos textos.

2.4.2. Matriz fundamental.

Existe una matriz que encapsula a la geometría epipolar: *la matriz fundamental* comunmente denotada como $\mathbf{F}$ que es 3×3 y rango 2, que puede ser estimada a partir de puntos correspondientes entre las imágenes. Dado un par de imágenes la matriz fundamental relaciona a un punto en una imagen con su correspondiente línea epipolar en la otra imagen de la siguiente manera:

$$\mathbf{l}' = \mathbf{F}\mathbf{x}. \tag{2.4}$$

De acuerdo a la ecuación B.1, si $\mathbf{x}'$ está sobre $\mathbf{l}'$, entonces $0 = \mathbf{x}'^{\top}\mathbf{l}' = \mathbf{x}'^{\top}\mathbf{F}\mathbf{x}$, de donde se concluye que la matriz fundamental está definida por la ecuación

$$\mathbf{x}'^{\top}\mathbf{F}\mathbf{x} = 0, \tag{2.5}$$

para cualquier par de correspondencias $\mathbf{x} \leftrightarrow \mathbf{x}'$ en las imágenes.

2.4.3. Estimación de la matriz fundamental.

Dados suficientes correspondencias $\mathbf{x} \leftrightarrow \mathbf{x}'$, la ecuación 2.5 puede ser usada para determinar a la matriz $\mathbf{F}$. Con las coordenadas homogéneas —ver apéndice B— de las imágenes;

$$\mathbf{x} = [x\ y\ 1]^\top \qquad \text{y} \qquad \mathbf{x}' = [x'\ \ y'\ \ 1]^\top$$

cada correspondencia resulta en una ecuación lineal en las entradas desconocidas de $\mathbf{F}$. Consideremos ahora la expresión matricial de la ecuación 2.5:

$$\mathbf{x}'^\top \mathbf{F}\mathbf{x} = (\,x' \quad y' \quad 1\,) \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

de esta expresión podemos notar específicamente que la ecuación correspondiente a un par de puntos $\mathbf{x} = (x\ y\ 1)^\top$ y $\mathbf{x}' = (x'\ y'\ 1)^\top$ está dada por

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0. \tag{2.6}$$

Si denotamos las entradas de $\mathbf{F}$ como un vector

$$\mathbf{f} = (f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33})^\top,$$

la ecuación 2.6 puede ser expresada como un producto punto:

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\mathbf{f} = 0.$$

De un conjunto de n correspondencias, se obtiene un conjunto de n ecuaciones lineales en la forma

$$\mathbf{A}\mathbf{f} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{f} = \mathbf{0}, \tag{2.7}$$

que es un conjunto de ecuaciones homogéneas y $\mathbf{f}$ puede ser determinado de acuerdo a alguna escala. Para que una solución exista, la matriz $\mathbf{A}$ debe tener rango cuando menos 8 y la solución puede ser determinada por métodos lineales como el espacio nulo de $\mathbf{A}$.

El algoritmo —ver el algoritmo 2.1— utilizado para calcular a la matriz fundamental dadas un conjunto de correspondencias es el denominado *algoritmo de los 8 puntos normalizados* propuesto por Longuet-Higgins [1981] al cual posteriormente Hartley [1997]

propuso una normalización para eliminar problemas numéricos convirtiéndolo en un algoritmo robusto y confiable el cual envuelve únicamente la construcción y solución por mínimos cuadrados de un conjunto de ecuaciones lineales —ecuación 2.7—.

Existe una solución que usa un mínimo de 7 correspondencias que puede ser consultado en [Hartley and Zisserman, 2004].

Algoritmo 2.1: *Algoritmo de los 8 puntos normalizados.*

Objetivo: *Dadas* $n \geq 8$ *correspondencias* $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$, *determinar la matriz* $\mathbf{F}$ *tal que* $\mathbf{x}_i^\top \mathbf{F} \mathbf{x}_i = 0$.
Algoritmo:

1. **Normalización:** Transformar las coordenadas de las imágenes de acuerdo a $\hat{\mathbf{x}}_i = \mathbf{T}\mathbf{x}_i$, y $\hat{\mathbf{x}'}_i = \mathbf{T}'\mathbf{x}'_i$, donde $\mathbf{T}$ y $\mathbf{T}'$ están definidas como

$$\mathbf{T} = \begin{bmatrix} k & 0 & -k\bar{x} \\ 0 & k & -k\bar{y} \\ 0 & 0 & 1 \end{bmatrix} \quad \text{y} \quad \mathbf{T}' = \begin{bmatrix} k' & 0 & -k'\bar{x}' \\ 0 & k' & -k'\bar{y}' \\ 0 & 0 & 1 \end{bmatrix}$$

donde

$$\bar{\mathbf{x}} = \begin{bmatrix} \bar{x} \\ \bar{y} \\ \bar{z} \end{bmatrix} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i, \quad \text{y} \quad k = \frac{\sqrt{2}}{\frac{1}{N} \sum_{i=1}^{N} \parallel \mathrm{x}_i - \bar{\mathrm{x}} \parallel},$$

$\mathbf{x}'$ y k' se definen de manera similar.

2. Encontrar la matriz fundamental $\hat{\mathbf{F}}'$ correspondiente a $\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i$ haciendo:

   a) **Solución lineal.** Determinar $\hat{\mathbf{F}}$ del vector singular correspondiente al valor singular mas pequeño de $\hat{\mathbf{A}}$, donde $\hat{\mathbf{A}}$ está definida como la ecuación 2.7 compuesta por las correspondencias $\hat{\mathbf{x}}_i \leftrightarrow \hat{\mathbf{x}}'_i$.

   b) **Restricción.** Remplazar $\hat{\mathbf{F}}$ por $\hat{\mathbf{F}}'$ de manera que $\det\hat{\mathbf{F}}' = \mathbf{0}$, para que la matriz fundamental sea singular. Esto se logra encontrando $\hat{\mathbf{F}}'$ que minimiza la norma Frobenius $\parallel \hat{\mathbf{F}} - \hat{\mathbf{F}}' \parallel$.
   En particular $\hat{\mathbf{F}}' = \mathbf{U}\mathrm{diag}(r, s, 0)\mathbf{V}^\top$ la minimiza.

3. **Denormalización:** Hacer $\mathbf{F} = \mathbf{T}'^\top \hat{\mathbf{F}}' \mathbf{T}$. Finalmente $\mathbf{F}$ es la matriz fundamental correspondiente a los datos originales $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$.

Capítulo 3

Detector de esquinas KLT.

En la adquisición automática y análisis de información visual y muchas veces es necesario establecer algunas correspondencias iniciales entre imágenes. En el capítulo anterior se proporcionaron definiciones y un algoritmo de visión geométrica necesarios para el proceso de calibración. Las definiciones y el algoritmo trabajan con primitivas geométricas tales como puntos y líneas, sin embargo las imágenes no son mediciones de puntos o líneas de la escena que ha sido registrada con la cámara, sino que son arreglos de números positivos que miden la cantidad de luz que incide en un sensor en una localidad particular en un lapso. Por lo tanto ¿cómo establecer una relación entre la visión geométrica y el hecho de que lo que se mide con una cámara no son puntos o líneas si no intensidades de luz?

Las correspondencias iniciales proporcionan información acerca de partes que fácilmente pueden ser ubicados en una misma escena registrada en una serie de imágenes, permitiendo a partir de tales correspondencias, concentrarse en la visión geométrica. Es por lo tanto necesario encontrar con un detector de esquinas, los puntos de alta curvatura —puntos característicos ó también conocidos como esquinas—. Entre los detectores de puntos característicos que existen en la literatura, podemos destacar los trabajos propuestos por Moravec [1977, 1979], Harris and Stephens [1988] y por Smith and Brady [1995]. Hacer una clasificación de los detectores de esquinas es una tarea difícil ya que muchas veces un detector propuesto resulta de una mejora de otro ya presentado. Sin embargo, los mencionados anteriormente son los más citados en los trabajos que requieren un detector de esquinas. El proceso de calibración que se describe en esta tesis, hace uso de un detector de esquinas propuesto en una serie de trabajos, Lucas and Kanade

[1981], Tomasi and Kanade [1991] y Shi and Tomasi [1994], el cual es conocido como el operador KLT — Kanade-Lucas-Tomasi —.

En el presente capítulo se exhiben las bases teóricas del detector elegido, así como su implementación secuencial y en paralelo.

3.1. KLT.

El detector de esquinas KLT fue propuesto unos pocos años después del operador Harris, formulado por Harris and Stephens [1988]. KLT, así como Harris, opera sobre una matriz de estructura local C, la cual tiene la forma

$$C = \begin{bmatrix} \left(\frac{\partial I}{\partial x}\right)^2 & \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} \\ \\ \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} & \left(\frac{\partial I}{\partial x}\right)^2 \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \\ \frac{\partial I}{\partial y} \end{bmatrix} \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} \end{bmatrix}. \tag{3.1}$$

En la práctica, las derivadas parciales $\partial I/\partial x$ y $\partial I/\partial y$ son usualmente remplazadas por convoluciones hechas con una mascara Prewitt ó Sobel, ó cualquier otra máscara que estime las derivas de una imagen. Sean $\lambda_1 \geq \lambda_2 \geq 0$ los valores propios de la matriz C. Como C es una matriz simétrica y positiva definida, λ_1 y λ_2 son ambos no negativos. Algunas interpretaciones geométricas pueden ser descritas de estos valores propios [Tomasi and Kanade, 1991]:

- Para una imagen perfectamente uniforme, $C = 0$ y $\lambda_1 = \lambda_2 = 0$.
- Para un borde perfecto blanco y negro, $\lambda_1 > 0$ y $\lambda_2 = 0$ donde el vector propio asociado a λ_1 es ortogonal al borde.
- Para una esquina de un cuadro negro contra un fondo blanco, $\lambda_1 \geq \lambda_2 > 0$; mientras más grande sea el contraste en una dirección, más grande será el valor propio de esa dirección.

Los vectores y valores propios codifican direcciones y magnitudes de bordes respectivamente, esto implica que una esquina debería ser marcada en una localidad donde el valor propio λ_2 es suficientemente grande [Tomasi and Kanade, 1991], más grande que un umbral λ_{min}, el cual puede ser estimado del valle más cercano a cero del histograma de λ_2, sin embargo tal valle no siempre está presente y por lo tanto un umbral debe ser proporcionado.

3.1.1. Algoritmo secuencial KLT.

A grandes rasgos, se deben encontrar las derivadas parciales de la imagen en la dirección x e y, construir para cada píxel a la matriz C, encontrar los valores propios de dicha matriz y formar una lista L con los valores propios λ_2 mayores a un umbral λ_{min}. La lista L así formada contiene información de los píxeles que potencialmente son esquinas, sin embargo, se espera que los píxeles con λ_2 de mayor magnitud sean las esquinas mas marcadas en la imagen, esto indica que debe centrarse en los píxeles con λ_2 más grandes, lo cual se logra ordenando a la lista L de mayor a menor. Finalmente se lleva a cabo una supresión de no-máximos en la lista. El algoritmo considera dos parámetros: un umbral λ_{min} sobre el valor propio λ_2 y una vecindad de radio r. Una descripción completa se proporciona en el algoritmo 3.1.

Algoritmo 3.1: *Algoritmo secuencial del detector de esquinas KLT.*

Objetivo: *Dada una imagen $I(x,y)$, un umbral λ_{min} y un radio r, encontrar los puntos característicos ó esquinas de dicha imagen.*
Algoritmo:

1. Cálculo de las derivadas $\partial I/\partial x$ y $\partial I/\partial y$.
2. Para cada punto en la imagen (x,y):
   - i construir la matriz local sobre una vecindad $\mathcal{R}$ de dimensión $(2r+1)\times(2r+1)$ centrada en (x,y) como sigue:
   $$C(x,y) = \begin{bmatrix} \sum_{\mathcal{R}} \left(\frac{\partial I}{\partial x}\right)^2 & \sum_{\mathcal{R}} \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} \\ \sum_{\mathcal{R}} \frac{\partial I}{\partial x}\frac{\partial I}{\partial y} & \sum_{\mathcal{R}} \left(\frac{\partial I}{\partial x}\right)^2 \end{bmatrix}.$$
   - ii calcular el valor propio λ_2 de la matriz $C(x,y)$.
   - iii si $\lambda_2 > \lambda_{min}$, guardar la terna (λ_2, x, y) en la lista potencial de esquinas L.
3. Ordenar L en orden decreciente de λ_2.
4. Supresión de no-máximos: Explorar de arriba para abajo con respecto a λ_2 la lista ordenada e ir eligiendo los puntos (x,y) que se no hayan removido con anterioridad. Cuando un punto es elegido como esquina, todos los puntos restantes de L que caigan dentro de la vecindad $\mathcal{R}$ centrada en el punto seleccionado son removidos.

3.1.2. Algoritmo paralelo KLT.

Para lograr el funcionamiento del operador KLT, es necesario aplicar todos los pasos del algoritmo secuencial. Analizando el porcentaje de tiempo necesitado para la ejecución

del operador KLT, se decidió paralelizar las tareas que más tiempo toman. Por ejemplo para una imagen de 4843776 píxeles, el porcentaje del tiempo requerido por el paso 1 fue de 24.83 %, el paso 2 fue de 22.66 % y el paso 3 fue de 39.76 %, que reunidos hacen un 87.25 % del tiempo consumido. El resto del tiempo le corresponde en conjunto al paso 4 además de diferentes llamadas a funciones que son necesarias en la implementación. Por lo tanto las tareas elegidas a paralelizarse son las llevadas a cabo en el paso 1, 2 y 3 del algoritmo secuencial. El paso 1 y 2 del algoritmo 3.1 se lleva a cabo desarrollando lo explicado en los siguientes párrafos.

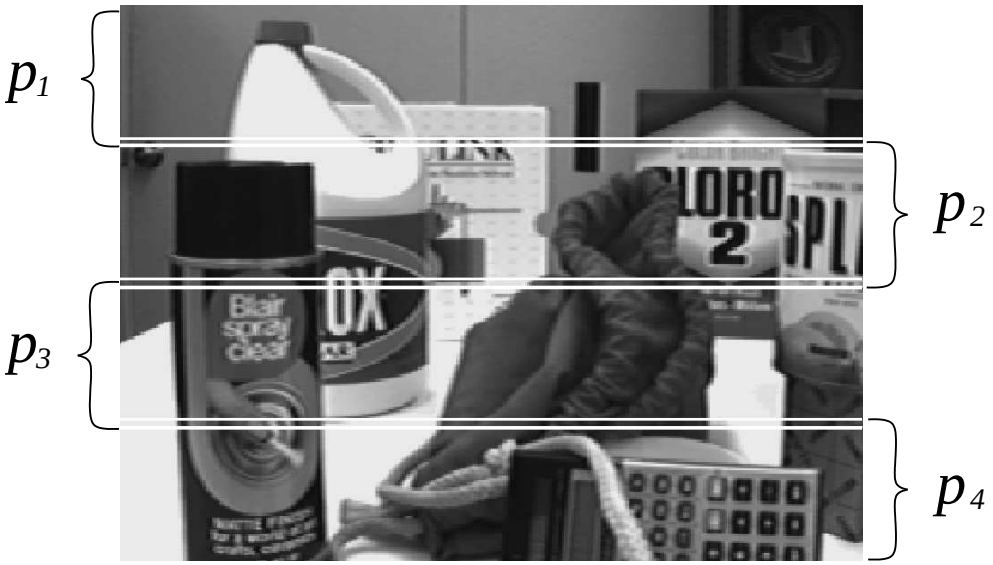


Figura 3.1: *Imagen dividida para cuatro procesadores con traslapes para calcular de manera correcta las derivadas sin perder información por causa de utilizar máscaras.*

División de la imagen para cada procesador. La primera consideración que se debe tener en cuenta, es la partición de la imagen de tal forma que cada procesador en la computadora paralela tenga una región de la imagen, pero también un pequeño traslape de la mitad del tamaño de mascara empleada para estimar las derivadas parciales de tal forma que puedan encontrarse $\partial I/\partial x$ y $\partial I/\partial y$ de manera correcta y por lo tanto también a la matriz C en todas las coordenadas (x, y) de la imagen. En la figura 3.1 se muestra una imagen dividida para 4 procesadores de una computadora paralela. Una vez fragmentada la imagen, el paso 1 y 2 del algoritmo secuencial se lleva a cabo en

cada procesador de la máquina paralela, y se construyen sublistas L_i, con $L = \bigcup L_i$, $i = 0 \ldots p-1$ y por lo tanto hay que ordenar con un enfoque paralelo la lista potencial L.

El ordenamiento paralelo es uno de los problemas más estudiados en computación científica debido a su interés teórico e importancia práctica y por lo tanto varios algoritmos de ordenamiento paralelo han sido propuestos. Quinn [1994] proporciona un estudio detallado sobre diferentes algoritmos, entre los que se encuentra tres algoritmos confiables sobre una máquina MIMD: Quicksort Paralelo, Hyperquicksort y PSRS. Quinn [1994] también muestra que los algoritmos paralelos Hyperquicksort y PSRS tienen un mejor desempeño y son más escalables que Quicksort Paralelo.

Ordenamiento Paralelo. El método de ordenamiento que se adoptó aquí es el Ordenamiento Paralelo por Muestreo Regular —PSRS por sus siglas en inglés— que fue propuesto por Shi and Schaeffer [1992]. El algoritmo paralelo PSRS encuentra pivotes para fragmentar los datos en subconjuntos ordenados de aproximadamente igual tamaño usando muestreo regular de las sublistas ordenadas de los datos en los procesadores.

Sea X de tamaño n el conjunto de datos a ser ordenados en una máquina MIMD con p procesadores. Sea $X_{i:j} = \{X_i, X_{i+1}, \ldots, X_j\}$, donde $0 \leq i \leq j < n$. Por simplicidad en el análisis del algoritmo paralelo, se asume $p^2 | n$ y $X_i \neq X_j$ para $i \neq j$. PSRS consta de tres fases:

- Cada uno de los p procesadores ordena una sublista contigua de tamaño $w = \frac{n}{p}$ usando algún algoritmo secuencial de ordenamiento. De manera más precisa, cada procesador i $(0 \leq i < p)$ ordena una sublista $X_{(i)w:(i+1)w-1}$. De las sublistas se recolectan $p-1$ muestras

$$X_{\frac{w}{p}}, \quad X_{\frac{2w}{p}}, \quad \ldots, \quad X_{\frac{(p-1)w}{p}}$$

  sobre cada sublista ordenada en los procesadores para formar en un procesador una muestra regular Y de $p(p-1)$ elementos.

- La muestra regular en el procesador maestro es ordenada de forma secuencial, resultando en una lista ordenada $Y_1, Y_2, \ldots, Y_{p(p-1)}$. Seleccionar $p-1$ pivotes para

particionar X distribuido en los procesadores de la manera siguiente

$$Y_{\frac{p}{2}}, \quad Y_{p+\frac{p}{2}}, \quad \ldots, \quad Y_{(p-2)p+\frac{p}{2}}$$

referidos como

$$y_1, \quad y_2, \quad \ldots, \quad y_{p-1}.$$

El particionamiento de X se completa como sigue: cada procesador encuentra donde cada uno de los $p-1$ pivotes divide la sublista que contiene, usando búsqueda binaria dividiéndola en p bloques y cada procesador envía el bloque i al procesador i.

- Finalmente después de que cada procesador contiene ya los bloques que le corresponde y que conforman una nueva sublista, se lleva a cabo un último ordenamiento secuencial. De esta manera cada procesador contiene una porción ordenada de X tal que ningún elemento en el procesador i es mayor que los elementos del procesador $i+1$. Por lo tanto solo se deben unir las sublistas para tener X ordenado.

La versión en paralelo del detector de esquinas KLT, se muestra en el algoritmo 3.2.

Algoritmo 3.2: *Algoritmo paralelo del detector de esquinas KLT.*

Objetivo: *Dada una imagen $I(x,y)$, un umbral λ_{min} y un radio r, encontrar los puntos característicos ó esquinas de dicha imagen de forma paralela.*
Algoritmo:

1. Distribuir la imagen entre los procesadores de acuerdo a lo explicado en la subsección 3.1.2.
2. Aplicar los pasos 1 y 2 del algoritmo 3.1 en cada porción de la imagen contenida en cada procesador i y formar sublistas L_i, donde $L = \bigcup L_i$, $i = 0 \ldots p-1$ en p procesadores.
3. Ordenar de mayor a menor con PSRS la lista L.
4. Recolectar la lista L en el procesador maestro y efectuar el paso 4 del algoritmo 3.1.

3.2. Experimentos.

Para probar los resultados de esta etapa del proceso de calibración, se efectuaron experimentos con imágenes de distintos tamaños y con diferente número de procesadores. Imágenes con tamaños de 393216, 1572864, 3538944, 6291456 píxeles y desde 1 hasta

12 procesadores. Cada combinación imagen-número de procesadores se ejecutó 30 veces y sus promedios son los datos que se analizan. Análisis del tiempo empleado, rapidez alcanzada, eficiencia de los procesadores y costo implicado sobre las ejecuciones del KLT en las diferentes imágenes, con y sin overhead se describen en los siguientes párrafos. Es importante recalcar que en p_1 el algoritmo utilizado es el secuencial.

Tiempo de ejecución. En la cuadro 3.1 se describe el desarrollo del algoritmo KLT en cuanto a tiempo necesitado en milisegundos y en la figura 3.2 se muestra una descripción gráfica del comportamiento del algoritmo de acuerdo al tamaño de las imágenes probadas y un número de procesadores $p = 1, \ldots, 12$. Cuando se ignora el tiempo de comunicación

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	393216	683	461	315	243	191	156	134	118	104	91	82	76
	1572864	2994	2057	1400	1103	862	710	600	523	462	418	385	356
	3538944	7183	4930	3349	2622	2049	1689	1441	1276	1131	1002	908	862
	6291456	13126	9061	6268	4892	3804	3121	2687	2375	2088	1876	1715	1600
Con	393216	683	519	357	342	279	237	226	227	229	306	375	366
	1572864	2994	2291	1574	1405	1190	1044	869	818	819	801	818	826
	3538944	7183	5426	3754	3149	2603	2270	1973	1888	1850	1751	1759	1694
	6291456	13126	9862	6931	5703	4734	4080	3580	3276	3117	2906	2737	2783

Cuadro 3.1: *Tiempos en milisegundos empleados en la ejecución del algoritmo paralelo 3.2, para imágenes de diferentes tamaños utilizando desde 1 hasta 12 procesadores con y sin overhead.*

entre las máquinas, el cuadro 3.1 y la figura 3.2 muestran que el algoritmo presenta el comportamiento deseado de reducir el tiempo de ejecución con cada procesador que se agrega. Sin embargo cuando se toma en cuenta la comunicación, el cuadro 3.1 muestra que a pesar de tener una reducción importante del tiempo procesando una imagen de 393216 píxeles hasta los primeros 6 procesadores, con 7 procesadores el decremento es leve y a partir de 8 procesadores comienza a incrementarse el tiempo necesitado, esto puede observarse también de la figura 3.2(a). Puede considerarse que es suficiente 6 procesadores para una imagen de este tamaño con este enfoque paralelo. Algo similar ocurre con la imagen de 1572864 píxeles, el decremento importante ocurre hasta los primeros 7 procesadores y a partir de 8 procesadores este decremento no es notable, incluso aumentando con 11 procesadores —ver figura 3.2(b)—. Con la imagen de 3538944 píxeles un decremento marcado se tiene hasta los primeros 7 procesadores y a partir de este punto, el decremento no es significativo incluso aumentando con 11 procesadores —ver figura 3.2(c)—. Con la imagen de mayor tamaño se tiene un decremento hasta los

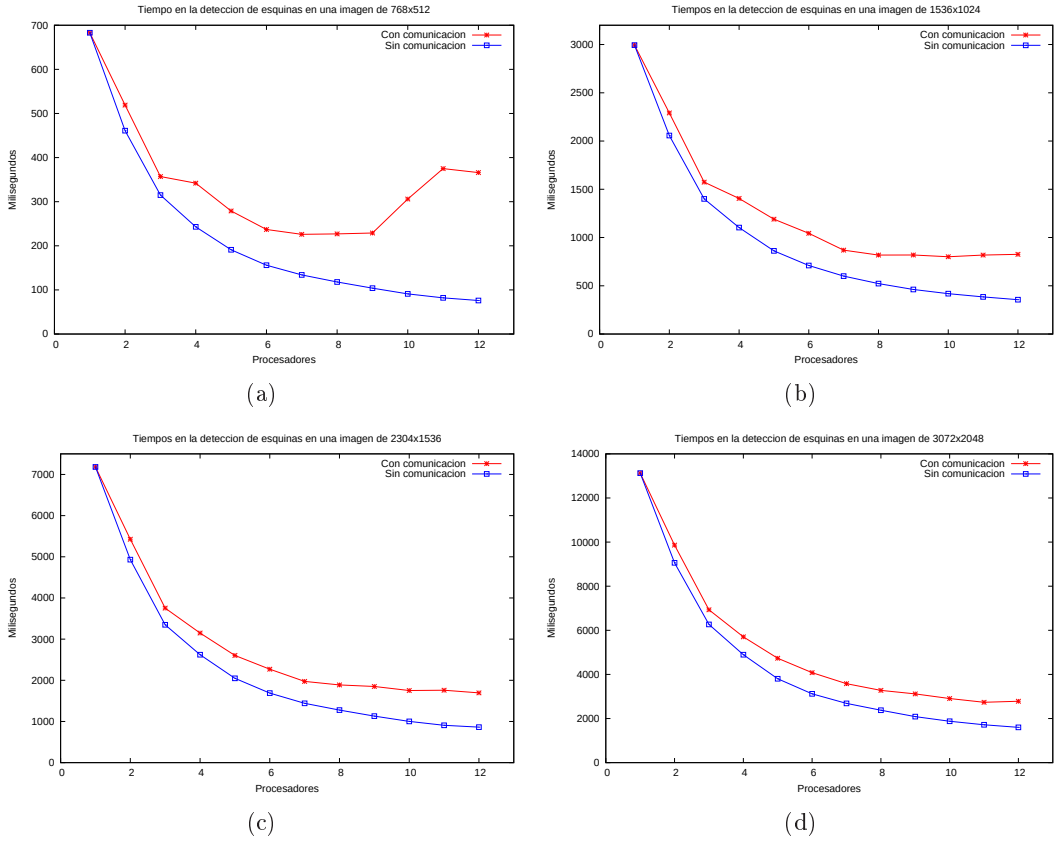


Figura 3.2: *Gráficas de lo tiempos promedios alcanzados con el algoritmo paralelo 3.2 con y sin comunicación entre los procesadores.*

primeros 11 procesadores y con 12 procesadores hubo un pequeño incremento del tiempo —ver figura 3.2(d)—.

Con los tiempos del cuadro 3.1 se calcularon las métricas de rapidez —speedup—, eficiencia y costo del algoritmo en la máquina paralela y son analizados en los párrafos siguientes.

Rapidez del algoritmo. Los resultados de la medición de rapidez son desplegados en forma numérica en el cuadro 3.2 y en forma gráfica en la figura 3.3. Con la imagen de 393216 píxeles, sin comunicación se logra una rapidez muy cercana al ideal y con 12 procesadores es 8.98 veces más rápido que el algoritmo secuencial —p_1—. Con comunicación, alcanza su máxima velocidad con 7, *i.e.*, es 3.02 veces más rápido que el algoritmo secuencial y a partir este punto la rapidez empieza a disminuir; puede pensarse

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	393216	1	1.48	2.16	2.81	3.57	4.37	5.09	5.78	6.56	7.50	8.32	8.98
	1572864	1	1.45	2.13	2.71	3.47	4.21	4.99	5.72	6.48	7.16	7.77	8.41
	3538944	1	1.45	2.14	2.73	3.50	4.25	4.98	5.62	6.35	7.16	7.91	8.33
	6291456	1	1.44	2.09	2.68	3.45	4.20	4.88	5.52	6.28	6.99	7.65	8.20
Con	393216	1	1.31	1.91	1.99	2.44	2.88	3.02	3.00	2.98	2.23	1.82	1.86
	1572864	1	1.30	1.90	2.13	2.51	2.86	3.44	3.66	3.65	3.73	3.66	3.62
	3538944	1	1.32	1.91	2.28	2.75	3.16	3.64	3.80	3.88	4.10	4.08	4.24
	6291456	1	1.33	1.89	2.30	2.77	3.21	3.66	4.00	4.21	4.51	4.79	4.71

Cuadro 3.2: *Rapidez alcanzada en la ejecución paralela del KLT.*

por lo tanto en liberar hasta 5 procesadores para otras tareas. Con la imagen de 1572864 píxeles, sin comunicación el paralelo es 8.41 veces más rápido con 12 procesadores que el secuencial. Con comunicación es 3.73 veces más rápido con 10 procesadores, sin embargo con 7 se tiene una rapidez similar, permitiendo poder liberar 5. Para la imagen de 3538944, sin comunicación se logra una rapidez con 12 procesadores de 8.33 veces el secuencial y con comunicación la máxima velocidad es también con 12 procesadores

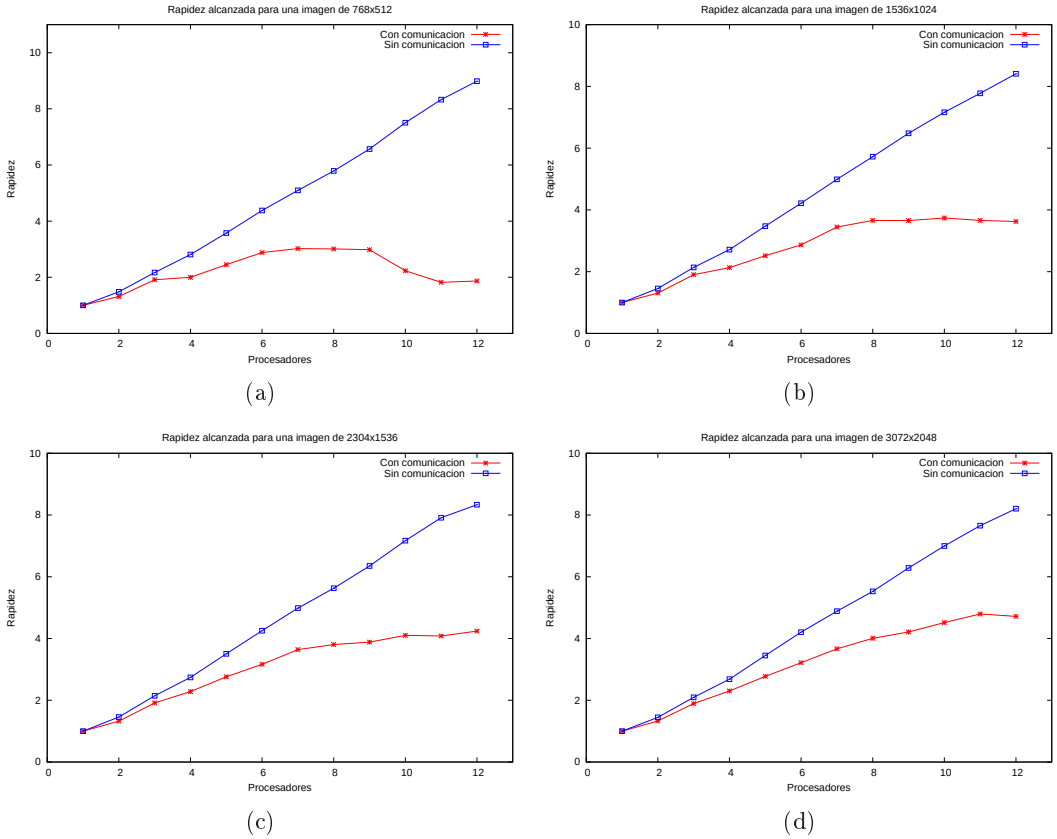


Figura 3.3: *Rapidez alcanzada del algoritmo paralelo KLT.*

—4.24—, pero se alcanza una velocidad similar con 10 procesadores, permitiendo liberar hasta dos procesadores. Finalmente para la imagen de 6291456 píxeles, sin comunicación la máxima velocidad se obtiene con 12 procesadores —8.20— y con comunicación con 11 —4.79—, pero se pueden liberar 2 procesadores ya que una rapidez similar se logra con 10.

Eficiencia del algoritmo. Los resultados de la eficiencia se muestran en el cuadro 3.3 y en la figura 3.4. Sin comunicación el algoritmo resulta estar en la mayoría de las veces

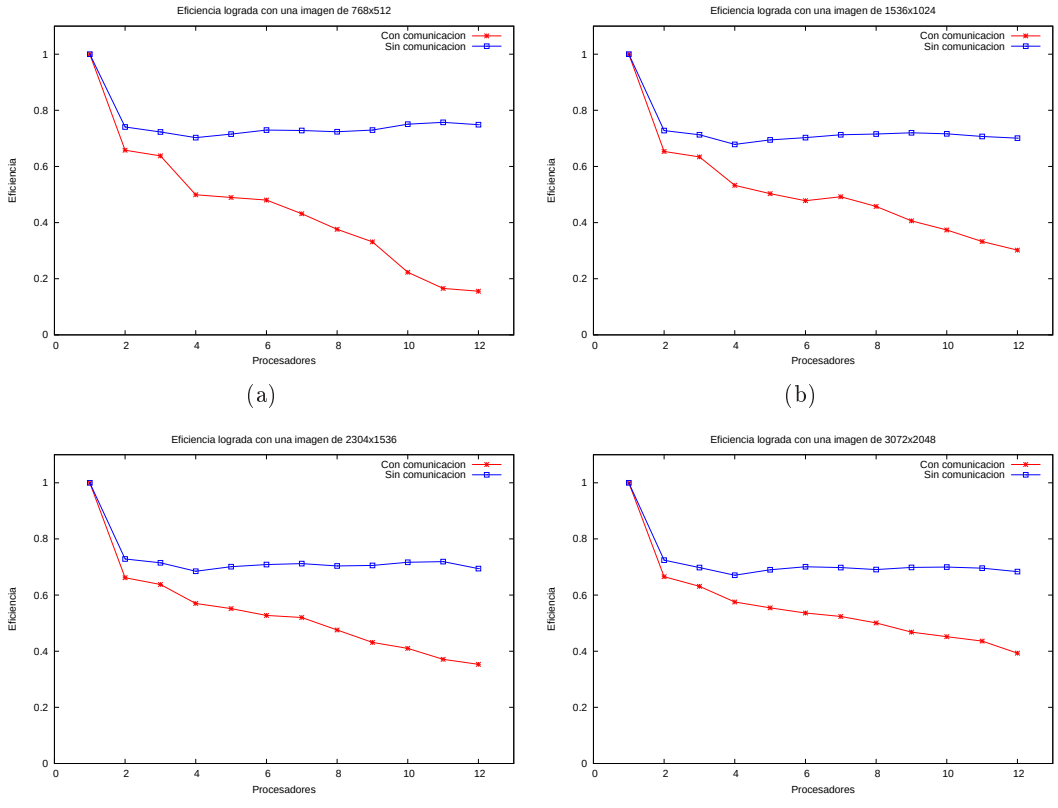


Figura 3.4: *Eficiencia del algoritmo paralelo KLT.*

por arriba del 70 % de eficiencia, solamente con las imágenes de 1572864 y 6291456, la eficiencia cae a un 67 % con 4 procesadores. En general se mantienen trabajando todos los procesadores en aproximadamente 70 %. Con comunicación el panorama es diferente, en general, la eficiencia se ve afectada mientras más procesadores se agreguen

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	393216	1	0.74	0.72	0.70	0.71	0.72	0.72	0.72	0.72	0.75	0.75	0.74
	1572864	1	0.72	0.71	0.67	0.69	0.70	0.71	0.71	0.72	0.71	0.70	0.70
	3538944	1	0.72	0.71	0.68	0.70	0.70	0.71	0.70	0.70	0.71	0.71	0.69
	6291456	1	0.72	0.69	0.67	0.69	0.70	0.69	0.69	0.69	0.69	0.69	0.68
Con	393216	1	0.65	0.63	0.49	0.48	0.48	0.43	0.37	0.33	0.22	0.16	0.15
	1572864	1	0.65	0.63	0.53	0.50	0.47	0.49	0.45	0.40	0.37	0.33	0.30
	3538944	1	0.66	0.63	0.57	0.55	0.52	0.52	0.47	0.43	0.41	0.37	0.35
	6291456	1	0.66	0.63	0.57	0.55	0.53	0.52	0.50	0.46	0.45	0.43	0.39

Cuadro 3.3: *Eficiencia lograda en la ejecución paralela del KLT.*

a la ejecución.

Costo del algoritmo. Los resultados de costo implicado en la ejecución del algoritmo 3.2, se muestran en el cuadro 3.4 y se despliegan en la figura 3.5. Sin comunicación

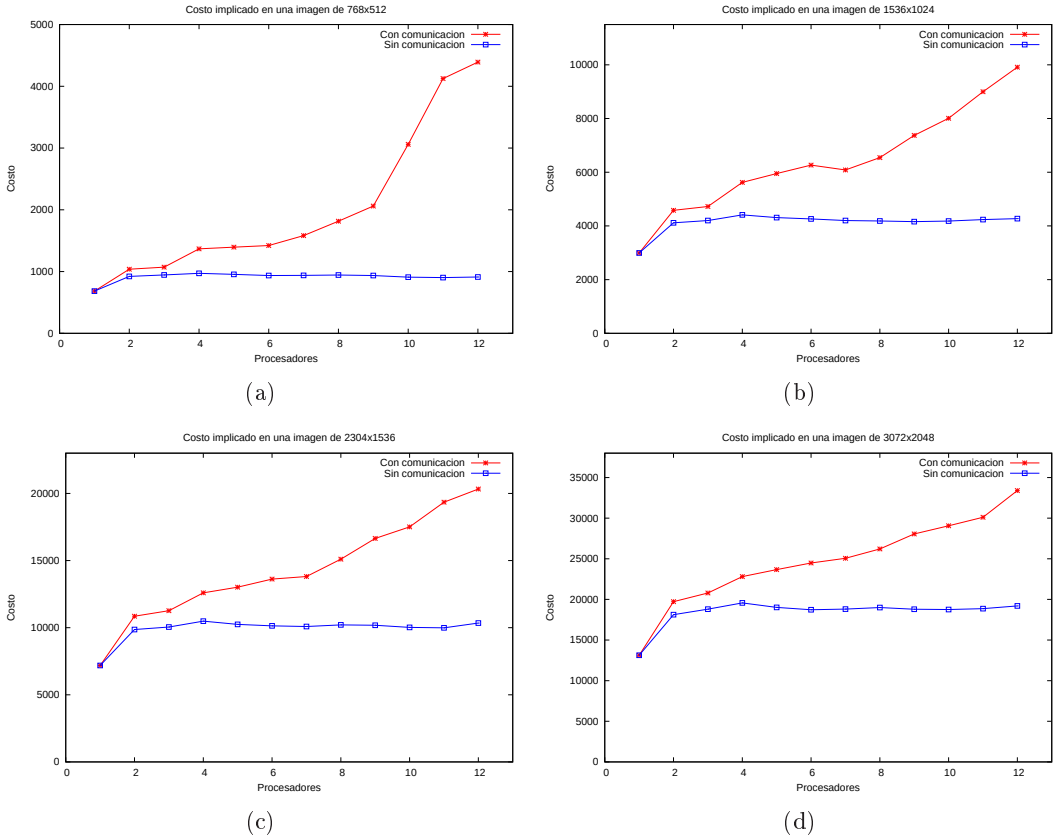


Figura 3.5: *Costo en el algoritmo paralelo KLT.*

en general con menos del doble de esfuerzo se ejecuta el algoritmo paralelo KLT. Sin embargo con comunicación, con cada procesador que se agrega a la ejecución en paralelo, se incrementa el costo en una manera aproximadamente lineal a excepción de cuando se

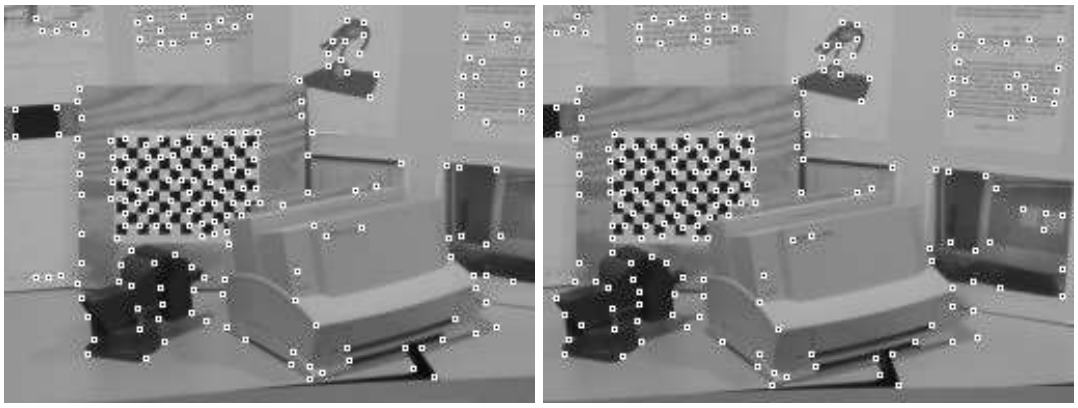
Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	393216	683	922	945	972	955	936	938	944	936	910	902	912
	1572864	2994	4114	4200	4412	4310	4260	4200	4184	4158	4180	4235	4272
	3538944	7183	9860	10047	10488	10245	10134	10087	10208	10179	10020	9988	10344
	6291456	13126	18122	18804	19568	19020	18726	18809	19000	18792	18760	18865	19200
Con	393216	683	1038	1071	1368	1395	1422	1582	1816	2061	3060	4125	4392
	1572864	2994	4582	4722	5620	5950	6264	6083	6544	7371	8010	8998	9912
	3538944	7183	10852	11262	12596	13015	13620	13811	15104	16650	17510	19349	20328
	6291456	13126	19724	20793	22812	23670	24480	25060	26208	28053	29060	30107	33396

Cuadro 3.4: *Costo implicado en el algoritmo paralelo KLT.*

probó la imagen más pequeña, en la cual hay un incremento del costo muy marcado a partir de 9 procesadores.

3.3. Discusión de resultados.

La implementación del algoritmo secuencial, fue exitosa, y las esquinas obtenidas en una imagen, fueron consistentes o muy cercanas con las esquinas que se obtienen de otra imagen registrada de la misma escena pero con un movimiento pequeño en la cámara, que es lo que se deseaba para asignar correspondencias en la etapa siguiente del proceso de calibración —ver figura 3.6—, tratando de no registrar demasiadas falsas correspondencias; sin embargo, está fuera del alcance de este trabajo de tesis la comparación de dicho algoritmo con otros detectores de esquinas. Por otra parte, la implementación en



(a) (b)

Figura 3.6: *Esquinas encontradas en un par de imágenes tomadas de la misma escena con un desplazamiento pequeño. Los valores utilizados son* $\lambda_{min} = 0$ *y* $r = 5$

paralelo del detector de esquinas KLT, fue satisfactoria, obteniendo los mismos resultados que el algoritmo secuencial, de tal forma se consiguen las mismas entradas para

la etapa siguiente del proceso de calibración. Se puede deducir que el comportamiento del algoritmo paralelo mejora en cuanto mayor sea el tamaño de la imagen a procesar. De los resultados de la sección 3.2, aunque se reducen los tiempos de ejecución, hay que tener en cuenta, la rapidez del algoritmo, la eficiencia de los procesadores y el costo incurrido de tal forma que se pueda administrar de manera razonable los recursos de la máquina paralela. Sin embargo no hay una señal clara de cuantos procesadores usar con este algoritmo para alcanzar una rapidez buena, con una eficiencia de los procesadores aceptable y un costo no muy elevado, por lo que durante el transcurso de lo siguientes dos capítulos se tomará una decisión de como agrupar los procesadores con ese fin para el algoritmo de calibración que se propondrá.

Capítulo 4

Correspondencia entre imágenes.

Dadas dos imágenes de una escena tomadas desde diferentes posiciones, el problema de *correspondencia consiste en establecer qué punto en una imagen corresponde a cuál punto en la otra imagen, en el sentido de que se trata de un mismo punto 3D proyectado en las imágenes* [Ma et al., 2003]. El objetivo del capítulo anterior y del actual, es extraer las primitivas geométricas de las mediciones fotométricas —imágenes—, y relacionarlas a través de las diferentes vistas, de tal forma que posteriormente permita concentrarse en la geometría del proceso de reconstrucción. En el presente capítulo se proporciona un breve análisis en la formulación del problema de la correspondencia y su solución con un enfoque de análisis de regiones locales en las imágenes.

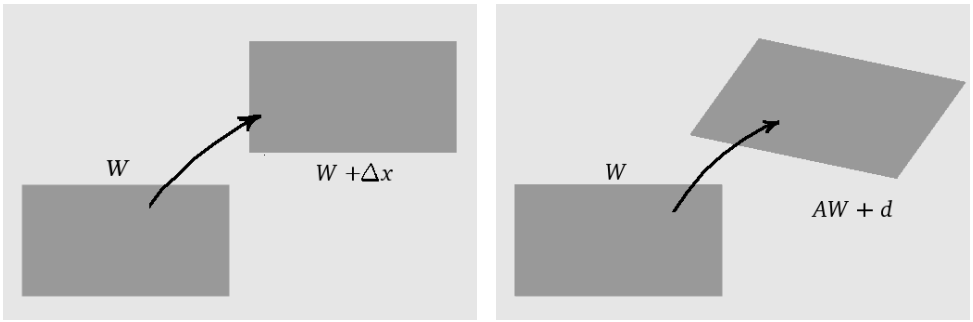
4.1. Establecimiento del problema de correspondencia.

Dadas dos cámaras $\mathbf{P}$ y $\mathbf{P}'$ —ver apéndice A— y un punto tridimensional $\mathbf{X}$, la cámara $\mathbf{P}$ registra a $\mathbf{X}$ en una imagen como $\mathbf{x} \sim \mathbf{K}[\mathbf{R}^\top - \mathbf{R}^\top\mathbf{t}]\mathbf{X}$ y $\mathbf{P}'$ en otra imagen como $\mathbf{x}' \sim \mathbf{K}'[\mathbf{R}'^\top - \mathbf{R}'^\top\mathbf{t}']\mathbf{X}$ de donde $\mathbf{x} \leftrightarrow \mathbf{x}'$ de acuerdo a la transformación $(\mathbf{R}', \mathbf{t}')$ [Ma et al., 2003]. Establecer correspondencias con un enfoque global intentando encontrar la transformación $(\mathbf{R}', \mathbf{t}')$ es una tarea imposible de realizar. Las siguientes subsecciones se concentran en un enfoque local. De igual manera se explora superficialmente el hecho de que aunque conozcamos la posición de las correspondencias, esto no implica que las intensidades de las proyecciones en $\mathbf{x}$ y $\mathbf{x}'$ sean exactamente iguales.

4.1.1. Modelos de deformación local.

Una solución a este problema es elegir una clase de deformación simple y restringir nuestra atención a regiones en las imágenes como si estuvieran bajo tal deformación. Dicha deformación ocurre en el dominio de la imagen en una ventana W centrada en $\mathbf{x}$ y en los valores de intensidad $I(\tilde{\mathbf{x}})$ con $\tilde{\mathbf{x}} \in W$. En los siguientes párrafos se examinan tres casos de deformación local comenzando desde el más simple.

Modelo de deformación traslacional. Puede entenderse como uno donde cada punto en la ventana W sufre exactamente el mismo movimiento, *i.e.*, un desplazamiento $\Delta\mathbf{x} = \mathbf{x_{i'}} - \mathbf{x_i}$ constante o $h(\tilde{\mathbf{x}}) = \tilde{\mathbf{x}} + \Delta\mathbf{x}$, $\forall\tilde{\mathbf{x}} \in W$. Este tipo de movimiento es válido solamente para porciones de la escena que son planas y cercanamente paralelas a la imagen y tienen un movimiento aproximadamente paralelo a ella. Aunque muy simple, es el núcleo de la mayoría de los algoritmos de apareamiento ó seguimiento —matching ó tracking— debido a su simplicidad y eficiencia de la implementación resultante. Un ejemplo de este tipo de movimiento es el que se emplea en el trabajo presentado por Zhang et al. [1995].



(a) Traslacional (b) Afín

Figura 4.1: *Dos tipos básicos de deformación local.*

Modelo de deformación afín. En este tipo de deformación, los puntos en la ventana W no sufren el mismo movimiento, en lugar de eso, el movimiento de cada punto depende linealmente de su lugar más un desplazamiento constante, *i.e.*, $h(\tilde{\mathbf{x}}) = \mathbf{A}\tilde{\mathbf{x}} + \mathbf{d}$, $\forall\tilde{\mathbf{x}} \in W$, con $\mathbf{A} \in \mathbb{R}^{2\times 2}$ y $\mathbf{d} \in \mathbb{R}^2$. El modelo de deformación traslacional es un caso particular de la deformación afín considerando $\mathbf{A} = \mathbf{I}$. Es una buena aproximación para regiones

planas pequeñas y cercanamente paralelas a la imagen, bajo una traslación arbitraria, rotación alrededor del eje óptico y una modesta rotación alrededor de un eje diferente al eje óptico. Este tipo de deformación es el empleado en el apareamiento de puntos presentado por Tuytelaars and Gool [2004].

En la figura 4.1 se muestra gráficamente el movimiento en las dos deformaciones mencionadas anteriormente.

Modelo de deformación proyectiva. Es una generalización del modelo afín pero considerando trasformaciones lineales en las coordenadas homogéneas, *i.e.*, $h(\tilde{\mathbf{x}}) \sim \mathbf{H}\tilde{\mathbf{x}}$, $\forall \tilde{\mathbf{x}} \in W$, donde $\mathbf{H} \in \mathbb{R}^{3\times 3}$ está definida hasta un factor de escala. Este modelo captura movimientos de cuerpos rígidos arbitrarios de regiones planas de mayor tamaño que en el modelo de deformación afín. El hecho de que cualquier superficie suave pueda ser aproximada arbitrariamente bien por una colección de planos habilita a este modelo a ser apropiado en cualquier parte de la imagen donde no haya discontinuidades ó bordes excluidos. Ejemplo del uso de este modelo de deformación puede ser revisado en el trabajo presentado por Tell and Carlsson [2000].

Considerando lo anterior, un modelo para la deformación de regiones entre imágenes de la misma escena está dado por

$$I(\mathbf{x}) = I'(h(\mathbf{x})), \quad \forall \mathbf{x} \in W \tag{4.1}$$

donde h representa un tipo de deformación traslacional, afín ó proyectiva.

4.1.2. Transformación de los valores de intensidad.

Bajo el conocimiento de que las imágenes son arreglos de números positivos que representan intensidad de luz incidiendo en un sensor en un lapso, supongamos que deseamos establecer la correspondencia para un píxel $\mathbf{x}$ en la imagen con una intensidad $I(\mathbf{x})$ en la imagen I'. Si el punto tridimensional $\mathbf{X}$ se proyectó a la imagen I' en $\mathbf{x}'$, es ingenuo pensar que el valor de intensidad registrado en I' sea exactamente igual al valor registrado en I, debido a que la formación de las imágenes es un proceso intrínsecamente sujeto a incertidumbre. También existe la posibilidad de encontrar píxeles con intensidades iguales a la intensidad registrada en $\mathbf{x}$, pero que ninguno resulte ser la correspondencia correcta $\mathbf{x}'$.

Una primera aproximación a las diferencias de las intensidades en los píxeles correspondientes, es codificarlas en un termino de ruido aditivo η que se comporta como una variable aleatoria Gaussiana interpretando a las múltiples fuentes independientes de incertidumbre. Una solución a la necesidad de adquisición de imágenes con niveles de intensidades apropiadas fue presentado por Debevec and Malik [1997] en el cual se construyen mapas de irradianza para corregir la intensidad de un píxel dado. La ecuación (4.1) es entonces modificada de acuerdo al término aditivo η como

$$I(\mathbf{x}) = I'(h(\mathbf{x})) + \eta(h(\mathbf{x})) \tag{4.2}$$

y de acuerdo a este modelo, la correspondencia se formulará matemáticamente como la solución de un problema de optimización, eligiendo transformaciones $\hat{h}$ que minimizan el efecto del ruido de acuerdo a algún criterio sujeto a la ecuación anterior sobre una región —ventana— W, i.e.

$$\hat{h} = \arg\min_{h} \sum_{\tilde{\mathbf{x}} \in W} \| \eta(h(\tilde{\mathbf{x}})) \|^2 = \arg\min_{h} \sum_{\tilde{\mathbf{x}} \in W} \| I(\tilde{\mathbf{x}}) - I'(h(\tilde{\mathbf{x}})) \|^2 \tag{4.3}$$

eligiendo como medida de discrepancia la norma del error aditivo. En las siguientes subsecciones de dan dos criterios para minimizar la ecuación (4.3) considerando una deformación traslacional en las regiones locales de las imágenes.

4.1.3. Criterio de Suma de Diferencias Cuadradas, SSD.

El criterio de Suma de Diferencias Cuadradas —*Sum-of-Squared-Differences, SSD*— es muy sencillo rápido de implementar. Si consideramos una ventana W en una imagen I y su correspondiente región $T(W)$ en la imagen I', entonces la diferencia o disimilitud entre las imágenes basada en SSD esta dada por

$$D = \int \int_{(x,y) \in W} [I(x,y) - I'(T(x,y))]^2 w(x,y) dx dy, \tag{4.4}$$

donde w es una función de peso, normalmente igual a uno o a una Gaussiana. Esta medida de disimilitud se puede expresar de manera discreta como

$$D = \sum_{j=-n}^{n} \sum_{k=-n}^{n} (I(x-j, y-k) - I'(x'-j, y'-k))^2 \tag{4.5}$$

donde $(2n+1)\times(2n+1)$ es el tamaño de la ventana bajo consideración. Como se aprecia, simplemente se calcula la diferencia de intensidades de las vecindades de los píxeles que se quieren comparar con el objetivo de encontrar la región $T(W)$ que minimice el criterio SSD con respecto a la región W, lo que significa que la región seleccionada es la menos diferente minimizando consecuentemente la ecuación (4.3). El valor en D es positivo y la región $T(W)$ que reporta el valor mínimo de error, es la que se considera como región deformada de W.

4.1.4. Criterio de Correlación Cruzada de media Cero Normalizada, ZNCC

A pesar de que SSD permite una solución de mínimos cuadrados, tiene la desventaja de ser muy sensible a cambios en las intensidades de las imágenes provocados por condiciones cambiantes de luz sobre el tiempo. Una mejor opción es la Correlación Cruzada de media Cero Normalizada —*Zero mean Normalized Cross Correlation, ZNCC*— el cual permite asociar similitudes entre regiones de las imágenes y cuya expresión está dada por

$$S = \frac{\int\int_{(x,y)\in W}\left[\left(I(x,y)-\bar{I}\right)\cdot\left(I'(T(x,y))-\bar{I}'\right)\right]w(x,y)dxdy}{\sqrt{\int\int_{(x,y)\in W}\left[I(x,y)-\bar{I}\right]^2 w(x,y)dxdy\cdot\int\int_{(x,y)\in W}\left[I'(T(x,y))-\bar{I}'\right]^2 w(x,y)dxdy}}, \tag{4.6}$$

con $\bar{I}$ el promedio de la intensidad de la imagen en el área considerada y w una función de peso como en el criterio SSD. Su forma discreta es

$$S = \frac{\sum_{j=-n}^{n}\sum_{k=-n}^{n}(I(x-j,y-k)-\bar{I})(I'(x'-j,y'-k)-\bar{I}')}{\sqrt{\sum_{j=-n}^{n}\sum_{k=-n}^{n}(I(x-j,y-k)-\bar{I})^2\cdot(I'(x'-j,y'-k)-\bar{I}')^2}}. \tag{4.7}$$

Este criterio de minimización es más costoso que el criterio SSD ya que es necesario el computo de la media y desviación estándar de las intensidades de los píxeles en la región W. A diferencia de SSD, los valores devueltos por este criterio están en el rango [-1,1], donde las regiones $T(W)$ más diferentes a W reportan valores S muy cercanos a -1 y las más parecidas valores muy cercanos a 1 y por lo tanto el valor más cercano a 1 representa a la región $T(W)$ que minimiza a la ecuación (4.3).

Estos y otros criterios de minimización para la ecuación (4.3) pueden ser investigados ampliamente en [Ma et al., 2003].

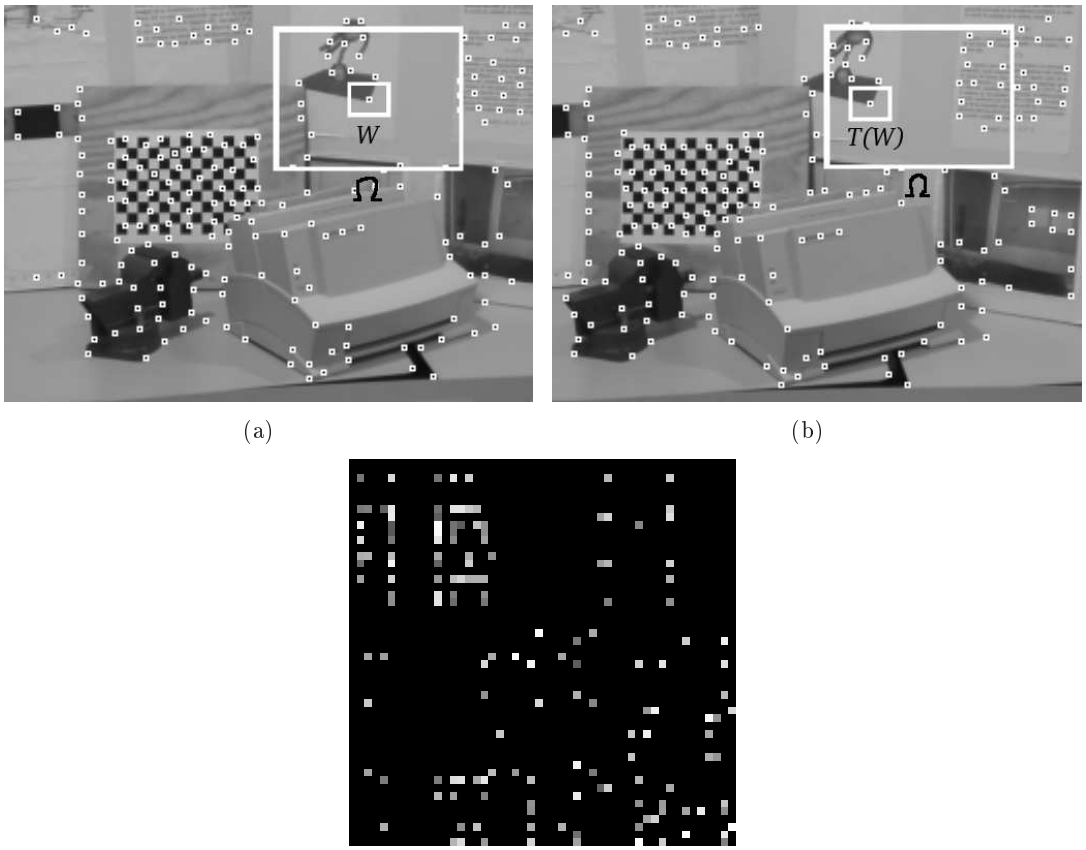
4.2. Algoritmo de correspondencias.

En esta sección se describe un algoritmo muy simple de apareamiento de puntos característicos para líneas base pequeñas —ver figura 2.4(a)— apoyado en el modelo de deformación traslacional. Ma et al. [2003] muestra alternativas para líneas base grandes sustentadas en el modelo de deformación afín.

4.2.1. Algoritmo secuencial de correspondencia.

Las listas de esquinas devueltas por el operador KLT —ver sección 3.1—, son usadas en la búsqueda de correspondencias entre pares de imágenes. Una vez que un punto característico es seleccionado, el objetivo es encontrar su correspondencia en la otra imagen basado en la optimización de alguna una de las ecuaciones (4.5) ó (4.7). A continuación se describe el enfoque novel de asignación de correspondencias considerado en este documento.

Cada elemento de una lista L se compara contra aquellos elementos de otra lista L' que se encuentren dentro una región de búsqueda Ω, que puede ser la imagen completa ó una región de tamaño suficiente como se muestra en las figuras 4.2(a) y 4.2(b); el resultado de dicha comparación se va almacenando en un arreglo bidimensional o mapa como el de la figura 4.2(c), en donde cada renglón representa una de las esquinas en la primera lista, y cada columna a una esquina de la segunda lista. Cada celda contiene el resultado de la comparación con SSD ó ZNCC entre elementos de L y L'. El mapa puede ser una matriz dispersa cuando Ω no cubre a la imagen completa, pues no todos los elementos de las listas se comparan entre si. El mapa así construido tiene que ser analizado con el fin de asignar correspondencias únicas entre las esquinas en base a una las consideraciones del algoritmo 4.1, en donde la asignación de una fila j con una columna k significa la asignación $c_j \leftrightarrow c_k$ para $c_j \in L$ y $c_k \in L'$. En el algoritmo 4.1, el umbral t así como la interpretación de en que momento v_{jk} se trata de un valor óptimo, depende del criterio de minimización de la ecuación (4.3). Para efectos de esta tesis, si SSD es usado, un valor óptimo es interpretado como el mínimo de la fila ó columna y un umbral t estrictamente positivo; en cuanto a ZNCC, un valor óptimo es el más cercano a 1 y el umbral $t \in [-1, 1]$. El algoritmo puede ser usado en cualquier situación en la que se tenga definido un criterio de optimización. En el algoritmo 4.2 se describe el proceso



(a) (b)

(c)

Figura 4.2: *Imágenes de las listas I en (a) e I' en (b) mostrando las regiones locales W y $T(W)$ las cuales minimizan la ecuación (4.3) dentro de una región de búsqueda Ω. En (c) se muestra un ejemplo del mapa de valores que se analiza con el algoritmo de asignación de correspondencias —algoritmo 4.1—.*

Algoritmo 4.1: *Algoritmo de asignación de correspondencias.*

Objetivo: *Dado un mapa de valores construido con algún criterio de optimización de la ecuación (4.3) y un umbral t, asignar correspondencias únicas.*
Algoritmo: Para cada fila del mapa de valores:

i Sea $v_{jk} > t$ el valor óptimo de la fila j con columna k, si v_{jk} es el valor óptimo de la columna k, entonces se hace la asignación $j \leftrightarrow k$ y se deshabilita la fila j y la columna k y termina la iteración, si no entonces:

ii Sea $v_{j'k} > t$ el valor óptimo de la columna k el cual se encuentra en la fila j', donde $j \neq j'$. Si $v_{j'k}$ es el valor óptimo de la fila j' entonces se hace la asignación $j' \leftrightarrow k$ y se deshabilita la fila j' y la columna k, y termina la iteración. Si este no es el caso entonces:

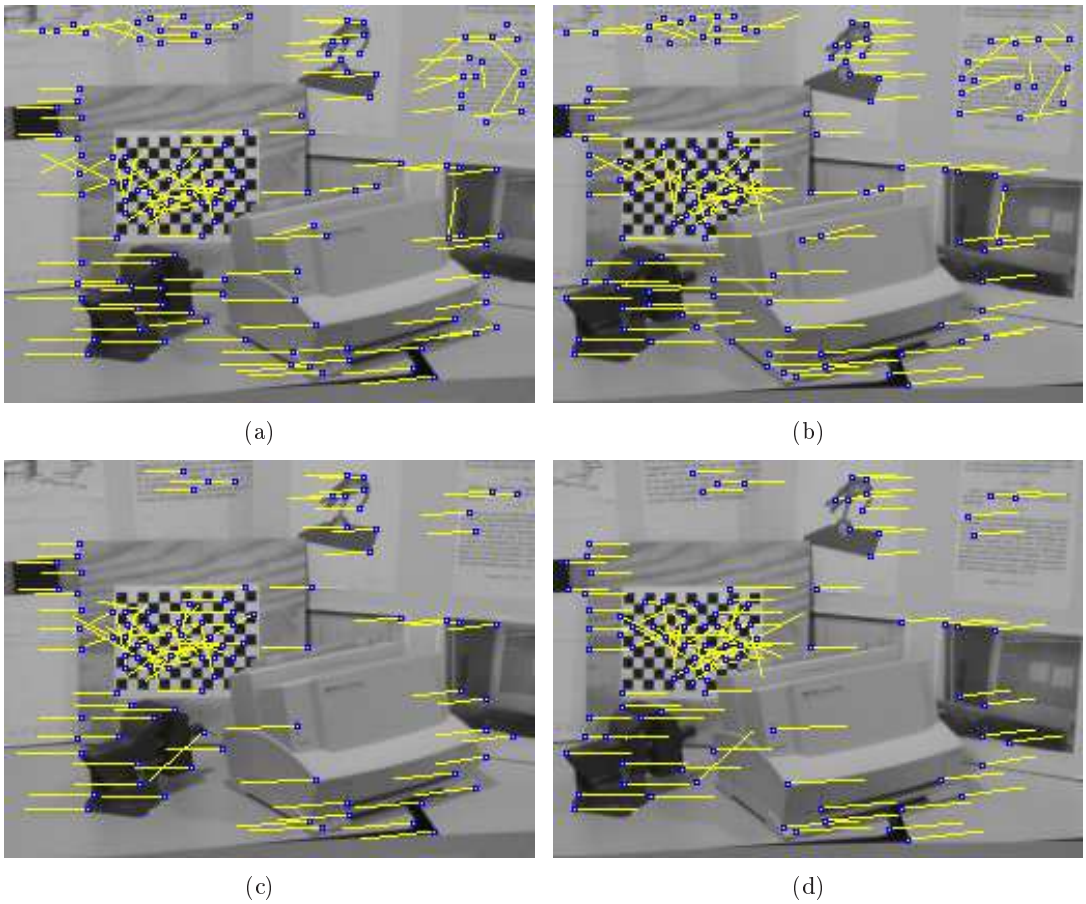
iii Se hace la asignación $j \leftrightarrow k$ y se deshabilita la fila j y la columna k.

Algoritmo 4.2: *Algoritmo de asignación de correspondencias a partir de un par de listas de esquinas.*

Objetivo: *Dadas un par de listas L y L', asignar correspondencias entre las imágenes que poseen dichas listas de esquinas*

Algoritmo:

- Para cada $c_j \in L$
  - i Seleccionar a todos los elementos de la lista L' que se encuentren dentro de una región de búsqueda Ω, resultando en una $L'_{c_j} \subseteq L'$.
  - ii Para cada $c_k \in L'_{c_j}$ aplicar alguna de las ecuaciones (4.5) ó (4.7) contra c_j y una vecindad W, y asignar el valor a M_{jk}.
- Ejecutar el algoritmo 4.1 con M y un umbral t.



(a) (b)

(c) (d)

Figura 4.3: *Imágenes izquierda y derecha relacionadas SSD con un umbral t=20,000 en (a) y (b) y en (c) y (d) con ZNCC a un umbral $t = 0.9$.*

de asignación de correspondencias a partir de un par de listas L y L'. En la figura 4.3 se muestra cómo un par de imágenes de una misma escena, han sido relacionadas de acuerdo a las esquinas contenidas en ellas. En ella se puede ver ejemplos de aplicar los dos diferentes criterios de optimización explicados anteriormente.

4.2.2. Algoritmo paralelo de correspondencias.

Usar SSD, es más rápido ZNCC, sin embargo el tiempo total de procesamiento ocupado por SSD en la construcción del mapa de valores ocupa mas del 80 % en la ejecución del algoritmo de correspondencias previamente presentado, por su parte ZNCC, necesita más del 90 % del tiempo de ejecución. Basado en lo anterior, el algoritmo paralelo que se formulará distribuirá las tareas de construcción del mapa de valores, *i.e.*, se procederá a distribuir los pasos i y ii del algoritmo secuencial —algoritmo 4.2—. La versión paralela propuesta se presenta en el algoritmo 4.3.

Algoritmo 4.3: *Algoritmo distribuido de asignación de correspondencias.*

Objetivo: *Dadas un par de listas L y L' asignar correspondencias entre las imágenes de dichas listas de manera paralela.*

Algoritmo:

i Fragmentar la lista L en L_p entre el número de procesadores disponibles de tal forma que $\bigcup L_p = L$ y $\bigcap L_p = \emptyset$ y poner a disponibilidad de todos los procesadores a la lista L'.

ii Cada procesador construye un submapa M_p con el fragmento L_p la lista L'.

iii El procesador principal reúne todos los submapas en el mismo orden en que son distribuidas las sublistas L_p y los une en un solo mapa M

iv El procesador principal ejecuta el algoritmo 4.1 con M y un umbral t.

4.3. Experimentos.

Ejecuciones con tres pares de listas de esquinas de 500, 2500, y 5000 elementos se llevaron a cabo para probar las mejoras obtenidas con el algoritmo paralelo en esta etapa del proceso de calibración. A diferencia de la etapa anterior no importa el tamaño de las imágenes, sino el tamaño de las listas, de la región de búsqueda Ω y del tamaño de la vecindad W. Las imágenes de las que se extrajeron las listas de esquinas son de 1024×768 píxeles, se eligió una región de búsqueda Ω igual a $\frac{1}{3}$ y a su vez W igual a $\frac{1}{30}$ de la dimensión de las imágenes y se usó ZNCC como criterio de minimización debido a

que es el que más costo computacional requiere. Los resultados de los experimentos se describen en la siguiente subsección. De igual manera que en el capítulo anterior, cada combinación tamaño-procesadores se ejecuto 30 veces y los promedios son los datos que se analizan. Se hace un análisis de los tiempos en la asignación de correspondencias de acuerdo al número de procesadores utilizados en las pruebas así como un análisis de rapidez, eficiencia y costo implicados en la ejecución del algoritmo paralelo.

Tiempo de ejecución. En el cuadro 4.1 y en la figura 4.4 se enseñan en milisegundos los tiempos requeridos para asignación de esquinas con las condiciones ya descritas.

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	346	180	124	99	82	70	63	56	52	48	44	42
	2500	7923	4009	2708	2140	1713	1434	1237	1106	984	890	814	751
	5000	32629	16450	11061	8535	6835	5707	4914	4322	3846	3477	3174	2935
Con	500	346	186	129	126	106	94	86	64	61	55	54	52
	2500	7923	4181	2890	2392	1969	1722	1513	1431	1336	1235	1155	1040
	5000	32629	16859	11449	9251	7562	6554	5769	5148	4692	4315	4024	3789

Cuadro 4.1: *Tiempos en milisegundos incurridos en la ejecución del algoritmo de asignación en paralelo —algoritmo 4.3, con listas de 500, 2500 y 5000 elementos con y sin comunicación entre los procesadores.*

De igual manera que en el capítulo anterior, sin comunicación el algoritmo presenta el comportamiento deseado de reducir el tiempo de ejecución si muchos procesadores son empleados. Con comunicación también existe un decremento del tiempo en general. Sin embargo con las listas de 500 esquinas, una pequeña perturbación se presentó en la tendencia de los datos obtenidos y este comportamiento fue persistente sin importar el orden en que se agregaron los procesadores a las pruebas. En los otros conjuntos de listas, a primera vista no aparece tal perturbación, pero si existe, en los análisis de rapidez se reportaran. Con los datos del cuadro anterior, cálculos de rapidez, eficiencia y costo son presentados en los siguientes párrafos.

Rapidez. La tabla 4.2 describe el resultado del cálculo de rapidez que el algoritmo puede alcanzar y una descripción gráfica se enseña en la figura 4.5. Con estos conjuntos de datos el algoritmo resulta ser cada vez más rápido con cada procesador que se agregue a la ejecución. La perturbación que aparece en las pruebas del conjunto de 500 elementos, no aparece en los otros conjuntos, eso se puede ver del comportamiento casi lineal del crecimiento de la rapidez del algoritmo. En esta etapa de calibración en paralelo, la

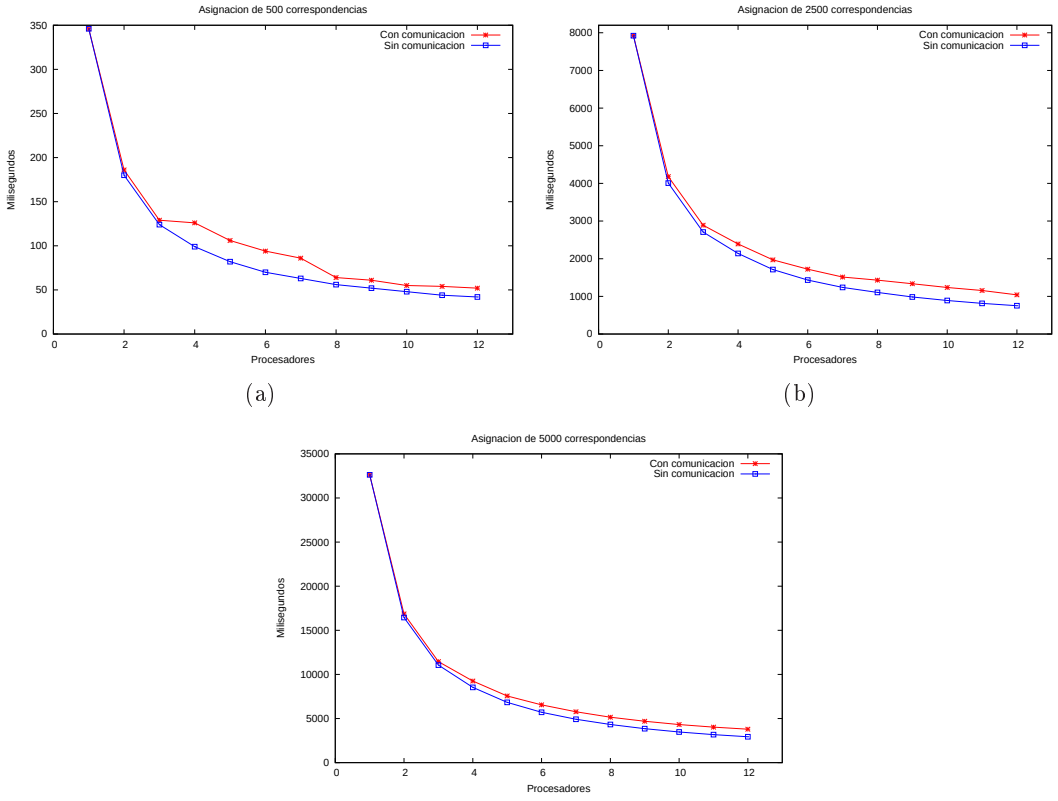


Figura 4.4: *Tiempos en milisegundos necesitados en la asignación de correspondencias.*

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	1	1.92	2.79	3.49	4.21	4.94	5.49	6.17	6.65	7.20	7.86	8.23
	2500	1	1.97	2.92	3.70	4.62	5.52	6.40	7.16	8.05	8.90	9.73	10.54
	5000	1	1.98	2.94	3.82	4.77	5.71	6.64	7.54	8.48	9.38	10.28	11.11
Con	500	1	1.86	2.68	2.74	3.26	3.68	4.02	5.40	5.67	6.29	6.40	6.65
	2500	1	1.89	2.74	3.31	4.02	4.60	5.23	5.53	5.93	6.41	6.85	7.61
	5000	1	1.93	2.84	3.52	4.31	4.97	5.65	6.33	6.95	7.56	8.10	8.61

Cuadro 4.2: *Rapidez lograda con el algoritmo paralelo de correspondencia.*

comunicación no influye demasiado debido a que solamente hay comunicación entre los procesadores al principio y al final del algoritmo paralelo cuando se distribuyen la lista L y cuando se recolecta el submapa M_p.

Eficiencia. El aprovechamiento de los recursos en esta etapa se ve reflejada en el análisis de eficiencia, cuyos datos se muestran en el cuadro 4.3 y su descripción gráfica en la figura 4.6. Como se puede ver, debido a que existe poca comunicación en el algoritmo paralelo de correspondencia, la eficiencia de los procesadores es en promedio del 70 %

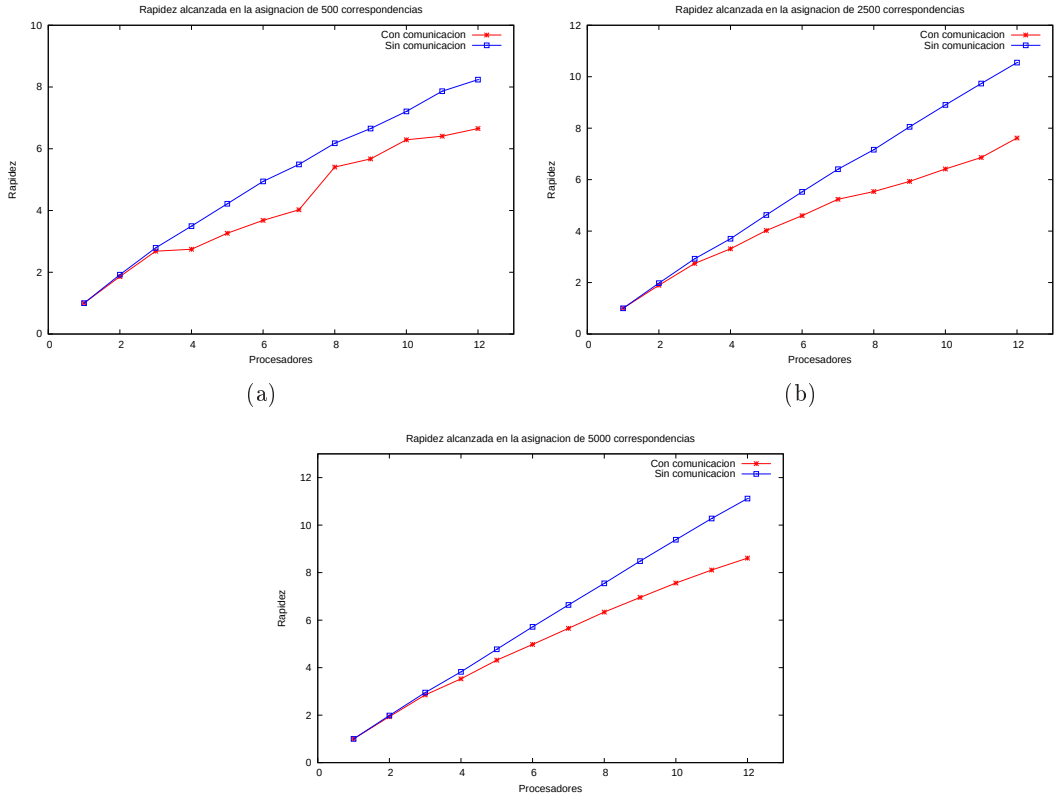


Figura 4.5: *Rapidez alcanzada con el algoritmo paralelo de correspondencia.*

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	1	0.96	0.93	0.87	0.84	0.82	0.78	0.77	0.73	0.72	0.71	0.68
	2500	1	0.98	0.97	0.92	0.92	0.92	0.91	0.89	0.89	0.89	0.88	0.87
	5000	1	0.99	0.98	0.95	0.95	0.95	0.94	0.94	0.94	0.93	0.93	0.92
Con	500	1	0.93	0.89	0.68	0.65	0.61	0.57	0.67	0.63	0.62	0.58	0.55
	2500	1	0.94	0.91	0.82	0.80	0.76	0.74	0.69	0.65	0.64	0.62	0.63
	5000	1	0.96	0.94	0.88	0.86	0.82	0.80	0.79	0.77	0.75	0.73	0.71

Cuadro 4.3: *Eficiencia adquirida con el algoritmo paralelo de correspondencias.*

para el conjunto de 500 datos, del 77 % para el de 2500 datos y del 83 % para el de 5000 elementos. Sin comunicación el algoritmo tiene un aprovechamiento promedio del 82 %, 92 % y 95 % respectivamente de los recursos de la máquina paralela. El aprovechamiento de los recursos de la máquina paralela con este algoritmo resulta ser mejor de lo que lo que lo hace el algoritmo paralelo descrito en el capitulo anterior.

Costo. Los cálculos en cuanto a costo en el que se incide se muestran en la tabla 4.4 y son gráficamente desplegados en la figura 4.7. Con y sin overhead, el esfuerzo requerido

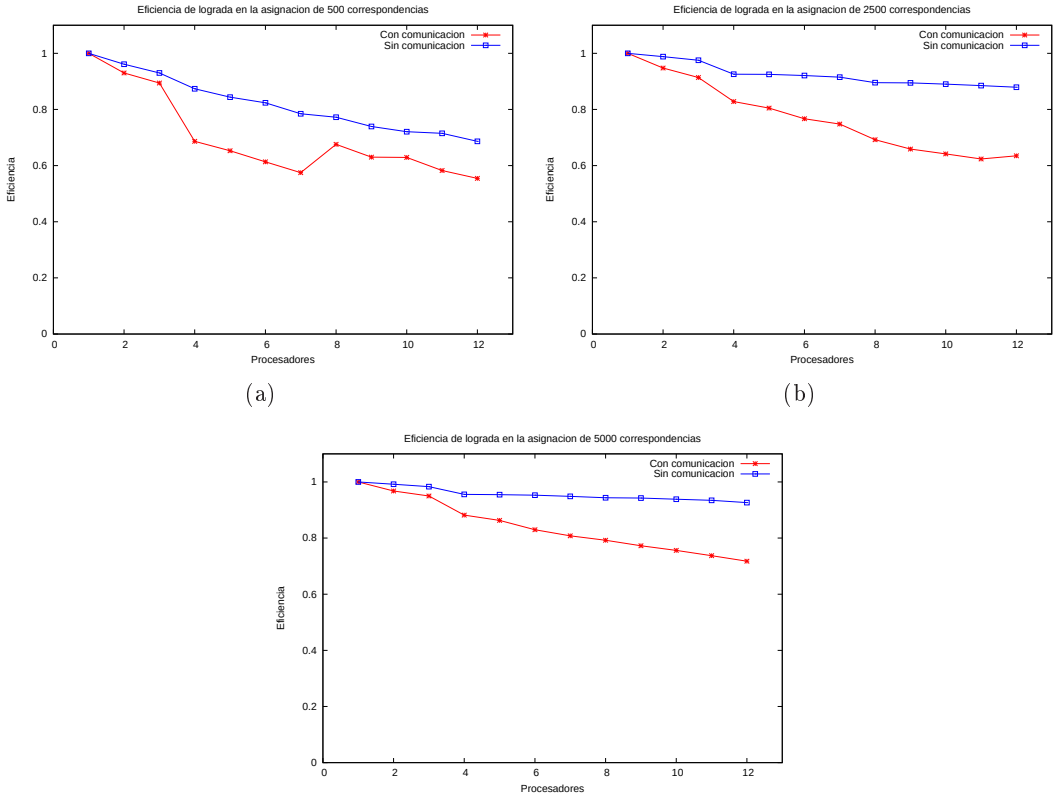


Figura 4.6: *Eficiencia lograda con el algoritmo paralelo de correspondencia.*

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	346	360	372	396	410	420	441	448	468	480	484	504
	2500	7923	8018	8124	8560	8565	8604	8659	8848	8856	8900	8954	9012
	5000	32629	32900	33183	34140	34175	34242	34398	34576	34614	34770	34914	35220
Con	500	346	372	387	504	530	564	602	512	549	550	594	624
	2500	7923	8362	8670	9568	9845	10332	10591	11448	12024	12350	12705	12480
	5000	32629	33718	34347	37004	37810	39324	40383	41184	42228	43150	44264	45468

Cuadro 4.4: *Costo involucrado en el algoritmo paralelo de correspondencias.*

para la ejecución del algoritmo paralelo de correspondencias, en ninguna combinación tamaño de lista-número de procesadores llega al doble del necesitado con el algoritmo secuencial.

4.4. Discusión de resultados.

La implementación de un algoritmo para buscar correspondencias en un par de imágenes cuya línea base es corta se logró en este capítulo. Se implementaron los dos criterios

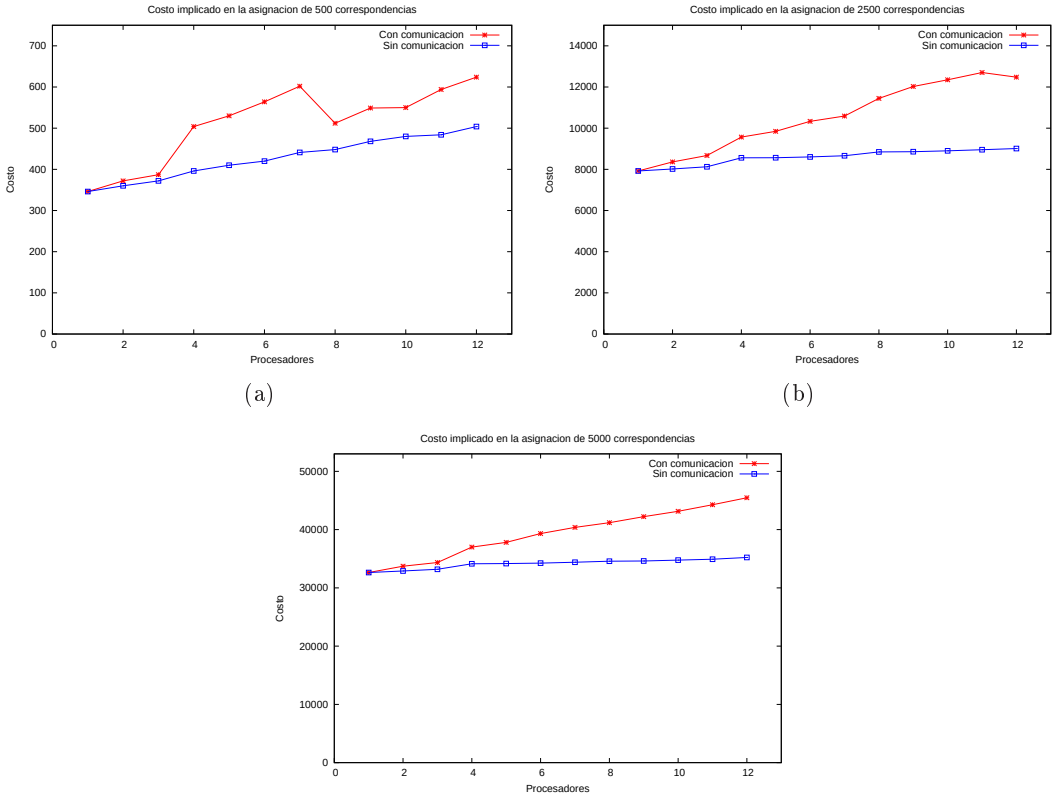


Figura 4.7: *Costo implicado en el algoritmo paralelo de correspondencia.*

de minimización de la ecuación de deformación de regiones entre las imágenes SSD y ZNCC, proporcionando las entradas a la siguiente etapa de calibración. En cuanto al algoritmo paralelo presentado, las salidas son las mismas que el algoritmo secuencial permitiendo que las entradas sean válidas para la siguiente etapa de calibración y poder seguir con la comparación con el algoritmo secuencial implementado como hasta ahora. Los análisis de tiempo, rapidez, eficiencia y costo que se reportan con la pruebas son muy aceptables.

Capítulo 5

Estimación robusta RANSAC.

Los algoritmos de estimación de modelos que se emplean en visión computacional usualmente trabajan bajo el supuesto que los datos de entrada a dichos algoritmos contienen una fuente de error las cuales siguen una distribución Gaussiana. Sin embargo, este supuesto no es siempre válido y los datos que se desvían de la distribución Gaussiana deben ser considerados como datos *atípicos* —outliers—.

De la etapa anterior del proceso de calibración, el conjunto de correspondencias obtenidas puede estar contaminado con falsas correspondencias si el umbral t en el algoritmo 4.1 es mal estimado. Una falsa correspondencia ocurre cuando la correspondencia en cuestión no representa al mismo punto 3D de la escena que fue capturada si no que las coordenadas en las imágenes son proyecciones de puntos 3D diferentes y esto es debido a que las primitivas son extraídas y correlacionadas con un análisis local entre las imágenes y por lo tanto propensas a ambigüedades. El uso de tal conjunto de correspondencias en el cálculo de la matriz fundamental, produciría una mala estimación de la geometría epipolar del par de cámaras que registró la escena. Es por lo tanto necesario seleccionar las correspondencias correctas, *i.e.*, los datos *típicos* —inliers— llevando a cabo una estimación robusta de la geometría epipolar, eliminando la mayoría de outliers posibles.

El algoritmo de estimación robusta que se usa en esta tesis, es el propuesto por Fischler and Bolles [1981], denominado RANSAC –RANdom SAmple Consensus–, que puede ser considerado como un algoritmo de optimización, ya que dada una función objetivo, determina el conjunto de parámetros que mejor describen al modelo [Lacey et al., 2000]. Sin embargo, existen varios métodos para el ajuste de modelos que pueden ser utilizados en la estimación de la geometría epipolar, *e.g.* Rousseeuw and Leroy [1987] mencionan

dos métodos, LMS — Least Median of Squares— y M-estimators.

5.1. RANSAC.

El principio del algoritmo se basa en tomar aleatoriamente N subconjuntos de tamaño s[1] del total de datos y calcular modelos con estos subconjuntos. Cada uno de estos modelos nos permite dividir el total de las datos en inliers y outliers y al final de haber calculado los N modelos se elige al que proporcione más inliers; en palabras de los autores –Fischler and Bolles [1981]:

El procedimiento RANSAC es opuesto a las técnicas convencionales de suavizado: en lugar de usar tantos datos como sea posible para obtener una solución inicial y entonces ir eliminando los datos inválidos, se usan pocos datos como sea posible y se engrandece este conjunto con datos consistentes, cuando sea posible.

La idea es muy simple y se describe en el algoritmo 5.1.

Algoritmo 5.1: *Algoritmo de estimación robusta RANSAC. Un mínimo de s elementos son requeridos para encontrar los parámetros libres del modelo.*

Objetivo: *Ajustar de manera robusta un modelo a un conjunto S de datos el cual contiene outliers.*

Algoritmo:

i Aleatoriamente seleccionar una muestra de s elementos de S y encontrar un modelo con esta muestra.

ii Determinar el subconjunto S_i de S compuesto por aquellos elementos que están a una distancia del modelo menor a un umbral t. El subconjunto S_i es el conjunto consenso de la muestra y define los inliers de S.

iii Si el tamaño de S_i es más grande que algún umbral T dado, estimar de nuevo el modelo, utilizando todos los elementos en S_i y terminar.

iv Si el tamaño de S_i es menor que T, seleccionar una nueva muestra y repetir lo anterior.

v Después de N intentos, el conjunto consenso S_i más grande es seleccionado y el modelo es de nuevo estimado usando todos los elementos en S_i.

Previendo que los datos y el número de intentos son suficientemente grandes, se espera que la posición del modelo identificado por RANSAC sea cercana a la solución global sin usar alguna información local acerca de la función objetivo sino muestreando la función objetivo en localidades discretas por medio de selección aleatoria de subconjuntos

[1] El tamaño del subconjunto depende del problema a resolver y debe ser un número suficiente para poder computar un modelo.

mínimos de los datos y generando soluciones candidatas las cuales son evaluadas. A continuación los parámetros usados son brevemente explicados.

5.1.1. Distancia umbral al modelo.

Necesitamos escoger una distancia t tal que con una probabilidad α, el elemento del conjunto sea un inlier. Esta distancia en la práctica se escoge empíricamente. Sin embargo, si asumimos que el error en las mediciones de los elementos sigue una distribución Gaussiana con desviación estándar σ, entonces un valor para t puede obtenerse. Debido a que se ha asumido que los elementos del conjunto deben seguir una distribución Gaussiana, el cuadrado de la distancia del elemento, $d_\perp^2$, es una suma cuadrados de variables Gaussianas y siguen una distribución χ_m^2 con m grados de libertad, donde m es igual a la codimensión del modelo. La probabilidad de que un valor de una variable aleatoria χ_m^2 sea menor que una k, está dada por la función de densidad acumulada $F_m(k^2) = \int_0^{k^2} \chi_m^2(\xi)\mathrm{d}\xi$. De la distribución acumulada se sigue entonces que

$$\begin{cases} \text{inlier} & d_\perp^2 < t^2 \\ \\ \text{outlier} & d_\perp^2 \geq t^2 \end{cases} \quad \text{con } t^2 = F_m^{-1}(\alpha)\sigma^2.$$

Usualmente α se fija a 0.95, así que hay una probabilidad 95 % de que el elemento sea un inlier. Hartley and Zisserman [2004] proporciona una explicación más detallada.

5.1.2. Número de muestras.

Es innecesario e irrealizable tratar con todos los subconjuntos posibles, en lugar de eso, el número de muestras N que se deben probar, se obtiene de manera que podamos asegurar con una probabilidad p de que al menos una de las muestras aleatorias esté libre de outliers. Suponiendo que w es la probabilidad de que un elemento sea un inlier y $\epsilon = 1 - w$ de que sea un outlier, las N selecciones de s elementos requeridas debe cumplir con que $(1 - w^s)^N = 1 - p$, por lo tanto

$$N = \log(1 - p)/\log(1 - (1 - \epsilon)^s). \tag{5.1}$$

5.1.3. Tamaño suficiente del conjunto consenso.

Una regla por omisión es finalizar si el tamaño del conjunto consenso es similar al número de inliers que se cree están en el conjunto completo de elementos. Para n

elementos, $T = (1 - \epsilon)n$. Por ejemplo si se estima que $\epsilon = 0.2$ de un conjunto de 12 elementos, entonces $T = (1 - 0.2)12 = 10$.

5.1.4. Determinación del número de muestras adaptativamente.

Usualmente se desconoce la fracción de outliers presentes en nuestro conjunto de datos, por lo tanto, no podemos determinar N. Para resolver este problema debemos empezar suponiendo el peor caso posible de ϵ y con ese valor calcular una N inicial, luego ir actualizando el valor de N de acuerdo a la proporción de outliers que se vaya encontrando. Este procedimiento de comparación de ϵ, se realiza en cada iteración y de esta manera el cálculo de N se alcanza en forma dinámica. Si el nuevo valor de N es menor al número de iteraciones ya realizadas, el algoritmo se detiene. En el algoritmo 5.2, se describe el cálculo del número máximo de muestras N de manera dinámica y en el algoritmo 5.3 se describe cómo interactúan los algoritmos 5.1 y 5.2.

Algoritmo 5.2: *Algoritmo que modifica el número máximo de iteraciones.*

Objetivo: *Calcular de manera adaptativa el número máximo de iteraciones.*

Algoritmo:
$N \leftarrow \infty$
cont$\leftarrow 0$
while $cont < N$ **do**
  Elegir una muestra y contar el número de inliers del modelo estimado de la muestra.
  $\epsilon \leftarrow 1 - \#inliers/\#totaldepuntos$
  $N \leftarrow$ ecuación (5.1) con $p = 0.99$
  $cont++$
end

5.2. Paralelización del RANSAC.

Para lograr la paralelización el algoritmo RANSAC, es necesario identificar cuáles son las tareas en las que el procesador ocupa más tiempo. En el algoritmo 5.3, esto ocurre durante la ejecución del ciclo **while**. Para confirmar esto, 5 ejecuciones fueron llevadas a cabo, con un conjunto de 500 datos. Los tiempos que tomaron las ejecuciones fueron de 1833, 3995, 3670, 4289 y 3492 milisegundos respectivamente. Por otra parte, se tuvo cuidado en medir también el tiempo que consumió el ciclo **while** durante las ejecuciones, arrojando tiempos de 1803, 3963, 3638, 4264 y 3466 milisegundos respectivamente, lo

Algoritmo 5.3: *Combinación de los algoritmos 5.1 y 5.2.*

Objetivo: *Ajustar de manera robusta un modelo a un conjunto S de elementos, el cual contiene outliers, actualizando de manera dinámica el número máximo de iteraciones.*

Algoritmo:

$N \leftarrow \infty$
$cont \leftarrow 0$
$m\_inliers \leftarrow 0$
while $cont < N$ **do**
  Aleatoriamente seleccionar una muestra de s elementos de S y encontrar un modelo con esa muestra.
  Determinar el conjunto de S_i los cuales están dentro de una distancia umbral t del modelo.
  if $\#S_i > m\_inliers$ **then**
    $mejorS_i \leftarrow S_i$
    $m\_inliers = \#S_i$
    $\epsilon \leftarrow 1 - m\_inliers/\#S$
    $N \leftarrow$ ecuación (5.1)
    if $m\_inliers >= T$ **then**
      Break
    end
  end
  $cont++$
end
if $m\_inliers \geq s$ **then**
  Estimar el modelo final con $mejorS_i$
  Determinar el conjunto de Inliers los cuales están dentro de una distancia umbral t del modelo final.
else if $m\_inliers=0$ **then**
  Error: *El algoritmo RANSAC no convergió a un conjunto de inliers*

que significa que ocupa en promedio cerca de un 99 % del tiempo. Esto sugiere distribuir las tareas llevadas en dicho ciclo en una implementación en paralelo.

5.2.1. Análisis de Paralelización.

La ejecución del algoritmo 5.3 selecciona una muestra en cada iteración. Por lo tanto n nodos seleccionarán n muestras en cada iteración del ciclo **while** y por consiguiente el contador de iteraciones se incrementará en n, *i.e.*, cada nodo necesitará N/n iteraciones del ciclo **while**. Un punto en consideración, es la coordinación del número máximo de iteraciones de tal forma que la suma de todos los contadores de iteraciones en cada procesador sea menor a N, esto en cada iteración, de otra manera algún procesador estaría calculando modelos y existir ya algún procesador que haya encontrado algún conjunto de inliers que actualiza N a un valor mas pequeño. Otro punto importante,

Algoritmo 5.4: *Algoritmo paralelo RANSAC en una máquina con memoria distribuida.*

Objetivo: *Ajustar de manera robusta un modelo a un conjunto S de datos el cual contiene outliers, actualizando de manera dinámica el número máximo de iteraciones con la utilización de programación paralela en memoria distribuida.*

Algoritmo:

Cada nodo tiene una copia del conjunto completo de datos S ó tiene acceso a ellos.
$N \leftarrow \infty$
$cont\_local \leftarrow 0$
$m\_inliers \leftarrow 0$
$bandConv \leftarrow$ true
while $cont\_local * p < N$ **do**
  Cada nodo selecciona de manera aleatoria una muestra de s puntos de S y estima un modelo con esa muestra.
  Cada nodo determina el conjunto de inliers S_i los cuales están dentro de una distancia umbral t del modelo que ha encontrado.
  Reducir sobre todos los nodos el máximo de los $\#S_i$ en una variable $g\_inliers$.
  if $g\_inliers > m\_inliers$ **then**
    $mejorS_i \leftarrow S_i$
    $m\_inliers \leftarrow g\_inliers$
    $r\_inliers \leftarrow \#S_i$
    $\epsilon \leftarrow 1 - m\_inliers/\#S$
    $N \leftarrow$ ecuación (5.1)
  end
  if $m\_inliers \geq T$ **then**
    Break
  end
  $cont\_local++$
end
Se reduce en el nodo *Maestro* la suma de la variable $cont\_local$.
El nodo *Maestro* recolecta todos los $r\_inliers$ y elige cuál es máximo y difunde el *nodo* que lo tuvo.
if $id\_nodo = nodo$ **then**
  if $\#mejorS\_i \geq s$ **then**
    Estimar el modelo final con $mejorS_i$.
    Se envía el modelo al nodo *Maestro*.
  else if $m\_inliers = 0$ **then**
    Error: *El algoritmo RANSAC no convergió a un conjunto de inliers.*
    $bandConv \leftarrow$ false;
  end
end
nodo difunde $bandConv$.
if $id\_nodo = Master$ & $bandConv$ **then**
  Se recibe en el nodo *Maestro* el modelo.
  Se establecen los *inliers* y *outliers* del modelo estimado.
end

es que el conjunto completo de los datos debe estar a disposición de los n nodos, de otra manera las muestras aleatorias que los nodos seleccionarían, estarían limitadas al subconjunto que le fue asignado y no al conjunto completo de datos, introduciendo un sesgo en los modelos que cada nodo estima. En el algoritmo 5.4, se muestra el pseudo código implementado en la paralelización del RANSAC.

5.3. Estimación de la matriz fundamental.

El algoritmo de ajuste robusto RANSAC puede ser utilizado en cualquier problema de estimación en el cual se tenga bien definido cómo calcular un modelo a un conjunto de elementos y como determinar distancia de cada elemento del conjunto al modelo calculado. En visión computacional se usa generalmente para estimar líneas, homografías, matrices de proyección, tensores trifocales y matrices fundamentales. Lo que nos ocupa en esta ocasión, es la estimación de la matriz fundamental. El cálculo de esta matriz dado un conjunto de datos es con el algoritmo de los 8 puntos normalizados [Longuet-Higgins, 1981] —ver algoritmo 2.1—. Con respecto a la clasificación, la distancia de Sampson ó distancia epipolar simétrica —ver apéndice B.2—, pueden emplearse con una probabilidad de $\alpha = 0.95\,\%$ de que el elemento sea un inlier para $t = 1.96\sigma$ píxeles. En la práctica σ es difícil de calcular y se le puede asignar el valor de 0.5 de píxel ó 1 píxel.

5.4. Experimentos.

5.4.1. Experimentos con datos sintéticos.

A fin de probar RANSAC bajo condiciones controladas se realizó una serie de ejecuciones con datos sintéticos sobre el algoritmo secuencial —algoritmo 5.3— para encontrar la estimación de la matriz fundamental cuyos resultados se explican a continuación. Un conjunto 500 correspondencias fueron generadas con la proyección de puntos tridimensionales sintéticos usando un par de matrices de proyección conocidas $\mathbf{P}$ y $\mathbf{P}'$ y a cuyas proyecciones se les añadió ruido Gaussiano con $\mu = 0$ y $\sigma = 1$. Para introducir ruido que no satisface la distribución Gaussiana, se trasladó aleatoriamente un subconjunto de 166 puntos proyectados con $\mathbf{P}'$, *i.e.*, se introdujeron outliers. Tales datos sintéticos son mostrados en la figura 5.1. Se ejecutó el algoritmo secuencial 20 veces para observar el

comportamiento del algoritmo secuencial RANSAC. Se sabe exactamente que los datos contienen 334 inliers. De la figura 5.2(a) se observa que en las 20 ocasiones, el número de

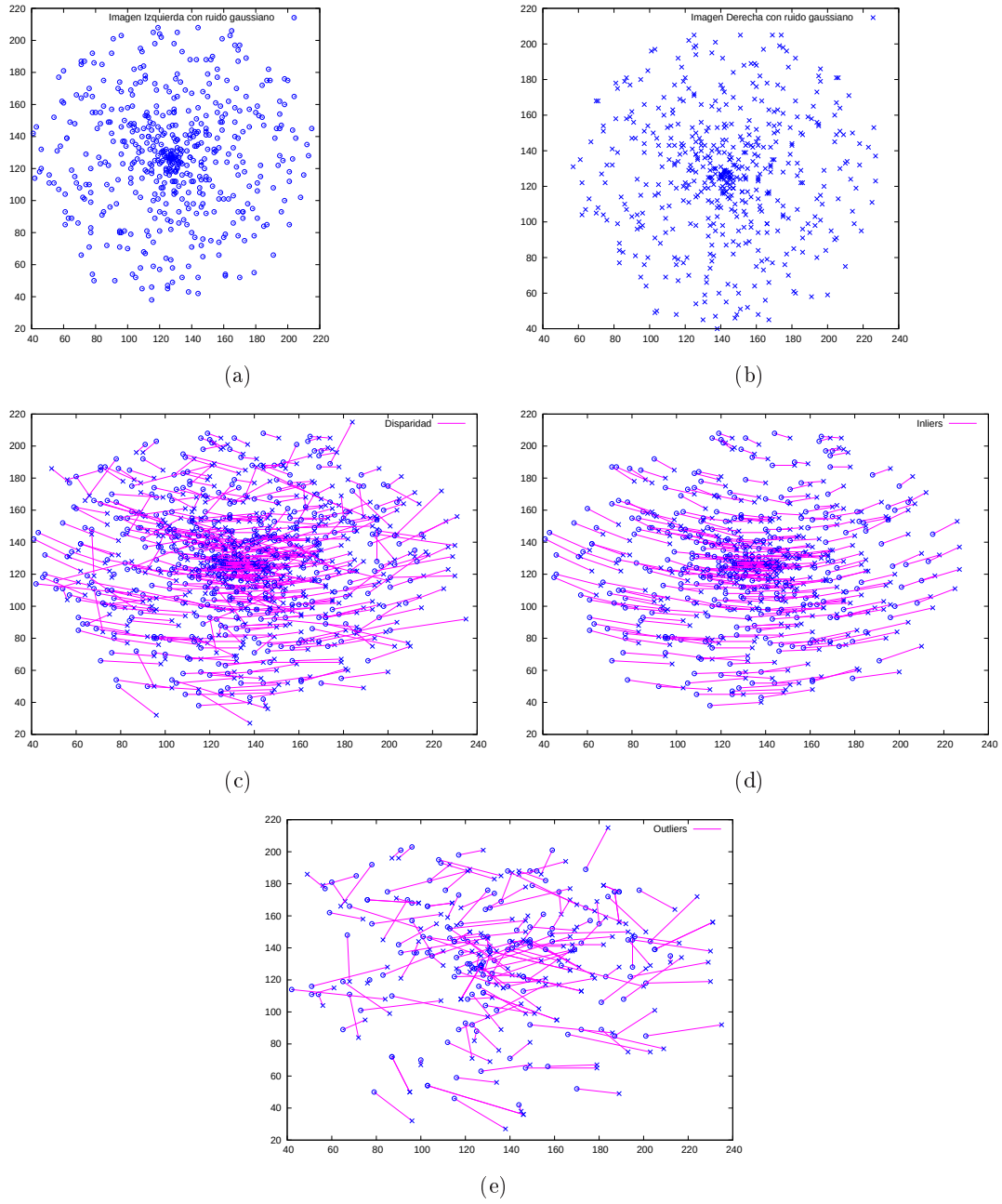


Figura 5.1: *(a) 500 puntos proyectados con* $\mathbf{P}$*, (b) proyección de los mismos 500 puntos con* $\mathbf{P}'$*, (c) disparidad de los puntos correspondientes con ruido Gaussiano y traslación aleatoria de 166 para producir outliers, (d) 334 inliers, (f) 166 outliers.*

inliers fue mayor a 300 y aunque no se alcanzó el número exacto de inliers, estos valores indican un grado confiable de eficacia proporcionando un conjunto aceptable de inliers estimados. $N = \infty$ es actualizado en la primera iteración de ciclo **while** de acuerdo a la ecuación (5.1) a un valor no mayor a

$$N = \log(1-p)/\log(1-(1-(1-s/\#S)^s)),$$

el cual es número finito, esto sucede ya que desde la primera iteración, cuando menos las s muestras serán consideradas como inliers del modelo que se estimó de ellos y esto evita que el algoritmo permanezca iterando infinitamente. Conforme al experimento descrito, $\#S = 500$, $s = 8$ desde la primera iteración, N debe valer cuando mucho a $N = 1.0636\text{e}15$ para una $p = 0.99$. No se puede establecer de manera clara la eficiencia para RANSAC debido a la naturaleza aleatoria de la selección de las muestras para estimar los modelos, *i.e.*, es un algoritmo heurístico, sin embargo, como se aclaró anteriormente, N tiene un número finito desde la primera iteración y por lo tanto no se queda ejecutando indefinidamente. Al final de las iteraciones necesarias se puede evaluar el desempeño del algoritmo, *e.g.* para el experimento descrito, la figura 5.2(b) se observa el número de iteraciones necesitadas para encontrar inliers y en la figura 5.2(c), el tiempo en milisegundos. En la figura 5.2(c) se observa que las 20 ejecuciones se incurrieron en menos de 2 segundos. La figura 5.2(d) muestra los inliers encontrados en una ejecución y la figura 5.2(e) a los outliers.

En la siguiente sección se describen experimentos con datos reales.

5.5. Experimento con datos reales.

Siguiendo la línea de experimentos presentado en los capítulos anteriores, pares de listas de esquinas obtenidas con KLT y después relacionadas en correspondencias fueron depuradas con RANSAC. Con el objetivo de probar al algoritmo paralelo propuesto —algoritmo 5.4—, conjuntos de correspondencias de 500, 2500 y 5000 elementos son usados como entradas. En los siguientes párrafos se describen los resultados con respecto a tiempo, rapidez, eficiencia y costo como se ha estado haciendo al respecto de los algoritmos paralelos.

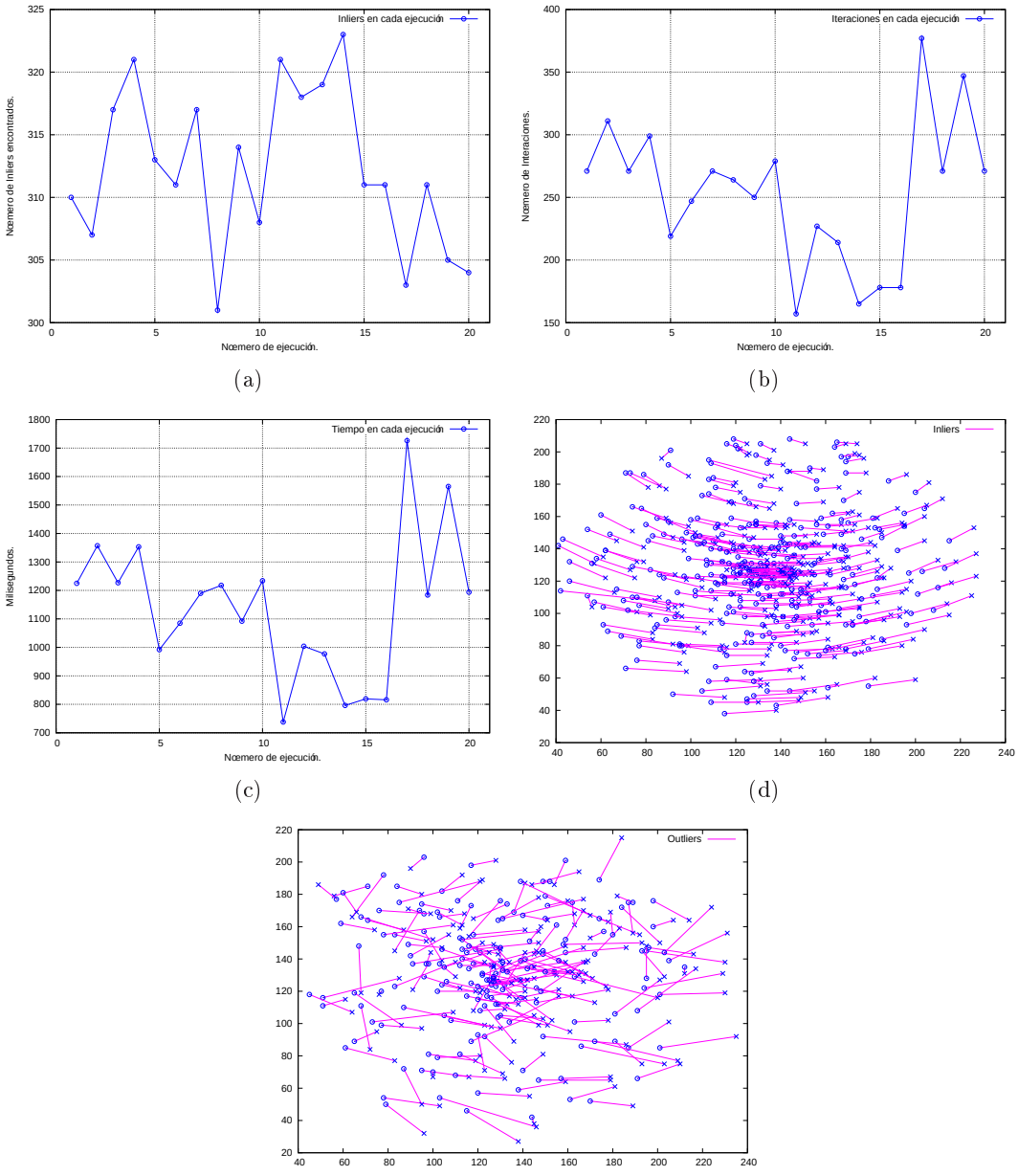


Figura 5.2: *20 ejecuciones del algoritmo secuencial RANSAC. (a) En las 20 ocasiones, el algoritmo encontró un número de inliers por arriba de 300, i.e. tuvo una eficacia de por arriba de 300/344=0.87 para encontrar inliers. (b) Iteraciones necesitadas en cada ejecución. (c) Tiempo en milisegundos necesitado para cada una de los 20 ejecuciones de RANSAC secuencial. (d) y (e) Ejemplo de inliers y outliers en encontrados en una de las 20 ejecuciones.*

Tiempo de ejecución. En el cuadro 5.1 se muestra a detalle los resultados obtenidos de la ejecución del algoritmo y en la figura 5.3 se da una descripción gráfica. En la figura

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	1123	599	367	293	238	213	199	164	148	136	110	111
	2500	4315	2174	1331	1003	890	720	677	555	527	527	440	395
	5000	10559	5289	3547	2523	2241	1937	1667	1452	1435	1337	1231	1215
Con	500	1123	612	382	494	440	466	512	463	443	474	409	434
	2500	4315	2203	1361	1217	1648	1294	1452	1334	1380	1565	1353	1281
	5000	10559	5353	3607	2915	4183	3431	3225	3010	3370	3311	3185	3231

Cuadro 5.1: *Tiempos en milisegundos conseguidos en la ejecución del algoritmo paralelo 5.4 sin y con comunicación entre los procesadores.*

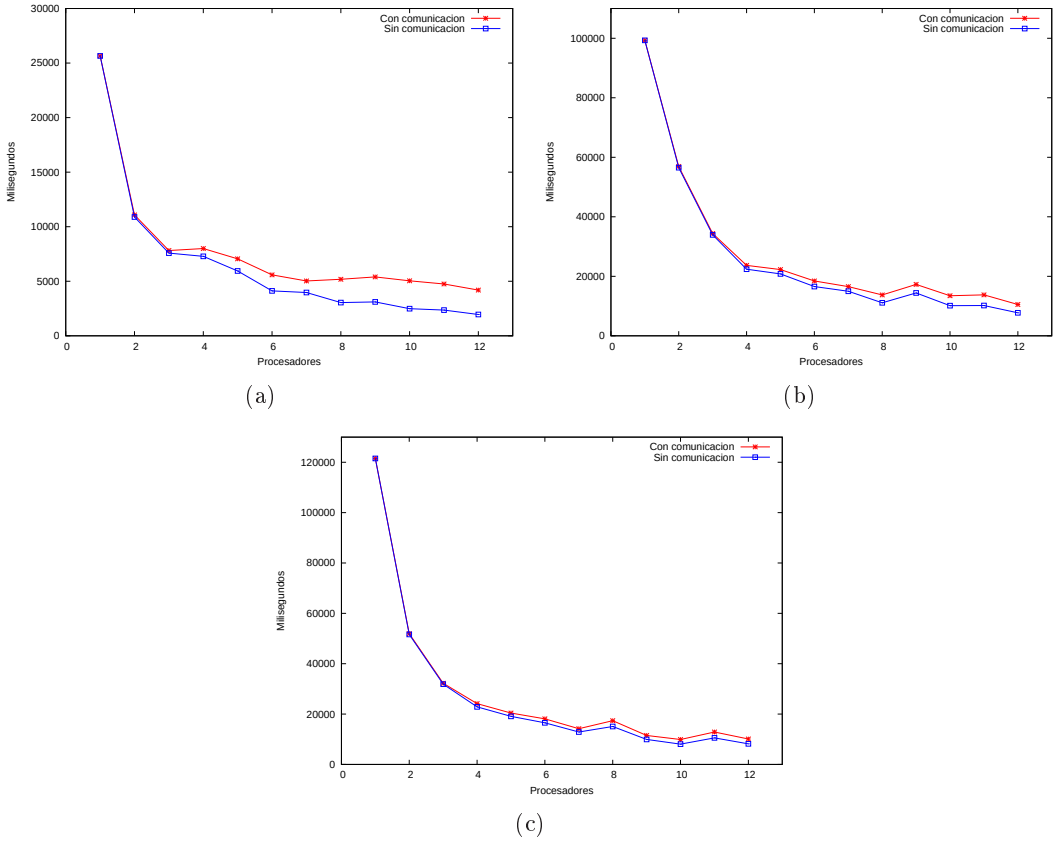


Figura 5.3: *Gráficas de tiempos promedios alcanzados con el algoritmo 5.4 sin y con comunicación entre los procesadores.*

se puede observar un comportamiento similar a los obtenidos en las secciones de experimentos de los capítulos anteriores, *i.e.*, con cada procesador se observa una disminución importante de tiempo, esto sugiere que RANSAC paralelo tiene el comportamiento deseado en un algoritmo paralelo. Sin embargo, las perturbaciones que se observan en las

gráficas son explicadas por la naturaleza heurística del algoritmo, como se pudo ver en los experimentos de los datos sintéticos.

Rapidez del algoritmo. Los resultados de la rapidez del algoritmo se describen en el cuadro 5.2, y en la figura 5.4. Se observa de la gráfica, que el algoritmo tiene un

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	1	1.87	3.05	3.83	4.71	5.27	5.64	6.84	7.58	8.25	10.20	10.11
	2500	1	1.98	3.24	4.30	4.84	5.99	6.37	7.77	8.18	8.18	9.80	10.92
	5000	1	1.99	2.97	4.18	4.71	5.45	6.33	7.27	7.35	7.89	8.57	8.68
Con	500	1	1.83	2.93	2.27	2.55	2.40	2.19	2.42	2.53	2.36	2.74	2.58
	2500	1	1.95	3.17	3.54	2.61	3.33	2.97	3.23	3.12	2.75	3.18	3.36
	5000	1	1.97	2.92	3.62	2.52	3.07	3.27	3.50	3.13	3.18	3.31	3.26

Cuadro 5.2: *Rapidez del la ejecución paralela de RANSAC.*

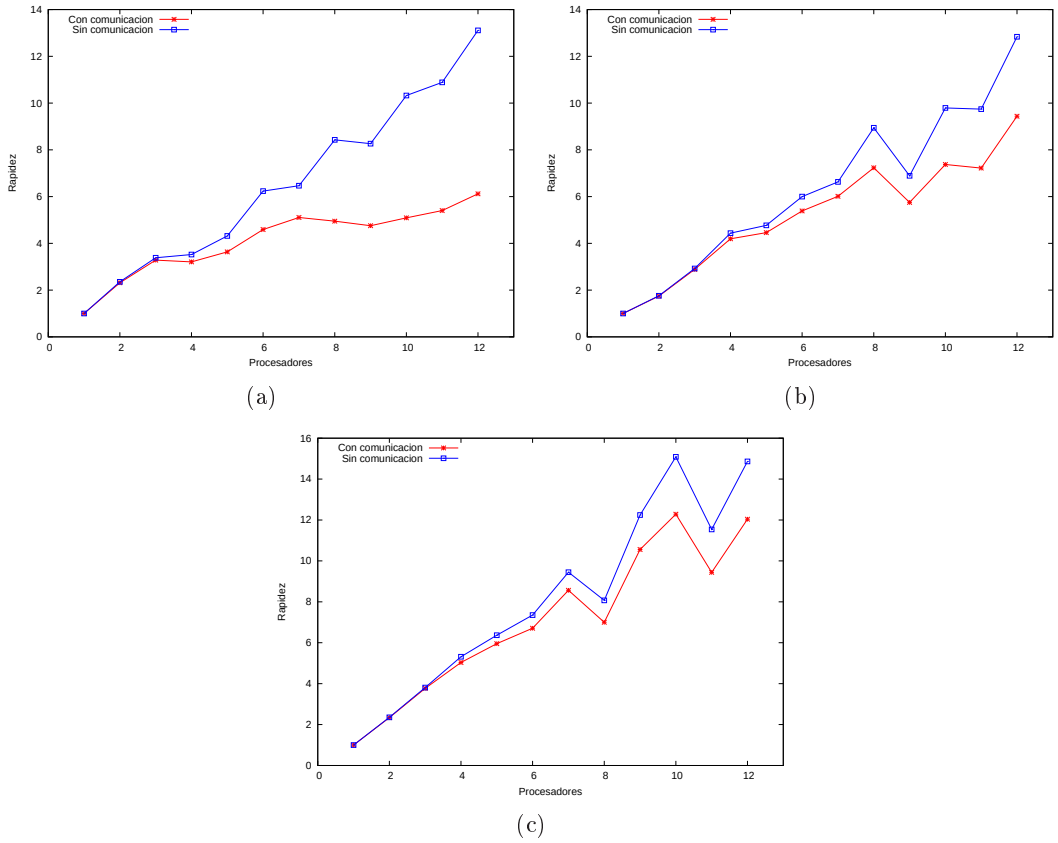


Figura 5.4: *Gráficas de rapidez del algoritmo paralelo 5.4 con y sin overhead entre los procesadores.*

comportamiento bastante cercano al ideal, pero, ¿cómo puede ser la rapidez más grande que el ideal? Aunque algunos autores argumentan que esto no puede ser posible, *e.g.*

Faber et al. [1986], en estas pruebas se presta atención a lo contrario. Quinn [1994] menciona un ejemplo de cómo esto puede ocurrir:

Cuando se resuelve un problema de búsqueda, un algoritmo secuencial puede perder mucho tiempo examinando una estrategia de callejón sin salida –dead-end strategy–. Un algoritmo paralelo persigue muchas posibles estrategias simultáneamente y uno de los procesos puede encontrar la solución muy rápido.

Esta situación se hace presente con RANSAC; mientras el algoritmo secuencial analiza cada una de las soluciones de manera estocástica, el algoritmo paralelo analiza múltiples soluciones al mismo tiempo de forma estocástica y suele suceder que un proceso encuentra una solución muy rápido. Esta explica por qué en algunas ocasiones se tiene una rapidez mayor al ideal.

Eficiencia del algoritmo. La información de eficiencia del algoritmo, está presentada en el cuadro 5.3 y su descripción gráfica en la figura 5.5. Cuando la eficiencia está por encima de 1, se justifica con el hecho de que es un algoritmo heurístico, justo como ya se explico en el párrafo anterior. Dejando de lado la naturaleza aleatoria del algoritmo,

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	1	1.87	3.05	3.83	4.71	5.27	5.64	6.84	7.58	8.25	10.20	10.11
	2500	1	1.98	3.24	4.30	4.84	5.99	6.37	7.77	8.18	8.18	9.80	10.92
	5000	1	1.99	2.97	4.18	4.71	5.45	6.33	7.27	7.35	7.89	8.57	8.68
Con	500	1	1.83	2.93	2.27	2.55	2.40	2.19	2.42	2.53	2.36	2.74	2.58
	2500	1	1.95	3.17	3.54	2.61	3.33	2.97	3.23	3.12	2.75	3.18	3.36
	5000	1	1.97	2.92	3.62	2.52	3.07	3.27	3.50	3.13	3.18	3.31	3.26

Cuadro 5.3: *Eficiencia obtenida en la ejecución paralela de RANSAC.*

las gráficas indican una utilización de los recursos de cómputo en la mayoría de las veces por encima del 80 % en la ejecución del algoritmo paralelo. De igual manera que en los experimentos de KLT en el capítulo 3 en un conjunto de datos pequeños el algoritmo se desempeña pobremente cuando se usan muchos procesadores.

Costo. En cuanto al esfuerzo necesitado por la máquina paralela, ver el cuadro 5.4 y la gráfica 5.6. Los datos muestran en general que el costo en el que el algoritmo paralelo incurre no alcanza el doble del tiempo y solamente en el conjunto de 500 correspondencias se acercó mucho al doble del tiempo.

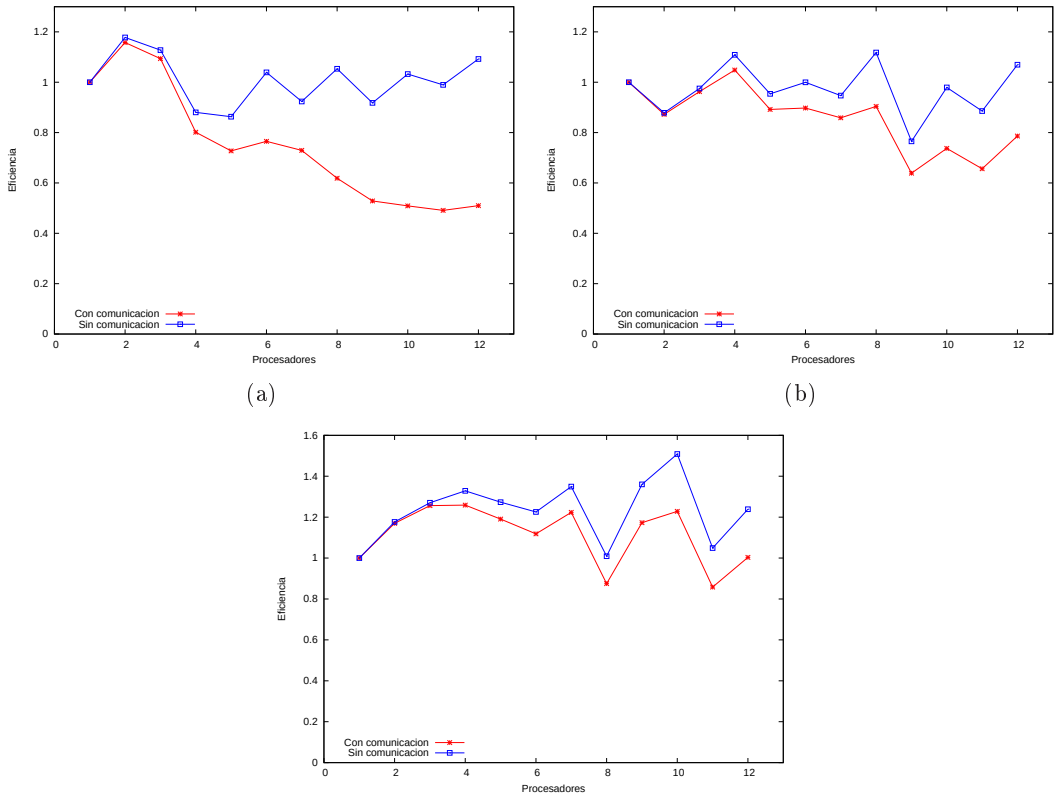


Figura 5.5: *Gráficas de eficiencia conseguidos con el algoritmo 5.4 con y sin comunicación entre los procesadores.*

Overhead	# de píxeles	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
Sin	500	1123	1198	1101	1172	1190	1278	1393	1312	1332	1360	1210	1332
	2500	4315	4348	3993	4012	4450	4320	4739	4440	4743	5270	4840	4740
	5000	10558	10578	10641	10092	11205	11622	11669	11616	12915	13370	13541	14580
Con	500	1123	1224	1146	1976	2200	2796	3584	3704	3987	4740	4499	5208
	2500	4315	4406	4083	4868	8240	7758	10164	10672	12420	15650	14883	15372
	5000	10558	10706	10821	11660	20915	20586	22575	24080	30330	33110	35035	38772

Cuadro 5.4: *Costo implicado en la ejecución paralela de RANSAC.*

5.6. Discusión de resultados.

Se obtuvo una buena implementación paralela del algoritmo de estimación robusta RANSAC, proporcionando reducciones considerables en el tiempo de ejecución. Como ya se mencionó anteriormente, RANSAC es un algoritmo que se usa en varios tipos de estimación, por lo que pruebas para diferentes tipos de estimación quedan pendientes y pueden ser objeto de un estudio más profundo.

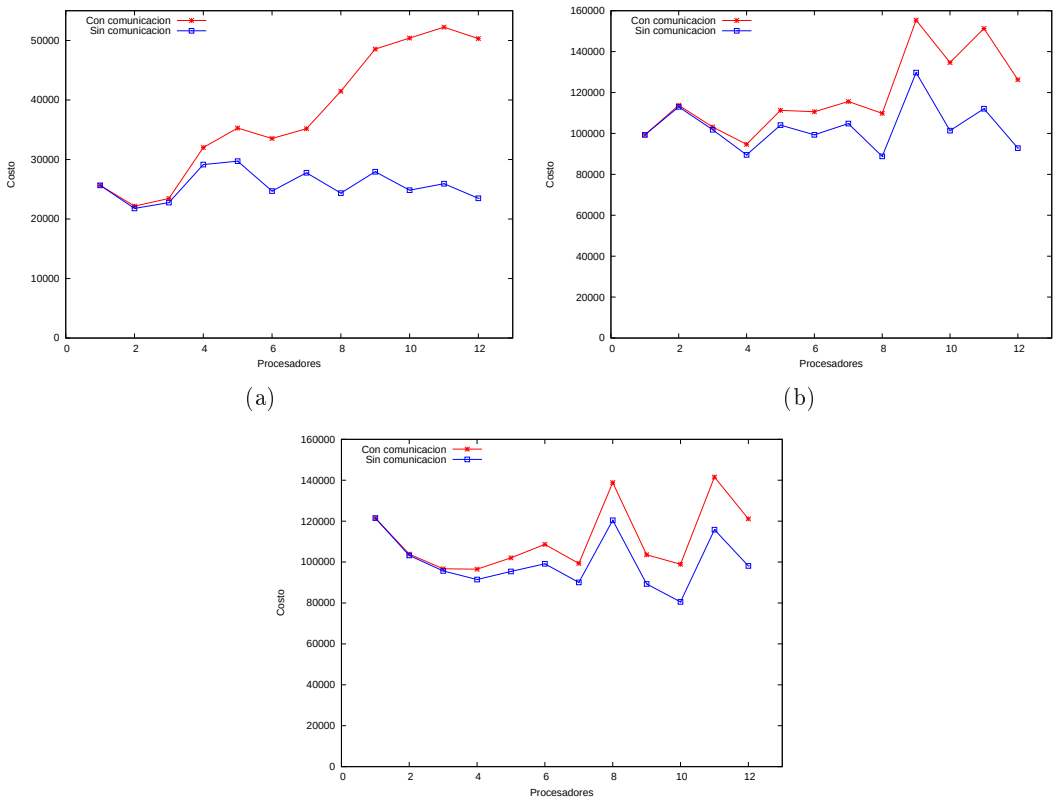
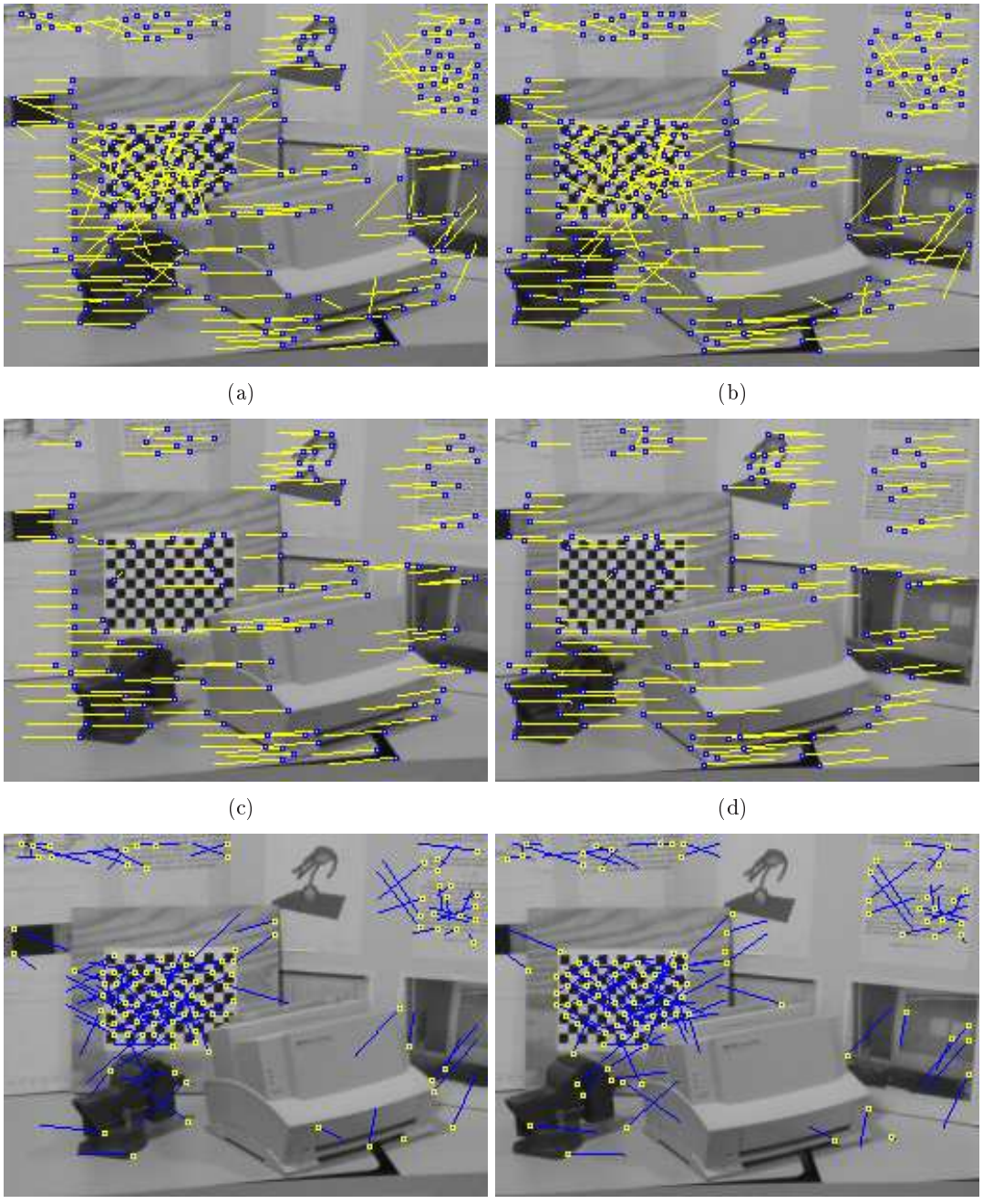


Figura 5.6: *Gráficas de costo involucrado con el algoritmo 5.4 con y sin comunicación entre los procesadores.*

En este capítulo se ve claro a lo que se refiere Kim K.-N. [1998]: *En niveles altos de visión, las estructuras computacionales regulares que se encuentran en niveles bajos de visión desaparecen y dependencias irregulares de datos aparecen. Por lo tanto es realmente difícil alcanzar un buen rendimiento sobre el proceso de visión entero.*

En la ultima gráfica —5.7— de este capítulo se muestran resultados de aplicar RANSAC en un par de imágenes con el objetivo de depurar el conjunto de correspondencias devueltas por al algoritmo secuencial o paralelo devuelto en el capítulo anterior.



(a) (b)

(c) (d)

(e) (f)

Figura 5.7: *En (a) y (b) están dibujadas en las imágenes 250 correspondencias encontradas con el algoritmo de asignación de correspondencias 4.2 con ZNCC con un umbral* $t = 0$. *Después de usar RANSAC con* $w = 0.01$, *una distancia umbral de 1 píxel en la depuración de la lista, los inliers son desplegados en (c) y (d) y los outliers en (e) y (f). Con los inliers encontrados, ahora se puede estimar a la matriz fundamental de forma mas adecuada entre estas dos imágenes con el algoritmo de los 8 puntos normalizados [Longuet-Higgins, 1981], algoritmo 2.1*

Capítulo 6

Algoritmo de calibración paralelo.

El proceso de calibración descrito en esta tesis el cual es común en muchos enfoques de reconstrucción tridimensional que se puede encontrar en los trabajos en la literatura involucra encontrar la geometría epipolar entre un número $n \geq 2$ de imágenes. Habiendo paralelizado cada etapa de calibración como se ha descrito hasta el momento, un algoritmo completo de calibración paralelizado se adopta simplemente mediante la secuencia ordenada de las etapas paralelizadas, justo como se establece el algoritmo secuencial pero con cada etapa ya paralelizada.

La computadora paralela para la que se ha hecho la implementación y en donde se han realizado las pruebas de los algoritmos, contiene 12 procesadores y para aprovecharlos, el algoritmo de calibración final debe usarlos todos. En este capítulo se presenta un algoritmo de calibración completo, *i.e.*, con todas las etapas de calibración paralelizadas. Además análisis de tiempos, rapidez, eficiencia y costo es llevado a cabo en la sección 6.1 con $p = 1, ..., 12$ procesadores. Por otra parte, en la sección 6.2 un algoritmo de calibración diseñado específicamente para usar los 12 procesadores se describe.

6.1. Secuencia ordenada de las etapas de calibración.

Tal como se hizo en los capítulos anteriores un análisis del algoritmo completo de calibración en cuanto a tiempo, rapidez, eficiencia y costo se lleva a cabo en esta sección. En el algoritmo 6.1 se muestra como se ordenan las etapas de calibración. Existe una analogía para algoritmo el secuencial como para el paralelo en la manera de ordenar las etapas del algoritmo presentado en esta tesis. El algoritmo maneja una variable $p \geq 1$,

en la cual se indica el número de procesadores a usar; cuando $p = 1$ se considera que se están utilizando los algoritmos secuenciales de los capítulos anteriores, y cuando $p \geq 2$ a los algoritmos paralelos. Se probó el algoritmo presentado con 3 conjuntos de imágenes.

Algoritmo 6.1: *Algoritmo de calibración, integrando cada etapa descritas en los capítulos anteriores .*

Objetivo: *Calibrar de un conjunto de m imágenes por medio de la estimación de la geometría epipolar entre cada par contiguo de imágenes usando p procesadores.*

Algoritmo:
$ImIzq \leftarrow I_0$
$lCorners \leftarrow$ Esquinas encontradas con KLT con p procesadores en $ImIzq$.
for *i=1, i<m, i++* **do**
  $ImDer \leftarrow I_i$
  $rCorners \leftarrow$ Esquinas encontradas con KLT con p procesadores en $ImDer$.
  $matchs_{i-1,i} \leftarrow$ Asignación de correspondencia entre $lCorners$ y $rCorners$ con p procesadores.
  $F_{i-1,i} \leftarrow$ estimación de la matriz fundamental con RANSAC con p procesadores, dado el conjunto de correspondencias $matchs_{i-1,i}$.
  $Imleft \leftarrow Imright$
  $lCorners \leftarrow rCorners$
end

Los resultados se muestran en la figura 6.1; las figuras 6.1(a), 6.1(b) y 6.1(c) muestran los tiempos que se necesito en la ejecución del algoritmo; de igual manera se puede ver que la rapidez del algoritmo es aceptable cuando se usan desde 2 hasta 12 procesadores, *i.e.*, mientras más procesadores se usen, más rápido es el algoritmo; en cuanto a la eficiencia de los procesadores se observa que mientras más procesadores se usan, empieza a ser ineficiente el algoritmo; en lo que al costo se refiere, se observa que dentro de los 12 procesadores, el algoritmo en general cuesta menos del doble de esfuerzo.

6.2. Algoritmo alternativo de calibración paralela.

Aunque se ha visto que existe una mejora con respecto a tiempo y rapidez, y la eficiencia de los procesadores es aceptable con un costo no muy elevado de esfuerzo de las máquinas, en esta sección se presenta un algoritmo que está diseñado para trabajar con tres grupos de procesadores. En general se pueden usar más de cuatro procesadores por grupo, sin embargo, por la arquitectura de la máquina para la que se implementó, se consideraron tres grupos de cuatro tratando de evitar que la comunicación entre procesadores influyera demasiado en la ejecución del algoritmo, minimizándola solamente

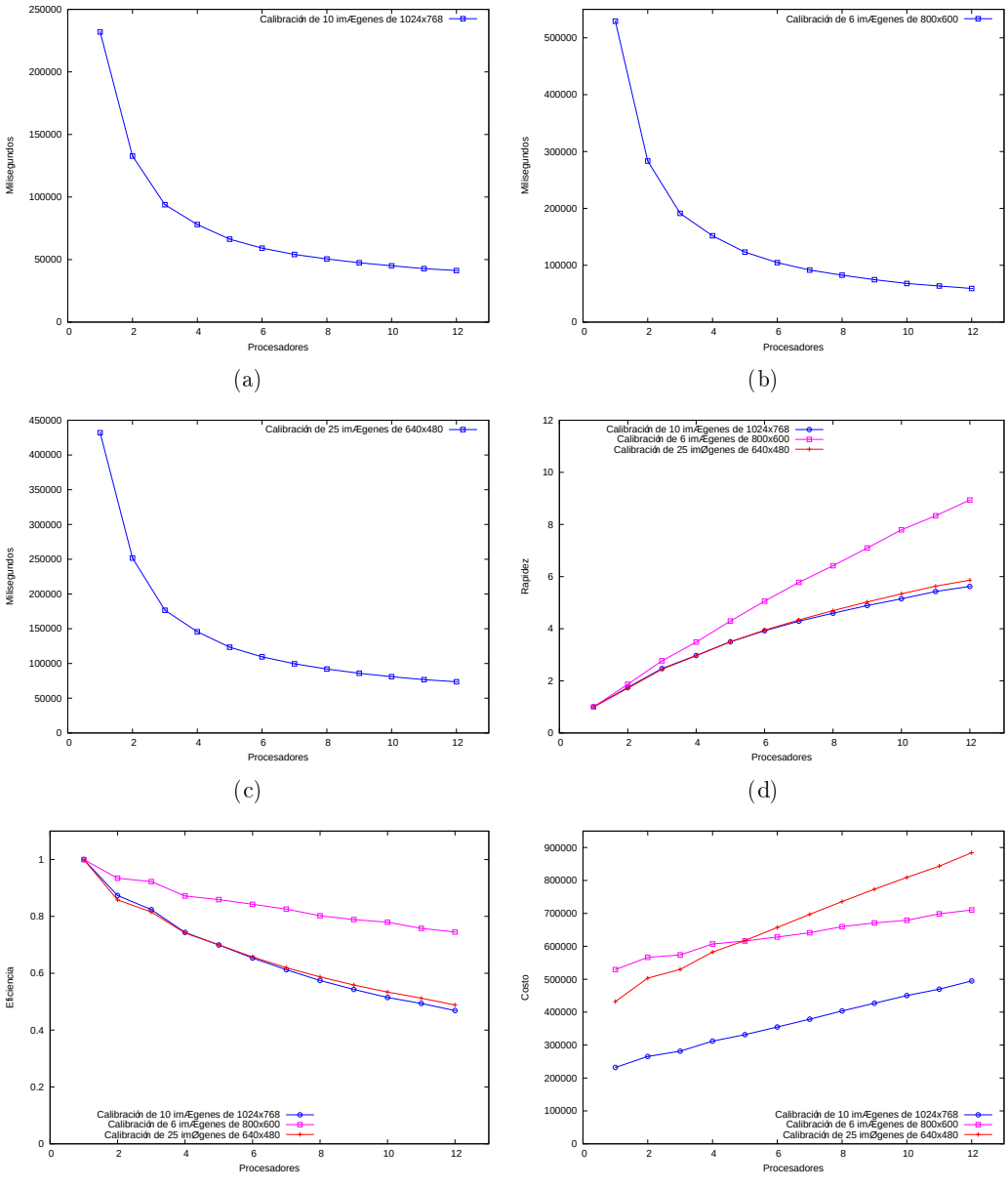


Figura 6.1: *Análisis de 3 conjuntos de imágenes. En la figura (a) los tiempos de necesitados para la calibración de 10 imágenes de 1024×768 píxeles, en (b) de un conjunto de 6 imágenes de 800×600 píxeles y (c) 25 imágenes de 640×480 píxeles. La figura (d) muestra la rapidez que se logra al usar $p = 1, \ldots, 12$ procesadores, (e) la eficiencia de los procesadores y (f) el costo implicado en el uso del algoritmo 6.1.*

a comunicación entre los grupos de procesadores que se encuentran en una sola computadora —descripción de la computadora paralela en la sección 1.5—. En la siguiente subsección se explica a detalle cómo se diseñó este algoritmo.

id_{global}/tam_{grupo}	0/4	1/4	2/4	3/4	4/4	5/4	6/4	7/4	8/4	9/4	10/4	11/4
$grupo$	0	0	0	0	1	1	1	1	2	2	2	2
id_{grupo}	0	1	2	3	0	1	2	3	0	1	2	3

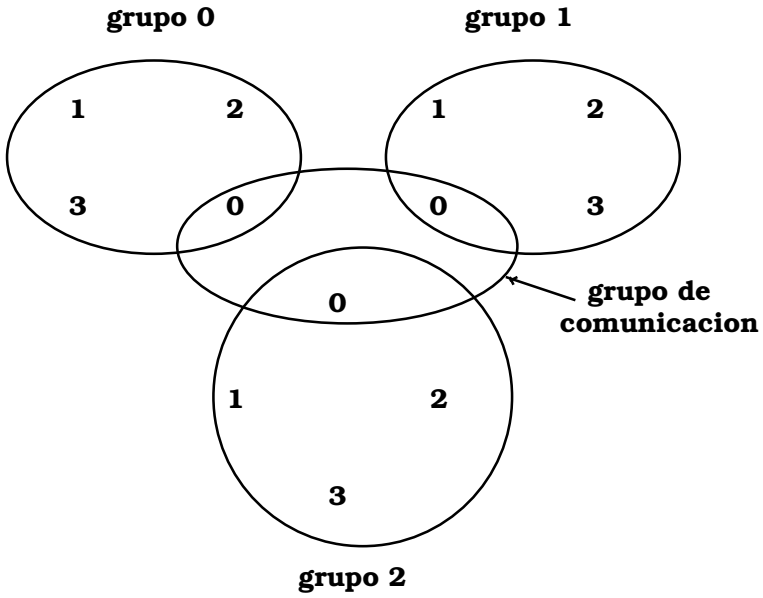


Figura 6.2: *Formación de los grupos de trabajo y de comunicación. El conjunto de los 12 procesadores es fragmentado formando 3 grupos de trabajo. El grupo de comunicación está configurado por los procesadores con identificación 0 de cada grupo.*

6.2.1. Formación de los grupos.

La objetivo del algoritmo que se presenta en esta subsección es minimizar el overhead entre el cúmulo de computadoras. Los algoritmos paralelos diseñados en los capítulos anteriores serán siempre usados, sin embargo, solo se podrá hacer un análisis sobre tiempo, rapidez, eficiencia y costo para $p = 12$ y comparar con el resultado obtenido en la sección anterior para $p = 12$. La formación de los grupos de trabajo así como el de comunicación entre los grupos se lleva a cabo de la siguiente manera. Un procesador con identificación global id_global estará asignado al grupo $grupo = id\_global/tam\_grupo$, en donde para esta implementación $tam\_grupo = 4$. En la figura 6.2 se muestra cómo quedan distribuidos los procesadores dentro de los grupos así como la formación con los

identificadores. El objetivo de dividir en grupos de procesadores es que varias imágenes sean procesadas al mismo tiempo paralelamente. En las gráficas de tiempos de los capítulos anteriores se observa como dentro de los primeros 4 procesadores hay una reducción importante del tiempo y a partir del quinto procesador, esta reducción empieza a decrecer más lentamente, es decir, si se incrementa cada vez mas y mas el número de procesadores, llega un momento en que se desperdicia mas recursos que la ganancia en tiempo, velocidad, etc. Por tal motivo, se puede ver a los grupos como pequeñas máquinas paralelas de cuatro procesadores procesando paralelamente las imágenes.

6.2.2. Flujo de las imágenes.

El conjunto de imágenes es asignado a los grupos de tal forma que se encuentre balanceada la asignación, *i.e.*, el número de imágenes diferirá cuando mucho en 1. Cada grupo encuentra la lista de esquinas, asigna correspondencias y estima la geometría entre las imágenes que le corresponde. Un detalle importante es no olvidarse de que hay que asignar correspondencias entre la primera imagen de un grupo con índice mayor a cero y la última del grupo menor que él. Esto de muestra mas a detalle en la figura 6.3.

6.2.3. Pruebas de calibración usando grupos de procesos.

El algoritmo que integra las ideas de las subsecciones anteriores se despliega en el algoritmo 6.2. Aunque en las pruebas que se hicieron fueron con 12 procesadores formando 3 grupos, en este algoritmo se define el número de grupos en la variable $NGRUPOS$ y cada procesador es asignado a un grupo de acuerdo a la siguiente de acuerdo al valor $global\_id/size\_working\_group$ y si el número de imágenes a calibrar es suficiente para que al menos una sea asignada a un grupo, se espera que el algoritmo sea funcional, sin embargo, es importante aclarar que pruebas con más de 12 procesadores no fueron llevadas a cabo durante el desarrollo de este documento de tesis.

Se probaron 8 conjuntos de imágenes con el fin de comparar los dos algoritmos de calibración. Los resultados se muestran en el cuadro 6.1. Se puede ver que en todos los casos de prueba, el algoritmo 6.2 tuvo un mejor rendimiento que el algoritmo 6.1 con 12 procesadores. La figura 6.4 muestra la proporción de mejora que se obtiene en el uso del algoritmo que trabaja con grupos de procesos, *e.g.*, en el primer conjunto de imágenes

Algoritmo 6.2: *Algoritmo de calibración empleando grupos de procesos.*

Objetivo: *Calibrar de un conjunto de m imágenes por medio de la estimación de la geometría epipolar entre cada par contiguo de imágenes usando grupos de procesos en p procesadores.*

Algoritmo:

$nv \leftarrow$ parte entera de $m/NGRUPOS$
$res\_nv \leftarrow$ el residuo entre $m/NGRUPOS$
$size\_working\_group \leftarrow p/NGRUPOS$
$working\_group \leftarrow$ parte entera de $global\_id/size\_working\_group$
Establecer grupos de trabajo
if $working\_group = Master$ **then**
  $media\_group \leftarrow 0$
else
  $media\_group \leftarrow 1$
end
Establecer el grupo de comunicación
if $working\_group < res\_nv$ **then**
  $nv++$
end
if $working\_group = Master$ **then**
  for $i=0,\ i<nv,\ i++$ **do**
    $ind_i \leftarrow i$
  end
else
  $nv++$
  if $working\_group < res\_nv$ **then**
    for $i=0,\ i<nv,\ i++$ **do**
      $ind_i \leftarrow i + working\_group * (nv-1) - 1$
    end
  else
    for $i=0,\ i<nv,\ i++$ **do**
      $ind_i \leftarrow i + working\_group * (nv-1) + res\_nv - 1$
    end
  end
end
if $working\_group = Master$ **then**
  $L_0 \leftarrow$ esquinas encontradas con KLT paralelo en $Imagen_0$, usando $p/NGRUPOS$ procesadores
end
for $i=1,\ i<nv,\ i++$ **do**
  $L_i \leftarrow$ esquinas encontradas con KLT paralelo en $Imagen_{ind_i}$, con $p/NGRUPOS$ procesadores
end
if $working\_group\_id = Master$ **then**
  if $media\_group\_id < size\_media\_group\text{-}1$ **then**
    Send L_{nv-1} al grupo, $media\_group\_id + 1$
  end
  if $media\_group\_id > 0$ **then**
    Recive L_0 del grupo $media\_group\_id - 1$
  end
end
for $i=0,\ i<nv\text{-}1,\ i++$ **do**
  $M_{i,i+1} \leftarrow$ asignación de correspondencias entre L_i y L_{i+1} usando $p/NGRUPOS$ procesadores
end
for $i=0,\ i<nv\text{-}1,\ i++$ **do**
  $F_{i,i+1} \leftarrow$ estimación de la matriz fundamental con RANSAC con $p/NGRUPOS$ procesadores, dado el conjunto de correspondencias $M_{i,i+1}$.
end

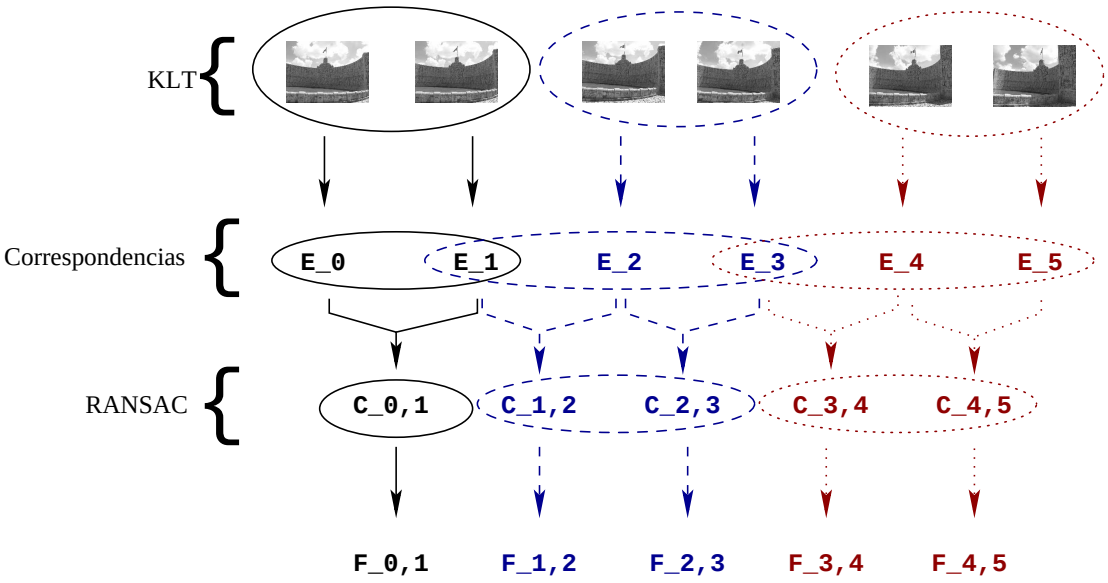


Figura 6.3: *Flujo de las imágenes en algoritmo paralelo. Cada grupo trabaja con cuatro procesadores y la información que utiliza cada uno de los grupos en cada etapa está encerrada en elipses. En este diagrama se muestra el flujo de la información a través del algoritmo. En este ejemplo 6 imágenes son procesadas, a cada grupo le es asignadas dos imágenes y encuentran las esquinas con el algoritmo KLT paralelo,* i.e., *cada grupo produce un par de listas de esquinas, a continuación el algoritmo de correspondencias en paralelo es usado para asignar correspondencias entre las listas encontradas. Como se observa en la figura, los grupos mayores a 0 —elipses punteadas— copian la ultima lista de esquinas del grupo menor a él para asignar correspondencias entre la última imagen del grupo menor con la primera de las imágenes asignadas a él. Finalmente el algoritmo RANSAC paralelo se emplea para estimar las matrices fundamentales entre cada par de imágenes.*

	# de imágenes	10	10	4	8	6	6	10	25
Algoritmo 6.1	milisegundos	53858	41244	50901	39872	24984	59194	69116	73689
	rapidez	6.22	5.62	7.69	4.97	6.17	8.94	4.49	5.86
	eficiencia	0.51	0.46	0.64	0.41	0.51	0.74	0.37	0.48
	costo	646296	494928	610812	478464	299808	710328	829392	884268
Algoritmo 6.2	milisegundos	41197	31645	37894	29921	21276	57226	54596	59748
	rapidez	8.13	7.33	10.34	6.63	7.25	9.24	5.68	7.23
	eficiencia	0.67	0.61	0.86	0.55	0.60	0.77	0.47	0.60
	costo	494364	379740	454728	359052	255312	686712	655152	716976

Cuadro 6.1: *Este cuadro muestra las mejoras en cuanto a los tiempos que se puede lograr con el uso del algoritmo de calibración 6.2. Se puede observar que en todos los casos, el algoritmo 6.2 es más rápido y eficiente y menos costoso que cuando se usa el algoritmo 6.1 con 12 procesadores.*

se necesitó solamente menos del 80 % de lo que se necesita con el primer algoritmo. El 3^{er} conjunto resulto ser el mas beneficiado mientras que el 6^o fue el que mejoró poco.

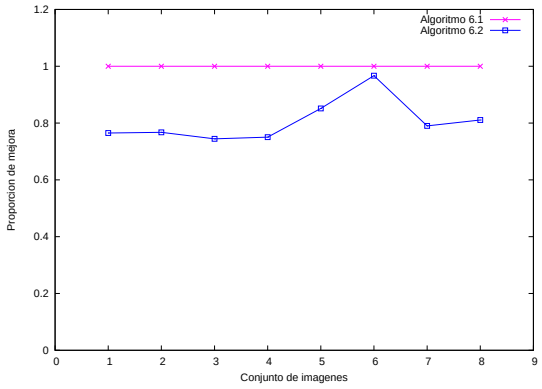


Figura 6.4: *Proporción de mejora con el uso del algoritmo 6.2. La gráfica muestra la proporción del tiempo necesitado usando el algoritmo 6.1 entre el tiempo que se necesita usando el algoritmo 6.2*

6.3. Discusión de resultados.

En este capítulo se presentaron dos algoritmos paralelos que estiman la geometría de un conjunto de $n \geq 2$ imágenes de una escena. El primer algoritmo que se explicó mostró un comportamiento aceptable de acuerdo a los resultados de rapidez, eficiencia y costo, *i.e.*, cuanto más procesadores se tenga disponibles, el algoritmo necesita menos tiempo para estimar la geometría. El último algoritmo presentado está diseñado para ser usado con grupos de procesos y las pruebas realizadas fueron con 12 procesadores formando 3 grupos de 4 procesadores. El segundo algoritmo mostró tener un mejor rendimiento que el primero cuando se usaron 12 procesadores.

La figura 6.5 describe gráficamente los resultados en cada etapa de calibración que se explicó en esta tesis. En la primera fila se muestra a las imágenes de entrada las cuales son de 800×600 píxeles. Es un conjunto de 6 imágenes pero para efectos de espacio solo se muestran las dos primeras y la última. En la segunda fila se enseña el resultado de haber procesado a cada imagen con el detector de esquinas KLT con parámetros $\lambda_{min} = 0$ y $r = 4$. En la tercera fila, las esquinas son correlacionadas con el criterio ZNCC y un umbral $t = 0.5$. La cuarta fila es el resultado de depurar las

correspondencias con RANSAC con respecto al algoritmo de los 8 puntos normalizados [Longuet-Higgins, 1981], una probabilidad $p = 0.99$, distancia umbral de 1 píxel.

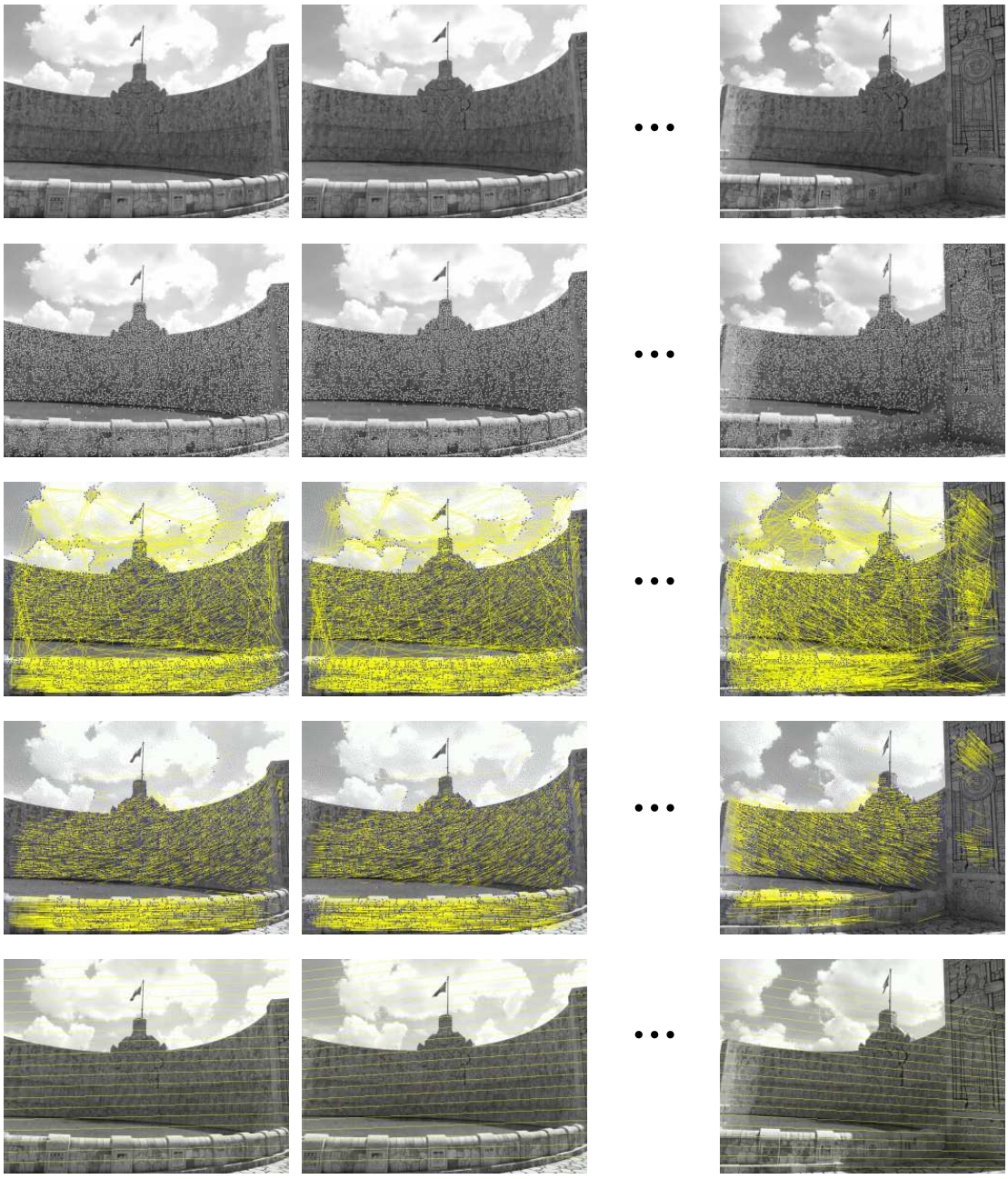


Figura 6.5: *Proceso de calibración. En la primera fila, las imágenes de entrada. En la segunda KLT sobre las imágenes. En la tercera fila están las correspondencias entre las esquinas. La cuarta fila está los inliers de las correspondencias. En la quinta fila hay ejemplos de líneas epipolares entre pares contiguos de imágenes.*

La figura 6.6 muestra una nube de puntos que se pueden reconstruir el estratum métrico con triangulación en función de las correspondencias encontradas y de la matrices fundamentales entre los pares de imágenes contiguos. Tópicos de triangulación pueden revisados en [Hartley and Sturm, 1997] y reconstrucción métrica en [Pollefeys et al., 2004] los cuales no son sujeto de estudio en esta tesis. Faugeras et al. [2001], Ma et al. [2003] y Hartley and Zisserman [2004] también proporcionan algoritmos para llegar hasta una estructura métrica como la mostrada acá.

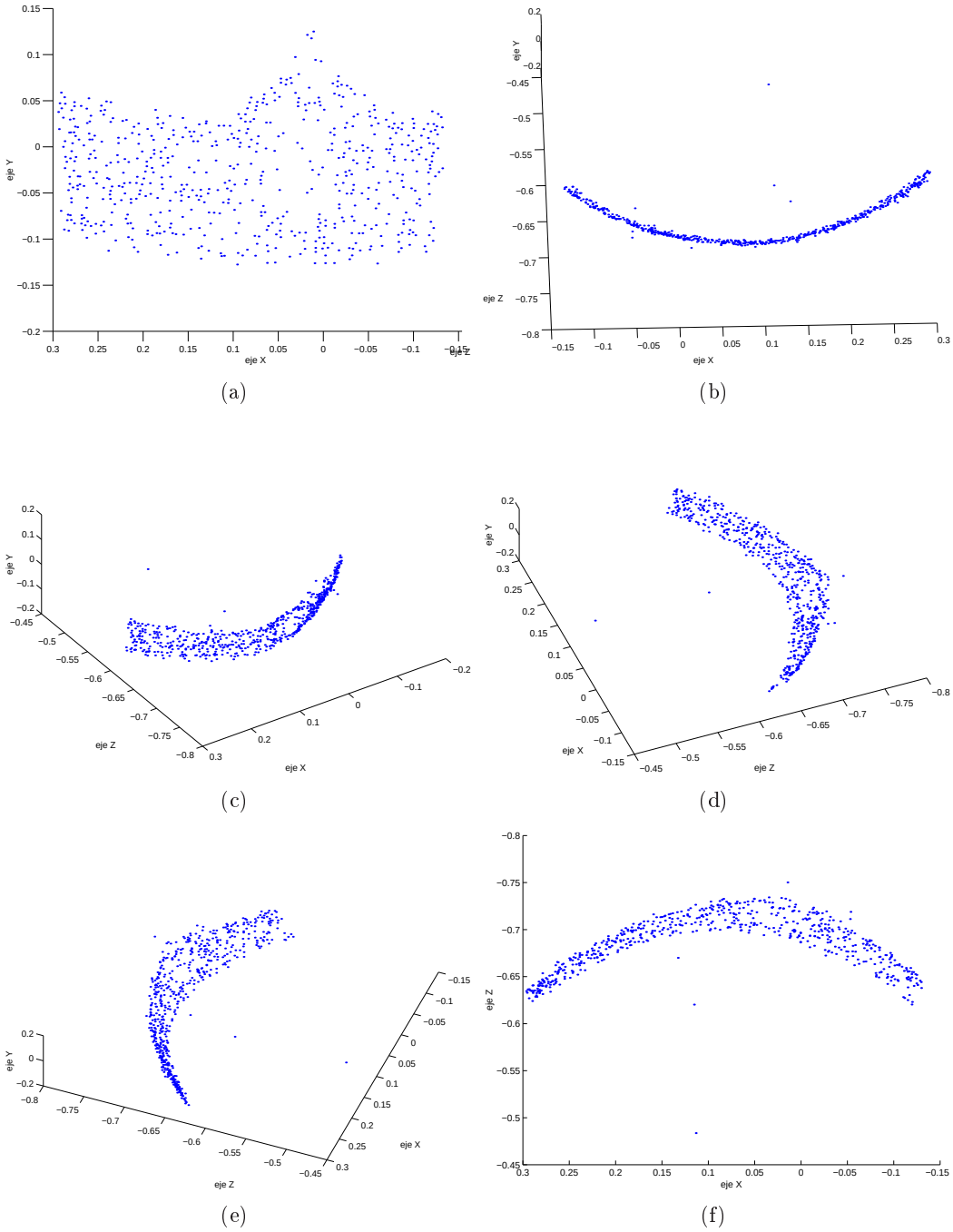


Figura 6.6: *Nube de puntos desde diferentes puntos de vista de la recuperación de la estructura métrica a partir de las correspondencias entre el conjunto de imágenes previamente presentado.*

Capítulo 7

Resumen y conclusiones.

7.1. Resumen.

El trabajo presentado en esta tesis es concerniente al proceso de calibración débil en paralelo lo cual es un proceso común inicial en muchas áreas de visión computacional. Además el proceso descrito es modular, implicando la posibilidad de sustituir alguno de los módulos, con la intención de conseguir resultados más precisos –situación que generalmente implica más costo computacional– o más rapidez en los cálculos –generalmente implicando menos precisión–. El proceso de calibración es completamente automático pero proporcionando *a priori* los parámetros necesitados para cada módulo, *i.e* no son consecuencia de los módulos anteriores.

También, los algoritmos están presentados de tal manera que cada uno de ellos puede ser usado de manera independiente, *i.e.*, pueden por si solos sin depender de las otras etapas de calibración, *e.g.*, formar parte de otros procesos de visión computacional. Al final, dos algoritmos de calibración se describen, pero uno de ellos, está diseñado para aprovechar las bondades de administrar grupos de procesos que el paradigma de programación paralela por medio de pase de mensajes y mejorando ligeramente el rendimiento de las etapas en conjunto.

Al principio del algoritmo, la etapa de localización de esquinas analiza a las imágenes y estas esquinas son proporcionas a la etapa de correspondencias; las esquinas encontradas en las imágenes son relacionadas para formar correspondencias entre pares contiguas de imágenes; estas correspondencias son entonces utilizadas en la estimación robusta de matrices fundamentales.

Los logros obtenidos muestran que es posible obtener resultados a un menor tiempo cuando se usa más de un procesador en la ejecución de un algoritmo diseñado en paralelo en visión computacional.

La tecnología ha proporcionado estupendas herramientas —arquitecturas paralelas, paradigmas de programación en paralela— pero sin una estrategia de programación paralela en los sistemas de visión, esta tecnología muy probablemente sea mal usada. Es por lo tanto evidente que una solución a la búsqueda de respuestas más rápidas y más precisas en una sistema de visión computacional puede encontrarse en un enfoque paralelo.

Apéndice A

Modelo de la cámara.

En este trabajo de tesis un modelo de cámara perspectiva es usado. Esto corresponde a una cámara pinhole ideal [Mark Pollefey, 1999]. El proceso geométrico para la formación de una imagen está bien ilustrado por Albrecht Dürer, como se muestra en la figura A.1.

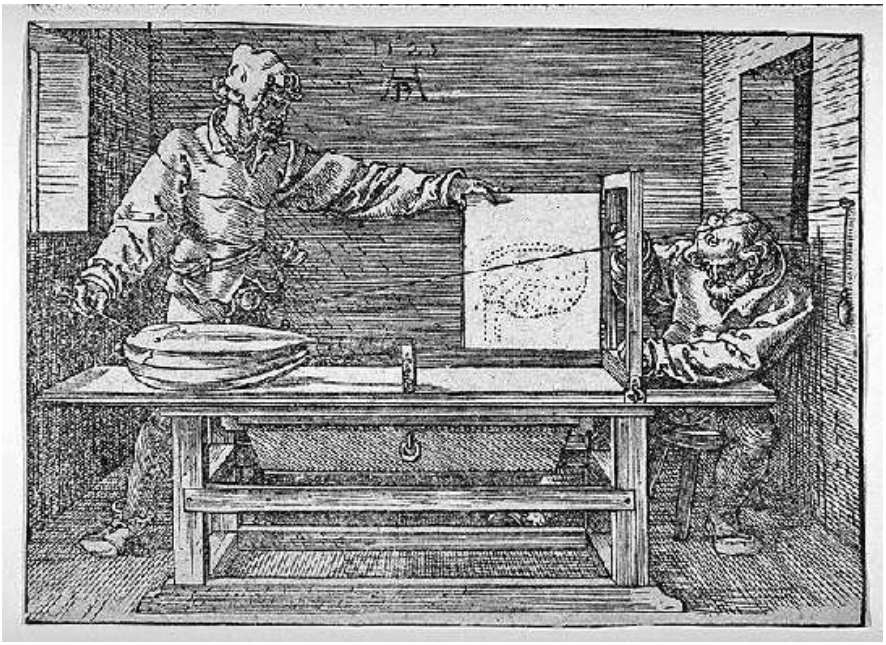


Figura A.1: *The draughtsman of the Lute, Albrecht Dürer.*

A.1. Modelo simple

El proceso de formación de una imagen está completamente determinado por la elección de un centro de proyección perspectiva $\mathbf{C}$ y un plano retinal $\mathcal{R}$. La proyección

de un punto de una escena, se obtiene de la intersección de la línea definida por este punto y por el centro de proyección, con el plano retinal. Esta proyección se ilustra en la figura A.2. En la figura se observa cómo el eje óptico pasa a través del centro de proyección $\mathbf{C}$ y es ortogonal al plano retinal $\mathcal{R}$. Su intersección con el plano retinal se define como el *punto principal* $\mathbf{c}$. La distancia entre $\mathbf{C}$ y $\mathcal{R}$ es la *distancia focal* f. La mayoría de las cámaras son descritas relativamente bién por este modelo, sin embargo en algunos casos, efectos adicionales como distorsión radial se deben tomar en cuenta.

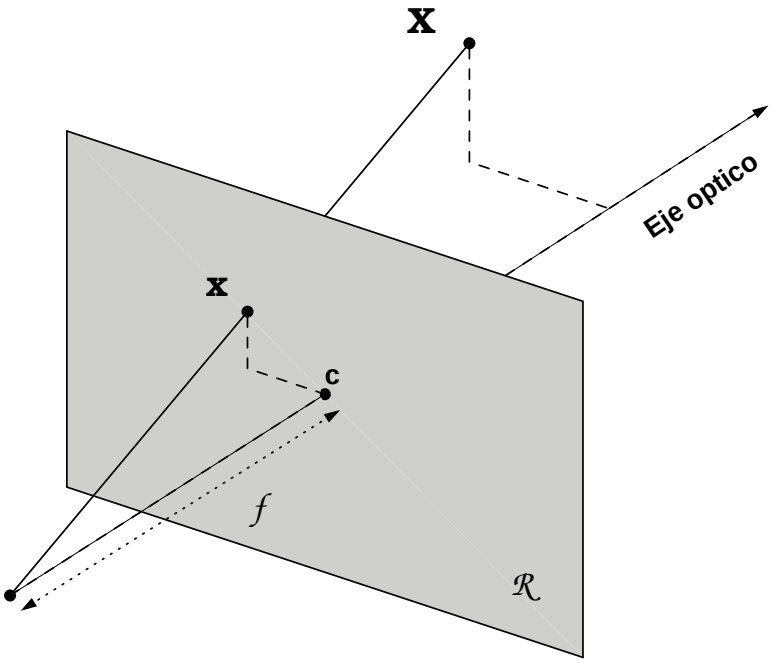


Figura A.2: *Proyección perspectiva.*

En el caso más simple, el centro de proyección es colocado en el origen del mundo, el plano de la imagen es el plano $\mathbf{Z} = 1$, resultando en una coordenada $Z = 1$ y una longitud focal $f = 1$. El proceso de creación de una imagen es

$$\hat{x}_{\mathcal{R}} = f\frac{X}{Z} \quad \hat{y}_{\mathcal{R}} = f\frac{Y}{Z}. \tag{A.1}$$

Para un punto en el espacio $\mathbf{X} = [X\ Y\ Z]^\top$ que se proyecta como $\hat{\mathbf{x}}_{\mathcal{R}} = [\hat{x}_{\mathcal{R}}\ \hat{y}_{\mathcal{R}}]^\top$, podemos usar la representación homogénea de puntos (ver apéndice B) para sustituir al punto $\hat{\mathbf{x}}_{\mathcal{R}}$ con $\mathbf{x}_{\mathcal{R}} = [x_{\mathcal{R}}\ y_{\mathcal{R}}\ 1]^\top$, donde $\frac{1}{Z}\mathbf{x}_{\mathcal{R}} = \mathbf{x}_{\mathcal{R}}$, o lo que es lo mismo decir según la representación de vectores homogéneos $\hat{\mathbf{x}}_{\mathcal{R}} \sim \mathbf{x}_{\mathcal{R}}$. Utilizando una representación homogénea de puntos, podemos reescribir a A.1 como

$$\frac{1}{Z}\begin{bmatrix} \hat{x}_{\mathcal{R}} \\ \hat{y}_{\mathcal{R}} \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}.$$

De todo lo anterior, se puede obtener una ecuación de proyección lineal como

$$\begin{bmatrix} x_{\mathcal{R}} \\ y_{\mathcal{R}} \\ 1 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{A.2}$$

A.2. Calibración intrínseca.

Con las cámaras actuales, la longitud focal f es diferente de 1, por lo tanto las coordenadas de la ecuación A.2 deberían ser escaladas con f. Por si fuera poco, las coordenadas en la imagen no corresponden a las coordenadas físicas del plano retinal. Con una cámara CCD[1], la relación depende del tamaño y forma de los píxeles y de la posición del CCD en la cámara. Las coordenadas de la imagen se obtienen por medio de la ecuación

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{f}{p_x} & (\tan\alpha)\frac{f}{p_y} & c_x \\ & \frac{f}{p_y} & c_y \\ & & 1 \end{bmatrix} \begin{bmatrix} x_{\mathcal{R}} \\ y_{\mathcal{R}} \\ 1 \end{bmatrix},$$

donde p_x y p_y son el ancho y largo de los píxeles, $\mathbf{c} = [c_x \;\; c_y \;\; 1]^\top$ es el punto principal y α es un ángulo debido a la oblicuidad de los píxeles mostrado en la figura A.3. Una notación simplificada de esta transformación como

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix} \begin{bmatrix} x_{\mathcal{R}} \\ y_{\mathcal{R}} \\ 1 \end{bmatrix} \tag{A.3}$$

será considerada, donde f_x y f_y son factores que toman en cuenta la longitud focal así como el ancho y largo de los píxeles, y s es el factor que considera el ángulo debido a la oblicuidad de los píxeles. La matriz triangular superior en la ecuación A.3, es la llamada *matriz de calibración* $\mathbf{K}$ de la cámara, por lo tanto la ecuación A.4 describe la transformación de las coordenadas retinales a las coordenadas de la imagen.

$$\mathbf{x} = \mathbf{K}\mathbf{x}_{\mathcal{R}} \tag{A.4}$$

Esta transformación está ilustrada en la figura A.3.

A.3. Matriz de proyección.

El cambio de posición de los puntos de una escena puede ser modelado como

$$\mathbf{X}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \mathbf{X}$$

[1] Charged-Coupled Device

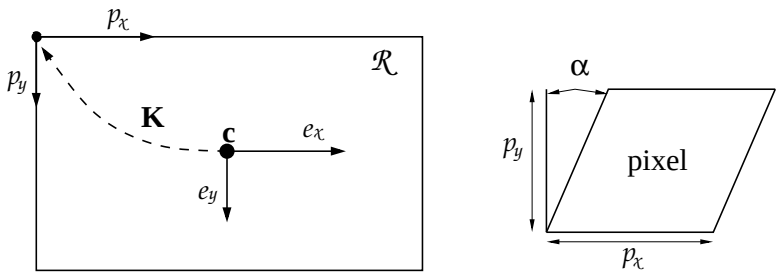


Figura A.3: *Transformación de coordenadas retinales a coordenadas de la imagen.*

donde $\mathbf{R}$ es una matriz de rotación y $\mathbf{t}$ es un vector de traslación. El movimiento de la cámara es equivalente al movimiento inverso de la escena y por lo tanto puede ser modelado como

$$\mathbf{X}' = \begin{bmatrix} \mathbf{R} & \mathbf{R}^\top \mathbf{t} \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \mathbf{X}, \tag{A.5}$$

y entonces combinando las ecuaciones A.2, A.3 y A.5, la siguiente expresión se obtiene para una cámara con alguna calibración intrínseca y con una posición y orientación específica:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x & s & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{R}^\top \mathbf{t} \\ \mathbf{0}_3^\top & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{A.6}$$

la cual puede ser simplificada a

$$\mathbf{x} \sim \mathbf{K}[\mathbf{R}^\top - \mathbf{R}^\top \mathbf{t}]\mathbf{X}$$

o

$$\mathbf{x} \sim \mathbf{P}\mathbf{X}, \tag{A.7}$$

donde la matriz $\mathbf{P}$ de 3×4 es la *matriz de proyección de la cámara*.

Apéndice B

Visión geométrica.

B.1. Conceptos preliminares.

Tal y como todos sabemos, un punto en el plano puede ser representado por el par de coordenadas $[x\ y]$ en $\mathbb{R}^2$ y de esta manera, es común identificar el plano con $\mathbb{R}^2$. Si consideramos a $\mathbb{R}^2$ como un espacio de vectores, el par $[x\ y]$ es un vector, *i.e.*, un punto es identificado como un vector. En esta sección se introduce la notación *homogénea* para puntos y líneas sobre un plano.

Vectores fila y vectores columna. Recordemos que una matriz, puede ser multiplicada por la izquierda por un vector fila y por la derecha por un vector columna, y muchas multiplicaciones de estas se llevarán a cabo a través de la presente tesis. Un vector columna estará siempre representado por una letra en negrilla como por ejemplo $\mathbf{x}$, y su transpuesta $\mathbf{x}^\top$ es el vector fila. De acuerdo a lo anterior, un punto en el plano es representado por el vector columna $\mathbf{x} = [x\ y]^\top$ y no por su transpuesta, el vector fila $\mathbf{x}^\top = [x\ y]$.

Representación homogénea de lineas. Una línea en el plano está representada por una expresión como $ax + by + c = 0$ y diferentes opciones de a, b y c resultan en diferentes líneas. De esta manera, una línea puede ser representada por el vector $[a\ b\ c]^\top$. La correspondencia entre vectores $[a\ b\ c]^\top$ no es uno a uno, ya que las líneas $ax+by+c=0$ y $(ka)x+(kb)y+(kc)=0$ son la misma para alguna $k \neq 0$, por lo tanto los vectores $[a\ b\ c]^\top$ y $k[a\ b\ c]^\top$ representan a la misma línea para $k \neq 0$. De hecho, dos vectores relacionados por una escala total son considerados como equivalentes, es decir $[a\ b\ c]^\top \sim k[a\ b\ c]^\top$. Una clase de equivalencia de vectores bajo esta relación se conoce como *vector homogéneo*. El conjunto de clases de equivalencia de vectores en $\mathbb{R}^3 - [0\ 0\ 0]^\top$ forman el *espacio proyectivo* $\mathbb{P}^2$. La notación $-[0\ 0\ 0]^\top$, indica que el vector $[0\ 0\ 0]^\top$, el cual no corresponde a alguna línea, está excluido.

Representación homogénea de puntos Un punto $\mathbf{x} = [x\ y]^\top$ está en una línea $\mathbf{l} = [a\ b\ c]^\top$ si y solo si cumple con la ecuación $ax + by + c = 0$. Esto puede ser escrito

en términos de un producto punto de vectores como $[x\ y\ 1][a\ b\ c]^\top = [x\ y\ 1]\mathbf{l} = 0$; note que $[x\ y]$ en $\mathbb{R}^2$ es representado en $\mathbb{R}^3$ como un vector de 3 elementos, donde el último elemento es un 1. Para cualquier $k \neq 0$, $[kx\ ky\ k]\mathbf{l} = 0$ si y solo si $[x\ y\ 1]\mathbf{l} = 0$, o lo que es lo mismo decir que $[kx\ ky\ k] \sim [x\ y\ 1]$; es natural por lo tanto considerar el conjunto de vectores $[kx\ ky\ k]^\top$ para diferentes valores de k como una representación del punto $[x\ y]^\top$ en $\mathbb{R}^2$. Entonces, de la misma manera que las líneas, los puntos son representados por vectores homogéneos. Un vector homogéneo arbitrario de un punto tiene la forma $\mathbf{x} = [x_1\ x_2\ x_3]^\top$, que representa al punto $[x_1/x_3\ x_2/x_3]^\top$ en $\mathbb{R}^2$. Obsérvese que para comprobar si un punto está en una línea se puede utilizar la ecuación

$$\mathbf{x}^\top\mathbf{l} = \mathbf{l}^\top\mathbf{x} = \mathbf{x}.\mathbf{l} = 0 \tag{B.1}$$

Intersección de líneas. Dadas las líneas $\mathbf{l} = [a\ b\ c]^\top$ y $\mathbf{l}' = [a'\ b'\ c']^\top$, su intersección queda establecida por el vector

$$\mathbf{x} = \mathbf{l} \times \mathbf{l}' \tag{B.2}$$

donde $\times$ representa el producto cruz. De $\mathbf{l}(\mathbf{l} \times \mathbf{l}') = \mathbf{l}'(\mathbf{l} \times \mathbf{l}') = 0$, vemos que $\mathbf{l}^\top\mathbf{x} = \mathbf{l}'^\top\mathbf{x} = 0$. Entonces $\mathbf{x}$ está en ambas líneas y por lo tanto es la intersección de las dos líneas.

La línea que pasa por dos puntos. Una expresión para la línea que pasa por los puntos $\mathbf{x}$ y $\mathbf{x}'$ puede ser derivado por un argumento enteramente análogo. Definiendo la línea $\mathbf{l}$ como $\mathbf{l} = \mathbf{x} \times \mathbf{x}'$ puede ser comprobado que ambos puntos $\mathbf{x}$ y $\mathbf{x}'$ están en $\mathbf{l}$. Así tenemos la ecuación

$$\mathbf{l} = \mathbf{x} \times \mathbf{x}' \tag{B.3}$$

Producto cruz. Si tenemos un vector $\mathbf{a} = [a_1\ a_2\ a_3]^\top$, entonces se define $[\mathbf{a}]_\times$ como

$$[\mathbf{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{B.4}$$

y el producto cruz está relacionado a esta matriz de acuerdo a

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times\mathbf{b} = \left(\mathbf{a}^\top[\mathbf{b}]_\times\right)^\top. \tag{B.5}$$

Observe que $[\mathbf{a}]_\times$ es una matriz antisimétrica, es decir, $[\mathbf{a}]_\times = -[\mathbf{a}]_\times^\top$.

B.2. Medidas de distancias a la matriz fundamental.

B.2.1. Distancia de Sampson

La *distancia de Sampson* se considera una aproximación de primer orden a la distancia geométrica. Fue usada originalmente por Sampson [Sampson, 1982] para ajustar

cónicas. En el caso particular de estimar la matriz fundamental la función de costo esta dada por

$$C = \sum_i \frac{(\mathbf{x}_i'^T \mathbf{F} \mathbf{x}_i)^2}{(\mathbf{F}\mathbf{x}_i)_1^2 + (\mathbf{F}\mathbf{x}_i)_2^2 + (\mathbf{F}^T\mathbf{x}_i')_1^2 + (\mathbf{F}^T\mathbf{x}_i')_2^2} \tag{B.6}$$

donde $(\mathbf{F}\mathbf{x}_i)_j^2$ representa el cuadrado de la j-ésima entrada del vector $\mathbf{F}\mathbf{x}_i$.

B.2.2. Distancia epipolar simétrica

La *distancia epipolar simétrica* se utiliza para minimizar la distancia que existe entre un punto y su línea epipolar calculada en las dos imágenes, esta distancia esta dada por

$$C = \sum_i d((x)_i', \mathbf{F}\mathbf{x}_i)^2 + d((x)_i, \mathbf{F}^T\mathbf{x}_i')^2 =$$

$$\sum_i (\mathbf{x}_i'^T \mathbf{F} \mathbf{x}_i)^2 \left(\frac{1}{(\mathbf{F}\mathbf{x}_i)_1^2 + (\mathbf{F}\mathbf{x}_i)_2^2} + \frac{1}{(\mathbf{F}^T\mathbf{x}_i')_1^2 + (\mathbf{F}^T\mathbf{x}_i')_2^2} \right) \tag{B.7}$$

Bibliografía

Alexandrov, A., Ionescu, M. F., Schauser, K. E., and Scheiman, C. (1997). LogGP: Incorporating long messages into the LogP model for parallel computation. *Journal of Parallel and Distributed Computing*, 44(1):71–79.

Austin, W. J. and Scaife, N. R. (1994). Reconfigurable Parallel Vision System: Informal Specification. Technical Report RM/94/4, Department of Computing and Electrical Engineering, Heriot-Watt University.

Bazewicz, J., Trystram, D., Ecker, K., and Plateau, B., editors (2000). *Handbook on Parallel and Distributed Processing.* Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bennett, S. (1994). *Real-time computer control (2nd ed.): an introduction.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Bertozzi, M. and Broggi, A. (1998). Gold: a parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Transactions on Image Processing*, 7(1):62–81.

Briceño Coronado, A. A. (2005). Reconstrucción métrica a través de la recuperación de la geometría epipolar. Tesis de Licenciatura, Facultad de Matemáticas, UADY.

Culler, D. E., Karp, R. M., Patterson, D. A., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R., and von Eicken, T. (1993). LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12.

Debevec, P. E. and Malik, J. (1997). Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 369–378, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Faber, V., Lubeck, O. M., and A B White, J. (1986). Superlinear speedup of an efficient sequential algorithm is not possible. *Parallel Comput.*, 3(3):259–260.

Faugeras, O., Luong, Q.-T., and Papadopoulou, T. (2001). *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications.* MIT Press, Cambridge, MA, USA.

Fischler, M. and Bolles, R. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395.

Fitzgibbon, A. W., Cross, G., and Zisserman, A. (1998). Automatic 3d model construction for turn-table sequences. In *SMILE'98: Proceedings of the European Workshop on 3D Structure from Multiple Images of Large-Scale Environments*, pages 155–170, London, UK. Springer-Verlag.

Flynn, M. J. (1972). Some computer organizations ans their effectiveness. *IEEE Transactions on Computers*, C-21(9):948–960.

Foster, I. (1995). *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Fua, P. (1993). A Parallel Stereo Algorithm that Produces Dense Depth Maps and Preserves Image Features. *Machine Vision and Applications*, 6(1):35–49.

Gropp, W., Lederman, S. H., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., and Snir, M. (1998). *MPI – The Complete Reference: Volume 2, the MPI-2 Extensions.* MIT Press, Cambridge, MA, EUA.

Hammond, K. and Michelson, G., editors (2000). *Research Directions in Parallel Functional Programming.* Springer-Verlag, London, UK.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference, University of Manchester*, pages 147–151. The Plessey Company.

Hartley, R. I. (1997). In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593.

Hartley, R. I. and Sturm, P. (1997). Triangulation. *Computer Vision and Image Understanding: CVIU*, 68(2):146–157.

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN:0521549518.

Jarvis, R. A. (1983). A perspective on range-finding techniques for computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:122–139.

Johnson, E. E. (1988). Completing an mimd multiprocessor taxonomy. *Computer Architecture News*, (16):44–47.

Kanade, T. and Webb, J. A. (1988). End of year report for parallel vision algorithm design and implementation, january 1987-january 1988. Technical Report CMU-RI-TR-88-11, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Kessler, C. W. (2006). Teaching parallel programming early. In *Proceedings of Workshop on Developing Computer Science Education – How Can It Be Done?* Linköpings universitet.

Kim K.-N., Ramakrishna R. S., C. T.-S. (1998). New parallel vision environment in heterogeneous networked computing. In *Proceeding of SPIE, Vol. 3452, SPIE 43rd Annual Meeting, San Diego, CA*.

Lacey, A., Pinitkarn, N., and Thacker, N. (2000). An evaluation of the performance of ransac algorithms for stereo camera calibrarion. In Mirmehdi, M. and Thomas, B., editors, *Proceedings of the British Machine Vision Conference BMVC 2000*, Bristol, UK. URL http://www.bmva.ac.uk/bmvc/2000/papers.

Laine, A. and Roman, G. (1991). A parallel algorithm for incremental stereo matching on simd machines. *IEEE Transactions on Robotics and Automation*, 7(1):123–134.

Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision (ijcai). In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 674–679.

Ma, Y., Soatto, S., Kosecka, J., and Sastry, S. S. (2003). *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag.

Mark Pollefey (1999). *Self-calibration and metric 3D reconstruction from uncalibrated image sequences*. PhD thesis, K.U.Leuven, ESAT-PSI.

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information*. W. H. Freeman, San Francisco.

Martin, R. P., Vahdat, A., Culler, D. E., and Anderson, T. E. (1997). Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *ISCA*, pages 85–97.

Michaelson, G. and Scaife, N. (2002). *Skeletons and Patterns for Parallel and Distributed Computing*, chapter Skeleton Realisations from Functional Prototypes, pages 129–153. Wiley.

Moravec, H. (1977). Towards Automatic Visual Obstacle Avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page 584.

Moravec, H. (1979). Visual Mapping by a Robot Rover. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 598–600.

Morrison, S, R. (2003). *Cluster Computing, Architectures, Operating Systems, Parallel_ Processing and Programming Languages*, volume 2.

Patrón P., A. (2006). Reconstrucción tridimensional a partir de una secuencia de imágenes. Master's thesis, Facultad de Matemáticas, UADY.

Pollefeys, M., Van Gool, L., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J., and Koch, R. (2004). Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232.

Quinn, M. J. (1994). *Parallel computing (2nd ed.): theory and practice.* McGraw-Hill, Inc., New York, NY, USA.

Rousseeuw, P. J. and Leroy, A. M. (1987). *Robust regression and outlier detection.* John Wiley & Sons, Inc., New York, NY, USA.

Salvi, J., Pagès, J., and Batlle, J. (2004). Pattern codification strategies in structured light systems. *Pattern Recognition*, 37(4):827–849.

Sampson, P. (1982). Fitting conic sections to 'very scattered' data: An interactive refinement of the Bookstein algorithm. *Computer Vision, Graphics, and Image Processing*, 18:97–108.

Scaife, N. R., Michaelson, G. J., and Wallace, A. M. (1996). A Method for Developing Parallel Vision Algorithms with an Example of Edge Tracking. Technical Report RM/96/1, Department of Computing and Electrical Engineering, Heriot-Watt University, Riccarton, Edinburgh.

Scaife, N. R., Michaelson, G. J., and Wallace, A. M. (1997). Four Skeletons for Computer Vision. In *Proceedings of 9th International Workshop on Implementation of Functional Languages*, University of St. Andrews.

Scaife Norman R. (2000). *A Dual Source, Parallel Architecture for Computer Vision.* PhD thesis, Heriot-Watt University.

Shi, H. and Schaeffer, J. (1992). Parallel Sorting by Regular Sampling. *Journal of Parallel and Distributed Computing*, 14(4):361–372.

Shi, J. and Tomasi, C. (1994). Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle.

Smith, S. M. and Brady, J. M. (1995). SUSAN – A new approach to low level image processing. Technical Report TR95SMS1c, Chertsey, Surrey, UK.

Snir, M. and Otto, S. (1998). *MPI-The Complete Reference: The MPI Core.* MIT Press, Cambridge, MA, USA.

Tell, D. and Carlsson, S. (2000). Wide baseline point matching using affine invariants computed from intensity profiles. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part I*, pages 814–828, London, UK. Springer-Verlag.

Tomasi, C. and Kanade, T. (1991). Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University.

Tuytelaars, T. and Gool, L. V. (2004). Matching widely separated views based on affine invariant regions. *Int. J. Comput. Vision*, 59(1):61–85.

Vergauwen, M. and Gool, L. V. (2006). Web-based 3d reconstruction service. *Machine Vision and Aplications*, 17(6):411–426.

Zhang, Z., Deriche, R., Faugeras, O., and Luong, Q.-T. (1995). A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artif. Intell.*, 78(1-2):87–119.